Kinodynamic motion planning for quadrotor-like aerial robots

Alexandre Boeuf

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Mathématiques Appliquées

Présentée et soutenue par :

M. ALEXANDRE BOEUF

le mercredi 5 juillet 2017

Titre :

Kinodynamic motion planning for quadrotor-like aerial robots

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes (L.A.A.S.)

Directeurs de Thèse :

M. THIERRY SIMEON

M. JUAN CORTES

Rapporteurs :

M. PAOLO ROBUFFO GIORDANO, INRIA RENNES

M. THIERRY FRAICHARD, INRIA GRENOBLE - RHONE ALPES

Membres du jury :

M. RACHID ALAMI, LAAS TOULOUSE, Président

M. JUAN CORTES, LAAS TOULOUSE, Membre

Mme MARILENA VENDITTELLI, UNIV. DEGLI STUDI DE ROME SAPIENZA, Membre

M. THIERRY SIMEON, LAAS TOULOUSE, Membre

Contents

Introduction

Motion planning is the field of computer science that aims at developing algorithmic techniques allowing the automatic computation of trajectories for a mechanical system. The nature of such a system will vary according to the fields of application. In computer animation for instance this system could be a humanoid avatar. In molecular biology it could be a protein. The field of application of this work being robotics, the system is here a robot. We define a *robot* as a mechanical agent controlled by a computer program. The modern meaning of the word seems to come from the English translation of the 1920 play *R.U.R.* (Rossum's Universal Robots), by Karel Capek, from Czech, *robotnik* (slave) derived from *robota* (forced labor). As for the term *robotics* (the science of robots), it was first used by Isaac Asimov in the 1941 short story *Liar!* The main feature of a robot is its ability to interact with its environment through motion. Being able to efficiently plan its movements is therefore a fundamental component of any robotic system.

More specifically we will focus here on *aerial* robotics, *i.e.* flying robots. Historically the first term used to refer to a remotely operated aircraft was *drone* (male bee). According to the military historian Steven Zaloga it was the American Commander Delmer Fahrney that first used it in homage to the British remote-control bi-plan *DH 82B Queen Bee*. However, since a flying robot is not necessarily remotely controlled, we will use in this work the term UAV (Unmanned Aerial Vehicles). Also note that not all UAVs are robots since a UAV could very well be remotely controlled by a human operator instead of a computer program. Aerial robotics has a wild range of applications. Apart from the sadly famous military ones, we can cite photography and video, inspection, surveillance, search and rescue, and even aerial manipulation as we will see later. For these civilian applications a class of devices is more and more used because of its scalability, agility, and robustness: the multi-rotor helicopters. In this work we will focus on the four-rotor kind called *quadrotor* or sometimes *quadcopter*.

The classic motion planning problem consists in computing a series of motions that brings the system from a given initial configuration to a desired final configuration without generating collisions with its environment, most of the time known in advance. Usual methods typically explore the system's configuration space regardless of its dynamics. By construction the thrust force that allows a quadrotor to fly is tangential to its attitude which implies that not every motion can be performed. This could, for example, be compared to the fact that a car can not move sideways. Furthermore, the magnitude of this thrust force is limited by the physical capabilities of the engines operating the propellers. Therefore the same applies to the linear acceleration of the robot. For all these reasons, not only position and orientation must be planned, higher derivatives must be planned also if the motion is to be executed. When this is the case we talk of *kinodynamic motion planning*.

In motion planning a distinction is usually made between the local planner and the global planner. The former is in charge of producing a valid trajectory between two configurations (or states) of the system without necessarily taking collisions into account. The later is the overall algorithmic process that is in charge of solving the motion planning problem by exploring the configuration space (or the state space) of the system. It relies on multiple calls to the local planner. The first contribution of this thesis is to propose a local planner that interpolates two states consisting of an arbitrary number of degrees of freedom (dof) and their first and second derivatives. Given a set of bounds on the dof derivatives up to the fourth order (snap), it quickly produces a near-optimal minimum time trajectory that respects those bounds. Although in the context of this thesis this local planner is applied to the quadrotor system which state is considered to be its flat outputs (the 3D position of the center of mass and the yaw angle) together with their first and second derivatives, it can more generally be applied to any system with either uncoupled dynamics or that is differentially flat.

In most of modern global motion planning algorithms, the exploration is guided by a distance function (or metric) and some of these algorithms are very sensible to it. The best choice is often to use the cost-to-go, *i.e.* the cost associated to the local method. For instance if the configuration space is an Euclidean space and the local method is the linear interpolation then the best metric is the Euclidean distance since it is also the cost-to-go. But in the context of kinodynamic motion planning, the cost-to-go is often the duration of the minimal-time trajectory. The problem in this case is that computing the cost-to-go is as hard (and thus as costly) as computing the optimal trajectory itself. The second contribution of this thesis is to propose the use of a specific metric that is a good approximation of the cost-to-go but which computation is far less time consuming.

The dominant paradigm in motion planning nowadays is *sampling-based motion planning*. This class of algorithms relies on random sampling of the configuration space (or state space) in order to quickly explore it. Several different strategies can be considered. A common sampling strategy is for instance uniform sampling. However, for some types of constrained problems this choice may not be the most efficient. It appears that in our context uniform sampling is in fact a rather poor strategy. We will indeed see that a great majority of uniformly sampled states can not be interpolated. The third contribution of this thesis is an incremental sampling strategy that significantly decreases the probability of this happening.

Please note that this work has been realized in the context of the European funded project ARCAS (Aerial Robotics Cooperative Assembly System) that has been conducted between the years 2012 and 2016 and that proposed the development and experimental validation of the first cooperative free-flying robot system for assembly and structure construction. It paved the way for a large number of applications including the building of platforms for evacuation of people or landing aircrafts, the inspection and maintenance of facilities and the construction of

structures in inaccessible sites and in space. The fourth contribution of this thesis is, through this project, the demonstration that an interface between symbolic and geometric planning, previously developed at LAAS-CNRS, can successfully be applied to aerial manipulation.

This document is organized as follows:

In Chapter 1 we introduce the general concepts used in motion planning along with a state of the art of there applications to robotics. We then do the same with kinodynamic motion planning. We finally provide an overview of the state of the art of motion planning in the context of aerial robotics.

The focus of Chapter 2 is the quadrotor system itself. We first give a model of its dynamics which in a second time allows us, after a brief introduction to the general principles of control theory, to define its control space and its state space. Its physical constraints in the control space are then discussed. We finally see that the system is differentially flat, which is having an impact on planning.

In Chapter 3 we focus on our proposition of steering method for a quadrotor. We first present the corresponding two-point boundary value problem. A method to solve it, that can be used as a local planner, is then proposed. It is a near time-optimal spline-based approach for which we will discuss the optimality of the computed solutions. We finally see how this method can be applied to more general systems.

In Chapter 4 we present the global approaches on which we have focused to solve the kinodynamic motion planning problem for a quadrotor. We show in particular how our local planner can be integrated into those global methods. We first present a decoupled approach together with a local optimization method of the global solution trajectory. We then focus on direct approaches. In that perspective we begin by addressing the problem of the metric in the state space. Follows the description of an incremental sampling strategy in the state space. We finally present two global direct approaches to kinodynamic motion planning and discuss the influence of both the metric and the sampling strategy on both of them.

The goal of Chapter 5 is to show that the trajectories that we plan using the methods described in Chapters 3 and 4 can actually be executed on a real physical system. We first present the controller we have chosen to track our trajectories. We then give an overview of our testbed: its different components, both in terms of hardware and software, and how they interconnect. We finally present the conducted experiments together with some of their results.

In Chapter 6 we detail the integration of our works into the ARCAS project in the context of which they were conducted. We first present the project itself, its objectives, its different partners, its subsystems and the general framework in which they are integrated. We then focus on the motion planning system and finally detail the link between symbolic and geometric planning.

Chapter 1

Motion planning: main concepts and state of the art

Contents

In this chapter we introduce the general concepts used in motion planning along with a state of the art of there applications to robotics and then more specifically to aerial robotics.

1.1 General notions

1.1.1 Modeling the system

As an introduction we stated that motion planning is the field of computer science that aims at developing algorithmic techniques allowing the automatic computation

of trajectories for a mechanical system. The classical way of modeling such a system is to use a *kinematic chain*. It consists of a set of rigid bodies called *links*. A rigid body is a closed subset $\mathcal{B}$ of the *workspace* $\mathcal{W}$, *i.e.* the space where motion takes place. Usually the workspace is the three dimensional Euclidean space $\mathbb{R}^3$. Links can be connected to each others. These connections are called *joints*. They are modeled as ideal movements between links such as relative rotations and translations. Note that any joint can be modeled as a set of rotations and translations. For example the so called *free-flying* joint which represents the unconstrained motion of a rigid body in the three dimensional Euclidean space $\mathbb{R}^3$ with respect to a given inertial frame is modeled by three translations and three rotations. Note that this is the case for a quadrotor when not considering differential constraints on the motion. This formalism provides a natural parametrization of a kinematic chain. Indeed, if every numerical value of the angles of rotation and displacements in translation are known then the geometrical state of the system in the workspace is entirely defined. A given set of such values, represented by a tuple of real numbers, is called a *configuration*, often noted q. One element of the tuple is called a *degree of freedom* (dof) and therefore the number of parameters (the length of the tuple) is referred to as the number of dof of the kinematic chain. The next section gives the formulation of the motion planning problem in this context.

1.1.2 Problem formulation and configuration space

The classic formulation of the motion planning problem is known as the *piano movers' problem* [Schwartz 1983]. It can be stated as follows: "Given a kinematic chain and a set of rigid bodies considered as obstacles which positions and orientations are known, is there a continuous collision-free motion that will take the kinematic chain from a given initial collision-free configuration to a desired final collision-free configuration?". Note that the original formulation was actually given for a system composed of only a single free-flying body. This more complete formulation is known as the *generalized mover's problem* [Reif 1979].

In 1983, Tomas Lozano-Péres introduced the use in robotics of the previously known notion of *configuration space* [Lozano-Pérez 1983], often noted $\mathcal{C}$. It is defined as the set of all possible configurations of a kinematic chain. Its topological nature and dimension depends on the system. For example the configuration space of a free-flying rigid body is the special Euclidean group of dimension three, $SE(3)$. An other classic example is the double pendulum which configuration space is the 2-torus, $\mathcal{C} = S^1 \times S^1 = \mathbb{T}^2$ (see Figure 1.1). In any case, $\mathcal{C}$ is a n-dimensional manifold, with n the number of dof of the chain. Given a kinematic chain and a set of obstacles in $\mathcal{W}$, we can define the subset $\mathcal{C}_{obst} \subseteq \mathcal{C}$ as the set of configurations in collision. We then define the *free space* and note $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obst}$ its complementary subset in $\mathcal{C}$, *i.e.* the set of collision-free configurations. This representation has the advantage of reducing the motion planning problem to the search of a continuous path for a point in $\mathcal{C}_{free}$. Formally the problem becomes the one of finding a continuous

application $\tau : [0\ 1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_{init}$ and $\tau(1) = q_{final}$, where q_{init} and q_{final} are the initial and final configurations of the system respectively. Solving the problem with this formulation consists in studying the connectivity of the free space. Indeed, a solution exists if and only if q_{init} and q_{final} are in the same connected component of $\mathcal{C}_{free}$. In the next section we present the main approaches that have been developed to tackle this problem.

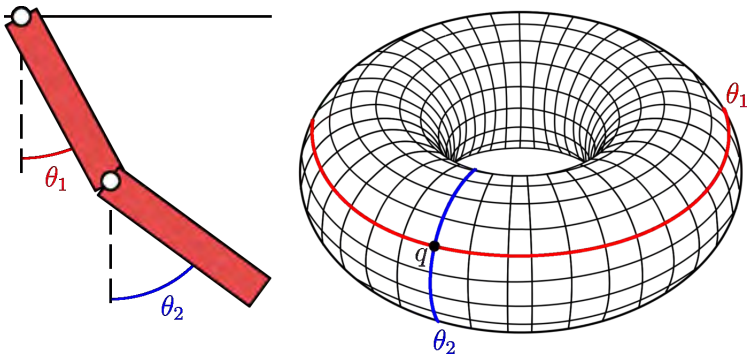


Figure 1.1: The double pendulum and its configuration space the 2-torus.

1.1.3 The planning methods

Even if the earliest works on motion planning date back from the late 60's [Nilsson 1969], most of the algorithmic works started in the early 80's. Numerous approaches have been developed over this last three decades. One can classify them into two main subsets: the *complete* methods and the other ones. A method is said to be complete when it is able not only to find a solution if one exists but also to determine the (non-)existence of solutions. These methods often rely on an explicit representation of $\mathcal{C}_{free}$ which is a problem that has been shown to be polynomial in the number of obstacles but exponential in the number of degrees of freedom (i.e. the *dimension of the problem*) [Reif 1979]. And this is indeed the complexity of the most efficient of them [Canny 1988a]. This practical inability to scale to real-life applications is the reason why these methods are bound to only have a theoretical interest.

Exploring the connectivity of $\mathcal{C}_{free}$ without having to explicitly represent it has been a motivation for the development of other methods. Some of them rely on a comprehensive discretization of $\mathcal{C}_{free}$ [Faverjon 1984, Lozano-Perez 1987] using cells or grids. They are said to be *resolution complete*. This means that if a solution exists, then there is a discretization resolution for which these methods are able to find it. Because the higher the problem's dimension goes the finer the required resolution becomes, the number of cells or grids tends to rapidly become quite big. These methods are therefore also unable to perform in reasonable time in high dimension.

Other classes of methods are using the *potential field approach* [Khatib 1986, Koren 1991]. The idea is to define a potential field other the configuration space which is the sum of an attractive field generated by the goal configuration and repulsing fields generated by the obstacles (*i.e.* the elements of $\mathcal{C}_{obst}$). An optimization process based on gradient descent techniques is then performed other this field. These techniques have good performances even in high dimension and are particularly well suited to tackle reactive obstacle avoidance problems. They however have a week spot which is their tendency to quickly get trapped into local minima of the potential field. The use of random walks [Barraquand 1990, Barraquand 1991] has been introduced to overcome this limitation, thus opening the way to sampling-based motion planning, which is the focus of our next section. Note that a more extensive overview of all these motion planning techniques can be found in Latombe's book [Latombe 2012].

1.2 Sampling-based motion planning

1.2.1 Overview

In the potential field method, the introduction of random walks as a tool to escape local minima opened the way to the use of sampling-based approaches in motion planning. But randomness here only plays its part when the main process gets stuck. What if exploration of the connectivity of $\mathcal{C}_{free}$ could be achieved mainly through the use of randomness? This idea, inherited from the Monte Carlo method [Metropolis 1949], is behind sampling-based motion planning. The general principle consists in randomly sampling configurations in $\mathcal{C}$ and dismiss the ones in $\mathcal{C}_{obst}$ with collision checking algorithms. This way, we get a discretization of $\mathcal{C}_{free}$ without having to explicitly represent it. If the sampling is uniform then as the number of samples increases the discretization itself becomes more and more uniform. This technique provides an efficient way to explore $\mathcal{C}_{free}$ but not its connectivity. To do that, the sampled configuration have to be linked inside a network of valid transitions between them. And this is the heart of sampling-based motion planning: what strategies should be used to try and link the sampled collision-free configurations? Many different approaches have been developed over the years, which can be divided into two main families: the probabilistic networks and the diffusion-based methods. See Figure1.2 for an illustration of typical behaviors of these methods. All these techniques have in common the fact that they are *probabilistically complete*, meaning that if a solution exists then it can be found given enough computing time. This section provides a presentation of the main components of these methods along with a description of some of the existing variants for each two main families. We will also introduce the concept of sampling-based kinodynamic planning needed to tackle the differential constraints imposed by some systems such as the quadrotor. Note that a more comprehensive overview of sampling-based motion planning

techniques can be found in LaValle's book [LaValle 2006].

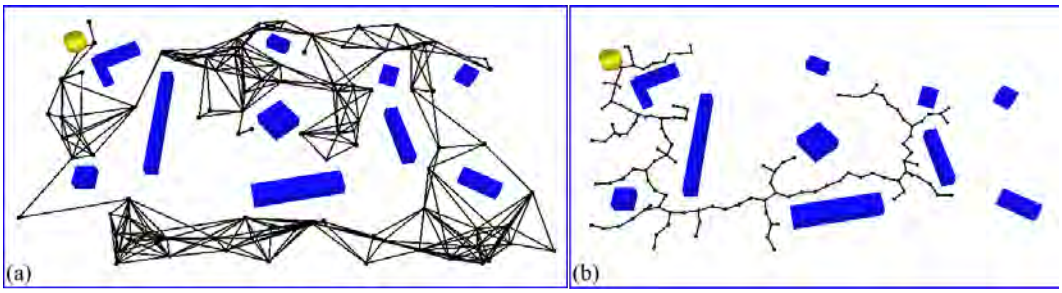


Figure 1.2: An illustration of typical behaviors of a probabilistic roadmap method (a) compared to a diffusion-based approach (b)

1.2.2 Main components

Sampling-based motion planning explores the connectivity of $\mathcal{C}_{free}$ by sampling random configurations and trying to connect them within a feasible transition network. Thus to represent this underlying network a data structure is needed. A natural candidate is the graph structure often called a *roadmap*. A node represents a configuration and an edge represents a valid transition between two configurations. Several things have to be noted here.

We assumed up to now that the configuration sampling was uniform. This is not necessarily the case. Uniform sampling is actually only one *sampling strategy* among many other possibilities. As we will see in sections 1.2.3 and 4.2.2, the sampling strategy can have a significant influence on the overall performances of a motion planning algorithm.

We say that an edge represents a valid transition between configurations. What is the nature of this transition, how is it computed and what does *valid* means in this context? When we talk of a transition between two configurations q_i and q_j, we actually refer to a continuous path in $\mathcal{C}$ which end points are these configurations. It is called a *local path*, as opposition to the global solution of the motion planning problem. It is computed by the *steering method* (also called the *local planner*). Formally, this method can be thought as a deterministic application

$$SM : \left(\begin{array}{rcl} \mathcal{C}^2 & \rightarrow & C^0\left([0\ 1], \mathcal{C}\right) \\ (q_i, q_j) & \mapsto & SM_{ij} \end{array} \right) \text{ s.t. } \left\{ \begin{array}{l} SM_{ij}(0) = q_i \\ SM_{ij}(1) = q_j \end{array} \right. \tag{1.1}$$

where $C^0\left([0\ 1], \mathcal{C}\right)$ is the set of continuous applications from $[0\ 1]$ to $\mathcal{C}$. The simplest (and therefore most commonly used) steering method is linear interpolation:

$$\forall (q_i, q_j) \in \mathcal{C}^2,\ \forall u \in [0\ 1],\ SM_{ij}(u) = (1-u).q_i + u.q_j$$

Note that in this case SM is symmetric, meaning that:

$$\forall (q_i, q_j) \in \mathcal{C}^2,\ \forall u \in [0\ 1],\ SM_{ij}(u) = SM_{ji}(1-u) \tag{1.2}$$

When this is the case the underlying graph is undirected. But a steering method is not necessarily symmetric as we will see in chapter 3, in which case the graph has to be directed.

As for what *valid* means, in the classic formulation of the piano movers' problem for instance it stands for collision-free. Note that the local path does not indeed necessarily lie in $\mathcal{C}_{free}$. This has to be a posteriori verified by the *collision checking module*. But collisions are not always the only constraints. For some robots, not every path in the free space can be executed. For example a car-like robot is not able to move sideways. Thus the local paths produced by the steering method also have to be *feasible* to be valid. In the example of the car-like robot, the system is said to be *non holonomic*. It means that the equations of motion are non integrable differential equations involving the time derivatives of the configuration variables. This is often the case when the system has less controls than configuration variables. For instance a car-like robot has two controls (linear and angular velocity) while its configuration space is of dimension three (position and orientation in the plane). Planning for non holonomic systems can sometimes be treated by carefully choosing a specifically designed steering method ([Dubins 1957]). A more comprehensive discussion on the subject can be found in [Laumond 1998]. Non holonomy thus implies kinematic constraints arising from the underlying dynamics of the system. But an other type of constraints can arise from dynamics. A system can be submitted for example to maximum velocity or acceleration because of its physical limitations. More generally a differential constraint is a bound on the modulus of the time derivatives of the degrees of freedom of the system. Planning collision-free trajectories for such systems implies to take into account both the kinematic constraints (collisions and possibly non holonomy) and the dynamics constraints. This is called the *kinodynamic motion planning problem* [Canny 1988b, Donald 1993]. Kinodynamic motion planning will be detailed in section 1.3.

We still have to mention a crucial component. When a new collision-free configuration is sampled the way it is tried to be linked to the current graph is called the *connection strategy*. It is the algorithmic core specific to each method and responsible for managing calls to the steering method. Designing such a strategy mainly consists in choosing what connections are to be tested and in which order. The following sections present two main families of connection strategies and their variants.

At that stage, either to increase efficiency in probabilistic roadmaps, or to guide the exploration towards empty regions in diffusion-based methods, a nearest neighbours search is often performed in order to select candidate nodes in the current graph and the order in which they should be considered. In this case, the definition

of a *metric* (or *distance function*) on the configuration space is needed. It is an application $M : \mathcal{C}^2 \to \mathbb{R}^+$ such that:

$$\forall (q_i, q_j, q_k) \in \mathcal{C}^3, \begin{cases} M(q_i, q_j) = 0 \implies q_i = q_j & : \text{coincidence} \\ M(q_i, q_j) = M(q_j, q_i) & : \text{symmetry} \\ M(q_i, q_k) \leq M(q_i, q_j) + M(q_j, q_k) & : \text{triangle inequality} \end{cases}$$

The most commonly used metric is the Euclidean metric $M_E(q_i, q_j) = \|q_i - q_j\|_2$. But this is not always the case. For car-like robots for intense the metric has to be related to the steering method, or even induced by it (see for instance [Laumond 1993, Giordano 2006]). Note that as discussed by [LaValle 2001] the ideal metric is the cost-to-go, *i.e.* for a given cost function the cost of bringing the system from q_i to q_j. The problem is that finding the cost-to-go is often as hard as solving the planning problem. It is therefore often important to find a good and computationally efficient approximation. Also note that if the cost-to-go is not symmetric neither is the ideal distance function. The ideal metric would then be a *quasi-metric*, which has the same properties of a metric, symmetry excepted. A more comprehensive discussion on this issue will be conducted in section 4.2.1. The next section presents a family of connection strategies known as the *probabilistic roadmaps*.

1.2.3 Probabilistic roadmaps

The first version of this connection strategy was first introduced by [Kavraki 1996] under the name *Probabilistic Roadmap Method* (PRM). It since gave birth to many variants. This strategy consists in two phases: a learning phase and a query phase. During the learning phase the connectivity of the free space is explored and stored in a graph as explained in the previous section. Note that this method was the first to use the idea of capturing the connectivity of the free space in a graph. The query phase consists in connecting the initial and final configurations to the graph and then perform a shortest path search.

During the learning phase described in Algorithm 1, the undirected graph G is stored in a set N of nodes and a set E of edges both initially empty. At each iteration a random configuration q is sampled. If q is not in collision then it is added to N and a set N_q of nearest neighbors of q in N is selected according to a given metric M and a threshold (possibly infinite, *i.e.* $N_q = N$). For each node n in N_q, and if n and q are not already connected in G (*i.e.* if n and q are not in the same connected component of G), the local path $SM(q, n)$ is computed and tested for collisions. If it is collision free then the edge (q, n) is added to E and the connected components of G are updated. The algorithm runs until a chosen learning time has passed.

For an initial configuration q_i and a final configuration q_f, the query phase consists in finding two nodes n_i and n_f in N such that n_i and n_f are in the same

Algorithm 1: PRM learning phase

```
N ← ∅;
E ← ∅;
G ← {N, E};
while time < learning time do
    q ← random configuration;
    if q ∈ C_free then
        N ← N ∪ {q};
        N_q ← NearestNeighbors(q, N);
        foreach n ∈N_q in increasing order of M(q, n) do
            if n and q not in same connected component then
                if SM(q, n) is valid then
                    E ← E ∪ {(q, n)};
                    UpdateConnectedComponents(G) ;
                end
            end
        end
    end
end
Return G;
```

connected component of G and both $SM(q_i, n_i)$ and $SM(q_f, n_f)$ are collision free. This is done the same way a new random configuration is added at each step of the learning phase. The query fails if no such nodes can be found. In this case an other learning phase if performed. Otherwise a shortest path search is conducted in G between q_i and q_f using for examples either a $\mathcal{A}^*$ or Dijkstra algorithm.

Learning the connectivity of the free space before making a planning query and then keeping this information between each next query is the reason why this method is particularly well suited for multiple planning queries for the same system in the same static environment. The class of methods inspired by PRM are thus often called *multiple queries methods*. Note however that this approach can be used in a single query mode. It is for example possible to initialize the node set with $N = \{q_i\}$ (instead of $N = \emptyset$) and choose q_f as the first configuration to add (instead of a random one). In this case the algorithm stops when q_i and q_f are in the same connected component of G (or when a given amount of time has passed).

Many variants of this method have been proposed since in order to try and increase the overall efficiency of the planning process. Some of them are targeting one well established drawback that is the difficulty of finding connections going through the thiner regions of the free space. This is known as the *narrow passage problem*. It is a consequence of the uniform sampling strategy of the configuration space. In regions where $\mathcal{C}_{obst}$ is dense, a poor coverage of the free space is indeed obtained when using uniform sampling. Therefore some approaches are using different sam-

pling strategies. A class of methods is increasing the probability of finding samples in C_{free} by targeting the border of C_{obst}. One possibility is to allow some of the samples to lie in $\mathcal{C}_{obst}$ and then try to "push" them towards $\mathcal{C}_{free}$ [Hsu 1998]. In the *Obstacle-Based PRM* approach [Amato 1998], the opposite strategy is implemented: samples lying in $\mathcal{C}_{free}$ are "pushed" towards $\mathcal{C}_{obst}$. In the *Gaussian sampling strategy* [Boor 1999] couples of close configurations are sampled and disregarded if one of them does not lie in $\mathcal{C}_{obst}$ while the other lies in $\mathcal{C}_{free}$. A different class of methods are adopting the opposite approach, namely trying to generate samples as far away from $\mathcal{C}_{obst}$ as possible by sampling the generalized Voronoi diagram of $\mathcal{C}_{free}$ (also called the *medial-axis*) [Wilmarth 1999, Lien 2003].

Note that as announced in section 1.2.2 we can see that different sampling strategies can be used and that they have a significant impact on the overall performances of a motion planning algorithm. In section 4.2.2 we propose a sampling strategy of our own designed to increase the efficiency of motion planning for a quadrotor.

Another drawback of random sampling is that it is not trivial to estimate the coverage quality of the free space. In the *Visibility-PRM* algorithm [Nissoux 1999, Siméon 2000], a simple on-line estimation of the proportion of the hyper-volume of the free space covered by the roadmap is proposed. With this information one can design a better stop condition for the learning phase than a maximum number of iteration or computing time. In this approach a sample is added to the roadmap only if it can be used to merge together two connected components or is not "visible" by any other node. The coverage estimation is then a function of the number of consecutive unsuitable sampled configurations. This approach has also the advantage of generating a roadmap containing fewer nodes, thus decreasing the number of local paths to be computed and therefore speeding up each iteration. Note that it also tackles very well the narrow passage problem.

Some methods are trying to increase efficiency by decreasing the number of calls to the collision checking module. Collision detection is indeed by far the most time consuming process of a classic PRM algorithm (about 90% of computing time [van Geem 2001]). In the *Lazy-PRM* algorithm [Bohlin 2000] the learning phase is performed without taking into account any of the obstacles. All calls to the collision checking module are done during the query phase, eliminating invalid local paths during the shortest path search. In the *Fuzzy-PRM* variant [Nielsen 2000], collision detection is performed with an incremental resolution which is leading to an overall decrease of the number of calls to the collision checking module.

Finally, some methods focus on optimality. Up to now we did not mention optimality because it was not relevant when considering the generalized mover's problem. Efficiently finding a solution if one exists was the only concern. However, aforementioned methods often produce low quality solutions. It has indeed been proven by [Karaman 2011] that a standard PRM does not converge towards the optimal solution. Optimality is relative to a given *cost function*. For example, length of the global solution path is often considered but some other criteria such

as clearance to the obstacles can be used. Whatever the considered cost function, a low quality solution can be problematic when trying to execute it on the physical system. A classic way to deal with this issue is to post-process the low quality solutions by means of an optimization method, also called a *smoothing method*. Examples are numerous but since it is not our point we won't give any of them here. Note that we will however give an example of smoothing method in section 4.1.2. An other way of dealing with optimality is to consider it while planning. This is is the case of a simplified version of PRM called *sPRM*, proposed in [Kavraki 1998], which converges towards the optimal. A more efficient proposition called PRM^* was since made by [Karaman 2011]. The next section presents a family of connection strategies known as the *diffusion-based methods*.

1.2.4 Diffusion-based methods

In this class of methods the underlying structure that is used to explore the free space is a special kind of graph: a *tree*, *i.e.* a graph with no cycles in it. More generally the structure is actually a *forest*, a set of trees. The idea here is to incrementally build this structure starting from either the initial configuration q_{init}, the final configuration q_{end} or both. The two main differences with the PRM-based methods is that no learning phase is required and not all $\mathcal{C}_{free}$ is explored. The search is indeed usually biased to solve one specific motion planning problem. This class of methods is thus often called *single queries methods*. A distinction is usually made between *unidirectional* and *multi-directional* methods. In the former a single tree is constructed with either q_{init} or q_{end} as its root until the other configuration is reached. In the latter several trees are constructed. In a *bidirectional* method in particular, two trees with roots q_{init} and q_{end} respectively are iteratively constructed until the two trees meet. These methods are particularly well suited to problems where either or both the initial and final configurations are in a highly constrained region of the free space as it is for example the case in the context of assembly maintainability [Chang 1995]. Choosing between an unidirectional and a bidirectional method usually depends on whether only one of the end configuration is constrained or both.

We have briefly mentioned the *Randomized Path Planner* (RPP) [Barraquand 1991] already as being the first randomized motion planning algorithm. By its use of the concept of random walks, it is also the first approach to diffusion-based methods.

Another approach called the *Ariadne's Clew Algorithm* (ACA) was proposed by [Bessiere 1993]. A genetic algorithm is used to generate a tree rooted at q_{init} while trying to optimize the distribution of its nodes other $\mathcal{C}_{free}$. In a second phase the goal configuration is tried to be linked to the tree.

In the bidirectional approach called *Expansive-Space Tree* (EST) and proposed by [Hsu 1997, Hsu 2000], the algorithm iteratively executes two steps labeled as

expansion and *connection* respectively. During the expansion step a node is selected in one of the two trees with a probability inversely proportional to the number of nodes lying in its neighborhood (according to a given metric and threshold). A given number of configurations are then sampled in this given neighborhood of the selected node and tried to be linked to it. The connection step consists in trying to link the newly added nodes to nearby nodes of the other tree.

Nowadays, the most popular diffusion-based method is without a doubt the *Rapidly-exploring Random Tree* (RRT) algorithm introduced by [Lavalle 1998]. Although initially thought as a tool to solve non-holonomic and kinodynamic motion planning problems (see next section) this approach has since been successfully adapted to problems without differential constraints [Kuffner 2000, LaValle 2000]. This method can be used either as unidirectional or bidirectional (or even multi-directional as we will see later). As in EST a construction phase and a connection phase are alternated but both the way the node to be expanded is selected in the tree and the way this expansion is made differ. At each iteration a random configuration, q_{rand}, is sampled in $\mathcal{C}$. Its nearest configuration in the tree, q_{near}, is then selected (according to a chosen metric). Given a steering method SM, as defined in (1.1), the local path $LP = SM(q_{near}, q_{rand})$ is generated. A node expansion process is then applied to generate a new configuration, q_{new}, in two possible ways according to the chosen version of the algorithm. For that, a fixed value $\varepsilon \in]0\ 1]$ has been previously set as a parameter of the algorithm. In the classic variant (usually referred to as *RRT-Extend*) the new configuration is $q_{new} = LP(\varepsilon)$. In the variant called *RRT-Connect* [Kuffner 2000] the expansion process is iterated as long as q_{new} is collision free. The expansion process (that we call *Expand*) is described in Algorithm 2 and illustrated in Figure 1.3 for the Extend version.

Algorithm 2: Node expansion of the RRT algorithm (Expand)

```
LP ← SM(q_near, q_rand);
q_new ← LP(ε);
if RRT-Connect then
    k ← 2;
    while kε ≤ 1 and LP(kε) ∈ C_free do
        q_new ← LP(kε);
        k ← k + 1;
    end
end
Return q_new;
```

If q_{new} is collision free, the local path $SM(q_{near}, q_{new})$ is tested for validity. If it is valid, both the node and the edge are added to the tree. The connection phase is then applied. In the unidirectional version the local path $SM(q_{new}, q_{end})$ is tested for validity (alternatively $SM(q_{new}, q_{init})$ if the tree is rooted at q_{end}). If it is valid the algorithm stops. In the bidirectional version (that we refer to as

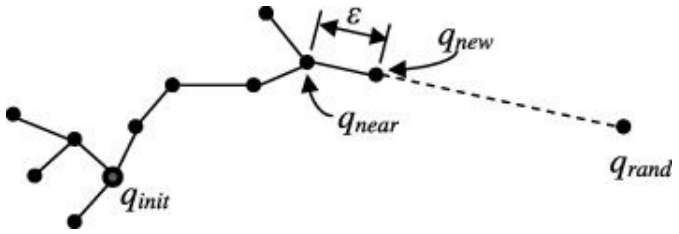


Figure 1.3: One expansion step of the RRT algorithm for the Extend version.

Bi-RRT) q_{new} is tried to be linked to nearby nodes in the other tree (as in EST). If one of the attempted connections is successful the algorithms stops. Note that in this bidirectional version each tree is alternatively expanded (as in EST). In a variant referred to as *balanced* the trees are kept with the same number of nodes. These steps are iterated until a maximum number of iteration has been reached. The unidirectional version of the algorithm is described in Algorithm 3 for a tree rooted at q_{init}.

Algorithm 3: Unidirectional RRT

$N \leftarrow \{q_{init}\}$;
$E \leftarrow \emptyset$;
$T \leftarrow \{N, E\}$;
$k \leftarrow 0$;
while $k < K$ **do**
  $q_{rand} \leftarrow$ random configuration;
  $q_{near} \leftarrow \text{NearestNeighbor}(q_{rand}, N)$;
  $q_{new} \leftarrow \text{Expand}(q_{near}, q_{rand})$;
  if $q_{new} \in \mathcal{C}_{free}$ **and** $SM(q_{near}, q_{new})$ *is valid* **then**
    $N \leftarrow N \cup \{q_{new}\}$;
    $E \leftarrow E \cup \{(q_{near}, q_{new})\}$;
    if $SM(q_{new}, q_{end})$ *is valid* **then**
      $N \leftarrow N \cup \{q_{end}\}$;
      $E \leftarrow E \cup \{(q_{near}, q_{end})\}$;
      Return T;
    end
  end
  $k \leftarrow k + 1$;
end
Return T;

A fundamental property of RRT is the Voronoi biasing. The probability that a node is selected for expansion is proportional of the volume of its Voronoi region. This implies that the algorithm rapidly explore the free space because it favors diffusion towards unexplored regions. Figure 1.4 shows the evolution of a tree constructed by the RRT algorithm and covering the free space.

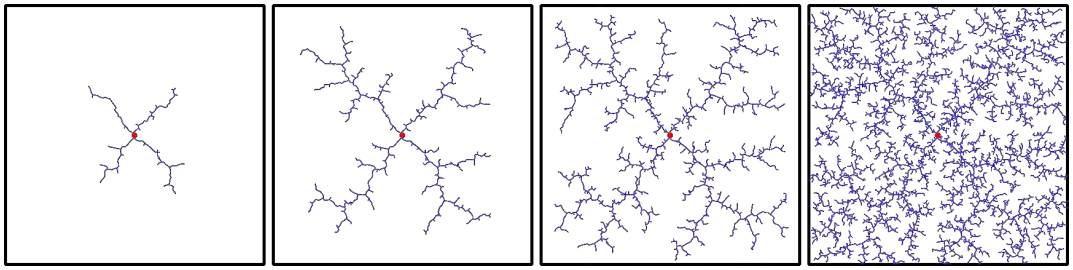


Figure 1.4: Evolution of a tree constructed by the RRT algorithm and covering the free space.

This method has since gave birth to many variants. For instance a *lazy* bidirectional version is proposed by [Sánchez 2003]. Similarly to PRM-based methods, some variants are targeting the sampling strategy. A *obstacle-based* variant tackling the narrow passage problem is proposed by [Rodriguez 2006] while a different approach avoiding the dense regions in proposed by [Khanmohammadi 2008]. In the *Dynamic-Domain RRT* [Yershova 2005] and its adaptive version [Jaillet 2005], the notion of visibility is used to better refined the sampling domain. Another approach based on dimension reduction uses principal component analysis techniques to better guide the exploration inside narrow passages [Dalibard 2009].

The Exploring/Exploiting Tree algorithm (EET) proposed by [Rickert 2008] is a combination between a potential field approach and a RRT-based approach. Information about the connectivity of the workspace gathered during exploration is exploited to compute a navigation function that defines a potential field. The planner gradually switches to a diffusion-based exploration when the exploitation method fails.

