Contributions to optimal and reactive vision-based trajectory generation for a quadrotor UAV

Bryan Penin

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THESE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Automatique, Productique et Robotique – Signal, Image, Vision – Télécommunications*

Par
Bryan Penin

Contributions to optimal and reactive vision-based trajectory generation for a quadrotor UAV

Thèse présentée et soutenue à Rennes, le 11/12/18
Unité de recherche : Inria Rennes
Thèse N° :

Rapporteurs avant soutenance :

Antonio Franchi	Chargé de recherche CNRS, LAAS, Toulouse
Tarek Hamel	Professeur, I3S, Université de Nice-Sophia-Antipolis

Composition du Jury :

Président :	Isabelle Fantoni	Directrice de Recherche CNRS, LS2N Nantes
Examinateurs :	Pierre-Brice Wieber	Chargé de Recherche Inria, Inria Grenoble
	Isabelle Fantoni	Directrice de Recherche CNRS, LS2N Nantes
Dir. de thèse :	François Chaumette	Directeur de recherche Inria, Inria/IRISA Rennes
Co-dir. de thèse :	Paolo Robuffo Giordano	Directeur de recherche CNRS, IRISA/Inria Rennes

Invitée
Marie-Véronique Serfaty — Responsable scientifique, DGA/MRIS, Paris

To my grandfather

In theory, there is no difference between theory and practice.
But, in practice, there is.
— Jan L.A. van de Snepscheut, 1953 - 1994, computer scientist

Acknowledgements

This thesis is the result of three amazing years at Inria in the Rainbow Group. During my time in such a great research team (formerly known as the LAGADIC Team) I have learned a lot both on professional and personal aspects. I really enjoyed working on such interesting topics. Finally, I had a lot of fun in the lab and outside. For these reasons, I am grateful to everyone who contributed to any of those things.

To start with, I would like to thank my supervisor Dr. François Chaumette for the great environment he built in the team over the years, his easy and free conversation and his scientific rigorousness. My great appreciation goes to my second supervisor Dr. Paolo Robuffo Giordano. I am grateful for his availability, his scientific support and his expert advise. Both of them provided an ideal environment with all the scientific tools, mutual respect climate and freedom I needed.

I wish to thank a friend and former colleague Dr. Riccardo Spica who really helped me starting the machine. From such a great example I learned about the job of a Ph.D student and I improved my way of doing things. I was really amazed by his scientific knowledge and the relevant advices he gave me during my first year.

Now, I want to thank my great friends and colleagues who managed the drone platform over the years with great skills and patience, namely Thomas and Pol.

I cannot refrain to thank my past and current colleagues at Inria for always making my Doctorate a pleasant experience. More than colleagues they became good friends, all of whom I enjoyed the company during conferences, travels, rock climbing and basketball sessions, memorable nights in Rennes and so on. Without a doubt we built a strong bounding that will last. Keep the same spirit and I wish you all good luck.

Finally, I want to dedicate these last lines of acknowledgement to my parents for always believing in me.

Contents

Chapter 1

Introduction

Contents

1.1 Quadrotors in robotics

In the same way research in robotic vehicle mobility favoured wheeled robots to derive fundamental results, among all Unmanned Aerial Vehicles (UAVs) quadrotors have been considered as the most flexible and versatile platforms worldwide for undertaking aerial research over the last 15 years. These vehicles are capable of agile motion and stable hovering in 3D space that offer ideal capabilities for many different applications including but not limited to: surveillance, search-and-rescue, reconnaissance, transport and inspection in complex environments. However, quadrotors are subject to much more uncertainty than ground vehicles (e.g., modelling, actuation and sensing uncertainty) and are more sensitive to external disturbances (e.g., wind gust, physical interaction with the environment or other robots). These challenges have actually enhanced the investigation on more complex research problems related to three-dimensional planning, control, localization and sensing. Although quadrotors are known to suffer from a much lower flight efficiency than fixed-wing aircraft, these low-cost platforms provide now sufficient flight endurance and payload for a number of indoor and outdoor applications and are now more and more seriously considered for commercial applications (e.g., package delivery, advertising, aerial photography) or emergency assistance (e.g., first-aid kit delivery Fig. 1.1a,

fire fighting Fig. 1.1c, disaster analysis Fig. 1.1b). Moreover, quadrotors are mostly equipped with vision sensors that bring them to the forefront of inspection and surveillance in complex and unstructured environments. Note that with the recent development of new lithium-lion batteries providing up to 2 hours of flight with a single charge, Impossible Aerospace Fig. 1.1c now opens a door to new exciting aerial applications.

(a) Quadrotors could be deployed to rapidly deliver first aid kits.

(b) A quadrotor used by British NGOs during the Nepal earthquake in April 2015.

(c) Prelaunch units of the new quadrotor US-1 developed by Impossible Aerospace have been sold to firefighters, police departments, and search and rescue teams across the United States. The system could carry thermal cameras or multispectral sensors for search-and-rescue applications.

Figure 1.1 – Example of practical applications completed by a quadrotor UAV

Endowed with a special actuation configuration that allows extremely high motion capabilities, quadrotors are inherently prone to high speed and agile flights. Since several years, the research community has been developing new control and planning methods in the field of three-dimensional dynamic motion for systems with fast control loops such as quadrotors. Today, quadrotors have reached a very satisfying level of autonomy and reliability for fundamental research applications. Yet, the active research in aerial robotics is pushing the limits of planning, control and sensing to address more complex and agile tasks.

Moreover, computation improvements have also motivated the revision of al-

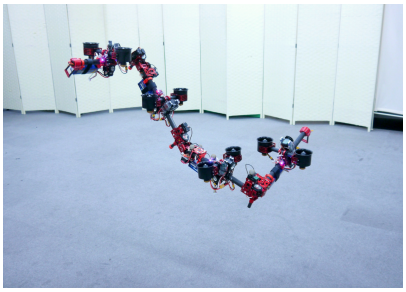
ready existing control and planning techniques (especially optimization-based methods) to the concept of aerial robots that can plan their motion online and quickly respond to changes in dynamic environments. Starting from this idea, many researchers have focused their effort on apprehending flight characteristics and properties of motion itself (optimality, representation and especially the notion of *smoothness*[1]) and their implications in control and planning for completing complex and reactive tasks such as aggressive grasping [5] or interception manoeuvres [6]. Several testbeds such as [7, 8, 9] have originated from these fundamental studies to demonstrate the feasibility of new motions that are close to the actuation limits (and singularities) such as flips [10], aerobatics [11], swing manoeuvres [12] or juggling [13].

Figure 1.2 – A classic quadrotor platform (from MikroKopter [2]) used for our experiments.

Originally composed of four propellers (see Fig. 1.2), quadrotors have been subject to numerous advanced mechanical design extensions depending on the aerial task to be performed. One can acknowledge overactuated variable-pitch quadrotors [14, 15] or aerial systems with tilting rotors (see e.g., [16]). Finally, there exists systems that can change their configuration: for a hexarotor in [17] or for a complex multi-body UAV in [18] (Fig. 1.3a). In a similar spirit [19] augments a quadrotor with a hooking system to enhance its motion capabilities and [20] with anchoring modules to extend its flight autonomy for instance Fig. 1.3b.

These platforms are mostly designed to facilitate physical interaction and navigation in cluttered environment. Yet, complex actuation leads to larger uncertainties and the complex control algorithms involved make them not mature enough for undertaking aggressive motions in 3D we are interested in.

[1] The exact meaning of this term will be clear later



(a) The multilink DRAGON system can change its shape for passing in narrow holes or carrying objects, from [18].

(b) The SpiderMAV uses an air-compressed module to shoot hooks attached with wires to stabilize itself, from [20].

Figure 1.3 – Complex augmented aerial systems capable of extending their flight capabilities

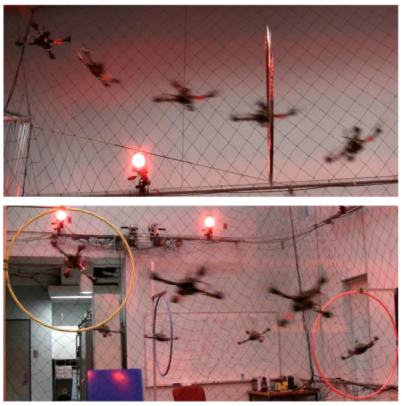
In the course of this thesis, several fundamental works have arisen from the research community to demonstrate that quadrotors are capable of performing complex tasks while exploiting their full potential especially in terms of agility. Along these works, state estimation and sensing algorithms have been improved to cope with high speed motions. The sensory channel has not changed much but has surely improved due to continuing progress in technology. Quadrotors can now be equipped with complex vision sensors such as a lidar or a kinect. For instance [21] equipped a quadrotor with a nodding Hokuyo lidar, a second lidar serving as an altimeter and a high-resolution stereo camera to perform autonomous flights at impressive speeds up to 18m/s. Among all vision sensors, cameras are still the most preferred ones and possess a long history in robotic control. New kind of cameras are even developed especially for these applications such as the *event-based* cameras [22].

Since quadrotors have fewer independent control actuators than degrees of freedom (four motors for controlling six degrees of freedom) they belong to the large class of underactuated mechanical systems. Controlling such systems is challenging to the nonlinear control community especially in terms of stability and robustness. Developing controllers for these systems is clearly motivated by the mechanical gain procured by their simple mechanical structure. A extensive study of the control of underactuated systems can be found in e.g., [23].

Even though nonlinear controllers have been developed for quadrotors, stable and robust control is still challenging when the system has to undertake aggressive manoeuvres. This is due to the fact that the robot attitude is not negligible and aerodynamics become significant and are difficult to model and to incorporate in control. In this context, proofs of convergence and stability are much more laborious

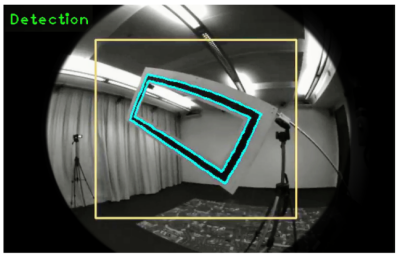
to establish. Nowadays, optimization-based planning methods appear to be more and more flexible and adapted for computing trajectories at the edge of the system motion capabilities for satisfying multiple (and possibly conflicting) goals while being subject to numerous (and possibly nonlinear) constraints.

Pioneered by [1] (Fig. 1.4a) in 2011 very recent works show how optimization techniques are able to produce aggressive flight modes based on the generation of feasible and smooth trajectories [24] demonstrated that quadrotors can even undertake agile motions in a complete autonomous way using vision as principal feedback Fig. 1.4b. All the papers cited in this section were published in 2016 at the earliest and shown in Fig. 1.4 and Fig. 1.5. This shows the current interest that has sparked in the field of agile manoeuvres.



(a) Smooth and fast flights between hoops, from [1].

(a) The quadrotor passing through the gap.



(b) View from the onboard camera

(b) Autonomous agile passage through a window using visual feedback, from [24].

Figure 1.4 – Aggressive trajectories performed at the GRASP Lab from the University of Pennsylvania (left) and at the Robotics & Perception Group - UZH ETH Zurich (right).

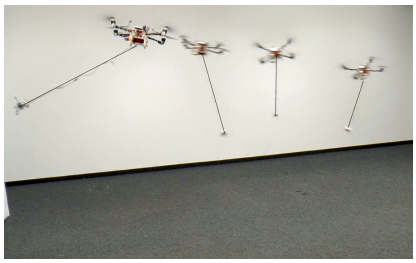


Figure 1.5 – Aggressive perching manoeuvres (left figure from [5]) and agile swing manoeuvres with a suspended load (right figure from [25])

1.2 Visual and optimal aspects in biology

It is not new, in many occasions research gets its inspiration from biology. Indeed, even small insects are able to deal with very complex tasks using visual feedback that we still cannot fully replicate today. It has been shown that a dragonfly aims at aligning its body and bearing to the prey's direction of motion to improve manoeuvrability and speed while closing the vertical distance to it (Fig. 1.6). They even predict the prey's motion such that there is virtually no prey motion on the eye for most of the flight [26].

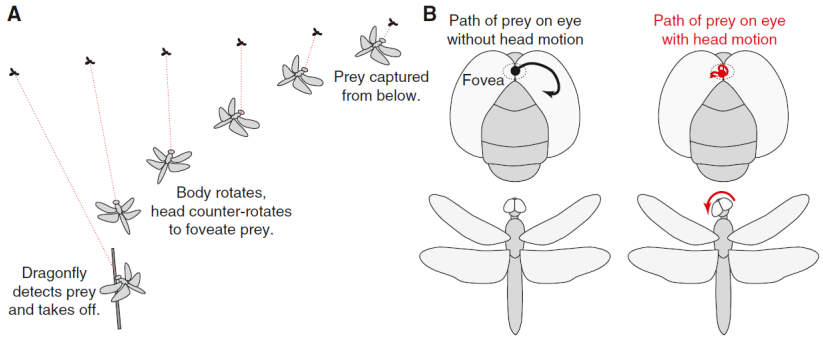


Figure 1.6 – Hunting approach of a dragonfly and head motion (from **??**).

Raptors approach their preys by following a spiral trajectory rather than a (shorter) straight path [27] in order to keep the prey in the field of view and to optimize flight speed (Fig. 1.7). Indeed, following a straight path would force the falcon to turn its head to keep visibility of the prey, an action which would produce significant aerodynamic disturbances and reduce flight speed.

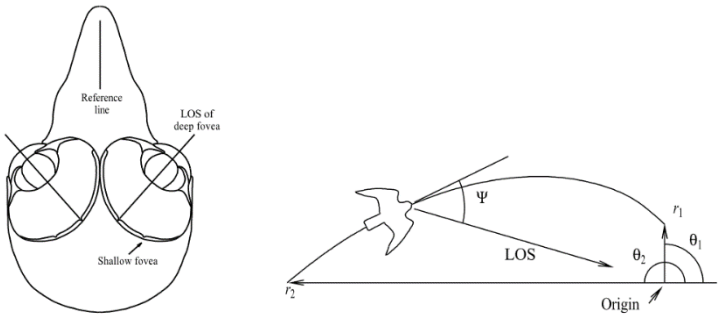


Figure 1.7 – Trajectory optimization in peregrine falcons. Anatomical structure of the falcon eye with the deep fovea (the area specialized in acute vision) pointing approximately 45 degrees to the side of the frontal line. Spiral trajectory followed by the falcon to optimize visual perception and air drag while flying toward its prey (from [27]).

We can see that trajectories are then the (natural) result of a joint performance maximizing both perception and action and are strongly shaped by biomechanical

constraints and visibility. Such optimal aspects have been transferred to aerial research and referred as perception based planning and sensor-based control.

1.3 Challenges

Quadrotors are under-actuated systems since they have four control inputs to control their six degrees of freedom. Such a simple configuration is desirable for performing agile motions but at the cost of shifting the difficulty to the control and the planning schemes. Literature flourishes with contributions on these topics. Most of the works consider applications requiring near-hovering flights and low speed motions that allow the use of much simpler control schemes. As the state-of-the-art in Chapt. 5 will show, recent works have been dedicated to pushing the quadrotors flight limits in order to perform agile manoeuvres in more complex scenarios. Now, although sensors are growing in accuracy and processors are becoming more powerful, reliable estimation of the robot state is still challenging knowing it is crucial in this context. To the latter purpose but also for designing control schemes, the use of cameras has been very popular in robotics since they can provide rich information by the observation of some visual features present in the scene. However, when attached to a quadrotor the underactuation may severely affect the visual perception since the camera will undergo possibly large rotations. For these reasons, aggressive control of quadrotors should account for the capacity of the visual feedback to provide reliable information.

Figure 1.8 represents a quadrotor with a down-facing camera that needs to move in the right direction while using the red dot on the ground as visual feedback for a visual-based control scheme (such as *visual servoing* [28]). Since the commanded velocities are defined in the image plane, in order to move in the desired direction, the robot must necessarily rotate clockwise so as to correctly orient the thrust force generated by the propellers. While doing so, the field of view of the camera will also move and the robot might lose visibility of the target. Guaranteeing visibility of the visual features is of paramount importance since loosing visual tracking leads to an increasingly poor state estimation (that would just be driven by the odometry, i.e., the onboard IMU) and, thus, possibly, to a controller/task failure.

Furthermore, when performing agile flights close to the physical limits of a quadrotor, motors might saturate, which leads to an inability to control the four independent degrees of freedom. A proper choice of the dynamic constraints and considerations on some motion properties such as smoothness are the common ingredients for planning feasible trajectories that can be accurately tracked by the real system.

Failure of the task can also be caused by the magnitude of modelling errors

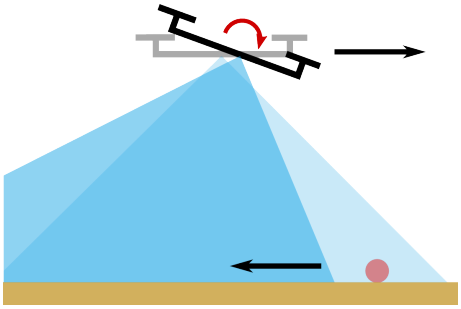


Figure 1.8 – Effect of underactuation for visual control of a quadrotor: the red target is repelled from the field of view as the quadrotor moves towards it.

and disturbances that are boosted at high speeds. Robustness is of paramount importance in that case and can be achieved by efficiently adapting the trajectory on-line according to changes of the environments or to the tracker response. A second reasoning (among others) is to model or identify the unknown variables and aspects in order to be directly incorporated in control. These techniques will help absorbing the uncertainties acting inside or on the system.

This thesis tackles the following challenges:

- The exploitation of the potential of a quadrotor in terms of agility to perform agile manoeuvres.
- The incorporation of a collection of vision-based constraints in planning for maintaining visibility.
- The efficient re-planning of reactive trajectories subject to multiple constraints.
- The incorporation of the state estimation uncertainty at the planning stage.

1.4 Thesis contributions

More precisely, this thesis focuses on developing real-time trajectory generation algorithms for undertaking aggressive motions while satisfying a collection of complex constraints with a particular care for visual perception. We rely on already existing trajectory controllers running at a high frequency for accurately tracking the optimal trajectories. Nevertheless, the design of such trajectories incorporates motion aspects that are beneficial for the tracker performance. These strategies use a Receding Horizon Control (RHC) approach for modifying online the reference trajectory in order to account for noise, disturbances and any other non-modelled effect. We are mostly interested in visual perception, therefore the presented planning strategies targets visual constraints for maintaining visibility and avoiding occlusions by obstacles present in the environment. Indeed, quadrotors can estimate their state

by collecting visual measurements from targets that have to remain visible during motion. The presented planning methods rely on efficiently solving nonlinear optimal control programs and are applied to the tracking of a moving target and navigation. This thesis also presents a contribution in uncertainty-aware planning under intermittent measurements collected from vision. The goal is to relax the visibility constraints that can be very restrictive for navigating in large environments. The contributions are listed below:

- An on-line re-planning algorithm for generating minimum-time trajectories under visibility constraints.
- A reactive re-planning strategy for aggressive target tracking while avoiding occlusions and collisions.
- A novel graph-search planner for finding robust minimum-time trajectories in the presence of intermittent visual measurements for a unicycle and a quadrotor.

Our work led to the following contributions:

- **Penin, Bryan** and Spica, Riccardo and Giordano, Paolo Robuffo and Chaumette, François. "Vision-Based Minimum-Time Trajectory Generation for a Quadrotor UAV" in IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2017.
- **Penin, Bryan** and Giordano, Paolo Robuffo and Chaumette, François, "Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions" IEEE Robotics and Automation Letters, 2018.
- **Penin, Bryan** and Giordano, Paolo Robuffo and Chaumette, François, "Minimum-Time Trajectory Planning Under Intermittent Measurements" IEEE Robotics and Automation Letters, 2019. Submitted to RAL/ICRA'19.

1.5 Thesis structure

This thesis is divided into three main parts. The first part (Part I) contains an introduction to the ingredients for control and planning for quadrotors. Then, the main tools used in our works are presented along with a state of the art in the related topics. The second part (Part II) highlights our contributions about optimization-based planning algorithms. The results illustrated in this part correspond to the following publications of the candidate [2, 3] and [4]. The third part (Part III) contains the thesis conclusions and future directions. Finally, we report complementary

information corroborating a few technical sections.

Outline of Part I

This part contains preliminaries on the different tools and techniques involved in this thesis and a state-of-the-art on the related topics.

Chapter 2 gives an introduction to the quadrotor system and an overview of classic control, planning and state estimation techniques. It also introduces the notion of differential flatness which is a fundamental property in trajectory planning. Finally, we bring to light the issues related to vision-based control.

Chapter 3 provides standard techniques for solving an optimal control problem.

Chapter 4 introduces Model Predictive Control and preliminary results of simple applications to trajectory generation with a quadrotor are presented.

Chapter 5 gives an extensive state of the art for the topics of online planning, vision-based control and optimal aggressive trajectory generation.

Outline of Part II

This part contains the author contributions

Chapter 6 provides our contributions on minimum-time vision-based planning in the presence of visibility constraints.

Chapter 7 presents our planning frameworks for addressing collisions and occlusions avoidance while tracking a moving target. A relaxed formulation is given for allowing reactive re-planning.

Chapter 8 introduces the uncertainty-aware graph-search algorithm that we developed for finding robust minimum-time trajectories in presence of intermittent visual measurements.

Outline of Part III

Chapter 9 reports the conclusions of the thesis and the main contributions brought to the state of the art are summarized. Moreover, some open issues are listed and we discuss future directions which would be worth exploring.

Appendix A gives the complete flat transformation and its inverse for the quadrotor.

Appendix B provides complementary information on the B-spline curves and details on the relevant manipulation algorithms used in this thesis.

Appendix C includes an introduction to numerical techniques for evaluating derivatives and especially complex-step differentiation. Practical results are given for comparing finite difference with complex-step difference.

Part I

Preliminaries and related works

Chapter 2

Planning and control of a quadrotor UAV

Contents

2.1 Introduction

Our focus in this chapter is on the modelling of the quadrotor dynamics and on the role of controllers and trajectory generation.

Because of the nonlinear dynamic behaviour, the control and guidance of quadrotors remain subjects of active research, especially in applications covering search-and-rescue, surveillance, inspection, etc. For these applications, high stability, high precision hovering ability, high bandwidth, and high manoeuvrability are important.

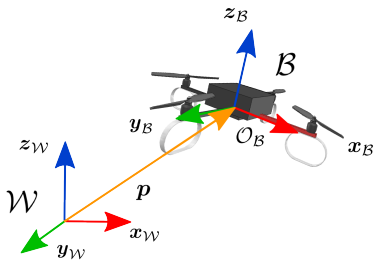


Figure 2.1 – Quadrotor model

Quadrotors have been widely adopted as experimental platforms for research in flying robotics. Reasons for the popularity of these vehicles include the ability to hover, mechanical simplicity and robustness, and their exceptional manoeuvrability due to typically high thrust-to-weight ratios explained by the relatively large off-center mounting of the propellers. Such a configuration offers very low rotational inertia, thus allowing large translational accelerations and extraordinarily fast rotational accelerations (when free of other body parts or payloads). These motion capabilities authorize complex and agile manoeuvres that have been demonstrated in [29, 30, 31, 24] for instance.

Moreover, having four propellers of small diameter reduces the damage in case of collision with an obstacle due to their low kinetic energy. This makes it safer to navigate in narrow and cluttered environments. Some works even studied recovery flight modes for quadrotors in case of complete loss of one to three propellers [32].

2.2 Quadrotor model

Now let us derive the general equations of motion for the quadrotor. With reference to Fig. 2.1, let us define a world frame $\mathcal{W} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$ (being $\boldsymbol{e}_i$ the i-th column of the identity matrix) and a body frame $\mathcal{B} = \{\boldsymbol{x}_\mathcal{B}, \boldsymbol{y}_\mathcal{B}, \boldsymbol{z}_\mathcal{B}\}$ with fixed origin $\mathcal{O}_\mathcal{B}$ attached to the center of mass (COM) and axis $\boldsymbol{z}_\mathcal{B}$ parallel to the propeller rotational axes. The configuration manifold is the special Euclidean group SE(3), which is the semi-direct product of $\mathbb{R}^3$ and the special orthogonal group SO(3) $= \left\{\boldsymbol{R} \in \mathbb{R}^{3\times3} \mid \boldsymbol{R}^T\boldsymbol{R} = \mathcal{I},\ det\,\boldsymbol{R} = 1\right\}$. Let us also assume, without loss of generality, that the robot COM corresponds to the barycentre of the propellers.

The robot state is

$$\boldsymbol{\chi} = \begin{pmatrix} \boldsymbol{r}_\mathcal{B} \\ {}^\mathcal{W}\boldsymbol{R}_\mathcal{B} \\ \boldsymbol{v}_\mathcal{B} \\ {}^\mathcal{B}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}} \end{pmatrix} \in SE(3) \times \mathbb{R}^6 \tag{2.1}$$

where $\boldsymbol{r}_\mathcal{B} \in \mathbb{R}^3$ is the position of the robot COM in $\mathcal{W}$, ${}^\mathcal{W}\boldsymbol{R}_\mathcal{B} \in$ SO(3) is the rotation

matrix from $\mathcal{W}$ to $\mathcal{B}$, $\boldsymbol{v}_{\mathcal{B}}$ the COM linear velocity expressed in $\mathcal{W}$ and ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$ the angular velocity expressed in $\mathcal{B}$.

In some cases, for more clarity, we will also use the roll, pitch and yaw (RPY) angles to represent the orientation of the robot. The rotation matrix corresponding to a given RPY configuration is given by:

$$
\begin{aligned}
{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}(\phi,\theta,\psi) &= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) && \text{(2.2a)}\\
&= \begin{pmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{pmatrix} && \text{(2.2b)}
\end{aligned}
$$

It is also immediate to verify that the inverse transformation is given by:

$$
\begin{aligned}
\theta &= \text{Arctan2}\left(-r_{31} \pm \sqrt{r_{32}^2 + r_{33}^2}\right) && \text{(2.3a)}\\
\phi &= \text{Arctan2}\left(r_{32}, r_{33}\right) && \text{(2.3b)}\\
\psi &= \text{Arctan2}\left(r_{21}, r_{11}\right) && \text{(2.3c)}
\end{aligned}
$$

where r_{ij} indicates the component on the i-th row and j-th column of ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$. The transformation has a *singularity of representation* for $cos(\theta) = 0$.

As known, the derivative of the rotation matrix is given by:

$$
{}^{\mathcal{W}}\dot{\boldsymbol{R}}_{\mathcal{B}} = {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}{}^{\mathcal{B}}\boldsymbol{\Omega}_{\mathcal{BW}} \tag{2.4}
$$

where ${}^{\mathcal{B}}\boldsymbol{\Omega}_{\mathcal{BW}}$ is the skew-symmetric matrix built with the components of ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$. More specifically, assuming that

$$
{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}} = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{2.5}
$$

we have

$$
{}^{\mathcal{B}}\boldsymbol{\Omega}_{\mathcal{BW}} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_y \\ -\omega_y & \omega_x & 0 \end{pmatrix} \tag{2.6}
$$

The map that relates ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$ to the corresponding skew-symmetric matrix ${}^{\mathcal{B}}\boldsymbol{\Omega}_{\mathcal{BW}}$ is often called *hat-map*. Its inverse typically takes the name of *vee-map*. The angular velocity of the robot is also related to the vector of roll, pitch and yaw angles

derivatives, indeed

$$
{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}} = \boldsymbol{R}_x(\phi)^T \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \boldsymbol{R}_x(\phi)^T \boldsymbol{R}_y(\theta)^T \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \boldsymbol{R}_x(\phi)^T \boldsymbol{R}_y(\theta)^T \boldsymbol{R}_z(\psi)^T \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix}
\tag{2.7}
$$

then

$$
{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}} = \mathbf{T}(\theta, \phi) \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}
\tag{2.8}
$$

where

$$
\mathbf{T}(\theta, \phi) = \begin{pmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\theta)\cos(\phi) \end{pmatrix}
\tag{2.9}
$$

Since

$$
det(\mathbf{T}(\theta, \phi)) = \cos(\theta)
\tag{2.10}
$$

the above relation is invertible out of the singularities of representation and its inverse is

$$
\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{pmatrix} {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}
\tag{2.11}
$$

As it is well known, each of the four propellers produces a force of modulus f_i along $\boldsymbol{z}_{\mathcal{B}}$ and a torque of modulus τ_i about $\boldsymbol{z}_{\mathcal{B}}$. Both can be modelled in first approximation as proportional to the square of the motor rotational speed ω_i

$$
f_i = k{\omega_i}^2
\tag{2.12a}
$$

$$
\tau_i = b{\omega_i}^2
\tag{2.12b}
$$

where k and b are the *thrust* and drag factors respectively. They are both positive and their value depends on the shape of the propellers.

We also introduce the following input transformation:

$$
\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} k & k & k & k \\ 0 & kl & 0 & -kl \\ -kl & 0 & kl & 0 \\ b & -b & b & -b \end{pmatrix} \begin{pmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \end{pmatrix} = \mathbf{A}\tilde{\boldsymbol{u}}
\tag{2.13}
$$

where l is the distance between the rotor axes of rotation and the geometric center of the quadrotor. The matrix $\mathbf{A}$ has always maximum rank, then the transformation

is also invertible. The transformed input vector comprises the total thrust force u_1 along $\boldsymbol{z}_\mathcal{B}$ and the torques u_2, u_3, u_4 around $\boldsymbol{x}_\mathcal{B}$, $\boldsymbol{y}_\mathcal{B}$ and $\boldsymbol{z}_\mathcal{B}$ respectively.

Having said that, the forces acting on the system are the gravity force directed along $\boldsymbol{z}_\mathcal{B} = \boldsymbol{e}_3$ and the total thrust force generated by the propellers and directed along $\boldsymbol{z}_\mathcal{B}$. We also assume that the robot center of mass is coincident with its geometric center, where the total thrust is applied. With these assumptions, the translational dynamics of the system is given by the following Newton's equation:

$$m\ddot{\boldsymbol{r}}_\mathcal{B} = -m\boldsymbol{g}\boldsymbol{e}_3 + u_1\boldsymbol{z}_\mathcal{B} \tag{2.14}$$

where m is the robot mass and $\boldsymbol{g} \in \mathbb{R}^3$ the (constant) gravity acceleration in the world frame. The angular acceleration is governed by the Euler's equation. Since the gravity force is applied to the robot center of mass, the only torque acting on the system is the one generated by the propellers, hence

$$\boldsymbol{J}^\mathcal{B}\dot{\boldsymbol{\omega}}_{\mathcal{BW}} + {}^\mathcal{B}\boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{J}^\mathcal{B}\boldsymbol{\omega}_{\mathcal{BW}} = \begin{pmatrix} u_2 \\ u_3 \\ u_4 \end{pmatrix} \tag{2.15}$$

where $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the inertia tensor. If the robot is assumed to have a perfect cylindrical symmetry with respect to the axis $\boldsymbol{z}_\mathcal{B}$, the inertia matrix is also diagonal and two of its eigenvalues, namely J_{xx} and J_{yy}, are equal. This makes it possible, if desired, to neglect the gyroscopic term ${}^\mathcal{B}\boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{J}^\mathcal{B}\boldsymbol{\omega}_{\mathcal{BW}}$ without introducing large modelling errors (see for example [29]).

Finally, we define the quadrotor dynamics with simplified notation as

$$\dot{\boldsymbol{r}} = \boldsymbol{v} \tag{2.16a}$$

$$\dot{\boldsymbol{v}} = \boldsymbol{g} - \frac{f}{m}\boldsymbol{z}_\mathcal{B} \tag{2.16b}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}[\boldsymbol{\omega}]_\times \tag{2.16c}$$

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}([\boldsymbol{J}\boldsymbol{\omega}]_\times\boldsymbol{\omega} + \boldsymbol{\tau}) \tag{2.16d}$$

where $[\cdot]_\times$ the usual skew-symmetric operator, $\boldsymbol{R} = {}^\mathcal{W}\boldsymbol{R}_\mathcal{B}$ and $(f, \boldsymbol{\tau}) \in \mathbb{R}^4$ are the total thrust and torques applied by the propellers, which can be expressed as a set of control inputs $\boldsymbol{u}$ in terms of the individual propeller thrusts $\boldsymbol{u} = (f_1, f_2, f_3, f_4) \in \mathbb{R}^4$ with the linear expression (2.13).

Most of agile control methods have proved that high performance flights can be performed with the present quadrotor model [29, 33, 30]. Nowadays, efforts seem to be less and less allocated to the development of more accurate dynamical modelling (the above equations of motion are approximate, see e.g., [34] for a more precise modelling). It seems these issues have reached an adequate level of maturity and

do not need further major improvements, at least concerning standard applications. Incorporating the motor dynamics would add a fifth order to the dynamics without significantly improving the performance. Modelling simplifications are even more and more considered (especially on the inputs constraints) in order to face with more complex and higher-level tasks applications [6, 35, 36]. Since one seeks to exploit the quadrotor's agility, it would be reasonable to consider aerodynamic effects, which become consistent when small aerial vehicles reach high velocities. However, these effects are rather complex to model and to incorporate into the control. For these reasons we choose to neglect any aerodynamic effect, entrusting the control action for their compensation. A philosophy largely exploited in control in robotics (more details are given in the following section).

2.3 General control and trajectory generation techniques for a quadrotor

Although quadrotors have a low mechanical complexity their control is still challenging. The major difficulty lies in the system underactuation, i.e., the coupling between the translational and rotational motions (2.16b). Since the number of independent inputs is less than its degrees of freedom some trajectories are not reachable making it difficult to find feasible trajectories and then design reasonable tracking control laws.

2.3.1 Quadrotor control

The most common nonlinear control techniques used to control quadrotor are backstepping [37], integral backstepping [38], sliding-mode control [39], feedback linearization [40] and combination of these methods [41]. Because of the highly nonlinear dynamics most of the works in the area use controllers that are derived from linearisation of the model around hover conditions [42]. Stability can be guaranteed for reasonably small roll and pitch angles [43]. These simplifications lead to neglecting the underactuation and alleviate the equations of motion to derive stability and convergence proofs.

Besides these common control schemes, several other control methods from the optimal control theory have been proposed in the literature for quadrotor control such as Linear Quadratic Regulator control (LQR) [44], Model Predictive Control (MPC) [45].

At a lower level, a common architecture for underactuated control consists in a two-loop design [46, 47], where the outer loop is the position control and the inner loop provides attitude (roll, pitch and yaw) control, as illustrated in Fig. 2.2. The

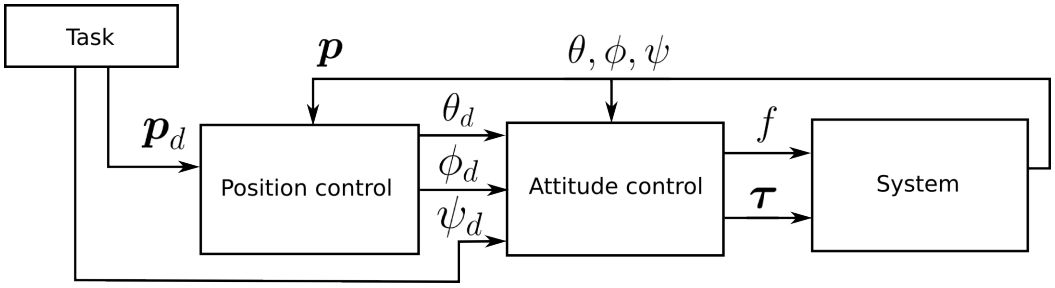


Figure 2.2 – Classic two loop controller. A task provides the desired position $\boldsymbol{p}_d$ and yaw angle ψ_d. The position controller generates the required pitch and roll angles to the attitude controller which applies the computed input thrust and torques to the system.

outer loop typically implements a PD control law on position and velocity with feed-forward terms to compensate for gravity and accelerations from a reference trajectory. A desired acceleration is computed and mapped to the desired collective thrust and a desired attitude of the simplified quadrotor model. The inner loop controls the attitude together with the altitude on the assumption that the attitude dynamics of a quadrotor are much faster than its position and velocity dynamics. Such a control strategy provides almost asymptotical stability. In practice a two cascaded control loop is often used instead of a single one for controlling the attitude, e.g., in [48, 49]. The reason is that it is more practical to separate the onboard processing on two independent units. One handles the state estimation and high-level control on the body rates while the second processing unit runs a low-level controller on the attitude.

Nevertheless, there are no stability and convergence guarantees when the attitude of the vehicle deviates substantially from hover conditions. These properties can be stated in the design of only a few controllers used for tracking these trajectories. [50] designs a nonlinear geometric controller in SO(3) which is almost global exponential stability for the load attitude tracking, and almost global exponential attractivity of the load position tracking. In [1] the system is underactuated; however, it is possible to design controllers that guarantee convergence from almost any point on SE(3). An other appropriate controller for aggressive manoeuvres is the one proposed by [51] that developed a nonlinear tracking controller on SE(3) and shown to have exponentially stable dynamics and almost globally exponential attractiveness of the complete dynamics (under some conditions and precise knowledge of the inertial parameters). This controller has been used for the simulations and the experiments presented in this thesis.

Control techniques can offer reasonably fast motions [30, 52] but generally lead to severe sub-optimalities when the system is subject to multiple nonlinear constraints. In this case, it appears to be more attractive to produce trajectories that fulfil a set

of boundary conditions and dynamic constraints, specific properties, and optimal criterion and then fed to an accurate trajectory tracker. Some work in this area has addressed complex and agile tasks such as aerobatic manoeuvres [29, 1] and ball catching [35]. Similar problems have been addressed using MPC [53, 54]. With these approaches, guarantees of convergence are only available when the linearised model is fully controllable [54] or if a control Lyapunov function can be synthesized [55]. As such it appears to be difficult to directly apply such techniques to the trajectory generation of a quadrotor. Learning algorithms have been successful in learning models [56], agile motions [10] or stabilization policies [57] using data from simulated and real world. Although very promising, these approaches do not appear to lend themselves (yet) to more general motion planning or trajectory generation, such as in environments with obstacles for instance.

2.3.2 Trajectory generation for a quadrotor

Once trajectory tracker algorithms are designed, the problem shifts to the higher level of task definition often assimilated into trajectory planning which is devoted to generate the reference inputs for the trajectory tracker.

In applications seeking agile manoeuvres, it is necessary to develop flight plans that leverage the dynamics of the system instead of simply viewing the dynamics as a constraint on the system. It is necessary to relax small angle assumptions and allow for significant excursions from the hover state. A recent focus has been the planning and following of trajectories that exploit the dynamical capabilities of these vehicles. Results include algorithms that plan trajectories from classes of motion primitives [6, 35], while others solve an optimal control problem for approximate or full vehicle dynamics (e.g. for minimum snap [1] or minimum time [58]).

Robust trajectory tracking is crucial especially for high speeds and accelerations. In [59] the authors chose to consider a more accurate dynamical model in the controller by incorporating the motor dynamics and aerodynamic drag effects. Indeed, identifying the model parameters and external disturbances will improve the controller performance. Similarly [60, 61] developed a controller compensating for aerodynamic effects and especially the drag. [60] was able to exhibit lower position errors even at flight speeds up to 18m/s in [21].

It is also important to account for the tracking precision and energy consumption when designing aggressive and complex trajectories. Otherwise the tasks may not be effectively completed due to instabilities or motor saturations.

A popular approach in robotics and first applied to manipulators is to produce *smooth* motions; i.e. trajectories with good continuity features; in particular, continuous velocity, accelerations and jerks in the interests of avoiding mechanical

resonance (e.g., for manipulators) and reducing stresses to the actuators and to the mechanical structure. Smooth trajectories will help performing aggressive and fast trajectories for a quadrotor while aiding the controller action. Finally, smoothness is desirable for maintaining the quality of onboard sensor measurements. Since vision is part of the planning scheme, smooth motions may help reducing *motion blur* in the image plane to facilitate visual tracking.

2.3.3 Smoothness

In this section we discuss the smoothness of motion in robotic applications. First of all, a movement is perceived to be smooth, when it happens in a continual fashion without any interruptions. It is closely related to effort minimization which is a major objective, especially in manufacturing for cost and ecological reasons, but it is also desirable for robots carrying limited energy source (e.g., robots for spatial and submarine exploration). Among others, these conclusions motivated the use of smooth trajectories to connect two states and was applied to robotics [62, 63] a few years later with robotic arms to name a few.

The resolution problem of optimal trajectory satisfying a smooth performance index is considered as an optimal control problem. So one of the keys in trajectory generation is the selection of an appropriate cost function.

As often research takes inspiration from direct observations from the nature. In [64] the authors observed that for reaching trajectories human appears to minimize the integral of the square of the norm of the jerk which is the time derivative of acceleration, hence, the third time derivative of position.

$$\dddot{\boldsymbol{r}} = \frac{d^3\boldsymbol{r}}{dt^3} \tag{2.17}$$

For a particular trajectory $x(t)$ that starts at t_0 and ends at time t_f, one can measure smoothness by calculating the jerk cost:

$$\int_{t=t_0}^{t_f} \|\dddot{x}(t)\|^2 dt \in \mathbb{R} \tag{2.18}$$

The derivative to minimize has motivated a large number of research especially in the neuroscience domain. These studies reveal some observations:

- the minimum jerk criterion does not produce acceleration jumps at the start and end points, while the minimum acceleration criterion does.
- it is related to the control effort minimization
- the jerk can be minimized independently on each axis [64]

Later [65] pushed the study to higher derivatives (snap: the fourth derivative of position, crackle: the fifth derivative of position, pop: the sixth derivative of position). They found that as the order of the derivative increased, the solution to the functional $x(t)$ approached a step function. It is indeed legitimate to ask ourselves: *What derivative to minimize ?*

- Acceleration is the simplest, but most naive to define as the goal, since it will imply the less possible thrust, thus constraining excessively the aggressiveness of the trajectory. Smooth trajectories are desirable, but with some aggressiveness to explore the time optimal possible trajectory
- Jerk is a better representative of the aggressiveness of the true system inputs [35] and, like the acceleration, has a direct link with thrust. Moreover [6] affirms that maintaining constraints on the acceleration and jerk leads to a continuous thrust during the manoeuvre, which is then supported by [66] when affirming that constraints on jerk are necessary for a smooth trajectory.
- Snap trajectories have also been proven effective to generate quadrotor trajectories [67], due to the linkage in the motors commands and body rate derivatives.

Trajectories that quadrotors can follow quickly and accurately should be at least continuous up to the third derivative of position (or C^3). This is because, for quadrotors, discontinuities in lateral acceleration require instantaneous changes in attitude and discontinuities in lateral jerk require instantaneous changes in angular velocity.

In the past 10 years this approach has been extended to quadrotors. Jerk is minimized in [68, 6, 69] where feasible trajectories are generated based on the decoupling of the rotational degrees of freedom. Analytic solutions for minimum jerk trajectories between collision-free points have been formulated in [70] using the Pontryagin's minimum principle (see e.g., [71]). In [1] the authors separated the optimal problem into four independent optimization problems and minimize the integral of the squared snap and the yaw acceleration since the inputs u_2 and u_3 are function of the fourth derivative of the positions and u_4 is function of the yaw angle second derivative.

[72] chose to optimize over the integral of the squared norm of the acceleration instead of snap, minimizing the energy that the considered helicopter needs. Compared to snap, acceleration directly translates into permanent additional thrust that all the motors have to provide, while snap just causes particular motors to spin up/down quickly. In[73] the jerk is minimized for a quadrotor and is more generally a choice for robot manipulators [65].

Minimizing the snap is present in many optimization frameworks in the literature (pioneered by [1], see also [74, 75]). Indeed, an even more important goal than finding minimum-time trajectories might be trajectory smoothness, especially for quadrotors. It plays a role in producing *safer* trajectories which will facilitate the trajectory controller action [76]. Although the first quantity acts on the optimal inputs, it mainly produces smooth trajectories for a quadrotor (by helping to reduce the angular acceleration) while using the actuation capability since the solution yields a trajectory with a larger peak speed relative to average speed (see [65] for more details).

Finally, in [67, 1] a optimal trajectory is computed to pass though a number of position keyframes in a continuous way instead of following straight lines which have an infinite curvature at the keyframes that would force the quadrotor to stop at each connection.