Quality of the solutions is also a concern for the diffusion-based methods. Optimality is asymptotically achieved by the algorithm *RRT** proposed by [Karaman 2011]. In this version the tree is locally reorganized each time a new node is added. More generally, generating high-quality paths with respect to a cost functional have been investigated by several authors. As previously mentioned, a cost function can assess the quality of a path but it is also possible to define the cost of a configuration. When such a functional is defined other the configuration space, authors often talk about the *cost space* of the system. Motion planning in a cost space is therefore referred to as *cost-space path planning*. In this context, the *Threshold-based RRT* (RRT$_{obst}$) was proposed by [Ettlin 2006b] for rough terrain navigation. The idea is to decide whether to accept or reject a new configuration generated by the RRT expansion step according to its cost. A multi-directional version (where more than two trees are constructed) can be found in [Ettlin 2006a]. In the *Transition-based RRT* (T-RRT) algorithm proposed by [Jaillet 2010], the idea is generalized to any cost function defined on $\mathcal{C}$. The transition test is based on the Metropolis test. A bidirectional version is proposed by [Devaurs 2013] and a multi-

tree variant can be found in [Devaurs 2014]. Finally, optimally is asymptotically achieved in two variants called T-RRT* and *Anytime T-RRT* [Devaurs 2015].

In the next section we address the kinodynamic motion planning problem.

1.3 Kinodynamic motion planning

In section 1.2.2, a brief definition of the kinodynamic motion planning problem has been given. In this section we give a more precise formulation of the problem and a state of the art of the literature on the subject.

1.3.1 Problem formulation

We define the *state* of a system as

$$\mathbf{x} = \begin{bmatrix} q \\ \dot{q} \\ \vdots \\ q^{(p)} \end{bmatrix} \in \mathcal{X}$$

with $q \in \mathcal{C}$ the configuration, $p \geq 1$ and $\mathcal{X}$ the *state space* of the system (also sometimes the *phase space*). Note that in the classic formulation $p = 1$ [Canny 1988b, Donald 1993], meaning that only position and velocity of the system are considered. For a configuration space of dimension n, the state space is therefore of dimension $n(p+1)$.

A mechanical system is typically controlled by a set of actuators. In the case of a robotic arm for example, these are the motors acting on the joints. This set of controls is modeled as a time-dependent vector of real values called the control variables. In the case of an arm it could be for example the tensions applied to each motor at a given time. This vector $\mathbf{u}(t) \in \mathcal{U} \subset \mathbb{R}^m$ is called the *control* (or the *command*) and $\mathcal{U}$ is the *control space*. The future state of the system thus depends on its current state and the applied control. The set of differential equations modeling this dependency is referred to as the equations of motion of the system:

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t))$$

where the function f is specific to the dynamics of the system. Given a specified control $\mathbf{u}(.)$ and initial conditions $\mathbf{x}(t_0) = \mathbf{x}_0$, a solution $\mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(.)$ to the equations of motion is called a *response* to the control $\mathbf{u}(.)$ for the initial conditions $\mathbf{x}_0$ at t_0.

The control is typically restricted to a certain *control region*, meaning that $\mathcal{U} \subsetneq \mathbb{R}^m$. Furthermore a piecewise continuous control $\mathbf{u}(.)$ defined on some time interval $t_0 < t < t_F$ with range on the control region $\mathcal{U}$ is called an *admissible*

control on $[t_0\ t_F]$. Note that an admissible control is necessarily bounded.

A system can also be submitted to physical constraints arising from its dynamics. These are functional equalities and/or inequalities restricting the range of values that can be assumed by control and/or state variables. Finally an admissible control on $[t_0\ t_F]$ is said to be *feasible* if and only if a response $\mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(.)$ exists and is defined for each $t \in [t_0\ t_F]$, and both $\mathbf{u}(.)$ and $\mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(.)$ satisfy all the physical constraints. We will note $\mathcal{U}_{[\mathbf{x}_0,t_0,t_F]}$ the set of feasible controls on $[t_0\ t_F]$ for the initial state $\mathbf{x}_0$. A similar definition of $\mathcal{C}_{free}$ can be given for the state space. We note $\mathcal{X}_{unvalid}$ the set of states that are either in collision or not satisfying the physical constraints, and we note $\mathcal{X}_{valid} = \mathcal{X} \setminus \mathcal{X}_{unvalid}$.

Given an initial state $\mathbf{x}_0$ at t_0 and a final state $\mathbf{x}_F$ both in $\mathcal{X}_{valid}$, the kinodynamic motion planning problem is then the one of finding both $t_F > t_0$ and $\mathbf{u} \in \mathcal{U}_{[\mathbf{x}_0,t_0,t_F]}$ such that:

$$\begin{cases} \forall t \in [t_0\ t_F],\ \mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(t) \in \mathcal{X}_{valid} \\ \mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(t_0) = \mathbf{x}_0 \\ \mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(t_F) = \mathbf{x}_F \end{cases}$$

We mentioned earlier in section 1.2.3 that some variants of the methods proposed to solve the generalized mover's problem were focusing on the quality of the solutions. This is even more the case for the kinodynamic motion planning problem since the earliest formulations were looking for the minimum-time solution. This is often still considered to be a relevant quality criterion for the kinodynamic motion planning problem (like path length is for the holonomic case) although other cost-functions can be considered.

In the next subsections we present the methods that have been proposed to solve the kinodynamic motion planning problem.

1.3.2 Decoupled approach

A classical way to approach the kinodynamic motion planning problem is to decompose it into two simpler subproblems: (1) planning a geometric path that respects kinematic constraints (collisions and possibly non-holonomy) and (2) planning the derivatives of the configuration variables along the path with respect to physical constraints. The first subproblem (often called the *path planning problem*) is exactly the one we described in section 1.1.2 and can thus be treated with one of the previously presented approaches. In the context of kinodynamic planning, a solution to the path planning problem is called a *quasi-static* solution because it can be seen as a trajectory in which each state has a zero velocity and therefore can be executed at very low velocity. The second subproblem is sometimes referred to as the *velocity planning problem* and the whole two steps approach as the *path-velocity*

decomposition [Kant 1986]. Velocity however is not always the only derivative that has to be considered and this is why we rather use the more general term: *decoupled approach*.

Solving the velocity planning problem implies finding a suitable time parametrization of the precomputed path. Usually, methods used to achieve this depend on the system. For manipulators for instance, the minimum-time solution is found by solving an optimal control problem in one dimension [Bobrow 1985, Shin 1985]. It has indeed been shown that torques of the actuators and there bounds can be written in function of position, velocity and acceleration of the end effector along the specified path. See [Shiller 1991] for an example of usage in a global motion planner. For other systems such as car-like robots, a feasible path can be "smoothed" (with a *smoothing method* that can be derived from a steering method) into a feasible trajectory ([Fleury 1995, Lamiraux 2001] and [Lamiraux 1998] for a car-like robot towing a trailer). For other mobile robots specific optimization procedures can be used, such as the one based on quintic Bézier splines proposed by [Lau 2009]. We finally can fine decoupled approaches used in the context of motion planning for aerial robots, see for instance [Richter 2016] (more examples in section 1.4). This approach is also very useful for planning in a dynamic environment, i.e. moving obstacles [Fraichard 1998] and/or multiple robots [Peng 2005].

Decoupled approaches can be very efficient tools but they have two major drawbacks. First, obtaining good quality paths is challenging since the considered cost-function may not even be defined in the configuration space. Minimum time is a good example. Since time is not considered during path planning it is hard to judge the quality of a solution path. One could argue that the shortest path could be a good bet but it is often the case that the fastest path is actually not the shortest one. Second, although a kinodynamic motion planning problem can have solutions, a decoupled approach could fail to find them. Indeed, if the problem has no quasi-static solution then the path planning step will fail. This is for example the case for a quadrotor that has to be tilted in order to go through a narrow slot-shaped passage (see section 4.2.4). In this instance, the planning process has to take place directly in the control space or the state space. This is often referred to as *direct planning*.

1.3.3 Direct planning

A direct planning method searches solutions directly in the control space or the state space rather than in the configuration space. Several options have been investigated. We will subdivide those into two main families: the deterministic approaches and the sampling-based methods.

1.3.3.1 Deterministic approaches

This family of methods of direct planning does not use randomness.

Optimal control:

If the system is simple enough, *optimal control* can be applied [Brockett 1982, Lewis 1995]. The problem is that it does not scale well and closed form solutions are only available for point mass systems in one [Ó'Dúnlaing 1987] or two dimensions [Canny 1990]. Plus, taking into account the kinematic constraints arising from the obstacles is not easy.

Numerical optimization:

An other possibility is to use *numerical optimization* techniques [Fernandes 1993, Betts 1998, Ostrowski 2000]. Problems are that these can be computationally expensive when applied to global trajectory planning and that they often get trapped in local minima. Although this particular drawback have been recently addressed [Zucker 2013, Schulman 2014], highly dynamical problems are still a challenge to these methods.

Grid search:

One of the earliest algorithm for kinodynamic motion planning [Sahar 1986] proposed to tessellate the joint space in order to find minimum-time trajectories for a robot arm. A best first graph search is performed other the tessellated joint space by using a dynamic scaling algorithm to determine velocity at each node in function of previous position and velocity.

In [Canny 1988b, Donald 1993] a breath-first search is performed on a discretized state space. To expand one state in the grid all possible combinations of saturated and null controls are applied during a fixed time step. This technique is applied to a point mass under Newtonian mechanics with velocity and acceleration bounds in 2D or 3D. This is the first provably good approximation of a solution to the minimum-time trajectory problem that is running in polynomial time. The approach has then been extended to more complicated systems [Donald 1995a, Donald 1995b, Heinzinger 1990, Reif 1997]. This method has also been used by [Fraichard 1993] to solve the *highway problem* in near-optimal time.

The problem here is the same as in the holonomic case. Although resolution complete, these methods suffer from the curse of dimensionality. Complexity is indeed exponential in the resolution and since the higher dimension goes the finer resolution has to get, these approaches do not scale well.

1.3.3.2 Sampling-based methods

The same idea of sampling-based planning used in the holonomic case can be applied to kinodynamic motion planning. One difference though is that $\mathcal{X}_{valid}$ takes the role of $\mathcal{C}_{free}$, meaning that valid states are sampled instead of collision free configurations. Another big difference is the steering method.

Steering-methods:

In order to adapt sampling-based planning methods to the kinodynamic case one has to rethink the steering method. Connecting two states by a valid trajectory in the absence of obstacles is a well known problem called a *two-point boundary value problem* (BVP). It involves solving the equations of motions with both initial an final conditions and possibly under constraints on the state. This is not easy in general and can be computationally quite heavy. But for specific systems a closed form solution to the BVP exists and in this case it can be used as a steering method in the state space.

In the context of optimal kinodynamic motion planning for instance, the RRT* algorithm has been adapted to the kinodynamic case by [Karaman 2010]. Sufficient conditions on the controllability of the system are provided for optimality. The method uses a steering method specific to the system. In this example the Dublins' vehicle, the double integrator and a combination of both, used as a simple 3D airplane model, are considered.

Linearizing the dynamics

When the system dynamics are linear, it may be possible to solve the BVP efficiently in closed form. Furthermore, it is possible to apply this approach to non-linear systems by linearizing the dynamics about an operating point. This approach has for instance been investigated by [Perez 2012] by proposing the LQR-RRT* algorithm. Linear quadratic regulation (LQR) is used within a RRT* algorithm both as a metric and a steering method. This approach has since been extended by [Goretkin 2013] to samples in the state-time space and to deal with quadratic cost functions.

Motion primitives:

A possible way of avoiding the difficulties of solving the BVP is to use motion primitives, i.e. precomputed solutions to the BVP. For instance a *Maneuver Automaton* is used by [Frazzoli 2000] as a steering method within a RRT-based algorithm. The states of the automaton are steady stable trajectories called *trim trajectories* and the transitions are maneuvers. In this work the authors are using a non linear controller in order to generate the primitives. The approach is demonstrated on a small autonomous helicopter. State lattice motion primitives are used by [Pivtoraiko 2011] in both deterministic (A* and D*) and probabilistic global planners (PRM and RRT). Motion primitives are generated with a BVP solver by

sampling the state space. In the probabilistic case, the same state sampling strategy used off-line to generate the lattice is used on-line during the exploration. This approach is therefore resolution complete. Similarly an off-line learning phase is performed by [Allen 2015]. During this phase a set of states are sampled in the free state space and the BVP problem is numerically solved between either some randomly chosen couples or all of them. A lookup table of the costs of the generated primitives is generated to be used as a steering method during the on-line phase. Machine learning techniques are also used to learn the reachability set of the samples. This information plays the part of the metric. The on-line step consists in a adaptation of the *Fast Marching Trees* (FMT) algorithm proposed by [Janson 2015] called *kino*-FMT. The approach is applied to both a fixed-wing UAV and a gravity-free spacecraft.

Forward propagation:

Finally, the most investigated approach that is also avoiding the BVP is forward propagation of the dynamics. In these methods new states are not generated through sampling but by applying a feasible control to an already generated state during a given period of time. This is done by integrating the equations of motion using your favorite ODE solver (e.g. fourth-order Runge-Kutta integrator). Computationally speaking this is way cheaper than trying to numerically solve the BVP. There is thus no need here for a steering method. In fact, because there is no steering method, this approach cannot be used within a probabilistic roadmap and is therefore only suitable for diffusion-based methods.

For instance it was first used within a RRT by [LaValle 2001]. At each iteration a new state $\mathbf{x}_{rand}$ is sampled in $\mathcal{X}_{valid}$. According to some metric (weighted Euclidean here) the nearest state in the tree $\mathbf{x}_{near}$ is selected. Given a time T (either randomly chosen or arbitrarily fixed), a random constant control $\mathbf{u}_{rand}$ is sampled in $\mathcal{U}$ and applied to $\mathbf{x}_{near}$ in order to generate a new state $\mathbf{x}_{new} = \mathbf{x}_{[0,\mathbf{x}_{near},\mathbf{u}_{rand}]}(T)$. An alternative is to chose among several randomly generated controls the one that brings the system the closer to $\mathbf{x}_{rand}$ (according to the metric). If the resulting trajectory $\mathbf{x}_{[0,\mathbf{x}_{near},\mathbf{u}_{rand}]}(.)$ (the response) is valid it is added to the tree together with $\mathbf{x}_{new}$. In the unidirectional version, the algorithm stops if $\mathbf{x}_{new}$ is close enough to the goal, and if it is close enough to the closest state in the other tree in a bidirectional version.

This same idea of forward propagation of the dynamics can also be used as is into the EST algorithm. This has been done by [Kindel 2000, Hsu 2002] with however a difference introduced by the fact that moving obstacles are considered. Sampling takes place in the state-time space witch is the state space augmented of the time dimension (notion introduced by [Fraichard 1998]).

These methods are using a metric in the state space in order to select the state to be propagated and thus to guide the search toward unexplored regions. The problem is that finding a good metric in the state space is not easy. In fact the

best metric would be the optimal cost-to-go but finding it is usually as hard as solving the BVP (see [LaValle 2001]). Plus, by applying a random control there is no reason that the system would be propagated in the direction of the sampled state and even if the best control is chosen among several tries this choice is based on the metric. In order to reduce the impact of the metric on the overall performances of the planner several approaches have been proposed.

For instance, an affine quadratic regulator (AQR) design has been used by [Glassman 2010] to approximate the exact minimum-time distance pseudo-metric at a reasonable computational cost.

The notions of constraint violation frequency and exploration information (success or failure of a control applied to a state) have been used by [Cheng 2001] for node selection in order to reduce the metric influence. The problem is that, as is, the algorithm is only complete for a certain class of problem. With the addition of a discretization of the state space, in order to exclude repeating states, resolution completeness has been obtained by [Cheng 2002].

A different approach called the *Path Directed Subdivision Tree* (PDST) has been proposed by [Ladd 2004]. The tree structure here is not the same: the nodes represent valid trajectories and edges are branch states (the first sate of a trajectory). The selection schedule is deterministic, greedy and the metric plays no part in it. It is based on a weighted priority of the nodes that doubles each time a trajectory is selected (lowest priority nodes are selected first). Information about coverage and exploration efficiency is maintained thanks to an adaptive subdivision scheme of the state space. The overall algorithm remains probabilistic though since that in order to expand a node, a random state is selected in the trajectory and a random control is applied to it thus generating a new trajectory (i.e. a new node). This approach has then been adapted by [Bekris 2007] in the context of re-planning using sensor information in the *Greedy Incremental Path-directed planner* (GRIP).

The *Discrete Search Leading continuous eXploration* (DSLX) planner proposed by [Plaku 2007] does not require a metric in the state space at all. It is still a sampling-based diffusion method forward propagating a tree in the state space but it uses a coarse-grained decomposition of the work space into regions and the projections of the states in the tree onto those regions in order to guide the search. The partition of the work space is associated with its adjacency graph in which an edge e_{ij} represents the fact that the two regions R_i and R_j are adjacent. These edges are weighted according to the frequency of exploration of regions at both ends and the average increase of coverage obtained by the previous exploration of those regions. At each iteration, a sequence of adjacent regions (called a *lead*) going from the start to the goal region is selected in the graph with a probability partly based on the edges weights. Non empty regions (i.e. containing at least one projected of state from the tree) are then selected in this lead with a probability based on their closeness to the goal (in terms of graph distance, no metric required) and their frequency of exploration. A selected region is then explored by selecting in it

the less selected state so far and applying a random control to it for several fixed durations until a collision occurs. Actually, the applied control, selected among several random tries, is the one that brings the system the closest to the next region in the lead. No need here for a metric in the state space, the Euclidean metric in the work space is used on the projections of the generate states. Weights are then updated and the algorithm loops until a solution is found (or maximum time as passed).

The *Informed Subdivision Tree* (IST) algorithm proposed by [Bekris 2008] first computes a roadmap that captures the connectivity of the configuration space using an approach such as PRM. IST then uses information from the roadmap to bias the tree expansion towards the goal in the state space. To avoid getting stuck in local minima induced by state space constraints, which is the risk of a greedy approach based on configuration space biases, IST employs the adaptive subdivision scheme from PDST and an edge penalization scheme.

The *Reachability-Guided RRT* (RG-RRT) algorithm proposed by [Shkolnik 2009] builds and maintains a standard RRT. The primary differences lie in the use of a node's reachable set to focus sampling on regions of the state space that are most likely to promote expansion under the differential constraints. By use of this adaptive sampling strategy the algorithm alleviates the sensitivity of randomized sampling for systems with differential constraints to the metric that is employed to expand the tree. It indeed utilizes the metric only for regions of the state space for which it is valid.

In the *Kinodynamic Planning by Interior-Exterior Cell Exploration* (KPIECE) proposed by [Şucan 2009, Şucan 2012] ideas from PDST and DSLX are combined. A chosen projection space $\mathcal{E}(\mathcal{X})$ of the state space $\mathcal{X}$ (e.g. the configuration space, the work space, etc.) is decomposed into a multi-level cell discretization such that each level is a partition of the previous one, the first level being the one with the finest resolution. Like in PDST, nodes in the exploration tree are motions represented by an initial state, a control and a duration. Formally it is a triple $(\mathbf{x}_0, \mathbf{u}, T)$. A node is considered to be inside a cell if at least one of its state (i.e. $\mathbf{x}_{[0,\mathbf{x}_0,\mathbf{u}]}(t)$ with $t \leq T$) is in the cell once projected onto $\mathcal{E}(\mathcal{X})$. When a motion is added to the tree it is subdivided so that each node added to the tree belongs inside exactly on cell at each level. This allows to identify each node to a unique sequence of cells from each level, one cell in the sequence being contained in the next one. This is called a cell chain. At each iteration of the algorithm, a cell chain is deterministically chosen by incrementally choosing at each level (starting from the last, i.e. the coarser) a cell according to its coverage (number of non empty cells of the previous level contained in it), frequency of selection and whether it is an interior or an exterior cell (based on the number of non-empty neighboring cells). A node $(\mathbf{x}_0, \mathbf{u}, T)$ is then chosen in the last cell of the chain according to a half-normal distribution. A time parameter t is uniformly sampled in $[0\ T]$ and a new motion is generated by applying a random control during a random period of time to the state $\mathbf{x}_{[0,\mathbf{x}_0,\mathbf{u}]}(t)$. Note that in this

algorithm no metric at all is required in any of the considered spaces.

Principal Component Analysis (PCA) has been used by [Li 2010] in order to counter undesirable biases introduced by the dynamics. During an offline training phase, PCA is executed on the coordinates of the nodes of a tree grown in the state space using RRT-Connect in order to learn the biases introduced by the dynamics. During the online planning phase the exploration procedure is modified so as to promote the propagation of the search tree towards the direction of the least significant components.

The *Stable Sparse RRT* (SST) algorithm proposed by [Li 2016] is using a classic selection/propagation scheme but with some modifications. The selection procedure called *Best First Selection* selects in a neighborhood (based on Euclidean metric) of a uniformly sampled node $\mathbf{x}_{rand}$ the node $\mathbf{x}_{BestNear}$ with the best path cost to the root. The propagation procedure called *MonteCarlo-Prop* selects a random constant control $\mathbf{u}_{rand}$ and applies it to $\mathbf{x}_{BestNear}$ for a random period of time T_{rand} chosen in a predetermined interval. Follows a pruning step that allows sparsity of the data structure. A set of witness states is maintained during the exploration. Given a predefined distance δ, a node newly added to the tree is tagged as witness if and only if it is at a distance (Euclidean metric) greater than δ from all the other witnesses. This way each node in the tree can be associated to exactly one witness from which it is at distance lesser than δ. This defines a neighborhood for each node: the set of nodes sharing the same witness. After the propagation step, if the response path $\mathbf{x}_{[0,\mathbf{x}_{BestNear},\mathbf{u}_{rand}]}(.)$ is collision free and if its end state $\mathbf{x}_{new} = \mathbf{x}_{[0,\mathbf{x}_{BestNear},\mathbf{u}_{rand}]}(T_{rand})$ has the best path cost in its neighborhood, all child-less nodes in the neighborhood with higher path cost are removed from the tree. In addition all remaining nodes in the neighborhood are tagged as inactive while $\mathbf{x}_{new}$ is tagged as active. This is useful for the selection step because this way only active nodes have to be considered. While SST is asymptotically near-optimal the authors proposed in the same work an anytime variant called SST* that is asymptotically optimal.

1.4 Motion planning for aerial robots

In this section we give an overview of the literature on motion planning for aerial robots. Some of the references bellow can be found in a more comprehensive review done by [Goerzen 2010].

1.4.1 Trajectory generation

In motion planning, a distinction is usually made between global and local planning. Classically the global planning problem is divided into several local planning problems. Up to now we referred to the local planner as the steering method. In the

context of aerial robotics, it consists in producing flyable trajectories between configurations or states of the UAV without taking obstacles into account. It is often referred as to the trajectory generation problem (also sometimes trajectory planning in the context of control theory). It is usually the case that authors propose trajectory planners specifically designed for a particular system.

We give here some examples of trajectory generation for quadrotors. Model predictive control has been used by [Singh 2001] to generate trajectories interpolating a given set of way-points. It has been proposed by [Cowling 2006] to minimize a cost function for one polynomial for each flat output (see section 2.4). This polynomials could be classically parametrized or either Laguerre or Chebyshev polynomials could be used. Real-time generation of minimum snap trajectories (actually minimum integral of squared snap) passing through a sequence of 3-D positions and yaw angles (key-frames), while ensuring safe passage through specified corridors and satisfying constraints on velocities, accelerations and inputs have been proposed by [Mellinger 2011]. Trajectory generation under constraints for a quadrotor from any state to hovering state has been proposed by [Hehn 2011]. The three degrees of freedom are decoupled, and time-optimal jerk trajectories are planned for each of them separately. The feasibility of the planned trajectories is then checked. If it is found to be infeasible, it is re-planned with reduced jerk constraints, which eventually guarantees feasibility. Similarly, optimal control theory principles have been used by [Mueller 2013] to generate in real-time smooth trajectories between non-hovering states. Each axis x, y and z is treated as a triple integrator with jerk as control input. The BVP is then independently solved for each axis, without constraints on the state and for a given time. The method is very fast but does not guaranty feasibility, i.e. respect of the physical bounds of the system. A very fast feasibility test is provided that allows to filter invalid trajectories. Finally, we can mention [Spica 2012b] that uses circular arc primitives in order to plan a trajectory that connects two arbitrary states while allowing the UAV to grasp a moving target at some intermediate time.

For other systems such as fixed-wing UAVs, the feasibility constraints are not the same. Here, maximum curvature of the path is the limiting factor. A solution to simultaneous arrival of multiple UAVs has been proposed by [Shanmugavel 2006] using Pythagorean hodograph curves. Later, Dubins path with clothoid arcs have been used by [Shanmugavel 2010].

1.4.2 Planning in the workspace

A simplified version of the motion planning problem for aerial robots is to plan a trajectory for the center of mass of the UAV and assume that the control algorithms will be able to follow it. Workspace and configuration space are in this particular case the same. Therefore some authors have proposed to plan directly in the workspace or at least a discretized version of it. For instance LADAR data

is used by [Vandapel 2005] to create a roadmap of free-space balls in the 3D Euclidean workspace of a UAV. Planning is done using graph search techniques in this roadmap. The differential flatness of a quadrotor implies planning smooth and correctly bounded trajectories for linear position and yaw angle (see section 2.4). In this spirit, mixed-integer programming techniques have been used by [Deits 2015] to generate such trajectories. The idea is to use a discretization of the free workspace into convex free regions. Each region is associated with a parametrized polynomial which takes values entirely in the region. A mixed-integer optimization problem that consists in selecting the sequence of regions and parameters of the polynomials such that their concatenation is a smooth, collision free and flyable trajectory from start to goal while minimizing either acceleration, jerk (first derivative of acceleration) or snap (second derivative of acceleration) is then solved.

1.4.3 Decoupled approach

The decoupled approach presented in section 1.3.2 being a simple an intuitive way of solving the kinodynamic motion planning problem, it has naturally often been applied to aerial robots. A Voronoi-based planner followed by spline smoothing for trajectory formation has been proposed by [Judd 2001]. The A* algorithm followed by direct optimization of the trajectory using an RTABU search has been used by [Suzuki 2005]. A multi level architecture has been used by [Scherer 2007]: an evidence grid with a Laplacian-based potential method as the outer loop, a reactive planner (dodger) to enforce soundness, a speed controller to convert the path into a trajectory, and an inner loop flight controller. The design and characterization of the inner-loop control law used in such a multi-level decoupled controller for an unmanned rotorcraft based on two types of path planners (quasi-3d implementations of an A* and a Voronoi-based planner) is covered by [Takahashi 2008]. The implementation of a two path planner modules is described by [Howlett 2007]. Model predictive control is used by [Singh 2001] to both generate and follow trajectories interpolating a given set of way-points. Cubic curves are used by [Wzorek 2006] to smooth paths generated by either a PRM or a RRT algorithm using linear interpolation as a steering method. The RRT* algorithm was used by [Richter 2016] to find a collision-free piece-wise linear path through the environment for the flat outputs (x, y, z, yaw) of a quadrotor (see section 2.4 for more details), initially considering only the kinematics of the vehicle and ignoring the dynamics. That path is then pruned to a minimal set of way-points, and a sequence of polynomial segments is jointly optimized to join those way-points into a smooth minimum-snap trajectory from start to goal. Utilizing a differentially flat model of the quadrotor and the associated control techniques, these paths can precisely be followed.

1.4.4 Finite-State Motion Model: The Maneuver Automaton

Motion primitives have also been used in the context of aerial robots. A set of solutions to the two-endpoint boundary value problem as a motion primitive set is stored by [Yakimenko 2000], but the approach does not deal with obstacles. We already cited the maneuver automaton used by [Frazzoli 2000] as a steering method within a RRT-based algorithm. The concept of a maneuver automaton for human piloted acrobatic flight has been investigated by [Piedmonte 2000, Gavrilets 2001]. The concept of the maneuver automaton within a receding horizon optimization framework has been used by [Schouwenaars 2004]. An A*-based planner (graph search) using motion primitives to connect (x, y, z, yaw) states has been used by [MacAllister 2013].

Chapter 2

Modeling and motion control of a quadrotor

Contents

The focus of this chapter is the quadrotor system. We will first give a model of its dynamics. This will allow us, after a brief introduction to the general principles of control theory, to define its control space and its state space. Its physical constraints in the control space will then be discussed. Finally, we will see that the system is differentially flat, which is having an impact on planning. Thanks to differential flatness we will indeed show that planning can be done in the space of the flat outputs rather than in the state space.

2.1 Quadrotor dynamics model

We present in this section our chosen model of a quadrotor system, represented in Figure 2.1. It consists of four propellers located at points P_i such that $\{P_1, P_2, P_3, P_4\}$ is a square centered in G, the center of mass of the system. We choose an orthonormal inertial reference frame $\mathcal{I} = \{O, \mathbf{i_1}, \mathbf{i_2}, \mathbf{i_3}\}$ and we note:

$$d = \|\overrightarrow{GP_i}\|_{i=1..4},\ \mathbf{b_1} = \frac{\overrightarrow{GP_1}}{d},\ \mathbf{b_2} = \frac{\overrightarrow{GP_4}}{d},\ \mathbf{b_3} = \mathbf{b_1} \times \mathbf{b_2},$$

We thus defined an orthonormal body-fixed frame $\mathcal{B} = \{G, \mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}\}$. We finally note $\mathbf{r} = \overrightarrow{OG} = [x\ y\ z]^T$ the position of the center of mass in $\mathcal{I}$ and $R \in SO(3)$ the rotational matrix from $\mathcal{B}$ to $\mathcal{I}$.

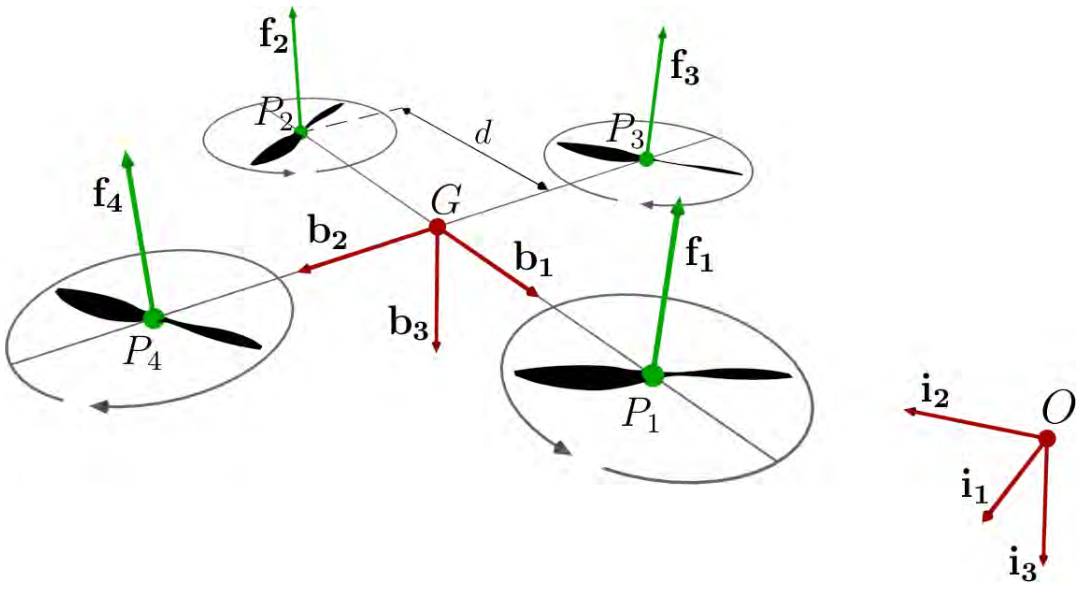


Figure 2.1: Quadrotor model

Each rotor i is rotating about the $\mathbf{b_3}$ axis at angular speed $\omega_i \in \mathbb{R}$ thus generating a thrust force $\mathbf{f_i} = -f_i\mathbf{b_3} \in \mathbb{R}^3$ applied at P_i. Note that this thrust is proportional to the square of the angular speed of the propeller: $f_i = c_{f\omega}\omega_i^2 \in \mathbb{R}$. This rotation is also generating a torque $\tau_i = \pm c_{\tau f} f_i \in \mathbb{R}$ about the $\mathbf{b_3}$ axis. The sign of τ_i depends on the orientation of the rotation (clockwise or counterclockwise). According to the orientations represented in Figure 2.1 we have $\tau_{1,2} = +c_{\tau f} f_{1,2}$ and $\tau_{3,4} = -c_{\tau f} f_{3,4}$. The constants $c_{f\omega}$ and $c_{\tau f}$ depends on the geometrical properties of the propellers.

Thrust forces are resulting in a total thrust force $\mathbf{f} = -f\mathbf{b_3}$ applied at G, with $f = \sum_i f_i$. Forces $\mathbf{f_3}$ and $\mathbf{f_4}$ are generating a moment about the $\mathbf{b_1}$ axis $M_1 = d(f_3 - f_4)$. Similarly, forces $\mathbf{f_1}$ and $\mathbf{f_2}$ are generating a moment about the $\mathbf{b_2}$ axis $M_2 = d(f_1 - f_2)$. Finally torques τ_i are resulting in a moment about the $\mathbf{b_3}$ axis $M_3 = \sum_i \tau_i$. We note $\mathbf{M} = [M_1\ M_2\ M_3]^T$ the total moment in the body-fixed frame and we have:

$$\begin{bmatrix} f \\ \mathbf{M} \end{bmatrix} = \Gamma \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}, \quad \text{with } \Gamma = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & d & -d \\ d & -d & 0 & 0 \\ c_{\tau f} & c_{\tau f} & -c_{\tau f} & -c_{\tau f} \end{bmatrix}$$

We finally note:

$\mathbf{\Omega} = [\Omega_x\ \Omega_y\ \Omega_z]^T \in \mathbb{R}^3$ the angular velocity in $\mathcal{B}$
$m \in \mathbb{R}$ the total mass
$J \in \mathbb{R}^{3\times3}$ the inertia matrix with respect to $\mathcal{B}$
$g \in \mathbb{R}$ the Earth gravitational acceleration
$\mathbf{e_3} = [0\ 0\ 1]^T$

Newton's second law of motion states that $m\ddot{\mathbf{r}} = \mathbf{W} + \mathbf{f}$ with $\mathbf{W} = mg\mathbf{e_3}$ the weight force. Note that by definition $R = [\mathbf{b_1}\ \mathbf{b_2}\ \mathbf{b_3}]$ hence $\mathbf{f} = -f\mathbf{b_3} = -fR\mathbf{e_3}$. The first equation of motion which describes the linear dynamics of the system can thus be written as follows:

$$m\ddot{\mathbf{r}} = mg\mathbf{e_3} - fR\mathbf{e_3} \tag{2.1}$$

Direct writing of the Euler's rotation equations gives the second equation of motion which describes the angular dynamics of the system:

$$J\dot{\mathbf{\Omega}} + \mathbf{\Omega} \times J\mathbf{\Omega} = \mathbf{M} \tag{2.2}$$

2.2 Motion control of a quadrotor

In this section we give some insights about the motion control of a quadrotor *i.e.* how a given planned trajectory can actually be executed by the physical system. In 1.3.1 we introduced the fact that the future state of the system at a given time t depends on its current state $\mathbf{x}(t)$ and the applied control $\mathbf{u}(t)$. We recall that the set of differential equations modeling this dependency is referred to as the equations of motion of the system: $\dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t))$, where the function f is specific to its dynamics. We gave those equations for our model in the previous section. The field of study which is concerned with the way of influencing the behavior of such systems is called *control theory*. It is not the focus of this thesis but in order to plan a flyable trajectory for a quadrotor it is important to understand some of its general principles.

2.2.1 A brief introduction to control theory

The objects being considered here are *dynamical systems*. Formally a dynamical system is a time-invariant "rule" that describes the evolution over time of a point in its ambient space. More specifically, such a system *outputs* (its state) are a deterministic function of its *inputs* (the control). Control theory studies how to influence its behavior. Roughly put, we seek to gain control of the outputs by carefully choosing the inputs.

Several different strategies have been devised to try and do this. For instance

when the outputs of the system are not taken into account to compute its inputs we talk of an *open-loop control* strategy. This would be the only strategy available for a "blind" system (*i.e.* deprived of any sort of sensors) since information about the state would not be available. One can see how this is not really an interesting control strategy (if it is actually one to begin with) since not knowing (or ignoring) the outputs seems to be rather problematic when it comes to try and control them.

We talk of *closed-loop control* or *feedback control* when the inputs of the system are computed by using its outputs. In such a control strategy the system is typically given a *reference*, which is the desired state to be attained. A *measure* of the current state is obtained via a set of *sensors*. An *error* is then computed by taking into account both the reference and the measure. It can be seen as a metric on the ambient space. This error is given to the *controller* that is in charge of computing the best set of inputs in order to keep the error both stable and close to zero. This so called *feedback loop* is illustrated by a diagram in Figure 2.2.

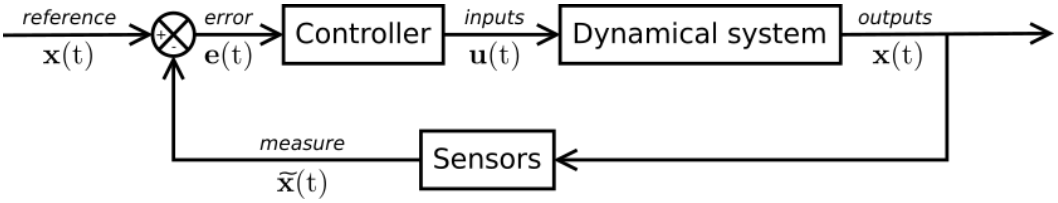


Figure 2.2: Diagram of a feedback loop in a closed-loop control strategy along with our notations in the case of a quadrotor.

If we want the quadrotor to hover at a fixed point then the reference remains constant. But for a quadrotor to follow a given trajectory the reference necessarily has to change over time. At each time step, a desired state $\overline{\mathbf{x}}(t)$ is retrieved from the desired trajectory and considered as reference. The current state $\mathbf{x}(t)$ of the system is measured by the sensors which provide a measure $\widetilde{\mathbf{x}}(t)$. From $\overline{\mathbf{x}}(t)$ and $\widetilde{\mathbf{x}}(t)$ the error $\mathbf{e}(t)$ is computed and fed to the controller which in turn computes the control $\mathbf{u}(t)$ that is given to the system.

Some components in particular must be carefully chosen or designed in any control strategy. First, it must be decided what should be considered both as inputs and outputs. We will see that for the quadrotor for instance several different sets of inputs can be considered. This will obviously influence the way the error is computed, which is also a choice that has to be well thought. But the heart of control theory is indubitably the design of the controller itself. We will present what controller we have chosen to test the validity of our trajectories in Chapter 5.

2.2.2 Control space and state space of a quadrotor

The actuators of a quadrotor are its propellers. For each one, we control the electrical tension given to the motor, therefore its angular velocity and therefore (as

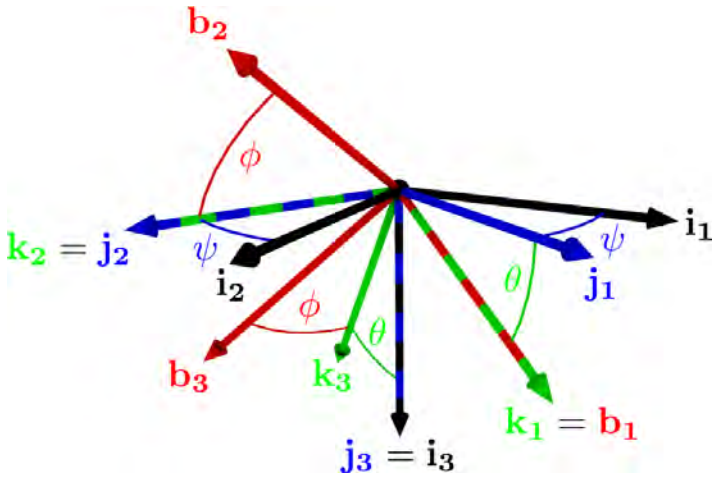


Figure 2.3: Nautical angles roll (ϕ), pitch (θ) and yaw (ψ)

explained in section 2.1) the generated thrust force. The inputs of the system could thus either be the four electrical tensions, angular velocities or thrust forces. But when considering the equations of motion, it seems more natural to use the net thrust and the total moment since they directly appear in those equations. We then have $\mathbf{u} = [f\ M_1\ M_2\ M_3]^T = [u_i]_{i=1..4}$. Note that this choice of inputs is not directly related to planning since we propose to plan for the flat outputs rather than for the controls. It is instead related to the choice of controller we made and that will be presented in Chapter 5. Those inputs are indeed the set of values that are the outputs of this controller and the inputs given to the system.

As for the state, we can directly consider the couple (position of the center of mass, rotation matrix) and write $\mathbf{x} = [\mathbf{r}\ R\ \dot{\mathbf{r}}\ \dot{R}]$. We can also locally parametrize the rotational matrix R using Euler angles roll, pitch and yaw noted ϕ, θ and ψ respectively (alternatively and more accurately also known as the Tait-Bryan angles, the Cardan angles or the nautical angles) as illustrated in Figure 2.3:

$$R = \begin{bmatrix} c_\psi c_\theta & -c_\theta s_\psi & s_\theta \\ c_\phi s_\psi + c_\psi s_\phi s_\theta & c_\phi c_\psi - s_\phi s_\psi s_\theta & -c_\theta s_\phi \\ s_\phi s_\psi - c_\phi c_\psi s_\theta & c_\psi s_\phi + c_\phi s_\psi s_\theta & c_\phi c_\theta \end{bmatrix} \text{ with } \begin{cases} c_x = \cos(x) \\ s_x = \sin(x) \end{cases}$$

In this case we have $\mathbf{x} = [x\ y\ z\ \phi\ \theta\ \psi\ \dot{x}\ \dot{y}\ \dot{z}\ \Omega_x\ \Omega_y\ \Omega_z]^T = [x_i]_{i=1..12}$. Note however that such a parametrization yields to singularities and thus using the rotation matrix is generally a better choice in practice. It is only introduced here because it will be used in the demonstration of the differential flatness of the system in section 2.4.

2.3 Physical constraints

In section 1.3 we stated that a system can be submitted to physical constraints arising from its dynamics. In this section we present what are those constraints for a quadrotor system.

2.3.1 In the space of the thrust forces amplitudes

In section 2.1 we defined the amplitudes f_i of the thrust forces $\mathbf{f_i}$ applied at points P_i. These positive quantities are linearly proportional to the square of the angular velocities ω_i of the propellers. Since that for each propeller, angular velocity is limited to a maximum value ω_{max} (that depends on the system), the thrust amplitudes are submitted to the following inequality constraints: $\forall i,\ 0 \leq f_i \leq f_{max}$ where $f_{max} = c_{f\omega}\omega_{max}^2$ is the maximum amount of thrust available per propeller. In other words the region of admissible thrusts is the hypercube $[0\ f_{max}]^4$.

2.3.2 In the control space

For simplification we note from now on $c = c_{\tau f}$. As a first approximation we could simply use the expressions of the input variables u_i written as functions of the amplitudes f_i of the thrust forces and the fact that $\forall i,\ 0 \leq f_i \leq f_{max}$ to establish that:

$$\begin{cases} 0 \leq u_1 \leq 4f_{max} \\ -df_{max} \leq u_2 \leq df_{max} \\ -df_{max} \leq u_3 \leq df_{max} \\ -2cf_{max} \leq u_4 \leq 2cf_{max} \end{cases}$$

This gives us an upper bound of the admissible control region but since the input variables are correlated we will see that it is actually not precise enough. Let us express the forces amplitudes as a function of the command.

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \Gamma^{-1}\mathbf{u} \iff 4cd \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} cdu_1 + 2cu_3 + du_4 \\ cdu_1 - 2cu_3 + du_4 \\ cdu_1 + 2cu_2 - du_4 \\ cdu_1 - 2cu_2 - du_4 \end{bmatrix}$$

Since $\forall i,\ 0 \leq f_i \leq f_{max}$, these four equalities become eight inequalities. We divide those into two groups of four noted A for the inequalities of the type $0 \leq f_i$

and B for the inequalities of the type $f_i \leq f_{max}$:

$$A: \begin{cases} 0 \leq cdu_1 + 2cu_3 + du_4 \\ 0 \leq cdu_1 - 2cu_3 + du_4 \\ 0 \leq cdu_1 + 2cu_2 - du_4 \\ 0 \leq cdu_1 - 2cu_2 - du_4 \end{cases} \quad \text{and} \quad B: \begin{cases} 0 \leq 4cdf_{max} - cdu_1 - 2cu_3 - du_4 \\ 0 \leq 4cdf_{max} - cdu_1 + 2cu_3 - du_4 \\ 0 \leq 4cdf_{max} - cdu_1 - 2cu_2 + du_4 \\ 0 \leq 4cdf_{max} - cdu_1 + 2cu_2 + du_4 \end{cases}$$

Each inequality defines an hyper-plan in the control space that divides it in half. The region of admissible controls is the intersection of those eight half spaces.

The intersection of half spaces defined by hyper-plans in four dimensions is hardly an easy thing to visualize. Let us drop one dimension by studying what happens for a fixed value of u_1. In other words we want to determine the region of admissible moments M for a given value of the net thrust f and we note this region $\mathcal{M}(f)$. We have already established that:

$$\mathcal{M}(f) \subset [-df_{max} \;\; df_{max}]^2 \times [-2cf_{max} \;\; 2cf_{max}] = \mathcal{B} \subset \mathbb{R}^3$$

Let us go a little further. The group of inequalities noted A defines four half spaces which intersection is a tetrahedron that we note $\mathcal{T}_A(f)$ and which vertices are:

$$\mathcal{T}_A(f) = \begin{bmatrix} V_{A1}(f) \\ V_{A2}(f) \\ V_{A3}(f) \\ V_{A4}(f) \end{bmatrix} = \begin{bmatrix} -df & 0 & -cf \\ df & 0 & -cf \\ 0 & -df & cf \\ 0 & df & cf \end{bmatrix}$$

Similarly the group of inequalities noted B defines four half spaces which intersection is a tetrahedron that we note $\mathcal{T}_B(f)$ and which vertices are:

$$\mathcal{T}_B(f) = \begin{bmatrix} V_{B1}(f) \\ V_{B2}(f) \\ V_{B3}(f) \\ V_{B4}(f) \end{bmatrix} = \begin{bmatrix} -d(f - 4fmax) & 0 & -c(f - 4fmax) \\ d(f - 4fmax) & 0 & -c(f - 4fmax) \\ 0 & -d(f - 4fmax) & c(f - 4fmax) \\ 0 & d(f - 4fmax) & c(f - 4fmax) \end{bmatrix}$$

In section 5.2.1 we present a quadrotor system for which some of the key values are:

$$f_{max} = 4.70 \text{ N}$$
$$d = 2.50 \times 10^{-1} \text{ m}$$
$$c = 1.54 \times 10^{-2} \text{ m}$$

Using those values we represent in Figure 2.4 both the tetrahedrons $\mathcal{T}_A(f)$ and $\mathcal{T}_B(f)$ for $f \in \left\{ \frac{f_{max}}{2}, \frac{3f_{max}}{2}, 2f_{max}, \frac{5f_{max}}{2}, \frac{7f_{max}}{2} \right\}$, alongside with the rectangular cuboid $\mathcal{B}$.

We thus have $\forall f \in [0 \; 4f_{max}],\ \mathcal{M}(f) = \mathcal{T}_A(f) \cap \mathcal{T}_B(f)$.
Note that for $f = 0$ and $f = 4f_{max}$, $\mathcal{M}(f) = \{[0\ 0\ 0]\}$.

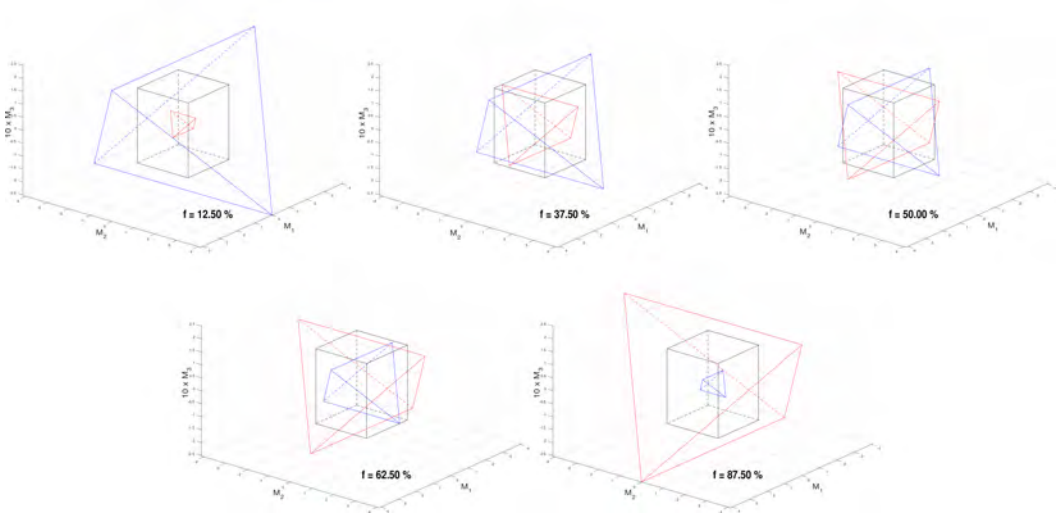


Figure 2.4: In red the tetrahedron $\mathcal{T}_A(f)$. In blue the tetrahedron $\mathcal{T}_B(f)$. In black the rectangular cuboid $\mathcal{B}$. The different values of the net thrust f are expressed as a percentage of the maximum net thrust $4f_{max}$. The M_3 axis has been scaled ten times.

Also note that for $f \leq f_{max}$, $\mathcal{T}_A(f) \subset \mathcal{T}_B(f)$ which implies $\mathcal{M}(f) = \mathcal{T}_A(f)$.
Similarly for $3f_{max} \leq f$, $\mathcal{T}_B(f) \subset \mathcal{T}_A(f)$ which implies $\mathcal{M}(f) = \mathcal{T}_B(f)$.
For $f_{max} < f < 3f_{max}$, $\mathcal{M}(f)$ is a truncated tetrahedron.
One exception though: $\mathcal{M}(2f_{max})$ is an octahedron.
All those cases are illustrated in Figure 2.5.

2.4 Differential flatness

Differential flatness was originally introduced by [Fliess 1992]. One possible definition of a flat system is that a set of outputs of the same size as the set of inputs can be found such that both the state and the inputs can be expressed as a function of these outputs and a finite number of their derivatives. More precisely let us consider the system $\mathcal{S}$ described by:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \quad \mathbf{x} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m$$

where $\mathbf{x}$ is the state and $\mathbf{u}$ the inputs. Then $\mathcal{S}$ is differentially flat if we can find $\mathbf{z} \in \mathbb{R}^m$ of the form

$$\mathbf{z} = \mathbf{z}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \ldots, \mathbf{u}^{(k)})$$

such that

$$\begin{cases} \mathbf{x} = \mathbf{x}(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(k)}) \\ \mathbf{u} = \mathbf{u}(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(k)}) \end{cases}$$

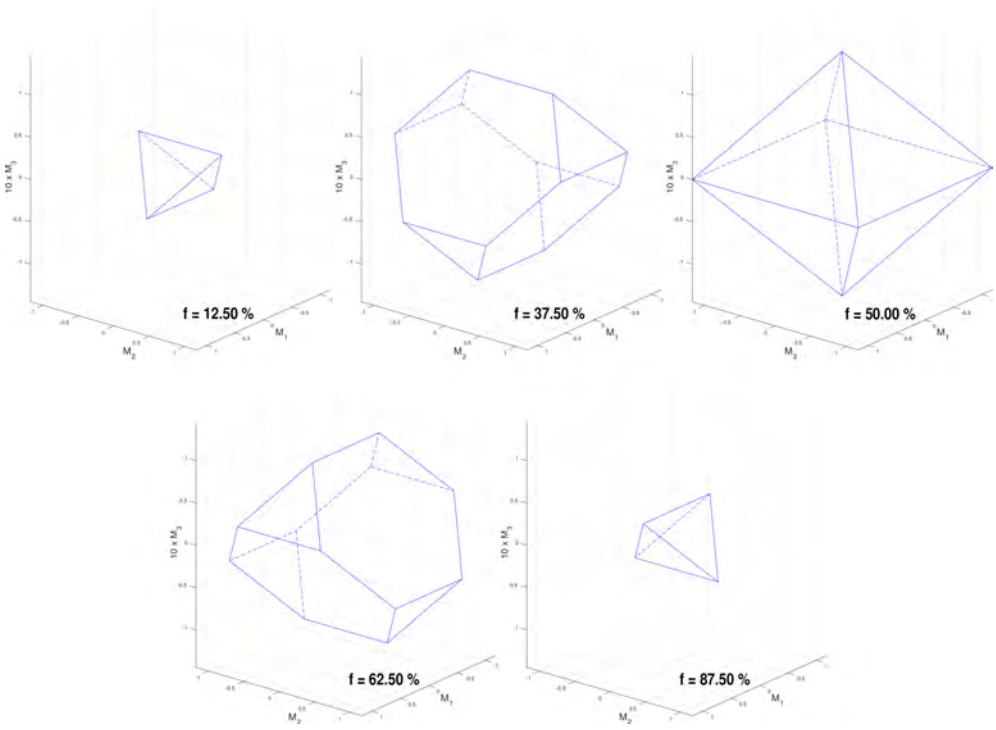


Figure 2.5: In blue the region of admissible moments $\mathcal{M}(f)$ for several representative values of the net thrust f expressed as a percentage of the maximum net thrust $4f_{max}$. The M_3 axis has been scaled ten times.

We call $\mathbf{z}$ the *flat outputs* of the system.

We would like here to recall the definition of the *hat* operator $\hat{\cdot} : \mathbb{R}^3 \to SO(3)$ such that $\hat{x}y = x \times y$ for all $x, y \in \mathbb{R}^3$. Its inverse operator is called the *vee* operator $.^\vee : SO(3) \to \mathbb{R}^3$. In motion kinematics, it is well-known that $\dot{R} = R\hat{\mathbf{\Omega}}$ (see for example [Hamano 2013]). This means that angular velocity is directly related to Euler angles and their derivatives. Note that this shows that the equations of motion of the quadrotor given in section 2.1 are indeed written as $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$.

We will show here that the quadrotor system is differentially flat for the outputs $\mathbf{z} = [x\ y\ z\ \psi]^T = [z_i]_{i=1..4}$. A slightly different version of the following proof can be found in [Mellinger 2011]. Let us first show that the state is a function of $\mathbf{z}$ and its derivatives. Obviously, position and linear velocity of the center of mass are functions of the outputs and their derivatives. Let us show that it is also the case for the rotational matrix R. We can first see that by definition $\mathbf{b_3} = R\mathbf{e_3}$. Hence, from the first equation of motion (2.1), we have:

$$\mathbf{b_3} = \frac{m}{f}\mathbf{t}, \quad \text{with} \quad \mathbf{t} = \begin{bmatrix} -\ddot{x} \\ -\ddot{y} \\ g - \ddot{z} \end{bmatrix} = \begin{bmatrix} -\ddot{z}_1 \\ -\ddot{z}_2 \\ g - \ddot{z}_3 \end{bmatrix}$$

Then $\|\mathbf{b_3}\| = \frac{m}{f}\|\mathbf{t}\| = 1$ which implies that $f = u_1 = m\|\mathbf{t}\|$. We have here expressed the first input variable as a function of the second derivatives of the first three flat outputs. As a consequence we also see that:

$$\mathbf{b_3} = \frac{\mathbf{t}}{\|\mathbf{t}\|}$$

Let us then consider an intermediate orthonormal frame $\mathcal{J} = \{\mathbf{j_1}, \mathbf{j_2}, \mathbf{j_3} = \mathbf{i_3}\}$ rotated from $\mathcal{I}$ of an angle $\psi = z_4$ about the $\mathbf{i_3}$ axis (see Figure 2.3). We have $\mathbf{j_2} = [-\sin z_4, \cos z_4, 0]^T$. Then $\mathbf{b_1}$ and $\mathbf{b_2}$ can be expressed as follows:

$$\mathbf{b_1} = \frac{\mathbf{j_2} \times \mathbf{b_3}}{\|\mathbf{j_2} \times \mathbf{b_3}\|} \text{ and } \mathbf{b_2} = \mathbf{b_3} \times \mathbf{b_1}$$

The rotational matrix $R = [\mathbf{b_1}\ \mathbf{b_2}\ \mathbf{b_3}]$ can thus be expressed almost everywhere (singularities when $\mathbf{j_2}$ and $\mathbf{b_3}$ are collinear) as a function of $\mathbf{z}$ and its derivatives. We recall that $\dot{R} = R\hat{\mathbf{\Omega}}$ thus $\mathbf{\Omega} = (R^T\dot{R})^\vee$ and so angular velocity is also a function of $\mathbf{z}$ and its derivatives. Therefore, from the second equation of motion (2.2), we see that this is also the case for momentum. Hence the three last input variables are also a function of the flat outputs and their derivatives. □

As we have just seen, the inputs can be expressed as a function of the flat outputs and their derivatives. These expressions are called the flat transformations. Note that although we took a shortcut in the demonstration above by not explicitly expressing momentum as a function of a flat outputs, it is of course entirely possible. See [Mellinger 2011] for instance for an expression of angular velocity. Differential flatness might only seems to be a theoretical property of a dynamical system but in practice it allows to move the trajectory generation problem from the state space to the space of the flat outputs. A smooth enough trajectory in the space of the flat outputs can indeed be translated as a trajectory in the control space using the flat transformations. To see why this is interesting in our case let us consider the second equation of motion (2.2). From it we can see that if we wanted to plan a trajectory in the state space we could not plan for each angular velocity component independently because of the coupled dynamics introduced by the vectorial product. As we will see in the next chapter this is not the case in the space of the flat outputs: we can plan each independently. Planning is therefore made easier.

In section 2.3.2 we expressed the set of admissible inputs. We want to plan a trajectory in the space of the flat outputs that, when translated into the control space, remains inside this admissible set. Or in other worlds, we want to plan an admissible trajectory. This proved to always be possible provided that the derivatives of the flat outputs are small enough. This comes from the fact that the flat transformations are smooth and that the input vector $\mathbf{u}$ tends to $[mg\ 0\ 0\ 0]^T$ (which is an admissible control) as the derivatives of the flat outputs tends to zero in norm. See [Spica 2012a] for instance for a more detailed explanation. As for actually expressing the limits on the derivatives in function of, for example,

the maximum thrust f_{max}, it remains to our knowledge an open problem. In the remaining of this thesis we will however use those limits and give them numerical values in various experiments. Please keep in mind that since the link between the physical limits of the system and the bounds on the derivatives of its flat outputs is not known to us, those values are bound to be empirical.

To summarize, we have to plan a smooth enough trajectory in the space of the flat outputs with bounds on the derivatives. The next chapter will be addressing this problem exactly.

Chapter 3

Steering method for a quadrotor

Contents

We saw in section 2.4 that thanks to the differentially flatness of the system we can plan for a trajectory in the space of the flat outputs and their derivatives rather than in the state space. We saw that the flat outputs are $\mathbf{z} = [\mathbf{r}^T, \psi]^T$ and that the steering method should plan for $\mathbf{x} = [\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}]^T$, that we abusively call the state of the quadrotor. In this chapter we focus on the steering method (or local planner), *i.e.* the way a local path is produced between two states. We first present the corresponding two-point boundary value problem. A method to solve it, that can be used as a steering method for a quadrotor, is then proposed. It is a near time-optimal spline-based approach for which we will discuss the optimality of the computed solutions. We will finally see how this method can be applied to more general systems.

3.1 A two-point boundary value problem

In this section we first define the general two-point boundary value problem in four dimensions and then see how it can be redefined as a collection of problems in one dimension.

3.1.1 Definition of the full problem

We recall that the flat outputs of a quadrotor are

$$\mathbf{z} = \begin{bmatrix} x \\ y \\ z \\ \psi \end{bmatrix} = [z_i]_{i=1..4} \in \mathcal{P} \subset \mathbb{R}^4$$

with $\mathcal{P}$ the space of the flat outputs. We can consider its state to be

$$\mathbf{x} = \begin{bmatrix} \mathbf{z} \\ \dot{\mathbf{z}} \\ \ddot{\mathbf{z}} \end{bmatrix} = [x_i]_{i=1..12} \in \mathcal{X} \subset \mathbb{R}^{12}$$

From now on we will abusively call $\mathcal{X}$ the state space.

The steering method has to produce a trajectory S in $\mathcal{P}$. Let us note $T \in \mathbb{R}^+$ its (unknown) total duration. Because the trajectory has to be continuous in $\mathcal{P}$, and in order to guaranty continuity of the angular velocity, S has to be of class $\mathcal{C}^3$ at least: $S \in \mathcal{C}^3([0\ T], \mathcal{P})$. This is due to the relationship between attitude and linear acceleration imposed by the equations of motion. In addition, since these (local) trajectories will be the elements of the solution provided by the (global) motion planner, we want to ensure continuity of the jerk (first derivative of acceleration) between two consecutive trajectories. The jerk is not part of the state as we choose to define it and is therefore not sampled. We thus have to impose an arbitrary value for it at the end-points. Zero seems to be the easiest and most natural choice. Thus, another constraint has to be imposed:

$$\dddot{S}(0) = \dddot{S}(T) = 0$$

The physical limitations of the system are imposing bounds on acceleration and jerk. For security reasons we will also consider bounds on velocity. Furthermore, since we want a continuous and bounded jerk, snap (second derivative of acceleration) has to be bounded too. We are also concerned by optimality: we want the minimal time solutions. This is important if we want to be able to use this method as local planner in algorithms such as RRT* or PRM*. Thus for a given couple of states $(\mathbf{x}_0, \mathbf{x}_F)$ the steering method has to provide a solution to the problem:

$$\begin{cases} \min T \in \mathbb{R}^+ \text{ s.t.} \\ S \in \mathcal{C}^3([0\ T], \mathcal{P}) \\ \\ \left[S(0)\ \dot{S}(0)\ \ddot{S}(0)\right] = \mathbf{x}_0 \in \mathcal{X} \\ \\ \left[S(T)\ \dot{S}(T)\ \ddot{S}(T)\right] = \mathbf{x}_F \in \mathcal{X} \\ \\ \dddot{S}(0) = \dddot{S}(T) = 0 \\ \forall t \in [0, T] \begin{cases} \dot{S}(t) \in \mathcal{V} \\ \ddot{S}(t) \in \mathcal{A} \\ \dddot{S}(t) \in \mathcal{J} \\ \ddddot{S}(t) \in \mathcal{S} \end{cases} \end{cases} \tag{3.1}$$

where $\mathcal{V}$, $\mathcal{A}$, $\mathcal{J}$ and $\mathcal{S}$ are zero centred intervals of $\mathbb{R}^4$. Note that $\mathcal{X} = \mathcal{P} \times \mathcal{V} \times \mathcal{A}$.

3.1.2 Independence of the outputs

In the full problem we can see that the outputs are two-by-two independent. This means that, with the exception of the total duration T of the trajectory, they have no parameters in common. We therefore can solve the problem independently for each output. Any given state $\mathbf{x}_k \in \mathcal{X}$ can be written as:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{z}_k \\ \dot{\mathbf{z}}_k \\ \ddot{\mathbf{z}}_k \end{bmatrix} = \begin{bmatrix} \left[z_i^k\right]_{i=1..4} \\ \\ \left[\dot{z}_i^k\right]_{i=1..4} \\ \\ \left[\ddot{z}_i^k\right]_{i=1..4} \end{bmatrix}$$

We can also write:

$$\begin{cases} \mathcal{V} = \prod_{i=1}^{4} [-v_i, v_i] \\ \mathcal{A} = \prod_{i=1}^{4} [-a_i, a_i] \\ \mathcal{J} = \prod_{i=1}^{4} [-j_i, j_i] \\ \mathcal{S} = \prod_{i=1}^{4} [-s_i, s_i] \end{cases}$$

with $(v_i\ a_i\ j_i\ s_i) \in \mathbb{R}_+^4$. Given a couple of states $(\mathbf{x}_0, \mathbf{x}_F)$, solving the problem independently for each output is then finding solutions S_i to the problem

$$\begin{cases} \min T \in \mathbb{R}^+ \text{ s.t.} \\ \\ \forall i = 1..4, \begin{cases} S_i \in \mathcal{C}^3([0\ T], \mathbb{R}) \\ \\ \left[S_i(0)\ \dot{S}_i(0)\ \ddot{S}_i(0)\ \dddot{S}_i(0)\right] = [z_i^0\ \dot{z}_i^0\ \ddot{z}_i^0\ 0] \\ \\ \left[S_i(T)\ \dot{S}_i(T)\ \ddot{S}_i(T)\ \dddot{S}_i(T)\right] = \left[z_i^T\ \dot{z}_i^T\ \ddot{z}_i^T\ 0\right] \\ \\ \forall t \in [0,T] \begin{cases} |\dot{S}_i(t)| \leq v_i \\ |\ddot{S}_i(t)| \leq a_i \\ |\dddot{S}_i(t)| \leq j_i \\ |\ddddot{S}_i(t)| \leq s_i \end{cases} \end{cases} \end{cases}$$

Note that in this formulation the total time of motion T is the same for each sub-problem. We propose to relax this constraint by allowing us to find a different T_i for each problem. This will simplify their resolution and therefore speed-up the overall process as we will illustrate in section 3.3.3. One issue though is that the solutions will not to be synchronized together. We will see later how to solve this. In order to simplify the notations we will now write for a given output of index i:

$$(v_i\ a_i\ j_i\ s_i) = (v_{max}\ a_{max}\ j_{max}\ s_{max})$$

$$[z_i^0\ \dot{z}_i^0\ \ddot{z}_i^0] = [x_0\ v_0\ a_0]$$

$$\left[z_i^T\ \dot{z}_i^T\ \ddot{z}_i^T\right] = [x_F\ v_F\ a_F]$$

Thus the problem in one dimension that has to be solved independently for each output can finally be written as:

$$\begin{cases} \min T_i \in \mathbb{R}^+ \text{ s.t.} \\ \\ S_i \in \mathcal{C}^3([0\ T_i], \mathbb{R}) \\ \\ \left[S_i(0)\ \dot{S}_i(0)\ \ddot{S}_i(0)\ \dddot{S}_i(0)\right] = [x_0\ v_0\ a_0\ 0] \\ \\ \left[S_i(T_i)\ \dot{S}_i(T_i)\ \ddot{S}_i(T_i)\ \dddot{S}_i(T_i)\right] = [x_F\ v_F\ a_F\ 0] \\ \\ \forall t \in [0,T_i] \begin{cases} |\dot{S}_i(t)| \leq v_{max} \\ |\ddot{S}_i(t)| \leq a_{max} \\ |\dddot{S}_i(t)| \leq j_{max} \\ |\ddddot{S}_i(t)| \leq s_{max} \end{cases} \end{cases} \tag{3.2}$$

3.2 A near time-optimal spline-based approach

In this section we present a near time-optimal spline-based approach to solving the BVP. We will first present the proposed closed-form solution. Then in the following sub-sections we will further detail the resolution. We will finally discuss the optimality of the computed solutions.

3.2.1 Closed-form solution

The BVP described by equation (3.2) can be seen as a classical optimal control problem. It is indeed a fourth order integrator. In this formulation, the state of the system is $X(t) = \left[S(t)\ \dot{S}(t)\ \ddot{S}(t)\ \dddot{S}(t)\right]^T$ and the control input is $u(t) = \ddddot{S}(t) \in [-s_{max}\ s_{max}]$. We can then write:

$$\dot{X}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} X(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u(t)$$

Pontryagin maximum principle (see for example [Bertsekas 1995]) states that, without constraints on the state, the optimal control u^* for the minimum time problem is necessarily saturated meaning that $\forall t \in [0\ T_i],\ u^*(t) \in \{-s_{max}, s_{max}\}$ with at most three control commutations. Thus finding the optimal control in this case is finding these control commutations. Depending on the order of the integrator this can be done analytically and therefore within reasonable computing time. This type of command is called a *bang-bang solution*.

In our case, the state is also constrained. In this case the optimal command is a so-called *bang-singular-bang solution* where $\forall t \in [0\ T_i],\ u^*(t) \in \{-s_{max}, 0, s_{max}\}$. Finding it is not trivial and usually requires computationally expensive numerical approaches. This is not well suited for integration in sampling-based motion planners since they make extensive use of the steering method (usually thousands of calls to solve constrained planning problems). Furthermore, solving independently the BVP in one dimension for each output will provide four different durations T_i. In order to produce a coherent trajectory these solutions have to be "synchronized" meaning that their durations have to matched to a single common duration. It is not trivial to see how to do that. We therefore propose a method to compute a trajectory with an imposed shape inspired by the bang-singular-bang solution that approaches the optimal. This will allow us to both quickly compute good solutions to the BVPs in one dimension and facilitate their synchronization.

The mathematical justification of the shape of the optimal bang-singular-bang solution is quite complicated. We propose here a more intuitive explanation. The motion is divided into several main phases illustrated with an example in Figure

3.1.

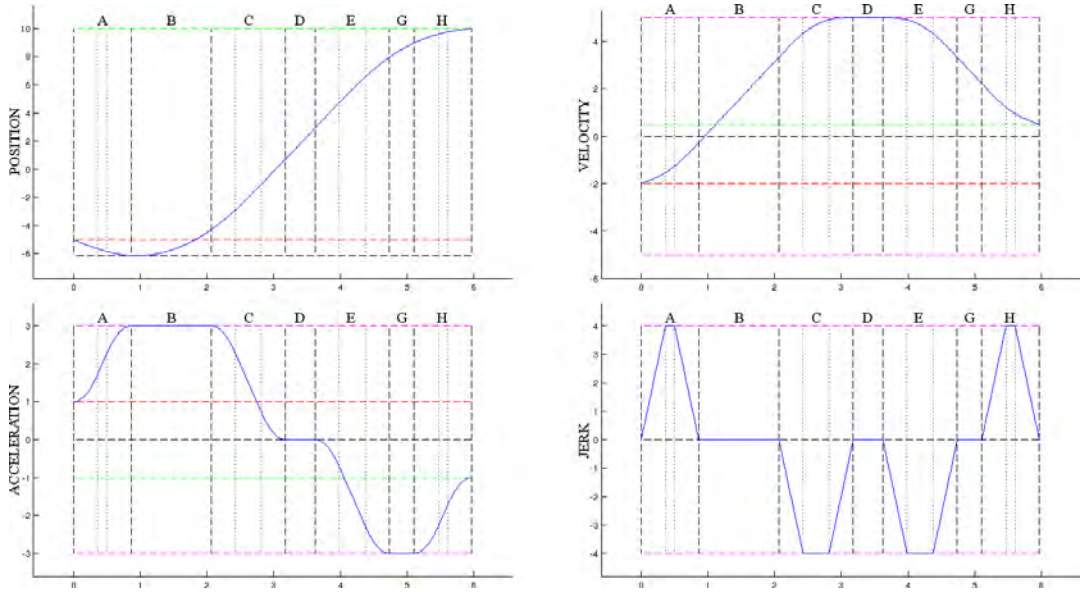


Figure 3.1: Example of trajectory provided by the steering method for one flat output. The bounds of each derivative are represented by pink dashed lines. Red and green dashed lines represent initial and final values, respectively. Note that, in order to show all the phases, $|v_D| = v_{max}$ and $|a_B| = |a_D| = a_{max}$ in this example. In a general case, some phases can have zero duration.

In order to minimize T_i, maximum velocity has to be reached as soon as possible and maintained as long as possible. Let the letter D refer to this phase of constant velocity and let $v_D \in [-v_{max}\ v_{max}]$ be the value of velocity during this phase. Time spent to reach this maximum velocity is also to be minimized meaning that the durations of the phases of velocity transitions from v_0 to v_D and from v_D to v_F have to be as short as possible. Hence time spent at maximum acceleration during those two phases has to be maximized. This implies to minimize the durations of acceleration variation phases. Let B refer to the phase of constant acceleration during the first phase of velocity variation and $a_B \in [-a_{max}\ a_{max}]$ be the value of this constant acceleration. Let G be the phase of constant acceleration during the second phase of velocity variation and $a_G \in [-a_{max}\ a_{max}]$ be the value of this constant acceleration. The phase in which acceleration varies from a_0 to a_B is noted A, and C is used to refer to the phase in which acceleration varies from a_B to zero. The phases where acceleration varies from zero to a_G and from a_G to a_F are noted E and H respectively.

This principle holds for higher derivatives. Time spent at maximum jerk during acceleration variations has to be maximized and durations of jerk variation phases have to be minimized. This implies to maximize time spent at maximum snap during a jerk variation phase and to minimize the durations of snap variation phases. This very last part is easy since, following our approach, snap can be discontinuous:

a snap variation phase is actually a snap commutation of duration zero ($\forall t \in [0, T_i]$, $\dddot{S}_i(t) \in \{-s_{max}, 0, s_{max}\}$). In other words, $\dddot{S}_i$ is a piecewise constant function, which implies that S_i is a piecewise polynomial function (a spline) of the fourth order.

In the discussion above, we have mentioned seven main phases in the (local) trajectory. For three of them (B, D, G), acceleration is zero and hence snap is also zero. The four others (A, C, E, H) are divided into three sub-phases $(1, 2, 3)$. Phases 1 and 3 correspond to jerk variations and hence for which $|\dddot{S}_i(t)| = s_{max}$. In phase 2, jerk is constant hence snap is zero. The sign of the snap is opposed during phases 1 and 3, which have the same duration. Table 3.1 presents the notation used for the durations of the phases and the expressions of the snap as functions of a_B and a_G.

Table 3.1: Value of the snap and duration of each phase of the spline

Phase	Value of the Snap	Duration
A_1	$\text{sign}(a_B - a_0).s_{max}$	$t_{A,1}$
A_2	0	$t_{A,2}$
A_3	$\text{sign}(a_0 - a_B).s_{max}$	$t_{A,1}$
B	0	t_B
C_1	-$\text{sign}(a_B).s_{max}$	$t_{C,1}$
C_2	0	$t_{C,2}$
C_3	$\text{sign}(a_B).s_{max}$	$t_{C,1}$
D	0	t_D
E_1	$\text{sign}(a_G).s_{max}$	$t_{E,1}$
E_2	0	$t_{E,2}$
E_3	-$\text{sign}(a_G).s_{max}$	$t_{E,1}$
G	0	t_G
H_1	$\text{sign}(a_F - a_G).s_{max}$	$t_{H,1}$
H_2	0	$t_{H,2}$
H_3	$\text{sign}(a_G - a_F).s_{max}$	$t_{H,1}$

The expression of the snap during phase A_1 is explained as follows: if $a_B > a_0$, $\ddot{S}_i$ has to be increasing, which implies that $\dddot{S}_i$ has to be positive. Since $\dddot{S}_i(0) = 0$, $\dddot{S}_i$ has to be increasing during phase A_1, and thus $\dddot{S}_i(t) = +s_{max}$. Following the same reasoning, $\dddot{S}_i(t) = -s_{max}$ if $a_B < a_0$. If $a_B = a_0$, then phase A is not needed, and thus $\dddot{S}_i(t) = 0$. A similar reasoning can be applied to understand the expressions for phases C, E and H.