To summarize, there seems to be no good general answer on which method is to be favoured between acceleration, jerk or snap and this might depend on the application. The next section introduces the property of *differential flatness*. We show that the inputs are a function of fourth-order derivative of position. For this reason and as it is done in [1] we choose to minimize the snap.

2.4 State estimation

The challenge is now to get a reliable knowledge of the robot's state since the performance of the controller depends on the quality of the state estimate. Many robotic applications rely on external centralized localization systems such as Vicon or global positioning system (GPS) or full SLAM systems [77, 78]. However, GPS signals are only available outdoor and are not sufficiently reliable and precise enough for some specific tasks (e.g., involving interaction with the environment or navigation among obstacles). One may often need accurate knowledge of the state and a solution is to exploit local observations of the environment.

To obtain a reliable response from a quadrotor for experimental purposes some preliminary tasks are required: the calibration of the Inertial Measurement Unit (IMU), an identification of offsets, biases, motor curves, etc. Once all of this is completed (see e.g., [79] for further details on the procedure) state estimation algorithms are implemented. Their role is to provide a reliable estimate of the system state based on the outputs of a channel of proprioceptive sensors (e.g., accelerometer, gyroscope,...) and exteroceptive sensors (e.g., altimeter, cameras, lidar,...).

State estimation algorithms are generally based on multi-sensor fusion to combine the sensory measurements and properties of different sensors (e.g., acquisition

rate, robustness to some noise, weight,...). In aerial robotics the attitude and angular velocity are the most important as they are primary variable in attitude control of the vehicle. A very popular choice is to combine an IMU composed of an accelerometer and a gyroscope with a camera to merge high rate acceleration and angular velocity measurements (from the IMU) with lower rate visual cues. These visual measurements can be used for, e.g., position, orientation and velocity estimation from the environment or for visual odometry [80]. Sensor-fusion algorithms have been successfully applied in many works to provide full autonomy of the robotic platforms, e.g., [81, 82]. Fully autonomous high-speed navigation has only been achieved in the last few years (e.g., [21, 83]). These recent developments were largely supported by the improvements of sensors in terms of measurement accuracy, compactness and acquisition rate. Most of the approaches use Extended Kalman filters (EKF) for its robustness and simplicity. However, no guarantees of convergence and stability are given. These questions are addressed in [84] by the design of an observer endowed with exponential stability and convergence guarantees. The observer fuses optical flow with inertial measurements to estimate the attitude, the linear velocity and the depth of a camera observing a planar target.

2.5 Differential flatness

Planning trajectories in high dimensional space is challenging especially with an underactuated system. In this section we show how control and planning problems can be simplified without any additional approximations by using the fact that the quadrotor dynamics are *flat*. This property makes the trajectories design easier and guarantees the trajectories are feasible, i.e., trajectories that satisfy the equations of motion.

2.5.1 Definition and properties

Differential flatness was primarily introduced by Fliess [85] in a differential algebraic context aimed at nonlinear system [85]. Then Martin, Murray, Rouchon, Lévine and Van Nieuwstadt [86, 87] made further study about this theory and its implications in trajectory generation. They discovered the existence of a set of *flat outputs* with nonlinear dynamic characteristics that allow exact linearisation of particular nonlinear systems. In a nutshell, for a differentially flat system, all states and inputs can be expressed as algebraic functions of a set of outputs and their derivatives. More specifically, a nonlinear system:

$$\dot{x} = f(x, u), \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m \tag{2.19}$$

is termed flat if we can find outputs $\sigma \in \mathbb{R}^m$ of the form

$$\gamma = \xi(x, u, \dot{u}, \ldots, u^r) \quad \xi : \mathbb{R}^n \times (\mathbb{R}^m)^{r+1} \to \mathbb{R}^m \tag{2.20}$$

such that

$$x = \phi_x(\gamma, \dot{\gamma}, \ldots, \gamma^l) \quad \phi : (\mathbb{R}^m)^r \to \mathbb{R}^n \tag{2.21a}$$

$$u = \phi_u(\gamma, \dot{\gamma}, \ldots, \gamma^l) \quad \psi : (\mathbb{R}^m)^{r+1} \to \mathbb{R}^m \tag{2.21b}$$

where ξ, ϕ_x and ϕ_u are smooth functions and γ is called the flat outputs. This means that the new system's description is given by the m algebraic variables $\gamma_i, i = 1, \ldots, m$. So for a differentially flat system, if given a desired trajectory γ_d, we can obtain all expected states, inputs and outputs:

$$x_d = \phi_x(\gamma_d, \dot{\gamma_d}, \ldots, \gamma_d^l) \tag{2.22a}$$

$$u_d = \phi_u(\gamma_d, \dot{\gamma_d}, \ldots, \gamma_d^l) \tag{2.22b}$$

Note that ξ is bijective.

In [88] the authors presented a catalogue of flat systems in 1995 including non-holonomic mobile robots, the Planar Vertical Take-Off and Landing (PVTOL) aircraft, the inverted pendulum and the ducted fan. They also provided insights on determining if a system is differentially flat by considering its mechanical structure. Since then, new flat systems have emerged, for instance, the ballbot robot under small angles assumptions [89] and especially the quadrotor in [1] and later revised and extended with the consideration of rotor drag effects in [59]. Several "protocentric aerial manipulators" (systems where the first joint of the manipulator coincides with the quadrotor center of mass) were proven to be flat [90] as well as a quadrotor tethered by cables/bars [91]. It is known from [1] that the quadrotor dynamics (2.16) are flat with flat outputs $\boldsymbol{\gamma} = (\boldsymbol{r}, \psi)^T \in \mathbb{R}^4$ [1], where ψ is the yaw angle from the usual roll/pitch/yaw decomposition of the rotation matrix $\boldsymbol{R}$. Indeed, under the assumption $f > 0$, one can find an invertible algebraic mapping of the form:

$$\boldsymbol{\chi} = \phi_\chi(\boldsymbol{r}, \boldsymbol{v}, \dot{\boldsymbol{v}}, \ddot{\boldsymbol{v}}, \psi, \dot{\psi}) \tag{2.23a}$$

$$(f, \dot{f}, \ddot{f}, \boldsymbol{\tau}) = \phi_u(\dot{\boldsymbol{v}}, \ddot{\boldsymbol{v}}, \dddot{\boldsymbol{v}}, \psi, \dot{\psi}, \ddot{\psi}) \tag{2.23b}$$

We report the complete proof of the flat transformation and its inverse transformation for the quadrotor dynamics in Appendix A. For simplicity of notation, we indicate with $\boldsymbol{\sigma} = (\boldsymbol{r}, \boldsymbol{v}, \dot{\boldsymbol{v}}, \ddot{\boldsymbol{v}}, \dddot{\boldsymbol{v}}, \psi, \dot{\psi}, \ddot{\psi})$ the vector of all quantities appearing on the right side of (2.23), and with $\boldsymbol{\sigma}_\chi = (\boldsymbol{r}, \boldsymbol{v}, \dot{\boldsymbol{v}}, \ddot{\boldsymbol{v}}, \psi, \dot{\psi})$ only those involved in (2.23a).

The implications of flatness for all these systems is that the trajectory generation problem can be reduced to simple algebra, in theory, and computationally attractive algorithms in practice. For instance, in the case of the quadrotor the state space

of dimension 12 can be reduced to a 4-dimensional space in which the integration of equation (2.19) (often costly and numerically challenging step) is not necessary. Traditional approaches to trajectory generation, such as optimal control, cannot be easily applied in many cases (see [88] for examples). Since the flat output functions are completely free, the only constraints that must be satisfied are the initial and final conditions on the endpoints, their tangents, and higher order derivatives. Any other constraints on the system, such as bounds on the inputs, can be transformed into the at output space and (typically) become limits on the curvature or higher order derivative properties of the curve. Moreover, any curve that satisfies the boundary conditions in the flat output space is a trajectory of the original system.

Referring to Fig. 2.3, the problem of finding curves that take the system from $x(0)$, $u(0)$ to $x(T)$, $u(T)$ is reduced to finding any sufficiently smooth curve that satisfies $\gamma^k(0)$ and $\gamma^k(T)$ up to some finite number l. There is no need to solve a two-point boundary value problem (BVP) if the system is differentially flat. Once all the boundary conditions and trajectory constraints are mapped into the flat output space, (optimal) trajectories can be planned in the flat output space and then lifted back to the original state and input space with (2.22). The idea is that this methodology alleviates adjoining the system dynamics in the optimal control problem formulation. Consequently, the number of variables in the optimal control problem is reduced to expedite real-time computation. Therefore, by converting the input constraints on the quadrotor to constraints on the curvature and higher derivatives of the position and the yaw angle, it is possible to design efficient techniques for the generation of feasible trajectories.

2.5.2 Existence

Differentially flat systems encompass all linear, controllable systems and many nonlinear systems as well. Although there is no general methods to judge whether the system $\dot{x} = f(x, u), x \in \mathbb{R}^n, u \in \mathbb{R}^m$ is differentially flat and it is difficulty to find the flat outputs for most known differentially flat systems, some researchers still found and proved the existence of flat outputs of some systems (like the one cited in the previous section). While Fliess [85, 92] and Charlet [93] provided necessary conditions and sufficient conditions separately for a class of systems, Chetverikov is the first to show necessary and sufficient conditions [94]. Yet, one frequently has to resort to trial and error to construct the flat outputs. Flat outputs of a system are not unique [88], it is therefore preferable to select the flat outputs leading to simple computations for the mappings ϕ_x and ϕ_u. Their choice can also be motivated by the design of the control laws or the planning formulation into a reduced or more relevant space. In [95] the flat outputs are chosen as a set of image features to simplify the planning in the image space for a grasping task performed by a

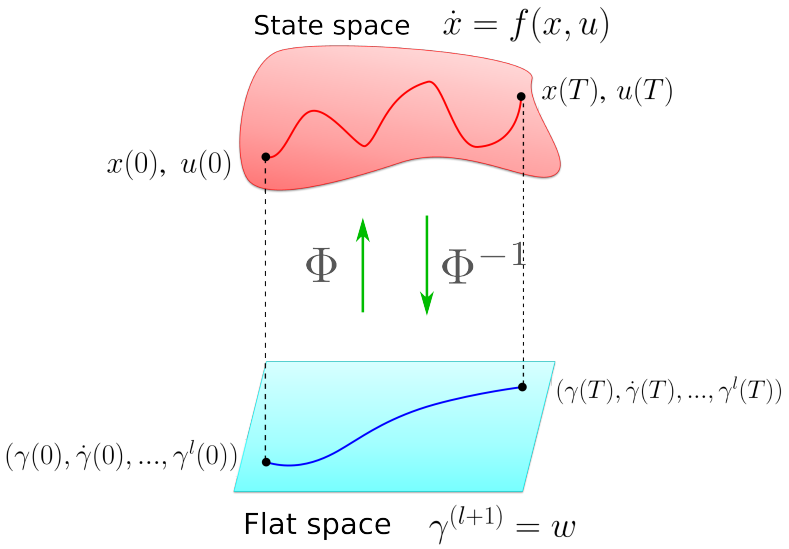


Figure 2.3 – Trajectories in the state space can be mapped in the *static* flat space (i.e., without dynamics) of lower dimension. Any smooth enough curve in the flat space will be feasible in the state space.

quadrotor.

In addition, a particular parametrization may also depend on the complexity of deriving the constraints from the outputs. However, it is generally recommended to use a parametrization that eliminates all equality constraints. Indeed, equality constraints are the most difficult to handle in nonlinear programming.

2.5.3 Differential flatness in control and trajectory planning

At this point, differential flatness plays a strictly practical role and quickly gained popularity for deriving control schemes and solving various optimal control problems as the state of the art in Chapt. 5 will show. Indeed, a transformation of the system into a linear equivalent description is obtained and then it is straightforward to design a controller based on linear control theory, e.g., [96] with disturbance rejection and [30] with the use of a LQR for controlling a hexacopter Fig. 2.4. Furthermore, classic polynomial control laws can be applied on the flat outputs and their derivatives and compared with the actual flat state measurements. Such a method has been efficiently validated in [37]. Such control laws can be adopted for tracking any trajectory $\boldsymbol{y}^d(t) = (\boldsymbol{\gamma}_d(t), \dot{\boldsymbol{\gamma}}_d(t), \ldots, \boldsymbol{\gamma}_d^{(l)}(t))$ directly in the space of the Brunovsky states by considering the new control inputs $\boldsymbol{\omega} = \boldsymbol{\gamma}^{(l+1)}$. Then, the real world inputs $\boldsymbol{u}$ are obtained via dynamic feedback and applied to the real system.

This strategy was adopted in e.g., [97] to design a visual-based controller for a UGV and [98] for controlling a tethered aerial robot. Asymptotical convergence can be achieved with an appropriate choice of the gains that can be determined by

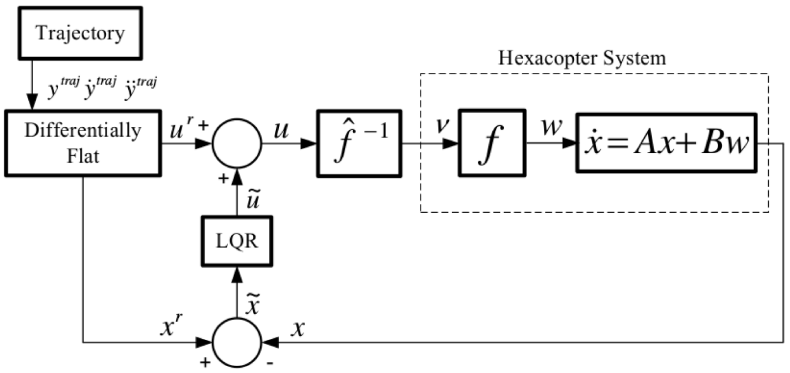


Figure 2.4 – System architecture for LQR control of a hexacopter. A function ($\hat{f}^{-1}$) converts the inputs u to the actual commands ν fed to the real system (commanded in acceleration with w) from [30].

pole-placement techniques to ensure good tracking and some robustness to model uncertainties as well. However, in general, after a given order $\underline{l} \leqslant l$ one may loose the capacity to observe the higher-order derivatives of the flat outputs. Typically, one resort to designing observers as also shown in [98, 37].

A similar strategy is found in optimization-based control: the optimal control outputs are computed in the flat space which are then lifted back to the space of the real control inputs, see e.g., [99].

As already discussed in Sect. 2.3 from a control perspective, most early research on quadrotor dynamics focused on near-hover operation. Now, in the context of fast motion, differential flatness has been considered as a strong system property that can be used for generating dynamically feasible trajectories for underactuated robotic systems leading to significant progress both in trajectory generation and control of quadrotor systems in the recent years (e.g., [1, 100, 101, 30] to name a few).

In [102] a comparison is proposed between differential flatness and dynamic feedback linearisation in motion planning. Indeed, the two properties are equivalent in the sense that any feedback linearisable system is also differentially flat and vice versa as it is demonstrated in [92]. Moreover, the feedback linearising outputs and the flat outputs of a system coincide.

2.6 Vision-based control

To be reactive to changes in the environment, the use of on-board cameras has become a fundamental necessity. Indeed, moderately invasive and low power, they come with numerous computer vision algorithms for tracking objects, mapping,

detecting obstacles but also for giving various measurements to non-linear observers (e.g., through active vision, state estimation).

For these reasons, incorporating visual cues in the loop has become a popular approach in robotic control for many years for designing robust and reactive control laws to complete positioning [103], grasping tasks [95] or navigation among obstacles [104] to name a few. However, vision is not without its challenges. Computation is intense and can result in low sample rate. Vision-based control techniques may be deceived by ambiguities between certain camera motions and scale since there exists a problem with scale recovery when using a single camera. Finally, they can suffer from delays between the image frame capture, transmission and processing. However, processors continue to improve and many vision-based autonomous applications are flourishing [24, 105, 106, 107].

2.7 Issues related to vision-based control

A historical technique is *visual servoing* [28]. Visual servoing is the fusion of results from many elemental areas including high-speed image processing, kinematics, dynamics, control theory, and real-time computing. This popular control scheme uses visual cues for *directly* controlling the robot's motion and referred as Image-Based Visual Servoing (IBVS). This strategy was originally developed in the context of industrial robots (see e.g., [108] for a survey on the topic), which are usually equipped with low-level high-gain control loops that allow neglecting the dynamics of the platform and, e.g., controlling it at the velocity level. Unfortunately, such simplification cannot be extended to quadrotors away from near-hovering conditions. Since quadrotors show non-negligible dynamics the visual control problem becomes significantly more complex. This is due to the inherent under-actuation that generates rotations of the camera which may conflict with the main servoing task (as detailed in the Introduction). Intuitively, in order to overcome this problem, the robot can either limit its rotational motion (thus reducing the acceleration and increasing the time needed to reach the desired position) or compensate the rotation by also moving upwards for increasing the size of the scene projected within the camera field of view.

Standard visual servoing approaches for underactuated systems, such as quadrotors, oftentimes do not explicitly ensure that the relevant image features stay in the camera's field of view, especially while the system is performing agile manoeuvres.

Preserving visibility is a substantial concern when vision is in the loop since losing track of features may lead to a failure of the task. The original formulation of visual servoing does not prevent critical configurations related to:

- field of view limits, i.e., the target may become invisible as the camera moves;
- occlusions, i.e., the image features may be occluded due to obstacles or body parts;
- singularities, i.e., specific features configurations can lead to an ill-conditioning of the interaction matrix (e.g., the cylinder singularity for three points);
- local minima, i.e., the control law may lead to a convergence to unexpected configurations. Local minima may appear with the use of redundant measurements that is the usual approach to avoid singularities;

Finally, one can complete the above list with issues related to aggressive motion planning:

- physical constraints such as feasibility constraints;
- camera underactuation, i.e., when a camera is attached to an underactuated robot such as a quadrotor the image features are more subject to being repelled from the camera center as the quadrotor perform translational motions.
- motion blur, i.e., the aggressiveness of the camera motion (especially with high angular accelerations) may lead to poor quality of the image frames that can jeopardize the vision algorithms.
- collision with obstacles or self-collision;

Note that two mechanical solutions could of course help reducing the effect of underactuation: 1) a pan-tilt camera could be mounted instead of a fixed camera [109]. Apart from the additional payload and extra consumption, sensing is still limited and one cannot guarantee that all visual features will remain visible. 2) fully actuated quadrotors with tilting propellers [15, 14] have been recently developed to gain full access to the 6 DOFs (see, e.g. [110, 111] for visual controllers). Therefore, a fixed camera would be less subject to rotation motions but still, if aggressive motions are performed rotations are possible.

In this thesis we provide optimization-based solutions ensuring visibility of visual features. Moreover, in Sect. 7.6 and Chapt. 8 we present solutions for relaxing the visibility constraints in complex environments.

Chapter 3

Optimization and numerical resolution

Contents

3.1 Introduction to optimization

In many occasions, the trajectory generation problem cannot be solved analytically. An exception is formed by linear systems. For general systems we can only solve the generation problem by repeatedly integrating the system equations and trying to minimize some errors between the computed trajectory and the desired trajectory. The resolution of more and more complex problems may not be possible by analytic resolution techniques. Moreover, we believe they may be too difficult and not generic enough. In this section we introduce optimization and especially direct optimization which we believe represents a more appropriate approach for the resolution of complex problems. The idea is to compute a finite sequence of optimal controls and states over a time horizon as a numerical approximation of the system

dynamics. Optimization is a well-understood field and is able to exhibit valid inputs and trajectories adapted to changes in the task, the environment and system dynamics.

3.2 Minimum-time trajectory generation problem

A very specific trajectory generation problem in robotics is the generation of time-optimal trajectories between two states. The problem of connecting a given initial state $\boldsymbol{\chi}_0$ at time t and a chosen final state $\boldsymbol{\chi}_f$ at a time $t+T$ can be formulated as that of constructing a feasible trajectory

$$\boldsymbol{\chi}(t)^* : [t, t+T] \rightarrow \boldsymbol{X} \tag{3.1}$$

for the state of the quadrotor where $\boldsymbol{X}$ denotes the state space, and T defines the trajectory duration. Now, an infinite number of trajectories can connect these two states. Since one seeks the generation of agile manoeuvres, we are particularly interested in minimizing the completion time T. Using an optimal formulation we define Problem 1 where $J(\boldsymbol{\chi}(.), \boldsymbol{u}(.)) \in \mathbb{R}$ defines the *cost function* or *objective function* that we want to minimize. Given the dynamic model (3.2d) at a generic time t, we seek for a solution to the following optimization problem.

Problem 1 *Find* $\boldsymbol{\chi}(s), \boldsymbol{u}(s), T, s \in [t, t+T]$, *such that:*

$$\min_{\boldsymbol{\chi}(s), \boldsymbol{u}(s), T} \quad J(\boldsymbol{\chi}(s), \boldsymbol{u}(s)) = T \tag{3.2a}$$

$$s.t. \quad \boldsymbol{\chi}(t) = \boldsymbol{\chi}_t, \tag{3.2b}$$

$$\boldsymbol{\chi}(t+T) = \boldsymbol{\chi}^*, \tag{3.2c}$$

$$\dot{\boldsymbol{\chi}} = \boldsymbol{h}(\boldsymbol{\chi}, \boldsymbol{u}), \tag{3.2d}$$

$$\boldsymbol{u}(s) \in \mathcal{U}, \forall s \in [t, t+T] \tag{3.2e}$$

where $\boldsymbol{\chi}_t$ is the current robot state, $\boldsymbol{\chi}^*$ is the desired one, and (3.2d) was introduced to represent (2.16) in a compact form.

Note that Problem 1 is quite general. In particular, it does not impose any constraint on the initial and final states which, e.g., do not have to be *hovering* states (i.e. with $\boldsymbol{R} = \mathbf{I}$, $\boldsymbol{v} \equiv \mathbf{0}$, and $\boldsymbol{\omega} \equiv \mathbf{0}$).

We also want to find a feasible trajectory for the quadrotor. This is encoded with (3.2d) that imposes the trajectory to respect the nonlinear dynamics equations (2.16).

The robot is finally subject to (nonlinear) inputs constraints (3.2e) due to the physical limits of its actuators. These constraints act on the minimum and maximum values of the system inputs. Vector $\boldsymbol{u}$ contains then the propellers individual

thrust ($\boldsymbol{u} = (f_1, f_2, f_3, f_4) \in \mathbb{R}^4$). Note that, since the propellers can not change their direction of rotation during the motion, we have to assume $f_i > 0$.

The set of admissible control inputs is the box $\mathcal{U} = [f_m, f_M]^4$ with $0 < f_m < f_M$ where f_m and f_M are determined by the physical characteristics of the motor, the available power, propeller, etc. One can prove that Problem 1 is solvable [1, 112], i.e., there always exist a parameter T such that (3.2e) are satisfied (with a reasonable choice of f_m and f_M).

3.3 Pontryagin's minimum principle

The problem of generating optimal trajectories for a quadrotor UAV has been addressed with analytic resolution [69, 58] using Pontryagin's minimum principle. This optimal control theory principle has been formulated by Lev Semenovich Pontryagin and his students in 1956 and defines a necessary, but not sufficient, condition for optimality of a system trajectory. The problem is a generalization of the Euler-Lagrange equations that also includes problems with constraints on the control inputs and applies to a large class of control problems.

Let us assume that we want to find a trajectory for the state and the input

$$\boldsymbol{\chi}(t)^* : [t, t+T] \rightarrow \boldsymbol{X} \tag{3.3a}$$

$$\boldsymbol{u}(t)^* : [t, t+T] \rightarrow \mathcal{U} \tag{3.3b}$$

that minimizes the cost function

$$J = \Phi(\boldsymbol{\chi}(t+T)^*) + \int_t^{t+T} F\left(\boldsymbol{\chi}(t)^*, \boldsymbol{u}(t)^*\right) dt \tag{3.4}$$

subject to

$$\dot{\boldsymbol{\chi}} = \boldsymbol{f}(\boldsymbol{\chi}(t), \boldsymbol{u}(t)), \quad \boldsymbol{\chi}(t)^* = \boldsymbol{\chi}_0, \quad \boldsymbol{\chi}(t+T)^* = \boldsymbol{\chi}_f \tag{3.5}$$

Neglecting the time dependencies, we define the *Hamiltonian* of the system as

$$H(\boldsymbol{\chi}, \boldsymbol{u}, \boldsymbol{p}) = F\left(\boldsymbol{\chi}, \boldsymbol{u}\right) + \boldsymbol{p}^T \boldsymbol{f}\left(\boldsymbol{\chi}, \boldsymbol{u}\right) \tag{3.6}$$

where $\boldsymbol{p}$ is also called the *costate* vector and plays a similar role to the Lagrange multipliers.

The Pontryagin's principle states that if $\boldsymbol{u}(t)^*$ is an optimal trajectory for the input and $\boldsymbol{\chi}(t)^*$ is the corresponding optimal trajectory for the state, then the following conditions hold

$$\dot{\boldsymbol{\chi}}(t)^* = \boldsymbol{f}\left(\boldsymbol{\chi}(t)^*, \boldsymbol{u}(t)^*\right) \tag{3.7a}$$

$$\dot{\boldsymbol{\chi}}(t)^* = \boldsymbol{\chi}_0 \tag{3.7b}$$

$$\dot{\boldsymbol{\chi}}(t+T)^* = \boldsymbol{\chi}_f \tag{3.7c}$$

$$\dot{\boldsymbol{p}}(t) = -\nabla_{\boldsymbol{\chi}} H\left(\boldsymbol{\chi}(t)^*, \boldsymbol{u}(t)^*, \boldsymbol{p}(t)\right) \tag{3.7d}$$

and for all $t \in [t, t+T]$

$$\dot{\boldsymbol{u}}(t)^* = \underset{\boldsymbol{u}\in\mathcal{U}}{\operatorname{argmin}} H(\boldsymbol{\chi}(t)^*, \boldsymbol{u}, \boldsymbol{p}(t)) \tag{3.8}$$

Moreover if the total time t_f is not fixed by the problem, the following condition also holds true

$$H(\boldsymbol{\chi}(t)^*, \boldsymbol{u}(t)^*, \boldsymbol{p}(t)) \equiv 0 \tag{3.9}$$

For a detailed explanation of the Pontryagin's principle, refer to e.g. [71]. It has hence been shown that the solution to the minimum-time problem is generally a bang-bang control policy [113], that is, a control policy in which the control signal switches between two or several extreme values. Pontryagin's minimum principle is exploited in [6, 69, 114] and [9] to generate minimum-time trajectories for a quadrotor. For instance [6] generates minimum-time interception trajectories for aggressively catching a ball in mid-air: bang-singular trajectories where the goal is to reach a given position at a given time, while minimizing the time required to stop after the intercept. These methods are computationally fast, with solution times on the order of microseconds which is compatible for closed-loop control.

However, these strategies are not able to account for geometric constraints and are independent of the yaw angle in order to decouple the quadrotor axes. In this thesis, we consider more complex constraints such as visibility constraints that are not compatible to Pontryagin's minimum principle as far as we know.

In the next section we show how an optimal control problem (OCP) can be turned into a nonlinear program (NLP) that is suited for numerical resolution.

3.4 Numerical solutions of optimal control problems using nonlinear programming

Nonlinear optimization describes the class of optimization problems when the objective or constraint functions are not linear and not known to be convex as well. These problems are considered as much more complex and difficult to solve and there is no effective method of solving them. However, there exist different approaches to their resolution that involve some compromises.

The problem of finding a local minimizer $x \in \mathbb{R}^n$ for a nonlinear function $F(x)$ subject to a set of nonlinear constraints $c \geqslant 0$, where $c(x) \in \mathbb{R}^n$, is a *nonlinear constrained* optimization problem. All the problems of interest to be solved in this thesis can be generalized into the form

Problem 2

$$\min_{x} \quad F(x) \tag{3.10a}$$

$$s.t. \quad c(x) \geqslant 0 \tag{3.10b}$$

Optimization problems of the form of Problem 2 can be a very difficult problem to solve. Algorithms to solve this problem may take many iterations and function evaluations. Moreover, *global* optimization of Problem 2 is a difficult problem and an open area of research. In this thesis, we will concentrate on using the well understood numerical techniques that will find *local minimum*.

Nearly all techniques for nonlinear programming are iterative, producing a sequence of subproblems related in some way to the original problem. Newton methods have rapid local convergence rates, but fail to converge from all starting points. Gradient descent methods converge from nearly any starting point but have poor local convergence properties. *Line-search* methods are one means of ensuring global convergence while attempting to maintain fast local convergence. Line-search methods limit the size of the step taken from the current point to the next iterate. Such methods generate a sequence of iterates of the form

$$x_{k+1} = x_k + \alpha p \tag{3.11}$$

where p is the search direction obtained from the subproblem, and α is a positive scalar *step-length* that has to be chosen carefully. However, determining a minimizer along p is an iterative process and frequently time consuming. Typically, x is determined by a finite process that ensures a reduction in $F(x)$. See [115] for an overview of line-search methods.

Two very different approaches may be considered to solve Problem 2, the *indirect* one and the *direct* one.

3.4.1 Indirect and direct methods for nonlinear programming

Most early numerical methods of solution to constrained optimal trajectory generation problems relied on either indirect or direct methods of solution. The indirect method relies on finding a solution to the Pontryagin's maximum principle presented earlier. Indirect methods turn the problem into an integration problem consisting of ordinary differential equation (ODE) or differential-algebraic equation (DAE). The resulting problem is a differential equation which is unfortunately often too complex to be integrated as is. When it is possible, this approach provides a complete (and often comparatively cheap) solution to the problem. However, this type of approach is usually applied on a specific system and/or task so the differential equation can be simplified enough to be integrated.

The direct method obtains solutions by direct minimization of the objective function, subject to the constraints of the optimal control problem. In the direct approach, the optimal control problem is transformed into a NLP. In a first approach, this can be done with the so-called direct *single–* and particularly– *multiple shooting* methods. The key strategy is to divide the time vector, state and control trajectories into a finite grid. Therefore, the direct approach directly solves a discretized approximation of the nominal problem using numerical optimization techniques. This allows turning an optimization problem of an infinite dimension (the search space is infinite dimensional) to a finite one in order to be efficiently solved by selecting outputs from a finite dimensional space. This results in finding a numerical solution to a *two-point boundary value problem* (BVP), if no closed form solution can be found. Examples along this line can be found in [116, 117]. These methods normally cannot meet the performance requirement for on-line calculation, especially when the system manoeuvring time is short. In addition, the optimal trajectories represented by the discrete collocation points are not continuous or smooth curves.

Is is known from e.g., [118] that direct methods are generally less precise but more robust to the initial solution guess than indirect methods. However, it appears that the computational requirements of direct methods are at least that of indirect methods. The collocation method of [119] and adjoint method [120] take part of the most relevant transcription methods to the trajectory generation problem.

3.4.2 Nonlinear solvers

Now that we showed how OCPs are discretized to obtain a structured NLP in non-convex form one has to select a nonlinear solver. Sequential Quadratic Programming (SQP) and Interior Point Methods (IPM) are popular *gradient-based* classes of methods considered to be effective and reliable for locally solving (3.10b). These methods are guided by the first- and second- order derivatives, i.e., the *gradients* and the *Hessian* matrix. Interior point refers to the fact that the slack variables are required to remain strictly positive throughout the optimization (more can be found in [121]). SQP ([122, 123]) is the technique we will use to solve the nonlinear programming problems presented in this thesis. The fundamental approach SQP is to solve a NLP by solving a sequence of quadratic programs (QP) that are easy to solve. More precisely, at each iteration of a SQP, one solves a QP subproblem that models Problem 2 locally at the current iterate. The solution to the QP is used as a search direction by a line-search algorithm to determine the next iterate.

SQP is known for its rapid convergence (a few SQP iterations) when iterating from an initial point (or *initial guess*) that is close to a (local) minimum but may

show erratic behaviour when the initial point is far. Moreover, SQP is not a feasible method; that is, neither the initial guess nor any of the subsequent iterations need to be *feasible* (a feasible point that satisfies the constraints). This is a major advantage since finding a feasible point when there are nonlinear constraints may be nearly as hard as solving the NLP itself. However, converging to a minimum would generally require more iterations than starting from a feasible point.

SQP solvers differ in the way the Hessian is approximated, the line-search is done, the QP subproblems are solved or the constraints are relaxed. SQP has been shown a powerful tool and because of its superlinear convergence rate and its ability to deal with nonlinear constraints. It is currently considered as one of the most powerful algorithm to solve numerous formulations of NLP.

Note that the gradients have to be supplied and their accuracy is crucial for local convergence. In principle the gradients can almost always be computed using very little additional computational effort. In practice, and especially with highly nonlinear programs analytic formulation of the Jacobians can be very complex, subject to errors and finally hard to code. We will see in Appendix C that there exist efficient numerical methods for accurately evaluating these functions.

The Hessian is even more complex and approximation methods exist such as the Broyden-Fletcher-Goldfarb-Shanno method (BFGS) [124]. This method builds an approximation of the Hessian based on successive gradient evaluations that are stored over for a given horizon. This method has proven to have a good performance even for non-smooth optimizations. Thus, the Hessian matrix will help accelerating local optimization.

Both SQP and IPM methods iterate from an initial guess for the optimization variables. This initial guess is therefore critical and has huge impact of the objective value of the local solution obtained. Using local optimization methods, one often has to resort to experimenting with the algorithm choice, parameters and initial guess. The methodology is not rigorously defined.

In the next section we show how Problem 1 can be turned into a NLP using differential flatness and parametrization with B-spline curves. The described method is the one we use in this thesis.

3.5 Differential flatness and B-spline curves for nonlinear programming

Referring to Sect. 2.5.1, we show how an optimal control problem can be reposed to allow direct optimization to occur within the output space (of the flat outputs) as opposed to the control space. First of all, we map the problem to the space of the

flat outputs and their derivatives that we indicate with $\boldsymbol{\sigma} = (\boldsymbol{r}, \boldsymbol{v}, \dot{\boldsymbol{v}}, \ddot{\boldsymbol{v}}, \dddot{\boldsymbol{v}}, \psi, \dot{\psi}, \ddot{\psi})$. Thanks to differential flatness, one can move the planning problem from the input space to the flat output space (i.e., the problem becomes a static problem): any sufficiently smooth trajectory of the flat outputs is, in fact, guaranteed to be an admissible trajectory for the original system dynamics. This property is extremely interesting for our purposes because it allows avoiding to deal with the non-linear differential equality constraint (3.2d), which would require the numerical integration of the system dynamics during the numerical optimization phase. A prediction of the state at any time in $[t_0, t_f]$ can, instead, be computed *algebraically* from the planned flat-output trajectory. For these reasons differential flatness has been widely used for trajectory planning in the past [125, 1, 112].

Mapping Problem 2 into the flat space gives the equivalent following problem

Problem 3

$$\min_{\boldsymbol{\sigma}(.)} \quad L(\boldsymbol{\sigma}) \tag{3.12a}$$

$$s.t. \quad g(\boldsymbol{\sigma}) \geqslant 0 \tag{3.12b}$$

Now, in solving Problem 3, we face two challenges: (i) instead of a finite set of variables, the optimization variable is a function $\boldsymbol{\sigma}(.)$ and (ii) the constraints must be enforced at all time instances. Therefore, the problem is infinite dimensional with an infinite number of constraints. To cope with the infinite dimensionality $\boldsymbol{\sigma}(.)$ is usually approximated with fixed parametric curves defined by a finite set of variables, a technique known as *parametrization*.

3.5.1 Parametrization of the flat outputs

There are many curves defined by a finite number of variables that can be used to approximate the outputs $\boldsymbol{\sigma}(.)$ (Fourier series, Legendre polynomials, Laguerre polynomials, Chebyshev polynomials, Taylor series, etc.). Now, a requirement is to accurately represent a basis of a trajectory with a reasonable number of decision variables that will constitute the degrees of freedom of the solver. A second important requirement of the curve is the ability to set a level of continuity C^k, without adding additional constraints. Specifying the level of continuity is necessary, since the states and inputs are a function of the outputs and their derivatives. A high degree single polynomial would be necessary to satisfy complex constraints but solving for the coefficients of high degree curves can be an inefficient and ill-conditioned operation. Finally, when a high number of basis functions is desired in order to satisfy multiple conditions still leaving some room for optimization, polynomial functions are not a good choice. Indeed to increase the number of parameters in a polynomial we need to increase its degree. *Local support* is also a desirable property of the basis

functions. Local support means that the curves only influence the curve locally to the current point of interest which is also favourable for numerically stable computer implementation.

A solution that meets the main requirements are Bézier polynomials or B-splines [126]. An exhaustive introduction on B-spline is given in Appendix B including the many interesting properties and manipulation algorithms these curves possess. These functions are obtained as a composition of a certain number of polynomials, each of whom is defined in a limited sub-domain of the overall function domain. The advantage of this solution is that we can increase the number of curve coefficients by increasing the number of polynomial components, while maintaining a low degree of the single polynomials. In particular a spline is said to be of degree p if it is composed by polynomials of degree p. A B-spline curve is constructed from Bézier curves joined together with a prescribed level of continuity between them. The points at which the curves are joined are called the *breakpoints* and are constructed so that they join with some level of continuity. The breakpoints are a strictly increasing sequence of real numbers. A non-decreasing sequence of real numbers containing $K+1$ breakpoints $\boldsymbol{U} = (u_0, ..., u_K)$ is called the *knot vector*. A breakpoint may appear multiple times in the interior of a knot vector and be referred as a breakpoint of *multiplicity* m. A recurrence relation is used to define the B-spline basis functions $B_{i,j}$ of the B-spline curves:

$$s(u) = \sum_{i=1}^{n} B_{i,p}(u)\boldsymbol{P}_i \quad n \geqslant k-1 \tag{3.13}$$

where $\boldsymbol{P}_i$ are the *control points* and the $B_{i,p}$ are *piecewise polynomial functions* of degree p (and order $k = p+1$) forming a basis for the vector space of all piecewise polynomial functions of the desired degree and continuity. Given the knot vector $\boldsymbol{U}$ and the degree p, the B-spline basis functions are defined by:

$$\begin{aligned} B_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \leqslant u \leqslant u_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ B_{i,p}(u) &= \frac{u-u_i}{u_{i+p}-u_i} B_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} B_{i+1,p-1}(u) \end{aligned} \tag{3.14}$$

Now, in the case of the quadrotor, given a vector of control points $\boldsymbol{P} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{n_r}, \psi_1, \ldots, \psi_{n_\psi}) \in \mathbb{R}^{3n_r+n_\psi}$, and two (fixed) normalized knot vectors $\boldsymbol{U}_p \in [0,1]^{K_p}, \boldsymbol{U}_\psi \in [0,1]^{K_\psi}$, the flat output trajectories can be represented as:

$$\begin{cases} \boldsymbol{r}(s) = \sum_{i=1}^{n_r} B_{i,k_r}\left(\frac{s-t}{T}\right)\boldsymbol{r}_i \\ \psi(s) = \sum_{i=1}^{n_\psi} B_{i,k_\psi}\left(\frac{s-t}{T}\right)\psi_i \end{cases}, \forall s \in [t, t+T], \tag{3.15}$$

where $B_{i,k}$ is the i-th B-spline basis function of order k, which can be computed recursively as described in [85].

Given (2.23), in order to ensure state continuity and input boundedness, one has to guarantee Lipschitz continuity of $\ddot{\boldsymbol{v}}$ and $\dot{\psi}$ (and continuity of lower order derivatives). This condition can be met by using *open-uniform* distributions of $K = n + k$ knots (i.e. $u_i = 0$, for $i = 1, \dots, k$, $u_i = 1$, for $i = n, \dots, K$, and $u_k, \dots, u_n$ equally spaced in $[0, 1]$) and by taking $k = k_r = 4$ for $\boldsymbol{r}$ and $k = k_\psi = 2$ for ψ.

Problem 1 can, finally, be restated as a NLP as follows.

Problem 4 *Find $\boldsymbol{P}, T$, such that:*

$$\min_{\boldsymbol{P},T} \quad T \tag{3.16a}$$

$$s.t. \quad \left\langle \boldsymbol{P}, \boldsymbol{B}_{k_r}^{d_r}(t), \boldsymbol{B}_{k_\psi}^{d_\psi}(t) \right\rangle = \boldsymbol{\sigma}_{\chi_t}, \tag{3.16b}$$

$$\left\langle \boldsymbol{P}, \boldsymbol{B}_{k_r}^{d_r}(t+T), \boldsymbol{B}_{k_\psi}^{d_\psi}(t+T) \right\rangle = \boldsymbol{\sigma}_{\chi^*} \tag{3.16c}$$

$$\boldsymbol{u}\left(\boldsymbol{P}, \boldsymbol{B}_{k_r}^{d_r}(s), \boldsymbol{B}_{k_\psi}^{d_\psi}(s)\right) \in \mathcal{U}, \ \forall s \in [t, t+T] \tag{3.16d}$$

where $\boldsymbol{B}_k^d(s) \in \mathbb{R}^n$ is d-th order derivative B-spline basis of order k evaluated at $s \in [t, t+T]$. The above formulation is adopted in our works to numerically solve optimal control problems with different costs and constraints.

Although, the system nonlinear dynamics equality constraints (2.16) become transparent due to the flatness transformation, their nonlinearities are in fact transferred to the other constraints, here, the real inputs constraints (3.16d). Indeed, the inputs obtained with $\boldsymbol{u} = \phi_u(\boldsymbol{r}, \dot{\boldsymbol{r}}, \ddot{\boldsymbol{r}}, \boldsymbol{r}^{(3)}, \boldsymbol{r}^{(4)}, \psi, \dot{\psi}, \ddot{\psi})$ are more complex to evaluate than in their original formulation (2.13).