Note the simplifying choices we have made. We impose both jerk and acceleration to go through zero during phase B, which implies that velocity reaches its maximum value v_D very smoothly. This is indeed the case for the optimal trajectory if velocity is saturating. In fact each time a component of the state is saturating,

the optimal trajectory goes through a singular arc, meaning that the command is zero (here the snap). But for trajectories for which there is no velocity saturation, going from a_B to a_G can be done more efficiently. We will see that this choice allows a simpler computation of the trajectory and is also useful during the synchronization process. However this necessarily leads to the computation of a sub-optimal solution. We will quantify this sub-optimality in section 3.3. From now on, both *optimal time* and *optimal velocity* are to be read as: *with respect to our closed-form solution.*

3.2.2 Duration of the phases

At this point, the local trajectory for one output (*i.e* its corresponding spline) is parametrized by a_B, a_G and the durations of all phases. This subsection explains how it can be parametrized using a single parameter: v_D. First, $t_{A,1}$, $t_{A,2}$, $t_{C,1}$ and $t_{C,2}$ can be expressed as functions of a_B, a_0, j_{max} and s_{max}. The same goes for $t_{H,1}$, $t_{H,2}$, $t_{E,1}$ and $t_{E,2}$ as functions of a_G, a_F, j_{max} and s_{max}. We provide next explanations only for $t_{A,1}$ and $t_{A,2}$ but the same principles are applied to all durations mentioned above. More details are provided in Appendix A. Let us define δ_{B0} =sign$(a_B - a_0)$. From Table 3.1 and equation (3.2), we can write:

$$a_B = \delta_{B0}.s_{max}.t_{A,1}^2 + \delta_{B0}.s_{max}.t_{A,1}.t_{A,2} + a_0 \tag{3.3}$$

and $\dddot{S}_i(t_{A,1}) = \delta_{B0}.s_{max}.t_{A,1}$, which implies $|\dddot{S}_i(t_{A,1})| = s_{max}.t_{A,1} \leq j_{max}$, and thus:

$$s_{max}.t_{A,1}^2 \leq \frac{j_{max}^2}{s_{max}} = a_{lim} \tag{3.4}$$

Phase A_2 is only needed when phases A_1 and A_3 are not enough to reach a_B. If there is no phase A_2, then $t_{A,2} = 0$. In this case, using equations (3.3) and (3.4), we can write:

$$|a_B - a_0| = s_{max}.t_{A,1}^2 \leq a_{lim}$$

If $|a_B - a_0| > a_{lim}$, then $t_{A,2} \neq 0$ (*i.e.* phase A_2 is needed) and $t_{A,1} = \frac{j_{max}}{s_{max}}$, its maximum value. In that case,

$$t_{A,2} = \frac{|a_B - a_0|}{j_{max}} - \frac{j_{max}}{s_{max}}$$

If $t_{A,2} = 0$, then

$$t_{A,1} = \sqrt{\frac{|a_B - a_0|}{s_{max}}}$$

The principle is the same for phases C, E and H. For phase C, $|a_B - a_0|$ is replaced by $|a_B|$, by $|a_G|$ in phase E and by $|a_G - a_F|$ in phase H. Complete expressions are provided in Appendix A. The spline is now parametrized by $(a_B, a_G, t_B, t_D, t_G)$.

Both couples (a_B, t_B) and (a_G, t_G) can actually be expressed as a function of v_D. Using Table 3.1 and equation (3.2), v_D can be expressed as a function of $(a_B, t_{A,1}, t_{A,2}, t_B, t_{C,1}, t_{C,2})$ and hence as a function of (a_B, t_B). Let us note $V_B : (a_B, t_B) \mapsto v_D$ this function. Let us also define $v_B^{min} = V_B(-a_{max}, 0)$ and $v_B^{max} = V_B(a_{max}, 0)$. For all v_D in $[v_B^{min}, v_B^{max}]$, there is a unique $a_B \in ([-a_{max}, \min(0, a_0)] \cup [\max(0, a_0), a_{max}])$ such that $V_B(a_B, 0) = v_D$ (see Appendix A for more details). If $v_D < v_B^{min}$, then $a_B = -a_{max}$ and phase B is needed. Its duration is simply $t_B = |v_D - v_B^{min}|/a_{max}$. If $v_D > v_B^{min}$, then $a_B = a_{max}$ and $t_B = |v_D - v_B^{max}|/a_{max}$. Hence, as previously announced (a_B, t_B) can be expressed as a function of v_D. The same goes for (a_G, t_G).

At this point the spline is parametrized by (v_D, t_D). Let t_C be the value of the time parameter at the end of phase C, and t_E its value at the beginning of phase E. As explained previously, these two values can be expressed as a function of v_D. This is also the case for $S_i(t_C)$ and $S_i(t_E)$. Let Δ_S be the function $v_D \mapsto S_i(t_E) - S_i(t_C)$. If $v_D \neq 0$, then $t_D = \Delta_S(v_D)/v_D$. Therefore, the spline is completely parametrized by the sole value of v_D. Note that, necessarily:

$$v_D.\Delta_S(v_D) \leq 0 \tag{3.5}$$

3.2.3 Optimal velocity

The spline is now parametrized by v_D only. This section explains how to compute the optimal value v_{opt} that minimizes $T_i(v_D)$. Let $\mathcal{V}_{valid}$ be the set of values of v_D that meet the constraint given by equation (3.5) and let us note $\mathcal{V}_0 = [\min(0, v_{opt}), \max(0, v_{opt})]$. For synchronization purposes, it is necessary that $\mathcal{V}_0 \subset \mathcal{V}_{valid}$, which implies:

$$\forall v \in \mathcal{V}_0, v.\Delta_S(0) \leq 0 \tag{3.6}$$

Let us note $\delta_0 = \text{sign}(\Delta_S(0))$. Since minimizing T_i requires maximizing $|v_D|$, equation (3.6) implies that, if Δ_S has zeros between $v = 0$ and $v = \delta_0.v_{max}$, then v_{opt} is the one of lowest absolute value. If not, $v_{opt} = \delta_0.v_{max}$.

3.2.4 Synchronization

The trajectory generation problem is now solved for each output independently. Hence, we have four different values T_i for $i = 1..4$. This subsection explains how to synchronize together these one-dimensional trajectories. Solving each problem provides a couple $(T_i, v_{opt,i})$ and the associated interval $\mathcal{V}_{0,i}$. The purpose of the definition of v_{opt} provided in the previous subsection is to guaranty that the application $v_D \mapsto T_i(v_D)$ is continuously strictly monotonic on $\mathcal{V}_{0,i}$. Furthermore, $\lim_{v_D \to 0} T_i(v_D) = +\infty$. This implies that, for any t in $[T_i, +\infty[$, there is a unique

$v \in \mathcal{V}_{0,i}$ such that $T_i(v) = t$. This property is used to synchronize the components. The slowest component of index j is identified and and we note $T = T_j$. For each $i \neq j$ a simple dichotomous search is used to find the unique $v_D \in \mathcal{V}_{0,i}$ such that $T_i(v_D) = T$ for the three other components. As a result, all the components have the same final time T. All outputs are synchronized.

3.2.5 Generalization to other robotic systems

In the context of this work we apply this method to the quadrotor system. This means solving the BVP in three dimensions if yaw is kept constant or in four dimensions otherwise. But really, any differentially flat system or for that matter any robotic system with uncoupled dynamics can be treated using this method, whatever the dimension. We therefore implemented it for any such system as a standalone C++ library named KDTP (for KinoDynamic Trajectory Planner).

A git repository is available at: git://git.openrobots.org/robots/libkdtp.git

3.3 Discussion on optimality

In section 3.2.1 we explained the simplifying choices we have made in order to establish a closed-form solution to the BVP in one dimension that allows both an easier computation and synchronization of the outputs. We stated that our solutions are necessarily sub-optimal. In this section we will first explain why this is the case in further details and then quantify this sub-optimality. For that we will focus first on the solutions to the BVP in one dimension an then study the solutions to the problem in three dimensions.

3.3.1 Optimality and velocity saturation

In our proposed closed-form solution to the BVP in one dimension, acceleration necessarily goes smoothly to zero at one time in the trajectory, meaning that we chose to set a zero snap (and jerk) at some point during this phase. For the optimal solution, this is only the case if velocity is saturating. During a saturation phase velocity is indeed constant and therefore higher derivatives are nil. If this is the case then our method computes the optimal solution. But if there is no velocity saturation then our method will compute a sub-optimal solution because in the optimal trajectory there is no need for the acceleration to smoothly go to zero. This is illustrated on two examples in Figure 3.2.

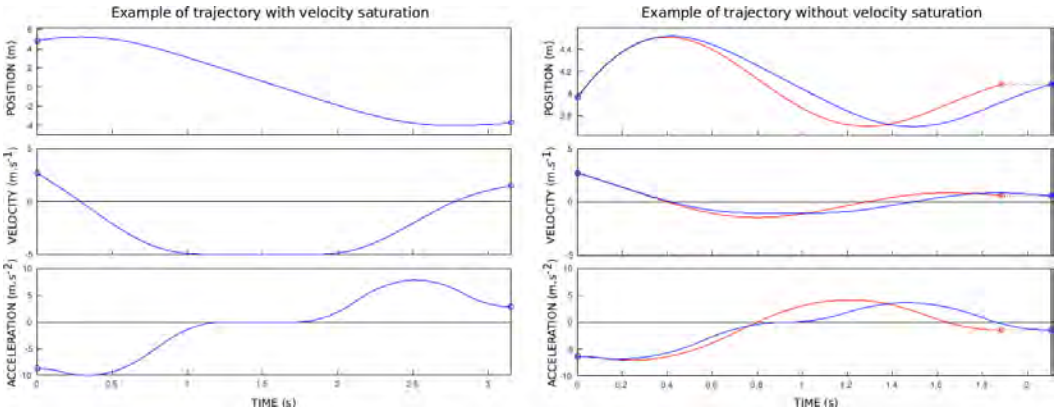


Figure 3.2: *Left*: velocity is saturating, our method therefore computes the optimal solution. *Right*: no velocity saturation. In blue, our sub-optimal solution. In red, the optimal solution. The length of the black dashed segment is the absolute error between the two solutions.

3.3.2 In one dimension

The goal of this section is to give an idea of the quality of our proposed closed-form solution to the BVP described by Equation (3.2). By that we mean comparing the time T_i computed by our method (that we here simply note T) to the actual optimal time T^*. We propose to do so by uniformly sampling a total of 10^4 couples of "states" $([x_0\ v_0\ a_0], [x_F\ v_F\ a_F])$ in $[-5\ 5]\times[-v_{max}\ v_{max}]\times[-a_{max}\ a_{max}]$, solve the problem with our method and compare the duration T of the result to the duration T^* of the optimal solution. In this experiment we use the following values:

$$\begin{aligned} v_{max} &= 5 \text{ m.s}^{-1} \\ a_{max} &= 10 \text{ m.s}^{-2} \\ j_{max} &= 20 \text{ m.s}^{-3} \\ s_{max} &= 50 \text{ m.s}^{-4} \end{aligned}$$

In order to compute the optimal time T^* we use ACADO Toolkit. It is a software environment and algorithm collection for automatic control and dynamic optimization implemented as self-contained C++ code (see http://acado.github.io). In our implementation, it treats the optimal control problem by formulating it as a multiple shooting problem that is solved using an SQP algorithm (Sequential Quadratic Programming). We use 20 nodes along the trajectory and the initial guess is a linear interpolation for each state and command variables. The KKT tolerance parameter is set to 10^{-12}.

To compare T and T^* we compute the relative error between them as follows:

$$E_{rel} = \frac{T - T^*}{\text{Max}(T, T^*)}$$

In total we actually sampled $17,251$ couples of states but disregarded $7,251$ of them (42.03%). One possible reason for it is that the BVP does not always have a solution. It is for example possible that velocity at one end is too high so that it is not possible, given the constraints on higher derivatives, to reduce it in time to avoid violating the velocity constraint. This will be explained in more details in section 4.2.2. In this case the ACADO algorithm will not converge. In our experiment this happened for $6,409$ couples of "states" (37.15%). The other reason for disregarding a sample is that we obtain $E_{rel} < -0.01$. In those cases we consider that the ACADO algorithm has converged poorly and therefore that the test is inconclusive. This happened for 842 samples (4.88%).

Among the 10^4 successful tries:

- For 22.60% of them we have $|E_{rel}| < 0.01$. This means that our method gives the same result as the ACADO algorithm within a 1% margin. We therefore consider that our method provides a trajectory with optimal duration.
- For the remaining 77.40%, we have $0.01 \leq E_{rel}$, meaning that our method provides a sub-optimal result. We propose to study the distribution of the relative error for those cases. An histogram and a box plot are represented in Figure 3.3. We also provide key statistical values in Table 3.2.

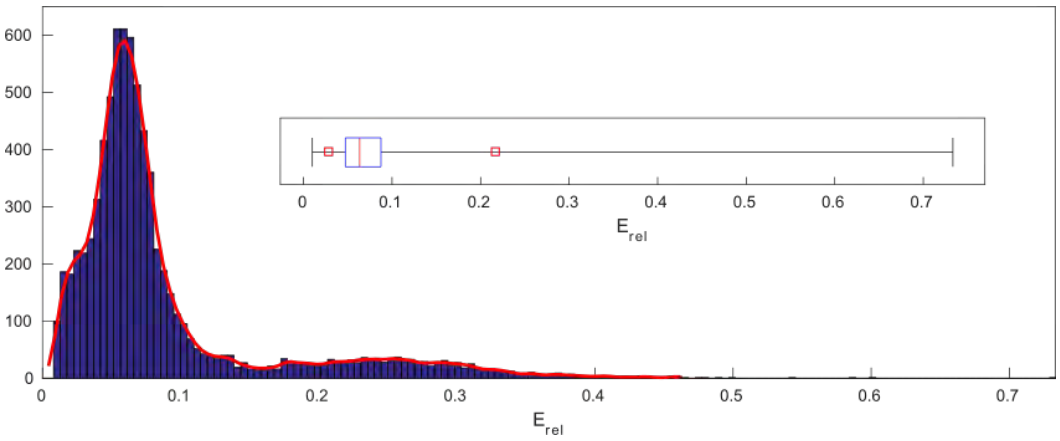


Figure 3.3: For the BVP in one dimension: the histogram of the distribution of the relative error for the $7,740$ sub-optimal cases alongside with its box plot. In the later, the ends of the whiskers are the extremal values, the red squares are the 10th and 90th percentiles, the edges of the blue box are the first and third quartiles and the red segment inside the box is the median.

In summary, among the 10^4 conclusive tests, 22.60% are optimal and among the remaining 77.40% the mean relative error is 8.84% with 90% of them having a relative error below 21.65%. In total, the mean relative error is therefore 6.85%.

At this point one can wonder why using a sub-optimal method when we do have the numerical tools to compute the optimal solution. The answer lies in the

Table 3.2: For the BVP in one dimension: key statistical values for the distribution of the relative error for the $7,740$ sub-optimal cases.

Minimum	0.01002
10th percentile	0.02889
First quartile	0.04721
Median	0.06351
Third quartile	0.08748
90th percentile	0.21653
Maximum	0.73342
Mean	0.08844
Standard deviation	0.07743

computing time needed to run those numerical methods. Lets compare the CPU times for the 10^4 conclusive tests. Those times are for a C++ implementation run on a single core of an Intel Xeon W3520 processor at 2.67GHz. For the ACADO algorithm the mean running time is 132.36 milliseconds (with a standard deviation of 32.42 milliseconds) whereas the mean running time for our method is 0.21 milliseconds (with a standard deviation of 0.50 milliseconds). Histograms of the distribution of the CPU times for both methods are represented in Figure 3.4.

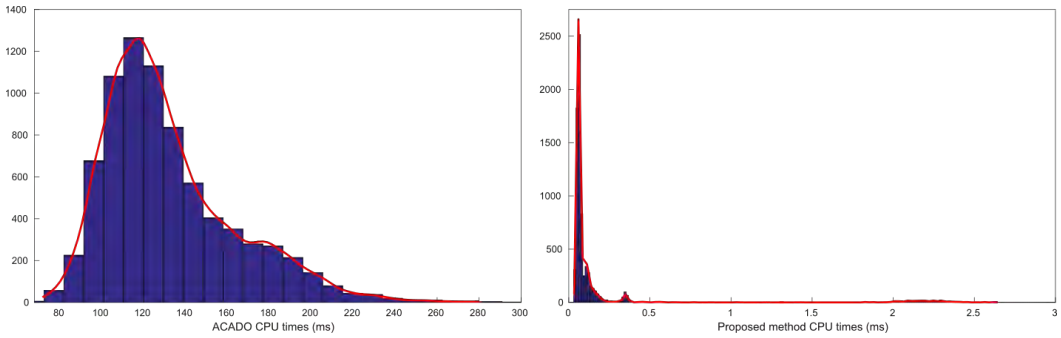


Figure 3.4: For the BVP in one dimension: the histograms of the distribution of the CPU times in milliseconds for both methods. On the left: the ACADO algorithm. On the right: the proposed method.

For being 6.85% sub-optimal in average we gain three orders of magnitude in mean running time.

3.3.3 In three dimensions

We focus here on a simplified version of the problem described by Equation (3.1). Without loss a generality we chose to keep the yaw angle constant. We therefore study the solutions of the BVP in three dimensions. The experimental setup is very similar to the one on the previous section. We sample 10^4 couples of states in $[-5\ 5]^3 \times [-v_{max}\ v_{max}]^3 \times [-a_{max}\ a_{max}]^3$, solve the problem with our method and

compare the duration T of the result to the duration T^* of the optimal solution. We use the same values as before for the bounds on the derivatives v_{max}, a_{max}, j_{max} and s_{max}. Here again we use ACADO Toolkit to compute the optimal solution.

We sampled a total of $45,474$ couples of states and disregarded $35,474$ of them (78.01%). For $3,010$ samples (6.62%) the ACADO algorithm did not converge properly. For the remaining $32,464$ samples (71.39%), it did not converge at all. The increased number of failures is explained by the dimension of the problem. It is enough that at least one component out of three of one of the two states is ill sampled (meaning that the interpolation under constraints for this component has no solution) for the all problem to have no solution.

Among the 10^4 successful tries, 41.41% have optimal duration. The mean relative error of the remaining 58.59% sub-optimal cases is 6.08% and 90% of them have a relative error below 8.84%. In total, the mean relative error is therefore 3.56%. We provide an histogram and a box plot of the distribution of the relative error for the $5,859$ sub-optimal cases in Figure 3.5 and key statistical values in Table 3.3.

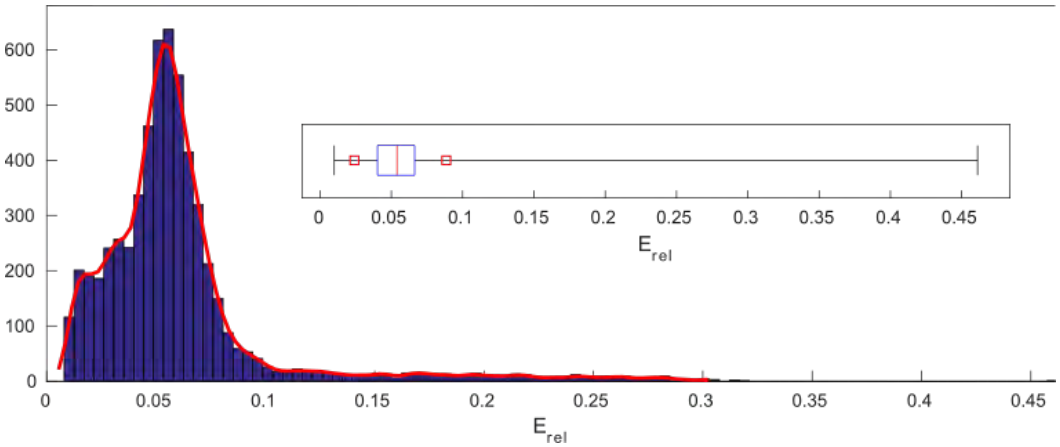


Figure 3.5: For the BVP in three dimensions: the histogram of the distribution of the relative error for the $5,859$ sub-optimal cases alongside with its box plot. In the later, the ends of the whiskers are the extremal values, the red squares are the 10th and 90th percentiles, the edges of the blue box are the first and third quartiles and the red segment inside the box is the median.

The increased accuracy of the method is also explained by the dimension of the problem. The probability of velocity saturation is increased by the fact that each sample in three dimensions is equivalent to three samples in one dimension. Hence the increased percentage of optimal solutions computed by our method. As expected the distribution of relative error for the sub-optimal cases is similar to the problem in one dimension.

Let us compare the CPU times. For the ACADO algorithm the mean running time is 601.30 milliseconds (with a standard deviation of 481.082 milliseconds) whereas the mean running time for our method is 0.61 milliseconds (with a standard

Table 3.3: For the BVP in three dimensions: key statistical values for the distribution of the relative error for the $5,859$ sub-optimal cases.

Minimum	0.01000
10th percentile	0.02365
First quartile	0.04030
Median	0.05394
Third quartile	0.06636
90th percentile	0.08838
Maximum	0.46108
Mean	0.06083
Standard deviation	0.04298

deviation of 0.80 milliseconds).

Chapter 4

Kinodynamic motion planning for a quadrotor

Contents

In this chapter we present the global approaches on which we have focused to solve the kinodynamic motion planning problem for a quadrotor. In particular we will show how the spline-based steering method that we described in the previous chapter can be integrated into those global methods. We first present a decoupled approach together with a local optimization method of the global solution trajectory. We then focus on direct approaches. In that perspective we begin by addressing the problem of the metric in the state space. Follows the description of an incremental sampling strategy in the state space. We finally present two global direct approaches to kinodynamic motion planning and discuss the influence of both the metric and the sampling strategy on both of them.

4.1 A decoupled approach

This section explains how the spline-based steering method presented in Chapter 3 can be used in a decoupled approach to kinodynamic motion planning. Our implementation of the decoupled approach consists of two stages: 1) planning a geometrically valid path in $\mathbb{R}^3$ for the center of mass of the quadrotor using its minimum bounding sphere for collision detection; 2) transforming this path into a trajectory in $\mathcal{X}$. Since we use the minimum bounding sphere for collision detection,

planning the yaw angle profile seems a bit artificial and has little interest. We therefore chose here to keep it constant. It is always possible if need be to chose a different yaw profile afterwards. We note $\mathbf{r} = [x\ y\ z] \in \mathbb{R}^3$ the position of the center of mass. In our current implementation, a classic sampling-based motion planning technique such as those described in section 1.2 is applied to explore $\mathbb{R}^3$ (typically either a bi-RRT or a classic PRM). Linear interpolation is used to connect sampled positions. The resulting path is thus a concatenation of n collision-free straight line segments in $\mathbb{R}^3$: $\{(\mathbf{r}_i, \mathbf{r}_{i+1})\}_{i=1..n}$. The next subsection explains how this geometrical path in $\mathbb{R}^3$ is transformed into a trajectory in $\mathcal{X}$.

4.1.1 From path to trajectory

A straightforward way of turning the geometrical path into a well defined trajectory is to apply our spline-based steering method to each pair of hovering states $([\mathbf{r}_i\ 0\ 0], [\mathbf{r}_{i+1}\ 0\ 0])$ along the path. By construction the local trajectories thus obtained are smoothly connected one to the next so that their concatenation is indeed a well defined and admissible trajectory (in terms on the bounds on the derivatives). Their is however a major problem with this approach. The result trajectory might not be collision-free anymore. The reason why is that our steering method, called between two hovering states, does not compute a straight line segment in $\mathbb{R}^3$. This is due to the fact that although the straight line segment is the optimal solution in terms of distance it is not the time-optimal solution.

In order to assure that the trajectory is collision-free we thus have to keep the previously computed straight line segments in $\mathbb{R}^3$ for the center of mass of the quadrotor. For a line segment of index i in the path, let us note:

$$l_i = \|\mathbf{r}_{i+1} - \mathbf{r}_i\| \text{ its length and } \mathbf{u}_i = \frac{\mathbf{r}_{i+1} - \mathbf{r}_i}{l_i} \text{ its unit direction vector}$$

We apply our steering method in one dimension to the couple $([0\ 0\ 0], [l_i\ 0\ 0])$ which gives us:

a duration $T_i \in \mathbb{R}$
a distance profile $(D_i : [0\ T_i] \to \mathbb{R})$
a velocity profile $(V_i : [0\ T_i] \to \mathbb{R}) = \dot{D}_i$
an acceleration profile $(A_i : [0\ T_i] \to \mathbb{R}) = \dot{V}_i$

We can now define the local trajectory of index i by:

$$\forall t \in [0\ T_i],\ \begin{cases} \mathbf{r}(t) = \mathbf{r}_i + D_i(t).\mathbf{u}_i \\ \dot{\mathbf{r}}(t) = V_i(t).\mathbf{u}_i \\ \ddot{\mathbf{r}}(t) = A_i(t).\mathbf{u}_i \end{cases}$$

The local trajectories are now well defined, collision-free and admissible, therefore so is the global trajectory. However the latter is far from being time-optimal because of the imposed stops at the ends of each local trajectories. We will see in the next

sub-section how to address this.

4.1.2 Optimization

A trajectory optimization method can be applied to improve the time-optimality of the previously obtained solution. We have implemented a simple but efficient method based on the random shortcut algorithm [Geraerts 2007]. This iterative, anytime algorithm works as follows: at each iteration, two states, $\mathbf{x}_1$ and $\mathbf{x}_2$, are randomly selected from the overall trajectory. Let us call $\mathbf{x}_A$ the initial state of the local trajectory in which $\mathbf{x}_1$ lies, and $\mathbf{x}_B$ the final state of the local trajectory in which $\mathbf{x}_2$ lies. The steering method is then applied to generate three new local trajectories between $(\mathbf{x}_A, \mathbf{x}_1)$, $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{x}_2, \mathbf{x}_B)$[1]. If they are collision-free, the cost of the trajectory $\mathbf{x}_A \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_2 \rightarrow \mathbf{x}_B$ is computed, the cost here being the overall flying time. If this cost is lower than the one of $\mathbf{x}_A \rightarrow ... \rightarrow \mathbf{x}_B$, this portion of the overall trajectory is replaced by the new one. This step is repeated until a given execution time, a given number of iterations or a given gain of the cost is reached. A example of result is illustrated in Figure 4.1.

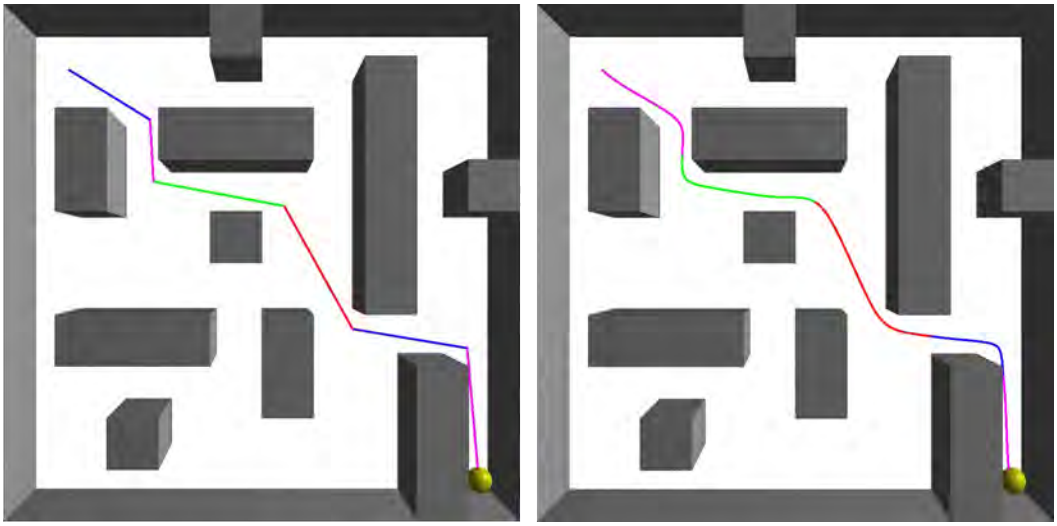


Figure 4.1: In a maze-like environment as seen from the top: the effect of the optimization method on a trajectory composed of six straight line segments in $\mathbb{R}^3$. *Left:* before optimization. *Right:* after optimization

This decoupled approach is computationally very efficient. It quickly returns high-velocity, agile trajectories in cluttered environments. It is however incomplete, in the sense that it will succeed only if a quasi-static solution exists for the bounding sphere. It will for example fail to solve some problems involving aggressive maneu-

[1]Note that the portion of the initial trajectory between $\mathbf{x}_A$ and $\mathbf{x}_1$ (idem for $\mathbf{x}_2$ and $\mathbf{x}_B$) is different to the local trajectory generated by a new call to the steering method, because zero jerk is imposed on $\mathbf{x}_1$ (and on $\mathbf{x}_2$) when splitting the trajectory.

vers, such as the one illustrated in the right hand side of Figure 4.6. The approach is also unsuitable to solve problems involving the transportation of a rigidly-attached large object and for which the bounding sphere is a too large approximation of the geometry of the system. To address these kinds of problems we need to use direct approaches.

4.2 Direct kinodynamic planning

In this section we focus on direct approaches on kinodynamic motion planning for a quadrotor. We first address the problem of the metric in the state space. Follows the description of an incremental sampling strategy in the state space. We finally present two global direct approaches to kinodynamic motion planning and discuss the influence of both the metric and the sampling strategy on both of them.

4.2.1 Quasi-metric in the state space

The efficiency of the state-space exploration using randomized kinodynamic motion planning algorithms relies on a good distance metric. However, as discussed in [LaValle 2001], computing the actual distance between two states is as hard (and thus as costly) as solving the corresponding optimal control problem. Our steering method provides a deterministic sub-optimal solution to such a control problem. Therefore, it defines a quasi-metric[2] $M^*_{SM} : (\mathbf{x}_0, \mathbf{x}_T) \mapsto T$ on the state space. Because of the dynamics of the system, a trajectory in the state space from $\mathbf{x}_0$ to $\mathbf{x}_T$ is indeed necessarily different from a trajectory from $\mathbf{x}_T$ to $\mathbf{x}_0$, and thus M^*_{SM} is not symmetric. Although this steering method is computationally fast, it is still too costly to be used for nearest neighbor search inside a sampling-based planner. This section presents a method to approximate the quasi-metric M^*_{SM} at a very low computational cost, and presents results that show its relevance.

4.2.1.1 Approximate quasi-metric

The complexity of the problem 3.1 defined in Chapter 3 is mainly due to its order (four) and to the inequality constraints on the derivatives. We propose to solve a simpler time optimal control problem for the third order (i.e. by considering the jerk as the control input), in one dimension and without constraints other than the bounds on the control input. The problem is then to find for each output of index

[2] A quasi-metric has all the properties of a metric, symmetry excepted.

i the couple (S_i, T_i) such that:

$$\begin{cases} \min T_i \in \mathbb{R}^+ \text{ s.t.} \\ \\ [S_i(0)\ \dot{S}_i(0)\ \ddot{S}_i(0)] = [x_0\ v_0\ a_0] \in \mathbb{R}^3 \\ \\ [S_i(T_i)\ \dot{S}_i(T_i)\ \ddot{S}_i(T_i)] = [x_{T_i}\ v_{T_i}\ a_{T_i}] \in \mathbb{R}^3 \\ \\ \forall t \in [0, T_i],\ |\dddot{S}_i(t)| \leq j_{max} \in \mathbb{R}^+ \end{cases} \tag{4.1}$$

For this simple integrator of the third order without constraints on the state, Pontryagin maximum principle (see for example [Bertsekas 1995]) says that the optimal control is necessarily saturated, *i.e.*:

$$\forall t \in [0, T_i],\ \dddot{S}_i(t) \in \{-j_{max}, j_{max}\}$$

with at most two control commutations. Solving (4.1) implies to find T_i and these (at most) two commutation times, which requires to solve polynomial equations of maximum degree four.

For a coordinate of index $i \in \{1, 2, 3\}$, let $u(t) = \dddot{S}_i(t)$ be the control function and u its initial value: $u = u(0) = \pm j_{max}$. Let t_1 be the duration of the first phase (during which $u(t) = u$), t_2 the duration of the second phase (during which $u(t) = -u$) and finally t_3 the duration of the third phase (during which $u(t) = u$). Note that $T_i = t_1 + t_2 + t_3$. In case of no control commutation, $t_2 = t_3 = 0$ and in case of one control commutation, $t_3 = 0$. Let us also note $a_k(t)$, $v_k(t)$ and $x_k(t)$ respectively the acceleration, velocity and position during the phase of index $k \in \{1, 2, 3\}$. For example during phase $k = 1$, *i.e.* $t \in [0,\ t_1]$:

$$\begin{cases} a_1(t) = ut + a_0 \\ \\ v_1(t) = \dfrac{u}{2}t^2 + a_0 t + v_0 \\ \\ v_1(t) = \dfrac{u}{6}t^3 + \dfrac{a_0}{2}t^2 + v_0 t + x_0 \end{cases}$$

The expressions for phases $k = 2$ and $k = 3$ are similar. Note however that $a_2(t) = a_2(t, t_1)$ and $a_3(t) = a_3(t, t_1, t_2)$ (the same goes for v and x).

In case of no control commutation, T_i is solution of (4.1) if and only if:

$$\begin{cases} T_i \geq 0 \\ a_1(T_i) = a_F \iff T_i = \dfrac{a_F - a_0}{u} \\ v_1(T_i) = v_F \\ x_1(T_i) = x_F \end{cases}$$

The candidate T_i is tested for $u = \text{sign}(a_F - a_0) j_{max}$.

In case of one control commutation, (T_i, t_1) is solution of (4.1) if and only if:

$$\begin{cases} T_i > t_1 > 0 \\ a_2(T_i, t_1) = a_F \iff T_i = 2t_1 + \dfrac{a_0 - a_F}{u} \quad (eq_1) \\ v_2(T_i, t_1) = v_F \quad (eq_2) \\ x_2(T_i, t_1) = x_F \quad (eq_3) \end{cases}$$

Once T_i expressed as a function of t_1 by (eq_1), solving (eq_2) is solving a second order polynomial equation in t_1. Positive solutions for $u = \pm j_{max}$ are reported in (eq_3) to be tested as candidates.

In case of two control commutations, (T_i, t_1, t_2) is solution of (4.1) if and only if:

$$\begin{cases} t_1 > 0, \quad t_2 > 0, \quad T_i > t_1 + t_2 \\ a_3(T_i, t_1, t_2) = a_F \iff T_i = 2t_2 + \dfrac{a_F - a_0}{u} \quad (eq_4) \\ v_3(T_i, t_1, t_2) = v_F \quad (eq_5) \\ x_3(T_i, t_1, t_2) = x_F \quad (eq_6) \end{cases}$$

Once T_i expressed as a function of t_2 by (eq_4), (eq_5) provides an expression of the form:

$$t_1 = At_2 + \frac{B}{t_2} + C$$

Then solving (eq_6) is solving $t_2[x_3(T_i, t_1, t_2) - x_F] = 0$ which is a fourth order polynomial equation in t_2. The unique solution of (4.1) is the minimum of all valid computed candidates.

The proposed quasi-metric is then defined as:

$$M_{SM} : (\mathbf{x}_0, \mathbf{x}_T) \mapsto \max_{i=1..4} T_i$$

4.2.1.2 Results

Here we present results of an experimental test to validate the proposed approximate quasi-metric. 10^4 pairs of kinodynamic states were randomly sampled in $\mathcal{X} = [-5, 5]^3 \times [-5, 5]^3 \times [-10, 10]^3$, considering $\mathcal{J} = [-20, 20]^3$ and $\mathcal{S} = [-50, 50]^3$. Note that, without loss of generality and for simplification purposes, we consider here a constant yaw. For each pair $(\mathbf{x}_1, \mathbf{x}_2)$, we computed the value $M^*_{SM}(\mathbf{x}_1, \mathbf{x}_2)$ of the quasi-metric induced by our steering method, the value $M_{SM}(\mathbf{x}_1, \mathbf{x}_2)$ given by the proposed approximation, and the value $ED(\mathbf{x}_1, \mathbf{x}_2)$ of the euclidean distance in $\mathbb{R}^3$ considering only the position of the center of mass. We study the distribution of

the relative error between M^*_{SM} and M_{SM}, i.e. the quantity:

$$RE_{M_{SM}}(\mathbf{x}_1, \mathbf{x}_2) = 1 - \frac{M_{SM}(\mathbf{x}_1, \mathbf{x}_2)}{M^*_{SM}(\mathbf{x}_1, \mathbf{x}_2)}$$

For comparison, we also provide the relative error RE_{ED} between M^*_{SM} and ED. Figure 4.2 shows histograms of the distributions of these errors, Table 4.1 shows key statistical values of these distributions and Table 4.2 gives mean CPU times in milliseconds for a single core of an Intel Xeon W3520 processor at 2.67GHz.

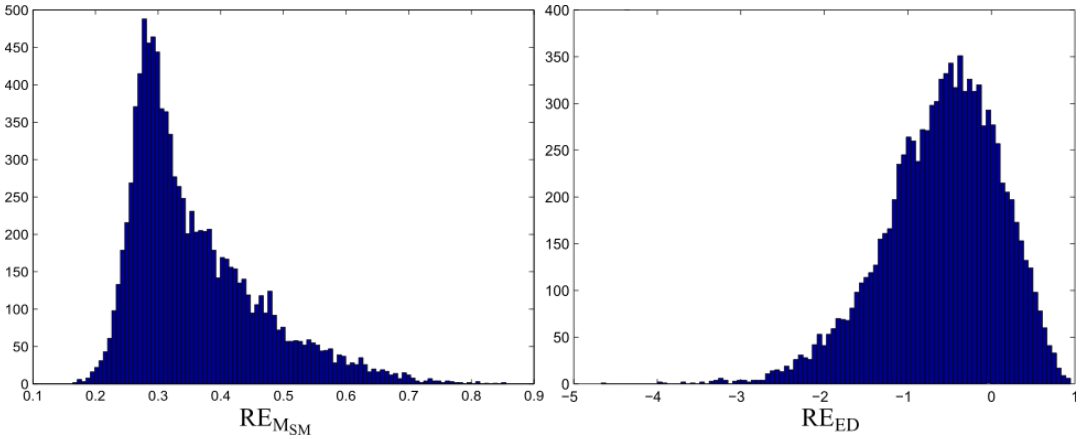


Figure 4.2: Histograms of the distributions of the relative errors. *Left:* our steering method. *Right:* the Euclidean distance.

The low standard deviation of the distribution of the relative error for the proposed quasi-metric is a measure of the quality of the approximation. These results also provide empirical evidence that M_{SM} and M^*_{SM} are equivalent since for all pairs $(\mathbf{x}_1, \mathbf{x}_2)$,

$$0.16396 \leq RE_{M_{SM}}(\mathbf{x}_1, \mathbf{x}_2) \leq 0.85540$$

which implies

$$\frac{1}{10}.M^*_{SM}(\mathbf{x}_1, \mathbf{x}_2) < M_{SM}(\mathbf{x}_1, \mathbf{x}_2) < 10.M^*_{SM}(\mathbf{x}_1, \mathbf{x}_2)$$

This means that M_{SM} and M^*_{SM} are inducing the same topology on $\mathcal{X}$ which means that the cost-to-go defined by our steering method is correctly evaluated by M_{SM}. This is clearly not the case for ED.

4.2.2 Sampling strategy

In many motion planning problems the workspace is bounded. In most of them it is a simple box in $\mathbb{R}^3$. Trajectories generated by the steering method presented in Chapter 3 do not guarantee respect to any bounds on the position of the center

Table 4.1: Distributions of the relative errors

Metric	M_{SM}	ED
Minimum	0.16396	−4.67050
Maximum	0.85540	0.94781
Mean	0.35918	−0.59440
Median	0.32806	−0.52272
Standard deviation	0.10308	0.69674

Table 4.2: Mean CPU times in milliseconds

M^*_{SM}	M_{SM}	ED
1.23×10^{-1}	5.81×10^{-3}	1.10×10^{-4}

of mass of the robot. Such constraints are then typically violated when samples are close to the boundary of the workspace and the velocity is high, so that it is not possible to decelerate to avoid crossing this positional limit. In a similar way, bounds on velocity can also be violated. If acceleration is too high and velocity is close to the limit, produced trajectories will be invalid because velocity can not be reduced in time to meet the constraints. Note however that the imposed shape for the trajectories produced by our steering method guarantees that bounds on acceleration are respected.

This section presents an incremental state-space sampling technique that increases the probability of generating connectible states. The definition of such states is first presented. Then the different steps of the method are explained. Finally some results are provided.

4.2.2.1 State connectibility

In this section we will use the notations and concepts defined in section 1.3.1. We also introduce the set $\mathcal{X}_{phys} \supseteq \mathcal{X}_{valid}$ that contains all the states that satisfy the physical constraints. We recall that given a state $\mathbf{x}_0 \in \mathcal{X}_{phys}$, a control u admissible on $[t_0\ t_F]$, is said to be feasible if and only if the associated response is such that $\forall t \in [t_0\ t_F],\ \mathbf{x}_{[t_0,\mathbf{x}_0,\mathbf{u}]}(t) \in \mathcal{X}_{phys}$. The set of such controls is noted $\mathcal{U}_{[\mathbf{x}_0],t_0,t_F}$

Provided a time t_0, we say that a state $\mathbf{x_0} \in \mathcal{X}_{phys}$ is *forward-connectible* if and only if

$$\mathcal{U}_{[\mathbf{x}_0,t_0,+\infty]} \neq \varnothing$$

This means that for a state that is not forward-connectible every admissible control is unfeasible, or in other terms that whatever admissible control is applied to the system, the physical constraints will be violated at some time in the future. This definition is similar to that of the inevitable collision states proposed by [Fraichard 2004]. Similarly, we say that a state $\mathbf{x}_0 \in \mathcal{X}_{phys}$ is *backward-connectible*

if and only if $\mathcal{U}_{[\mathbf{x}_0,-\infty,t_0]} \neq \varnothing$. A state that is both forward-connectible and backward-connectible is said to be connectible. A state that is either not forward-connectible or not backward-connectible is said to be non-connectible. The idea is coarsely illustrated on Figure 4.3.

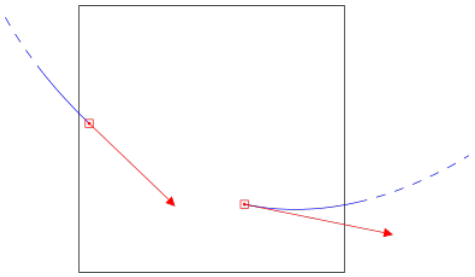


Figure 4.3: Examples of non-connectible states in two dimensions. Red squares are positions and red arrows are velocity vectors. Blue curves are examples of trajectories. Bounds on position are represented in black. The state on the left is not backward-connectible. The state on the right is not forward-connectible.

No solution to a kinodynamic motion planning problem can pass through a non-connectible state since that if it were the case the solution would not be feasible. Such states are therefore always useless and should not be sampled. In case of uniform sampling though, non-connectible states can be generated. The local paths computed to connect those states have then to be discarded *a posteriori* by the planner. This is rather inefficient since generating and testing a local path for validity is a costly operation. The goal of the sampling technique proposed below is to notably reduce the probability of generating non-connectible states, and hence to improve the performance of planning algorithms.

4.2.2.2 Proposed approach

The sampling technique proceeds in a decoupled and incremental way. First, acceleration is uniformly sampled. The idea is then to compute a set of velocity values for which the state is known to be non-connectible. Velocity is then uniformly sampled outside this set. Finally, given this couple (velocity, acceleration) a set of position values for which the state is known to be non-connectible is computed, and the position is then uniformly sampled outside this set.

4.2.2.3 Sampling velocity

This subsection explains how to compute the set of velocity values for which the state is known to be non-connectible, given a uniformly sampled acceleration value a_s. Explanations are given for one output. Figure 4.4 illustrates this explanation. Let us denote v_{max} and a_{max} as the bounds on the absolute value of velocity and

acceleration respectively. We study acceleration $a(t)$ and velocity $v_0(t)$ on a neighborhood around $t = 0$ for $a(0) = a_s$ and $v_0(0) = 0$. The idea is to apply a saturated acceleration variation and determine the extrema of $v_0(t)$ in this neighborhood. Using them, we can compute the limits on velocity v_s such that $v(t) = v_0(t) + v_s$ lies in $[-v_{max}, v_{max}]$. We use notations defined in section 3.2.1. For $t > 0$, phase A of our steering method is applied. Phase H is applied for $t < 0$. The sampled value a_s locally imposes a direction of variation of $v_0(t)$ on phases A and H. We want to reverse this direction of variation in minimum time. This is equivalent to driving $a(t)$ to zero in minimum time. For that, we set $a_B = a_G = -\text{sign}(a_s).a_{max}$. This corresponds to the highest acceleration variation achievable by our steering method. Note that, by construction, acceleration is symmetric during phases A and H (i.e $a(-t) = a(t)$) and $v_0(t)$ is anti-symmetric (i.e $v_0(-t) = -v_0(t)$). Since $a(t)$ is a second order spline strictly monotonic on phase A, it is straightforward to compute the unique $t_0 > 0$ such that $a(t_0) = 0$. The value $v_{bound} = v_{max} - |v_0(t_0)|$ is then the upper bound on the absolute value of v_s. This means that if $|v_s| > v_{bound}$ then $v(t) = v_0(t) + v_s$ will violate the constraints on velocity. A velocity value v_s is then uniformly sampled in $[-v_{bound},\ v_{bound}]$.

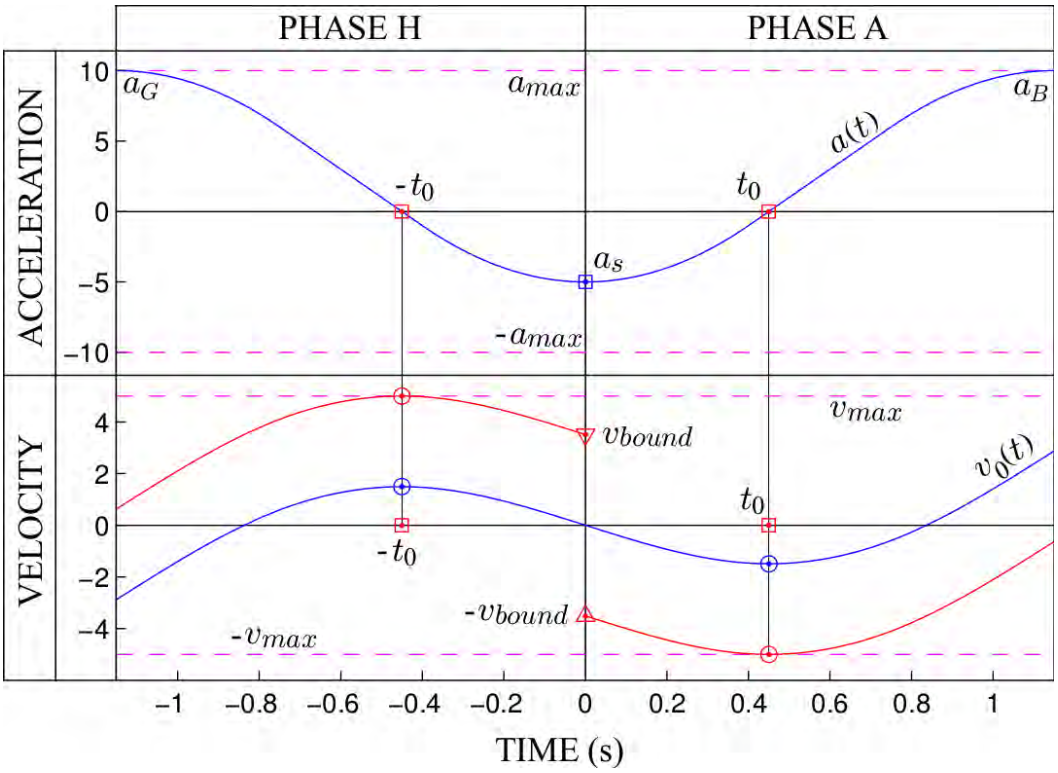


Figure 4.4: Acceleration and velocity (blue curves) around $t = 0$. Saturated acceleration variation is applied in order to determine the limits on initial velocity (red triangles). Red curves are velocities for these limits.

4.2.2.4 Sampling position

Given a couple (v_s, a_s) for one output, this subsection explains how to compute the set of position values for which the state is known to be non-connectible. Figure 4.5 illustrates this explanation. The principle is similar to the one in the previous subsection. Velocity $v(t)$ and position $x_0(t)$ are studied around $t = 0$ for $a(0) = a_s$, $v(0) = v_s$ and $x_0(t) = 0$. We apply a saturated velocity variation and determine the extrema of $x_0(t)$ in this neighborhood. Using them, we can compute the limits on position x_s such that $x(t) = x_0(t) + x_s$ lies in $[-x_{max}, x_{max}]$ (bounds on position). For $t > 0$, phases A to C of our steering method are applied. Phases E to H are applied for $t < 0$. We want to reverse the direction of variation of the position imposed by v_s as fast as possible. This is equivalent to driving $v(t)$ to zero in minimum time. For that, we set $v_D = -\text{sign}(v_s).v_{max}$ for both phases A to C and E to H. This corresponds to the highest velocity variation achievable by our steering method. The only difference here is that neither $v(t)$ nor $x_0(t)$ have symmetry proprieties. We compute $t^+ > 0$ such that $v(t^+) = 0$ and $t^- < 0$ such that $v(t^-) = 0$. If $v_s \geq 0$ then $x^+ = x_{max} - x_0(t^+)$ and $x^- = -x_{max} - x_0(t^-)$ else $x^+ = x_{max} - x_0(t^-)$ and $x^- = -x_{max} - x_0(t^+)$. A position value x_s is then uniformly sampled in $[x^-,\ x^+]$.

4.2.2.5 Results

We provide here some results concerning the sampling strategy. The conducted experiment consisted in testing the validity of local paths computed between uniformly sampled pairs of states in $\mathcal{X} = [-5, 5]^3 \times [-5, 5]^3 \times [-10, 10]^3$, with $\mathcal{J} = [-20, 20]^3$ and $\mathcal{S} = [-50, 50]^3$. Yaw was kept constant. We measured the percentage of valid paths (i.e. which lies entirely in $\mathcal{X}$) over 10^4 calls. We then repeated this operation using our sampling technique. With uniform sampling only 11.53% of the produced paths were valid. With our sampling technique 95.58% of the paths were valid. Moreover, we can test for a given state if each output is respecting the constraints defined by our sampling technique, i.e. if velocity lies in $[-v_{bound},\ v_{bound}]$ and if position lies in $[x^-,\ x^+]$. When this is not the case we consider that the state is not connectible. Our sampling technique is obviously generating connectible states every time whereas with uniform sampling about 90% of the generated states are not connectible (with respect to this criteria).

4.2.3 Global methods for kinodynamic planning

This section describes variants of RRT and PRM algorithms, to deal with a non-symmetric steering method, and provides some results on the influence of both the proposed quasi-metric and the sampling strategy on them. These algorithms are well known, but had to be adapted to the non-symmetry of the steering method and the associated quasi-metric. Because of this non-symmetry, the underlying graph is

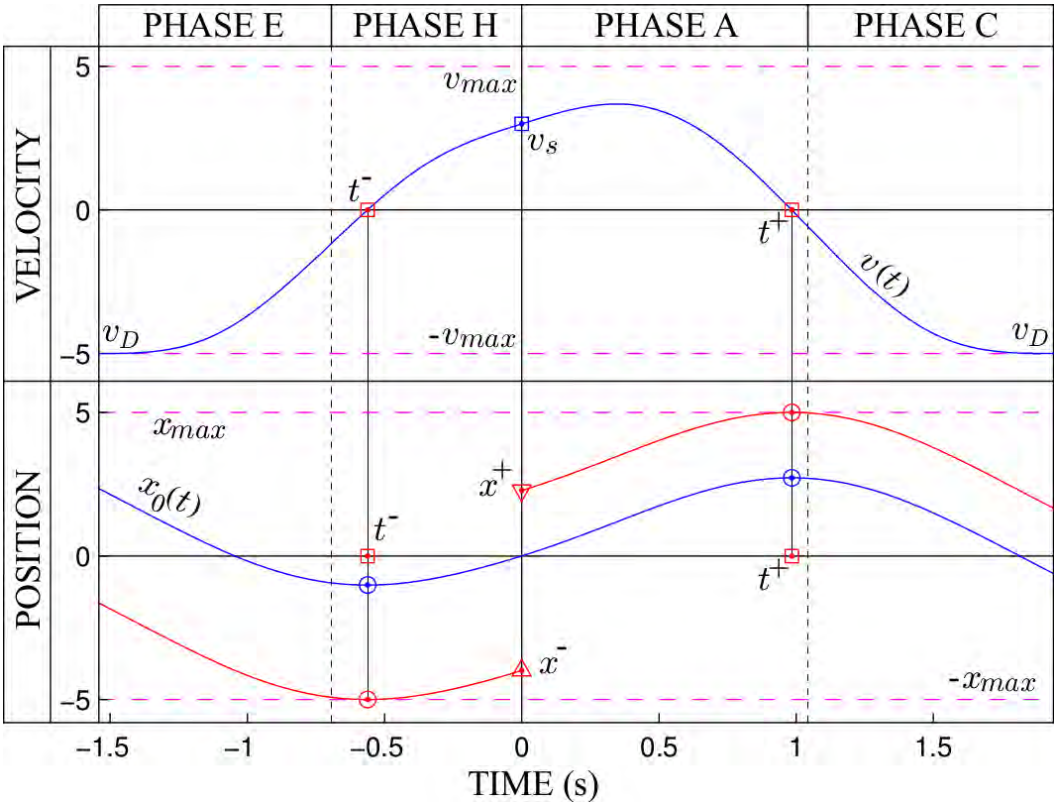


Figure 4.5: Velocity and position (blue curves) around $t = 0$. Saturated velocity variation is applied in order to determine the limits on initial positions (red triangles). Red curves are positions for these limits.

directed. We first present a directed bi-RRT then a directed PRM using the forest method and we finally show and discuss some results.

4.2.3.1 Directed Bi-RRT

In the well known undirected version of the Bi-RRT algorithm (see section 1.2.4), two trees $\mathcal{T}_S$ and $\mathcal{T}_G$ are constructed in parallel. $\mathcal{T}_S$ grows from the start configuration and $\mathcal{T}_G$ from the goal configuration. Each iteration for one of the trees consists of sampling a configuration q_{rand}, finding its nearest neighbor q_{near} in the tree (according to a defined metric), and extending it toward q_{rand} (using a steering method) to create a new configuration q_{new}. Each time an expansion is successful for one of the trees, a direct connection is attempted between q_{new} and its nearest neighbor in the other tree. The algorithm ends if this local path is valid (i.e. when the trees are connected).

In our directed version, both the steering method and the quasi-metric M_{SM} are non-symmetric, and thus have to be called taking care of the order of the two states.

The nearest neighbors $N_S(\bar{\mathbf{x}})$ and $N_G(\bar{\mathbf{x}})$ of a state $\bar{\mathbf{x}}$ in $\mathcal{T}_S$ and $\mathcal{T}_G$ respectively are defined as such:

$$\begin{cases} N_S(\bar{\mathbf{x}}) = \arg\min_{\mathbf{x}\in\mathcal{T}_S} M_{SM}(\mathbf{x}, \bar{\mathbf{x}}) \\ N_G(\bar{\mathbf{x}}) = \arg\min_{\mathbf{x}\in\mathcal{T}_G} M_{SM}(\bar{\mathbf{x}}, \mathbf{x}) \end{cases}$$

For an expansion of $\mathcal{T}_S$, we test the local path $(N_S(\mathbf{x}_{rand}), \mathbf{x}_{new})$ for validity. In case of success, the algorithm ends if the local path $(\mathbf{x}_{new}, N_G(\mathbf{x}_{new}))$ is valid. For an expansion of $\mathcal{T}_G$, the local path $(\mathbf{x}_{new}, N_G(\mathbf{x}_{rand}))$ is tested for validity, and the algorithm ends in case of validity of the local path $(N_S(\mathbf{x}_{new}), \mathbf{x}_{new})$.

4.2.3.2 Directed PRM

At each iteration of the undirected version of the PRM algorithm (see section 1.2.3), a collision free configuration q is sampled and added to the graph $\mathcal{G}$. For every connected component G_i of $\mathcal{G}$, connections are attempted between q and each node of G_i in increasing order of distance from q until one is successful. A threshold on this distance can be considered with the aim to reduce computational cost. In our directed version, we consider the strongly connected components G_i of $\mathcal{G}$. Moreover, we maintain during the execution the adjacency matrix $A_{\mathcal{G}}$ of the transitive closure of the graph of the strongly connected components of G. This square matrix, whose dimension is the number of strongly connected components, is defined by $A_{\mathcal{G}}[i][j] = 1$ if a path in $\mathcal{G}$ exists from every node of G_i to every node of G_j and $A_{\mathcal{G}}[i][j] = 0$ otherwise. If $A_{\mathcal{G}}[i][j] = 1$ we say that G_i is connected to G_j. Note that $A_{\mathcal{G}}[i][j] = A_{\mathcal{G}}[j][i] = 1$ if and only if $i = j$.

At each iteration, a valid state $\mathbf{x}$ is sampled and added to $\mathcal{G}$ (which has n strongly connected components). Its strongly connected component $G_{n+1} = \{\mathbf{x}\}$ is added to the matrix $A_{\mathcal{G}}$. For every connected component G_i of $\mathcal{G}$ ($i = 1..n$), if G_i is not connected to G_{n+1}, connections from every node $\mathbf{x}_j$ of G_i to $\mathbf{x}$ are attempted in increasing order of $M_{SM}(\mathbf{x}_j, \mathbf{x})$ until one is valid. As for the undirected version, a threshold on the value of M_{SM} can be considered here. $A_{\mathcal{G}}$ is updated if neecessary. Then, if G_{n+1} is not connected to G_i, connections from $\mathbf{x}$ to every node $\mathbf{x}_j$ of G_i are attempted in increasing order of $M_{SM}(\mathbf{x}, \mathbf{x}_j)$ until one is valid.

If used in single query mode, the algorithm ends when the strongly connected component of the initial state is connected to the strongly connected component of the goal state.

4.2.4 Experimental results

Results presented below show the influence of the quasi-metric and the sampling technique on the two previously presented motion planners. Experiments have been conducted on two different environments shown in Figure 4.6 and for the

Table 4.3: B: boxes, S: slots, P: PRM, R: RRT, M: proposed metric, E: euclidean metric, I: incremental sampling, U: uniform sampling

Experiment	BPMI	BPEI	BPMU	BPEU
CPU time (s)	0.05648	0.07884	3.284	4.409
Flying time (s)	8.180	8.772	8.000	8.126
Number of nodes	12.11	13.77	78.64	88.33
% of not connectible nodes	0	0	82.53	84.38

Experiment	BRMI	BREI	BRMU	BREU
CPU time (s)	0.02780	0.04088	0.04144	0.05612
Flying time (s)	9.674	10.84	9.365	10.09
Number of nodes	8.79	8.84	9.18	10.77
Number of iterations	26.45	45.04	45.58	65.02
% of not connectible nodes	50.34	54.94	51.51	59.85

Experiment	SPMI	SPEI	SPMU	SPEU
CPU time (s)	1.505	1.220	578.5	444.2
Flying time (s)	9.074	8.979	8.615	8.387
Number of nodes	71.93	61.59	767.9	725.8
% of not connectible nodes	0	0	82.53	84.38

Experiment	SRMI	SREI	SRMU	SREU
CPU time (s)	2.165	2.466	558.8	512.9
Flying time (s)	25.42	34.72	33.96	55.98
Number of nodes	334.5	565.6	4429.4	8813.0
Number of iterations	982.9	2502.9	30253.7	196233.3
% of not connectible nodes	25.60	34.86	79.41	82.48

same quadrotor whose diameter is equal to 0.54 meters. We consider $\mathcal{V} = [-5, 5]^3$, $\mathcal{A} = [-10, 10]^3$, $\mathcal{J} = [-20, 20]^3$, $\mathcal{S} = [-50, 50]^3$ (using SI base units). Yaw is kept

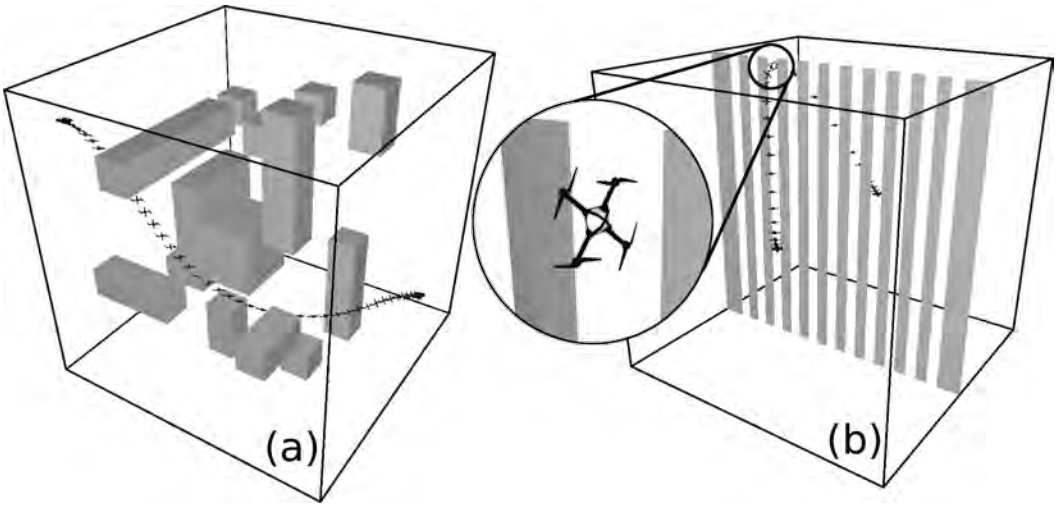


Figure 4.6: Testing environments (a) *boxes* (b) *slots*

constant. The first environment, referred to as *boxes*, is a cube with side length of 10 meters filled with box shaped-obstacles of different sizes. The second environment, referred to as *slots*, is also a cube with side length of 10 meters but divided in two halves by a series of aligned obstacles separated by 0.40 meters (hence smaller than the robot diameter). This problem is particularly challenging since going across these obstacles requires to find a path in a very narrow passage in the state-space. Every combination of environment, algorithm, metric and sampling strategy has been tested. Results are provided in Tab. 4.3 for CPU and flying times in seconds, number of nodes (and iterations for the RRT) and percentage of not connectible nodes (with respect to the criteria defined in section 4.2.2.5). Each experiment is designated by an acronym whose meaning is explained in the caption. Results are averaged over 100 runs and are for an implementation in C, integrated in our motion planning software Move3D [Siméon 2001], and run on a single core of an Intel Xeon W3520 processor at 2.67GHz.

Results show a significant improvement of the performance of both algorithms thanks to the integrations of the proposed techniques. However, one can clearly see that the metric and the sampling technique have a more notable effect on one or the other planner. Results for the PRM algorithm shows that the sampling method has a great influence on its performance. Its integration indeed improves CPU time by two orders of magnitude for both environments. On the other hand, one can see that for the *slots* environment CPU times are slighlty worse with the use of the metric. This can be explained by the difference of computing time between our quasi-metric and the euclidean distance. This is also observed in one case for the RRT algorithm (SRMU vs. SREU). For the RRT algorithm, results show that the influence of the metric is more important. This was to be expected since RRT-based algorithms are known to be very sensitive to the metric. One can see that the number of iterations is significantly reduced, meaning that the search is better guided. The improvement produced by the sampling technique is also very significant for the *slots* environment but less noticeable for the *boxes* environment. This can be explained by the fact that, in RRT-based algorithms, the sampled states are not tested for connections but used to define a direction for extension. A new state is then generated according to that direction. One can see that, for the *boxes* environment, about half of these states are not connectible regardless of the sampling method. Finally note that flying times are given for the raw, non-smoothed trajectories, which explains the rather large difference of path quality between PRM and RRT results.

Chapter 5

Experimental work

Contents

The goal of this chapter is to show that the trajectories that we plan using the methods described in the two previous chapters can actually be executed on a real physical system. We first present the controller we have chosen to track our trajectories. We then give an overview of our testbed: its different components, both in terms of hardware and software, and how they interconnect. We finally present the conducted experimentation together with some of its results.

5.1 Geometric tracking controller on $SE(3)$

In section 2.2.1 we have briefly introduced some general notions about control theory and have explained in section 2.2.2 what are the control space and the state space of a quadrotor. We also recall that in section 2.4 we have defined its flat outputs for which we plan our trajectories rather than doing it for the actual state. In this section we present the controller that we have chosen in order to follow our planned trajectories. It is a geometric tracking controller on $SE(3)$ proposed by [Lee 2010]. A comprehensive presentation of it can obviously be found in the referenced article, but we would like to summarize here its main characteristics.

5.1.1 Overview

In addition to the notations defined in Chapter 2 we will need some new ones. We consider a nominal trajectory $\overline{\mathbf{z}}$. The over line notation $\overline{.}$ indicates that we are dealing with a nominal quantity (or reference) as defined in section 2.2.1, whereas the over tilde notation $\widetilde{.}$ indicates a measurement (or estimation).

$$\overline{\mathbf{z}} : \left(\begin{array}{c} [0\ T] \to \mathcal{P} \\ t \mapsto [\overline{x}(t)\ \overline{y}(t)\ \overline{z}(t)\ \overline{\psi}(t)]^T \end{array} \right), \text{ with } \left\{ \begin{array}{l} T \in \mathbb{R}^{+*} \\ \mathcal{P} \text{ the space of the flat outputs} \end{array} \right.$$

Note that this trajectory in the space of the flat outputs is the result of the planning methods presented in the two previous chapters and therefore we also have access to both $\dot{\overline{\mathbf{z}}}$ and $\ddot{\overline{\mathbf{z}}}$. For the sake of convenience we will omit to write from now on the time parameter and simply write $\overline{\mathbf{r}} = [\overline{x}\ \overline{y}\ \overline{z}]^T$ the desired position of the center of mass at any given time. Therefore we simply have $\overline{\mathbf{z}} = [\overline{\mathbf{r}}\ \overline{\psi}]^T$.

In section 2.4 we saw that from $\ddot{\overline{\mathbf{r}}}$ and $\overline{\psi}$ we can define the first nominal body axis $\overline{\mathbf{b_1}}$. The inputs of this controller are $\overline{\mathbf{r}}$, $\dot{\overline{\mathbf{r}}}$, $\ddot{\overline{\mathbf{r}}}$, $\overline{\mathbf{b_1}}$, the nominal angular velocity $\overline{\mathbf{\Omega}}$ expressed in the body-fixed frame and the nominal angular acceleration $\dot{\overline{\mathbf{\Omega}}}$. Its outputs are $\overline{f}$ the desired net thrust and $\overline{\mathbf{M}}$ the desired total moment expressed in the body-fixed frame. In order to close the loop it also relies on the estimated values of the rotational matrix $\widetilde{R}$, the position of the center of mass $\widetilde{\mathbf{r}}$, its velocity $\dot{\widetilde{\mathbf{r}}}$ and the angular velocity $\widetilde{\mathbf{\Omega}}$

The overall structure is as follows. As a first step both the desired net thrust and $\overline{f}$ and a desired third body axis $\mathbf{b_{3_d}}$ different from the nominal $\overline{\mathbf{b_3}}$ are computed. As a second step a desired rotational matrix R_d different from the nominal $\overline{R}$ is defined and used to compute the desired total moment $\overline{\mathbf{M}}$. The first step is referred to as *trajectory tracking* whereas the second step is referred to as *attitude tracking*.

5.1.2 Trajectory tracking

First, the tracking errors are defined as follows:

$$\begin{array}{ll} \mathbf{e}_r = \widetilde{\mathbf{r}} - \overline{\mathbf{r}} & \text{the error in position} \\ \mathbf{e}_v = \dot{\widetilde{\mathbf{r}}} - \dot{\overline{\mathbf{r}}} & \text{the error in velocity} \end{array}$$

Then for some positive quantities k_r and k_v, the vector $\mathbf{t}$ that is in the direction of the desired thrust is defined as follows:

$$\mathbf{t} = -k_r\mathbf{e}_r - k_v\mathbf{e}_v - mg\mathbf{e_3} + m\ddot{\overline{\mathbf{r}}},$$

where m is the total mass, g the Earth gravitational acceleration and $\mathbf{e_3} = [0\ 0\ 1]^T$.

From it, both the third desired body axis and the desired net thrust are defined

as follows:

$$\mathbf{b_{3_d}} = -\frac{\mathbf{t}}{\|\mathbf{t}\|} \text{ and } \overline{f} = -\mathbf{t} \bullet \widetilde{R}\mathbf{e_3},$$

where the operator $\bullet$ is the dot product.

5.1.3 Attitude tracking

At that step, a desired rotational matrix R_d different from the nominal $\overline{R}$ is computed. Its third axis $\mathbf{b_{3_d}}$ has already been defined in the previous step. Its second axis is defined as follows:

$$\mathbf{b_{2_d}} = \frac{\mathbf{b_{3_d}} \times \overline{\mathbf{b_1}}}{\|\mathbf{b_{3_d}} \times \overline{\mathbf{b_1}}\|}$$

In order for R_d to be orthonormal, the first axis cannot be the nominal $\overline{\mathbf{b_1}}$. Instead, $\overline{\mathbf{b_1}}$ is projected onto the plane normal to $\mathbf{b_{3_d}}$, which simply put corresponds to compute $\mathbf{b_{1_d}} = \mathbf{b_{2_d}} \times \mathbf{b_{3_d}}$. The tracking errors can now be defined from $R_d = [\mathbf{b_{1_d}}\ \mathbf{b_{2_d}}\ \mathbf{b_{3_d}}]$ as follows:

$$\mathbf{e}_R = \frac{1}{2}\left(R_d^T\widetilde{R} - \widetilde{R}^T R_d\right)^\vee \quad \text{the rotational error}$$

$$\mathbf{e}_\Omega = \widetilde{\mathbf{\Omega}} - \widetilde{R}^T R_d \overline{\mathbf{\Omega}} \quad \text{the angular velocity error}$$

where $.^\vee$ is the vee operator defined in section 2.4.

For some positive quantities k_R and k_Ω, the desired moment is then computed as follows:

$$\overline{M} = -k_R\mathbf{e}_R - k_\Omega\mathbf{e}_\Omega + \widetilde{\mathbf{\Omega}} \times J\widetilde{\mathbf{\Omega}} - J\left(\hat{\widetilde{\mathbf{\Omega}}}\widetilde{R}^T R_d\overline{\mathbf{\Omega}} - \widetilde{R}^T R_d\dot{\overline{\mathbf{\Omega}}}\right),$$

where J is the inertia matrix and $\hat{}$ is the hat operator defined in section 2.4.

In our current implementation though we simplify this expression and only keep:

$$\overline{M} = -k_R\mathbf{e}_R - k_\Omega\mathbf{e}_\Omega$$

The reason is that we do not have the inertia matrix and using an approximation of it would do more harm than good.

5.2 ART: the Aerial Robotics Testbed

In this section we provide an overview of our set-up.

5.2.1 Hardware

The Aerial Robotics Testbed (ART) serves as an experimental benchmarking tool to test and validate the planning, control, and estimation algorithms on aerial robotics developed by the researchers at LAAS. It consists of a safety net surrounding an indoor flight volume with a ground area of 6 meters by 3 meters covered with protective mattresses for a height of 4 meters. It is monitored by several cameras from OptiTrack (see http://optitrack.com) that are able to provide a 6DoF tracking of any object equipped with reflective markers at a frequency from 30 Hz up to 200 Hz.

We use a quadrotor from MikroKopter (see http://www.mikrokopter.de) that we assembled ourself. It is equipped with an accelerometer and a gyroscope. On board is running a brush-less controller developed at LAAS-CNRS that takes the four nominal angular velocities of the propeller as control inputs. See Figure 5.1.

Figure 5.1: A MikroKopter quadrotor

5.2.2 Software

On this section we present the different pieces of software used in the ART and explain how they interconnect. A work flow diagram is represented in Figure 5.2.

Kinodynamic motion planning

In Chapter 3 we presented a spline-based steering method that has been implemented as a standalone C++ library named KDTP (for KinoDynamic Trajectory

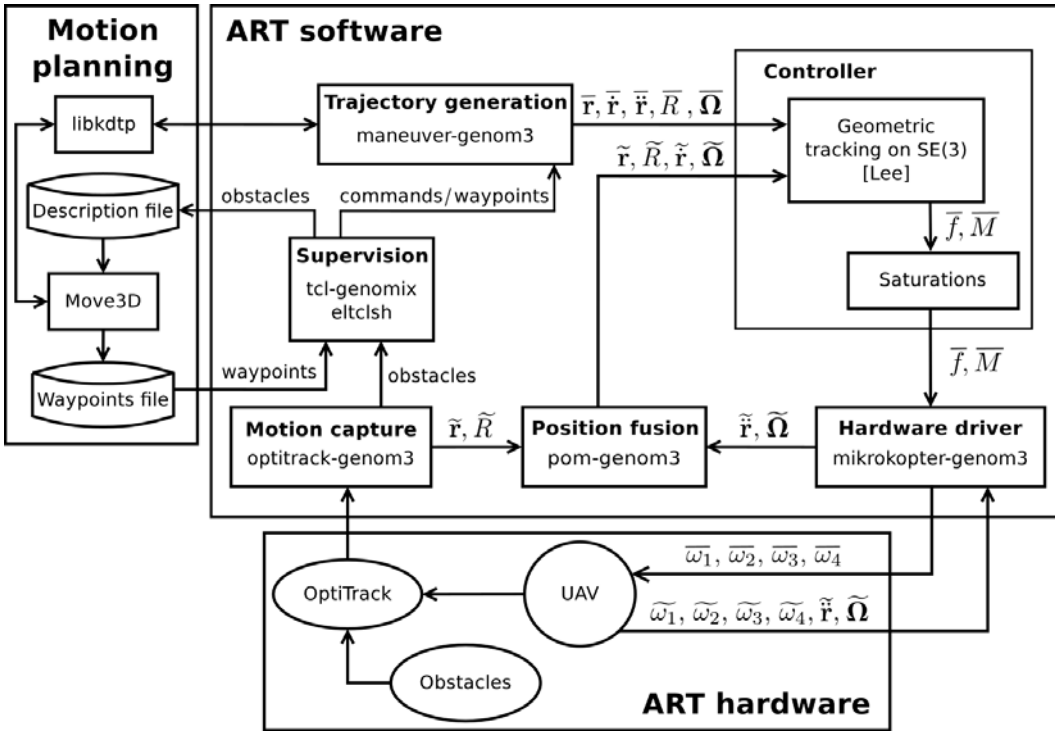


Figure 5.2: Work flow diagram of the ART software architecture

Planner). At the planning level it is used as a local planner by our software Move3D [Siméon 2001] in which the global planning methods presented in Chapter 4 are implemented. Both the environment and the system are described in a file that is loaded at initialization. A trajectory is represented by a concatenation of local trajectories produced by libkdtp. Once a trajectory has been planned, it is exported into a file as a list of waypoints that are the end states of the local trajectories, one per line. Since we impose both zero angular velocity and zero angular acceleration for those states, the exported format is: $x \quad y \quad z \quad \psi \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad \ddot{x} \quad \ddot{y} \quad \ddot{z}$

Sources:
KDTP: https://git.openrobots.org/projects/libkdtp (author: Alexandre Boeuf)
Move3D: https://redmine.laas.fr/projects/move3d (author: RIS team)

Middleware

The Generator of Modules GenoM is a tool to design real-time software architectures. It encapsulates software functions inside independent components. GenoM is more specifically dedicated to complex on-board systems, such as autonomous mobile robots or satellites, that require:

- The integration of heterogeneous functions with different real-time constraints

and algorithm complexities (control of sensors and actuators, data processing, task planning, etc.).