As already said, to guarantee the continuity of the state, the position must be continuous up to the third order of derivation while the yaw angle must be continuous up to the first order. To keep the degree of the spline as low as possible we use two different splines: one for the position vector and an other (scalar) one for the yaw angle. The parameter s can be directly equal to the time and we will choose the knot vector so that all the internal nodes have multiplicity 1. For a 4-th order *clamped* B-spline with n control points, the knot vector is

$$\boldsymbol{U} = \left(0, 0, 0, 0, 0, \frac{1}{(n-k)}, \dots, \frac{n-k-1}{(n-k)}, 1, 1, 1, 1, 1\right) \tag{3.17}$$

The number n of control points obviously depends on the number of conditions that we want to impose to the spline and on the redundancy we want to keep for

further optimization. For each of the two connecting trajectories we must satisfy boundary conditions determined by the initial and final states. Also in this case the continuity of the state is guaranteed by the continuity of the position up to the third order of derivation and of the yaw angle up to the first order of derivation. This results in a total amount of 8 conditions on the position spline and 4 conditions on the yaw spline. Therefore, in order to satisfy these conditions, we need at least eight control points for the position ($n_r = 8$) and four control points for the yaw ($n_\psi = 4$). If we choose these values we end up with two square linear systems in the control points that can be conveniently written in a matrix form

$$\boldsymbol{A}_{\boldsymbol{r}_\mathcal{B}} \boldsymbol{P}_{\boldsymbol{r}_\mathcal{B}} = \boldsymbol{B}_{\boldsymbol{r}_\mathcal{B}}, \tag{3.18a}$$

$$\boldsymbol{A}_\psi \boldsymbol{p}_\psi = \boldsymbol{b}_\psi \tag{3.18b}$$

where the system variables are

$$\boldsymbol{P}_{\boldsymbol{r}_\mathcal{B}} = \begin{pmatrix} \boldsymbol{p}_{\boldsymbol{r}_\mathcal{B},1}^T \\ \boldsymbol{p}_{\boldsymbol{r}_\mathcal{B},2}^T \\ \vdots \\ \boldsymbol{p}_{\boldsymbol{r}_\mathcal{B},8}^T \end{pmatrix}, \quad \boldsymbol{p}_\psi = \begin{pmatrix} p_{\psi,1} \\ p_{\psi,2} \\ \vdots \\ p_{\psi,4} \end{pmatrix} \tag{3.19}$$

The coefficients matrices $\boldsymbol{A}_{\boldsymbol{r}_\mathcal{B}}$ and $\boldsymbol{A}_\psi$ contain the values of the B-spline basis functions and their derivatives at the initial and final times:

$$\boldsymbol{A}_{\boldsymbol{r}_\mathcal{B}} = \begin{pmatrix} B_{1,1}(t_0) & B_{2,1}(t_0) & \dots & B_{8,1}(t_0) \\ B_{1,2}^{(1)}(t_0) & B_{2,2}^{(1)}(t_0) & \dots & B_{8,2}^{(1)}(t_0) \\ B_{1,3}^{(2)}(t_0) & B_{2,3}^{(2)}(t_0) & \dots & B_{8,3}^{(2)}(t_0) \\ B_{1,4}^{(3)}(t_0) & B_{2,4}^{(3)}(t_0) & \dots & B_{8,4}^{(3)}(t_0) \\ B_{1,1}(t_f) & B_{2,1}(t_f) & \dots & B_{8,1}(t_f) \\ B_{1,2}^{(1)}(t_f) & B_{2,2}^{(1)}(t_f) & \dots & B_{8,2}^{(1)}(t_f) \\ B_{1,3}^{(2)}(t_f) & B_{2,3}^{(2)}(t_f) & \dots & B_{8,3}^{(2)}(t_f) \\ B_{1,4}^{(3)}(t_f) & B_{2,4}^{(3)}(t_f) & \dots & B_{8,4}^{(3)}(t_f) \end{pmatrix} \tag{3.20}$$

and

$$\boldsymbol{A}_\psi = \begin{pmatrix} B_{1,1}(t_0) & B_{2,1}(t_0) & B_{3,1}(t_0) & B_{4,1}(t_0) \\ B_{1,2}^{(1)}(t_0) & B_{2,2}^{(1)}(t_0) & B_{3,2}^{(1)}(t_0) & B_{4,2}^{(1)}(t_0) \\ B_{1,1}(t_f) & B_{2,1}(t_f) & B_{3,1}(t_f) & B_{4,1}(t_f) \\ B_{1,2}^{(1)}(t_f) & B_{2,2}^{(1)}(t_f) & B_{3,2}^{(1)}(t_f) & B_{4,2}^{(1)}(t_f) \end{pmatrix} \tag{3.21}$$

Finally the known terms are determined by transforming the boundary conditions

$\boldsymbol{\chi}_0$ and $\boldsymbol{\chi}_f$ into the equivalent conditions on the flat outputs and their derivatives

$$\boldsymbol{B}_{\boldsymbol{r}_\mathcal{B}} = \begin{pmatrix} \boldsymbol{r}_{\mathcal{B}_0}^T \\ \boldsymbol{v}_{\mathcal{B}_0}^T \\ \ddot{\boldsymbol{r}}_{\mathcal{B}_0}^T \\ \dot{\boldsymbol{a}}_{\mathcal{B}_0}^T \\ \boldsymbol{r}_{\mathcal{B}_f}^T \\ \boldsymbol{v}_{\mathcal{B}_f}^T \\ \ddot{\boldsymbol{r}}_{\mathcal{B}_f}^T \\ \dot{\boldsymbol{a}}_{\mathcal{B}_f}^T \end{pmatrix}, \quad \boldsymbol{b}_\psi = \begin{pmatrix} \psi_0 \\ \dot{\psi}_0 \\ \psi_f \\ \dot{\psi}_f \end{pmatrix} \tag{3.22}$$

The system has a unique solution, provided that $t_0 \neq t_f$ and that the knots are properly chosen.

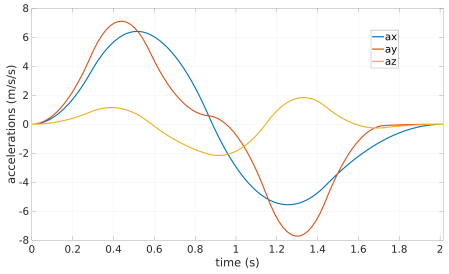
Any coefficient vector that satisfies the linear constraints (3.18) automatically satisfies the initial and final state constraints. For simplifications one can fix these conditions to reduce the number of decision variables. Regarding the B-spline knot vector, this means that the initial and final elements are fixed.

At this point, any general-purpose optimization strategy can be used to find a numerical solution to Problem 4. Unfortunately, due to the non trivial non-linearity of (3.16d), Problem 4 cannot be proven to be convex. The optimization will thus, in general, return a local minimum. Figure 3.1 shows a solution trajectory and the inputs profiles obtained from the resolution of Problem 4 with an initial hovering state at position $\boldsymbol{r} = (0, 0, 1)$ and a final hovering state $\boldsymbol{r} = (2.5, 2.5, 1)$. Note that satisfaction of the inputs constraints cannot be guaranteed (although it is the case in this example). The reason is that the inputs are discretized and the inputs can be very sharp (see Fig. 3.1c). The discretization time-step might be chosen in response to the sharpness of the inputs.

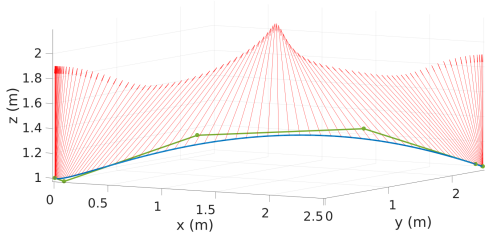
To sum up, there are three steps to trajectory generation based on the differential flatness theory; i) The first is to choose flat outputs, so the system can be mapped to a lower dimensional output space. Meanwhile, the cost function, the boundary and constraints can also be mapped to the output space; ii) The second is to choose a suitable basis function to parametrize flat outputs; iii) After parametrizing the selected outputs, we need to solve a set of coefficients.

In this thesis, we use nonlinear programming to solve for the coefficients of the B-splines to minimize the cost function subject to bound conditions and trajectory constraints in flat output space. Then, we obtain the flat output trajectories satisfying the constraints expressed by the computed coefficients.

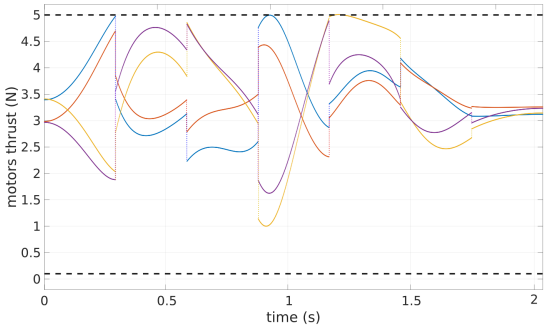
Note that by suitably parametrizing trajectories with basis functions in the flat space and by considering linear inequalities in the flat space to model constraints on



(a) Acceleration profiles that are typical in bang-bang control.



(b) Trajectory (blue line) and total thrust direction (red arrows). The green line represents the convex hull of the spline control points (green dots).



(c) Motors thrust within bounds $f_m = 0.1N$ and $f_M = 5N$.

Figure 3.1 – Results from the resolution of the minimum-time Problem 4 using the SQP method of the *fmincon* function in Matlab. The minimum-time trajectory has a duration of $T^* = 2.02s$ and is found after 13 SQP iterations

states and inputs $\boldsymbol{u}$ it is possible to turn this optimization problem into a quadratic program that can be solved in real-time for planning. This simplification will be discussed further in Sect. 4.5 and applied in Chapt. 8.

3.6 Summary

In this chapter, we presented a brief overview of the classical numerical methods for solving constrained optimal control problems. To sum up, constrained optimal control problems do not contain a closed form solution, approximation techniques can be employed for a numerical solution. The advantage of indirect methods is that very accurate solutions can be obtained. The main disadvantage of indirect methods is their lack of robustness to a poor initial guess. In this thesis we mainly focus in the parametrization of trajectories with B-splines whose control points constitute the decision variables of the nonlinear programs we will define. Several properties of such curves will be exploited in the design of the planning problems.

Now that we presented a way of solving nonlinear problems we focus on the on-line generation of optimal trajectories.

Chapter 4

Model predictive control: toward trajectory re-planning

Contents

4.1 Introduction and context

One can observe that the research community in aerial robotics is exploring more and more complex objectives such as tracking a fast moving target, avoiding dynamic obstacles, passing through several way-points and so on. More details are given in the state of the art in Chapt. 5. Designing a controller to perform these tasks would be challenging and could possibly lead to severe sub-optimality. With optimization-based controllers, it is easy to include different (and possibly conflicting) objectives and constraints. Now, due to increasing performance of computers, nonlinear programming tends to be more and more tractable and able to substitute popular controllers. Having robots planning their own trajectories becomes more and more practicable. However, in order to be efficient the important goal of optimal trajectory generation is to construct, in real time, a solution that optimizes the system objective while satisfying system dynamics, as well as state and actuation constraints.

Moreover, as briefly discussed in Sect. 2.2, since the system's model is imperfect, model and parametric uncertainty (e.g., inertial parameters) may lead to substantial deviation from the reference trajectory. Classic solutions include adaptive control [127] for estimating the nominal parameters online. However, trajectories that "excite" enough the estimation may be hard to find and may even conflict with the main task especially if we are interested in minimum-time control. A second approach is to design robust control laws typically implementing feed-forward terms for instance [59]. Finally, another strategy consists in directly designing specific trajectories that are robust to these modelling errors or nominal parameters uncertainty [128]. MPC tackles these issues in a slightly different approach: the reference trajectory and/or command inputs are adapted in real-time via optimization techniques as a feedback to cope with disturbances and modelling errors. MPC acts more as a high-level controller in this approach. Note that MPC techniques can incorporate uncertainty in the control process and are referred as Robust MPC and Stochastic MPC.

Several randomized trajectory generation techniques (such as RRT^* and A^*), originally applied to the mission level (see e.g., [129, 130, 131]), have been recently reported in complex scenarios [132] and real-time applications for dynamic systems [133, 132]. We will introduce and discuss more about search-based methods further in this thesis since they also play a decisive role in complex motion planning nowadays.

In this section, we shed light on *reactive* replanning methods and especially Model Predictive Control (MPC) also known as Receding Horizon Control (RHC). By reactive we mean able to generate solutions on-line fast enough to respond efficiently to sudden changes in the environment (obstacles, target, ...) or new situations.

4.2 Principle

In Model Predictive Control, an open-loop trajectory is found by solving a finite-horizon constrained optimal control problem starting from the current state. The optimal controls of this trajectory are then applied to the system for a certain fraction of the horizon length, after which the process is repeated (see Fig. 4.1). Note that the essence of MPC is to optimize over the predictions of a process behaviour. Therefore, the process model is essential.

MPC is a family of algorithms which give the possibility to:

- explicitly include in the problem formulation constraints on state, input, output variables, and logic relations;
- consider hundreds of control and controlled variables;

- transform the control problem into an optimization one, where different, and sometimes conflicting, goals can be stated;
- use very detailed physical (nonlinear, DAE, ...) models with continuous and integer variables.
- close an optimal control loop

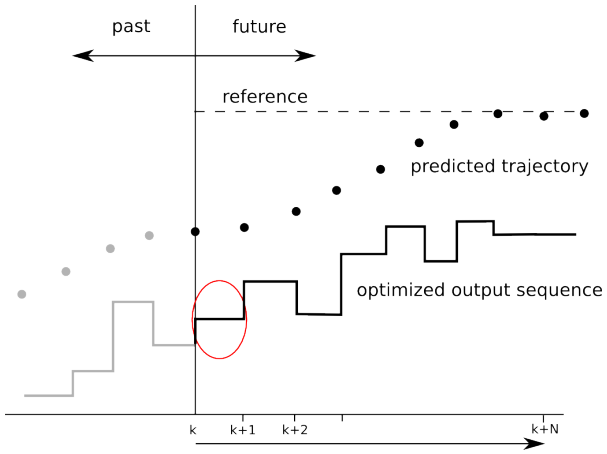


Figure 4.1 – At time k the future sequence of control variables is computed based on the prediction of the future states. Then the first value of the optimal control sequence is applied to the system (circled in red). At time $k+1$ the optimization is repeated with the same prediction horizon.

To implement the receding horizon control strategy, a constrained (and often nonlinear) optimization control problem must be solved on-line. Due to the complexity of solving a nonlinear programming problem in real-time, the computational delay cannot be ignored. This is particularly important in aerial and aerospace applications, where the dynamics is high and the internal control loops are very short. Originally applied in the process control industry where dynamics are relatively slow, the application of receding horizon control to aerial vehicles has been proposed and analysed by several researchers [134, 36]. Most popular applications include system stabilization, evasive manoeuvres, obstacle avoidance and target tracking.

The receding horizon strategy offers many benefits in this environment, such as the inherent ability to deal with constraints in the state and control. Examples of such constraints commonly encountered include dynamic terrain obstacles, dynamic or pop-up threats, saturations on the actuators, impair of the capacity of a vehicle. However, a few requirements are needed, we must guarantee the convergence of the algorithm at each computation, and guarantee the fastness of the convergence. Indeed, the faster the algorithm is, the less the previous solution is out-dated so the

more the algorithm will be able to improve it instead of just adapt it to the new situation.

Moreover, these approaches serve little practical purpose until stable and efficient computational techniques are developed to provide real-time solutions to the underlying constrained nonlinear optimal control problems. Closed-loop stability has been well defined in [135] but gets much more difficult to prove when the problem is nonlinear.

In this thesis we take inspiration from the replanning scheme of receding horizon control for iteratively solving nonlinear programs. This general idea has been widely employed in research with several planning strategies in order to face the heavy computation loads and to meet real-time [69, 35, 36].

4.3 Receding horizon formulation: the linear case

Let us define a general linear optimal control problem by considering the following system

$$x(k+1) = Ax(k) + Bu(k) \tag{4.1a}$$

$$y(k) = Cx(k) \tag{4.1b}$$

where $x \in \mathbb{R}^n$ is the state vector of dimension n, $u \in \mathbb{R}^m$ is the input vector of dimension m and $y \in \mathbb{R}^p$ is the output vector of dimension p. At time k we want to compute the sequence of future control variables

$$u(.) = [u(k), u(k+1), ...u(k+N-1)]^T \tag{4.1c}$$

minimizing the objective

$$J(x(.), u(.), k) = \sum_{i=0}^{N-1} (\|x(k+i)\|_Q^2 + \|u(k+i)\|_R^2) + \|x(k+N)\|_S^2 \tag{4.2}$$

where $Q = Q^T \geqslant 0,\ R = R^T \geqslant 0,\ S = S^T \geqslant 0$ play the role of weighting matrices and N denotes the *prediction horizon*. Finally, $\|x(k+N)\|_S^2$ is the *terminal cost*.

The optimal solution to this problem is given by the state-feedback control law

$$u^0(k+i) = -\mathcal{K}(i)x(k+i), \quad i = 0, 1, ..., N-1 \tag{4.3}$$

where $u^0(.)$ is the sequence of optimal inputs and the gain $\mathcal{K}(i)$ is given by the expression

$$K(i) = (R + B^T P(i+1)B)^{-1} B^T P(i+1)A \tag{4.4}$$

and $P(i)$ is the solution of the difference Riccati equation

$$P(i) = Q + A^T P(i+1)A - A^T P(i+1)B(R + B^T P(i+1)B)^{-1} B^T P(i+1)A \tag{4.5}$$

with terminal condition

$$P(N) = S \tag{4.6}$$

The weighting matrix S plays a role in closed loop stability, a typical choice is a quadratic Lyapunov function especially for the generalization to nonlinear and constrained systems, see [136] for instance. Finally, recalling the Lagrange equation

$$x(k+i) = A^i x(k) + \sum_{j=0}^{i-1} A^{i-j-1} B u(k+j), \quad i > 0 \tag{4.7}$$

and defining

$$X(k) = \begin{pmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+N-1) \\ x(k+N) \end{pmatrix}, \; \mathcal{A} = \begin{pmatrix} A \\ A^2 \\ \vdots \\ A^{N-1} \\ A^N \end{pmatrix}, \; U(k) = \begin{pmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+N-2) \\ u(k+N-1) \end{pmatrix} \tag{4.8}$$

$$\mathcal{B} = \begin{pmatrix} B & 0 & 0 & \cdots & 0 & 0 \\ AB & B & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A^{N-2}B & A^{N-3}B & A^{N-4}B & \cdots & B & 0 \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \cdots & AB & B \end{pmatrix} \tag{4.9}$$

then the future state variables are given by

$$X(k) = \mathcal{A}x(k) + \mathcal{B}U(k) \tag{4.10}$$

The optimal cost can be found as

$$\bar{J}(x(k), u(.), k) = X^T(k) diag(Q) X(k) + U^T(k) diag(R) U(k) \tag{4.11}$$

where the term $x^T(k) diag(Q) x(k)$ can be ignored since it does not depend on $U(k)$.

Linear constraints on the state, inputs and outputs can easily be handled with this framework. Let us define the following constraints the problem is subject to

$$x(k+1) = Ax(k) + Bu(k) \tag{4.12a}$$

$$\underline{u} \leqslant u(k+i) \leqslant \bar{u} \quad i = 0, 1, ..., N-1 \tag{4.12b}$$

$$\underline{x} \leqslant x(k+i) \leqslant \bar{x} \quad i = 0, 1, ..., N-1 \tag{4.12c}$$

$$\underline{y} \leqslant y(k+i) \leqslant \bar{y} \quad i = 0, 1, ..., N-1 \tag{4.12d}$$

This problem (quadratic cost function, linear constraints) can be easily solved by means of a QP method with very reasonable computational time (which obviously depends on the problem size).

However, for constrained (and obviously nonlinear) systems the control law is implicitly defined, i.e., its value can be numerically computed through the solution of the optimization problem, but its analytic expression is unknown. However, the principle is the same, only the first element (in general) of the input sequence of the open-loop solution is applied to the system which defines a time-invariant control law for the closed-loop of the form $u^0(k) = \chi_{RH}(k)$. For the linear case we have

$$u^0(k) = -\mathcal{K}(0)x(k) \tag{4.13}$$

with

$$\mathcal{K}(0) = (R + B^T P(1)B)^{-1} B^T P(1)A \tag{4.14}$$

obtained by iterating the Riccati equation backwards from

$$P(N) = S \tag{4.15}$$

Now, one can address the case of nonlinear systems which constitutes a much wider class than linear systems. Considering the applications, one can resort to local linearization of the system dynamics and constraints which is a classical technique in chemical industry. Note that this technique was successfully applied to the control of robot manipulators [137] to achieve fast motions.

When applied to nonlinear systems, the algorithm may demand tremendous computational power, and can exhibit poor convergent stability if not implemented properly. For instance, if one chooses a large prediction horizon the solver may not be able to compute a solution fast enough. If too short, the resulting solution may lead to instabilities and even to a failure of the task since the problem may not have enough degrees of freedom to converge to a solution. These difficulties have largely prevented its application to stability critical nonlinear systems with fast dynamics. Increasingly powerful and affordable computing facilities combined with better understanding of receding horizon control's stability properties have revived interests in this area. See e.g., [135, 138] for a good review of recent work in this field.

4.4 An application of MPC to quadrotor control

In this section we present some simulation and experimental results to illustrate the role of a MPC scheme applied to trajectory tracking with a quadrotor in presence of obstacles[1].

[1]This work is the fruit of a common work conducted with Gerardo Rodriguez who completed his Master Thesis at Inria Rennes

Since MPC relies on a model equations we choose to use the Brunosvky equivalent linear form (A.32) in the flat space. Without loss of generality we do not plan over the yaw angle that we assume it is kept constant. Let us define a classic objective function

$$J(\boldsymbol{r}(.), \boldsymbol{u}(.)) = \sum_{i=0}^{N-1} (\|\boldsymbol{r}_i - \boldsymbol{r}_i^*\|_Q^2 + \|\boldsymbol{u}_i\|_R^2) + \|\boldsymbol{r}_N - \boldsymbol{r}_N^*\|_S^2 \tag{4.16}$$

where $\boldsymbol{r}_i = (x_i, y_i, z_i)^T$ and $\boldsymbol{u} = (x_i^{(4)}, y_i^{(4)}, z_i^{(4)})^T$ are the predicted trajectory and inputs respectively, $\boldsymbol{r}_i^*$ denotes the i-th reference point. We choose the Gerono lemniscate as the desired reference path defined at time t by

$$x(t) = r\cos(\alpha t)\sin(\alpha t), \tag{4.17a}$$
$$y(t) = r\cos(\alpha t), \tag{4.17b}$$
$$z(t) = z_0 \tag{4.17c}$$

Therefore, $\boldsymbol{r}_i^*$ is defined as a moving point on the reference path and we penalize the deviation of the quadrotor from this reference point over the prediction horizon N. In order to evaluate the MPC response to obstacle avoidance, we add two planes along the X and Y-axis located in x_{wall_1} and y_{wall_2} and a spherical obstacle of radius r_{obs} at position $\boldsymbol{r}_{obs}$ that overlap with the reference path (see Fig. 4.2). We model the quadrotor as a sphere of radius l centered in $\boldsymbol{r}$. The geometric constraints are defined as

$$\|\boldsymbol{r} - \boldsymbol{r}_{obs}\| \geqslant r_{obs} + l, \tag{4.18a}$$
$$x \leqslant x_{wall_1} + l, \tag{4.18b}$$
$$y \leqslant y_{wall_2} + l \tag{4.18c}$$

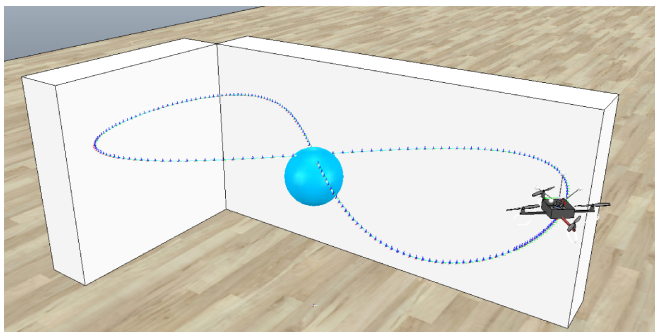


Figure 4.2 – Simulated environment. The blue line represents the lemniscate path to track. The quadrotor has to avoid collisions with a sphere and two walls while keeping a minimal distance with the reference path.

As already discussed, thanks to differential flatness any smooth trajectory in the space of flat outputs can be followed provided that derivatives are correctly bounded.

Nevertheless, this powerful property can be difficult to apprehend and implement for real systems involving highly nonlinear dynamics. The following section details alternatives to the definition of the dynamic constraints for the quadrotor.

4.4.1 A relaxed formulation based on differential flatness

In some cases, changing the planning space to the flat space may not grant obvious physical meaning and makes the equations, and particularly the expression of the real system inputs much more complex than their original formulation.

Now, a very debated question is how to define these constraints on the dynamics. Basically, it is inherent to questions of formulation complexity, conservatism and planning strategy. Namely, it would be better of course to consider the real physical limitations of the motors as constraints to guarantee that the trajectories are feasible by the real system but considering the application it may be more attractive to use less complex constraints (linear if possible) on different level of the dynamics.

[139] addresses the generation of smooth trajectories in the kinodynamic state space with inequality constraints on the absolute value of the derivatives of the flat outputs defined as follows

$$|\boldsymbol{v}| < v_{max} \tag{4.19a}$$
$$|\dot{\boldsymbol{v}}| < a_{max} \tag{4.19b}$$
$$|\ddot{\boldsymbol{v}}| < j_{max} \tag{4.19c}$$
$$|\dddot{\boldsymbol{v}}| < s_{max} \tag{4.19d}$$
$$\tag{4.19e}$$

Some works have focused on estimating the feasible set in flat output space by polytopic approximations (e.g., [140]), however this set is generally a non-convex function of nonlinear inequalities and is a hard optimization problem unto itself.

In this preliminary work we consider the abovementioned constraints (4.19) to achieve the desired replanning rate and closed-loop stability. Indeed, using constraints directly on the motors thrust with constraint (3.2e) were found too complex for the solver.

Similar choices were made in [6] using constraints on the acceleration and the jerk. Constraints on the total thrust are evaluated afterwards and the problem is rescaled until they are satisfied. The conservative nature of the jerk bounds means that only a fraction of the allowable body rates is typically used. If these exceed limitations, it was shown that a feasible trajectory can always be found by reducing the allowable jerk values.

4.4.2 Results and delay compensation

The MPC scheme was carried out using the ACADO toolkit [141] which implements a multiple-shooting algorithm. ACADO solves multiple shooting problems thanks to a SQP algorithm, together with state-of-the-art techniques to condense, relax, integrate and differentiate the problem. The quadrotor dynamics were simulated using V-Rep[2] at 150Hz. The generated trajectories of the flat outputs were sent to TeleKyb [142] at a rate of 30Hz which then computed the actual control inputs using the geometric controller developed in [143].

A major issue in the implementation of receding horizon control is handling the computational delay associated with the real-time optimization. We present here a simple method for designing an initial guess and take delay into consideration for the replanning. We select a section of the optimal states sequence which is sent to the controller. Since the control loop runs faster than the solver, we choose to interpolate cubic splines between the optimal states to smooth the controller action. Finally, to compensate for the delay (assumed constant at 1/30ms) we predict the initial state for the next OCP by projecting the previous solution in the future (i.e., 30ms ahead of the current time).

We opted for the following bounds: $v_{max} = 1.5m/s$, $a_{max} = 4m.s^{-2}$, $j_{max} = 15m.s^{-3}$, $s_{max} = 100m.s^{-4}$. The robot is able to plan trajectories that avoid the obstacles with a prediction horizon $N = 50$. The considered optimal control problem is solved within around 30 ms. The robot profiles during simulation are shown in Fig. 4.3.

To illustrate the reactivity of MPC we conducted a second simulation where a human operator is sending velocity commands (up to 1.5m/s) to the quadrotor via a joystick. We encode collision avoidance constraints so that safe and high-speed navigation among obstacles is handled by the MPC action, see Fig. 4.4. We impose the following bounds: $v_{max} = 1.5m/s$, $a_{max} = 10m.s^{-2}$, $j_{max} = 30m.s^{-3}$, $s_{max} = 150m.s^{-4}$.

4.5 Summary

The results presented in this chapter demonstrate the potential of real-time receding horizon control for constrained systems with fast dynamics. Real-time RHC control represents a revolutionary alternative to the traditional linear or nonlinear controller design with many benefits.

First, in most cases, a global system model and objective function are easier to obtain than a traditional linear or nonlinear controller that works globally. For

[2] http://www.coppeliarobotics.com/

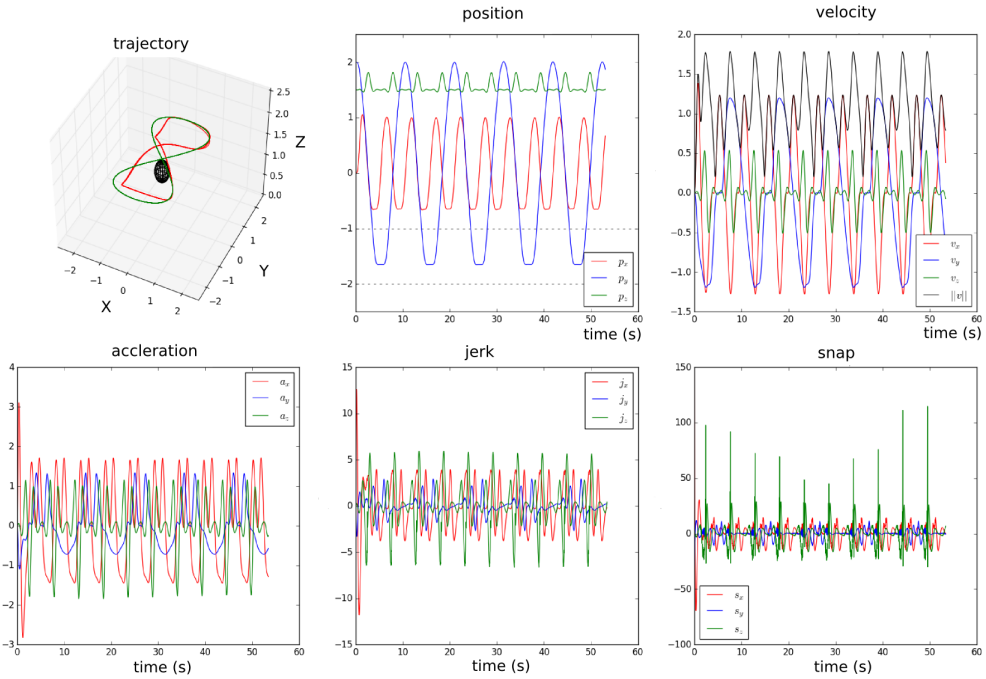


Figure 4.3 – The upper left figure shows the robot path (in red) and the reference path (in green). The obstacles are represented with the black volumes. The other figures show the derivatives of the flat outputs x, y, z.

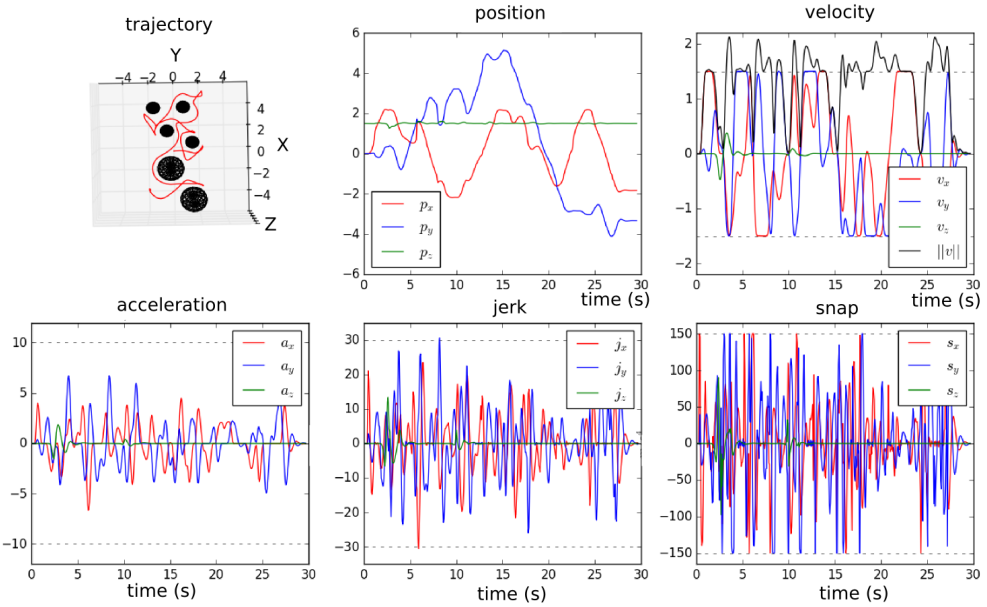


Figure 4.4 – The upper left figure shows the robot path in a cluttered environment. The robot is able to avoid the static obstacles even though the operator is sending high velocity commands with the joystick. The other figures show that the dynamic constraints are respected thanks to the MPC action. Note that the quadrotor reaches linear velocities up to 1.5m/s

a complex nonlinear system, classical controller design techniques may show weak stability proofs and may not exhibit flexible actions to the different possible situations and environment changes. In comparison, given an accurate nonlinear model and adequately defined objective function, real-time RHC could provide a global optimal control that is elegant and flexible. For example, RHC can be easily reconfigured by changing the model or any parameter (see [144] for an illustration of RHC strategies applied to complex systems).

Second, real-time RHC can provide optimal control solutions, even for systems with complex constraints such as actuator saturation, operational limits, terrain avoidance, etc. In contrast, it is extremely difficult to design a classic controller for constrained systems.

Third, with accurate modelling and precise objective definition, system performance could be far more superior than classic linear or nonlinear controller, particularly for very aggressive manoeuvring that pushes the constraint boundaries.

Fourth, in many cases, real-time RHC eliminates the necessity of both inner loops and outer loops that is common in classic tracking and stability control design. Instead, trajectory generation and robust control are performed in a single integrated design with potentially better performance and higher bandwidth. In this chapter we used ACADO which is specifically designed for implementing MPC problems. Yet, handling more complex and nonlinear constraints on the motors thrust did not lead to satisfactory results, especially for real-time control. In our thesis we opted for a different on-the-shelf nonlinear solver to generate feasible and reactive trajectories in the presence of multiple nonlinear constraints for accomplishing several vision-based tasks.

Now that we have presented the main ingredients and concepts used in this thesis, we introduce the most relevant contributions identified in the literature that are related to our work.

Chapter 5

Aggressive trajectory generation and vision-based planning for a quadrotor: related works

Contents

5.1 Optimization-based methods

Many dynamic manoeuvres have been performed in the recent years using optimization methods. They include fast translations [29], ball catching [35] and flights through narrow gaps [24] for instance. In [24] a quadrotor flies through a window using vision as feedback in a complete autonomous way using only onboard sensing and computing. High angular rates were achieved in real experiments. The aggressiveness of motions is mainly limited by the quality of the visual feedback (especially due to motion blur).

Due to the growing computation power of computers it becomes more and more practicable to generate trajectories online. Yet, to meet this challenging demand, several works have been developing mathematical tools to efficiently generate feasible trajectories close to the actuation limits by relying on more or less conservative

approaches. We can refer to the following leading strategies listed below. All of these methods rely on a particular optimal criterion.

- Using a class of lightweight motion primitives: Several approaches merge optimal trajectories and reactive re-planning by generating computationally lightweight motion primitives as an implicit feedback control law. These simple curves (polynomials, splines, lines) constitute more or less rich trajectories that are easy to manipulate and evaluate at a lower level. [6, 35] use the Pontryagin's minimum principle to generate candidate time-optimal trajectories between two states that are sent to the controller after checking that the constraints are satisfied. In [35] a two-dimensional quadrotor model is considered, the axes are decoupled and conservative feasibility tests are developed on the total thrust and the angular rates to validate the generated trajectories that are shown in Fig. 5.1. Nevertheless, position constraints are not considered. To make such an algorithm successful, the classic paradigm in control schemes is addressed, namely the trade-off between trajectory quality (i.e., in terms of feasibility, optimality, constraints satisfaction, ...) and planning rate (which needs to be high for such an agile system).

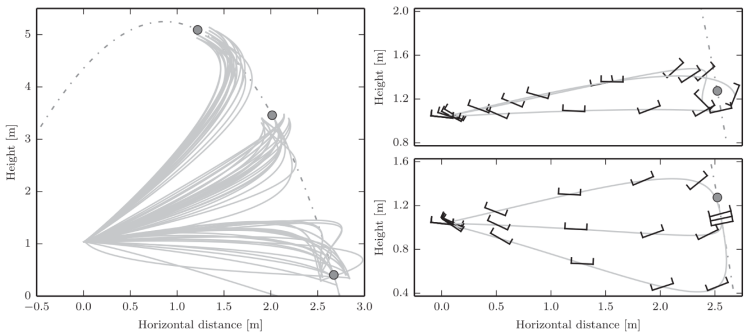
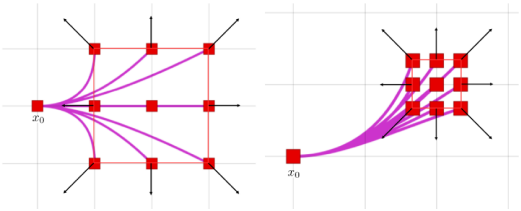


Figure 5.1 – Sampled motion primitives for a catching manoeuvre. The presented algorithm is able to generate about 6700 feasible motion primitives per second, from [35].

- using graph-search methods: the approach relies on exploring the state space with probabilistic methods such as RRT, RRT* or A* algorithms. A path is built as a succession of straight paths [145] or curves [146] or motion primitives (as shown in Fig. 5.2) forming a set of *vertices* connected by *edges* with a certain level of continuity [147, 133]. Generally, various (and complex) constraints can be considered. Constraints can be checked at each extended *vertex* using simple tests. If constraints are satisfied the vertex is added to the graph.

- using Mixed-Integer Programming: this method involves problems in which some of the variables are integers. The algorithm is usually employed for



(a) Discretized Acceleration. (b) Discretized Jerk.

Figure 5.2 – Example of motion primitives from an initial state for an acceleration-controller system (left) and a jerk-controller system (right). The black arrow indicates the correpsonding control inputs, from [147].

finding collision-free paths passing through a set of keyframes by minimizing some criterion. This planning method is capable of handling a large set of constraints but usually only enforces collision avoidance with obstacles or body parts and plans a single trajectory with integer constraints (see e.g., [31] Fig. 5.3 or [148] for a single quadrotor, [149] for multiple agents and [150] for multi-body system). The environment is often partitioned into convex sub-regions in the configuration space, constraints are linear and differential flatness is used for tractability reasons but the solver generally takes seconds to hundreds of seconds to determine a proper solution.

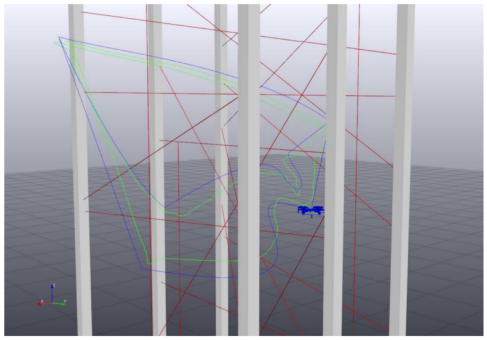


Figure 5.3 – Agile path tracking with a micro quadrotor in a heavily cluttered environment of strings and poles. The algorithm takes about 10 minutes to return a solution, from [31].

- using direct optimization: this method (that we presented in Chapt. 3) is capable of dealing with generic problems with various constraints. Nonlinear problems can be solved numerically using SQP for instance to generate online optimal solutions if properly posed. Re-planning strategies have been demonstrated by successively solving OCP, such as MPC [36].

- using a combination of the aforementioned techniques: in many occasions, a collision-free path is built and a second step involves an optimization program that computes a path of higher resolution taking the system dynamics into

account. Examples along this line are numerous, e.g., [148] combines mixed-integer programming and direct optimization, [33] combines graph-search and direct optimization to perform high-speed flights for a quadrotor, and [151] combines a variant of the three methods.

5.2 Graph-search approaches

These approaches that we briefly introduced earlier in Sect. 5.1 are widely exploited in the literature. Also very generic and computationally more and more attractive, they have been successfully applied to solve many motion planning problems for UAVs. Searching algorithms such as RRT* or A* are known for suffering from the *curse of dimensionality*–the ability to properly scale to high-dimensional space. Nowadays, several techniques have been proposed to overcome these limitations. Many authors separate the problem into two steps: an optimal path is found through graph-search without considering dynamic constraints. Then, an optimal dynamic trajectory is generated by optimizing over a collection of waypoints. This second step plays the role of enforcing dynamic feasibility and constraints by adapting the trajectory speed [43], jerk [69] or time [1] such that input constraints are not violated. This process is generally done recursively or using a scaling algorithm [35, 152] until dynamic constraints become active. These approaches are usually sufficiently fast to provide a feedback loop by re-planning the reference trajectory at every controller update. In [75] the authors combine RRT* with polynomial trajectory generation to compute dynamically feasible trajectories for a quadrotor using a two-step approach (see Fig. 5.4). A variant of RRT for flat systems is detailed in [153] to produce smooth dynamically feasible motion plans in real-time and for online navigation in dynamic environments with a quadrotor in [133]. Both works exploit differential flatness to build a look-up table of pre-computed feasible motion primitives.

However, several classes of problems cannot be treated using this approach. For instance the robot orientation cannot be properly considered at the geometric stage in general. The sole counter example is [147] where constraints on the quadrotor attitude are considered at the planning stage (see Fig. 5.5).

Finally, such techniques become more and more adapted to real-time planning and successfully applied to navigation in unknown environments, see e.g., [154, 151, 155].

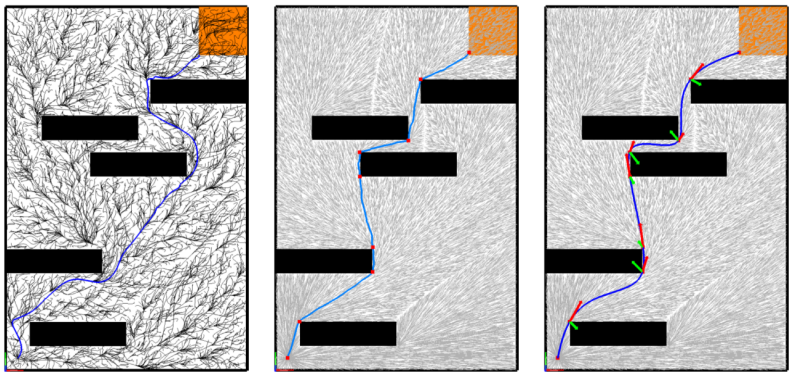


Figure 5.4 – A path is built from a straight-line RRT* (middle figure) and then refined to obtain a minimum-snap trajectory that is feasible for the real system (right figure). The approach is much faster than a RRT* with a polynomial steer function (left figure), from [75]).

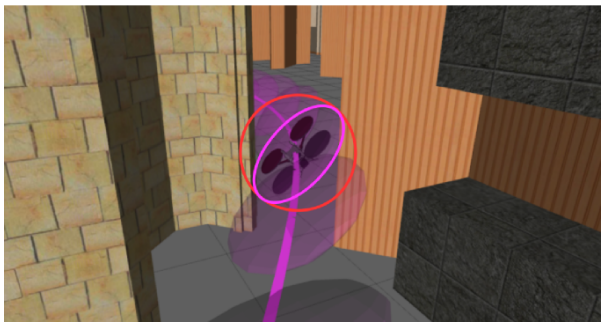


Figure 5.5 – A minimum-jerk trajectory is found for ensuring safe navigation among obstacles. The collision avoidance relies on the robot occupancy modelled as an ellipsoid (in pink) which is more accurate than a spherical model (in red), from [132]).

5.3 The minimum-time problem

Naturally, the generation of aggressive trajectories often resorts to the minimization of time. Having the flying time T as a decision variable is very complicated. It is a free parameter (i.e., it is not directly subject to any constraints) but it strongly acts on the dynamic constraints and the shape of the trajectory (see Fig. 5.6).

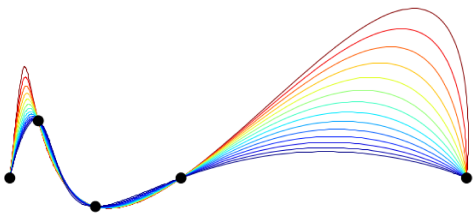


Figure 5.6 – Shape of a trajectory passing through waypoints with different flying times, from [75].

If trajectories are parametrized with polynomials, T appears in each derivative as a nonlinear decision variable resulting in re-evaluation of the basis at each solver iteration. Moreover, as T converges to zero numerical stability can be compromised.

[156] overcame with this numerical issue by minimizing instead the settling time $N \in \mathbb{N}$, namely the number of discrete-sampling intervals required to reach the goal. This method greatly alleviates the numerical resolution and is able to suppress high frequency chattering due to measurement noise especially occurring in the vicinity of the origin. In [137] a similar penalization is considered but in a hierarchical optimization framework to approach minimum-time trajectories for robot manipulators.

In a similar philosophy, many works rely on scaling approaches [69] to perform motion close to the actuation limits. Bang-bang strategies [58] are designed for quadrotors. [139] relies on a closed-form solution to compute arcs in the kinodynamic state space that tend to minimize the flying time. Here, the intuition is to minimize the time spent to reach the full speed during a flying phase resulting in bang-similar-bang. In the end it is shown that this implies to maximize time spent at maximum snap during jerk variations and to minimize the durations of snap variations.

[157] reformulates the minimum-time problem by expressing the quadrotor dynamics in a new set of "transverse" coordinates with respect to the reference path. However real-time could not be achieved.

In our works (e.g., [2, 4]), due to the underactuation a quadrotor equipped with a camera may have to increase its height in order to enlarge its field of view for converging faster towards a visual target while keeping it in the image plane. In this context, rescaling strategies might not be able to reproduce such a behaviour and will lead to severe sub-optimal solutions since it only acts on the single temporal parameter. The same observations can be made when addressing collision avoidance.

5.4 Vision-based control for the underactuated quadrotor

As already discussed, most vision-based approaches assume first-order or fully-actuated systems. Classic methods cannot be directly applied to quadrotors due to their complex dynamics and inherent underactuation that conflicts with the main servoing task.

In the next section we present how the issues related to vision with underactuated systems are addressed in the literature.