- An homogeneous integration of these functions in a control architecture which requires coherent and predictable behaviors (starting, ending, error handling), and standard interfaces (configuration, control flow, data flow).
- The management of parallelization, physical distribution and portability of the functions.
- Simple procedures to add, modify or (re)use the functions by non-specialists.

GenoM generates the source code of components by using:

- A generic template, common for all components. This guarantees that all components share the same consistent behavior. The template itself is not part of GenoM, so that different template kind can be developed easily.
- A formal description of the components interface. This description is based on a simple language using OMG IDL for data types definitions and a custom syntax for the description of a more detailed component model.

The project is released under an open-source, BSD-like license. See the project page https://git.openrobots.org/projects/genom3

In addition, we use the genomix HTTP server that is a generic interface between clients and genom components (using the generic genom C client template). Control is done by the mean of specific HTTP GET requests. See the project page https://git.openrobots.org/projects/genomix

Supervision

The *tcl-genomix* component provides a TCL package that interacts with the genomix HTTP server and can control GenoM3 components. It can prompt interactively for arguments of services. Services can be invoked synchronously or asynchronously and callbacks can be triggered whenever services complete. It is used in combination with *eltclsh* (editline tcl shell) that is an interactive shell for the TCL programming language. It provides command line editing, history browsing as well as variables and command completion thanks to editline features. The completion engine is programmable in a way similar to tcsh, and comes with an intelligent completion for the full tcl language by default. In our current implementation, routines handling the retrieval of waypoints in a file and the writing of the position of the obstacles in the description file are written in a tcl script and can be called from eltclsh. At initialization, this script is also in charge of setting up several parameters like the gyroscope and accelerometer calibration

of the UAV and the maximum values of the flat output derivatives for KDTP and of connecting all GenoM3 components together.

Sources:
tcl-genomix: https://git.openrobots.org/projects/tcl-genomix (author: Anthony Mallet)
eltclsh: https://git.openrobots.org/projects/tcl-genomix (author: Anthony Mallet)

Trajectory generation

This component called *maneuver-genom3* is in charge of both handling simple maneuvers like vertical take-off/landing or goto and waypoint-based trajectories. It relies on libkdtp for local trajectory planing between waypoints and is controlled via commands entered on the eltclsh shell. For instance a specific command will add all the waypoints from a given file so that they can be interpolated on the run by libkdtp. Its outputs are the nominal position of the center of mass $\overline{\mathbf{r}}$, the nominal velocity of the center of mass $\dot{\overline{\mathbf{r}}}$, the nominal acceleration of the center of mass $\ddot{\overline{\mathbf{r}}}$ and the nominal matrix of rotation $\overline{R}$.

Sources:
https://git.openrobots.org/projects/maneuver-genom3 (author: Anthony Mallet)

Controller

In section 5.1 we presented the geometric tracking controller that we are using. It has been implemented and encapsulated in a component called *nhfc-gemom3*. NHFC stands for Near-Hovering Flight Controller. This name refers to a previous version of the controller and is now outdated. Because Lee's controller does not guarantee that the computed command is admissible, a saturation step has been implemented. It first checks if $\overline{f} \leq 4f_{max}$. If not $\overline{f}$ is set to $4f_{max}$ and $\overline{\mathbf{M}}$ to $[0\ 0\ 0]^T$. Otherwise the desired moment is checked. The four desired thrust forces are computed (see section 2.1):

$$\begin{bmatrix} \overline{f_1} \\ \overline{f_2} \\ \overline{f_3} \\ \overline{f_4} \end{bmatrix} = \Gamma^{-1} \begin{bmatrix} \overline{f} \\ \overline{\mathbf{M}} \end{bmatrix}$$

If one of them is outside the interval $[0,\ f_{max}]$ the maximum λ such that

$$\Gamma^{-1} \begin{bmatrix} \overline{f} \\ \lambda\overline{\mathbf{M}} \end{bmatrix} \in [0,\ f_{max}]^4$$

is searched by dichotomy on the interval $[0,\ 1[$. The new saturated desired moment is set to $\lambda\overline{\mathbf{M}}$. This corresponds to projecting the desired moment onto the frontier of the admissible moments $\mathcal{M}(\overline{f})$ illustrated in section 2.3.2 in the direction of the origin.

Sources:
https://git.openrobots.org/projects/nhfc-genom3 (author: Marco Tognon)
Note that the description on the web page is outdated.

Hardware driver

This component called *mikrokopter-genom3* is a low level hardware controller for MikroKopter quadrotors. It takes the nominal net thrust $\overline{f}$ and the nominal moment vector in body frame $\overline{\mathbf{M}}$ as control inputs and computes the propellers velocities $(\omega_i)_{i=1..4}$ accordingly. Those are then sent to the UAV.

Sources:
https://git.openrobots.org/projects/mikrokopter-genom3 (author: Anthony Mallet)

Motion capture

This component called *optitrack-genom3* is in charge of exporting motion capture data from an OptiTrack system.

Sources:
https://git.openrobots.org/projects/optitrack-genom3 (author: Anthony Mallet)

Position fusion

This component called *pom-genom3* collects measurements from other components, and generate a fused state estimation from these sources.

Sources:
https://git.openrobots.org/projects/pom-genom3 (author: Anthony Mallet)

5.3 Experimentation

In this section we present an simple experiment that we conducted in order to show that a planned trajectory can be followed.

5.3.1 Set-up

The experimental set-up is as follows. Three cylindrical obstacles of length 1.35 m and diameter 9.56 cm are hanging from the ceiling at mid-hight 1.20 m from the ground. They are placed at $(x; y)$ positions relative to the origin of our testbed $(-0.94\ m; 0.36\ m)$, $(0.06\ m; -0.32\ m)$ and $(1.02\ m; -0.32\ m)$ respectively. The 3D model we use for collision detection is a sphere of diameter 50 cm centred at the origin of the quadrotor's body frame. At the planning level both the altitude and the yaw angle are kept constant at 1.2 m and 0° respectively. We plan a trajectory going from the initial hovering position $(-2.0\ m, 0.0\ m, 1.2\ m)$ to the final hovering position $(2.0\ m, 0.0\ m, 1.2\ m)$. The planning method is the decoupled approach with a bi-directional RRT algorithm. In the next section we give the results for two runs with different maximum velocities. In the first one we set $v_{max} = 1\ m.s^{-1}$ and in the second $v_{max} = 2\ m.s^{-1}$. The other bounds are $a_{max} = 5\ m.s^{-2}$, $j_{max} = 20\ m.s^{-3}$ and $s_{max} = 50\ m.s^{-4}$. See Figure 5.3 for a photography of our experimental set-up and Figure 5.4 for a 3D representation with an example of planned trajectory.

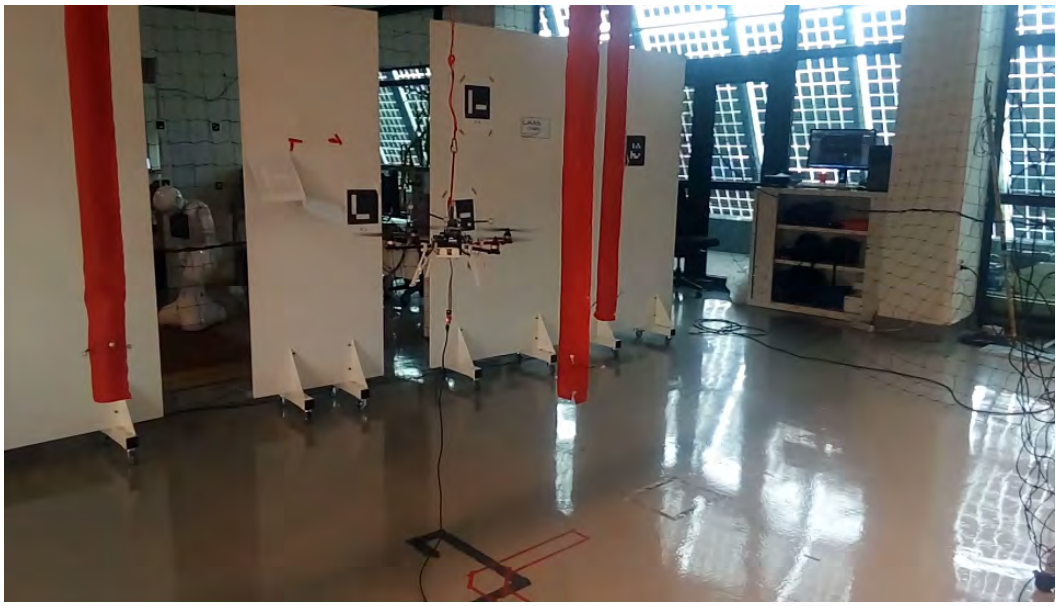


Figure 5.3: Our experimental set-up: the three cylindrical obstacles equipped with reflective markers and the MikroKopter quadrotor.

5.3.2 Results

In this section we show the results of the experiments for $v_{max} = 1\ m.s^{-1}$ in Figure 5.5 and $v_{max} = 2\ m.s^{-1}$ in Figure 5.6. The X axis is the time in seconds. In the upper part of each figure, the Y axis is the positions in meters. The red color is for the x component an the green color for the y component. Since both z and the yaw angle were kept constant, we did not represent them. The dashed curves are

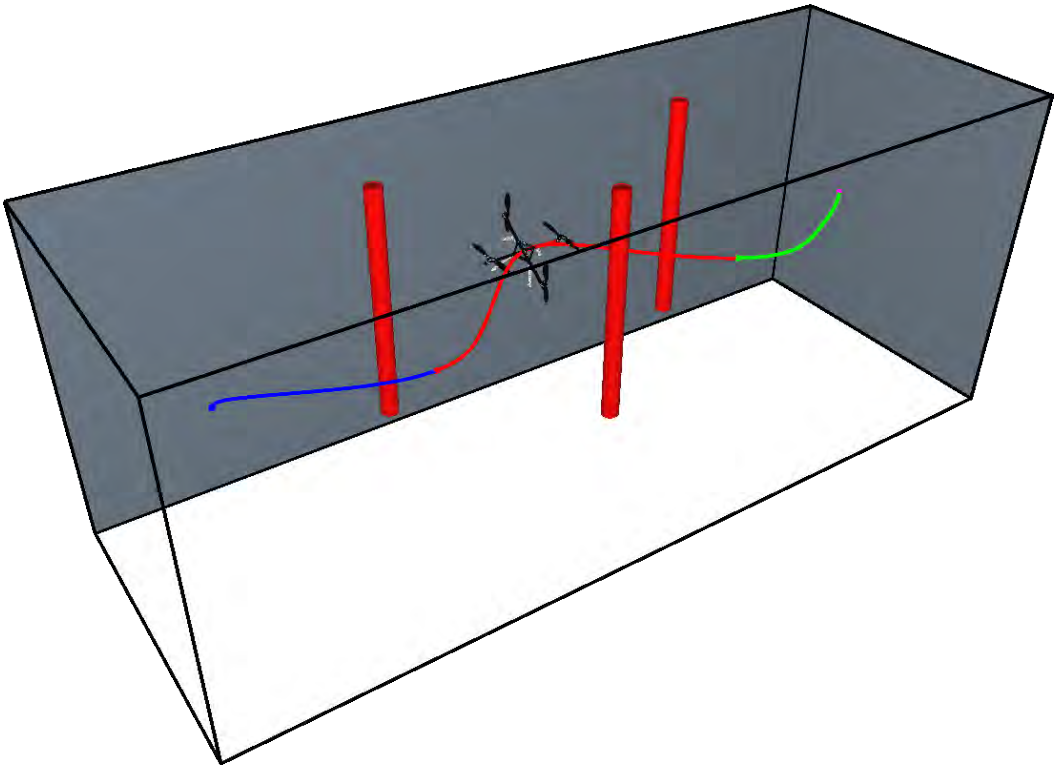


Figure 5.4: 3D representation of our experimental set-up in Move3D with an example of planned trajectory

the nominal quantities $\overline{x}$ and $\overline{y}$ while the plain curves are the measured quantities $\widetilde{x}$ and $\widetilde{y}$. In the lower part of each figure, the Y axis is the errors in meters. They are computed as follows: $e_x = \widetilde{x} - \overline{x}$ and $e_y = \widetilde{y} - \overline{y}$. The red curves are for e_x and the green curves for e_y.

For the case $v_{max} = 1\ m.s^{-1}$ the tracking error in kept under 5 centimeters, and under 10 centimeters for for the case $v_{max} = 2\ m.s^{-1}$.

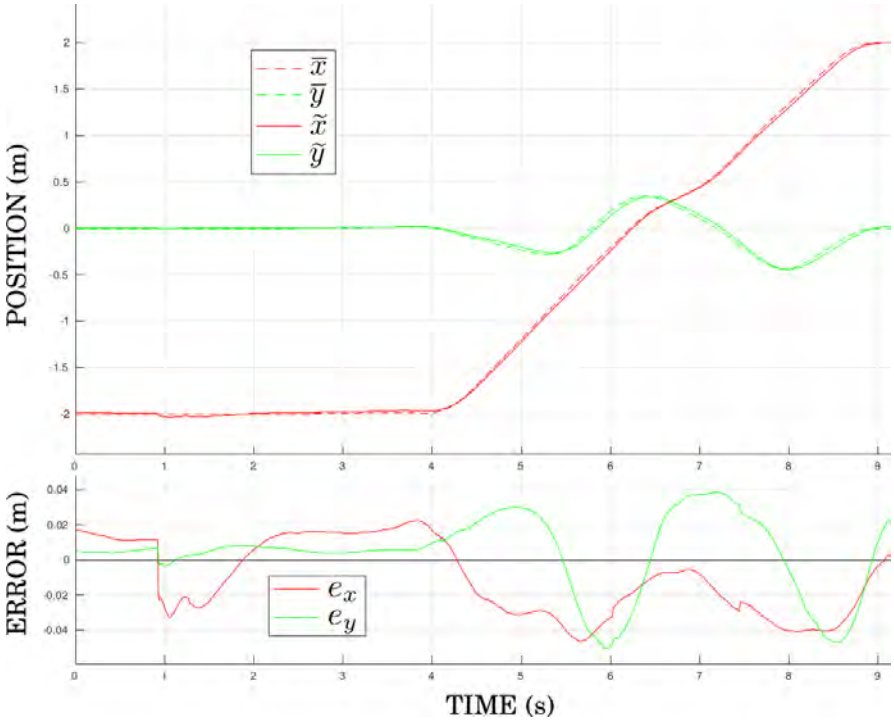


Figure 5.5: Results for $v_{max} = 1\ m.s^{-1}$

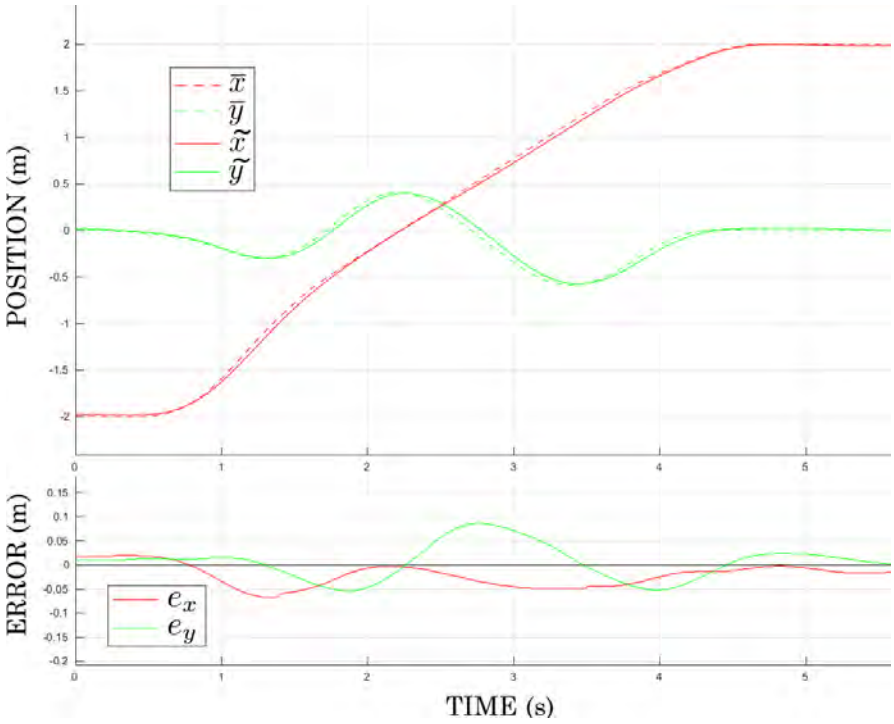


Figure 5.6: Results for $v_{max} = 2\ m.s^{-1}$

Chapter 6

Integration into the ARCAS project

Contents

In this chapter we detail the integration of our works into the ARCAS project in the context of which they were conducted. We first present the project itself, its objectives, its different partners, its subsystems and the general framework in which they are integrated. We then focus on the motion planning system and finally detail the link between symbolic and geometric planning.

6.1 The ARCAS project

For a more detailed presentation of the project consult its website:
`www.arcas-project.eu`

6.1.1 Overview

ARCAS (Aerial Robotics Cooperative Assembly System) was a European funded project conducted between the years 2012 and 2016 that proposed the develop-

(a) Indoor ARM for light loads (b) Outdoor ARM for medium loads (c) Outour ARM for heavy loads

Figure 6.1: The three platforms used in the ARCAS project. Both ARMs from (a) and (b) are quadrotors with eight propellers while the one from (c) is a Flettner helicopter (bi-rotor helicopter).

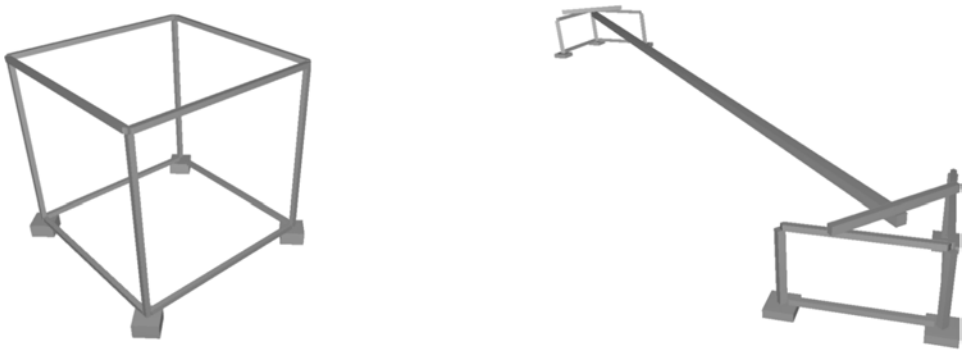
ment and experimental validation of the first cooperative free-flying robot system for assembly and structure construction. It paved the way for a large number of applications including the building of platforms for evacuation of people or landing aircrafts, the inspection and maintenance of facilities and the construction of structures in inaccessible sites and in space.

The ARCAS project had three main axis: structure assembly by a team of ARMs (Aerial Robot with Manipulator), industrial inspection and space manipulation. The latter was a side-part of the project and primarily involved the control community. The industrial inspection consisted in using UAVs either to visually inspect industrial plants or carry a smaller wheeled robot that sticks to pipes for finer inspections. Finally the structure assembly was the part combining work on assembly sequence planning, task planning, motion planning and execution architecture (control, collision avoidance, and so on). The system developed can work with different ARMs, the three platform used are presented in Figure 6.1.

To have simple yet interesting structures to assemble the project focused on structures made of bars. Some examples can be seen in Figure 6.2. The simplicity comes from the clipping mechanism: when two pieces are brought together they clip ensuring a strong link. On the other hand the complexity comes from the need for cooperative transport of certain long bars requiring two (or more) ARMs to strongly cooperate. Moreover the robots are using manipulators which must be compliant to avoid any problems. To carry out the assembly of the structures, the complete system must exhibit a set of properties: multi-robot plans, cooperative transport, perception and localization and visual servoing.

6.1.2 Consortium

This European funded project involved a consortium of eight partners and two third parties:



(a) Cube structure (12 parts)

(b) Long bridge structure (15 parts)

Figure 6.2: Examples of structures from the ARCAS project. In (a) there are 12 parts composing a cube, but there also exist a variant with four additional diagonal bars for a total of 16 parts. In (b) the last part to assemble is the long bar linking the two supports together.

Partners:

- FADA-CATEC (Center for Advanced Aerospace Technologies), Spain, project coordinator.
- DLR (German Aerospace Center), Germany.
- UNINA (Università degli Studi di Napoli Federico II), Italy.
- LAAS-CNRS (Laboratory for Analysis and Architecture of Systems), France.
- USE (Universidad de Sevilla), Spain.
- UPC (Universitat Politècnica de Catalunya), Spain.
- STI (Spacetech GmbH), Germany.
- AIR (ALSTOM Inspection Robotics, GE Inspection Robotics since 2015), Switzerland.

Third parties:

- UNIBAS (Università degli Studi della Basilicat), Italy.
- UNICAS (Università degli Studi di Cassino), Italy.

6.1.3 Objectives

The detailed scientific and technological objectives were:

- New methods for motion control of a free-flying robot with mounted manipulator (or ARM for Aerial Robot with Manipulator) in contact with a grasped object as well as for coordinated control of multiple cooperating ARMs in contact with the same object (e.g. for precise placement or joint manipulation).
- New flying robot perception methods to model, identify and recognize the scenario and to be used for the guidance in the assembly operation, including fast generation of 3D models, aerial 3D SLAM (Simultaneous Location And Mapping), 3D tracking and cooperative perception.
- New methods for the cooperative assembly planning and structure construction by means of multiple flying robots with application to inspection and maintenance activities.
- Strategies for operator assistance, including visual and force feedback, in manipulation tasks involving multiple cooperating flying robots.

6.1.4 The ARCAS system

The project was subdivided into three main subsystems. These are illustrated in a schematic of the hight level architecture of the ARCAS system in Figure 6.3.

The perception system was in charge of all computer vision related tasks in the system. These include SLAM, inspection and task monitoring functions like for instance providing feedback to the ARMs in order to achieve an assembly task by visual servoing.

The modeling and control system was in charge of developing all the needed control methods. These include, for a single ARM, not only trajectory tracking during navigation tasks (including while carrying an object) but also autonomous grasping and insertion of an object into a structure. Not to forget the coordinated control of several ARMs performing those same tasks in cooperation.

Finally the system in which this work has been integrated is the cooperative assembly planning system. Its inputs are:

- The geometry of the (supposed static) environment (a 3D model of it).
- The available resources both in terms of flying robots and assembly parts.
- The "blueprint" and location of the desired structure (*i.e.* the precise location of each part of the final assembly).

Its goal is then to produce a complete plan to assemble the structure.

This subsystem is itself modular since it involves three main modules. The first one, called the *assembly planner*, consists in finding a suitable assembly plan, which

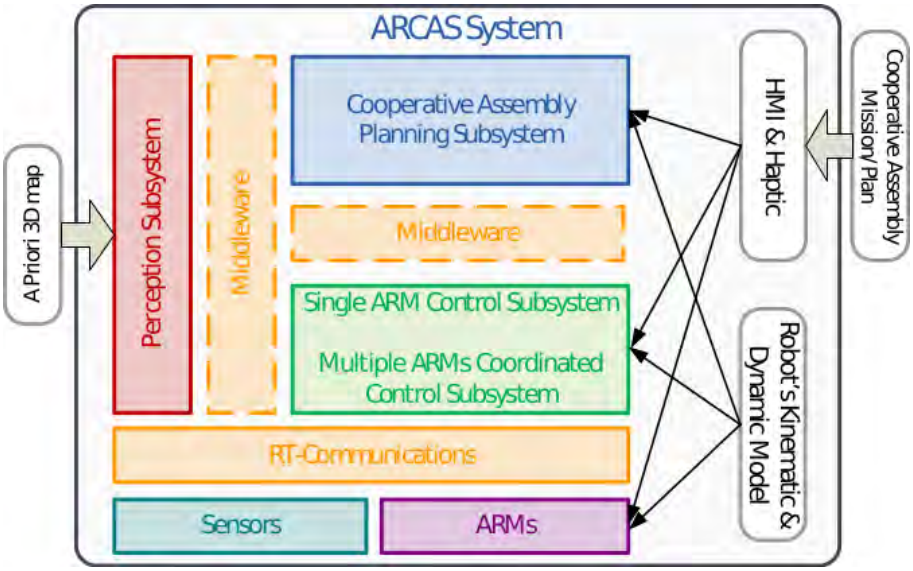


Figure 6.3: Hight level ARCAS architecture

means finding a coherent order in which the parts will be inserted. This has to take into account the structural integrity of the structure during its assembly.

This assembly plan is then given to the *symbolic planner* that is in charge of finding an efficient ordered set of assignments for the ARMs: which ones will perform which tasks an when. This step is often also called *task planning*. The symbolic planner used within the ARCAS system is called MATP (Multi Aerial-robots Task Planner). It is an HTN based planner (see [Ghallab 2004] for more details) developed at LAAS-CNRS.

Each symbolic task computed by it has to be turned into an actual motion plan which usually involves several requests to the motion planner. This is performed by a piece of software developed at LAAS-CNRS and called GTP (Geometric Task Planner). It will be further detailed in section 6.3. GTP is an interface between MATP and the lower level that is the motion planning system described in the next section. The output of the complete planning level is then used by an execution architecture that supervises the execution of the tasks and ensure that there won't be collisions or any other problems.

At the beginning of the project we linked the three planners in a linear workflow: the output of a planner was the input of the next. Later, when the integration between HATP and GTP was ready we used a combined planner as illustrated in Figure 6.4. It allows to test the feasibility of a symbolic task during the symbolic planning rather than after a complete plan is produced. This allows to earlier detect when a symbolic valid solution is in fact not feasible because of geometry.

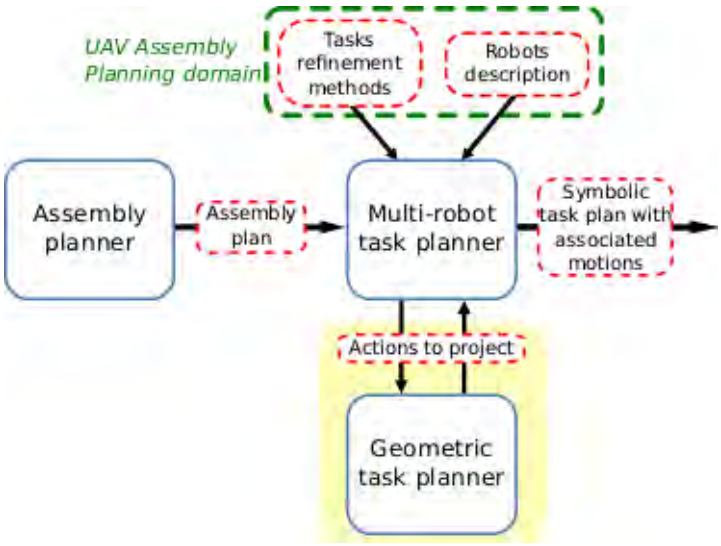


Figure 6.4: The assembly planning system in ARCAS. The assembly plan in given as a dependency tree listing the operation to assemble the structure. From it, HATP produces a symbolic plan and allocate the actions to the agents. During this process GTP is called to assess the feasibility of the actions which it does by calling the motion planner. The computed valid trajectories are kept along with the actions of the symbolic plan.

6.2 The motion planning system

In this section we present the motion planning system which turns the symbolic requests of GTP into a set of motion plans. We focus here on the three basic tasks which are navigation, transportation and coordinated transportation and manipulation. More complicated tasks such as grasping and insertion of an object along with the combination of the twos (*pick and place*) will be treated in the next section.

6.2.1 Navigation

This tasks consists, for a single ARM without payload, in navigating among a set of static obstacles which positions and orientations are known. Both initial and final states are here chosen to be at hovering. For simplification purposes we do not plan for the motion of the manipulator and set its configuration to be fixed along the trajectory. Because in this case the smallest bounding sphere is judged to be a good enough approximation of the geometrical shape of the system, this motion panning problem can be solved very efficiently using the decoupled approach presented in section 4.1. Figure 6.5 illustrates a result of this approach on a simple navigation problem for one quadrotor with no payload.

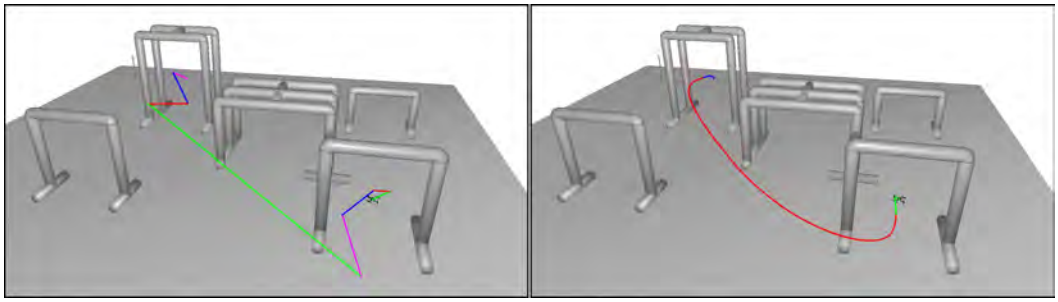


Figure 6.5: A result of the decoupled approach for one quadrotor with no payload. *Left*: geometric path obtained in the first stage. *Right*: smoothed trajectory.

6.2.2 Transportation for a single ARM

The task is here nearly the same. The ARM still has to navigate among known obstacles, both initial and final states are at hovering and the configuration of the manipulator remains fixed during the motion. But the ARM is now grasping an object, meaning that the geometry of the system has changed. The smallest bounding sphere might not be a good enough approximation of the shape of the system anymore and thus a decoupled approach might fail to find a geometric path to begin with. A direct approach to kinodynamic motion planning such as one of the twos presented in section 4.2.3 is therefore used here.

6.2.3 Coordinated transportation and manipulation

Once again the setup is almost identical. The only difference here is that not only one ARM is grasping an object but two or more are grasping the same object. We chose to treat this case with a centralized quasi-static approach. *Quasi-static* means that we do not compute trajectories but geometric paths that do not contain any information about dynamics. As for *centralized*, it means that we do not plan independently for each ARM but do consider the set of ARMs plus the object as a unique system, *i.e.* a single kinematic chain. Since dynamics are not involved here we can also plan for the motion of the manipulators for manipulation tasks.

Manipulating an object with several robots imposes important constraints that have to be treated at the motion planning level. More precisely, several robots that simultaneously grasp an object form a closed kinematic chain, whose motion is restricted to a sub-manifold of the configuration-space of the unconstrained multi-robot system. This sub-manifold cannot be represented and parameterized in the same way as the configuration-space of a serial (open-chain) robots. Therefore, motion planning for closed-chain mechanisms is a difficult problem that requires specific methods.

The method applied here builds on previous work carried out at LAAS-CNRS

by [Cortés 2004] on the extension of sampling-based motion planning algorithms for closed-chain mechanisms. The basic principle is to separate configuration variables into two sets: independent variables and dependent variables. The planner directly acts on the independent variables, while the values of dependent variables are obtained by solving loop-closure equations. A suitable decomposition of the mobile system into a set of kinematic chains involving independent or dependent variables is essential for a good performance of the planner. From now on, kinematic chains involving independent parameters will be called *active subchains* and those involving dependent parameters will be called *passive subchains*, since they follow the motions computed for the active ones. Each passive subchain should correspond with a non-redundant mechanism whose end-frame can span full-rank subsets of the workspace. In general, this requires three joint variables for a planar mechanism and six for a spatial mechanism. Closed-form inverse kinematics methods are usually available for such mechanisms.

Once the system decomposition is defined, configurations of complex closed-chain mechanisms can be sampled using an algorithm called RLG [Cortes 2002]. RLG performs a geometrically-guided random sampling for the active subchains that notably increases the probability of obtaining real solutions for passive subchains when solving the loop-closure equations. Local paths connecting samples are computed by interpolating the configuration of active subchains and solving inverse kinematics for the passive subchains at each intermediate point along the path. Following this approach, sampling-based planners, such as PRM or RRT, see section 1.2, can be extended to deal with closed-chain mechanisms.

An example of decomposition into active/passive subchains is illustrated in Figure 6.6 for a system composed of two quadrotors equipped with planar 3R manipulators cooperatively transporting a bar. The bar is considered as a free-flying object, with 6 degrees of freedom. The relative location of each quadrotor with respect to the bar is defined by two variables $\{x, z\}$ that define the position of the center of mass on a plane perpendicular to the bar, and one rotation R_y around an axis parallel to the bar. Note that only these 3 relative degrees of freedom are possible because of the motion constraints imposed by the planar manipulator. Then, the spatial configuration of each arm is defined from a reference frame attached to each quadrotor and the values of the joint variables. Since the bar is grasped by the arms, kinematic loop-closure constraint have to be imposed to the system. For motion planning using the aforementioned approach, the independent variables are those defining the absolute location of the bar and the relative location of the quadrotors with respect to the bar, whereas the dependent variables are the degrees of freedom of the arms.

A similar decomposition approach can be applied to other types of flying robots equipped with other types of arms. Figure 6.7 shows an example involving two helicopters with 7-DoF Kuka LWR arms. Extracting the object from the hole is a geometrically-constrained problem that requires simultaneous translation and

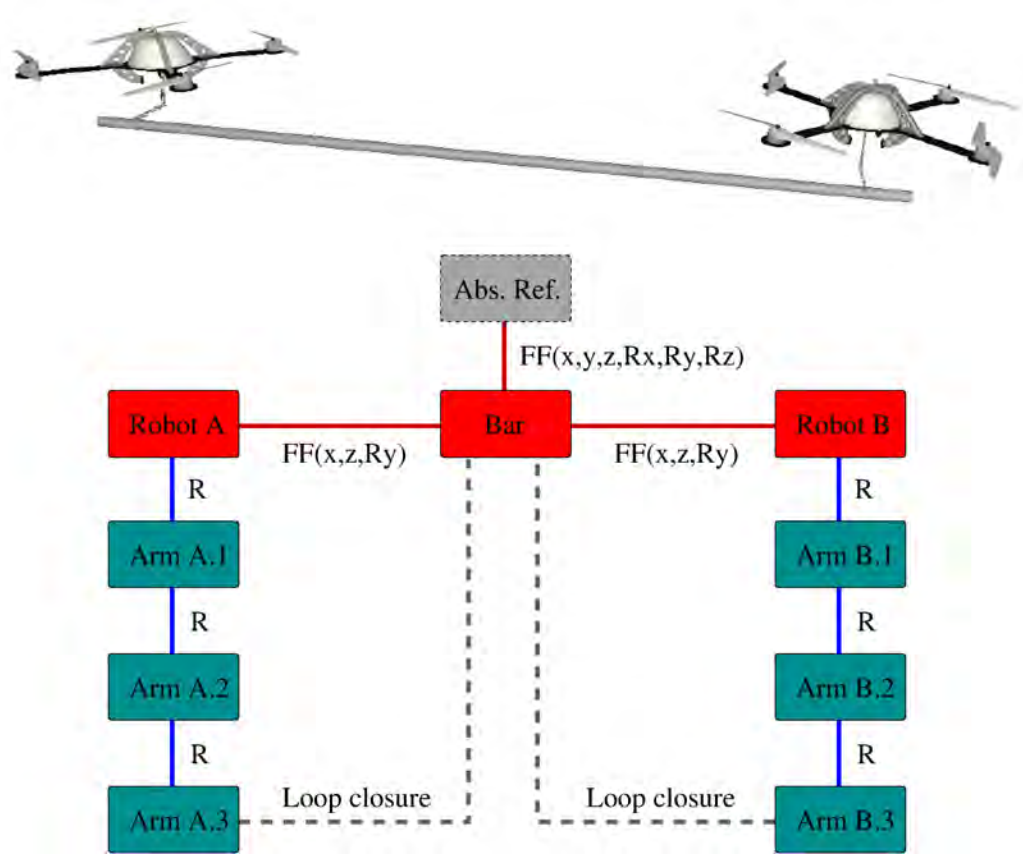


Figure 6.6: Illustration of the decomposition into active/passive subchains for a system composed of two quadrotors with planar 3R manipulators transporting a bar. *Top*: Image of the system. *Bottom*: The corresponding kinematic diagram, with active and passive subchains colored in red and blue, respectively.

rotation. Solving this problem with an RRT-like algorithm extended to closed kinematic chains requires about 3 seconds of CPU time on a single processor in are current implementation.

6.3 Symbolic-Geometric planning to ensure plan feasibility

This section describes the interface between symbolic and geometric planning. The piece of software implementing this interface is called GTP (Geometric Task Planner) and was developed at LAAS-CNRS. We first present an overview together with our motivations. We then describe more specifically its implementation.

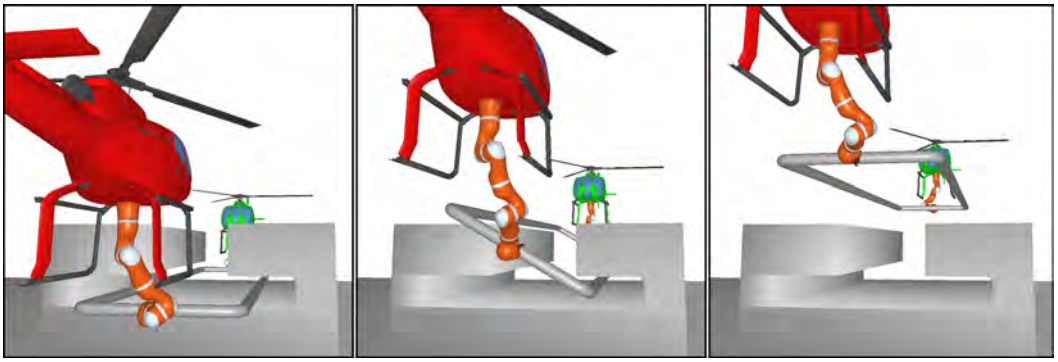


Figure 6.7: Coordinated manipulation using two helicopters with 7-DoF Kuka LWR arms.

6.3.1 Geometric Task Planner

As previously mentioned, MATP produces plans that consist of sequences of symbolic tasks. The problem is that there is no guarantee that these tasks can actually be carried out at the geometric level. For instance a task involving a motion can be geometrically unfeasible because there is no collision free trajectory of the system what implements it. In order to ensure feasibility of produced plans, each task has to be tested for geometric feasibility. This is the main role of GTP.

A motion planning problem consists in finding in a given environment a collision free path between two given configurations of a robotic system. Hence for a motion planner the symbolic representation bears no meaning. For instance the task described by the sentence, *go pick this object and bring it over there*, does not fit the description of a motion planning problem. This is actually a sequence of three motion planning problems: *go*, *pick* and *bring* for configurations to be determined. When given such a task GTP is in charge of doing this decomposition by first finding which are these configurations and then calling the motion planner in order to solve the different motion planning problems which solutions are trajectories of the system.

Geometry of the environment is obviously time dependent since carrying out a task modifies it. GTP is also in charge of maintaining this geometric knowledge over time. We call *world context* a set of robots configurations, grasping informations and positions of objects and obstacles. MATP is actually providing a task to be carried out together with its parent task (i.e. the task that directly precedes it). This way GTP is able to associate to each task its initial and final world contexts, the initial world context of a task being the final world context of its parent task. This means that GTP is maintaining the geometric equivalent of the symbolic data structure that MATP builds.

6.3.2 Tasks: decomposition and implementation

Given a world context, GTP has to decompose a task into a sequence of motion planning problems. This decomposition depends on the nature of the task and hence have to be provided by the user the same way the symbolic domain (*i.e.* the set of possible symbolic tasks) is. In this subsection we provide an example of decomposition for a symbolic task called *pick*, which consists for a given robot in picking up a given object. Two other tasks called *place* and *pickAndPlace* are then described. The advantage of using the task *pickAndPlace* instead of the tasks *pick* then *place* is then discussed.

We decompose the symbolic task *pick* into four motion planning problems, as illustrated in Figure 6.8. From its current hovering configuration $Q1$ the system (without payload) has first to reach a safe hovering configuration $Q2$ in the proximity of the object to pick. The manipulator configurations are the same for both $Q1$ and $Q2$. This is exactly the navigation problem described in section 6.2.1. A decoupled approach is therefore used. From $Q2$ the system has to reach an approach configuration $Q3$ that prepares it for the grasping motion. It involves modifying both the configuration of the manipulator and the pose of the UAV. We use here a classical geometric approach corresponding to a quasi-static motion. From $Q3$ the system has to reach the grasping configuration $Q4$ and then extract the object in order to reach an extraction configuration $Q5$. These two motions are actually controlled by visual servoing during execution, but the planner has to verify feasibility by ensuring that collision free trajectories can be found. This is why here we simply use linear interpolations.

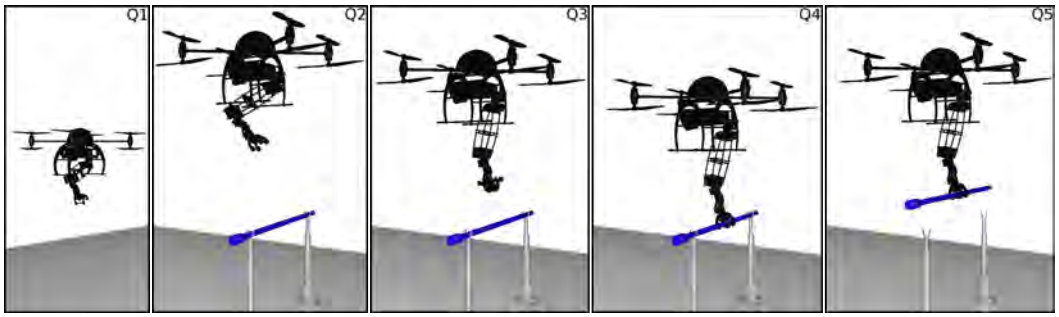


Figure 6.8: The symbolic task *pick* decomposed into four motion planning problems: a navigation problem $Q_1 \rightarrow Q_2$, a quasi-static approach $Q_2 \rightarrow Q_3$, a grasping motion $Q_3 \rightarrow Q_4$ and an extraction motion $Q_4 \rightarrow Q_5$.

Initial configuration excepted, all intermediate configurations are computed according to the grasping configuration $Q4$. Hence this configuration is the first computed by GTP by way of random sampling. If the sampled grasping configuration $Q4$ is collision free then both $Q3$ and $Q5$ and the corresponding motions are computed and tested for collisions. In case of success, $Q2$ is computed and tested, along with the motion $Q_2 \rightarrow Q_3$. Finally the motion $Q_1 \rightarrow Q_2$ is generated. In

case of failure of any of these steps, a new grasping configuration $Q4$ is generated. The planner considers the task as not being feasible after a predefined number of unsuccessful attempts (provided by the user) has been reached.

The decomposition of the task *place*, which consists, for an ARM carrying an object, in inserting it in a given slot, is very similar to the decomposition of the task *pick*. The main configuration (corresponding to $Q4$ in the previous description) to be sampled here is the insertion configuration. Other configurations corresponding to $Q2$, $Q3$ and $Q5$ are generated according to it. The main difference is that the navigation problem can not be solved for a bounding sphere because of the payload. We use here a direct kinodynamic approach, as discussed in section 6.2.2.

The task *pickAndPlace* is a combination of the tasks *pick* and *place*. A first approach for MATP was first to call *pick* then *place*. Problem is that the grasping configuration generated for the picking task may lead to the non feasibility of the placing task and thus to a loss of efficiency. We proposed to treat this problem by using a different approach. We have implemented a specific decomposition of the task *pickAndPlace*. The idea is to generate the insertion configuration in function of a sampled grasping configuration. Motion problems are then solved in the same order as for the *pick* and *place* tasks, the two navigation problems being solved last. Experimental results for several runs on various scenarios have shown that this approach leads to an improvement of about 30% of the average CPU time needed to solve the problem.

Conclusion

We presented the quadrotor system: its dynamics model, its inputs and its outputs. We defined its physical constraints and saw how they translate into the input space. We showed that the system is differentially flat and therefore that any smooth enough trajectory in the space of the flat outputs can be executed provided that the derivatives are correctly bounded. We however did not explicitly defined those bounds.

We formulated the kinodynamic motion planning problem and saw that in order to solve it, the state space has to be explored in place of the configuration space. We saw that this can be done by sampling inputs and forwarding the dynamics rather than sampling states directly. This is however not the kind of techniques we chose to study here. Instead, we tried to see how the classic methods used in sampling-based motion planning have to be adapted when sampling the state space.

First, we proposed a local planning method able to interpolate any two states defined by position, velocity and acceleration. It generates a fourth-order spline that respects a given set of bounds on its derivatives up to snap. We discussed the quality of such solutions and saw that in our test run the mean sub-optimality is of 6.85% but that it also comes with a decreased running time of three orders of magnitude when compared to a SQP algorithm. Because it quickly produces smooth enough trajectories which derivatives can be bounded at will, when applied to the flat outputs of a quadrotor this method can be used as a steering method for this system.

We shown that this local planner can be used in a decoupled approach to solving the kinodynamic motion planning for a quadrotor. A geometric path is first generated for the quadrotor's bounding sphere. The path is then transformed into a trajectory by using the local planner. The trajectory is finally locally optimized with a random short cut algorithm that also uses the local planner. This method is very efficient but will not solve more constrained problems for which there is no quasi-static solution, which is the reason why we next studied the direct approaches.

In that perspective, we first discussed the problem of the metric in the state space. We stated that the ideal metric is the cost-to-go, *i.e.* the duration of the minimum time trajectory. Because computing it is as hard and thus as costly as computing the minimum time trajectory itself, many approaches to kinodynamic motion planning tend to avoid this problem by adapting the algorithms so that they do not rely on the metric to guide the search. We proposed instead to use a metric that correctly estimates the cost-to-go but that do not require as much computing time. We provided empirical evidence that the two metrics are equivalent.

We then addressed the issue of the sampling strategy. We noticed that, in a PRM like algorithm for instance, more than 90% of the attempted connections were failing. When investigating into the reasons why, we discovered that the culprit was the sampling strategy. It is indeed the case that uniform sampling tends to generate states that can be neither the starting point nor the end point of any valid trajectory. We qualified those states as *non-connectible*. We then proposed an incremental sampling strategy which goal is to decrease the probability of generating a non-connectible state.

Finally, two well known methods that are the Bi-RRT algorithm and the classic PRM were adapted for a directed graph. Using them, we studied the influence of the proposed metric and proposed sampling strategy on two different motion planning problems. Results show a significant improvement of the performance of both algorithms thanks to the integrations of the proposed techniques.

In order to show that the planned trajectories can actually be executed, we set up some experimentation. We first described our test-bed, both in terms of hardware and software. We then presented the experimental set-up and some results.

We devoted the last part of this thesis to the presentation of the ARCAS project in the context of which these works were conducted. We first gave an overview of the project itself: its objectives, its different partners, its subsystems and the general framework in which they are integrated. We then focused on the motion planning system. We explained in particular how each different type of motion planning problem, such as navigation for a single UAV with payload, without payload or co-ordinated transportation of a payload by two or more UAVs, is solved by a different method. We finally detailed the link between symbolic and geometric planning by for instance giving an example of task decomposition.

Limitations and future works

In this thesis, we have assumed that any smooth-enough trajectory in the space of the flat outputs can be executed provided that the derivatives are correctly bounded. A major limitation of this approach is that it is not straightforward to actually compute these bounds. To our knowledge, this is still an open problem. A possible strategy inspired by [Mueller 2013] could be to solve the BVP without constraints but with an imposed flight duration. A feasibility test can then be performed to check if the trajectory corresponds to admissible inputs. The idea would be to numerically find the minimum flight duration such that the trajectory is admissible (possibly with a simple search by dichotomy). An other possibility is to try and compute the bounds on the derivatives of the flat outputs either in closed form or numerically.

In this thesis, we only studied the influence of both the proposed metric and sampling strategy on a directed Bi-RRT and a directed PRM. It would be interesting to see how it would influences the performances of other algorithms. It could also be interesting to see how the sub-optimality of the proposed steering method affects the solutions to the RRT* and PRM* algorithm. Secondly, since our focus here was the quadrotor system, we did not apply our approach to problems in higher dimension. A study on how it would perform in this case should definitively be conducted. Moreover it would be interesting to check if as expected the computing time of the steering method linearly increases with dimension. Finally we did not compare our approach to the methods with forward propagation of the dynamics. A bench-marking of the state of art including our proposal remains to be performed.

Concerning our proposed approach to solving the BVP, it was recently brought to our attention that similar works were conducted in the context of online trajectory generation both for reactive behaviours to unforeseen events ([Kröger 2010, Kröger 2011]) and from a control perspective ([Bianco 2014, Bianco 2011, Bianco 2003]). For lack of time we unfortunately could not provide a review of these works and therefore position our own propositions against those. This will have to done in future works.

Our sampling strategy takes into account the bounding box of the workspace and the velocity constraints. It could be interesting to keep the same incremental decoupled scheme and also take the obstacles into account. In other worlds it could be beneficial to try and develop a sampling strategy that avoids generating inevitable collision states as defined by [Fraichard 2004]. It is however not straightforward to see how the decoupled scheme could adapt to obstacles of generic shape, *i.e.* that are not axis-oriented rectangular cuboids.

Finally, our experimental chapter is quite modest and the tracking errors are still rather large at high velocity. This is mainly due to the fact that our test-bed is relatively recent and not yet fully mature. We are planning to improve this in the very near future and hopefully be then able to execute more agile maneuvers.

Appendix A

Calculation details of the steering method

In this appendix we provide calculation details on certain aspects of the steering method presented in Chapter 3.

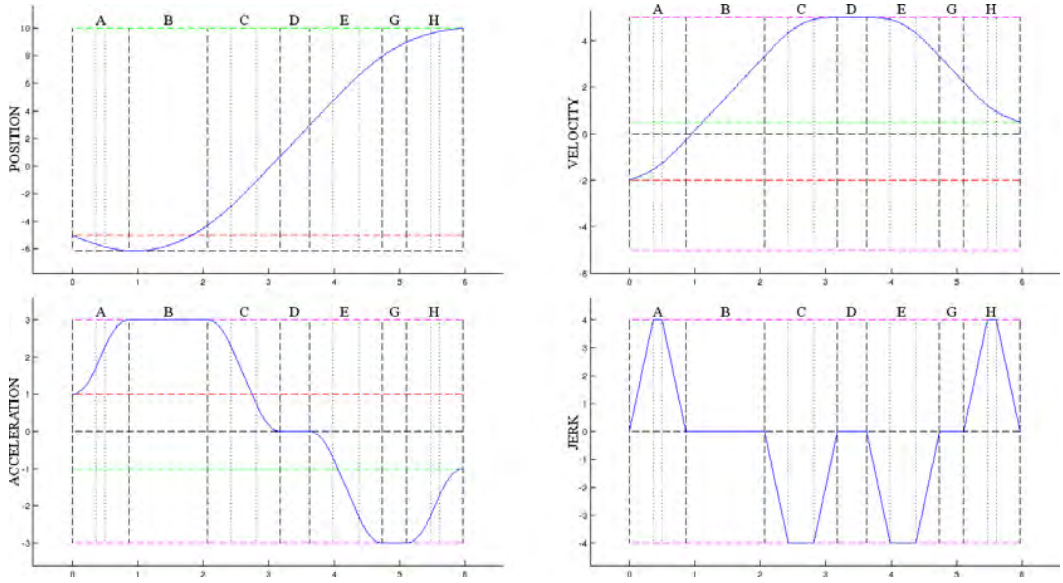


Figure A.1: Example of trajectory provided by the steering method for one degree of freedom. The bounds of each derivative are represented by pink dashed lines. Red and green dashed lines represent initial and final values, respectively. Note that, in order to show all the phases, $|v_D| = v_{max}$ and $|a_B| = |a_D| = a_{max}$ in this example. In a general case, some phases can have zero duration.

A.1 Reminders

In this section we would like to recall the goal of our steering-method and update certain notations. Because we were specifically treating the quadrotor system in Chapter 3 we considered its flat outputs $\mathbf{z} = [z_i]_{i=1..4}^T$. Let us consider here a more general configuration $\mathbf{q} = [q_i]_{i=1..n}^T$ and its associated state $\mathbf{x} = [\mathbf{q}\ \dot{\mathbf{q}}\ \ddot{\mathbf{q}}]$. We recall that we solve the BVP independently for each degree of freedom of index i. We can

therefore omit the index notation for the sake of simplicity. For the bounds on the derivatives we note:

$$[v_{max}\ a_{max}\ j_{max}\ s_{max}] := [v_i^{max}\ a_i^{max}\ j_i^{max}\ s_i^{max}]$$

Similarly for the values to be interpolated we note:

$$[x_0\ v_0\ a_0] := \left[q_i^0\ \dot{q}_i^0\ \ddot{q}_i^0\right] \text{ and } [x_F\ v_F\ a_F] := \left[q_i^F\ \dot{q}_i^F\ \ddot{q}_i^F\right]$$

We solve for each degree of freedom of index i:

$$\begin{cases} \min T_i \in \mathbb{R}^+ \text{ s.t.} \\ S_i \in \mathcal{C}^3([0\ T_i], \mathbb{R}) \\ [S_i(0)\ \dot{S}_i(0)\ \ddot{S}_i(0)\ \dddot{S}_i(0)] = [x_0\ v_0\ a_0\ 0] \\ [S_i(T_i)\ \dot{S}_i(T_i)\ \ddot{S}_i(T_i)\ \dddot{S}_i(T_i)] = [x_F\ v_F\ a_F\ 0] \\ \forall t \in [0, T_i] \begin{cases} |\dot{S}_i(t)| \leq v_{max} \\ |\ddot{S}_i(t)| \leq a_{max} \\ |\dddot{S}_i(t)| \leq j_{max} \\ |\ddddot{S}_i(t)| \leq s_{max} \end{cases} \end{cases}$$

We also recall the notations of the different phases of the proposed solution as it is illustrated in Figure A.1.

A is the phase of the first acceleration variation from $\ddot{S}_i(t) = a_0$ to $\ddot{S}_i(t) = a_B$.
B is the first phase of constant acceleration $\ddot{S}_i(t) = a_B$.
C is the phase of the second acceleration variation from $\ddot{S}_i(t) = a_B$ to $\ddot{S}_i(t) = 0$.
ABC is the phase of the first velocity variation from $\dot{S}_i(t) = v_0$ to $\dot{S}_i(t) = v_D$.

D : the phase of constant velocity $\dot{S}_i(t) = v_D$.

E is the phase of the third acceleration variation from $\ddot{S}_i(t) = 0$ to $\ddot{S}_i(t) = a_G$.
G is the second phase of constant acceleration $\ddot{S}_i(t) = a_G$.
H is the phase of the fourth acceleration variation from $\ddot{S}_i(t) = a_G$ to $\ddot{S}_i(t) = a_F$.
EHG is the phase of the second velocity variation from $\dot{S}_i(t) = v_D$ to $\dot{S}_i(t) = v_F$.

Each phase noted X has a duration noted t_X
Therefore $T_i = t_A + t_B + t_C + t_D + t_E + t_G + t_H$

Each phase noted X of acceleration variation ($X \in \{A, C, E, H\}$) is divided in three sub-phases:
X_1 is the phase of the first jerk variation from $\dddot{S}_i(t) = 0$ to $\dddot{S}_i(t) = j_X$
X_2 is the phase of constant saturated jerk $\dddot{S}_i(t) = j_X$
X_3 is the phase of the second jerk variation from $\dddot{S}_i(t) = j_X$ to $\dddot{S}_i(t) = 0$

The phases X_1 and X_3 have the same duration $t_{X,1}$
The phase X_2 has a duration of $t_{X,2}$
Therefore $t_X = 2.t_{X,1} + t_{X,2}$.

Let $s_A \in \{-1, 0, 1\}$ be the sign of the snap during phase A_1.
Let $s_C \in \{-1, 0, 1\}$ be the sign of the snap during phase C_1.
Let $s_E \in \{-1, 0, 1\}$ be the sign of the snap during phase E_1.
Let $s_H \in \{-1, 0, 1\}$ be the sign of the snap during phase H_1.

A.2 Expression of the spline

Using all these notations we can now give the expression of the solution spline S_i in the form of Algorithm 4. For a given $t \in [0, T_i]$, the return value is $\left[S_i(t),\ \dot{S}_i(t),\ \ddot{S}_i(t),\ \dddot{S}_i(t),\ \ddddot{S}_i(t)\right]$. Let be $(t_{cur},\ x_{cur},\ v_{cur},\ a_{cur},\ j_{cur})$ a set of variables we will use to store at the end of each phase the values of the previous one. The tabs $\mathcal{T}_S$ and $\mathcal{T}_D$ respectively contains the values of the snaps and durations of all fifteen phases. They are the result of a previous computation that we detail later.

A.3 Durations of the phases

In this section we detail the expression of the durations of all phases.

Let us start with $t_{A,1}$.
We recall that j_A is the value of the jerk during phase A_2.
Using the expression of the spline we have: $j_A = s_A.s_{max}.t_{A,1}$
As $|j_A| = s_{max}.t_{A,1} \leq j_{max}$, we have:

$$t_{A,1} \leq \frac{j_{max}}{s_{max}} \quad \left(\iff s_{max}.t_{A,1}^2 \leq \frac{j_{max}^2}{s_{max}} \right)$$

We also have:

$$a_B = s_A.s_{max}.t_{A,1}^2 + s_A.s_{max}.t_{A,1}.t_{A,2} + a_0$$

Phase A_2 is only needed when phases A_1 and A_3 are not enough to reach a_B. If we do not have a phase A_2, i.e. if $t_{A,2} = 0$, then:

$$a_B = s_A.s_{max}.t_{A,1}^2 + a_0 \quad \left(\iff t_{A,1} = \sqrt{\frac{|a_B - a_0|}{s_{max}}} \right)$$

Algorithm 4: Expression of the spline

if $t < 0$ ***or*** $t > T_i$ **then**
  return error code
end

$t_{cur} \leftarrow 0$
$x_{cur} \leftarrow x_0$
$v_{cur} \leftarrow v_0$
$a_{cur} \leftarrow a_0$
$j_{cur} \leftarrow 0$
$\mathcal{T}_S \leftarrow s_{max}.\,[s_A,\ 0,\ -s_A,\ 0,\ s_C,\ 0,\ -s_C,\ 0,\ s_E,\ 0,\ -s_E,\ 0,\ s_H,\ 0,\ -s_H]$
$\mathcal{T}_D \leftarrow$
$[t_{A,1},\ t_{A,2},\ t_{A,1},\ t_B,\ t_{C,1},\ t_{C,2},\ t_{C,1},\ t_D,\ t_{E,1},\ t_{E,2},\ t_{E,1},\ t_G,\ t_{H,1},\ t_{H,2},\ t_{H,1}]$

for $k \in [\![1,\ 15]\!]$ **do**
  if $t_{cur} \ \leq\ t \ \leq\ t_{cur} + \mathcal{T}_D[k]$ **then**
    $t_{rel} \leftarrow t - t_{cur}$
    $j \leftarrow \mathcal{T}_S[k].t_{rel} + j_{cur}$
    $a \leftarrow \dfrac{\mathcal{T}_S[k]}{2}.t_{rel}^2 + j_{cur}.t_{rel} + a_{cur}$
    $v \leftarrow \dfrac{\mathcal{T}_S[k]}{6}.t_{rel}^3 + \dfrac{j_{cur}}{2}.t_{rel}^2 + a_{cur}.t_{rel} + v_{cur}$
    $x \leftarrow \dfrac{\mathcal{T}_S[k]}{24}.t_{rel}^4 + \dfrac{j_{cur}}{6}j_{cur}.t_{rel}^3 + \dfrac{a_{cur}}{2}.t_{rel}^2 + v_{cur}.t_{rel} + x_{cur}$
    return $[x,\ v,\ a,\ j,\ \mathcal{T}_S[k]]$
  end
  $t_{cur} \leftarrow t_{cur} + \mathcal{T}_D[k]$
  $x_{cur} \leftarrow \dfrac{\mathcal{T}_S[k]}{24}.\mathcal{T}_D[k]^4 + \dfrac{j_{cur}}{6}.\mathcal{T}_D[k]^3 + \dfrac{a_{cur}}{2}.\mathcal{T}_D[k]^2 + v_{cur}.\mathcal{T}_D[k] + x_{cur}$
  $v_{cur} \leftarrow \dfrac{\mathcal{T}_S[k]}{6}.\mathcal{T}_D[k]^3 + \dfrac{j_{cur}}{2}.\mathcal{T}_D[k]^2 + a_{cur}.\mathcal{T}_D[k] + v_{cur}$
  $a_{cur} \leftarrow \dfrac{\mathcal{T}_S[k]}{2}.\mathcal{T}_D[k]^2 + j_{cur}.\mathcal{T}_D[k] + a_{cur}$
  $j_{cur} \leftarrow \mathcal{T}_S[k].\mathcal{T}_D[k] + j_{cur}$
end

return error code

Then we can see that:

$$\begin{cases} s_A = 1 \implies s_A.s_{max}.t_{A,1}^2 \leq \dfrac{j_{max}^2}{s_{max}} \implies a_B - a_0 = |a_B - a_0| \leq \dfrac{j_{max}^2}{s_{max}} \\ \\ s_A = -1 \implies s_A.s_{max}.t_{A,1}^2 \geq -\dfrac{j_{max}^2}{s_{max}} \implies a_B - a_0 = -|a_B - a_0| \geq -\dfrac{j_{max}^2}{s_{max}} \end{cases}$$

So we have :

$$|a_B - a_0| > \frac{j_{max}^2}{s_{max}} \implies t_{A,2} \neq 0$$

In this case we have :

$$\begin{aligned} t_{A,1} = \frac{j_{max}}{s_{max}} \quad &\implies \quad a_B = \frac{s_A.j_{max}^2}{s_{max}} + s_A.j_{max}.t_{A,2} + a_0 \\ \\ &\implies \quad t_{A,2} = \frac{|a_B - a_0|}{j_{max}} - \frac{j_{max}}{s_{max}} \end{aligned}$$

In brief :

$$\textbf{if } |a_B - a_0| > \frac{j_{max}^2}{s_{max}} \textbf{ then } \begin{cases} t_{A,1} = \dfrac{j_{max}}{s_{max}} \\ \\ t_{A,2} = \dfrac{|a_B - a_0|}{j_{max}} - \dfrac{j_{max}}{s_{max}} \end{cases} \textbf{ else } \begin{cases} t_{A,1} = \sqrt{\dfrac{|a_B - a_0|}{s_{max}}} \\ \\ t_{A,2} = 0 \end{cases}$$

Using the same line of reasoning for phases C, E and H we get :

$$\textbf{if } |a_B| > \frac{j_{max}^2}{s_{max}} \textbf{ then } \begin{cases} t_{C,1} = \dfrac{j_{max}}{s_{max}} \\ \\ t_{C,2} = \dfrac{|a_B|}{j_{max}} - \dfrac{j_{max}}{s_{max}} \end{cases} \textbf{ else } \begin{cases} t_{C,1} = \sqrt{\dfrac{|a_B|}{s_{max}}} \\ \\ t_{C,2} = 0 \end{cases}$$

$$\textbf{if } |a_G| > \frac{j_{max}^2}{s_{max}} \textbf{ then } \begin{cases} t_{E,1} = \dfrac{j_{max}}{s_{max}} \\ \\ t_{E,2} = \dfrac{|a_G|}{j_{max}} - \dfrac{j_{max}}{s_{max}} \end{cases} \textbf{ else } \begin{cases} t_{E,1} = \sqrt{\dfrac{|a_G|}{s_{max}}} \\ \\ t_{E,2} = 0 \end{cases}$$

$$\textbf{if } |a_G - a_F| > \frac{j_{max}^2}{s_{max}} \textbf{ then } \begin{cases} t_{H,1} = \dfrac{j_{max}}{s_{max}} \\ \\ t_{H,2} = \dfrac{|a_G - a_F|}{j_{max}} - \dfrac{j_{max}}{s_{max}} \end{cases} \textbf{ else } \begin{cases} t_{H,1} = \sqrt{\dfrac{|a_G - a_0|}{s_{max}}} \\ \\ t_{H,2} = 0 \end{cases}$$

We now have to compute t_B, t_D and t_G. Let us start by giving the expression of v_D. Using the expression of the spline we have:

$$\begin{aligned} v_D &= s_A.s_{max}.t_{A,1}^3 + \frac{3.s_A.s_{max}.t_{A,1}^2.t_{A,2}}{2} + 2.s_A.s_{max}.t_{A,1}^2.t_{C,1} \\ &+ s_A.s_{max}.t_{A,1}^2.t_{C,2} + s_A.s_{max}.t_B.t_{A,1}^2 + \frac{s_A.s_{max}.t_{A,1}.t_{A,2}^2}{2} \\ &+ 2.s_A.s_{max}.t_{A,1}.t_{A,2}.t_{C,1} + s_A.s_{max}.t_{A,1}.t_{A,2}.t_{C,2} + s_A.s_{max}.t_B.t_{A,1}.t_{A,2} \\ &+ 2.a_0.t_{A,1} + a_0.t_{A,2} + s_C.s_{max}.t_{C,1}^3 + \frac{3.s_C.s_{max}.t_{C,1}^2.t_{C,2}}{2} \\ &+ \frac{s_C.s_{max}.t_{C,1}.t_{C,2}^2}{2} + 2.a_0.t_{C,1} + a_0.t_{C,2} + v_0 + a_0.t_B \end{aligned}$$

Phase B is only necessary if phases A and C are not enough to reach v_D.
If $t_B = 0$ then:

$$\begin{aligned} v_D &= s_A.s_{max}.t_{A,1}^3 + \frac{3.s_A.s_{max}.t_{A,1}^2.t_{A,2}}{2} + 2.s_A.s_{max}.t_{A,1}^2.t_{C,1} \\ &+ s_A.s_{max}.t_{A,1}^2.t_{C,2} + \frac{s_A.s_{max}.t_{A,1}.t_{A,2}^2}{2} + 2.s_A.s_{max}.t_{A,1}.t_{A,2}.t_{C,1} \\ &+ s_A.s_{max}.t_{A,1}.t_{A,2}.t_{C,2} + 2.a_0.t_{A,1} + a_0.t_{A,2} + s_C.s_{max}.t_{C,1}^3 \\ &+ \frac{3.s_C.s_{max}.t_{C,1}^2.t_{C,2}}{2} + \frac{s_C.s_{max}.t_{C,1}.t_{C,2}^2}{2} + 2.a_0.t_{C,1} + a_0.t_{C,2} + v_0 \end{aligned}$$

We can see from this expression that v_D depends from s_A, s_C, $t_{A,1}$, $t_{A,2}$, $t_{C,1}$ and $t_{C,2}$ *i.e.* from a_B (see expressions above). We have different cases according to the value of a_B.
Let $cond_A$ and $cond_C$ be:

$$cond_A = \begin{cases} 1 \text{ if } |a_B - a_0| > \dfrac{j_{max}^2}{s_{max}} \\ 0 \text{ otherwise} \end{cases} \quad \text{and } cond_C = \begin{cases} 1 \text{ if } |a_B| > \dfrac{j_{max}^2}{s_{max}} \\ 0 \text{ otherwise} \end{cases}$$

Let $case_{AC}$ be a variable that will represent all possible cases according to the value of a_B:

$$case_{AC} = 1000.(s_A + 1) + 100.(s_C + 1) + 10.cond_A + cond_C$$

Even thought there are in theory $3 \times 3 \times 2 \times 2 = 36$ possible cases, in reality we will only use eight of them:

$$case_{AC} \in \{200, 201, 210, 211, 2000, 2001, 2010, 2011\}$$

This comes from the fact that we will not treat the cases where $s_A = 0$ or $s_C = 0$ which eliminates all the cases of the form $1xxx$ and $x1xx$ (20 cases). Those cases corresponds to $a_B = a_0$ and $a_B = 0$, they can be handled in a much simpler way. We will not treat the cases of the form $00xx$ and $22xx$ either (8 cases). Those cases corresponds to $s_A = s_C \neq 0$, which means $|a_B| < |a_0|$. In this case it makes more sens to set $a_B = a_0$ (and then $t_{A,1} = t_{A,2} = 0$).

$$case_{AC} = 200 \implies v_D = v_0 - a_B.\sqrt{-\frac{a_B}{s_{max}}} + (a_0 + a_B).\sqrt{\frac{a_0 - a_B}{s_{max}}}$$

$$case_{AC} = 201 \implies v_D = v_0 - \frac{a_B^2}{2.j_{max}} + (a_0 + a_B).\sqrt{\frac{a_0 - a_B}{s_{max}}} + \frac{a_B.j_{max}}{2.s_{max}}$$

$$case_{AC} = 210 \implies v_D = v_0 + \frac{j_{max}.(a_0 + a_B) - 2.a_B.\sqrt{-s_{max}.a_B}}{2.s_{max}} + \frac{a_0^2 - a_B^2}{2.j_{max}}$$

$$case_{AC} = 211 \implies v_D = v_0 + \frac{a_0^2 - 2.a_B^2}{2.j_{max}} + \frac{j_{max}.(a_0 + 2.a_B)}{2.s_{max}}$$

$$case_{AC} = 2000 \implies v_D = v_0 + \frac{2.(a_0 + a_B).\sqrt{a_B - a_0} + 2.a_B.\sqrt{a_B}}{2.\sqrt{s_{max}}}$$

$$case_{AC} = 2001 \implies v_D = v_0 + \frac{a_B^2}{2.j_{max}} + (a_0 + a_B).\sqrt{\frac{a_B - a_0}{s_{max}}} + \frac{a_B.j_{max}}{2.s_{max}}$$

$$case_{AC} = 2010 \implies v_D = v_0 + \frac{j_{max}.(a_0 + a_B) + 2.a_B.\sqrt{s_{max}.a_B}}{2.s_{max}} - \frac{a_0^2 - a_B^2}{2.j_{max}}$$

$$case_{AC} = 2011 \implies v_D = v_0 + \frac{2.a_B^2 - a_0^2}{2.j_{max}} + \frac{j_{max}.(a_0 + 2.a_B)}{2.s_{max}}$$

We now have a function $V_B : a_B \mapsto V_B(a_B) = v_D$. With the same line of reasoning, and using an expression of the spline coming from an other algorithm where the exploration starts from $\mathbf{q}_F$, we also have the function $V_G : a_G \mapsto V_G(a_G) = v_D$. What we actually want is V_B^{-1} and V_G^{-1}. With those functions the spline is entirely defined by the choice of v_D. Indeed, from v_D comes $a_B = V_B^{-1}(v_D)$ and then s_A, s_C, $t_{A,1}$, $t_{A,2}$, $t_{C,1}$ and $t_{C,2}$. And we will see that the implementation of V_B^{-1} will even give us t_B. In the same way, $a_G = V_G^{-1}(v_D)$ gives s_E, s_H, $t_{E,1}$, $t_{E,2}$, t_G, $t_{H,1}$ and $t_{H,2}$.

The problem here is that given $v_D \in [-v_{max}, v_{max}]$ we can not directly decide in which case we are. We have first to map the different cases (which are in facts different intervals of a_B) to the corresponding intervals of v_D. This is possible because V_B is bijective on $[-a_{max}, 0] \cup [a_0, a_{max}]$ if $a_0 > 0$, (on $[-a_{max}, a_0] \cup [0, a_{max}]$ if $a_0 < 0$). When this is done, each time we need to compute $V_B^{-1}(v_D)$ we find in which interval v_D is and we apply the corresponding expression to get a_B. It is possible that we do not find the good interval because v_D is not reachable with $t_B = 0$. In this case we set a_B to $\pm a_{max}$ according to the situation. We then simply have:

$$t_B = \left| \frac{v_D - V_B(a_B)}{a_B} \right|$$

We now have everything but t_D. Let us note x_C the value of $S(t)$ at the end of phase C and x_E its value at the beginning of phase E. These values depends from v_D and can be calculated using the same set of principles described above. Let us note Δ_x the quantity $x_E - x_C$. We can now give the expression of t_D:

$$t_D = \frac{\Delta_x}{v_D}$$

Note that $t_D > 0 \implies \mathbf{sign}(\Delta x) = \mathbf{sign}(v_D)$. Actually the sign of v_D is given by the sign of Δ_x for $v_D = 0$ (let us not that sign $s_{\Delta 0}$).

The last thing to do now is to find the value of v_D that will minimize T_i. As expected, the highest $|v_D|$ is, the lowest is T_i. We search for the zeros of Δ_x on $[-v_{max},\ 0]$ if $s_{\Delta 0} = -1$, on $[0,\ v_{max}]$ otherwise. This is done numerically with a simple secant method. If there are no zeros on the interval then $v_D = \pm v_{max}$, else v_D is set to the zero with the lowest absolute value. The actual optimal is the zero with the highest absolute value but this is done for synchronization purposes. Let us note v_{opt} the calculated v_D. The profiles of $T_i(v_D)$ and $\Delta_x(v_D)$ can be seen on a example on Figure A.2.

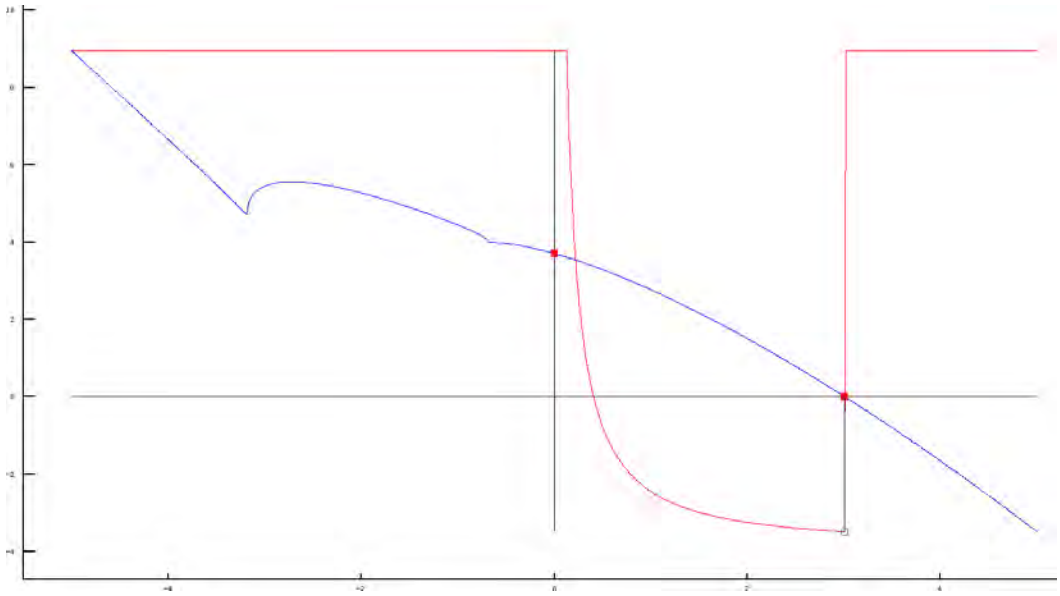


Figure A.2: The profiles of T_i and Δ_x as functions of v_D on a example. On the X axis is v_D. On the Y axis is Δ_x. The blue curve is $\Delta_x(v_D)$. The red curve is $T_i(v_D)$ put to scale. The value of T_i is set to an arbitrary high value when $v_D.\Delta_x(v_D) < 0$. The value of v_{opt} on this example is represented by a the red square on the X axis.