5.4.1 Visibility constraints and occlusion avoidance

A common approach in IBVS is to decouple the rotational kinematics of the vehicle from the image features. The image feature error is projected in a "rotation-compensated" camera frame or "virtual plane" (see Fig. 5.7) which is horizontal and has the same position and yaw angle of the real camera's image plane. Thus, by re-projecting the image points using attitude measurements the camera rotation is decoupled from the translation motion. This virtual plane also facilitates the estimation of depth of image points. This strategy has been applied in several works, e.g., in [95, 158, 159, 160, 105] for the design of globally stable dynamic IBVS schemes. Although these works develop controllers that guarantee the image error in the rotation-compensated frame will converge to zero, the quadrotor underactuation is not explicitly taken into account by the control design. Therefore, it is still possible for the image features to completely leave the camera field of view if the system has significant rotation, resulting in tracking failure for high speed manoeuvres.

[161] presents several IBVS control techniques which decouple the image space from the task space by using spherical image moments as features [162]. Since the image error becomes a function of position only, large rotations could still occur, making the system vulnerable to failure as previously described.

Although, dynamic visual servoing schemes have been developed for second order or under-actuated systems (e.g., [161], [159], [163] or [164] for quadrotors), the underlying assumptions fail for high-speed manoeuvres and in any case, do not take into account possible loss of visibility or occlusions. Yet, the effort was allocated to proving stability of the closed-loop dynamics and providing robustness analysis.

In [165, 166] a dynamic IBVS controller based on a backstepping method is proposed using first-order spherical image moments as visual features. Both papers provide interesting passivity properties and rigorous proofs of closed-loop stability, but the proposed interaction matrices remain ill-conditioned as the image feature is insensitive to change in altitude. Hence, performance suffers from a low rate of convergence in altitude. Following these works [167] later eliminates the need of height estimation and the use of an external sensor for measuring the translational velocity. Guarantees of convergence are given for landing on a moving target. However, features are assumed to remain visible at all times and such a controller may not be applied to higher-speed translational motions.

[168] introduced a controller that takes into account the quadrotor underactuation and uses a virtual spring force to prevent the robot from rotating too much. However, a small change in roll or pitch may cause a large change in the proposed interaction matrix. This clearly reduces the quadrotor reactivity and, in any case,

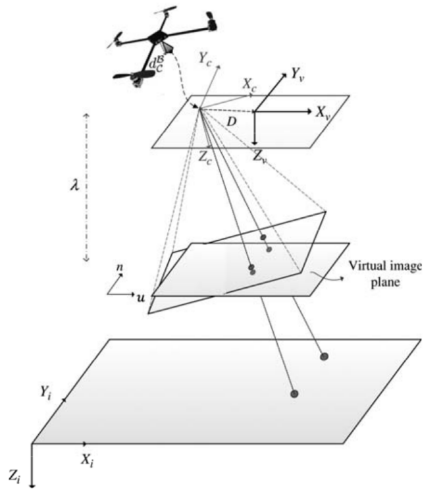


Figure 5.7 – Virtual image plane always parallel to the ground, from [159].

does not strictly guarantee the satisfaction of visibility constraints.

Some authors focused on feature estimation for recovering the visual-based task rather than avoiding occlusions [169] or visibility losses [170]. In [171] the authors propose a vision-based algorithm to autonomously track a moving object. The tracking algorithm is robust to occlusions but does not avoid them and assumes the target stays in the field of view.

Potential fields are classically used for designing control laws for repelling visual cues from the projected obstacles in the image [172],[173] but this technique may strongly conflict with the nominal servoing task and increase the chance of falling into local minima. Gradient Projection Methods (GPM) use the system redundancy to mitigate the completion of two tasks [174],[175]. The secondary task gradient is projected on the null-space of the main task and uses the remaining redundancy to complete the avoidance task [174]. However, if all DOF are used one cannot apply this approach. Obviously, the redundancy formalism does not appear reasonable when dealing with underactuated robots such as quadrotors. Using the same redundancy formalism spirit a 6-DOF robot is controlled in [175] while simultaneously avoiding occlusions and joint limits. A relaxed control law is proposed which uses all DOF to simply prevent the main task error from increasing while performing a secondary task.

Another approach is to use *activation functions* in the control law to enable smooth transitions between safe and forbidden regions in the image plane [176]. The control acts on features that are out of some confidence area in order to release some degrees of freedom to manage others tasks. However, this technique raises stability issues.

After an examination of the relevant literature we can conclude that vision-based control laws for underactuated systems, such as quadrotors, oftentimes do not explicitly ensure that the relevant image features stay in the field of view of the camera and hardly deal with occlusions. In any case, they can be applied to perform agile manoeuvres. Note that visual servoing controllers for fully actuated second-order systems have been proposed in the literature (see, e.g. [110, 111].

In the context of vision-based optimization, using visual features as flat outputs has been considered. [177] extended their peer work [178] by encapsulating an image-based flatness formulation inside a MPC scheme by using the target image coordinates of a mobile robot as flat outputs. Now, visibility constraints appear in the flat space and thus are more simple to satisfy. However, this work only considers a fixed overhead camera. Further improvements were made more recently in [160] by finding flat outputs in the image plane considering a fixed camera attached to a 2D quadrotor to perform visual-based agile grasping in the XZ plane Fig. 5.8. The authors presented a trajectory generation method which guarantees dynamic feasibility and enables incorporating visual constraints as linear constraints. However, the existence of differential flatness is only possible with some model conservatism/approximations. Indeed, the mapping was done in a virtual image plane which is it not affected by the pitch angle. Therefore, the visibility constraints are not specified in the *real* image plane and may be too restraining for large rotations.

Figure 5.8 – Aggressive catching manoeuvres at 3m/s in the sagittal plane using a monocular camera. The catching strategy is inspired from the natural behaviour of the bald eagle snatching its prey, from [160].

The generation of motion primitives candidates presented in [35] has been applied to autonomous landing on a visual target in [170]. The target visibility is not guaranteed but a Kalman filter is used to estimate the target position in case of partial visibility losses. Nevertheless, this technique might not be suitable to complex 3D motions when the trajectory is shaped by visibility constraints for instance. Indeed, in many occasions the quadrotor may have to accelerate upwards in order to compensate for the camera rotation that inherently repels the image features from the image plane center. An other relevant degree of freedom used for

keeping visibility of point features is to also combine rotations along $z_{\mathcal{B}}$ to exploit the "shape" of the field of view (if one considers a square field of view). This is something that we observed in our works [2, 3] but rarely seen in the literature.

Recently, self-collision and simple occlusion avoidance tasks for a humanoid were incorporated in a quadratic optimization problem in [179]. Visibility of the feature is handled by using an avoidance task as in [176]. A more precise occlusion avoidance formulation is proposed in [180] but uses a larger set of visual constraints. In [181] the authors explored a randomized kinodynamic hybrid path planning approach applied to a manipulator for finding a feasible path. It satisfies a great deal of constraints both in the image and in the joint space but takes a few minutes to return a solution (see Fig. 5.9).

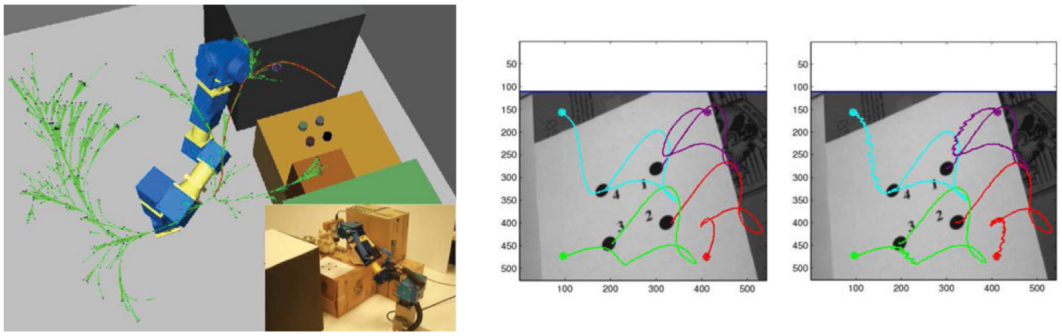


Figure 5.9 – A visual servoing task is performed with a manipulator while avoiding occlusions and loss of the visibility features, from [181].

More and more works incorporate perception objectives within an optimization program to keep some visual features in the field of view of the camera in order to improve the quality of the vision-based state estimation [182] (see Fig. 5.10a) or for keeping a visual feedback of a moving target [106] as shown in Fig. 5.10b.

Finally, MPC has taken down many of these issues through numerical optimization. [101] implemented a dense hybrid optimal visual servoing scheme to steer the underactued quadrotor towards a goal pose encoded as a desired goal set of image features. It uses a Perspective-n-Point (PnP) algorithm to estimate the goal state then an optimal trajectory minimizes the reprojection error of the features along the trajectory and the deviation of the path from the goal state. Although a large set of image features are considered there is no guaranty that enough features remain in the field of view since it is not encoded as a hard constraint. [183] uses barrier functions in a MPC framework to keep a quadrotor in the field of view of a mobile platform with a upward-looking camera in the presence of external disturbances. In [74] smooth collision-free trajectories are generated for multiple quadrotors by predicting the agents motion using pose observations. A stochastic MPC was implemented in [107] for autonomous aerial grasping. The MPC action

(a) A quadrotor is equipped with a front-looking camera and is asked to fly at 3m/s around a region of interest while keeping it visible in the field of view of its camera, from [182].

(b) A quadrotor tracking a moving target with a downward-looking camera. The quadrotor is able to increase its height in order to compensate for the camera rotation as the quadrotor moves forward, from [106].

Figure 5.10 – Examples of optimal navigation merging aggressive motion and perception objectives.

is able to respect visibility constraints but the achieved trajectories are close to near-hovering. Field of view and inputs constraints were considered in a fully autonomous aggressive target tracking receding horizon framework relying on onboard sensors with a downward camera attached to the quadrotor used for estimating the target position [106] which constitutes one of the most relevant works. The real robot velocity hits 5 m/s and was able to compensate for the camera rotation by accelerating upwards (see Fig. 5.10b). The authors algorithm generates smooth trajectories by minimizing the relative velocity error and the jerk first and then penalizing the relative position error after some proximity threshold is reached. We believe such a strategy contributes in improving motion stability but may however abate the motion aggressiveness.

In the aforementioned works, only a few consider hard visibility constraints for a fully actuated robot for aggressive motion, i.e., [106]. A second relevant work considers tracking a moving target with a quadrotor while avoiding obstacles in an unknown environment by generating on-line smooth and dynamically feasible trajectories provided that the target stays in the field of view [184].

5.5 Perception and uncertainty-aware planning

Since vision plays a major role in state estimation, many works have merged visual objectives with stochastic problems.

In [185] the authors resort to an RRT* algorithm to find optimal and online paths that minimize the pose uncertainty by driving a quadrotor equipped with a downward-looking camera toward regions of rich texture (see Fig. 5.11). The approach relies on photometric information of the ground in the context of visual

odometry. This work is one of the first to incorporate perception goals in path planning with a quadrotor. However, the planning takes place in a 2.5D set-up (motion in the horizontal plane and at a given fixed height) and is not designed for generating dynamic motions.

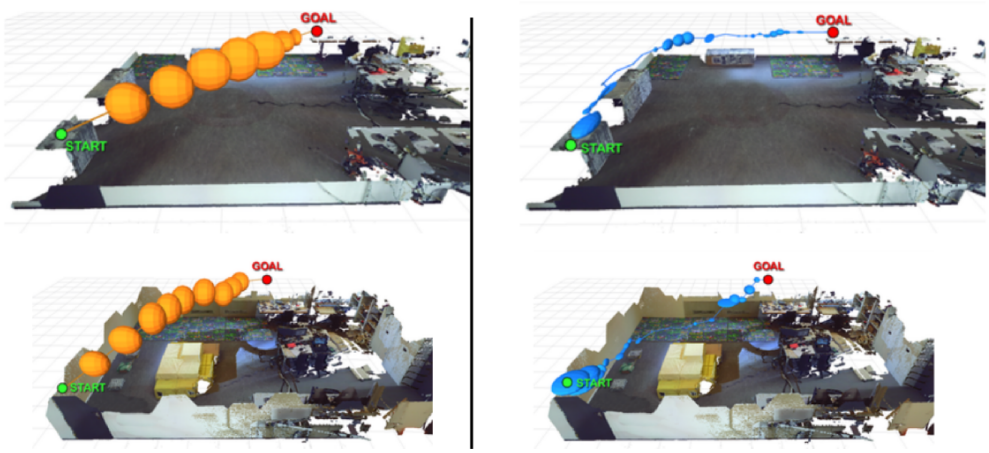


Figure 5.11 – The two figures on the left show paths obtained without taking the state uncertainty into account (represented by the orange ellipsoids). With the presented method the planner is able to find paths with minimal uncertainty (the blue ellipsoids are much smaller), from [185].

A comment we can make on trajectory smoothness (already mentioned in the Introduction of this thesis) is that smoothness may also play a decisive role in vision-based trajectory since a jerky camera motion with high angular acceleration especially will make the projection of a 3D point onto the image suffer from motion blur, making it very complicated, if not impossible, to extract meaningful information. This issue has been recently raised in [182] in the context of robust visual perception with a fast moving autonomous quadrotor. The authors adopt a MPC framework to optimize over perception objectives for providing robust and reliable visual feedback during motion. The authors choose to maximize the visibility of a collection of points of interest by penalizing the deviation of their projections from the image center and the velocity of their projections in the image plane (see Fig. 5.10a). Here, a forward-looking camera is attached. The quadrotor is able to exploit the height of the room to compensate for the pitch while moving to manage the visibility of points of interest. Moreover, the planning naturally mostly acts on the system's heading since rotating around $\boldsymbol{z}_{\mathcal{B}}$ directly affects the visibility and at a lower energy cost than for accelerating upwards (since the total thrust is also penalized).

Recently, the Robotics & Perception Group at ETH developed a new dynamic vision sensor or *event-based camera* [22] which is way less sensitive to motion blur and change of illumination and has a lower-latency compared to classic CCD cam-

eras. A second solution exploited in [21] is to control the exposure time of cameras to limit motion blur. Thus, motion blur issues can be managed both by hardware and control solutions.

In [186] the authors consider a nonholonomic robot that has to reach a goal area of a given size delimiting the admissible position uncertainty (Fig. 5.12). The authors implemented a RRT variant where uncertain states are modelled as boxes.

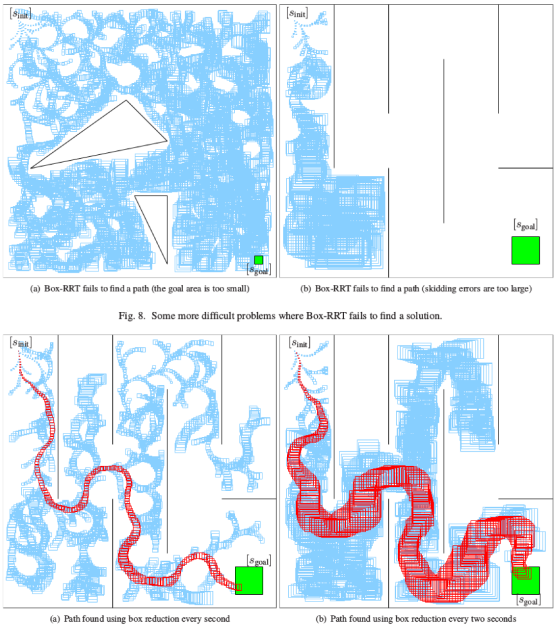


Figure 5.12 – A robust path is found for a unicycle. Notice that the problem may not be feasible if the final constraint is too tight or if the level of uncertainties is too large (upper pictures), from [186].

Other recent works have considered underactuated robots and sensor limitations in the context of active exploration [187, 188]. However, in these works the robot dynamics are simplified and the input constraints (i.e. the propeller speed) are not strictly imposed. An active sensing strategy considering the full quadrotor dynamics was proposed in [189], but without considering strict input constraints. Moreover, these works focus on environment coverage and a correct robot localization and none of them attempts to maintain visibility with respect to a specific set of features, which could, instead, be useful for target tracking applications.

5.6 Summary

In contrast to the presented works, our contributions merge vision-based and motor thrust constraints for the full dynamics of the quadrotor within fast and efficient

receding horizon frameworks that are capable of generating smooth and feasible optimal trajectories at the camera rate (30Hz) even though the problem is highly nonlinear. The re-planning strategies have been tested in various simulation tests and also with a real quadrotor by relying on an external motion capture Vicon system.

We have seen that a few works are tackling the issue of minimizing the uncertainty along a specific trajectory. However, depending on the environment topology and the visual features present in the scene, such a trajectory may return a solution that takes large detours before reaching the goal. Indeed, assuming an optimal solution exists providing continuous visual sensing (e.g., the system passes by every visual landmarks present on the scene), the resulting trajectory would be sub-optimal in terms of completion time and energy. We address this issue in our work presented in Chapt. 8.

Part II

Contributions

Chapter 6

Aggressive vision-based trajectory generation

Contents

6.1 Introduction

In this section we derive several optimal frameworks to perform reactive tracking of a moving target while ensuring visibility constraints. We demonstrate that the defined optimal problems are suited for a re-planning strategy inspired from MPC. To do so, we present our *hot-start* algorithm and the different techniques used for aiding the SQP solver converge to a local minimum within the given time allocation.

6.2 Reactive target tracking: a minimum-time optimal problem

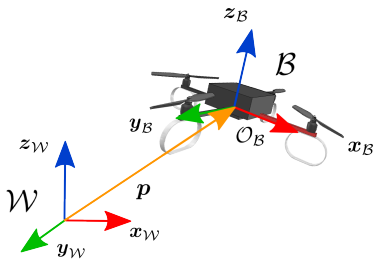


Figure 6.1 – Quadrotor model

Referring again to Fig. 6.1 let us assume the robot to be equipped with an on-board camera whose pose w.r.t. $\mathcal{B}$ is known from a preliminary calibration. Without loss of generality we assume that the camera is down-facing with optical center in $\mathcal{O}_{\mathcal{B}}$ and optical axis parallel to $\boldsymbol{z}_{\mathcal{B}}$. An image processing algorithm (whose design is beyond the scope of this work) provides a measure of the perspective projection of a collection of N fixed 3-D points w.r.t. the frame $\mathcal{B}$ given as follows

$$\boldsymbol{\beta}_i = \frac{\boldsymbol{R}^T(\boldsymbol{r}_i - \boldsymbol{r})}{\boldsymbol{z}_{\mathcal{B}}{}^T \boldsymbol{R}^T(\boldsymbol{r}_i - \boldsymbol{r})} = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \in \mathbb{P}^2, \quad i = 1, \dots, N \tag{6.1}$$

where $\boldsymbol{r}_i \in \mathbb{R}^3$ is the *known* position of the features in the inertial frame and $\mathbb{P}^2$ is the space of 3-D homogeneous vectors. We assume that the number of points and their configuration is such that the complete pose $(\boldsymbol{r}, \boldsymbol{R})$ of the robot can be reconstructed using visual information only. In particular, we consider $N = 4$ points on the ground plane since this is sufficient for our 3D case (one could also consider more complex features such as image moments [162]).

We also want to consider the field of view constraints so the object does not leave the image. The simplest way to solve this problem is to prescribe a maximum attitude angle (e.g., $\arccos(\boldsymbol{e}_3 \cdot \boldsymbol{R}\boldsymbol{e}_3) \leqslant \beta_{max}$). A trajectory could then be planned simultaneously using the maximum attitude constraint and the reduced field of view to constrain the relative positions. However, this approach is more conservative than desired, especially when aggressive maneuvers are necessary. Certainly, we do not want to restrict the maximum attitude. Instead, we directly incorporate the field of view as constraints in the optimization by defining the (square) image domain as

$$\Omega = \{\boldsymbol{\beta} \in \mathbb{P}^2 \ s.t. \ \max(\boldsymbol{\beta}^T \boldsymbol{x}_{\mathcal{B}}, \boldsymbol{\beta}^T \boldsymbol{y}_{\mathcal{B}}) \leqslant \tan(\alpha)\} \tag{6.2}$$

where α is the camera field of view: the measurement (6.1) is available iff $\boldsymbol{\beta}_i \in \Omega$. Later in Sect. 6.6 the visibility constraints will be defined on the unit sphere considering the spherical projection of the image features.

The sensory equipment is completed by an inertial measurement unit (IMU) providing a measure of the robot angular velocities $\boldsymbol{\omega}$ and specific force $\boldsymbol{R}^T(\dot{\boldsymbol{v}} - \boldsymbol{g})$ at a much higher frequency than the camera frame rate. We assume that a state estimator, such as the ones described in [190, 79], uses the visual and inertial measurements to provide an estimation of the current robot state at the IMU rate. Note, however, that between two image frames, the robot pose estimation can only be updated by dead-reckoning of the IMU data. Due to noise and IMU biases, this "inter-frame" estimation is expected to be of much lower accuracy than the one obtained after visual measurements.

6.2.1 Problem definition

Thanks to the flatness property it is possible to move the trajectory planning problem from the control space to the output space. Since the flat transformation is invertible, as it has been shown in Appendix A, we can transform the conditions on the initial and final states in equivalent conditions on the flat outputs and their derivatives that we indicate with

$$\boldsymbol{\sigma}_i = (\boldsymbol{r}_i, \boldsymbol{v}_i, \dot{\boldsymbol{v}}_i, \ddot{\boldsymbol{v}}_i, \psi_i, \dot{\psi}_i) \tag{6.3a}$$

$$\boldsymbol{\sigma}_f = (\boldsymbol{r}_f, \boldsymbol{v}_f, \dot{\boldsymbol{v}}_f, \ddot{\boldsymbol{v}}_f, \psi_f, \dot{\psi}_f) \tag{6.3b}$$

Given the dynamic model (2.16), the input transformation (2.13), and the measurement equation (6.1), at a generic time t, we seek for a solution to the following optimization problem.

Problem 5 *Find* $T, \boldsymbol{\chi}(s), \boldsymbol{u}(s), s \in [t, t+T]$, *such that:*

$$\min_{\boldsymbol{\chi}(s), \boldsymbol{u}(s), T} \quad T \tag{6.4a}$$

$$s.t. \quad \boldsymbol{\chi}(t) = \boldsymbol{\chi}_t \tag{6.4b}$$

$$\boldsymbol{\chi}(t+T) = \boldsymbol{\chi}^* \tag{6.4c}$$

$$\dot{\boldsymbol{\chi}} = \boldsymbol{h}(\boldsymbol{\chi}, \boldsymbol{u}) \tag{6.4d}$$

$$\boldsymbol{u}(s) \in \mathcal{U}, \forall s \in [t, t+T] \tag{6.4e}$$

$$\boldsymbol{\beta}_i(s) \in \Omega, \forall s \in [t, t+T], i = 1, \ldots, N \tag{6.4f}$$

where $\boldsymbol{\chi}_t$ is the current robot state, $\boldsymbol{\chi}^*$ is the desired one, and (6.4d) was introduced to represent (2.16) in a compact form.

As shown in Sect. 3.2 Problem 5 does not impose any constraint on the initial and final states, however, if both $\boldsymbol{\chi}_t$ and $\boldsymbol{\chi}^*$ are *hovering* states, $\boldsymbol{\beta}_i(t) \in \Omega, \boldsymbol{\beta}_i(t+T) \in \Omega, \forall i = 1, \dots N$, and the *hovering input* $\boldsymbol{u} = (mg/4, mg/4, mg/4, mg/4) \in \mathcal{U}$ is not an isolated point in $\mathcal{U}$, then a solution to Problem 5 always exists. Indeed, in this case, it is always possible to find a sufficiently large T such that the robot moves in near-hovering conditions, along a feasible and *almost* straight trajectory from the initial pose to the desired one [1, 112]. Now, due to the absence of rotational motions, the linear trajectory in 3-D space is also mapped to linear trajectories of the image features from $\boldsymbol{\beta}_i(t)$ to $\boldsymbol{\beta}_i(t+T)$. Thanks to the convexity of the image domain Ω, this guarantees that the feature visibility will be maintained.

Problem 5 contains non-linear algebraic and differential constraints and, to the best of our knowledge, does not admit an explicit analytic solution. As it is often the case in these situations, we then attempt to find a sub-optimal solution using a numeric resolution strategy as discussed in the next section.

6.3 Numerical resolution

In its original form, Problem 5 is not suited for a direct numerical resolution. First of all, the system dynamic equation (6.4d) represents a non-linear differential equality constraint, which is particularly hard to deal with in a numerical resolution scheme. In addition to this, the search space of the problem (the control input time law $\boldsymbol{u}(t)$) is infinite dimensional. As explained in Sect. 3.5 in order to overcome these problems, we exploit differential flatness for eliminating constraint (6.4d) and we use B-spline parametrization to obtain a finite representation of the search space.

As mentioned in the above section, the robot can move from any state to any other, provided that the limits on the propeller rotational speeds are not too strict. As a consequence, whatever are the initial and final states there always exist a feasible trajectories. Nevertheless with the introduction of the B-spline parametrization, we have reduced the search space so that it may not contain these feasible transfer trajectories. Moreover, even when our search space contains some feasible solutions, it may not contain the optimal one in the sense that we might still find a better solution if we enlarged the search space.

Problem 5 can be restated as a NLP as follows.

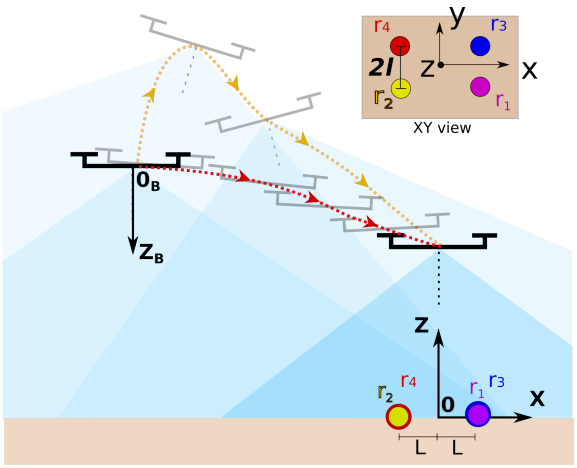


Figure 6.2 – Consider two pairs of dots on the ground horizontal plane (XY view in the upper right corner). It would be possible to cope with the field of view constraints by planning a near hovering trajectory (e.g. path in red), but in this work we aim at finding a trajectory similar to the path in yellow which is much more dynamic (and with a shorter completion time).

Problem 6 *find* $\boldsymbol{P}, T$*, such that:*

$$\min_{\boldsymbol{P},T} \quad T \tag{6.5a}$$

$$s.t. \quad \boldsymbol{\sigma}_{\chi}(t) = \boldsymbol{\sigma}_{\chi_t}, \tag{6.5b}$$

$$\boldsymbol{\sigma}_{\chi}(t+T) = \boldsymbol{\sigma}_{\chi^*}, \tag{6.5c}$$

$$\boldsymbol{\beta}_i(s) \in \Omega, \forall s \in [t, t+T], i = 1, \dots, N, \tag{6.5d}$$

$$\boldsymbol{u}(s) \in \mathcal{U}, \ \forall s \in [t, t+T], \tag{6.5e}$$

where $\boldsymbol{\sigma}_{\chi_t} = \phi_{\chi}^{-1}(\boldsymbol{\chi}_t)$ *and* $\boldsymbol{\sigma}_{\chi^*} = \phi_{\chi}^{-1}(\boldsymbol{\chi}^*)$.

At this point, any general-purpose optimization strategy can be used to find a numerical solution to Problem 6. Unfortunately, due to the non trivial non-linearity of (6.5d–6.5e), Problem 6 cannot be proven to be convex. The optimization will thus, in general, return a local minimum.

6.4 Recursive online control

Once Problem 6 is solved, the resulting flat output trajectory could be used in (2.23) for computing the control inputs $\boldsymbol{u}$ to be fed to the system. In practice, however, different sources of disturbance (e.g. noise, miscalibrations, neglected dynamics, and so on) will make the robot to quickly diverge from the planned trajectory when using such an open-loop control strategy. In order to cope with these uncertainties and

disturbances, we then incorporate a *feedback* action in the considered optimization schemes.

The reasons are multiple: i) since the B-spline order is minimal the snap is piecewise continuous and the inputs can be very sharp (see Fig. 6.3). If one sends such values to the controller we can observe a deviation from the original trajectory. Now, if one increases the B-spline order we can see that the system model integration on smoother inputs result in a more accurate resulting trajectory. ii) having an extra feedback action from the controller provides more robustness to uncertainties and a higher stability.

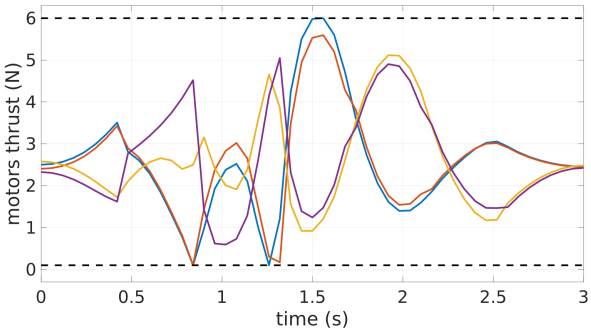


Figure 6.3 – Example of (bounded) motors thrust profiles considering a parametrization of the flat outputs with a 4-th order B-spline basis.

As already introduced in Sect. 4.2 we take inspiration from Model Predictive Control [191] to perform an *on-line* re-planning of an optimal trajectory by solving Problem 6 each time a new visual measurement is available. By doing so, we expect to improve the system performance while, more importantly, ensuring the satisfaction of the visibility constraint (6.5d). Finally, instead of feeding the optimal inputs sequence directly to the system we send the optimal trajectory to the trajectory controller [51] as in [143, 1] at the solver rate.

On the one hand, this allow to reject, to some extent, the disturbances acting on the system. On the other hand, however, the optimality of the resulting trajectory can be compromised and, more importantly, the visibility constraints (6.5d) can be violated.

6.4.1 Trajectory re-planning strategy

A major issue in the implementation of receding horizon control is handling the computational delay associated with the real-time optimization. We present here our method for designing an initial guess and take delay into consideration for the re-planning.

The re-planning strategy is best explained by a visual example, shown in Fig. 6.4.

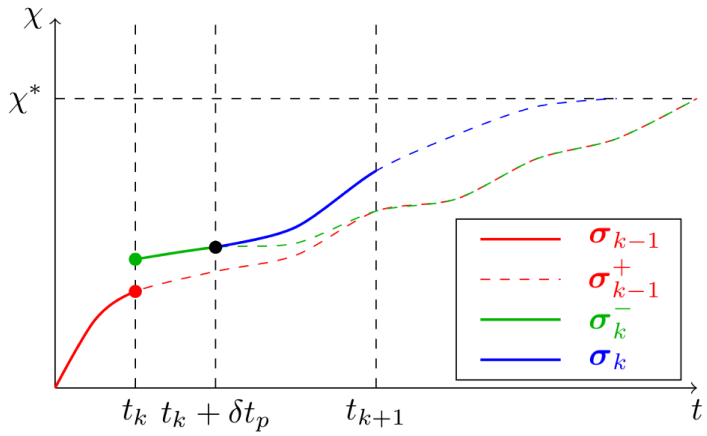


Figure 6.4 – Single instance of the re-planning process. The red line represents the trajectory computed in a previous planning iteration. The robot is following this trajectory when, at time t_k, a visual measurement and a new state estimation become available (green dot). The red trajectory is split and clamped to this measurement, resulting in the green line. The first part of this latter is immediately used as reference for the controller. The second part (the dashed green line) is fed as initial guess to the solver of Problem 6, and also used to predict the state in which the system will be at time $t_k + \delta t_p$, when the optimization will be over. Finally, the blue line is the new optimal trajectory resulting from the numerical resolution of Problem 6. The process is repeated again at t_{k+1}, when a new measurement is available.

Let us assume that, in a previous planning step, at time $t = t_{k-1}$, the resolution of Problem 6 generated a trajectory $\boldsymbol{\sigma}_{k-1}$, represented in red in the figure. The system is now at time $t = t_k$ and a new visual measurement becomes available to be used in the innovation step[1] of the state observer to produce an estimation of the current system state $\hat{\boldsymbol{\chi}}_t$. This estimation will, in general, be different from the expected system state $\Phi_\chi(\boldsymbol{\sigma}_{k-1}(t))$ due to the non-idealities mentioned above. A new optimal trajectory should, hence, be planned by solving Problem 6 and using the current state estimate to compute the initial condition $\boldsymbol{\sigma}_{\chi_t}$.

Unfortunately, the resolution of Problem 9 requires a non-negligible time to complete. This time will, in general, vary, depending on the quality of the initial guess for the optimization variables, on the number of necessary iterations and on the available computational resources. Here, for simplicity, we assume that the processing will be concluded after, at most, a constant maximum duration δt_p, possibly by introducing a watchdog timer and accepting an intermediate sub-optimal solution.

For computing the system control inputs while the optimization is running, we

[1] Note that we trigger the planning at camera rate and not at the estimation rate. This is motivated by computational limitations and by the fact that, as already mentioned, the inter-frame estimation obtained by dead reckoning is expected to have a much lower accuracy.

simply "adapt" the previous trajectory to the new initial conditions by using a fast procedure that does not involve the resolution of Problem 6. First of all, we *split* the trajectory $\boldsymbol{\sigma}_{k-1}$ at time t_k, as described in Sect. 6.4.2, to extract only its second part $\boldsymbol{\sigma}_{k-1}^{+}$ (the dashed red curve in Fig. 6.4). Then, we look for a new trajectory $\boldsymbol{\sigma}_{k}^{-}$ (represented in green in Fig. 6.4) that is "as close as possible" to $\boldsymbol{\sigma}_{k-1}^{+}$, but starts from $\Phi_{\chi}^{-1}(\hat{\boldsymbol{\chi}}_t)$. Details about this step are provided in Sect. 6.4.3. Note that this "temporary" trajectory $\boldsymbol{\sigma}_{k}^{-}$ is sub-optimal and its calculation does not take into account any of the actuation and visibility constraints (6.5d–6.5e), which, as a consequence, could be violated. However, we accept this risk in order to be able to provide an immediate update of the reference trajectory to the new state estimation while a better solution is being computed by appropriately resolving Problem 9 as follows.

During the optimization process, the system will, most probably, move away from the current state $\boldsymbol{\chi}_t$. As a consequence, if $\boldsymbol{\chi}(t)$ were used as initial condition in (6.4b) (or, equivalently, (6.5b)), the newly planned trajectory would not start from the actual state of the robot at time $t + \delta t_p$. We mitigate this problem by using the trajectory $\boldsymbol{\sigma}_{k}^{-}$ also to predict (by a simple B-spline evaluation) the value of the flat outputs corresponding to the state $\hat{\boldsymbol{\chi}}_{t_{opt}}$ in which the system will be when the optimization will be over. This value is used as initial condition in Problem 6.

Finally, since we use recursive optimization methods to find a solution to Problem 6, we also need to provide an initial guess for the optimal trajectory. This initial guess is computed by splitting the trajectory $\boldsymbol{\sigma}_{k}^{-}$ at time $t + \delta t_p$ (green dashed line in Fig. 6.4) as described in Sect. 6.4.2 and taking the second part (green dashed line in Fig. 6.4) of the trajectory.

The optimization can finally run and a new optimal trajectory (the blue one in Fig. 6.4) will be generated. Such trajectory will be used to control the system starting from time $t + \delta t_p$ until a new measurement becomes available at time $t = t_{k+1}$. At the arrival of a new measurement the above procedure is repeated.

This strategy allows to re-plan online an optimal trajectory each time a new visual measurement is available. Each one of the generated trajectories could be used directly in (2.23b) to calculate the robot inputs. As already mentioned, however, an alternative possibility is, instead, to use them as references for a fast trajectory tracker. This second possibility is appealing because it allows to fully exploit the sensing capabilities of the robot: between two visual measurements, in fact, an estimation of the quadrotor state can be obtained, at a much higher frequency, by using the IMU for dead reckoning. A fast trajectory tracker can, thus, use this information to reduce the effect of non-idealities between two planning steps.

Note that, as the quadrotor approaches the desired state, the planning distance

and time horizon tend to zero, potentially introducing numerical issues in the resolution of Problem 6. To overcome this problem, when the system is close to the desired goal, we deactivate the re-planning and directly feed the trajectory tracker with the desired state $\boldsymbol{\chi}^*$.

6.4.2 B-spline splitting

An advantage of using B-spline trajectories for motion planning is that there exist lightweight and easy algorithms to perform different manipulations on their shape. One such manipulation, that we perform multiple times in the recursive algorithm described in Sect. 6.4.1, is the *splitting*. Details about how to split a B-spline curve at a point and how to calculate the knots and control points of the resulting parts can be found in many sources, such as [192].

An undesirable effect of the splitting operation is that it also modifies the knot sequence and possibly (depending on the position of the split) even eliminates some knots. In order to maintain a constant number of uniformly distributed knots (and thus a constant number of control points acting "evenly" on the whole spline length), after the split, we perform a sequence of *knot insertion* and *knot removal* operations (see [192]) meant to redistribute the knots of the new trajectory evenly.

De Boor's algorithm is a generalization of de Casteljau's algorithm. It provides a fast and numerically stable way for finding a point on a B-spline curve given a u in the domain. The core of the algorithm lies in the knot multiplicity rule: if a knot u is inserted m times to a B-spline/NURBS curve, the last generated new control point is the point on the curve that corresponds to u. Meaning that we only need to insert u enough number of times so that u becomes a knot of multiplicity m. If u is already a knot of multiplicity s, then inserting it $m - s$ times would be sufficient. Indeed, after inserting u m times, the triangular computation scheme yields one point. Because the given B-spline/NURBS curve must pass by this new point, it is the point on the curve corresponding to u. Note that this argument holds even if u is inserted as an existing knot. The depicted procedure is more formalized in Appendix B.2.1. This technique is also applied to evaluate the spline at u. We simply need to insert u m times and the last point is $\boldsymbol{p}(u)$

Since the given B-spline curve is subdivided at its knots, each curve segment has no internal knots. Moreover, the subdivision process makes the internal knots to have multiplicity $m + 1$, and the curve segment is "clamped" at the first and last control points of each curve segment.

In the process of subdividing a B-spline curve, a large number of control points will be introduced. Therefore, manipulating a B-spline curve is easier than manipulating its component Bézier curves. Moreover, the B-spline curve of degree p is

C^{p-m} continuous at a knot point, where m is the multiplicity of the corresponding knot. When we manipulate a B-spline curve by moving control points, this continuity is always maintained. However, if a B-spline curve is subdivided into a sequence of Bézier curves, maintaining the continuity at the joining control points would be a challenging task. Consequently, handling a B-spline curve is much easier than handling a sequence of Bézier curves.

6.4.3 Adapting previous trajectories to new initial conditions

In this section we describe how to efficiently "adapt" a previously computed B-spline trajectory (e.g. the trajectory $\boldsymbol{\sigma}_{k-1}^{+}$ represented by a red dashed line in Fig. 6.4) to a new estimation of the current robot state (green dot in Fig. 6.4). To perform this operation we exploit two important properties of B-splines:

- The *local support* property stands that the shape of the curve in a knot span (s_k, s_{k+1}) is only determined by a subset of k of the B-spline control points.
- The *convex hull* property guarantees, instead, that in each knot span, the spline curve is locally contained in the convex hull of the same subset of control points. In practice this allows to conclude that changing the first control points (those determining the initial state of the system) will not affect the shape of the spline towards its end (in particular the final system state will not change) and that two splines with similar control points (according to some norm) are also geometrically close to each other.

Given a spline $\boldsymbol{\sigma}_{k-1}^{+}$, with control points $\boldsymbol{P}$, the control points $\boldsymbol{P}^{-}$ of the new spline $\boldsymbol{\sigma}_{k}^{-}$ can then be computed by solving the following linear quadratic optimization.

Problem 7 *Find a vector of control points $\boldsymbol{P}^{-}$ such that*

$$\min_{\boldsymbol{P}^{-}} \quad \frac{1}{2}\sum_{j=1}^{n_r}\left\|\boldsymbol{r}_j - \boldsymbol{r}_j^{-}\right\|^2 + \frac{1}{2}\sum_{j=1}^{n_\psi}\left\|\psi_j - \psi_j^{-}\right\|^2 \tag{6.6}$$

$$s.t. \quad \boldsymbol{\sigma}_{\chi}(t) = \boldsymbol{\sigma}_{\chi_t}, \tag{6.7}$$

Note that Problem 7 does not take into account the actuation and visibility constraints in (6.5d–7.10h). While we cannot formally guarantee that these constraints will not be violated, we want to stress that the resulting trajectory is only used for a short amount of time, namely the time needed for the numerical resolution of Problem 6. Introducing a saturation of the control commands one still guarantees the satisfaction of (7.10h) at the cost of introducing a deviation of the robot from its nominal trajectory. Finally, by introducing some *security margins* in the definition of Ω, one could also reduce the probability of losing feature track in practice.

6.5 Simulation setup and results

In this section we report the results obtained by using our planning method in a physically realistic simulation environment. In this thesis we used the on-the-shelf optimization library NLOPT [193] that we present below.

6.5.1 The NLOPT algorithm

NLOPT is a free/open-source library for nonlinear optimization and implements a number of optimization algorithms routines in including: It also provides stopping routines to stop iterating once some termination criterion is satisfied, e.g., maximal number of iterations or function-value $ftol$, step tolerance $xtol$ and especially the maximal running time (which is not the case of ACADO) to control the minimal variations in local searches and to stop when sufficient precision is reached. This feature is desirable for our re-planning strategy.

More precisely, we use the SQP C++ routine implemented in NLOPT as the SLSQP (Sequential Least-Squares Quadratic Programming) algorithm from [194]. It optimizes successive second-order (quadratic/least-squares) approximations of the objective function, with first-order (affine) approximations of the constraints. The approximations of the objective function are done via the Broyden-Fletcher-Goldfarb-Shanno method (BFGS) to build an approximation of the Hessian matrix.

6.5.2 Simulation results

The quadrotor dynamics were simulated using V-Rep[2] with a time step of 6 ms. The planning strategy described in Sects. 6.3 and 6.4 was implemented in C++ and the SQP method of NLOPT was used to numerically resolve Problem 6. The generated trajectories were sent to TeleKyb which then computed the actual control inputs using controller [79, 143].

We simulated visual measurements at a rate of 15Hz for four targets positioned in $(\pm 0.2, \pm 0.1)$. In our implementation, each planning operation (resolution of Problem 6) takes about 30ms during which the system uses an adaptation of a previously planned trajectory, obtained by resolving Problem 7. Thus, a new trajectory is sent to the controller at the rate of 30Hz.

The simulated camera had a field of view of 90 degrees ($\alpha = \pi/4$) and each propeller could generate thrusts between 0.1 N and 7 N. For realism purposes, we introduced a Gaussian noise into the state measurements (up to 2% absolute error) and into the motors thrust sent by the controller (up to 5% absolute error). We also purposely used different inertial parameters for the re-planning algorithm and for

[2] http://www.coppeliarobotics.com/

the actually simulated quadrotor in V-Rep in order to introduce presence of (typical) modelling errors between planned trajectory and actual execution. In particular, we used the following values:

	mass	Inertia matrix (diagonal)
Planning	1.0	(0.01562 0.01562 0.03125)
Simulation	1.08	(0.016 0.0145 0.027)

Table 1. Inertial parameters used for the re-planning and in V-Rep

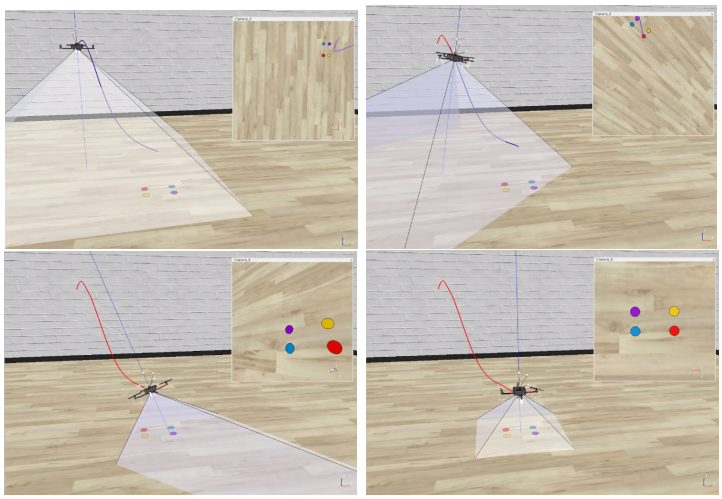


Figure 6.5 – Successive snapshots taken from V-Rep at different time instants. The straight line represents the vertical axis of the camera, the blue line is the planned trajectory and the red line is the actual system trajectory. The camera view is shown in the upper right corner.