Bibliography

[Allen 2015] Ross Allen et Marco Pavone. *Toward a real-time framework for solving the kinodynamic motion planning problem.* In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 928–934. IEEE, 2015. (Cited in page 23.)

[Amato 1998] Nancy M Amato, O Burchan Bayazit et Lucia K Dale. *OBPRM: An obstacle-based PRM for 3D workspaces.* 1998. (Cited in page 13.)

[Barraquand 1990] Jérôme Barraquand et Jean-Claude Latombe. *A Monte-Carlo algorithm for path planning with many degrees of freedom.* In Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on, pages 1712–1717. IEEE, 1990. (Cited in page 8.)

[Barraquand 1991] Jerome Barraquand et Jean-Claude Latombe. *Robot motion planning: A distributed representation approach.* The International Journal of Robotics Research, vol. 10, no. 6, pages 628–649, 1991. (Cited in pages 8 et 14.)

[Bekris 2007] Kostas E Bekris et Lydia E Kavraki. *Greedy but safe replanning under kinodynamic constraints.* In Proceedings 2007 IEEE International Conference on Robotics and Automation, pages 704–710. IEEE, 2007. (Cited in page 24.)

[Bekris 2008] K Bekris et L Kavraki. *Informed and probabilistically complete search for motion planning under differential constraints.* In First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR), Chicago, IL, 2008. (Cited in page 25.)

[Bertsekas 1995] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas et Dimitri P Bertsekas. Dynamic programming and optimal control, volume 1. Athena Scientific Belmont, MA, 1995. (Cited in pages 47 et 63.)

[Bessiere 1993] Pierre Bessiere, Juan-Manuel Ahuactzin, El-Ghazali Talbi et Emmanuel Mazer. *The "Ariadne's clew" algorithm: global planning with local methods.* In Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on, volume 2, pages 1373–1380. IEEE, 1993. (Cited in page 14.)

[Betts 1998] John T Betts. *Survey of numerical methods for trajectory optimization.* Journal of guidance, control, and dynamics, vol. 21, no. 2, pages 193–207, 1998. (Cited in page 21.)

[Bianco 2003] C Guarino Lo Bianco et Roberto Zanasi. *Smooth profile generation for a tile printing machine.* IEEE Transactions on Industrial Electronics, vol. 50, no. 3, pages 471–477, 2003. (Cited in page 101.)

[Bianco 2011] C Guarino Lo Bianco et Oscar Gerelli. *Online trajectory scaling for manipulators subject to high-order kinematic and dynamic constraints.* IEEE Transactions on Robotics, vol. 27, no. 6, pages 1144–1152, 2011. (Cited in page 101.)

[Bianco 2014] Corrado Guarino Lo Bianco et Fabio Ghilardelli. *A discrete-time filter for the generation of signals with asymmetric and variable bounds on velocity, acceleration, and jerk.* IEEE Transactions on Industrial Electronics, vol. 61, no. 8, pages 4115–4125, 2014. (Cited in page 101.)

[Bobrow 1985] James E Bobrow, Steven Dubowsky et JS Gibson. *Time-optimal control of robotic manipulators along specified paths.* The international journal of robotics research, vol. 4, no. 3, pages 3–17, 1985. (Cited in page 20.)

[Bohlin 2000] Robert Bohlin et Lydia E Kavraki. *Path planning using lazy PRM.* In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, volume 1, pages 521–528. IEEE, 2000. (Cited in page 13.)

[Boor 1999] Valdrie Boor, Mark H Overmars, Van der Stappen et A Frank. *The Gaussian sampling strategy for probabilistic roadmap planners.* In Robotics and automation, 1999. proceedings. 1999 ieee international conference on, volume 2, pages 1018–1023. IEEE, 1999. (Cited in page 13.)

[Brockett 1982] Roger W Brockett. Control theory and singular riemannian geometry. Springer, 1982. (Cited in page 21.)

[Canny 1988a] John Canny. The complexity of robot motion planning. MIT press, 1988. (Cited in page 7.)

[Canny 1988b] John Canny, Bruce Donald, John Reif et Patrick Xavier. On the complexity of kinodynamic planning. IEEE, 1988. (Cited in pages 10, 18 et 21.)

[Canny 1990] John Canny, Ashutosh Rege et John Reif. *An exact algorithm for kinodynamic planning in the plane.* In Proceedings of the sixth annual symposium on Computational geometry, pages 271–280. ACM, 1990. (Cited in page 21.)

[Chang 1995] Hsuan Chang et Tsai-Yen Li. *Assembly maintainability study with motion planning.* In Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on, volume 1, pages 1012–1019. IEEE, 1995. (Cited in page 14.)

[Cheng 2001] Peng Cheng et Steven M LaValle. *Reducing metric sensitivity in randomized trajectory design.* In Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, volume 1, pages 43–48. IEEE, 2001. (Cited in page 24.)

[Cheng 2002] Peng Cheng et Steven M LaValle. *Resolution complete rapidly-exploring random trees.* In ICRA, pages 267–272, 2002. (Cited in page 24.)

[Cortes 2002] Juan Cortes, Thierry Siméon et Jean-Paul Laumond. *A random loop generator for planning the motions of closed kinematic chains using PRM methods.* In Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on, volume 2, pages 2141–2146. IEEE, 2002. (Cited in page 94.)

[Cortés 2004] Juan Cortés et Thierry Siméon. *Sampling-based motion planning under kinematic loop-closure constraints.* In Algorithmic Foundations of Robotics VI, pages 75–90. Springer, 2004. (Cited in page 94.)

[Cowling 2006] Ian D Cowling, James F Whidborne et Alastair K Cooke. *Optimal trajectory planning and LQR control for a quadrotor UAV.* In UKACC International Conference on Control, 2006. (Cited in page 27.)

[Dalibard 2009] Sébastien Dalibard et Jean-Paul Laumond. *Control of probabilistic diffusion in motion planning.* In Algorithmic Foundation of Robotics VIII, pages 467–481. Springer, 2009. (Cited in page 17.)

[Deits 2015] Robin Deits et Russ Tedrake. *Efficient mixed-integer planning for UAVs in cluttered environments.* In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 42–49. IEEE, 2015. (Cited in page 28.)

[Devaurs 2013] Didier Devaurs, Thierry Siméon et Jorge Cortes. *Enhancing the transition-based RRT to deal with complex cost spaces.* In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 4120–4125. IEEE, 2013. (Cited in page 17.)

[Devaurs 2014] Didier Devaurs, Thierry Siméon et Jorge Cortes. *A multi-tree extension of the Transition-based RRT: Application to ordering-and-pathfinding problems in continuous cost spaces.* In Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, pages 2991–2996. IEEE, 2014. (Cited in page 18.)

[Devaurs 2015] Didier Devaurs, Thierry Siméon et Juan Cortés. *Efficient sampling-based approaches to optimal path planning in complex cost spaces.* In Algorithmic Foundations of Robotics XI, pages 143–159. Springer, 2015. (Cited in page 18.)

[Donald 1993] Bruce Donald, Patrick Xavier, John Canny et John Reif. *Kinodynamic motion planning.* Journal of the ACM (JACM), vol. 40, no. 5, pages 1048–1066, 1993. (Cited in pages 10, 18 et 21.)

[Donald 1995a] Bruce Randall Donald et Patrick G. Xavier. *Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open-chain manipulators.* Algorithmica, vol. 14, no. 6, pages 480–530, 1995. (Cited in page 21.)

[Donald 1995b] Bruce Randall Donald et Patrick G. Xavier. *Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds.* Algorithmica, vol. 14, no. 6, pages 443–479, 1995. (Cited in page 21.)

[Dubins 1957] Lester E Dubins. *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents.* American Journal of mathematics, vol. 79, no. 3, pages 497–516, 1957. (Cited in page 10.)

[Ettlin 2006a] Alan Ettlin et Hannes Bleuler. *Randomised rough-terrain robot motion planning.* In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pages 5798–5803. IEEE, 2006. (Cited in page 17.)

[Ettlin 2006b] Alan Ettlin et Hannes Bleuler. *Rough-terrain robot motion planning based on obstacleness.* In Control, Automation, Robotics and Vision, 2006. ICARCV'06. 9th International Conference on, pages 1–6. IEEE, 2006. (Cited in page 17.)

[Faverjon 1984] Bernard Faverjon. *Obstacle avoidance using an octree in the configuration space of a manipulator.* In Robotics and Automation. Proceedings. 1984 IEEE International Conference on, volume 1, pages 504–512. IEEE, 1984. (Cited in page 7.)

[Fernandes 1993] C Fernandes, L Gurvits et ZX Li. *Optimal nonholonomic motion planning for a falling cat.* In Nonholonomic motion planning, pages 379–421. Springer, 1993. (Cited in page 21.)

[Fleury 1995] Sara Fleury, Philippe Soueres, Jean-Paul Laumond et Raja Chatila. *Primitives for smoothing mobile robot trajectories.* IEEE transactions on robotics and automation, vol. 11, no. 3, pages 441–448, 1995. (Cited in page 20.)

[Fliess 1992] M Fliess, J Lévine, Ph Martin et P Rouchon. *On differentially flat nonlinear systems.* In IFAC SYMPOSIA SERIES, pages 159–163, 1992. (Cited in page 38.)

[Fraichard 1993] Th Fraichard et Christian Laugier. *Kinodynamic planning in a structured and time-varying workspace.* In Geometric Reasoning for Perception and Action, pages 19–37. Springer, 1993. (Cited in page 21.)

[Fraichard 1998] Thierry Fraichard. *Trajectory planning in a dynamic workspace: a'state-time space'approach.* Advanced Robotics, vol. 13, no. 1, pages 75–94, 1998. (Cited in pages 20 et 23.)

[Fraichard 2004] Thierry Fraichard et Hajime Asama. *Inevitable collision states - A step towards safer robots?* Advanced Robotics, vol. 18, no. 10, pages 1001–1024, 2004. (Cited in pages 66 et 101.)

[Frazzoli 2000] Emilio Frazzoli, Munther A Dahleh et Eric Feron. *Robust hybrid control for autonomous vehicle motion planning.* In Decision and Control, 2000. Proceedings of the 39th IEEE Conference on, volume 1, pages 821–826. IEEE, 2000. (Cited in pages 22 et 29.)

[Gavrilets 2001] Vladislav Gavrilets, Emilio Frazzoli, Bernard Mettler, Michael Piedmonte et Eric Feron. *Aggressive maneuvering of small autonomous helicopters: A human-centered approach.* The International Journal of Robotics Research, vol. 20, no. 10, pages 795–807, 2001. (Cited in page 29.)

[Geraerts 2007] Roland Geraerts et Mark H Overmars. *Creating high-quality paths for motion planning.* The International Journal of Robotics Research, vol. 26, no. 8, pages 845–863, 2007. (Cited in page 61.)

[Ghallab 2004] Malik Ghallab, Dana Nau et Paolo Traverso. Automated planning: theory and practice. Elsevier, 2004. (Cited in page 91.)

[Giordano 2006] Paolo Robuffo Giordano, Marilena Vendittelli, J-P Laumond et Philippe Soueres. *Nonholonomic distance to polygonal obstacles for a car-like robot of polygonal shape.* IEEE Transactions on Robotics, vol. 22, no. 5, pages 1040–1047, 2006. (Cited in page 11.)

[Glassman 2010] Elena Glassman et Russ Tedrake. *A quadratic regulator-based heuristic for rapidly exploring state space.* In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 5021–5028. IEEE, 2010. (Cited in page 24.)

[Goerzen 2010] Chad Goerzen, Zhaodan Kong et Bernard Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance.* Journal of Intelligent and Robotic Systems, vol. 57, no. 1-4, pages 65–100, 2010. (Cited in page 26.)

[Goretkin 2013] Gustavo Goretkin, Alejandro Perez, Robert Platt et George Konidaris. *Optimal sampling-based planning for linear-quadratic kinodynamic systems.* In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 2429–2436. IEEE, 2013. (Cited in page 22.)

[Hamano 2013] Fumio Hamano. *Derivative of Rotation Matrix Direct Matrix Derivation of Well Known Formula*. arXiv preprint arXiv:1311.6010, 2013. (Cited in page 39.)

[Hehn 2011] Markus Hehn et Raffaello D'Andrea. *Quadrocopter trajectory generation and control*. IFAC Proceedings Volumes, vol. 44, no. 1, pages 1485–1491, 2011. (Cited in page 27.)

[Heinzinger 1990] Greg Heinzinger, Paul Jacobs, John Canny et Bred Paden. *Time-optimal trajectories for a robot manipulator: A provably good approximation algorithm*. In Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on, pages 150–156. IEEE, 1990. (Cited in page 21.)

[Howlett 2007] JK Howlett, M Whalley, P Tsenkov, G Schulein et M Takahashi. *Flight evaluation of a system for unmanned rotorcraft reactive navigation in uncertain urban environments*. In Annual forum proceedings - American Helicopter Society, volume 63, page 507. American Helicopter Society, Inc., 2007. (Cited in page 28.)

[Hsu 1997] David Hsu, Jean-Claude Latombe et Rajeev Motwani. *Path planning in expansive configuration spaces*. In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, volume 3, pages 2719–2726. IEEE, 1997. (Cited in page 14.)

[Hsu 1998] David Hsu, Lydia E Kavraki, Jean-Claude Latombe, Rajeev Motwani, Stephen Sorkin*et al. On finding narrow passages with probabilistic roadmap planners*. In Robotics: The Algorithmic Perspective: 1998 Workshop on the Algorithmic Foundations of Robotics, pages 141–154, 1998. (Cited in page 13.)

[Hsu 2000] David Hsu. *Randomized single-query motion planning in expansive spaces*. PhD thesis, Stanford University, 2000. (Cited in page 14.)

[Hsu 2002] David Hsu, Robert Kindel, Jean-Claude Latombe et Stephen Rock. *Randomized kinodynamic motion planning with moving obstacles*. The International Journal of Robotics Research, vol. 21, no. 3, pages 233–255, 2002. (Cited in page 23.)

[Jaillet 2005] Léonard Jaillet, Anna Yershova, Steven M La Valle et Thierry Siméon. *Adaptive tuning of the sampling domain for dynamic-domain RRTs*. In Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on, pages 2851–2856. IEEE, 2005. (Cited in page 17.)

[Jaillet 2010] Léonard Jaillet, Juan Cortés et Thierry Siméon. *Sampling-based path planning on configuration-space costmaps*. Robotics, IEEE Transactions on, vol. 26, no. 4, pages 635–646, 2010. (Cited in page 17.)

[Janson 2015] Lucas Janson, Edward Schmerling, Ashley Clark et Marco Pavone. *Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions.* The International journal of robotics research, page 0278364915577958, 2015. (Cited in page 23.)

[Judd 2001] Kevin B Judd et Timothy W McLain. *Spline based path planning for unmanned air vehicles.* In AIAA Guidance, Navigation, and Control Conference and Exhibit, pages 1–9. Montreal, Canada, 2001. (Cited in page 28.)

[Kant 1986] Kamal Kant et Steven W Zucker. *Toward efficient trajectory planning: The path-velocity decomposition.* The International Journal of Robotics Research, vol. 5, no. 3, pages 72–89, 1986. (Cited in page 20.)

[Karaman 2010] Sertac Karaman et Emilio Frazzoli. *Optimal kinodynamic motion planning using incremental sampling-based methods.* In 49th IEEE conference on decision and control (CDC), pages 7681–7687. IEEE, 2010. (Cited in page 22.)

[Karaman 2011] Sertac Karaman et Emilio Frazzoli. *Sampling-based algorithms for optimal motion planning.* The International Journal of Robotics Research, vol. 30, no. 7, pages 846–894, 2011. (Cited in pages 13, 14 et 17.)

[Kavraki 1996] Lydia E Kavraki, Petr Švestka, Jean-Claude Latombe et Mark H Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* Robotics and Automation, IEEE Transactions on, vol. 12, no. 4, pages 566–580, 1996. (Cited in page 11.)

[Kavraki 1998] Lydia E Kavraki, Jean-Claude Latombe et E Latombe. *Probabilistic roadmaps for robot path planning.* 1998. (Cited in page 14.)

[Khanmohammadi 2008] Sohrab Khanmohammadi et Amin Mahdizadeh. *Density avoided sampling: An intelligent sampling technique for rapidly-exploring random trees.* In Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on, pages 672–677. IEEE, 2008. (Cited in page 17.)

[Khatib 1986] Oussama Khatib. *Real-time obstacle avoidance for manipulators and mobile robots.* The international journal of robotics research, vol. 5, no. 1, pages 90–98, 1986. (Cited in page 8.)

[Kindel 2000] Robert Kindel, David Hsu, J-C Latombe et Stephen Rock. *Kinodynamic motion planning amidst moving obstacles.* In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, volume 1, pages 537–543. IEEE, 2000. (Cited in page 23.)

[Koren 1991] Yoram Koren et Johann Borenstein. *Potential field methods and their inherent limitations for mobile robot navigation.* In Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, pages 1398–1404. IEEE, 1991. (Cited in page 8.)

[Kröger 2010] Torsten Kröger et Friedrich M Wahl. *Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events.* IEEE Transactions on Robotics, vol. 26, no. 1, pages 94–111, 2010. (Cited in page 101.)

[Kröger 2011] Torsten Kröger. *Online trajectory generation: Straight-line trajectories.* IEEE Transactions on Robotics, vol. 27, no. 5, pages 1010–1016, 2011. (Cited in page 101.)

[Kuffner 2000] James J Kuffner et Steven M LaValle. *RRT-connect: An efficient approach to single-query path planning.* In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on, volume 2, pages 995–1001. IEEE, 2000. (Cited in page 15.)

[Ladd 2004] Andrew M Ladd et Lydia E Kavraki. *Fast tree-based exploration of state space for robots with dynamics.* In Algorithmic Foundations of Robotics VI, pages 297–312. Springer, 2004. (Cited in page 24.)

[Lamiraux 1998] Florent Lamiraux et J-P Laumond. *From paths to trajectories for multi-body mobile robots.* In Experimental Robotics V, pages 301–309. Springer, 1998. (Cited in page 20.)

[Lamiraux 2001] Florent Lamiraux et J-P Lammond. *Smooth motion planning for car-like vehicles.* IEEE Transactions on Robotics and Automation, vol. 17, no. 4, pages 498–501, 2001. (Cited in page 20.)

[Latombe 2012] Jean-Claude Latombe. Robot motion planning, volume 124. Springer Science & Business Media, 2012. (Cited in page 8.)

[Lau 2009] Boris Lau, Christoph Sprunk et Wolfram Burgard. *Kinodynamic motion planning for mobile robots using splines.* In 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2427–2433. IEEE, 2009. (Cited in page 20.)

[Laumond 1993] J-P Laumond et Philippe Soueres. *Metric induced by the shortest paths for a car-like mobile robot.* In Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on, volume 2, pages 1299–1304. IEEE, 1993. (Cited in page 11.)

[Laumond 1998] Jean-Paul Laumond, S Sekhavat et F Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. Springer, 1998. (Cited in page 10.)

[Lavalle 1998] Steven M Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* 1998. (Cited in page 15.)

[LaValle 2000] Steven M LaValle et James J Kuffner Jr. *Rapidly-exploring random trees: Progress and prospects.* 2000. (Cited in page 15.)

[LaValle 2001] Steven M LaValle et James J Kuffner. *Randomized kinodynamic planning.* The International Journal of Robotics Research, vol. 20, no. 5, pages 378–400, 2001. (Cited in pages 11, 23, 24 et 62.)

[LaValle 2006] Steven M LaValle. Planning algorithms. Cambridge university press, 2006. (Cited in page 9.)

[Lee 2010] Taeyoung Lee, Melvin Leoky et N Harris McClamroch. *Geometric tracking control of a quadrotor UAV on SE (3).* In Decision and Control (CDC), 2010 49th IEEE Conference on, pages 5420–5425. IEEE, 2010. (Cited in page 75.)

[Lewis 1995] Frank L Lewis et Vassilis L Syrmos. Optimal control. John Wiley & Sons, 1995. (Cited in page 21.)

[Li 2010] Yanbo Li et Kostas E Bekris. *Balancing state-space coverage in planning with dynamics.* In ICRA, pages 3246–3253, 2010. (Cited in page 26.)

[Li 2016] Yanbo Li, Zakary Littlefield et Kostas E Bekris. *Asymptotically optimal sampling-based kinodynamic planning.* The International Journal of Robotics Research, vol. 35, no. 5, pages 528–564, 2016. (Cited in page 26.)

[Lien 2003] Jyh-Ming Lien, Shawna L Thomas et Nancy M Amato. *A general framework for sampling on the medial axis of the free space.* In Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on, volume 3, pages 4439–4444. IEEE, 2003. (Cited in page 13.)

[Lozano-Pérez 1983] Tomas Lozano-Pérez. *Spatial planning: A configuration space approach.* Computers, IEEE Transactions on, vol. 100, no. 2, pages 108–120, 1983. (Cited in page 6.)

[Lozano-Perez 1987] Tomas Lozano-Perez. *A simple motion-planning algorithm for general robot manipulators.* Robotics and Automation, IEEE Journal of, vol. 3, no. 3, pages 224–238, 1987. (Cited in page 7.)

[MacAllister 2013] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey et Maxim Likhachev. *Path planning for non-circular micro aerial vehicles in constrained environments.* In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 3933–3940. IEEE, 2013. (Cited in page 29.)

[Mellinger 2011] Daniel Mellinger et Vijay Kumar. *Minimum snap trajectory generation and control for quadrotors.* In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 2520–2525. IEEE, 2011. (Cited in pages 27, 39 et 40.)

[Metropolis 1949] Nicholas Metropolis et Stanislaw Ulam. *The monte carlo method.* Journal of the American statistical association, vol. 44, no. 247, pages 335–341, 1949. (Cited in page 8.)

[Mueller 2013] Mark W Mueller, Markus Hehn et Raffaello D'Andrea. *A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification.* In 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3480–3486. IEEE, 2013. (Cited in pages 27 et 100.)

[Nielsen 2000] Christian L Nielsen et Lydia E Kavraki. *A two level fuzzy PRM for manipulation planning.* In Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on, volume 3, pages 1716–1721. IEEE, 2000. (Cited in page 13.)

[Nilsson 1969] Nils J Nilsson. *A mobile automaton: An application of artificial intelligence techniques.* Rapport technique, DTIC Document, 1969. (Cited in page 7.)

[Nissoux 1999] Carole Nissoux. *Visibilité et méthodes probabilistes pour la planification de mouvement en robotique.* PhD thesis, 1999. (Cited in page 13.)

[Ó'Dúnlaing 1987] Colm Ó'Dúnlaing. *Motion planning with inertial constraints.* Algorithmica, vol. 2, no. 1-4, pages 431–475, 1987. (Cited in page 21.)

[Ostrowski 2000] James P Ostrowski, Jaydev P Desai et Vijay Kumar. *Optimal gait selection for nonholonomic locomotion systems.* The International journal of robotics research, vol. 19, no. 3, pages 225–237, 2000. (Cited in page 21.)

[Peng 2005] Jufeng Peng et Srinivas Akella. *Coordinating multiple robots with kinodynamic constraints along specified paths.* The International Journal of Robotics Research, vol. 24, no. 4, pages 295–310, 2005. (Cited in page 20.)

[Perez 2012] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling et Tomas Lozano-Perez. *LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics.* In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 2537–2542. IEEE, 2012. (Cited in page 22.)

[Piedmonte 2000] M Piedmonte et Eric Feron. *Aggressive maneuvering of autonomous aerial vehicles: a human-centered approach.* In ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-, volume 9, pages 413–420, 2000. (Cited in page 29.)

[Pivtoraiko 2011] Mihail Pivtoraiko et Alonzo Kelly. *Kinodynamic motion planning with state lattice motion primitives.* In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2172–2179. IEEE, 2011. (Cited in page 22.)

[Plaku 2007] Erion Plaku, Lydia E Kavraki et Moshe Y Vardi. *Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning.* In Robotics: Science and Systems, pages 326–333, 2007. (Cited in page 24.)

[Reif 1979] John H Reif. *Complexity of the mover's problem and generalizations extended abstract.* In Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science, pages 421–427, 1979. (Cited in pages 6 et 7.)

[Reif 1997] J Reif et H Wang. *Non-uniform discretization approximations for kinodynamic motion planning.* Algorithms for Robotic Motion and Manipulation, pages 97–112, 1997. (Cited in page 21.)

[Richter 2016] Charles Richter, Adam Bry et Nicholas Roy. *Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments.* In Robotics Research, pages 649–666. Springer, 2016. (Cited in pages 20 et 28.)

[Rickert 2008] Markus Rickert, Oliver Brock et Alois Knoll. *Balancing exploration and exploitation in motion planning.* In Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pages 2812–2817. IEEE, 2008. (Cited in page 17.)

[Rodriguez 2006] Samuel Rodriguez, Xinyu Tang, Jyh-Ming Lien et Nancy M Amato. *An obstacle-based rapidly-exploring random tree.* In Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, pages 895–900. IEEE, 2006. (Cited in page 17.)

[Sahar 1986] Gideon Sahar et John M Hollerbach. *Planning of minimum-time trajectories for robot arms.* The International journal of robotics research, vol. 5, no. 3, pages 90–100, 1986. (Cited in page 21.)

[Sánchez 2003] Gildardo Sánchez et Jean-Claude Latombe. *A single-query bi-directional probabilistic roadmap planner with lazy collision checking.* In Robotics Research, pages 403–417. Springer, 2003. (Cited in page 17.)

[Scherer 2007] Sebastian Scherer, Sanjiv Singh, Lyle Chamberlain et Srikanth Saripalli. *Flying fast and low among obstacles.* In Proceedings 2007 IEEE International Conference on Robotics and Automation, pages 2023–2029. IEEE, 2007. (Cited in page 28.)

[Schouwenaars 2004] Tom Schouwenaars, Bernard Mettler, Eric Feron et Jonathan How. *Hybrid model for trajectory planning of agile autonomous vehicles.* Journal of Aerospace Computing, Information, and Communication, vol. 1, no. 12, pages 629–651, 2004. (Cited in page 29.)

[Schulman 2014] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg et Pieter Abbeel. *Motion planning with sequential convex optimization and convex collision checking.* The International Journal of Robotics Research, vol. 33, no. 9, pages 1251–1270, 2014. (Cited in page 21.)

[Schwartz 1983] Jacob T Schwartz et Micha Sharir. *On the piano movers' problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers.* The International Journal of Robotics Research, vol. 2, no. 3, pages 46–75, 1983. (Cited in page 6.)

[Shanmugavel 2006] Madhavan Shanmugavel, Antonios Tsourdos, Rafał Zbikowski, Brian A White, CA Rabbath et N Lechevin. *A solution to simultaneous arrival of multiple UAVs using Pythagorean Hodograph curves.* In 2006 American Control Conference, pages 6–pp. IEEE, 2006. (Cited in page 27.)

[Shanmugavel 2010] Madhavan Shanmugavel, Antonios Tsourdos, Brian White et Rafał Żbikowski. *Co-operative path planning of multiple UAVs using Dubins paths with clothoid arcs.* Control Engineering Practice, vol. 18, no. 9, pages 1084–1092, 2010. (Cited in page 27.)

[Shiller 1991] Zvi Shiller et Steven Dubowsky. *On computing the global time-optimal motions of robotic manipulators in the presence of obstacles.* Robotics and Automation, IEEE Transactions on, vol. 7, no. 6, pages 785–797, 1991. (Cited in page 20.)

[Shin 1985] Kang Shin et N McKay. *Minimum-time control of robotic manipulators with geometric path constraints.* IEEE Transactions on Automatic Control, vol. 30, no. 6, pages 531–541, 1985. (Cited in page 20.)

[Shkolnik 2009] Alexander Shkolnik, Matthew Walter et Russ Tedrake. *Reachability-guided sampling for planning under differential constraints.* In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 2859–2865. IEEE, 2009. (Cited in page 25.)

[Siméon 2000] Thierry Siméon, J-P Laumond et Carole Nissoux. *Visibility-based probabilistic roadmaps for motion planning.* Advanced Robotics, vol. 14, no. 6, pages 477–493, 2000. (Cited in page 13.)

[Siméon 2001] Thierry Siméon, Jean-Paul Laumond et Florent Lamiraux. *Move3D: a generic platform for path planning.* In Proc. IEEE ISATP, 2001. (Cited in pages 73 et 79.)

[Singh 2001] Leena Singh et James Fuller. *Trajectory generation for a UAV in urban terrain, using nonlinear MPC.* In American Control Conference, 2001. Proceedings of the 2001, volume 3, pages 2301–2308. IEEE, 2001. (Cited in pages 27 et 28.)

[Spica 2012a] Riccardo Spica. *Planning and control for aerial grasping with a quadrotor UAV.* PhD thesis, Master Thesis, DIAG, Università di Roma La Sapienza, 2012. (Cited in page 40.)

[Spica 2012b] Riccardo Spica, Antonio Franchi, Giuseppe Oriolo, Heinrich H Bülthoff et Paolo Robuffo Giordano. *Aerial grasping of a moving target with a quadrotor UAV.* In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 4985–4992. IEEE, 2012. (Cited in page 27.)

[Şucan 2009] Ioan A Şucan et Lydia E Kavraki. *Kinodynamic motion planning by interior-exterior cell exploration.* In Algorithmic Foundation of Robotics VIII, pages 449–464. Springer, 2009. (Cited in page 25.)

[Şucan 2012] Ioan A Şucan et Lydia E Kavraki. *A sampling-based tree planner for systems with complex dynamics.* IEEE Transactions on Robotics, vol. 28, no. 1, pages 116–131, 2012. (Cited in page 25.)

[Suzuki 2005] Shinji Suzuki, Yutaka Komatsu, Satoshi Yonezawa, Kazuya Masui et Hiroshi Tomita. *Online four-dimensional flight trajectory search and its flight testing.* In AIAA Guidance, Navigation, and Control Conference and Exhibit, San Francisco, CA, 2005. (Cited in page 28.)

[Takahashi 2008] M Takahashi, G Schulein et M Whalley. *Flight control law design and development for an autonomous rotorcraft.* In Annual forum proceedings - American Helicopter Society, volume 64, page 1652. American Helicopter Society, Inc., 2008. (Cited in page 28.)

[van Geem 2001] C van Geem, T Siméon et J Cortés. *Progress Report on Collision Detection.* Second Year Deliverables of the MOLOG Project, 2001. (Cited in page 13.)

[Vandapel 2005] Nicolas Vandapel, James Kuffner et Omead Amidi. *Planning 3-d path networks in unstructured environments.* In Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pages 4624–4629. IEEE, 2005. (Cited in page 28.)

[Wilmarth 1999] Steven A Wilmarth, Nancy M Amato et Peter E Stiller. *MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space.* In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, volume 2, pages 1024–1031. IEEE, 1999. (Cited in page 13.)

[Wzorek 2006] Mariusz Wzorek, Gianpaolo Conte, Piotr Rudol, Torsten Merz, Simone Duranti et Patrick Doherty. *From motion planning to control-a navigation framework for an autonomous unmanned aerial vehicle.* In 21th Bristol UAV Systems Conference, 2006. (Cited in page 28.)

[Yakimenko 2000] Oleg A Yakimenko. *Direct method for rapid prototyping of near-optimal aircraft trajectories.* Journal of Guidance, Control, and Dynamics, vol. 23, no. 5, pages 865–875, 2000. (Cited in page 29.)

[Yershova 2005] Anna Yershova, Léonard Jaillet, Thierry Siméon et Steven M LaValle. *Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain.* In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, pages 3856–3861. IEEE, 2005. (Cited in page 17.)

[Zucker 2013] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell et Siddhartha S Srinivasa. *Chomp: Covariant hamiltonian optimization for motion planning.* The International Journal of Robotics Research, vol. 32, no. 9-10, pages 1164–1193, 2013. (Cited in page 21.)

Résumé : La planification de mouvement est le domaine de l'informatique qui a trait au développement de techniques algorithmiques permettant la génération automatique de trajectoires pour un système mécanique. La nature d'un tel système varie selon les champs d'application. En animation par ordinateur il peut s'agir d'un avatar humanoïde. En biologie moléculaire cela peut être une protéine. Le domaine d'application de ces travaux étant la robotique aérienne, le système est ici un UAV (Unmanned Aerial Vehicle: véhicule aérien sans pilote) à quatre hélices appelé quadrirotor. Le problème de planification de mouvements consiste à calculer une série de mouvements qui amène le système d'une configuration initiale donnée à une configuration finale souhaitée sans générer de collisions avec son environnement, la plupart du temps connu à l'avance. Les méthodes habituelles explorent l'espace des configurations du système sans tenir compte de sa dynamique. Cependant, la force de poussée qui permet à un quadrirotor de voler est par construction parallèle aux axes de rotation des hélices, ce qui implique que certains mouvements ne peuvent pas être effectués. De plus, l'intensité de cette force de poussée, et donc l'accélération linéaire du centre de masse, sont limitées par les capacités physiques du robot. Pour toutes ces raisons, non seulement la position et l'orientation doivent être planifiées, mais les dérivées plus élevées doivent l'être également si l'on veut que le système physique soit en mesure de réellement exécuter le mouvement. Lorsque c'est le cas, on parle de planification *kinodynamique* de mouvements. Une distinction est faite entre le planificateur local et le planificateur global. Le premier est chargé de produire une trajectoire valide entre deux états du système sans nécessairement tenir compte des collisions. Le second est l'algorithme principal qui est chargé de résoudre le problème de planification de mouvement en explorant l'espace d'état du système. Il fait appel au planificateur local. Nous présentons un planificateur local qui interpole deux états comprenant un nombre arbitraire de degrés de liberté ainsi que leurs dérivées premières et secondes. Compte tenu d'un ensemble de limites sur les dérivées des degrés de liberté jusqu'au quatrième ordre (snap), il produit rapidement une trajectoire en temps minimal quasi-optimale qui respecte ces limites. Dans la plupart des algorithmes modernes de planification de mouvements, l'exploration est guidée par une fonction de distance (ou métrique). Le meilleur choix pour celle-ci est le cost-to-go, *c.a.d.* le coût associé à la méthode locale. Dans le contexte de la planification kinodynamique de mouvements, il correspond à la durée de la trajectoire en temps minimal. Le problème dans ce cas est que calculer le cost-to-go est aussi difficile (et donc aussi coûteux) que de calculer la trajectoire optimale elle-même. Nous présentons une métrique qui est une bonne approximation du cost-to-go, mais dont le calcul est beaucoup moins coûteux. Le paradigme dominant en planification de mouvements aujourd'hui est l'échantillonnage aléatoire. Cette classe d'algorithmes repose sur un échantillonnage aléatoire de l'espace d'état afin de l'explorer rapidement. Une stratégie commune est l'échantillonnage uniforme. Il semble toutefois que, dans notre contexte, ce soit un choix assez médiocre. En effet, une grande majorité des états uniformément échantillonnés ne peuvent pas être interpolés. Nous présentons une stratégie d'échantillonnage incrémentale qui diminue considérablement la probabilité que cela ne se produise.

Mots clefs : Planification kinodynamique de mouvement, robotique aérienne, quadrirotor

Abstract: Motion planning is the field of computer science that aims at developing algorithmic techniques allowing the automatic computation of trajectories for a mechanical system. The nature of such a system vary according to the fields of application. In computer animation it could be a humanoid avatar. In molecular biology it could be a protein. The field of application of this work being aerial robotics, the system is here a four-rotor UAV (Unmanned Aerial Vehicle) called quadrotor. The motion planning problem consists in computing a series of motions that brings the system from a given initial configuration to a desired final configuration without generating collisions with its environment, most of the time known in advance. Usual methods explore the system's configuration space regardless of its dynamics. By construction the thrust force that allows a quadrotor to fly is tangential to its attitude which implies that not every motion can be performed. Furthermore, the magnitude of this thrust force and hence the linear acceleration of the center of mass are limited by the physical capabilities of the robot. For all these reasons, not only position and orientation must be planned, higher derivatives must be planned also if the motion is to be executed. When this is the case we talk of *kinodynamic motion planning*. A distinction is made between the local planner and the global planner. The former is in charge of producing a valid trajectory between two states of the system without necessarily taking collisions into account. The later is the overall algorithmic process that is in charge of solving the motion planning problem by exploring the state space of the system. It relies on multiple calls to the local planner. We present a local planner that interpolates two states consisting of an arbitrary number of degrees of freedom (dof) and their first and second derivatives. Given a set of bounds on the dof derivatives up to the fourth order (snap), it quickly produces a near-optimal minimum time trajectory that respects those bounds. In most of modern global motion planning algorithms, the exploration is guided by a distance function (or metric). The best choice is the cost-to-go, *i.e.* the cost associated to the local method. In the context of kinodynamic motion planning, it is the duration of the minimal-time trajectory. The problem in this case is that computing the cost-to-go is as hard (and thus as costly) as computing the optimal trajectory itself. We present a metric that is a good approximation of the cost-to-go but which computation is far less time consuming. The dominant paradigm nowadays is *sampling-based motion planning*. This class of algorithms relies on random sampling of the state space in order to quickly explore it. A common strategy is uniform sampling. It however appears that, in our context, it is a rather poor choice. Indeed, a great majority of uniformly sampled states cannot be interpolated. We present an incremental sampling strategy that significantly decreases the probability of this happening.

Key words: Kynodynamic motion planning, aerial robotics, quadrotor