Figure 6.5 shows some snapshots of the simulation. The robot started from an initial hovering state at $\boldsymbol{r} = (-1.1, 1.1, 2)$ and $\psi = 1.6$ rad and was required to reach another hovering state with $\boldsymbol{r}^* = (0, 0, 0.6)$ and $\psi^* = 0$. The solid red line in Fig. 6.5 shows the resulting quadrotor trajectory in space while the blue line represents the currently planned trajectory. Figure 6.6 shows the predicted evolution (given the currently planned trajectories) of the four points in the image plane at equally spaced time instants. The actual evolution of the four image point coordinates is shown in Fig. 6.7 whereas Fig. 6.8 shows the thrust generated by each propeller. The dashed lines in Figs. 6.6 to 6.8 represent the constraints.

The robot was able to accomplish the task in a total time of approximately 2.3 s over which the trajectory planning algorithm was triggered 34 times.

During motion, the quadrotor reached a translational speed up to 1.0 m/s along the X axis, and rotations up to 20 deg as illustrated in Fig. 6.9. From Fig. 6.7

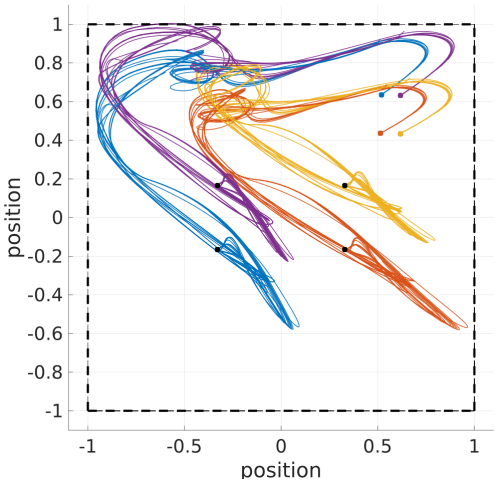


Figure 6.6 – Image feature trajectories planned at different planning steps. Four dashed segments represent the boundaries of the image domain. The image features are initially in the upper right corner.

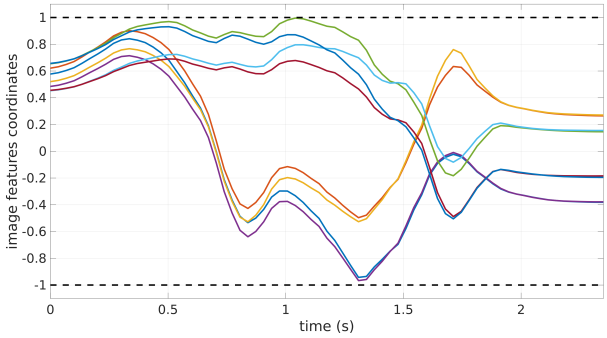


Figure 6.7 – Actual image feature coordinates measured during the re-planning. The horizontal dashed lines represent limits of the image domain.

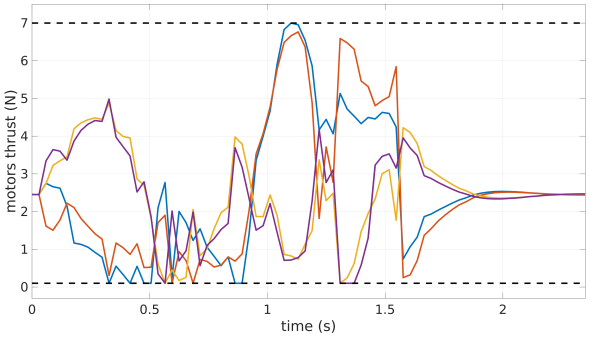


Figure 6.8 – Motor thrusts evolution for the four propellers with horizontal dashed lines representing the actuation domain $\mathcal{U} = [0.1, 7]$.

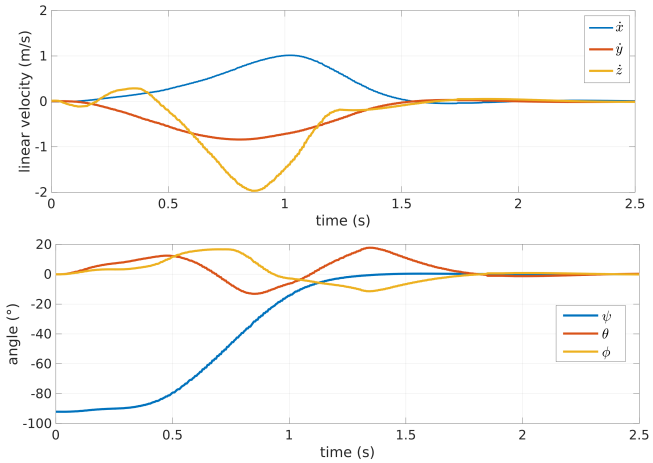


Figure 6.9 – Linear velocity (upper figure) and pitch and roll angles (bottom figure) during motion.

one can see that the features moved very close to the limits of the field of view. Finally Fig. 6.8 shows that also the motor thrusts hit the actuation limits. These results clearly show that the performed trajectory was rather aggressive and that the actuation and sensing capabilities of the robot were exploited. Therefore, we showed that in the presence of modelling uncertainties and noise, the feedback introduced by updating the reference trajectory was able to reject some of these disturbances while satisfying the several constraints. We encourage the reader to watch the video[3] attached to the concerned contribution [2]: there, we show how an "open-loop" execution of the initially planned trajectory quickly fails to meet the visibility constraints because of the (purposely introduced) actuation noise and model uncertainties. On the other hand, as discussed, the online re-planning allows gaining a sufficient level of robustness against these non-idealities. As it is common, a high planning rate is privileged against optimality to some extent.

The rest of this chapter presents our second contribution [3] that addresses more complex vision-based tasks.

6.6 Vision-based target tracking

In this section, we first address the case of tracking a moving 3D target with a front-looking camera and ensuring final visibility of the target at the (unique) image center. We assume that the relative pose of the target can be estimated using vision only as done in [106] with some preliminary knowledge of the target model (e.g., the target radius in case of a sphere). We are also interested in allowing the quadrotor

[3] https://www.youtube.com/watch?v=mZrS2wutZCI

undertaking aggressive manoeuvres for reaching the target in near minimum-time conditions. The final goal is to reach a hovering state such that the target appears at the image center while keeping a safety distance from the target (see Fig. 6.10).

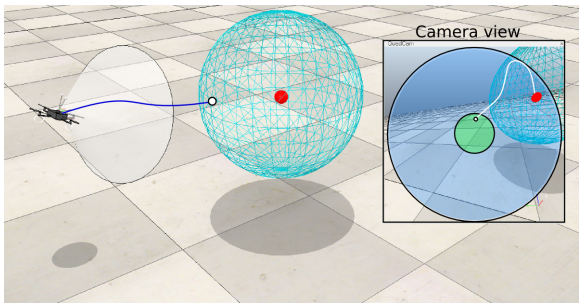


Figure 6.10 – The quadrotor has to follow the target in red while keeping a safety distance represented by the light blue sphere. The terminal constraint in the image space is represented by the green circle. The blue circle represents the field of view inside which the feature trajectory must lie. Here, the optimal trajectory in blue steers the quadrotor towards a final position (white dot) and the resulting image feature trajectory is the one in white

6.6.1 Multi-objective cost function

For achieving the aforementioned behaviour we adopt multi-objective programming and minimize the following weighted cost function at running time t with weights $w_i > 0 \in \mathbb{R}$.

$$\begin{aligned} J = w_1 \int_t^{t+T} \|\ddot{\boldsymbol{v}}(t)\|^2 dt + w_2 \int_t^{t+T} \left\|\ddot{\psi}(t)\right\|^2 dt \\ + w_3 \int_t^{t+T} \|d(t) - R_s\|^2 dt + w_4 \int_t^{t+T} \|\boldsymbol{v}(t)\|^2 dt \end{aligned} \tag{6.8}$$

The latter is divided into four parts: the first and the second terms minimize the norm of the snap and the norm of the yaw acceleration respectively. They are used for encouraging smoothness while still exploiting the quadrotor aggressiveness [1]. The third term minimizes the error between the Euclidean distance $d(t)$ of the target to the camera and a value R_s defined as a safety distance: the radius square of the sphere centred on the target at position $\boldsymbol{q}$ (see Fig. 6.10). Naturally, to achieve more aggressive trajectories, one can enforce this cost. The fourth term penalizes the path length. It is an approximation of the sum of the path segments $\sum_{k=0}^{k=N-1} \|\boldsymbol{r}(k+1) - \boldsymbol{r}(k)\|^2$ which appears to be equivalent to penalizing the linear velocity over the time horizon. Indeed, for a circular symmetric target the quadrotor may fly around the target indefinitely without changing the image feature position, an effect arising from the residual available degrees of freedom and that one can prevent by adding this fourth term.

In the previous section (Sect. 6.2) we minimized the completion time T to generate aggressive trajectories [2] . Here, we are dealing with more complex constraints and we will show that we can still exploit the quadrotor's agility. Moreover, we avoid the penalization of such a complex parameter and the re-evaluation of the B-spline bases at each solver iteration. The completion time T is then a fixed parameter and should be chosen as a rough upper bound of the time required to reach the final pose vicinity. If T is too small, the trajectories might be infeasible. If too high, the system may be less reactive.

Finally, by suitably weighing these costs we are able to trigger the expected behaviour and prevent the objectives from conflicting with each other.

6.6.2 Visibility constraints

In this section we propose an alternative to (6.5d) for the formulation of constraints on the image features. Let us define the *spherical* projection of a target point in 3D w.r.t. the frame $\mathcal{B}$ as the bearing vector

$$\boldsymbol{\beta} = \frac{\boldsymbol{R}^T(\boldsymbol{r} - \boldsymbol{r})}{\|\boldsymbol{R}^T(\boldsymbol{r} - \boldsymbol{r})\|} = \frac{\boldsymbol{m}}{\|\boldsymbol{m}\|} \in \mathbb{S} \tag{6.9}$$

where $\boldsymbol{r} \in \mathbb{R}^3$ is the position of the feature in the world frame and $\mathbb{S}$ is the surface of the unit sphere and $\boldsymbol{m} = (u, v, 1)$ is the image measurement from which $\boldsymbol{\beta}$ is computed.

Defining $\boldsymbol{e_c}$ as the camera optical axis in the frame $\mathcal{B}$, namely, $\boldsymbol{x}_{\mathcal{B}}$ (or $\boldsymbol{y}_{\mathcal{B}}$) for a front-looking camera and $-\boldsymbol{z}_{\mathcal{B}}$ for a down-looking camera, the visibility constraint is written as

$$\boldsymbol{\beta}^T(t)\boldsymbol{e_c} \geqslant \cos(\alpha/2), \forall s \in [t, t+T] \tag{6.10}$$

where α is the angle of view of the camera and $\boldsymbol{\beta}$ is given by (6.1). Fig. 6.11 shows that (6.10) is equivalent as constraining the feature bearing angle β but whose numerical evaluation is more complex. In contrast to the previous formulation (6.5d) a single constraint is defined using (6.10). However, the field of view of the camera is modelled as a cone (see Fig. 6.10) which is a less realistic representation than a pyramid-shaped field of view.

Now, instead of imposing equality constraints on the final position in order to guarantee visual convergence of the target to the camera center, we rather define a terminal constraint such that the feature bearing angle has to belong to a (smaller) angular area at the camera centre (see Fig. 6.10) with

$$\boldsymbol{\beta}^T(t)\boldsymbol{e_c} \geqslant \cos(\gamma/2), \forall s \in [t + T_v, t + T], \quad 0 \leqslant T_v \leqslant T \tag{6.11}$$

where γ is the angle defining the circular region of convergence in the image. The time T_v is a parameter that defines at what time the feature shall enter the vicinity

region. It can be tuned to affect the convergence rate towards this region. Namely, a value closer to 0 will demand a longer activation of the constraint. T_v will also vary depending on the camera orientation due to the quadrotor dynamics.

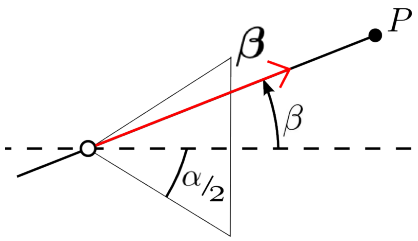


Figure 6.11 – The visual constraints (6.10) is equivalent as ensuring that the angular position β of an image point P is lower than the field of view angle $\alpha/2$

With the above definitions, one can encapsulate visibility constraints and visual convergence for any camera orientation. The approach considered in [101] penalizes the motion aggressiveness since it minimizes the deviation of the image features from the center of the camera in the image plane. Therefore it does not fully exploit the image space while, in our case, the target is free to move away from the camera center in order to allow large rotations of the camera and therefore large accelerations of the quadrotor.

Following the same strategy as in Sect. 6.2, we exploit differential flatness and parametrize the flat outputs with B-splines with control points $\boldsymbol{P}$. We define the following problem with the cost function (6.8)

Problem 8 *Find* $\boldsymbol{P}$ *such that:*

$$\min_{\boldsymbol{P}} \quad J \tag{6.12a}$$

$$s.t. \quad \boldsymbol{\sigma}_\chi(t) = \boldsymbol{\sigma}_{\chi_t}, \tag{6.12b}$$

$$\boldsymbol{\sigma}^{(i)}(t+T) = \boldsymbol{\sigma}_{\chi^*}, i = 1, \dots, 3, \tag{6.12c}$$

$$\boldsymbol{u}(s) \in \mathcal{U}, \forall s \in [t, t+T], \tag{6.12d}$$

$$\boldsymbol{\beta}^T(t) e_c \geqslant \cos(\alpha/2), \forall s \in [t, t+T], \tag{6.12e}$$

$$\boldsymbol{\beta}^T(t) e_c \geqslant \cos(\gamma/2), \forall s \in [t+T_v, t+T] \tag{6.12f}$$

6.7 Simulation and experimental results

The video[4] attached to this work [3] shows the reactive target tracking considering the scenario depicted in Fig. 6.10 to validate the proposed replanning strategy with the visibility constraints detailed in Sect. 6.6.2. Figure 6.12 shows successive snapshots of the simulated environment. We show in Fig. 6.13 that the computed inputs

[4] https://www.youtube.com/watch?v=mvvF1I72HM8

and images features trajectories are maintained within their allowed domains. We exploit again the replanning strategy described in Sect. 6.4.1 and the SQP optimization routine from NLOPT to compute the optimal solutions.

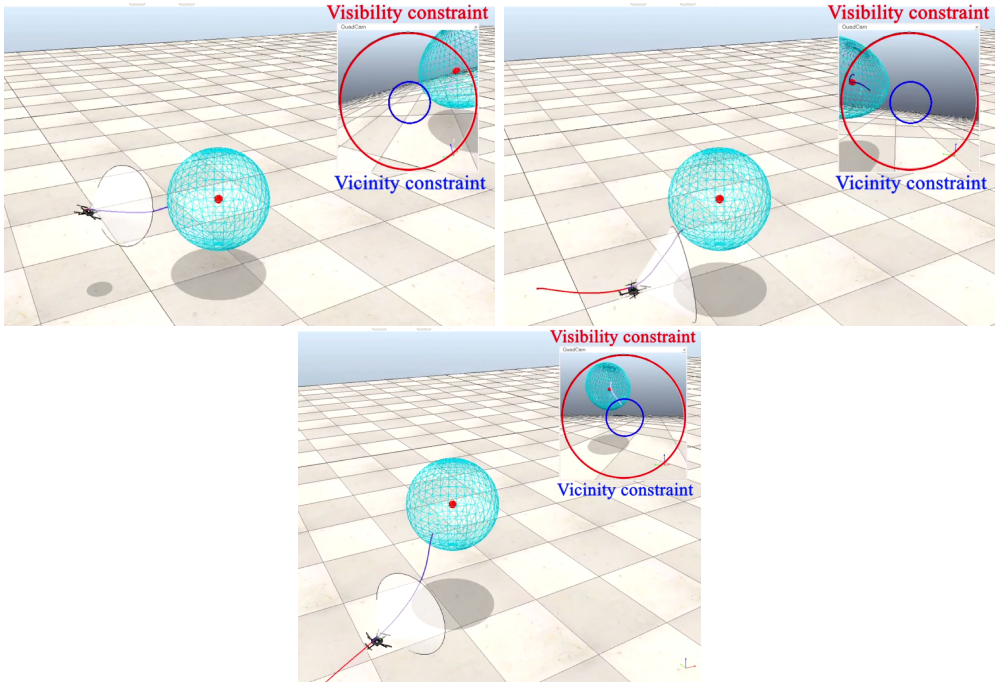
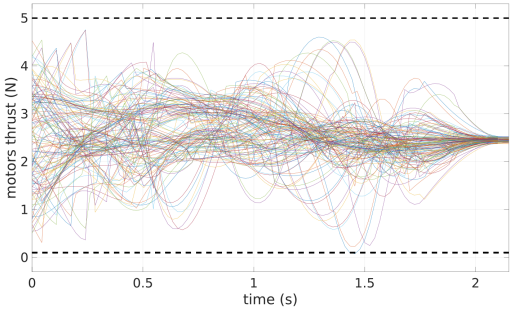
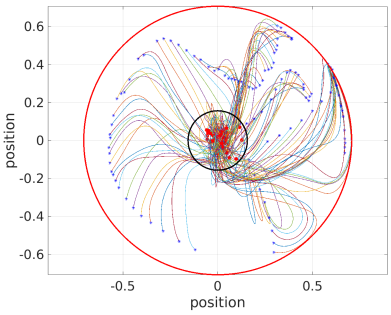


Figure 6.12 – Successive snapshots taken from V-Rep at different time instants. The blue line is the planned trajectory and the red line is the actual system trajectory. The camera view is shown in the upper right corner where the red circle represents the field of view limits and the blue circle represents the vicinity constraint (6.11).



(a) Optimal motor thruts profiles constrained between 0.1N and 5N.



(b) Images features trajectories in the image plane with a field of view of 90°.

Figure 6.13 – A representative set of the computed inputs Fig. 6.13a and image features Fig. 6.13b trajectories from the resolution of Problem 8.

We also conducted the same re-planning strategy with a real quadrotor. We used a MK-Quadro equipped with a front-looking camera with a field of view of

45°. The setup included an on-board ODROID-XU4 Linux computer running ROS and the TeleKyb framework for interfacing the replanning algorithm which ran on a standard desktop PC (Quadcore Intel i7 CPU@2.6 GHz). A Vicon motion capture system was employed for giving state measurements of the quadrotor and position measurements of the tracked target. We used AprilTags as a generic target attached on top of another MK-Quadro controlled remotely (Fig. 6.14). The video shows the general behaviour of the system.

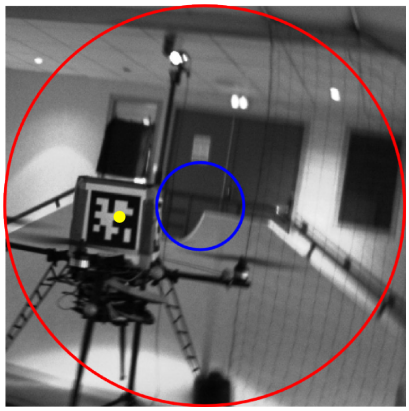


Figure 6.14 – The center of the box (yellow dot) attached on a second quadrotor and covered by four ApriTags had to remain inside the field of view of the camera (red circle) during the entire flight and appear inside the blue circle in the end of the computed trajectories

Chapter 7

On collisions and occlusions avoidance

In this chapter we address vision-based navigation in the presence of obstacles for target tracking applications. Since we consider the visibility constraint on the tracked target it is crucial to also avoid the occlusions generated by the obstacles themselves with the target. To do so, we design a new constraint formulation in the image space by drawing the analogy with volumetric constraints used for collision avoidance. In order to improve the convergence towards a local minimum we use complex-step differentiation (CS) to efficiently approximate the gradients of the nonlinear terms in the cost function and of the nonlinear constraints.

7.1 Contributions

In Sect. 6.2 we presented preliminaries for online minimum-time trajectory re-planning under field of view constraints. In this chapter we show that reactive manoeuvres can still be achieved without minimizing time and in the presence of more complex constraints. Besides, we improve the accuracy and the numerical stability of the gradients evaluation by using complex-step differentiation which aids the SQP convergence. Our method differs from most relevant works (e.g., [180, 181, 179]) under three main aspects:

- efficient and reactive online re-planning strategy considering an underactuated robot
- soft occlusion avoidance formulation in the image space
- visual constraints independent of the camera/UAV configuration

7.2 Constraints formulation

In this section, we describe the vision-based optimization scheme for avoiding collisions with static spherical obstacles of radius R_{occ} and the occlusions they might generate with the tracked target, see Fig. 7.1. Collision-free trajectories with N static spherical obstacles of inflated radius $R_{col} > R_{occ}$ at position $\boldsymbol{r_{obs}}$ are generated using volumetric constraints such as

$$\|\boldsymbol{r} - \boldsymbol{r_{obs}}_i\|^2 > {R_{col}}_i^2, \quad i = 1, \dots, N \tag{7.1}$$

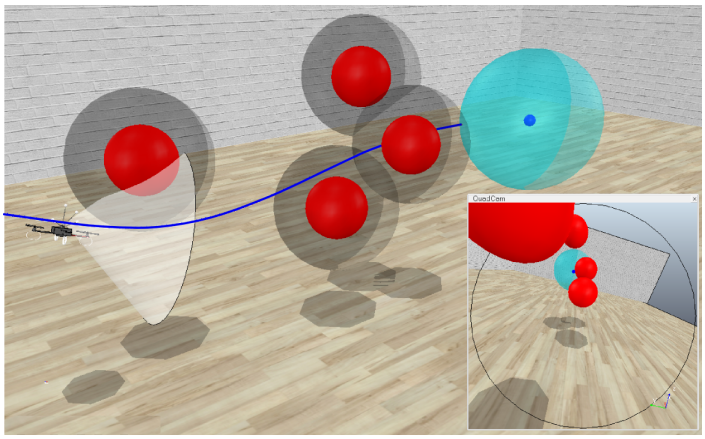


Figure 7.1 – An optimal trajectory for a quadrotor equipped with a camera is generated for reaching a minimum distance with a target (blue sphere) while avoiding collisions with spherical obstacles (inflated dark spheres) and occlusions of the target from the obstacles (red spheres).

The occlusion constraint can be modelled analogously to (7.1), but in the image plane, as follows

$$\|\boldsymbol{\beta} - \boldsymbol{\beta_{obs}}_i\|^2 > a_1^2, \quad i = 1, \dots, N \tag{7.2}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\beta_{obs}}$ are the image projections of the target and the obstacle center respectively. Using spherical projection, a sphere is projected as an circle in the image (see Fig. 7.2). Therefore we choose a_1 as the radius of this projected circle. For an obstacle of radius R_{occ} one has [195]

$$a_1 = \frac{R_{occ}}{\|\mathbf{O}\|} \tag{7.3}$$

where $\mathbf{O}$ is the vector of coordinates of $\boldsymbol{r_{obs}}$ in the camera frame.

Using perspective projection a sphere is projected as an ellipse in the image plane. In this case, we choose a_1 as the length of the semi-minor axis of the projected ellipse of the spherical obstacle of radius R_{occ}, see Fig. 7.3. Assuming knowledge

of the size of an obstacle in 3D with coordinates $\boldsymbol{r_{obs}} = (X_o, Y_o, Z_o)$ in the image plane one has [196]

$$a_1^2 = \frac{R_{occ}^2}{4(Z_o^2 - R_{occ}^2)} \tag{7.4}$$

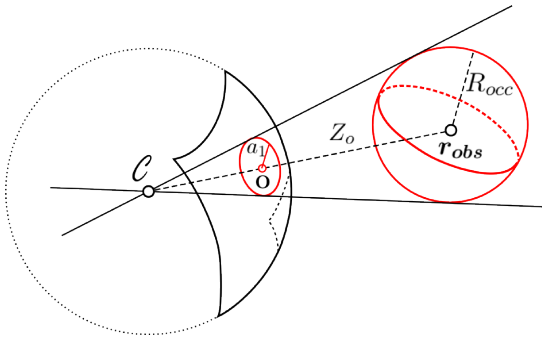


Figure 7.2 – The spherical projection of a sphere in the image of a camera at position $\mathcal{C}$ is a circle of radius a_1.

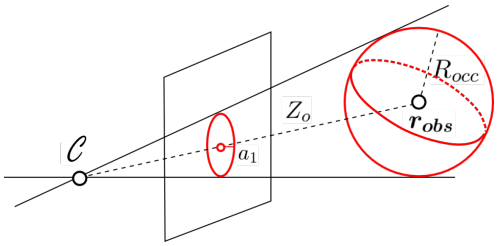


Figure 7.3 – The perspective projection of a sphere in the image plane of a camera at position $\mathcal{C}$ is an ellipse of semi-minor axis a_1.

With constraint (7.2) one seeks to prevent the target from *colliding* the projected obstacles (see Fig. 7.1) in the image space by keeping a minimum length a_1 that grows as the depth Z_o of the obstacle in the camera frame decreases. As the quadrotor moves towards the target the occlusion constraints from the obstacles passing behind the camera are of course discarded.

However, when dealing with occlusion avoidance with a quadrotor there exist configurations where strict avoidance is not feasible. Indeed, when the target goes exactly below an obstacle (for a down-looking camera, see Fig. 7.4) the quadrotor may not have sufficient actuation capability or sufficient space for avoiding any occlusions. These situations may occur for any camera orientation. Therefore, in order to avoid such critical situations and always provide a feasible solution, we introduce a *slack variable* λ within the occlusion constraint to authorize partial occlusion if necessary. Considering (7.4) we set

$$a_1^2 = \frac{(R_{occ} - \lambda)^2}{4(Z_o^2 - (R_{occ} - \lambda)^2)}, \quad s.t \quad 0 \leqslant \lambda \leqslant R_{occ} \tag{7.5}$$

The λ term plays the role of relaxing a hard constraint when the solver encounters not feasible situations. At the most critical configurations when the target is below an obstacle (Fig. 7.4), the collision constraint can be reduced or even cancelled by having λ reach the value R_{occ} which is the actual radius of the obstacle (lower than R_{col}). With equation (7.5) we observed very reactive responses from the system in case of occlusions. This may be due to the fact that small changes of λ induce a strong action on the occlusion constraint (7.2). Besides, by imposing a straightforward upper bound (R_{occ}) for λ, its action will take effect only in case of violations of constraint (7.2).

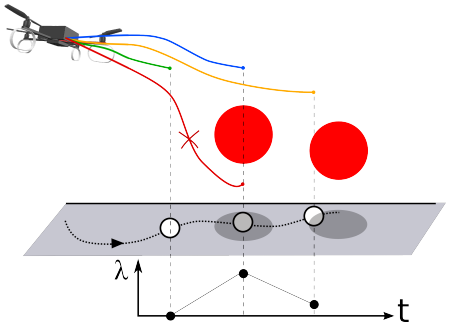


Figure 7.4 – In case of strong and sudden occlusions when the target is below an obstacle, the red trajectory cannot be a viable solution since it may violate actuation and/or spatial constraints. We instead allow minimal constraint violations to keep the solver efficient. The blue trajectory is then a relaxed solution where λ hits its limit value R_{occ} (obstacle radius). The orange trajectory represents the case of a less relaxed occlusion constraint where λ eventually reaches a smaller value and is zero when there are no occlusions (green trajectory)

It would make sense of course to use the ellipse semi-major axis defined as:

$$a_2^2 = \frac{R_{occ}^2(X_o^2 + Y_o^2 + Z_o^2 - R_{occ}^2)}{4(R_{occ}^2 - Z_o^2)^2} \tag{7.6}$$

This formulation would give more conservative occlusion avoidance constraints. However, in practice one has $X_o^2 + Y_o^2 \ll Z_o^2$ especially if one considers the limited camera field of view. This is why we consider the semi-minor axis (7.4) which also has the advantage of being less complex.

Of course, the occlusion constraint and the slack variable λ introduce conservatism to some extent. However, the main objective of λ is to improve stability and continuity of the solution in case of critical configurations.

7.3 Optimization problem definition

Finally, the current optimization problem related to the scenario shown in Fig. 7.1 can be stated as the following static NLP considering the cost function (6.8)

Problem 9 *Find* $\boldsymbol{P}, \lambda$ *such that:*

$$\min_{\boldsymbol{P},\lambda} \quad J + w_5\|\lambda\|^2 \tag{7.7a}$$

$$s.t. \quad \boldsymbol{\sigma}_\chi(t) = \boldsymbol{\sigma}_{\chi_t}, \tag{7.7b}$$

$$\boldsymbol{\sigma}^{(i)}(t+T) = \boldsymbol{\sigma}_{\chi^*}, i = 1, \dots, 3, \tag{7.7c}$$

$$\boldsymbol{u}(s) \in \mathcal{U}, \forall s \in [t, t+T], \tag{7.7d}$$

$$\boldsymbol{\beta}^T(t)e_c \geqslant \cos(\alpha/2), \forall s \in [t, t+T], \tag{7.7e}$$

$$\boldsymbol{\beta}^T(t)e_c \geqslant \cos(\gamma/2), \forall s \in [t+T_v, t+T] \tag{7.7f}$$

$$\|\boldsymbol{r}(s) - \boldsymbol{r_{obs}}\|^2 > R_{col}^2, \forall s \in [t, t+T], \tag{7.7g}$$

$$\|\boldsymbol{\beta}(s) - \boldsymbol{\beta}_{obs}\|^2 > a_1^2, \forall s \in [t, t+T], \tag{7.7h}$$

$$0 \leqslant \lambda \leqslant R_{occ} \tag{7.7i}$$

where $\boldsymbol{\sigma}_{\chi_t} = \phi_\chi^{-1}(\boldsymbol{\chi}_t)$ and $\boldsymbol{\sigma}_{\chi^*} = \phi_\chi^{-1}(\boldsymbol{\chi}^*)$. For a final hovering state $\boldsymbol{\chi}^*$ one has of course $\boldsymbol{\sigma}^{(i)}(t+T) = 0, i = 1, \dots, 3$. We choose not to explicitly constrain the final Cartesian position which is considered as a free parameter to be determined by the optimization algorithm. We seek feasible trajectories with constraint (7.7d) but, due to unknown target motion, the quadrotor might not have sufficient actuation to always satisfy the visibility constraint (7.7e).

The costs and constraints gradients in the flat space are computed analytically. However, we think it is more efficient to estimate (numerically) the gradients of the visual and inputs constraints instead of deriving their complex and heavy analytic formulation. To do so, we use complex-step differentiation [197]. It can be shown that the first-order derivative of a function $f \in \mathbb{R}$ can be approximated as

$$\frac{\partial f}{\partial x} = \frac{\Im(f(x + \boldsymbol{i}h))}{h} + \mathcal{O}(h^2), \quad h \in \mathbb{R} \tag{7.8}$$

where $\boldsymbol{i}$ is the complex number such that $\boldsymbol{i}^2 = -1$ and $\Im(z)$ denotes the imaginary part of a complex number z. This technique is attractive to determine first derivatives since it only requires a single evaluation of the function and avoids the problem of subtractive cancellation of classic finite approximation (round-off errors). Therefore, it is known to have superior accuracy (close the analytic accuracy) and numerical stability as analysed in [198]. It is also less intrusive in terms of program transformation than automatic differentiation which can sometimes require large and deep source overloads. On the other hand, complex differentiation requires some mathematical adaptations to be used with complex values (the square root or the absolute value functions for instance) which increases the computational cost. The choice between these two methods then hinges on a trade-off between ease of

implementation and execution efficiency and is further discussed in [199]. More details and comparison results are given in Appendix C.

7.4 A reactive re-planning framework with a down-looking camera

In this section, we express the re-planning strategy for tracking a mobile target on the ground (as shown in Fig. 7.4). We choose to consider a down-looking camera with a (more realistic) pyramid-shaped field of view as in our work [2] in order to also show that the planning strategy is able to exploit the corners of the field of view. The perspective projection of the target is again considered

$$\boldsymbol{\beta} = \frac{\boldsymbol{R}^T(\boldsymbol{r} - \boldsymbol{r})}{\boldsymbol{z_B}^T \boldsymbol{R}^T(\boldsymbol{r} - \boldsymbol{r})} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \in \mathbb{P}^2 \tag{7.9}$$

where $\mathbb{P}^2$ is the space of 3-D homogeneous vectors.

In this case, we adopt the penalization of the position error between the quadrotor and the target in the XY plane denoted as $\boldsymbol{e}_{xy} \in \mathbb{R}^2$. Moreover, we let the final height z_T free but bounded for limiting the motion along $\boldsymbol{e}_3$. This lighter formulation will basically compute trajectories for steering the quadrotor to a final hovering position such that the target appears at the camera center.

We define the following optimal problem

Problem 10 *Find* $\boldsymbol{P}, \lambda$ *such that:*

$$\min_{\boldsymbol{P},\lambda} \quad J_2 = w_1 \int_t^T \|\ddot{\boldsymbol{v}}(t)\|^2 dt + w_2 \int_t^T \|\ddot{\psi}(t)\|^2 dt \tag{7.10a}$$

$$+ w_3 \int_t^T \|\boldsymbol{e}_{xy}(t)\|^2 dt + w_4 z_T + w_5 \|\lambda\|^2 \tag{7.10b}$$

$$s.t. \quad \boldsymbol{\sigma}_\chi(t) = \boldsymbol{\sigma}_{\chi t}, \tag{7.10c}$$

$$\boldsymbol{\sigma}^{(i)}(t+T) = 0, i = 1, \ldots, 3, \tag{7.10d}$$

$$\boldsymbol{r}_{xy}(t+T) = \boldsymbol{r}^*_{xy}, \tag{7.10e}$$

$$\psi(t+T) = \psi^*, \quad \textit{(arbitrary value)} \tag{7.10f}$$

$$\boldsymbol{\beta}(s) \in \Omega, \forall s \in [t, t+T], \tag{7.10g}$$

$$\boldsymbol{u}(s) \in \mathcal{U}, \ \forall s \in [t, t+T], \tag{7.10h}$$

$$\|\boldsymbol{r}(s) - \boldsymbol{r_{obs}}\|^2 > R^2_{col}, \forall s \in [t, t+T], \tag{7.10i}$$

$$\|\boldsymbol{\beta}(s) - \boldsymbol{\beta}_{obs}\|^2 > a_1^2, \forall s \in [t, t+T], \tag{7.10j}$$

$$0 \leqslant \lambda \leqslant R_{occ}, \tag{7.10k}$$

$$Z_{min} \leqslant z_T \leqslant Z_{max} \tag{7.10l}$$

where $\boldsymbol{r}^*_{xy}$ is the target position in the XY plane at time t and the (square) image domain $\Omega = \{\boldsymbol{\beta} \in \mathbb{P}^2 \ s.t. \ \max(|\boldsymbol{\beta}^T \boldsymbol{x}_{\mathcal{B}}|, |\boldsymbol{\beta}^T \boldsymbol{y}_{\mathcal{B}}|) \leqslant \tan(\alpha)\}$.

It should be noted that the gradient of the occlusion avoidance constraint (7.10g) gives three possible *descent* directions for the decision variables for satisfying this constraint. More precisely, the SQP algorithm can: i) increase the distance between $\boldsymbol{\beta}(s)$ and $\boldsymbol{\beta}_{obs}$, ii) reduce the quantity a_1 by increasing the distance Z_o, iii) relax the constraint by increasing λ. In order to restrain the second direction that steers the solution towards the local minima of infinite height ($a_1 \to 0$) we put an upper limit on the quadrotor final height z_T with an additional constraint (7.10l). We also minimize z_T (linear in $\boldsymbol{P}$) to prevent the camera from staying at the maximum height Z_{max}. We also consider a minimum height Z_{min} for preventing the quadrotor from flying below the obstacles since our main focus is to show its ability to avoid occlusions. This is indeed a behaviour the quadrotor may exhibit. This formulation allows efficient re-planning at the rate of 30Hz. Simulation results are given in the next sections.

We also give more insight on the different system behaviours triggered by different tuning of the weights in the cost function (7.10b).

7.5 Simulation results

The presented approach was validated in a physically realistic environment. The inertial parameters of the quadrotor were slightly biased on purpose to introduce model uncertainties. The quadrotor dynamics were simulated using V-Rep at 150Hz. We exploit again the re-planning strategy described in Sect. 6.4.1 and the SQP optimization routine from NLOPT to compute the optimal solutions. The generated trajectories of the flat outputs were sent to TeleKyb which then computed the actual control inputs using controller [143].

The tracked target shown in Fig. 7.4 is manually commanded in velocity saturated at 1.0 m/s. We considered three spherical obstacles of radius $R_{occ} = 0.15m$ and of inflated radius $R_{col} = 0.4m$ (see Fig. 7.7). The video[1] attached to this work [3] shows the pertinence of having occlusion constraints in vision-based navigation since obstacles may occlude the target in many occasions if not considered at the planning stage. Finally, we show the efficiency of our re-planning framework in avoiding immediate and sudden occlusions. When critical situations are encountered the slack variable is able to keep a stable flight by allowing very brief occlusions.

[1] https://www.youtube.com/watch?v=mvvF1I72HM8

Table 7.1 shows some performance indexes from the on-line resolution of Problem 10 for a flight duration of about 90 seconds. One has respectively: the number of computed trajectories, the duration T_{max} for the stopping criterion, the planning horizon T, the range of SQP iterations and the mean iteration, the step and constraint tolerance and finally the percentage of total optimization instances that failed to return a solution within T_{max}. When it occurs the previous solution is returned.

loops	T_{max}	T	$iter_{range}$	$\overline{SQP_{iter}}$	tol	ratio
2734	33ms	3.5s	$[6, 63]$	31	10^{-4}	6%

Table 7.1 – Settings and performance of the solver for a 90 seconds flight

In the proposed formulations, the weights in the cost function are free parameters and need to be tuned empirically in order to generate the expected behaviour. Since the optimal problem is complex and highly nonlinear, any set of weights can generate a different response. However, from our experience the tuning triggers consistent responses and does not necessitate a particularly fine analysis. Generally, a higher w_1 will produce smoother trajectories and less aggressive motion. A higher w_3 will increase the convergence rate towards the target and a lower w_5 will tend to produce more relaxed occlusion constraints. Table 7.2 shows the numerical parameters used for the simulation. Moreover, we have used five different sets of weights (Table 7.3) an discuss their consistent effect on the generated trajectories shown in Fig. 7.7.

T	w_1	w_2	w_3	w_4	w_5
3.5s	$1e^{-5}$	$5e^{-3}$	$1e^{1}$	5	$5e^{3}$

Table 7.2 – Considered parameters for the simulation

Trajectory	w_1	w_2	w_3	w_4	w_5
orange	$1e^{-5}$	$5e^{-3}$	$5e^{1}$	5	$5e^{3}$
blue	$1e^{-4}$	$5e^{-3}$	$1e^{1}$	$1e^{-1}$	$5e^{3}$
green	$1e^{-3}$	$5e^{-3}$	$1e^{1}$	$1e^{1}$	$1e^{3}$
yellow	$1e^{-5}$	$5e^{-3}$	$1e^{1}$	1	$5e^{4}$
pink	$1e^{-3}$	$5e^{-3}$	$1e^{1}$	5	$1e^{4}$

Table 7.3 – Values of the weights used for generating the trajectories shown in Fig. 7.7

The hot-start method presented in our work [2] definitely helps the solver as showed in Table 7.4. Using the exact same conditions we compared the performance of the solver with:

- our hot-start algorithm (M1) in Sect. 6.4.1
- using the previous solution (M2)

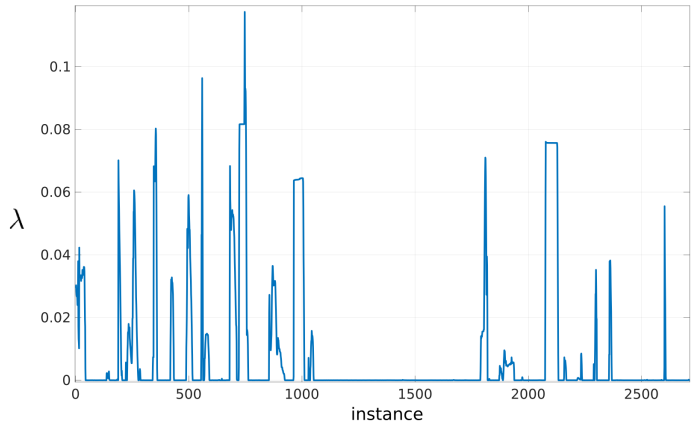


Figure 7.5 – Evolution of λ for the simulation. The peaks reveal the presence of sudden occlusions. The flat section (from around instance 1100 to instance 1700) respresents a flight period when the target is far from the obstacles. Therefore, λ is very close to 0. Here λ did not hit the upper limit $R_{occ} = 0.15m$.

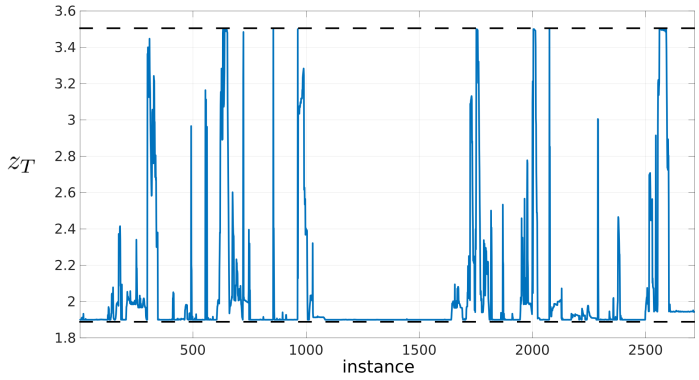


Figure 7.6 – Evolution of z_T for the simulation. The final height was bounded between 1.9m and 3.5m. Again, z_T reaches its minimum value when no collisions and occlusions occur.

- using the initial solution (a straight line from the initial to the final flat state) (M3)

method	mean number of SQP iterations	failure ratio
M1	25	2.8%
M2	33	20%
M3	43	84%

Table 7.4 – Convergence comparison between three different initial guess strategies. Failure ratio represents the percentage of total solutions that fail to converge within the current rate (1/30 ms).

Methods M2 and M3 clearly fail to meet the solver performance achieved using method M1. Moreover, M3 even quickly led to a failure of the task. The ratio can

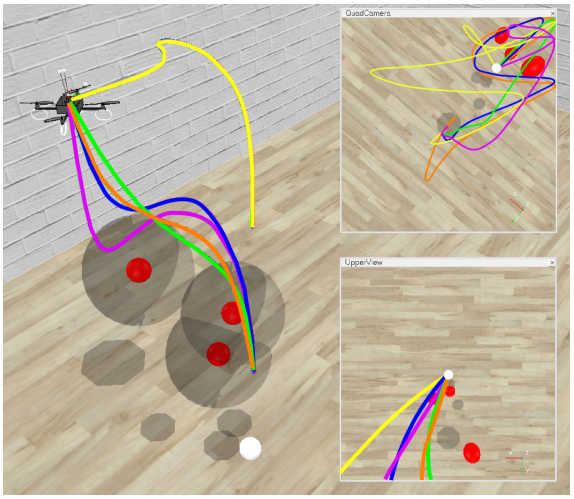


Figure 7.7 – Different solution trajectories returned with different sets of weights (in Table 7.3) in our simulation environment. The camera view is shown in the upper right corner with the image trajectories and a view from above the target is shown below. The orange trajectory uses the chosen values (in Table 7.2) with a higher w_3. One can observe a faster convergence towards the image center (with an overshoot). The blue one has a higher w_1 resulting in a smoother trajectory than the orange one. The green one is even smoother and is more compliant to occlusions (less jerky) since w_5 is lower. The yellow trajectory is very sharp (low w_1) because the occlusion constraints are more respected (high w_5) and the final height is less penalized (low w_4). The pink solution gives a different and smoother path (high w_1) that benefits less from the constraints relaxation (since w_5 is high).

quickly escalate. Indeed if the solver fails in returning a solution within 1/30 ms, its last (infeasible) iterate will still be used as initial guess to the next solver instance, probably generating an increasing depreciation of the initial guess.

7.6 Summary

In this work, we adapted our previous approach [2] on vision-based optimal trajectory generation to a wider context by considering reactive target tracking and both occlusions and collisions avoidance for either a front or a down looking camera (or any other camera/UAV configuration).

The quadrotor trajectories are mainly driven by vision while seeking aggressive but smooth trajectories that respect actuation and sensor limits for any camera orientations. Then, starting from a good initial guess the solver is able to return an optimal solution about 94% of the time within 1/30ms allowing an online replanning strategy capable of absorbing noise, disturbances and any non-modelled effect for long duration flights. The same strategy was applied during an experiment using a real quadrotor for the case of a front-looking camera.

In contrast to the literature, we proposed a method that explicitly handles visi-

bility constraints and occlusion avoidance within a fast online re-planning strategy. Besised, the occlusion constraint is expressed as a single constraint per object in contrast to [180]. Finally, we coped with the issue that the discontinuity of occlusion constraints can generate by introducing a minimal relaxation in Sect. 7.2.

The optimization problem in the current contribution differs in several points from the minimum-time Problem 6, especially in terms of costs and constraints. Although the re-planning strategy is the same, we emphasize the general online re-planning efficiency with different optimal problems. Moreover, the complex-step differentiation method plays a non negligible role in the framework. Indeed, optimality and stability of the re-planning framework has been improved by accelerating the gradient evaluation and improving its accuracy and numerical stability compared to our previous work [2].

To be successful, our path-planning approach requires a complete knowledge of the environment and robot model. These requirements can be limiting in many real applications. The need for such exact knowledge could be relaxed by accounting for modelling and calibration uncertainties at planning stage. Finally, future work includes validating the method with real (and maybe dynamic) obstacles and the use of vision only for estimating the target relative position. In this case the relaxation term will play an even more decisive role.

Chapter 8

Toward visual constraints relaxation: planning under intermittent measurements

Contents

8.1 Introduction

The role of navigation in robotics is to find a path moving a robot from its current state to a goal state. Practical experiments on path following show us that

paths cannot always be followed, because nothing is as perfect in reality as assumed during the planning (e.g., dead reckoning is not perfect in the real world). So there is a strong need to take into account the uncertainties during the planning phase. Indeed, model, sensors and environment uncertainties are inherent to many robotic applications and may lead to a failure of the task or impair the possibility to accurately follow a path if disregarded at the planning stage. For these reasons uncertainty-aware planning, also called *belief-space* planning, has received considerable attention in recent years. The concept of robust path planning can be tracked back to the mid 1990s. A class of control techniques that operate over the belief space, known as partially-observable Markov decision processes (POMDPs) [200] has been derived to address the above problem. Another class of works exploits local optimal control policies assuming a linear quadratic Gaussian (LQG) control strategy. However, these approaches suffer from the "curse of dimensionality", in particular POMDPs are notorious for their computational complexity that may prohibit their application for navigation in complex or uncertain environments in high dimensional state spaces. In [201] a more scalable LQG variant is proposed and applied to environments with discontinuous sensing regions. An approximate solution to POMDPs is given in [202] but with the use of considerable pre-processing. To deal with more complex objectives, deterministic planners such as RRT* and A* have become very popular since they benefit from asymptotic optimality and can explore the whole configuration space efficiently. In [203] a graph-search based on A* is proposed by discretizing the environment into cells for finding a safe route for a unicycle vehicle. Active visual perception with a quadrotor has been addressed in [185] for determining the path with minimal state uncertainty considering photometric information, and in [204] for maximizing visual coverage of a scene in presence of obstacles, localization and sensing uncertainty. Recently, [205] proposed an approximate POMDP control policy based on an initial guess trajectory returned by a RRT planner in a discretized environment.

In this work, we aim at planning a trajectory from an initial to a final state in presence of obstacles and input constraints for non-trivial robotic systems (like a quadrotor). We assume that the state is not available (especially the position) but on-board sensors (including a camera) are used to reconstruct the state with some estimation algorithm fusing position measurements in the world frame reconstructed from vision. Note that these measurements can be intermittent because of limited field of view, maximum range and so on. We want that the path guarantees some desired level of uncertainty in the reconstructed state despite the fact that measurements are not always available. More precisely, the goal state has to be reached with a bounded position uncertainty to guarantee some confidence level on the robot's location. Therefore, the system has to collect sufficient information from

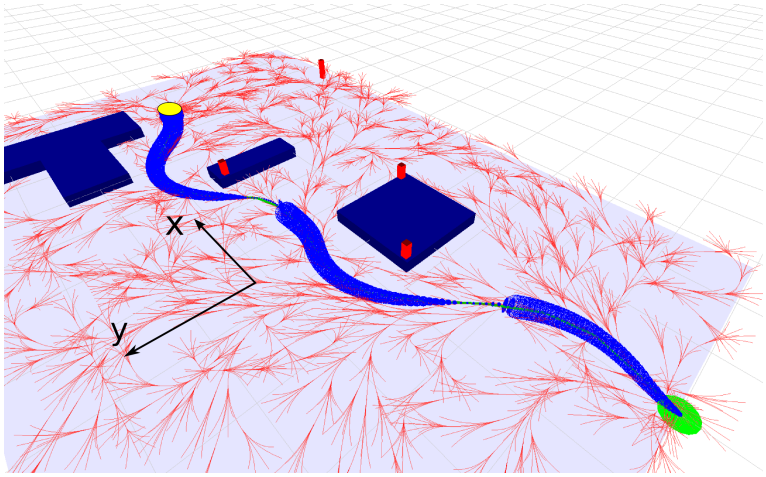


Figure 8.1 – Simulation environment for our framework. An optimal collision-free trajectory for the unicycle case connects the initial state (green dot) and a final state (yellow dot) in presence of obstacles (blue boxes). The pose uncertainty is represented by the blue ellipsoids whose size is reduced as soon as a landmark (red bars) is close enough to the robot and enters the field of view of the simulated camera attached to the robot. We assume the landmarks are not occluded by the obstacles. The propagated edges of the two graphs are rendered as the red curves.

visual landmarks sparsely placed in the environment to satisfy this final constraint (see Fig. 8.1 considering a unicycle equipped with a front-looking camera with reference to Fig. 8.2). Basically, the shape of the trajectory will vary depending on the level of uncertainty, i.e., process and measurement noises (e.g., see [206] with a unicycle) but also on the given initial and final states which is, to the best of our knowledge, not the case in the literature.

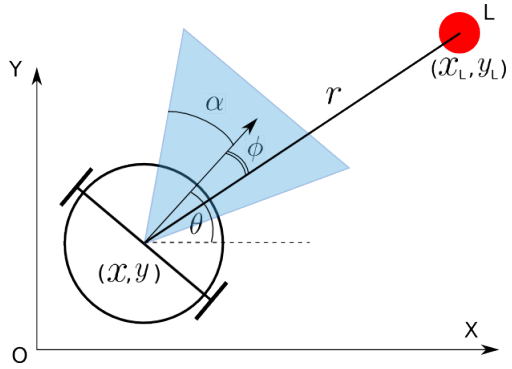


Figure 8.2 – A unicycle equipped with a fixed camera receives position measurements updates whenever a visual landmark (red dot) is close enough and enters the field of view.

Literature in perception-aware planning has generally focused on maximizing observability [185, 207, 208] (or minimizing the state uncertainty) based on some criteria e.g., the trace or the smallest eigenvalue of the covariance matrix. This strategy definitely helps in finding a path that tries to collect as much as information

as possible for preventing the state uncertainty to increase too much. However, the path itself may be severely suboptimal in terms of length and duration (e.g., [209] for optimal self-calibration of UAVs). Indeed, the path length is generally not constrained and can be excessively long, especially if the robot needs to pass by all the regions/beacons with richest information.

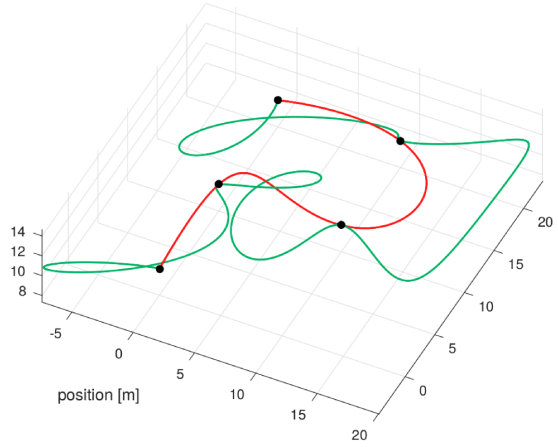


Figure 8.3 – A minimum snap (red) passing through waypoints and a trajectory (green) yielding well-observable states. This trajectory is much longer and much more complex, from [209].

In this work, we propose a minimum-time planning algorithm for dynamic systems returning feasible and robust trajectories that do not guarantee minimal state uncertainty along the trajectory but a bounded state uncertainty with given bounds at the goal, which we consider is a more practical application.

8.2 Contributions

This work focuses on finding robust paths for a robotic system by taking into account the state uncertainty and the probability of collision. We are interested in dealing with intermittent exteroceptive measurements (e.g., collected from vision). We assume these cues provide reliable measurements that will update a state estimation algorithm wherever they are available. The planner has to manage two tasks: reaching the goal in a minimum time and collecting sufficient measurements to reach the goal state with a given confidence level. We present a robust perception-aware bi-directional A* planner for *differentially flat* systems such as the unicycle and the quadrotor UAV and use a derivative-free Kalman filter to approximate the belief dynamics in the flat space. We also propose an efficient way of ensuring continuity and feasibility between the graphs by exploiting the *convex-hull* property of B-spline curves.

In a previous chapter (Sect. 7.6) we considered hard visibility constraints that may become too restrictive for minimum-time planning. In this chapter, we propose to relax these constraints by allowing intermittent visual cues losses to perform faster trajectories in larger and more complex environments. We implement a bi-directional A* algorithm that grows two graphs, one from the initial state and one from the final state (see Fig. 8.8). A solution trajectory is built by connecting the two graphs. This work blends the following features within graph-search algorithm: (i) incorporation of model and sensor uncertainty in collision avoidance and perception, (ii) generation of minimum-time and feasible trajectories for flat dynamic systems, (iii) incorporation of discontinuous visual measurements that are function of the robot's attitude, (iv) efficient graph connection using the convex hull property of B-spline curves.

Our work is mostly based on the recent work of [147] and [132] that propose an efficient A* planner in the flat space of a quadrotor which is applied to aggressive and precise collision avoidance that is function of the robot attitude in cluttered environments. In contrast to [132] we include perception constraints and state uncertainty and directly minimize the time. To the best of our knowledge, this is the first time minimum-time trajectories are generated in a graph-search planner while accounting for uncertainty in the visual perception which is affected by the system's attitude. to a unicycle for illustrating the approach, and then also to the case of a quadrotor for demonstrating the feasibility on a much more complex system while performing aggressive motions.

The rest of this chapter is organized as follows. Sect. 8.3 introduces differential flatness and the modelling of a quadrotor. Sect. 8.4 presents the uncertainty-aware planner formulated as a graph-search problem. How the graph is built is described in Sect. 8.5. The graphs rewiring is detailed in Sect. 8.6. In Sect. 8.7 simulation and experimental results are presented for a quadrotor with an onboard camera. Finally we draw some conclusions and future directions in Sect. 8.8.

8.3 Preliminaries

8.3.1 Differential flatness

As already said in Sect. 2.5 differentially flat systems are systems whose state $\boldsymbol{\chi}$ and inputs $\boldsymbol{u}$ can be expressed as algebraic functions of flat outputs derivatives up to some suitable order [210]. Let us remind the reader that differential flatness is often used in planning for the following reasons: i) the problem size is reduced, ii) any smooth enough curve in the flat space is feasible by the real system, iii) the system dynamics are linear in the flat space. The proposed algorithm is applicable

to systems represented as d independent chains of integrators of a given order r of the form

$$\boldsymbol{\eta}^{(r)} = \boldsymbol{\nu} \tag{8.1}$$

where $\boldsymbol{\nu} \in \mathbb{R}^d$ denotes the new inputs and $\boldsymbol{\eta} \in \mathbb{R}^d$ are the flat outputs. Let us define the system state in the flat space as

$$\boldsymbol{s} = \left(\boldsymbol{\eta}, ..., \boldsymbol{\eta}^{(r-1)}\right) \in \mathbb{R}^{d(r-1)} \tag{8.2}$$

In the rest of the chapter we consider the system position as the flat outputs, e.g., $\boldsymbol{\eta} = (x, y, z)$ in the three-dimensional space.

8.3.2 Application to a quadrotor UAV

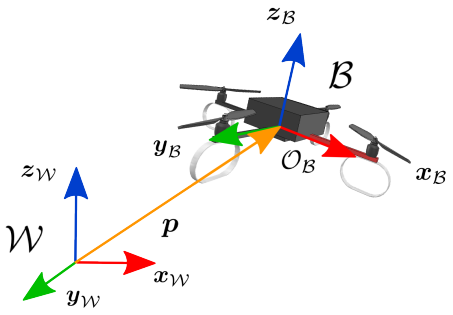


Figure 8.4 – Quadrotor model

With reference to Fig. 8.4 the quadrotor is known to be flat with flat outputs (x, y, z, ψ) [59]. The quadrotor dynamics can be decoupled into four linear subsystems of the form

$$x^{(4)} = u_1,\ y^{(4)} = u_2,\ z^{(4)} = u_3,\ \psi^{(2)} = u_4 \tag{8.3}$$

where $\boldsymbol{\nu} = (u_1, u_2, u_3, u_4)$ defines the new control inputs in the flat space. For the sake of simplicity, we do not plan over the yaw angle ψ that is assumed to be constant at zero. Moreover, we consider the quadrotor as three triple-integrators controlled in jerk along axes X, Y and Z (i.e., $\boldsymbol{\eta} = (x, y, z) \in \mathbb{R}^3$). With the above simplifications we seek to alleviate the planner whose complexity grows exponentially with the state dimension. Finally, we consider the state vector $\boldsymbol{s} = (\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}, \ddot{\boldsymbol{\eta}})$ and jerk inputs $\boldsymbol{\nu} = \boldsymbol{\eta}^{(3)}$. We consider that the quadrotor is equipped with a fixed downward-looking camera capable of providing reliable position measurements when fixed landmarks on the ground enter the limited field of view.

8.3.3 Application to a unicycle

With reference to Fig. 8.2, as usual, the kinematic model of a unicycle is

$$\begin{cases} \dot{x} = v\cos(\theta) & \text{(8.4a)} \\ \dot{y} = v\sin(\theta) & \text{(8.4b)} \\ \dot{\theta} = \omega & \text{(8.4c)} \end{cases}$$

where v and ω are the forward and angular velocities inputs of the robot respectively while (x, y) are the coordinates of the center of the rear axle and θ is the robot orientation in the world frame. It can be shown that the unicycle is flat with flat outputs [211]:

$$\boldsymbol{\eta} = (x, y) \in \mathbb{R}^2 \tag{8.5}$$

Namely, the system can be fully linearised and described by a double integrator with state $\boldsymbol{s} = (\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) \in \mathbb{R}^4$

$$\ddot{x} = u_1, \quad \ddot{y} = u_2 \tag{8.6}$$

where u_1 and u_2 are the new control inputs.

The real system inputs and angular position can be obtained from the flat outputs and their derivatives as follows

$$v = \sqrt{\dot{x}^2 + \dot{y}^2}, \quad \omega = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}, \quad \theta = \text{Arctan2}\left(\frac{\dot{x}}{\dot{y}}\right) \tag{8.7}$$

We will see in the next section why it is interesting to propagate the above dynamic models in the flat space instead of (8.4) (for the unicycle case) to extend the two A* graphs.

8.4 Problem formulation

We aim at solving an optimal control problem connecting an initial state $\boldsymbol{s}_{init}$ and a final state $\boldsymbol{s}_{goal}$ in a minimum time T. Let us define the following optimal control problem.

Problem 11 *Find the input $\boldsymbol{\nu}$ and time T such that:*

$$\min_{\boldsymbol{\nu}, T} \quad T \tag{8.8a}$$

$$s.t. \quad \boldsymbol{s}(0) = \boldsymbol{s}_{init}, \tag{8.8b}$$

$$\boldsymbol{s}(T) = \boldsymbol{s}_{goal}, \tag{8.8c}$$

$$max\{eig(\boldsymbol{P}^{\boldsymbol{\eta}}(T))\} \leqslant \bar{\lambda}, \tag{8.8d}$$

$$(\dot{\boldsymbol{\eta}}(\tau), \ddot{\boldsymbol{\eta}}(\tau), \boldsymbol{\eta}^{(3)}(\tau)) \in \mathcal{X}^{free} \quad \forall \tau \in [0, T] \tag{8.8e}$$

where $\text{eig}(\boldsymbol{P}^{\boldsymbol{\eta}}(T)) \in \mathbb{R}^d$ contains the eigenvalues of the position covariance matrix $\boldsymbol{P}^{\boldsymbol{\eta}}$ at the goal state $\boldsymbol{s}_{goal}$ and $\mathcal{X}^{free} := [-\bar{v}, \bar{v}]^3 \times [-\bar{a}, \bar{a}]^3 \times [-\bar{j}, \bar{j}]^3$ denotes the hypercube space of the admissible velocities, accelerations and jerks. The desired bound on the position uncertainty is defined by $\bar{\lambda} > 0$. For nonlinear systems such as a quadrotor, the Extended Kalman Filter (EKF) is often used for approximating the belief dynamics. The EKF is based on a linearization of the system dynamics which results in cumulative errors due to the local linearization assumption. In this paper, since we plan directly in the flat space we can use a so-called *derivative-free* Kalman filter without the need for derivatives and Jacobians calculations. Moreover, the state estimation accuracy of a derivative-free Kalman filter can be improved w.r.t. a standard EKF, especially for nonlinear systems [212]. Considering the linear equivalent system in the flat space one defines the process model. When a landmark is visible we have

$$\dot{\boldsymbol{s}} = A\boldsymbol{s} + B\boldsymbol{u} + \boldsymbol{\zeta}, \quad \boldsymbol{y} = C\boldsymbol{s} + \boldsymbol{\epsilon} \tag{8.9}$$

where $\boldsymbol{\zeta}$ is the process noise and $\boldsymbol{\epsilon}$ is the measurement noise. For a flat system controlled in acceleration (i.e., $r = 2$), assuming the velocity is estimated through filtering of position measurements, the matrices A, B, and C are given by

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{8.10}$$

In the next sections we will show how Problem 11 can be transformed from an infinite dimensional optimal control problem to a finite dimensional bi-directional graph-search problem. We choose to extend two graphs to improve the search, to increase the rate of convergence and the chance to find a solution especially in complex and cluttered environments. Moreover, the use of two graphs generally propagates fewer vertices than with a single graph [213].

8.4.1 Motion primitives

As in [147] we use polynomials to parameterize the flat state components and generate motion primitives to explore the flat space in a discrete way. More precisely, by applying a number of sampled constant inputs $\boldsymbol{\nu}_k \in [-\bar{j}, ..., \bar{j}]^3$ along each axis for a duration $\tau > 0$ one can iteratively build a graph $\mathcal{G}(V, E)$ rooted in state $\boldsymbol{s}_0$. Here, V defines the set of discrete states denoted as the *vertices* $\boldsymbol{s}$ in the graph representation that are connected with a motion primitive referred as an *edge* in the set E (e.g., see Fig. 8.1). For a flat system controlled in jerk (i.e,. $r = 3$) a

motion primitive represents the state $\boldsymbol{s}(t)$ starting at state $\boldsymbol{s}_0$ for $t \in [0, \tau]$ with a curve defined as

$$\boldsymbol{s}(t) = M(\boldsymbol{\nu}_k, \boldsymbol{s}_0, t) := \begin{bmatrix} \boldsymbol{\nu}_k \frac{t^3}{6} + \ddot{\boldsymbol{\eta}}_0 \frac{t^2}{2} + \dot{\boldsymbol{\eta}}_0 t + \boldsymbol{\eta}_0 \\ \boldsymbol{\nu}_k \frac{t^2}{2} + \ddot{\boldsymbol{\eta}}_0 t + \dot{\boldsymbol{\eta}}_0 \\ \boldsymbol{\nu}_k t + \ddot{\boldsymbol{\eta}}_0 \end{bmatrix} \tag{8.11}$$

These trajectories reflect the system dynamics thanks to differential flatness and provide the minimum jerk between the states $\boldsymbol{s}_0$ and $\boldsymbol{s}(\tau)$ [35]. The free flat space will be explored with a propagation of these motion primitives further detailed in Sect. 8.5. Naturally, changing the admissible bounds for $\boldsymbol{\nu}_k$ and duration τ will affect the free space coverage.

Problem 11 can be reformulated as Problem 12 in the graph representation where we seek the trajectory connecting the initial and goal states with the optimal control sequence $\boldsymbol{\nu}_k^*$ and the minimal number N^* of motion primitives.

Problem 12 *Find the sequence $\boldsymbol{\nu}_k$ and N such that:*

$$\min_{\boldsymbol{\nu}_k, N} \quad N \tag{8.12a}$$

$$s.t. \quad \boldsymbol{s}_0 = \boldsymbol{s}_{init}, \tag{8.12b}$$

$$\boldsymbol{s}_N = \boldsymbol{s}_{goal}, \tag{8.12c}$$

$$max\{eig(\boldsymbol{P}_N^{\boldsymbol{\eta}})\} \leqslant \bar{\lambda}, \tag{8.12d}$$

$$(\dot{\boldsymbol{\eta}}_k, \ddot{\boldsymbol{\eta}}_k, \boldsymbol{\eta}_k^{(3)}) \in \mathcal{X}^{free} \quad \forall k \in [\![0, N]\!] \tag{8.12e}$$

where $\boldsymbol{P}_N^{\boldsymbol{\eta}}$ is the covariance matrix on the position at the goal vertex. The resulting trajectory will have a total time $N^*\tau$. Finally, collisions are avoided by considering the robot shape as representative of the position uncertainty ellipse (or ellipsoid in 3D) whose estimation is detailed in the next section. Motion primitives that violate the collision constraints are not added to the graph.

The advantage of graph-search planners in contrast to optimization-based methods (and especially gradient-based) is that complex constraints are not directly part of the optimization problem but are checked at each vertex expansion. Moreover, optimization-based methods may not be adapted to problems involving discontinuous constraints gradients as for (8.12d) that is the solution of a stochastic process with intermittent Kalman updates and possibly large periods without any sensing information. Evaluating such a gradient for gradient-descent solvers would be challenging and computationally intense since it is also completely re-evaluated at each iteration. In the next section we show how state uncertainty is included in visual perception and in collision avoidance to guarantee perception of visual measurements and safe navigation to a given level of confidence.

8.4.2 State estimation uncertainty

Let σ be the major axis of the uncertainty ellipse $\boldsymbol{P}^{\boldsymbol{\eta}}$ at a given state. Then, for a 99% confidence level one has $\sigma_{99\%} = \sqrt{9.21}\sqrt{\lambda}$ (from the Chi-Square probabilities) where λ is the largest eigenvalue of $\boldsymbol{P}^{\boldsymbol{\eta}}$. This confidence ellipse defines the region that contains 99% of all samples that can be drawn from the Gaussian distribution. We take a circle (a sphere in 3D) with radius $\sigma_{99\%}$ as representative of the robot occupancy. It will vary with the pose uncertainty and will be incorporated in the planner for ensuring robust collision-free paths.

Now, let us include the position uncertainty in the visual measurements.

8.4.2.1 Unicycle case

A visual landmark at *known* position $\boldsymbol{\eta}_L = (x_L, y_L)$ in the world frame (see Fig. 8.2) is visible when it lies at a given range r from the camera at position $\boldsymbol{\eta} = (x, y)$ and when its bearing angle ϕ lies within $[-\alpha, \alpha]$. One has

$$\phi = \text{Arctan2}\left(\frac{y - y_L}{x - x_L}\right) - \theta, \tag{8.13a}$$

$$r = \|\boldsymbol{\eta} - \boldsymbol{\eta}_L\| \tag{8.13b}$$

The uncertainty $\Delta\phi \in \mathbb{R}$ related to the bearing angle ϕ can be obtained as a function of the state uncertainty and is given by

$$\Delta\phi = \begin{pmatrix} \frac{\partial\phi}{\partial\boldsymbol{\eta}} & \frac{\partial\phi}{\partial\dot{\boldsymbol{\eta}}} \end{pmatrix}^T \begin{pmatrix} P_{\boldsymbol{\eta}} & 0 \\ 0 & P_{\dot{\boldsymbol{\eta}}} \end{pmatrix} \begin{pmatrix} \frac{\partial\phi}{\partial\boldsymbol{\eta}} & \frac{\partial\phi}{\partial\dot{\boldsymbol{\eta}}} \end{pmatrix} \tag{8.14}$$

where $P_{\boldsymbol{\eta}}$ and $P_{\dot{\boldsymbol{\eta}}}$ denote the position covariance matrix and the linear velocity covariance matrix respectively. The covariance matrices of the flat states are evaluated with the Kalman filter along each valid discretized motion primitive. Applying the same process for the range condition (with lower and upper bounds $\underline{r}$ and $\bar{r}$), one can ensure a (theoretical) 99% confidence on the perception if the following upper bound conditions are satisfied at a given state

$$|\phi| + \sqrt{9.21}\Delta\phi < \alpha \tag{8.15a}$$

$$\underline{r} < r + \sqrt{9.21}\Delta r < \bar{r} \tag{8.15b}$$

8.4.2.2 Quadrotor case

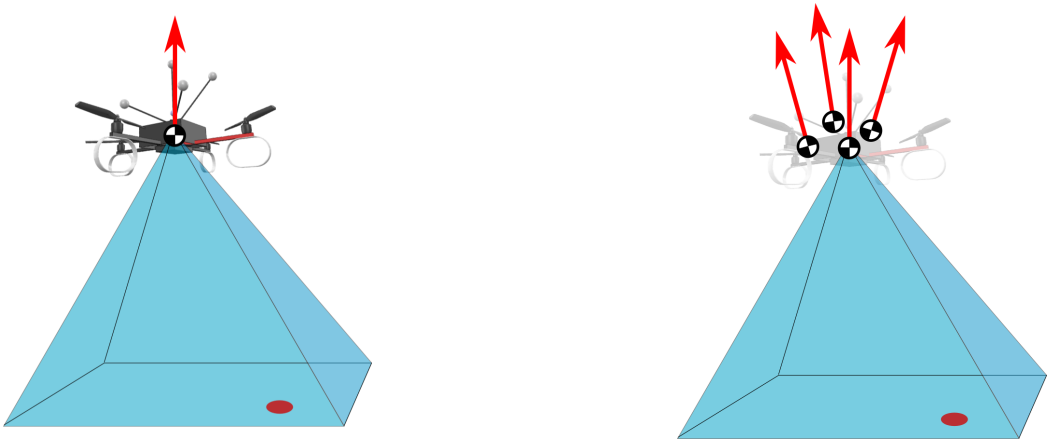
With reference to Fig. 8.6, in order to check that a landmark at *known* position $\boldsymbol{r}_L$ is visible at a given time instant, we check the following perception conditions on angles β_1 and β_2 on both X-Z and Y-Z planes. On a given plane one has

$$\beta_i = \arccos\left(\frac{(\boldsymbol{p} - \boldsymbol{r}_L).\boldsymbol{l}_i}{\|\boldsymbol{p} - \boldsymbol{r}_L\|}\right), \quad i = 1, 2 \tag{8.16}$$

where $\boldsymbol{p}$ is the camera position in the world frame and $\boldsymbol{l}_i$ are given by

$$\boldsymbol{l}_1 = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \boldsymbol{t},\ \boldsymbol{l}_2 = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \boldsymbol{t} \tag{8.17}$$

With $\boldsymbol{g}$ as the (constant) gravity acceleration and $\boldsymbol{a}$ as the robot acceleration in the world frame one has $\boldsymbol{t} = (\boldsymbol{a}+\boldsymbol{g})/\|\boldsymbol{a}+\boldsymbol{g}\|$. Again due to uncertainty on its state (and especially its pose) a landmark may not be perceived as expected at a given time instant (see Fig. 8.5 and Fig. 8.7). Therefore, to guarantee robust visual perception we extend conditions (8.16) with the uncertainty on the bearing angles β_i.



(a) Without state uncertainty the quadrotor state is known exactly, so are the landmarks position in the image.

(b) With state uncertainty the system state is not known exactly. This uncertainty is reflected in the image plane (see Fig. 8.7).

Figure 8.5 – Simulated quadrotor and visual perception of a landmark on the ground at a given time instant in the graph space.

Since the β_i are function of position and acceleration, the bearing uncertainty $\Delta\beta \in \mathbb{R}$ on a given plane can be computed as

$$\Delta\beta = \begin{pmatrix} \frac{\partial\beta}{\partial\boldsymbol{\eta}} & \frac{\partial\beta}{\partial\ddot{\boldsymbol{\eta}}} \end{pmatrix}^T \begin{pmatrix} \boldsymbol{P}^{\boldsymbol{\eta}} & 0 \\ 0 & \boldsymbol{P}^{\ddot{\boldsymbol{\eta}}} \end{pmatrix} \begin{pmatrix} \frac{\partial\beta}{\partial\boldsymbol{\eta}} & \frac{\partial\beta}{\partial\ddot{\boldsymbol{\eta}}} \end{pmatrix} \tag{8.18}$$

where $\boldsymbol{P}^{\ddot{\boldsymbol{\eta}}}$ is the acceleration covariance matrix. This way one can ensure a (theoretical) 99% confidence on the perception if the following upper bound conditions are satisfied on both planes X-Z and Y-Z at a given state

$$|\beta_i| + \sqrt{9.21}\Delta\beta < 2\alpha,\ i = 1, 2 \tag{8.19}$$

These conditions allow an exact and fast evaluation of the visibility and only rely on the flat state. Moreover, it allows us to consider a realistic pyramid-shaped field of view (since we do not plan over the yaw). The update step of the Kalman filter is

therefore applied with the simulated measurements whenever conditions (8.19) are met along the propagated motion primitives.

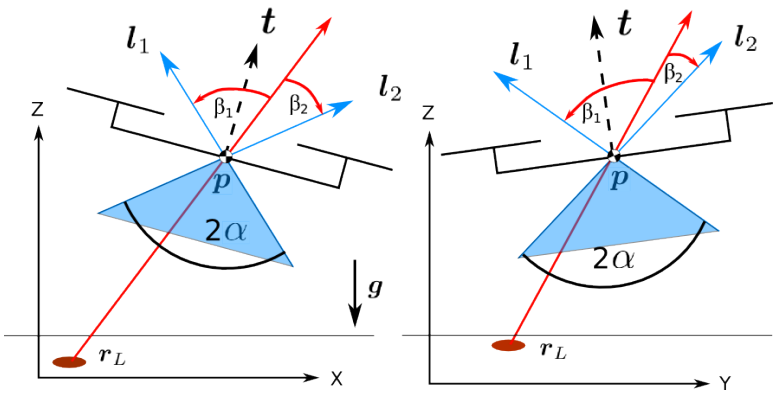
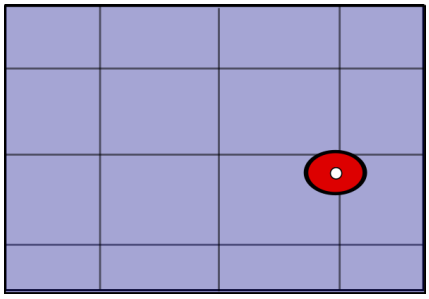


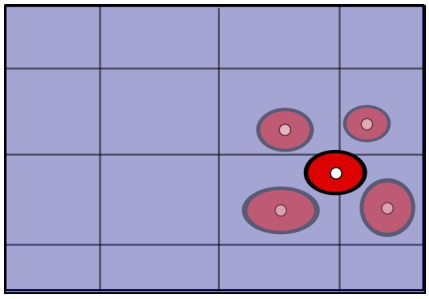
Figure 8.6 – The quadrotor in the vertical planes X-Z and Y-Z with $\psi = 0$. For simplicity let us assume that the robot COM $\boldsymbol{p}$ corresponds to the camera position. Having uncertainties on the state affects the visual perception. To evaluate if a landmark (red blob) is visible under the state uncertainty we check that condition (8.19) is satisfied on both planes.

8.5 Building the graph

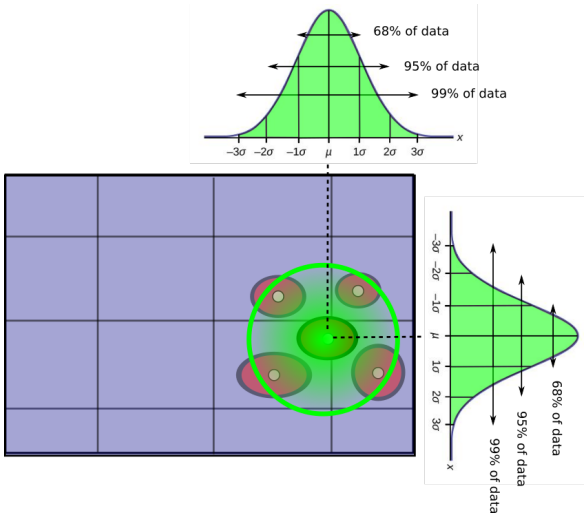
In this section we show how to exploit some vertices to efficiently explore the free space with the design of an heuristic function in order to build the graph. Traditionally, distance-based heuristics are used but they are not very relevant for dynamic or nonholonomic systems that cannot change their velocity, acceleration or orientation instantaneously. That is why a heuristic function more appropriate for second- or higher-order systems has been proposed in [147], by taking smoothness into account. As well know, A* algorithms rely on two functions: the heuristic function $h(\boldsymbol{s}, \boldsymbol{s}')$ that encodes an (optimistic) approximation of the *cost-to-go* from a vertex $\boldsymbol{s}$ to a goal vertex $\boldsymbol{s}'$ and the function $g(\boldsymbol{s})$ which represents the cost of vertex $\boldsymbol{s}$. Without a heuristic, A* is equivalent to a Dijkstra search, but encoding some theoretic information into the heuristic function can greatly reduce the number of expansions in favouring exploration toward promising directions/areas. We use the heuristic function proposed in [147] that originates from the resolution of Pontryagin's minimum principle and invite the reader to refer to the latter paper for more information. This function now encodes the "effort" required to connect two states given the considered control input (e.g., velocity, acceleration or jerk) and is used to select vertices leading to the exploration toward regions with minimal energy in order to encourage smooth trajectories. The cost of an edge itself is $g(\boldsymbol{s}) = \tau$ because we want to minimize the time. We propose a bi-directional A* algorithm that builds two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$. $\mathcal{G}_1$ starts at the initial state $\boldsymbol{s}_0$ and $\mathcal{G}_2$ starts



(a) A landmark on the ground perceived in the image plane without state uncertainty.



(b) Possible landmark positions in the image plane due to state uncertainty on a quadrotor.



(c) If the green 3-sigma ellipse is fully contained in the field of view, the landmark is guaranteed at 99% to be visible at a given time instant (in theory).

Figure 8.7 – Effect of state uncertainty on the visual perception for the quadrotor case. If the robot is asked to follow a given path, due to state uncertainty a visual landmark may not be perceived as expected (Fig. 8.7b) but will lie in a given probabilistic region (Fig. 8.7c) whose size is defined by a given confidence level which is related to the Gaussian distribution.

at the final state $\boldsymbol{s}_N$. Both graphs will be propagated and connected to return full trajectories from $\boldsymbol{s}_{init}$ to $\boldsymbol{s}_{goal}$.

Designing an efficient space exploration is tedious when complex tasks are involved and should not rely on a too strong *a priori*. Namely, in our case, the search should not be biased towards the goal since it may not collect sufficient visual cues from the landmarks to satisfy (8.12d). When multiple goals are present one may bias the search towards these goals, here, the landmarks. However, ensuring convergence to the final goal is not straightforward, especially for dynamic systems with perception goals. In the end, we choose to not rely on any exploration a priori to be able to deal with any environment and landmark configurations (provided a solution exists). We will rely instead on random-based exploration by smoothly propagating vertices toward states sampled randomly in the free space. Algorithm 1 runs for a given number of iterations I and is detailed below. Note that the uncertainty is only propagated on graph $\mathcal{G}_1$ with the Kalman filter since graph $\mathcal{G}_2$ is grown *backwards* (i.e., from the final goal $\boldsymbol{s}_{goal}$ towards the initial state $\boldsymbol{s}_{init}$). With reference to Fig. 8.9, the algorithm procedures are detailed below:

Sample: returns an independent and uniformly distributed random sample vertex $\boldsymbol{s}_{rand}$ in the free space.

NearVertices: given a sample vertex $\boldsymbol{s}_{rand}$, a graph $\mathcal{G} = (V, E)$ and a ball region $\mathfrak{B}_r$ of a given radius ρ, the set of near vertices is defined as $Near(\boldsymbol{s}, \mathcal{G}, \rho) = \{\boldsymbol{s} \in V : \mathrm{d}(\boldsymbol{s}, \boldsymbol{s}_{rand}) \leqslant \rho\}$ where d is the Euclidean distance and $\rho = \gamma(\log(K)/K)^{1/q}$ is the radius for expansion with K is the number of vertices and q is the space dimension. The ball radius helps capturing vertices when the graph is hollow and shrinks with the number of vertices to reduce the computation time. We use a constant radius γ_c for finding connections candidates (see procedure *ConnectG*).

GetSortedList: given a list of vertices V and a goal $\boldsymbol{s}'$, this function returns a list L_s of the sorted vertices $\boldsymbol{s} \in V$ in increasing heuristic cost $h(\boldsymbol{s}, \boldsymbol{s}')$.

ChooseBestParent: the vertex with lowest h cost from a list of vertices is chosen for expansion. We seek to find the parent vertex that will expand vertices towards the given goal with the lowest energy (highest smoothness).

BestVertices: when no near vertices are found in $\mathfrak{B}_r$, this function finds the vertex $\boldsymbol{s}$ in graph $\mathcal{G}_1$ with lowest cost $h(\boldsymbol{s}, \boldsymbol{s}_N)$ and analogously for $\mathcal{G}_2$ with $h(\boldsymbol{s}, \boldsymbol{s}_0)$.

ExtendVertex: propagates a motion primitive from a given parent vertex. This function includes the belief state propagation with the Kalman filter and collision and feasibility tests.

InsertVertices: valid vertices/edges are added to the graph and marked as *children* from their parent vertex.

InsertVertex: this function inserts a single vertex/edge pair.

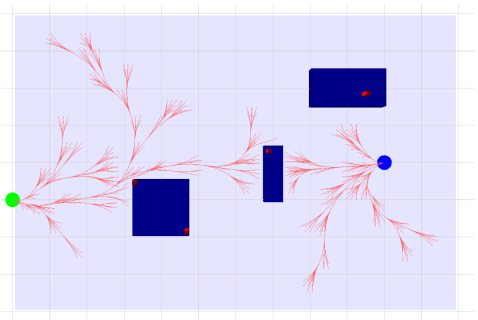
Algorithm 1 Bi-A*

$g(\boldsymbol{s}^1_{best}) \leftarrow \infty,\ g(\boldsymbol{s}^2_{best}) \leftarrow \infty$
for $i \leftarrow 1$ **to** I **do**
  $\boldsymbol{s}_{rand} \leftarrow Sample()$
  $(X^1_{near}, X^2_{near}) \leftarrow NearVertices(\boldsymbol{s}_{rand}, \mathcal{G}_1, \mathcal{G}_2, \rho)$
  $(X^1_c, X^2_c) \leftarrow NearVertices(\boldsymbol{s}_{rand}, \mathcal{G}_1, \mathcal{G}_2, \gamma_c)$
  if $X^1_c \neq$ **and** $X^2_c \neq$ **then**
    $(\boldsymbol{s}^1_{new}, \boldsymbol{s}^2_{new}) \leftarrow ConnectG(X^1_c, X^2_c)$
    $(\boldsymbol{P}^{\boldsymbol{\eta}}_N, coll) \leftarrow BackProp(\boldsymbol{s}^2_{new}, P_{\boldsymbol{s}^1_{new}}, \mathcal{G}_2)$
    if $\max\{\text{eig}(\boldsymbol{P}^{\boldsymbol{\eta}}_N)\} \leqslant \bar{\lambda}$ **and** $!coll$
      if $g(\boldsymbol{s}^1_{new}) + g(\boldsymbol{s}^2_{new}) < g(\boldsymbol{s}^1_{best}) + g(\boldsymbol{s}^2_{best})$
        $\boldsymbol{s}^1_{best} = \boldsymbol{s}^1_{new},\ \boldsymbol{s}^2_{best} = \boldsymbol{s}^2_{new}$
  end if
  if $X^1_{near} \neq$ **then**
    $L_s \leftarrow GetSortedList(X^1_{near})$
    $\boldsymbol{s}^*_1 \leftarrow ChooseBestParent(L_s, \boldsymbol{s}_N)$
    $L^c_1 \leftarrow ExtendVertex(\boldsymbol{s}^*_1)$
    $G_1 \leftarrow InsertVertices(L^c_1)$
  end if
  if $X^2_{near} \neq$ **then**
    $L_s \leftarrow GetSortedList(X^2_{near})$
    $\boldsymbol{s}^*_2 \leftarrow ChooseBestParent(L_s, \boldsymbol{s}_0)$
    $L^c_2 \leftarrow ExtendVertex(\boldsymbol{s}^*_2)$
    $G_2 \leftarrow InsertVertices(L^c_2)$
  end if
  if $X^1_{near} =$ **and** $X^2_{near} =$ **then**
    $(\boldsymbol{s}^*_1, \boldsymbol{s}^*_2) \leftarrow BestVertices(\mathcal{G}_1, \mathcal{G}_2)$
    $L^c_1 \leftarrow ExtendVertex(\boldsymbol{s}^*_1)$
    $G_1 \leftarrow InsertVertices(L^c_1)$
    $L^c_2 \leftarrow ExtendVertex(\boldsymbol{s}^*_2)$
    $G_2 \leftarrow InsertVertices(L^c_2)$
  end if
end forreturn $\boldsymbol{s}^1_{best}, \boldsymbol{s}^2_{best}$

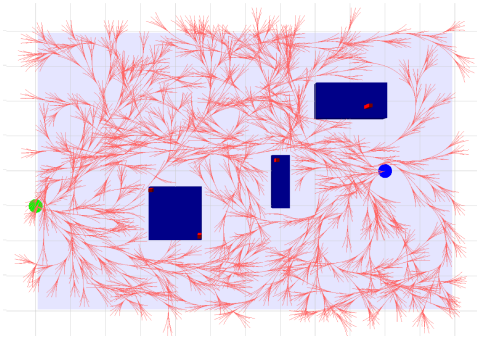
ConnectG: this procedure is triggered whenever vertices from both graphs are found in the procedure *NearVertices* within a second ball region of constant radius γ_c centered on $\boldsymbol{s}_{rand}$. Indeed, we seek pairs of vertices in a vicinity region to perform connection tests (see Algorithm 2) using the function *solveQP* presented in Sect. 8.6. Note that γ_c can for instance, be chosen as the "spatial resolution" of motion primitives or larger to find more connections.

BackProp: given a vertex $\boldsymbol{s}_0$ with a covariance matrix $\boldsymbol{P}_0$ from graph $\mathcal{G}_1$, once a connection is found we *back-propagate* the state uncertainty through $\mathcal{G}_2$ by considering the sampled states between $\boldsymbol{s}_0$ and the goal $\boldsymbol{s}_N$ (see Fig. 8.9).

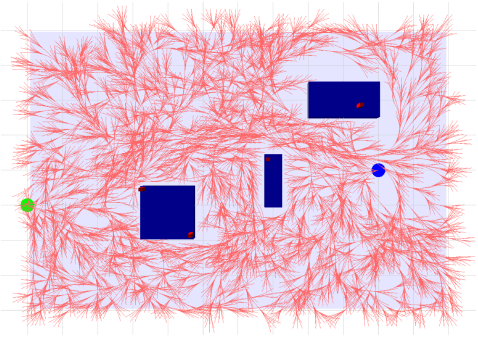
Algorithm 1 aims at finding the most direct trajectory towards the goal, especially in case of low process noise and tries to mimic a couple of nice properties of classic graph-search planners, namely: i) expansion towards unexplored regions; ii) probabilistic *completeness* due to a uniform *random walk*; iii) asymptotic optimality.



(a) $I = 100$, $\mathcal{G}_1$: 271 vertices. $\mathcal{G}_2$: 187 vertices



(b) $I = 100$, $\mathcal{G}_1$: 2427 vertices. $\mathcal{G}_2$: 2302 vertices



(c) $I = 100$, $\mathcal{G}_1$: 5774 vertices. $\mathcal{G}_2$: 4081 vertices

Figure 8.8 – Planner performance in exploring a 2D environment. When the graphs are hollow and no near vertices are found, both graphs propagate vertices toward each other (Fig. 8.8a). This helps finding a direct path that may be the optimal solution in case of low process noise. $\mathcal{G}_2$ extends fewer vertices in this environment because of the obstacles that are slightly cutting down its expansion. One can see that the free space is explored in a reasonable uniform way.

Algorithm 2 ConnectG

Input: X_c^1, X_c^2

```
success = false
for s_1 in X_c^1 do
    for s_2 in X_c^2 do
        if h(s_1, s_2) < h̄ then
            success ← solveQP(s_1, s_2)
            if success = true then
                return (s_1, s_2)
            end if
        end if
    end for
end for
return 0
```

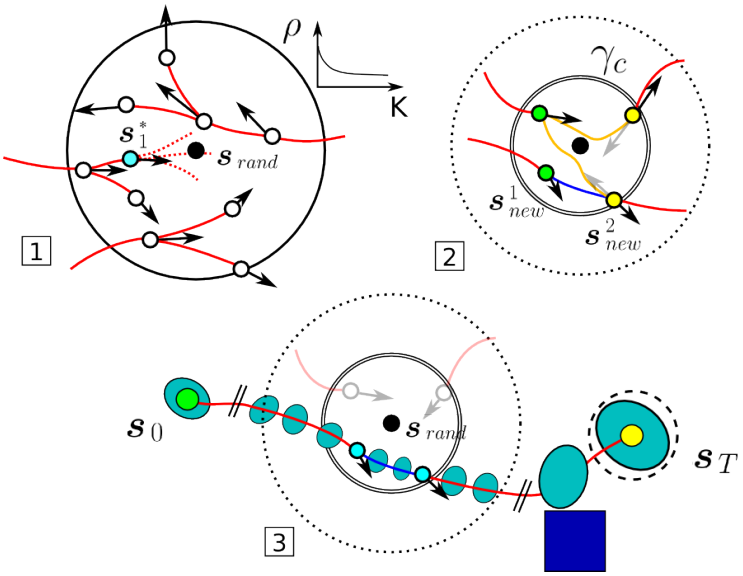


Figure 8.9 – 2D case: Inside a ball of radius ρ centered at $\boldsymbol{s}_{rand}$ (black dot), picture 1 shows how the vertex with lowest cost h is chosen for expansion (in cyan). The black arrows represent the vertex velocity vector. If vertices from graph $\mathcal{G}_1$ (in green) and from graph $\mathcal{G}_2$ (in yellow) are found inside a ball region of fixed radius γ_c, connections trials are performed except for connections with a high h cost (orange lines). Note that we consider the opposite velocity (and higher derivatives) vectors for vertices coming from graph $\mathcal{G}_2$. If a candidate connection is found (blue line) the uncertainty is propagated along $\mathcal{G}_2$ starting from $\boldsymbol{s}^1_{new}$ (picture 3). If no collisions are found between the obstacles (blue box) and the uncertain robot occupancy (turquoise ellipses) and if the final constraint (8.12d) is satisfied on $\boldsymbol{P}_N^{\boldsymbol{\eta}}$, a solution trajectory is reconstructed from the initial vertex $\boldsymbol{s}_0$ (green dot) to the goal vertex $\boldsymbol{s}_N$ (yellow dot) and its total cost is evaluated.

Algorithm 2 performs connection trials on the vertices in X_c^1, X_c^2 if their heuristic cost is lower than a given value $\bar{h}$. This value can be chosen off-line to skip connections that may require "too much" energy (see Fig. 8.10). Usually, graph-

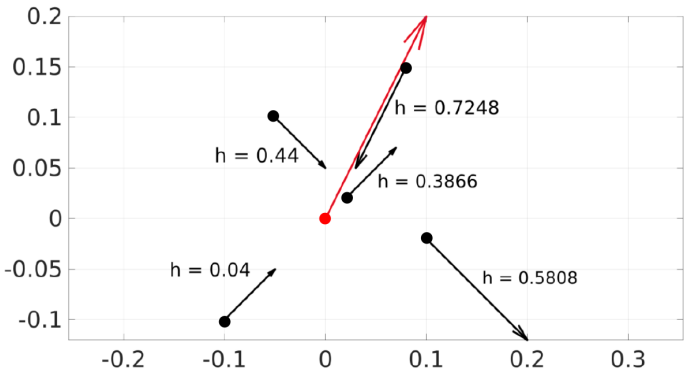


Figure 8.10 – Computation of the heuristic cost $h(\boldsymbol{s}_1, \boldsymbol{s}_2)$ between multiple vertices $\boldsymbol{s}_1$ and a goal vertex $\boldsymbol{s}_2$ at the origin. The arrows represent the vertices linear velocity. As we can see the vertex with the lowest cost is the one whose velocity vector is the most "aligned" with the goal velocity vector. Vertices that are closer to the goal have a higher cost.

search planners for dynamic systems involve two steps. First, an optimal path is found ignoring the system dynamics. Then a refining step is performed by optimizing over a selection of state keyframes along the path. The resulting trajectory is smoothed and more adapted to dynamic systems (see e.g., [147], [33]). However, the shape of this trajectory may strongly differ from the original path (e.g., in [33]). In our context, visual perception cannot be guaranteed with such a technique and it does not take into account the uncertainty in collision avoidance. A key role of the bi-directional planner used in this work is that if a connection is made, the initial and final states are exactly connected, which is generally not the case in the graph-search planners literature. For instance, [147] stops the search when a vertex becomes close enough to the goal state $\boldsymbol{s}_{goal}$, a condition that may not be met if not properly tuned. In this work we aim instead at finding the optimal trajectory that will be directly tracked by the real system without additional refining steps.

Next section details how the connection between the two graphs is performed in an optimal and efficient way.

8.6 Connecting the graphs

Connecting the two graphs is a critical step. One has to ensure state continuity between two candidate vertices $\boldsymbol{s}_1 \in V_1$ and $\boldsymbol{s}_2 \in V_2$ in a given time T_c. This problem is known as the Boundary Value Problem (BVP). Moreover, one wants to ensure feasibility as well and connections have to be evaluated quickly since the

process may be called many times. We propose an optimal formulation to the BVP that can be solved as a single convex quadratic program. We exploit the *convex-hull* property of B-splines in order to impose constraints directly on the curve control points to alleviate the solver.

8.6.1 Solving the constrained BVP

The problem we want to solve is the following

Problem 13 *Find* $\boldsymbol{s}, \boldsymbol{\nu}$ *such that:*

$$\min_{\boldsymbol{s},\boldsymbol{\nu}} \quad \int_0^{T_c} \|\boldsymbol{\nu}(\tau)\|^2 d\tau \tag{8.20a}$$

$$s.t. \quad \boldsymbol{s}(0) = \boldsymbol{s}_1, \tag{8.20b}$$

$$\boldsymbol{s}(T_c) = \boldsymbol{s}_2, \tag{8.20c}$$

$$(\dot{\boldsymbol{\eta}}(\tau), ..., \boldsymbol{\eta}^{(r)}(\tau)) \in \mathcal{X}^{free} \quad \forall \tau \in [0, T_c] \tag{8.20d}$$

We penalize the input norm to obtain a smooth connection trajectory. For the quadrotor one minimizes the jerk norm (i.e., $r = 3$). Now, we parameterize the flat state $\boldsymbol{s}$ as B-splines to turn the infinite dimensional problem to a finite one with a limited number of coefficients that can be solved numerically. A trajectory s is parameterized as

$$s(t) = \sum_{i=0}^{i=n} B_{i,p}(\tau)\boldsymbol{P}, \quad \forall \tau \in [0, T] \tag{8.21}$$

where B_i is a polynomial basis of degree p (of order $k = p + 1$) and $\boldsymbol{P} \in \mathbb{R}^{n+1}$ represents the set of coefficients.

Considering constraints on the real system inputs would require a nonlinear solver. We choose to simplify the problem for the unicycle case by replacing these constraints with bounds on the linear velocity $\dot{\boldsymbol{\eta}}$ and acceleration $\ddot{\boldsymbol{\eta}}$.

8.6.2 A linear quadratic program based on B-splines

The reason we use B-splines is for their *convex hull* property that will allow us to write linear inequality constraints directly function of the B-spline control points. A similar approach has been used in [214] for manipulators.

These techniques are attractive and mostly adopted because of the convex hull property of the B-spline parametrization, which states that a spline is always contained in the convex hull of its B-spline coefficients [126]. This way, spline constraints can be relaxed to constraints on the B-spline coefficients and we are ensured that the B-spline curve will satisfy the same constraints. Replacing semi-infinite sets of constraints by finite, yet conservative sets is called a B-spline relaxation. B-spline

relaxations can only be applied on splines, meaning that all constraints must be written as derivatives, antiderivatives, or polynomials of splines. Therefore, a nonlinear change of variables needs to be adopted to transform all constraints into bounds on spline functions [215] The major advantage is that B-spline relaxations avoid time gridding of the constraints, while they guarantee constraint satisfaction at all times. The disadvantage is that B-spline relaxations also introduce some conservatism. This conservatism can be reduced by choosing a higher dimensional basis, at the cost of introducing extra constraints [214]. This conservatism stems from the distance between the control polygon and the spline itself. Knot insertion is generally the preferred technique, since, in this way, the conservatism reduces quadratically with the number of constraints, while this decrease is only linear when using degree elevation [216]. In addition, knot insertion allows refining the control polygon only locally, whereas degree elevation always affects the entire control polygon. Since using a higher dimensional basis translates into more constraints, it is necessary to make a trade-off between conservatism and computational complexity (number of constraints).

For nonlinear systems, one can resort to convex approximations of the feasible set [217, 140]. Inevitably, this method introduces conservatism in the problem. Moreover, some feasible sets do not admit such a polytopic approximation, e.g. obstacle avoidance constraints.

Constraints (8.20d) can be mapped in the space of the control points. Let us differentiate the B-spline of degree p defined on the *clamped* knot vector of size $n+k+1$ such that $u_i \leqslant u_{i+1}, i = 0, \dots, n-k$

$$U = (\underbrace{0, \dots, 0}_{p+1}, u_{p+1}, \dots, u_{k-p-1}, \underbrace{1, \dots, 1}_{p+1}) \tag{8.22}$$

The first derivative can be expressed as a function of the control points $\boldsymbol{P}$ with

$$s^{'}(u) = p \sum_{i=0}^{i=n-1} B_{i+1,p-1}(t) \frac{\boldsymbol{P}_{i+1} - \boldsymbol{P}_i}{T(u_{i+p+1} - u_{i+1})} \tag{8.23}$$

Let us define the vector of new coefficients

$$\boldsymbol{Q}_i = p \frac{\boldsymbol{P}_{i+1} - \boldsymbol{P}_i}{T(u_{i+p+1} - u_{i+1})}, \quad \forall i \in [\![0, n-1]\!] \tag{8.24}$$

For the second derivative one has

$$s^{''}(u) = \sum_{i=0}^{i=n-2} B_{i+2,p-2}(t) \boldsymbol{R}_i \tag{8.25}$$

where $\boldsymbol{R}_i$ are the control points of the second derivative. One has

$$\boldsymbol{R}_i = (p-1) \frac{\boldsymbol{Q}_{i+1} - \boldsymbol{Q}_i}{T(u_{i+p+1} - u_{i+2})}, \quad \forall i \in [\![0, n-2]\!] \tag{8.26}$$

Now we can express $\boldsymbol{Q}$ and $\boldsymbol{R}$ as functions of $\boldsymbol{P}$ with

$$\boldsymbol{Q} = \boldsymbol{A_Q}\boldsymbol{P}, \quad \boldsymbol{R} = \boldsymbol{A_R}\boldsymbol{Q} = \boldsymbol{A_R}\boldsymbol{A_Q}\boldsymbol{P} \tag{8.27}$$

where matrix $\boldsymbol{A_Q} \in \mathbb{R}^{n\times(n+1)}$ and $\boldsymbol{A_R} \in \mathbb{R}^{(n-1)\times n}$. Similarly, control points of the third derivative are given by

$$\boldsymbol{S}_i = (p-2)\frac{\boldsymbol{R}_{i+1} - \boldsymbol{R}_i}{T(u_{i+p+1} - u_{i+3})}, \quad \forall i \in [\![0, n-3]\!] \tag{8.28}$$

Now, one can easily set semi-infinite bounds on the derivatives coefficients $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{S}$ that are linear in $\boldsymbol{P}$. All the constraints can be rewritten as functions of the control points $\boldsymbol{P}$. One wants to solve the following problem on each axis.

Problem 14 *Find $\boldsymbol{P}$ such that:*

$$\min_{\boldsymbol{P}} \boldsymbol{P}^T(\boldsymbol{B}_r^T\boldsymbol{B}_r)\boldsymbol{P} \tag{8.29a}$$

$$s.t. \left\langle \boldsymbol{P}, B_j(0)\right\rangle = \boldsymbol{\eta}_1^{(i)}, \ \forall (i,j) \in [\![0, r-1]\!]^2 \tag{8.29b}$$

$$\left\langle \boldsymbol{P}, B_j(T_c)\right\rangle = \boldsymbol{\eta}_2^{(i)}, \ \forall (i,j) \in [\![0, r-1]\!]^2 \tag{8.29c}$$

$$-\bar{v} \leqslant \boldsymbol{Q}_i \leqslant \bar{v}, \quad \forall i \in [\![0, n-1]\!] \tag{8.29d}$$

$$-\bar{a} \leqslant \boldsymbol{R}_i \leqslant \bar{a}, \quad \forall i \in [\![0, n-2]\!] \tag{8.29e}$$

$$-\bar{j} \leqslant \boldsymbol{S}_i \leqslant \bar{j} \quad \forall i \in [\![0, n-3]\!] \tag{8.29f}$$

where $\boldsymbol{B}_r$ is the r-th derivative of the B-spline basis function. We recall that $r = 3$ for the case of a quadrotor and we minimize the jerk.

The problem can be written in the compact form of a quadratic program with

Problem 15 *Find $\boldsymbol{P}$ such that:*

$$\min_{\boldsymbol{P}} \quad \frac{1}{2}\boldsymbol{P}^T H \boldsymbol{P} + f^T\boldsymbol{P} \tag{8.30a}$$

$$s.t. \quad A\boldsymbol{P} \leqslant b, \tag{8.30b}$$

$$A_{eq}\boldsymbol{P} = b_{eq} \tag{8.30c}$$

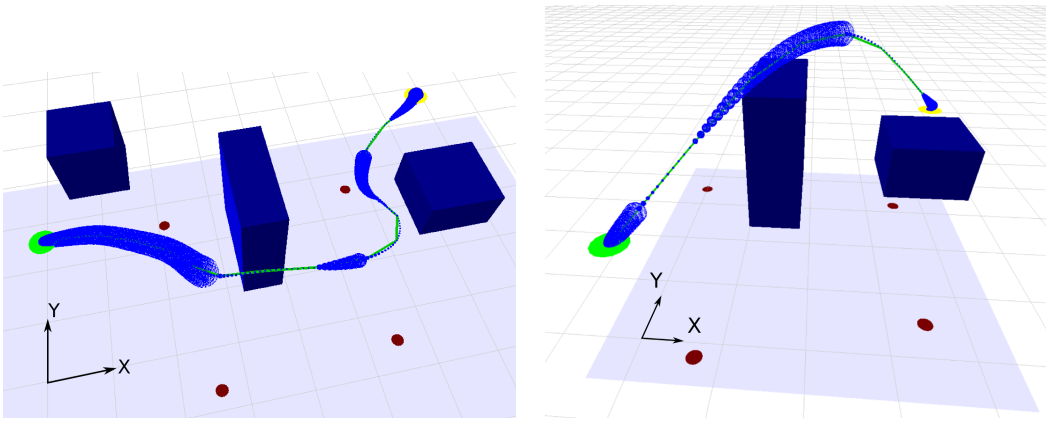
$$l_b \leqslant \boldsymbol{P} \leqslant u_b \tag{8.30d}$$

where $H \in \mathbb{R}^{n\times n}$ denotes a positive (semi-)definite Hessian matrix, $f \in \mathbb{R}^n$ is the gradient vector (null here), $A \in \mathbb{R}^{m\times n}$ is the constraint matrix and $A_{eq} \in \mathbb{R}^{r\times n}$ is the equality constraint matrix. The upper and lower bounds on the constraints are defined by the vectors $b \in \mathbb{R}^m$ and $b_{eq} \in \mathbb{R}^r$ and bounds on the decision variables can be set with $l_b \in \mathbb{R}^n$ and $u_b \in \mathbb{R}^n$ but will be null in our case. Here we will not impose bounds on $\boldsymbol{P}$, i.e, (8.30d) is not considered.

Problem 14 is easily solved using qpOASES [218] that implements an online active set strategy. Note that the connection time T_c is fixed and we found that choosing $T_c = \tau$ generates a reasonable amount of successful connections.

8.7 Simulation and Experimental results

In this section we show some results from different scenarios for the 3D quadrotor. Figure 8.11 shows two optimal trajectories and Fig. 8.12 shows the constrained derivatives along a connection considering B-splines of order 4. With the given degree we can see that the curves (e.g., the jerks) are not penalized by conservatism. Finally, Fig. 8.13 shows the tracking performance for a simulated quadrotor in V-Rep using controller [143]. We can see that despite considering the third-integrator model approximation we can follow the optimal trajectory quite accurately regarding the attitude tracking.



(a) An optimal trajectory providing robust collision avoidance and guaranteed visual perception.

(b) In this case the quadrotor is able to increase its height to compensate for the rotation of the camera and to enlarge its field of view.

Figure 8.11 – Two optimal trajectories for the quadrotor with $\tau = 0.35s$ and $\bar{j} = 10m.s^{-3}$ in a 12x8x5m^3 operating region considering four visual landmarks (red dots). The initial and final states are chosen such that no landmark is visible so the quadrotor starts with some uncertainty and is able to reach the goal with a bounded uncertainty by observing the landmarks during its motion. Note that the motion primitives are not represented.

For the experiment illustrated in this section we used a MK-Quadro equipped with a front-looking camera with a field of view of 45° tracking the AprilTags with ViSP [219]. The setup includes an on-board ODROID-XU4 Linux computer running ROS and the TeleKyb framework for controlling the quadrotor. An optimal trajectory computed off-line using a jerk input $\bar{j} = 4m.s^{-3}$ is tracked by the system (see Fig. 8.14) in presence of two obstacles (blue boxes) and four landmarks. Finally, Problem 14 is solved within 5 ms for the 3D quadrotor after about 90 SQP iterations and the planner is able to find an optimal trajectory in 5 to 10 seconds.

The related video[1] shows 6 different solution trajectories tracked by the quadrotor. Apart from environment differences, several solutions may be found with very

[1] https://www.youtube.com/watch?v=VD3WyA1wEaQ

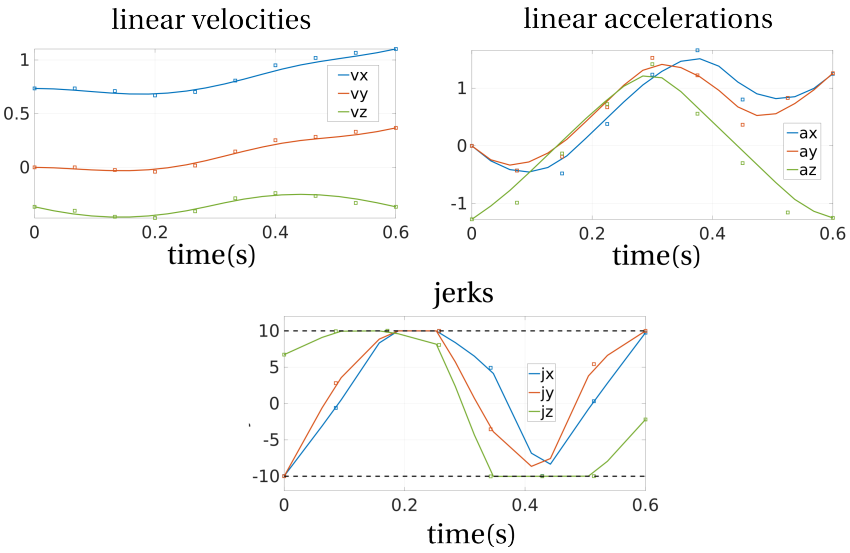


Figure 8.12 – Constrained velocities, accelerations and jerks along the connection trajectory with bounds $\bar{v} = 2m.s^{-1}$, $\bar{a} = 4m.s^{-2}$ and $\bar{j} = 10m.s^{-3}$. The small squares represent the B-spline control points.

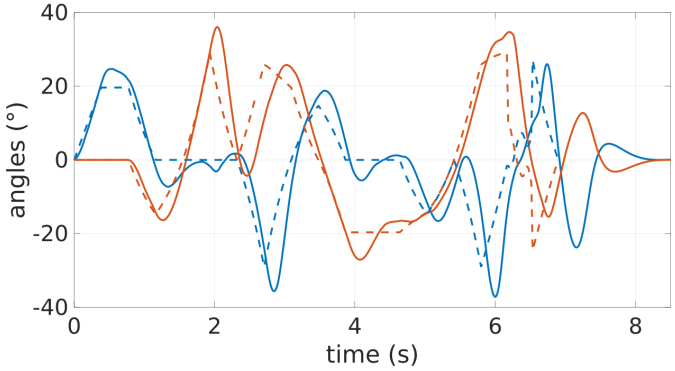


Figure 8.13 – Plots of the attitude tracking. The dashed lines are the command values while the solid lines show the actual robot attitude in V-Rep.

different characteristics and shape. This is explained by the random-based search of the algorithm and by the non-convex nature of the optimal program driving the solutions to local minima. The fifth trajectory Fig. 8.11b shows an interesting behaviour, the quadrotor flies above the two obstacles by increasing its height. This way, the field of view is extended and sufficient visual measurements can be collected. Thus, the presented algorithm is able to generate this behaviour that we depicted in the previous work [2]. Finally, we chose very low values for the process and the measurement noise. The computed trajectory (the sixth one) actually resembles to a minimum-time trajectory in the sense that the quadrotor flies directly toward the goal state at a constant height without "passing" by the visual landmarks. One can conclude that the proposed algorithm is able to generate consistent

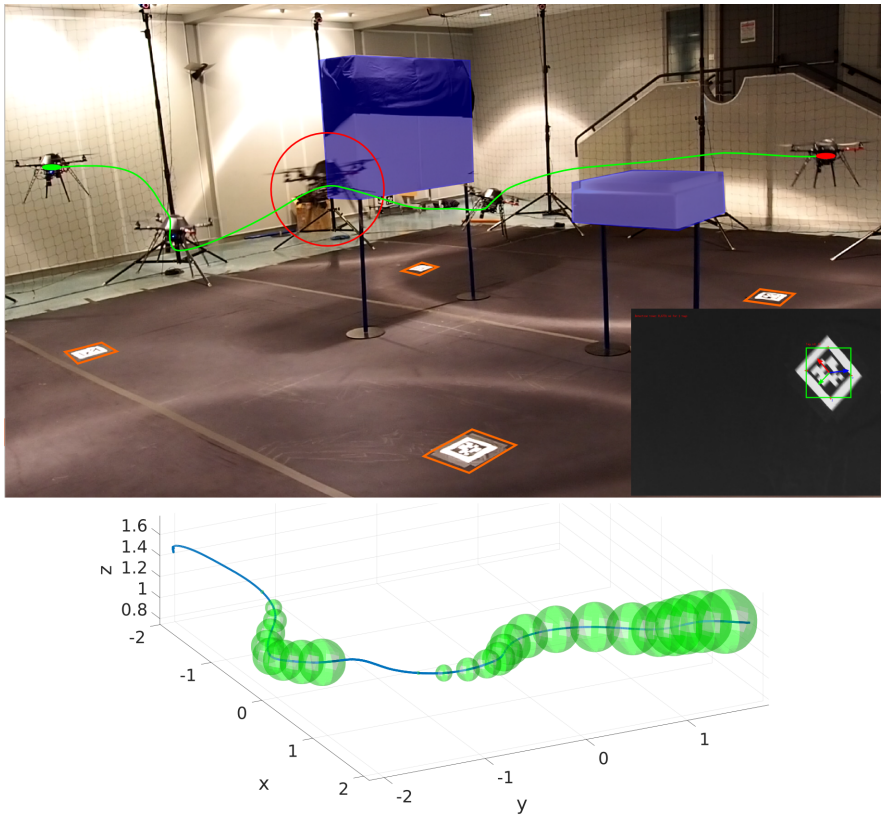


Figure 8.14 – Snapshots of the quadrotor during the experiment. The quadrotor is tracked with a Vicon system and follows an optimal trajectory (in green) along which landmarks (the AprilTags in orange) are visible on some portions. The lower right figure shows an Apriltag tracked using ViSP when the quadrotor is at the configuration circled in red. The evolution of the uncertainty is shown below after running the Kalman filter on the recorded data and using the AprilTags detection. The landmark at the goal is not taken into account in the planning and is was used to check that the quadrotor reaches its goal within the expected confidence region.

solutions regarding the noise magnitude.

8.8 Summary and future directions

In this chapter we proposed to incorporate perception constraints in a graph-search planner for planning minimum-time and feasible trajectories for flat dynamic systems. We believe that the optimization frameworks presented in the previous works which rely on gradient-descent methods may not be adapted to the considered goals in this section. They provided limited state space exploration and were not able to generalize to the large environments considered in this work. The optimization framework allows exact connection between a given initial and final states while ensuring collision avoidance and bounded final uncertainty at the goal by accounting for the state uncertainty at the planning stage. We considered visual measurements

that are a function of the attitude and proposed an efficient optimal graph rewiring by exploiting the convex-hull property of B-splines. Of course, other exteroceptive sensors could be considered such as laser range finders or sonar arrays along with more accurate measurement noise models. One could as well incorporate uncertainties related to wheels skid and odometry bias in the unicycle case. The planner success rate depends on the motion primitives parameters, the ball regions radius and the maximal number of iterations. It could be possible to re-plan optimal trajectories during motion and even consider dynamic obstacles for the unicycle case. We assumed the position of the landmarks is known but it would be possible to incorporate their position uncertainty in the planner. Finally, we believe the triple-integrator approximation of the quadrotor dynamics could become closer to a fourth-integrator model by having an additional noise in the current model. This would give a more adequate representation of the real quadrotor dynamics which would result in a more feasible trajectory.

Part III

Conclusion and future directions

Chapter 9

Conclusion and future directions

In this last chapter, we wish to review the main theoretical and experimental results achieved in the thesis and point out some issues that are still left open. Regarding the latter, we also intend to indicate possible directions to follow for further investigation and research.

9.1 Summary and contributions

The goal of this thesis was mainly to explore the generation of reactive trajectories for a quadrotor subject to visibility constraints and inputs constraints. The considered system consists in a regular quadrotor equipped with a fixed camera (either down- or front-looking). Since we focused on exploiting the potential of a quadrotor in terms of agility to perform aggressive motions, the effect of underactuation could not be ignored, especially because visual perception is very sensitive to the inherent rotation motions. Moreover, we assumed visual feedback played crucial roles either for state estimation or for tracking a moving target. Thus, a collection of visual constraints were formulated in this thesis such as visibility constraints (Sect. 6.7) followed by occlusion avoidance constraints (Sect. 7.6) encapsulated in an optimization framework.

In Sect. 6.7 we proposed re-planning strategies inspired from Model Predictive Control to generate minimum-time and feasible trajectories while keeping a set of visual features in the field of view of the camera. We proposed a hot-start algorithm for building initial guess trajectories by exploiting properties of B-splines curves. We were able to efficiently re-plan optimal trajectories at a rate of 30Hz. This preliminary work led to the resolution of more complex problems in Sect. 7.6 for tackling collisions and occlusions avoidance in real-time. We proposed a multi-objective nonlinear program, first for tracking a free moving target in 3D space with a front-looking camera then for avoiding occlusions generated by spherical obsta-

cles in the environment considering a down-looking camera. We dealt with sudden occlusions and critical camera/target configurations that would lead to a failure of the solver in principle by adding a slack variable to the nonlinear program that acts as a *damper* to relax the occlusions avoidance constraints. Such a parameter helped improving the continuity of the solution and therefore the stability of motion. We showed that tuning the optimization programs do not require fine analysis (although it may be interesting to show a sensitivity analysis) and that we are able to manage conflicting goals to exhibit the desired behaviours. Finally, we succeeded in computing optimal trajectories in real-time by improving the evaluation of the gradients necessary for the SQP solver with the use of complex-step differentiation. With this method we are able to obtain a lightweight evaluation of the complex gradients with a near analytic precision. We showed that the hot-start algorithm definitely contributed in the success of the re-planning framework.

In Chapt. 8 we addressed the problem of planning under intermittent visual measurements provided by visual landmarks scattered in the environment. The objective was to relax the vision-based constraints developed in the previous contributions which may highly restrict the robot motions to limited operating spaces. We proposed a graph-search algorithm that takes state and measurement uncertainties into account to find robust and collision-free trajectories that satisfy a confidence level at the goal state. In contrast to several works we searched for robust minimum-time paths ensuring some confidence level instead of paths with minimal uncertainty. This strategy may make more sense depending on the configuration of the obstacles and the landmarks in the environment. Indeed, if the system and the sensors are well known, the uncertainty is low, thus one may navigate almost directly to the goal and arrive with a limited state uncertainty. On the other hand, if the system is subject to large uncertainties in a complex environment, one also seeks the most direct path connecting the goal state by collecting a sufficient (and ideally a minimal) amount of visual measurements. The proposed algorithm grows two graphs based on a A* variant which are connected smoothly to build a full feasible trajectory for the considered flat dynamic systems. The motivation was to improve the rate of convergence and to plan a trajectory that connects the initial and final states exactly, which is not often the case in the literature. To the best of our knowledge, this is the first time minimum-time trajectories are generated in a graph-search planner while accounting for uncertainty in the visual perception which is affected by the robot attitude. We demonstrated the utility of the algorithm by considering a unicycle and a 3D quadrotor.

In this thesis we extensively used properties of the differential flatness both for trajectory planning and state estimation. We used a derivative-free Kalman filter for the latter in Chapt. 8. This technique contributed in improving the fastness

of the algorithm although is it not suited for real-time planning (at least for the quadrotor case). Differential flatness played a significant role in simplifying trajectory generation to meet real-time planning and for the propagation of feasible motion primitives in Chapt. 8.

Properties of the B-spline curves were also largely exploited in the presented contributions. In Sect. 6.7 we took advantage of the properties of compact form and smoothness along with powerful manipulation algorithms to instantly build initial guess trajectories. Then, such a parametrization allowed the use of a reasonable number of control points to satisfy the considered constraints. Finally, in Chapt. 8 we exploited the convex hull property to develop a quadratic program for smoothly connecting the two graphs with B-spline curves while satisfying linear inequality constraints.

9.2 Open issues and future perspectives

However, even if our approaches presented several promising results it also highlighted some limitations both theoretical and practical.

One limitation of our work is that we relied on an external motion-capture Vicon system providing accurate state estimates at a high frequency. In our works, we assumed that navigation relies on a state estimation algorithm which is updated with the visual measurements extracted from computer vision. The design of the image processing algorithms was beyond the scope of this thesis. Although, it would be relatively easy to implement them using ViSP.

In order to fully prove the presented works, we believe experimental results should be conducted by relying on onboard sensors (a single camera and an inertial measurement unit) and computers for detecting, localizing, and tracking moving objects. As pointed out in Chapt. 8, motion blur may impair computer vision algorithms. One possible solution could be to reduce the system aggressiveness or to directly incorporate motion blur as an additional noise affecting the perception. Another approach could be the minimization of motion blur at the planning state (e.g., by minimizing the features velocity in the image plane or by penalizing the angular rates of the quadrotor). As explained in the experimental sections, we equipped our robot with an ODROID-XU4. However, we think that the new, and more powerful, NVIDIA JETSON TX2 module could be a more reasonable solution for a complete onboard implementation of the vision, planning and state estimation modules.

Although [160] warned that finding images features as flat outputs might be impossible if one considers the full dynamics of the 3D quadrotor we tried to de-

rive a differential flat mapping between the state space and the image space but the equations become very complex (so would be the constraints on the inputs) if one considers the rotation matrix. Furthermore, it would require knowledge of high-order derivatives that are not directly measurable. Maybe *partial differential flatness* (as in e.g., [220]) might bring a weaker but elegant mapping of the dynamics in the image plane.

Now, further improvements and extensions could be considered in Sect. 7.6. We only dealt with spherical obstacles, but more complex shapes could be considered. One possibility could be to find the (minimal) enclosing sphere (i.e., the smallest sphere containing the object). For elongated objects such as bars, one could cover the object main axis with a finite number of spheres as it is done sometimes for collision avoidance with robot manipulators (a technique known as "sphere swept models" see e.g., [221]). However, dealing with dynamic obstacles seems quite challenging. To do so, vision would play a major role in estimating the object's position. Thus, as always, there is a trade-off between conservatism/accuracy and ease of computation.

Finally, more accurate noise models could replace the current ones in Chapt. 8 without adding an excessive burden to the computing. Thus, conservatism could be reduced and more precise trajectories could be found.

9.3 Final thoughts

Aerial navigation has received considerable attention over the last 15 years. Very powerful and efficient forms of optimal planning methods have emerged to tackle more complex scenarios and environments involving obstacles or additional moving parts that traditional controllers were not able to address. In 2011, [1] demonstrated the ability of optimization techniques to design feasible and high-speed flight plans for quadrotors for passing through obstacles with substantial pitch and roll angles. The presented strategy inspired many works to study aggressive motions, feasibility and re-planning strategies. Six years later, [24] was able to reproduce such scenarios in a fully autonomous fashion by re-planning optimal trajectories using vision as principal feedback. This work merges planning and vision and proved that active vision is definitely a key to the future of aerial navigation especially associated with agile manoeuvres. Moreover, since deep-learning has handily surpassed every existing computer vision techniques for tracking, detecting and localizing, it has been very seriously considered in robot control and planning. Now, deep-learning constitutes the state-of-the-art approach across computer vision, audio, and natural language processing and is largely adopted and studied by the research community. Therefore deep-learning is gaining in maturity exponentially and is even about to

replace the action of many existing complex controllers. The reason is not especially because of the recent theoretical improvements made in the field by mainly because of the impressive fastness of today's computers. Indeed, Moore's law stating that the number of transistors in an integrated circuit doubles about every two years is coming to an end (mainly because of heat exchange and electronic disturbances due to proximity of circuits). However, the power of computers keeps growing due to parallelization of processors and the use of graphics processing units (GPU). Now, robots are even able to efficiently develop robust and complex flight plans by learning from their "mistakes" (even from crashes [222]). Besides, learning can be accelerated by using simulation data. If properly done, it can save a considerable amount of time and of course avoid tedious experimental setups and replacements of spare parts. However, such techniques are expensive since they require a considerable computational power which is not always affordable. Although the strength of deep-learning is its ability to generalize and adapt to new situations, a few aspects remain unclear. We are not yet able to prove its robustness, stability and convergence, central criteria that might refrain its adoption in industry especially at the control level. Moreover, huge data sets are needed and may be difficult to obtain depending on the targeted task. Besides, it is known that slightly altering the input data (e.g., changing a few pixels in the input image) of a *well*-trained neural network can lead to absurd outputs. Interpretability in deep learning referring to understanding why a system makes a certain decision is a hot topic and an open problem.

For these reasons, more interpretable techniques such as model-based and analytic control techniques which rely on a long history of research developments will still play a major role in the future. Note that the hidden process of patterns generation in deep-learning has already been modelled to replicate some of its properties to some extent. For instance, a visual servoing task is proposed in [223] that eliminates the need for detecting and tracking image features by using photometric Gaussian mixtures. This strategy subsamples the images and extracts photometric data at increasing levels of precision until convergence. This technique mimics deep-learning and is able to drastically improve the convergence domain of a classical visual servoing task and to reduce the computation expense. Finally, new computers and sensors are being developed and will give the possibility to address even more complex scenarios. Even-based cameras seem to constitute promising elements of the future sensory channel of aerial robots and any system involving fast camera motions.

To conclude, this thesis tackled and revealed challenging problems that are currently addressed actively by the research community. A key aspect to the success of future autonomous and complex navigation seems to lie in coupling of trajectory planning and the considered tasks especially including vision-based perception

objectives. Algorithms requiring extensive computation loads can now be easily deployed on on-board computers to generate safe and robust motions on-line for reactively responding to changes in the environments.

Appendix A

The proof of differential flatness for the quadrotor

In this appendix we give the differential flatness transformation and its inverse for the quadrotor.

A.1 Flat transformation

Defining

$$\boldsymbol{t} := \ddot{\boldsymbol{r}}_{\mathcal{B}} + g\boldsymbol{e_3} \tag{A.1}$$

and considering u_1 is always positive, from (2.14) we obtain the direction of the robot vertical axis

$$\boldsymbol{z}_{\mathcal{B}} = \frac{\boldsymbol{t}}{\|\boldsymbol{t}\|} \tag{A.2}$$

and also the total thrust

$$u_1 = m\|t\| \tag{A.3}$$

Given the yaw angle $\psi = \sigma_4$ we can define the vector:

$$\boldsymbol{y}_{\mathcal{B}} := \boldsymbol{R}_z(\psi)\boldsymbol{e}_2 = (-\sin(\psi) \quad \cos(\psi) \quad 0)^T \tag{A.4}$$

and from (2.2) it is easy to verity that:

$$\boldsymbol{y}_{\mathcal{C}} \times \boldsymbol{z}_{\mathcal{B}} = \cos(\varphi)\boldsymbol{x}_{\mathcal{B}} \tag{A.5}$$

Provided that $\cos(\varphi) > 0$, we are the able to compute $\boldsymbol{x}_{\mathcal{B}}$ as

$$\boldsymbol{x}_{\mathcal{B}} = \frac{\boldsymbol{y}_{\mathcal{C}} \times \boldsymbol{z}_{\mathcal{B}}}{\|\boldsymbol{y}_{\mathcal{C}} \times \boldsymbol{z}_{\mathcal{B}}\|} := \frac{\tilde{\boldsymbol{x}_{\mathcal{B}}}}{\|\tilde{\boldsymbol{x}_{\mathcal{B}}}\|} \tag{A.6}$$

The last axis of the frame $\mathcal{B}$ is simply given by

$$\boldsymbol{y}_{\mathcal{B}} = \boldsymbol{z}_{\mathcal{B}} \times \boldsymbol{x}_{\mathcal{B}} \tag{A.7}$$

and the rotation matrix describing the full 3D orientation of the robot is

$$^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}} = (\boldsymbol{x}_{\mathcal{B}} \quad \boldsymbol{y}_{\mathcal{B}} \quad \boldsymbol{z}_{\mathcal{B}})^T \tag{A.8}$$

Now we take the first derivative of (2.14)

$$m\dot{\boldsymbol{a}}_{\mathcal{B}} = \dot{u}_1 \times \boldsymbol{z}_{\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{BW}} \times u_1 \boldsymbol{z}_{\mathcal{B}} \tag{A.9}$$

Projecting the equation along $\boldsymbol{z}_{\mathcal{B}}$ we obtain

$$\dot{u}_1 = m\boldsymbol{z}_{\mathcal{B}}^T \dot{\boldsymbol{a}}_{\mathcal{B}} \tag{A.10}$$

We can now substitute $\dot{u}_1$ and u_1 back in (A.9) getting

$$\boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{z}_{\mathcal{B}} = \frac{1}{\|t\|}[\dot{\boldsymbol{a}}_{\mathcal{B}} - (\boldsymbol{z}_{\mathcal{B}}^T \dot{\boldsymbol{a}}_{\mathcal{B}})\boldsymbol{z}_{\mathcal{B}}] \tag{A.11a}$$

$$= \frac{1}{\|t\|}(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\dot{\boldsymbol{a}}_{\mathcal{B}} := \boldsymbol{h} \tag{A.11b}$$

We assumed in (2.5) that $\boldsymbol{\omega}_{\mathcal{BW}}$ has components ω_x, ω_y and ω_z in the body frame, i.e.

$$\boldsymbol{\omega}_{\mathcal{BW}} = \omega_x \boldsymbol{x}_{\mathcal{B}} + \omega_y \boldsymbol{y}_{\mathcal{B}} + \omega_z \boldsymbol{z}_{\mathcal{B}} \tag{A.12}$$

then

$$\boldsymbol{h} = (\omega_x \boldsymbol{x}_{\mathcal{B}} + \omega_y \boldsymbol{y}_{\mathcal{B}} + \omega_z \boldsymbol{z}_{\mathcal{B}}) \times \boldsymbol{z}_{\mathcal{B}} = -\omega_x \boldsymbol{y}_{\mathcal{B}} + \omega_y \boldsymbol{x}_{\mathcal{B}} \tag{A.13}$$

and hence

$$\omega_x = -\boldsymbol{h}^T \boldsymbol{y}_{\mathcal{B}} \tag{A.14a}$$

$$\omega_y = \boldsymbol{h}^T \boldsymbol{x}_{\mathcal{B}} \tag{A.14b}$$

The third component ω_z is found by considering that from (2.4)

$$\omega_z = \boldsymbol{y}_{\mathcal{B}}^T \dot{\boldsymbol{x}}_{\mathcal{B}} \tag{A.15}$$

and

$$\dot{\boldsymbol{x}}_{\mathcal{B}} = (\boldsymbol{I} - \boldsymbol{x}_{\mathcal{B}}\boldsymbol{x}_{\mathcal{B}}^T)\frac{\dot{\tilde{\boldsymbol{x}}}_{\mathcal{B}}}{\left\|\dot{\tilde{\boldsymbol{x}}}_{\mathcal{B}}\right\|} \tag{A.16}$$

Then, since $\boldsymbol{y}_{\mathcal{B}}^T \boldsymbol{x}_{\mathcal{B}} = 0$, we can conclude that

$$\omega_z = \boldsymbol{y}_{\mathcal{B}}^T \frac{\dot{\tilde{\boldsymbol{x}}}_{\mathcal{B}}}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|} = \frac{\tilde{\boldsymbol{y}_{\mathcal{B}}}}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}(-\boldsymbol{x}_{\mathcal{C}} \times \dot{\psi}\boldsymbol{z}_{\mathcal{B}} + \boldsymbol{y}_{\mathcal{C}} \times \boldsymbol{h}) \tag{A.17a}$$

$$= \frac{1}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}[\boldsymbol{x}_{\mathcal{C}}^T(\boldsymbol{y}_{\mathcal{B}} \times \dot{\psi}\boldsymbol{z}_{\mathcal{B}}) - \boldsymbol{y}_{\mathcal{C}}^T(\boldsymbol{y}_{\mathcal{B}} \times \boldsymbol{h})] \tag{A.17b}$$

$$= \frac{1}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}(\boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{x}_{\mathcal{B}}\dot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T \boldsymbol{z}_{\mathcal{B}}\omega_y) \tag{A.17c}$$

Once the values of ω_x, ω_y and ω_z are known we are able to compute $\boldsymbol{\omega}_{\mathcal{BW}}$ as:

$$\boldsymbol{\omega}_{\mathcal{BW}} = {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{A.18}$$

To calculate the angular acceleration ${}^{\mathcal{B}}\dot{\boldsymbol{\omega}}_{\mathcal{BW}}$ we operate in the same way. By deriving (A.9) with respect to time we obtain:

$$m\ddot{\boldsymbol{a}}_{\mathcal{B}} = \ddot{u}_1\boldsymbol{z}_{\mathcal{B}} + 2\boldsymbol{\omega}_{\mathcal{BW}} \times \dot{u}_1\boldsymbol{z}_{\mathcal{B}} + \dot{\boldsymbol{\omega}}_{\mathcal{BW}} \times u_1\boldsymbol{z}_{\mathcal{B}} + \boldsymbol{\omega}_{\mathcal{BW}} \times (\boldsymbol{\omega}_{\mathcal{BW}} \times u_1\boldsymbol{z}_{\mathcal{B}}) \tag{A.19}$$

Projecting this equation along $\boldsymbol{z}_{\mathcal{B}}$ we have:

$$m{\boldsymbol{z}_{\mathcal{B}}}^T\ddot{\boldsymbol{a}}_{\mathcal{B}} = \ddot{u}_1 + {\boldsymbol{z}_{\mathcal{B}}}^T[\boldsymbol{\omega}_{\mathcal{BW}} \times (\boldsymbol{\omega}_{\mathcal{BW}} \times u_1\boldsymbol{z}_{\mathcal{B}})] \tag{A.20}$$

from which we can isolate $\ddot{u}_1$:

$$\ddot{u}_1 = m{\boldsymbol{z}_{\mathcal{B}}}^T[\ddot{\boldsymbol{a}}_{\mathcal{B}} - \boldsymbol{\omega}_{\mathcal{BW}} \times (\boldsymbol{\omega}_{\mathcal{BW}} \times \frac{u_1}{m}\boldsymbol{z}_{\mathcal{B}})] \tag{A.21}$$

Substituting $\ddot{u}_1$ in (A.19) and putting

$$\boldsymbol{\delta} := \ddot{\boldsymbol{a}}_{\mathcal{B}} - \boldsymbol{\omega}_{\mathcal{BW}} \times (\boldsymbol{\omega}_{\mathcal{BW}} \times \frac{u_1}{m}\boldsymbol{z}_{\mathcal{B}}) = \ddot{\boldsymbol{a}}_{\mathcal{B}} - \boldsymbol{\omega}_{\mathcal{BW}} \times \|\boldsymbol{t}\|\boldsymbol{h} \tag{A.22}$$

we obtain

$$\dot{\boldsymbol{\omega}}_{\mathcal{BW}} \times \boldsymbol{z}_{\mathcal{B}} = \frac{1}{u_1}[m\boldsymbol{\delta} - m({\boldsymbol{z}_{\mathcal{B}}}^T\boldsymbol{\delta})\boldsymbol{z}_{\mathcal{B}} - 2\boldsymbol{\omega}_{\mathcal{BW}} \times \dot{u}_1\boldsymbol{z}_{\mathcal{B}}] \tag{A.23a}$$

$$= \frac{1}{\|\boldsymbol{t}\|}[(\mathbf{I} - \boldsymbol{z}_{\mathcal{B}}{\boldsymbol{z}_{\mathcal{B}}}^T)\boldsymbol{\delta} - 2({\boldsymbol{z}_{\mathcal{B}}}^T\dot{\boldsymbol{a}}_{\mathcal{B}})\boldsymbol{h}] := \boldsymbol{l} \tag{A.23b}$$

Now assuming that ${}^{\mathcal{B}}\dot{\boldsymbol{\omega}}_{\mathcal{BW}} = (m \quad n \quad o)^T$, and hence

$$\dot{\boldsymbol{\omega}}_{\mathcal{BW}} = m\boldsymbol{x}_{\mathcal{B}} + n\boldsymbol{y}_{\mathcal{B}} + o\boldsymbol{z}_{\mathcal{B}} \tag{A.24}$$

It is easy to verify that

$$m = -\boldsymbol{l}^T\boldsymbol{y}_{\mathcal{B}} \tag{A.25a}$$

$$n = \boldsymbol{l}^T\boldsymbol{x}_{\mathcal{B}} \tag{A.25b}$$

The third component o is found by taking the derivative of (A.19)

$$o = \frac{1}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}(\boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{x}_{\mathcal{B}}\dot{\psi}^2 + \boldsymbol{x}_{\mathcal{C}}^T\dot{\boldsymbol{x}}\mathcal{B}\dot{\psi} + \boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{x}_{\mathcal{B}}\ddot{\psi} - \boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}\omega_y\dot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{h}\omega_z + \boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}n) \tag{A.26a}$$

$$- \frac{\tilde{\boldsymbol{x}}_{\mathcal{B}}^T\dot{\tilde{\boldsymbol{x}}}_{\mathcal{B}}}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|^3}(\boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{x}_{\mathcal{B}}\dot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}\omega_y) \tag{A.26b}$$

$$= \frac{1}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}(\boldsymbol{x}_{\mathcal{C}}^T\dot{\boldsymbol{x}}\mathcal{B}\dot{\psi} + \boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{x}_{\mathcal{B}}\ddot{\psi} - \boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}\omega_y\dot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{h}\omega_z + \boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}n - \boldsymbol{x}_{\mathcal{B}}^T\dot{\tilde{\boldsymbol{x}}}_{\mathcal{B}}\omega_z) \tag{A.26c}$$

Since

$$\boldsymbol{y}_{\mathcal{C}}^T \boldsymbol{h} = -{\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}}^T (\boldsymbol{y}_{\mathcal{C}} \times \boldsymbol{z}_{\mathcal{B}}) = -{\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}}^T \tilde{\boldsymbol{x}}_{\mathcal{B}} = -\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|\omega_x \tag{A.27a}$$

$$\tilde{\boldsymbol{x}}_{\mathcal{B}}^T \dot{\tilde{\boldsymbol{x}}}_{\mathcal{B}} = \boldsymbol{x}_{\mathcal{C}}^T (\boldsymbol{x}_{\mathcal{B}} \times \boldsymbol{z}_{\mathcal{B}} \dot{\psi}) - \boldsymbol{y}_{\mathcal{C}}^T (\boldsymbol{x}_{\mathcal{B}} \times \boldsymbol{h}) = \boldsymbol{y}_{\mathcal{C}}^T \boldsymbol{z}_{\mathcal{B}} \omega_x - \boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{y}_{\mathcal{B}} \dot{\psi} \tag{A.27b}$$

we have

$$o = \frac{1}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}[\boldsymbol{x}_{\mathcal{C}}^T \dot{\boldsymbol{x}}_{\mathcal{B}} \dot{\psi} + \boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{x}_{\mathcal{B}} \ddot{\psi} + \boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{y}_{\mathcal{B}} \omega_z \dot{\psi} - \boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{z}_{\mathcal{B}} \omega_y \dot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T \boldsymbol{z}_{\mathcal{B}} (n - \omega_x \omega_z)] - \omega_x \omega_y \tag{A.28}$$

Moreover from (2.4) we obtain

$$\boldsymbol{x}_{\mathcal{B}}^T \dot{\boldsymbol{x}}_{\mathcal{B}} = 0 \tag{A.29a}$$

$$\boldsymbol{y}_{\mathcal{B}}^T \dot{\boldsymbol{x}}_{\mathcal{B}} = \omega_z \tag{A.29b}$$

$$\boldsymbol{z}_{\mathcal{B}}^T \dot{\boldsymbol{x}}_{\mathcal{B}} = -\omega_y \tag{A.29c}$$

then

$$\boldsymbol{x}_{\mathcal{C}}^T \dot{\boldsymbol{x}}_{\mathcal{B}} = \boldsymbol{x}_{\mathcal{C}}^T (\boldsymbol{y}_{\mathcal{B}} \omega_z - \boldsymbol{z}_{\mathcal{B}} \omega_y) \tag{A.30}$$

and we conclude that

$$o = \frac{1}{\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\|}[2(\boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{y}_{\mathcal{B}} \omega_z - \boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{z}_{\mathcal{B}} \omega_y) \dot{\psi} + \boldsymbol{x}_{\mathcal{C}}^T \boldsymbol{x}_{\mathcal{B}} \ddot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T \boldsymbol{z}_{\mathcal{B}} (n - \omega_x \omega_z)] - \omega_x \omega_y \tag{A.31}$$

Finally, from Sect. 2.2 we compute the remaining inputs (u_2, u_3, u_4)

As already said, the strength of differential flatness is to transform the system such that the equations of motion for the flat output variables become trivial. Using the flat output and its derivatives, the system of (2.16) can be written in the Brunovsky linear canonical form:

$$\frac{d^4 x}{dt} = v_1 \tag{A.32a}$$

$$\frac{d^4 y}{dt} = v_2 \tag{A.32b}$$

$$\frac{d^4 z}{dt} = v_3 \tag{A.32c}$$

$$\frac{d^2 \psi}{dt} = v_4 \tag{A.32d}$$

One can define a new system $\dot{\boldsymbol{z}}(t) = A\boldsymbol{z}(t) + B\boldsymbol{v}(t)$ with new control inputs $\boldsymbol{v}$ with $\boldsymbol{v} = (v_1, v_2, v_3, v_4)^T$. With

$$A = \begin{bmatrix} A_1 & 0_{4\times4} & 0_{4\times4} & 0_{4\times2} \\ 0_{4\times4} & A_1 & 0_{4\times4} & 0_{4\times2} \\ 0_{4\times4} & 0_{4\times4} & A_1 & 0_{4\times2} \\ 0_{2\times4} & 0_{2\times4} & 0_{2\times4} & A_2 \end{bmatrix} \tag{A.33}$$

$$B = \begin{bmatrix} 0_{3\times4} \\ B_1 \\ 0_{3\times4} \\ B_2 \\ 0_{3\times4} \\ B_3 \\ 0_{1\times4} \\ B_4 \end{bmatrix} \tag{A.34}$$

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad A_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{A.35}$$

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{A.36a}$$

$$B_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \tag{A.36b}$$

$$B_3 = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \tag{A.36c}$$

$$B_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.36d}$$

$$\tag{A.36e}$$

A.1.1 Inverse flat transformation

Now, we deal with the inverse problem of the one studied in Appendix A.1: given the state of the robot in terms of $\boldsymbol{r}_{\mathcal{B}}$, $\dot{\boldsymbol{r}}_{\mathcal{B}}$, ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ and ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}$ and possibly the input vector $\boldsymbol{u}$, we want to compute the value of the flat outputs and their derivatives. The position vector and its derivative are simply contained in the state and can immediately be extracted from it. Using the equations reported at the beginning of Sect. 2.2 we can compute the roll, pitch and yaw angles and their derivatives from the state components ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ and ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}$. The linear acceleration is given by (2.14)

$$\ddot{\boldsymbol{r}}_{\mathcal{B}} = \frac{u_1}{m}\boldsymbol{z}_{\mathcal{B}} - g\boldsymbol{e}_3 \tag{A.37}$$

If the thrust is fixed then $\ddot{\boldsymbol{r}}_{\mathcal{B}}$ is univoquelly defined, otherwise any value satisfying the equation

$$(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\ddot{\boldsymbol{r}}_{\mathcal{B}} = -g(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\boldsymbol{e}_3 \tag{A.38}$$

is valid. In general we can split $\ddot{\boldsymbol{r}}_{\mathcal{B}}$ into its components orthogonal and parallel to the local axis $\boldsymbol{z}_{\mathcal{B}}$

$$\ddot{\boldsymbol{r}}_{\mathcal{B}} = -g(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\boldsymbol{e}_3 + (u_1 - g\boldsymbol{z}_{\mathcal{B}}^T\boldsymbol{e}_3)\boldsymbol{z}_{\mathcal{B}} \tag{A.39}$$

and it is clear that the latter component can be chosen at will (inside an admissible interval) since it is controlled by the total thrust input u_1. Once we know $\ddot{\boldsymbol{r}}_{\mathcal{B}}$, we define

$$\boldsymbol{f} = \ddot{\boldsymbol{r}}_{\mathcal{B}} + g\boldsymbol{e}_3 \tag{A.40}$$

and we compute the component of $\dot{\boldsymbol{a}}_{\mathcal{B}}$ orthogonal to $\boldsymbol{z}_{\mathcal{B}}$ from (A.11) solving

$$(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\dot{\boldsymbol{a}}_{\mathcal{B}} = \|\boldsymbol{f}\|^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}} \times \boldsymbol{z}_{\mathcal{B}} \tag{A.41}$$

The minimum norm solution is

$$\dot{\boldsymbol{a}}_{\mathcal{B}} = \|\boldsymbol{f}\|^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}} \times \boldsymbol{z}_{\mathcal{B}} \tag{A.42}$$

If $\dot{u}_1$ is fixed, then we must add to $\dot{\boldsymbol{a}}_{\mathcal{B}}$ a component along the $\boldsymbol{z}_{\mathcal{B}}$ axis such that (A.10) is satisfied, i.e.,

$$\dot{\boldsymbol{a}}_{\mathcal{B}} = \|\boldsymbol{f}\|^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}} \times \boldsymbol{z}_{\mathcal{B}} + \frac{\dot{u}_1}{m}\boldsymbol{z}_{\mathcal{B}} \tag{A.43}$$

Assuming that the torque inputs u_2, u_3 and u_4 are known we can compute the angular acceleration in the body frame from Sect. 2.2

$${}^{\mathcal{B}}\dot{\boldsymbol{\omega}}_{\mathcal{B}\mathcal{W}} = \boldsymbol{J}\left[\begin{pmatrix} u_2 \\ u_3 \\ u_4 \end{pmatrix} - {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}} \times \boldsymbol{J}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}\right] \tag{A.44}$$

and using (A.31) we compute $\ddot{\psi}$ as

$$\ddot{\psi} = \frac{(o + \omega_x\omega_y)\|\tilde{\boldsymbol{x}}_{\mathcal{B}}\| - \left[2\left(\boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{y}_{\mathcal{B}}\omega_z - \boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}\omega_y\right)\dot{\psi} + \boldsymbol{y}_{\mathcal{C}}^T\boldsymbol{z}_{\mathcal{B}}(n - \omega_x\omega_z)\right]}{\boldsymbol{x}_{\mathcal{C}}^T\boldsymbol{x}_{\mathcal{B}}} \tag{A.45}$$

where n and o are the last two components of ${}^{\mathcal{B}}\dot{\boldsymbol{\omega}}_{\mathcal{B}\mathcal{W}}$. We also compute the component of $\boldsymbol{\delta}$ orthogonal to $\boldsymbol{z}_{\mathcal{B}}$ by solving the system

$$(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\boldsymbol{\delta} = [\|\boldsymbol{f}\|\dot{\boldsymbol{\omega}}_{\mathcal{B}\mathcal{W}} + 2(\boldsymbol{z}_{\mathcal{B}}^T\dot{\boldsymbol{a}}_{\mathcal{B}})\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}] \times \boldsymbol{z}_{\mathcal{B}} \tag{A.46}$$

derived from (A.23). Again the minimum norm is:

$$\boldsymbol{\delta} = \left[\|\boldsymbol{f}\|\dot{\boldsymbol{\omega}}_{\mathcal{B}\mathcal{W}} + 2(\boldsymbol{z}_{\mathcal{B}}^T\dot{\boldsymbol{a}}_{\mathcal{B}})\boldsymbol{\omega}_{\mathcal{B}\mathcal{W}}\right] \times \boldsymbol{z}_{\mathcal{B}} \tag{A.47}$$

If $\dot{u}_1$ is not fixed and we chose $\dot{\boldsymbol{a}}_{\mathcal{B}}$ according to (A.42), then the above equations simplify to

$$(\boldsymbol{I} - \boldsymbol{z}_{\mathcal{B}}\boldsymbol{z}_{\mathcal{B}}^T)\boldsymbol{\delta} = \dot{\boldsymbol{\omega}}_{\mathcal{B}\mathcal{W}} \times \|\boldsymbol{f}\|\boldsymbol{z}_{\mathcal{B}} \tag{A.48}$$

and

$$\boldsymbol{\delta} = \dot{\boldsymbol{\omega}}_{\mathcal{B}\mathcal{W}} \times \|\boldsymbol{f}\|\boldsymbol{z}_{\mathcal{B}} \tag{A.49}$$

It is easy to demonstrate that if $\ddot{u}_1$ is also given, then in order to satisfy (A.21) we have to add to $\boldsymbol{\delta}$ a component along $\boldsymbol{z}_{\mathcal{B}}$, i.e.,

$$\boldsymbol{\delta} = [\|\boldsymbol{f}\|\dot{\boldsymbol{\omega}}_{\mathcal{BW}} + 2(\boldsymbol{z}_{\mathcal{B}}^T\dot{\boldsymbol{a}}_{\mathcal{B}})\boldsymbol{\omega}_{\mathcal{BW}}] \times \boldsymbol{z}_{\mathcal{B}} + \frac{\ddot{u}_1}{m}\boldsymbol{z}_{\mathcal{B}} \tag{A.50}$$

Finally, inverting (A.22), we obtain

$$\ddot{\boldsymbol{a}}_{\mathcal{B}} = \boldsymbol{\delta} + \boldsymbol{\omega}_{\mathcal{BW}} \times (\boldsymbol{\omega}_{\mathcal{BW}} \times \|\boldsymbol{f}\|\boldsymbol{z}_{\mathcal{B}}) \tag{A.51}$$

Appendix B

Parametrization using B-splines

An overview of B-splines, from which much of the following is derived, can be found in De Boor [126].

B.1 B-spline curve properties

In this section we provide a complete list of B-spline properties. Several of them are exploited in this thesis to derive algorithms to serve planning and optimization purposes.

- $B_{i,p}(u)$ is a piecewise polynomial of degree p;
- $B_{i,p}(u)$ has a minimum local support, i.e. it is equal to zero outside the interval $[u_i, u_{i+p+1}]$.;
- Non negativity: the basis functions are positive;
- Geometry invariance: the B-spline basis function defines a partition of the unity, i.e.

$$\sum_{i=0}^{n} B_{i,p}(s) = 1 \;\; \forall u \in [u_0, u_K] \tag{B.1}$$

  which assures the B-spline is invariant under affine transformations (translation, rotation or scaling) of its control points;
- Local support: the function $B_{i,k}$ is zero outside $[u_i, u_{i+k}]$. This means that the change of a control point $\boldsymbol{P}_i$ only modifies the spline in the interval $[u_i, u_{i+k+1}]$
- the B-spline can be scaled or translated in time by scaling or translating the knot vector. The derivatives will scale or translate accordingly, in particular if $\hat{\boldsymbol{U}} = \lambda \boldsymbol{U}$ then $\hat{u}^{(i)}(t) = \frac{u^{(i)}(t)}{\lambda^i}$;

- *convex hull*: because the basis functions are positive and sum up to one, a spline is always contained in the convex hull of its *control polygon* which is the convex hull of the spline control points. This polygon corresponds to the piecewise linear interpolation of the spline coefficients.
- the B-spline is of class $\mathbf{C}^{\infty}$ in the interior of every knot span and it is of class $\mathbf{C}^{p-m}$ in a knot of multiplicity m;
- the number of knots $K+1$ is related to the number of control points n and to the order of the curve k by $K = n + k$
- the derivative of a B-spline is also a B-spline of lower degree. Indeed

$$s^{(i)}(t) = \sum_{i=1}^{n-1} B_{i,p}(t)\boldsymbol{P} \quad u_0 \leqslant t \leqslant u_K \tag{B.2}$$

and it is possible to efficiently compute the r-th order derivative of the basis functions in terms of the basis functions of degree $p-i$ defined on the same knot vector $\boldsymbol{U}$

$$B_{i,p}^{(r)}(u) = \frac{p!}{(p-r)!} \sum_{l=0}^{r} a_{r,l} B_{i+1,k-r}(u) \tag{B.3}$$

where the coefficients $a_{r,l}$ are defined in a recursive way

$$a_{0,0} = 1 \tag{B.4a}$$

$$a_{r,0} = \frac{a_{r-1,0}}{u_{i+p-r+1} - u_i} \tag{B.4b}$$

$$a_{r,i} = \frac{a_{r-1,l} - a_{r-1,l-1}}{u_{i+p-r+l+1} - u_{i+l}}, \quad for \quad l = 1, ..., r-1 \tag{B.4c}$$

$$a_{r,r} = \frac{-a_{r-1,l-1}}{u_{i+p+1} - u_{i+1}} \tag{B.4d}$$

Thanks to all these properties, B-splines have been widely used in different applications such as computer graphics, data interpolation and trajectory planning, e.g., [101], [224]. For an exhaustive description of the B-splines and their properties see [225].

Figure B.1 shows basis functions of a B-spline curve. If the initial and final knots have multiplicity k then the B-spline curve is *clamped*– the first and last control points coincide with the endpoints of the curve, i.e. $s(u_0) = \boldsymbol{P}_0$ and $s(u_K) = \boldsymbol{P}_n$.

B.2 Manipulation algorithms

B-spline curves benefit from very powerful algorithms such as evaluation, knot insertion, knot removal and subdivision. A brief description of these algorithms are given in this section.

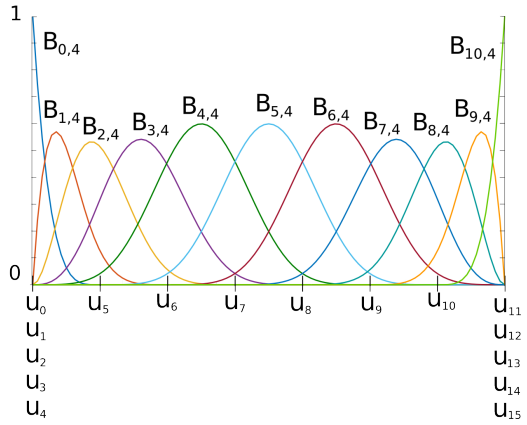


Figure B.1 – Basis functions of an degree four (order five) B-spline with 11 control points. The knot vector contains 16 knots.

- evaluation algorithm: A B-spline curve can be evaluated at a parametric point u using the *De Boor* algorithm given in *De Boor* and explained in Appendix B.2.1.

- knot insertion algorithm: A knot can be inserted into a B-spline without changing the geometry of the curve. The new curve is identical to the old one, with a new basis.

  The algorithm is known as *Boehm's algorithm*. Inserting knots is generally used for refining the curve at a specific region and corresponds to an order elevation. As a result, the modified curve will get closer to its convex hull. If a knot is inserted at u as many times as the order of the original B-spline, the curve and the convex hull will coincide at u.

- knot removal algorithm: Knots can be removed for data reduction and curve approximation.

- subdivision algorithm: A B-spline curve can be subdivided into multiple B-splines without changing the shape of the original curve. A subdivision at a specific parameter u can be performed either by applying the de Boor algorithm at u or by inserting a knot p times at u, where p is the B-spline curve degree. Such a process uses the De Boor algorithm and is detailed in the next section.

B.2.1 The curve subdivision algorithm

This section details the procedure for subdividing a B-spline curve of degree p with control points $\boldsymbol{P} = (P_0, P_1, \ldots, P_n)$ related to the knot vector $\boldsymbol{U}$ in two B-spline

curves without modifying the shape of the original curve. This algorithm is the core of the hot-start algorithm presented in Sect. 6.4.2.

First, the function *findspan* finds the knot span $[u_k, u_{k+1})$ containing u (see [226] p.80 for details on the algorithm). From the convex hull property (see the previous section), $s(u)$ lies in the convex hull defined by the control points $\boldsymbol{P_k} = (P_{k-p}, P_{k-p+1}, \ldots, P_k)$. Now, we show the procedure for subdividing a B-spline curve in two at the point u. Thus, the output of the algorithm consists in two B-spline curves, one has the *left* curve defined by the pair $\{\boldsymbol{P}_L, \boldsymbol{U}_L\}$ and the *right* curve defined by the pair $\{\boldsymbol{P}_R, \boldsymbol{U}_R\}$ (see Fig. B.2). The algorithm is referred as the De Boor algorithm and necessitates operations only on the subset $\boldsymbol{P_k}$ of the control points $\boldsymbol{P}$. The algorithm performs the insertion of the knot u p times. The shape of the curve is unchanged but the two curve halfs become independent. Indeed, curves of degree p corresponding to a knot vector with a knot u of multiplicity p have their local support contained in either $[0, u]$ or $[u, 1]$. In Fig. B.3 we show how the control points of the two curves are determined using the De Boor algorithm. In the end one has $\boldsymbol{P}_L = (P_0, P_{1,1}, P_{2,2}, \ldots, P_{p,p})$ with knot vector $\boldsymbol{U}_L = ([0, u), \underbrace{u}_{p})$ and $\boldsymbol{P}_R = (P_{p,p}, P_{p-1,p}, \ldots, P_{1,p}, P_p)$ with knot vector $\boldsymbol{U}_R = (\underbrace{u}_{p}, (u, 1])$.

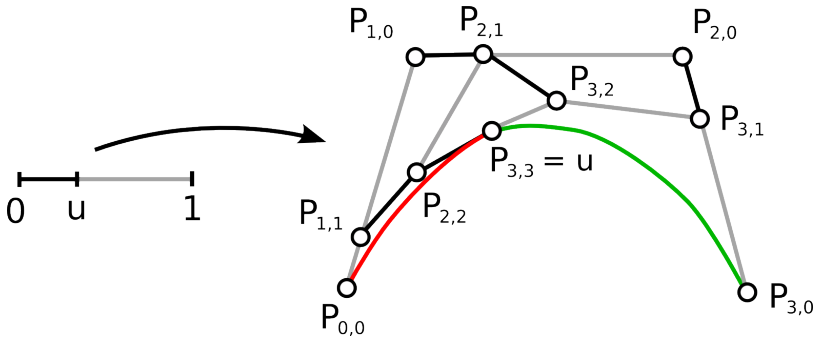


Figure B.2 – Running the De Boor algorithm at the parametric point u. At the final knot insertion, the last control point $P_{3,3} = u$ resulting in two independent *left* and *right* curves.

For our hot-start algorithm, we are interested in the *right* section since it corresponds to the *future* trajectory that will be adapted and used as initial guess for the next solver instance. Once the original curve (i.e., the previous solution) is split, we need to add the potential missing knots until the length of $\boldsymbol{U}_L$ matches the length of $\boldsymbol{U}$. Indeed,the length of $\boldsymbol{U}_L$ varies with the position of u in the knot span. Such an operation is done with the *knot insertion* algorithm detailed in [226] p.161. To do so, control points are also added. Then, $\boldsymbol{U}_L$ is rescaled between $[0, 1]$ to finally match the original uniform knot vector $\boldsymbol{U}$. In the end, we have a new vector of control points $\hat{\boldsymbol{P}}_L$ of same length as $\boldsymbol{P}$ defining the new B-spline curve with the pair $\left\{\hat{\boldsymbol{P}}_L, \boldsymbol{U}\right\}$ of degree p.

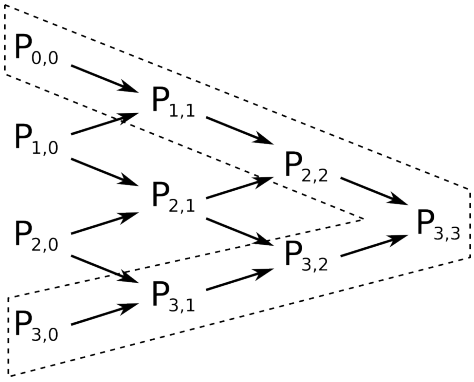


Figure B.3 – Data flow diagram for the De Boor algorithm. The enveloppe contains the control points of the two independent curves.

Appendix C

Gradient evaluation

C.1 On derivatives evaluation

Since we rely on gradient descent optimization algorithm, the quality of the gradient evaluation is central for driving the optimal solution towards a local minimum. Moreover, we are concerned with its accuracy and computational burden.

Differentiation results are well-known for certain classes of functions (quadratic functions for instance), but can be tricky for others. Although, analytic differentiation can be computed for complex constraints or terms in the cost function, their formulation generally inflates and may be tedious and subject to errors. Moreover, when one needs to code them it may take a huge amount of space and memory. Therefore, numerical solutions may become more attractive. In this section, we discuss and compare the most popular numerical differentiation techniques: finite difference, automatic differentiation, complex-step differentiation. Here, the main concerns are the accuracy, the numerical stability and the computation load.

C.2 Gradient approximation techniques

The technique of differentiation was introduced independently by Isaac Newton (1642−1727) and Gottfried Leibniz (1646−1716). Formally, the slope of the tangent line at a point x is the limit of the ratio of the change in the function to the change in the independent variable, as that change approaches 0, i.e.:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{C.1}$$

The quotient in (C.1) is referred to as the Newton quotient or the difference quotient. Another way of expressing the derivative of a function derives from its expansion in a Taylor series, introduced by Brook Taylor in 1715. The Taylor series expresses any analytic real or complex function at a real or complex number a by an infinite

sum over its derivative terms:

$$f(x) = f(a) + \frac{f^{'}(a)}{1!}(x-a) + \frac{f^{''}(a)}{2!}(x-a)^2 + ... = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \quad \text{(C.2)}$$

Hence, for an analytic function, differentiation is equivalent to evaluating terms of a Taylor series.

C.2.1 Finite difference method

One method that is very commonly used is finite differencing. Although it is not known for being particularly accurate or computationally efficient, the biggest advantage of this method lies in the fact that it is extremely easy to implement.

Taking $a = x + \Delta x$ in equation (C.2) and reordering we can obtain an expression for the first order derivative similar to (C.1):

$$f^{'}(x) = \frac{f(x+\Delta x) - f(x)}{\Delta x} + \mathcal{O}(\Delta x) \quad \text{(C.3)}$$

An expression like (C.3) is called a Finite Difference (FD) approximation, in this case the first-order forward approximation for the first derivative, where the differential step is taken in the positive direction. The symbol $\mathcal{O}$ expresses the error related to truncating the Taylor series in the second order derivative. Fourth-order accuracy can be achieved, of course at the price of increasing computational cost. Whatever the order of accuracy, all FD approximations involve a truncation error depending on the step size Δx. FD approximations are still the most classic, simple and intuitive approaches to approximate derivatives of a function, and are widely used in numerical schemes. However they suffer from numerical issues related to the "step-size dilemma", that is, the desire to choose a small step size to minimize truncation error while avoiding the use of a step so small that errors due to subtractive cancellation become dominant [122].

Working with arbitrarily small steps Δx is not feasible on a computer. FD schemes, as the name suggests, involve some difference operator in the numerator, and this difference itself is an intrinsic problem. For a given step size Δx, and particularly for small steps, the differences of the values of our function at successive evaluation points may become small, leading to a loss of significant digits as one approaches machine precision, and eventually a value zero for the numerator and the derivative when the computer fails to recognize the difference between the two numbers. This problem is known as subtractive cancellation or term cancellation. Since in numerical simulations we often have little hints on the actual shape of the functions involved, subtractive cancellation is not straightforward to control, which forces us into a conservative choice of step size at the expense of larger truncation

errors. On the other hand, large values of Δx may create instabilities and affect the data quality. In the next section we introduce a less intuitive technique when Δx is a complex number that provides better numerical accuracy and stability than FD.

C.2.2 Complex-step differentiation

Most naturally, derivatives of real functions are evaluated using real numbers, but the less intuitive idea of using an imaginary number in real functions differentiation has been shown capable of overcoming the term cancellation inherent to the ordinary FD method, as well as reducing the associated approximation error. The use of complex variables in numerical differentiation was introduced by [227], describing a method for computing the derivatives of any analytic function. After falling into oblivion for 20 years, this theory reappeared in the scientific literature when Squire and Trapp [197] formally presented the Complex Step Method (CS) to obtain a very simple expression for estimating the first and second derivatives of a real function using a purely imaginary number i ($i^2 = -1$). This estimate is suitable for use in modern numerical computing and has been shown to be very accurate, extremely robust and surprisingly easy to implement, while retaining a reasonable computational cost. Further research on the subject has been carried out by [228] for sensitivity analysis.

The CS method can be very easily derived from the Taylor series expansion of $f(x + \boldsymbol{i}\Delta x)$, i.e.,

$$f(x + \boldsymbol{i}\Delta x) = f(x) + \boldsymbol{i}\Delta x f^{'}(x) + \frac{(\boldsymbol{i}\Delta x)^2}{2!} f^{''}(x) + \frac{(\boldsymbol{i}\Delta x)^3}{3!} f^{'''}(x) + \dots \tag{C.4a}$$

$$= \sum_{n=0}^{\infty} \frac{(\boldsymbol{i}\Delta x)^n}{n!} f^{(n)}(x) \tag{C.4b}$$

Taking the imaginary part on both sides and reordering we obtain the CS expression for the first derivative found by [197]

$$f^{'}(x) = \frac{\Im(f(x + \boldsymbol{i}\Delta x))}{\Delta x} + \mathcal{O}(\Delta x^2) \tag{C.5}$$

Note that, $\Im(f(x)) = 0$ because x is set to be a real number. Compared to (C.1) this solution is not a function of differences, which ultimately provides better accuracy than a standard finite difference. The second order term in the Taylor series expansion of $f(x + \boldsymbol{i}\Delta x)$ appears with a factor of $\boldsymbol{i}^2$, meaning that it is a real quantity. Compared with (C.3) the truncation error is now of order Δx^2, thus smaller.
An expression for the second order derivative can be found by taking the real part of (C.4) and reordering,

$$f^{''}(x) = \frac{2(f(x) - \Re(f(x + \boldsymbol{i}\Delta x)))}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{C.6}$$

Using the Taylor series expansion of $f(x - \boldsymbol{i}\Delta x)$, it can be verified that

$$Re(f(x + \boldsymbol{i}\Delta x)) = \Re(f(x - \boldsymbol{i}\Delta x)) \tag{C.7}$$

Therefore, (C.6) can be written as

$$f''(x) = \frac{2(f(x) - \Re(f(x - \boldsymbol{i}\Delta x)))}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{C.8}$$

Equation (C.5) and (C.6) are the most basic equations that can be found using (C.4). The numerical advantages of the CS method are noticeable: Equation (C.5) actually shows a single term in the numerator rather than a difference, and hereby circumvents the instability related to term cancellation inherent to all classic, real valued FD approximations besides being more accurate. Equation (C.6) and (C.8) allows to compute an approximation to the second derivative in a single step that cannot be achieved by any FD approximation.

Generalizations to high order derivatives made by [229] and [230] were done by converting the Taylor series into a Fourier series (Taylor expansion of $f(x + \Delta x e^{i\theta})$, i.e.,

$$f(x + \Delta x e^{i\theta}) = f(x) + \Delta x e^{i\theta} f'(x) + \frac{\Delta x^2}{2!} e^{2i\theta} f''(x) + \frac{\Delta x^3}{3!} e^{3i\theta} f'''(x) + \dots \tag{C.9a}$$

$$= \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} e^{ni\theta} f^{(n)}(x) \tag{C.9b}$$

In the expression (C.9) the imaginary step does not vanish with even powers of the Taylor series which allows to compute high order derivatives by combining different Δx steps values and using the real or imaginary part without the limitations of the ordinary CS method. The main limitation of this formulation is that the real and imaginary steps are set to be orthogonal ($e^{i\theta} = \cos\theta + \boldsymbol{i}\sin\theta$) depending on a parameter θ. In other words we cannot choose the relation between real and imaginary step sizes which brings many advantages as discussed below.

For first derivatives the complex-step approach does not suffer subtraction cancellation errors as in standard numerical finite-difference approaches. Therefore, since an arbitrarily small step-size can be chosen, the complex-step method can achieve near analytical accuracy. However, implementation of the complex-step approach for second derivatives does suffer from round-off errors. Therefore, an arbitrarily small step-size cannot be chosen. Moreover, one of the limitations of the CS method is that only the first-order derivative is accessible using the imaginary part of the function, while second derivatives are proportional to i^2 and have to be evaluated by taking the real part of the function.

The advantages of the complex-step approximation approach over a standard finite difference include: 1) the Jacobian approximation is not subject to subtractive cancellations inherent in roundoff errors, 2) it can be used on discontinuous

functions, and 3) it is easy to implement in a black-box manner, thereby making it applicable to general nonlinear functions.

C.2.3 Automatic differentiation

In terms of implementation, the continuous approach can only be derived by hand, while the discrete approach to differentiation can be implemented automatically if the program that solves the discretized governing equations is provided. This method is known as *algorithmic differentiation*, *computational differentiation* or *automatic differentiation*. It is a well-known method based on the systematic application of the chain rule of differentiation to computer programs [231]. This approach is as accurate as other analytic methods, and it is considerably easier to implement.

C.2.4 Implementations

The implementation of any of the derivative calculation methods, for practical purposes, should be as automated as possible. Changing the source code manually is not only an extremely tedious task, but is also likely to result in the introduction of coding errors in the program. There are two main possibilities for implementing algorithmic differentiation: by source code transformation or by using derived data types and operator overloading. To implement algorithmic differentiation by source transformation, the whole code must be processed with a parser and all the derivative calculations are introduced as extra lines of code. The resulting extended code is greatly enlarged and it becomes practically unreadable. This fact constitutes an implementation disadvantage as it becomes impractical to debug this new extended code.

However, for CS several mathematical functions need to be rewritten in their complex form before implementing the gradient evaluation. In the next section we provide a few functions that we needed in our optimal control problems.

C.2.5 Table of complex functions

For CS we need to keep both the real and the complex part of every functions involved in the constraints evaluation. Therefore, some functions need to be used in their complex form, see table C.1.

Of course the computation is increased and from practical aspects, the code length is inevitably larger after implementing the aforementioned transformation but only the parts involved in the constraints evaluation are concerned, which is not the case of AD that requires a full overload of the code. In the next section, we compare the performance of FD versus CS using Matlab and show how using CS affects the solver performance.

Function	Complex form
$\sqrt{z}$	$\frac{\sqrt{2}}{2}(\sqrt{\sqrt{x^2+y^2}+x}+\boldsymbol{i}\mathrm{sgn}(y)\sqrt{\sqrt{x^2+y^2}-x})$
$\cos(z)$	$\cos(x)\cosh(y)-\boldsymbol{i}\sin(x)\sinh(y)$
$\sin(z)$	$\sin(x)\cosh(y)+\boldsymbol{i}\cos(x)\sinh(y)$
z^2	$x^2-y^2+2\boldsymbol{i}xy$
$z_1 \cdot z_2$	$\overline{z_1}\cdot z_2 = (x_1x_2+y_1y_2)+\boldsymbol{i}(x_1y_2-y_1x_2)$

Table C.1 – Complex formulation of a few classic functions with $z = x + \boldsymbol{i}y$

C.3 Comparison results

Considering the collision avoidance constraint of a single obstacle

$$-\|\boldsymbol{r}-\boldsymbol{r_{obs}}\|^2 + {R_{col}}^2 < 0 \tag{C.10}$$

Its gradient is

$$\nabla_{col} = -2\boldsymbol{r}(\boldsymbol{r}-\boldsymbol{r_{obs}}) \tag{C.11}$$

and can be seen in Fig. C.1

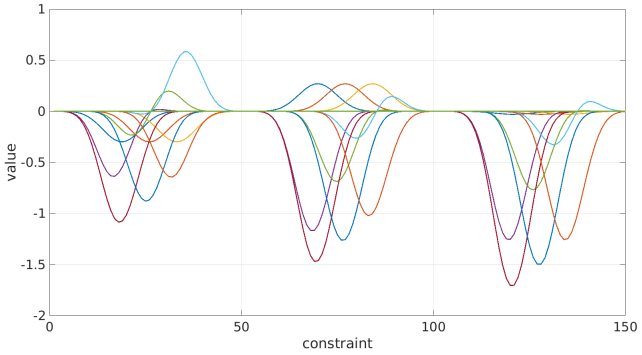


Figure C.1 – Analytic evaluation of the collision avoidance constraints gradient.

Figure C.2 shows that the relative error between the analytic gradient of (C.10) and its approximate with FD and CS is of the order of Matlab floating-point relative accuracy (2.2204e-16). Whereas the error with FD is larger to a factor of 1e10. Moreover, the precision of CS seemed not to be affected by different step values, which is not the case for FD.

Table C.2 shows the average time taken to approximate the nonlinear constraints gradient by FD and CS. Interestingly, CS seems to be about twice as fast as FD. This result shows the advantage of having a single evaluation of the considered function with CS instead of two for FD.

Now, from the above result, we can assume CS provides results very close to the analytic form. In Fig. C.3 we compare the absolute error between FD and CS

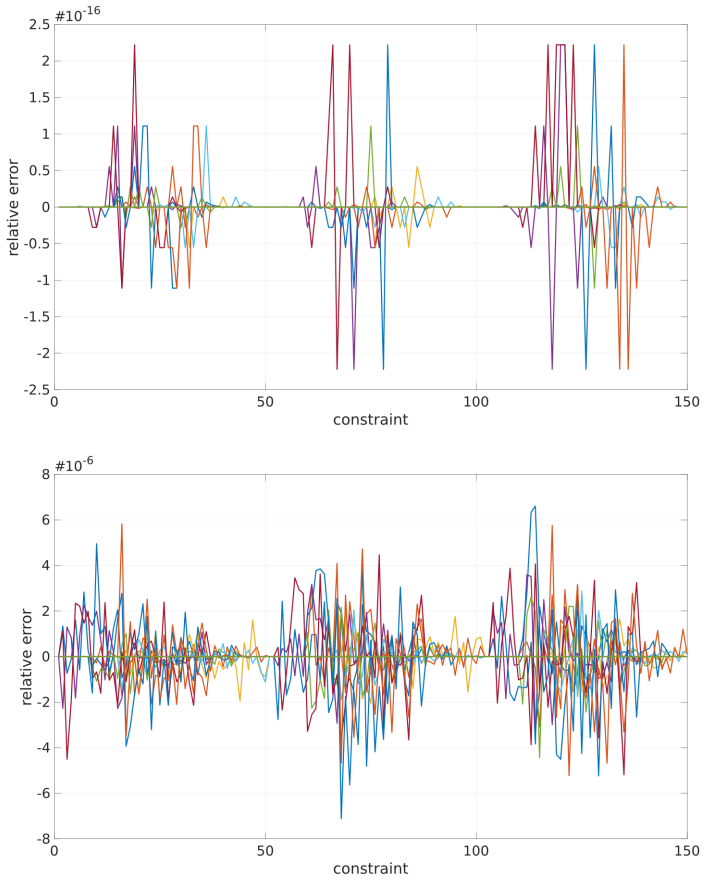


Figure C.2 – Relative error with CS and $h = 1e - 10$.

Method	Computation time
FD	$1.62e^{-2}$ s
CS	$0.84e^{-2}$ s

Table C.2 – Average computation time of the constraints gradient with FD and CS (with Matlab)

approximate of a single element of the occlusion constraints with respect to different step values h. As we can see, the error increases when h is too small ($h < 1e^{-12}$)

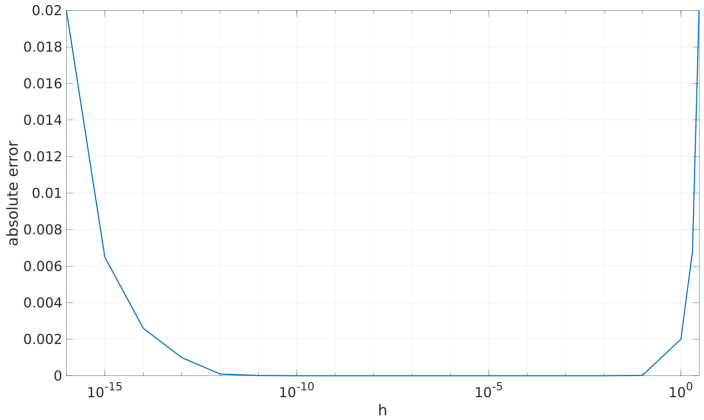


Figure C.3 – Absolute error with respect of step h.

and too large ($h > 1e^{-1}$) which is due to the typical term cancellation effect related to FD. Finally, the graph shows that a large range of values can give pretty good approximates.

Bibliography

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 2520–2525.

[2] B. Penin, R. Spica, P. Robuffo Giordano, and F. Chaumette, "Vision-based minimum-time trajectory generation for a quadrotor uav," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'17*, 2017.

[3] B. Penin, P. Robuffo Giordano, and F. Chaumette, "Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions," *IEEE Robotics and Automation Letters*, 2018.

[4] ——, "Minimum-time trajectory generation under intermittent measurements." IEEE, 2019. Submitted to RAL/ICRA'19.

[5] J. Thomas, M. Pope, G. Loianno, E. W. Hawkes, M. A. Estrada, H. Jiang, M. R. Cutkosky, and V. Kumar, "Aggressive flight with quadrotors for perching on inclined surfaces," *Journal of Mechanisms and Robotics*, vol. 8, no. 5, p. 051007, 2016.

[6] M. Hehn and R. D'Andrea, "Real-time trajectory generation for interception maneuvers with quadrocopters," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 4979–4984.

[7] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robot. Automat. Mag.*, vol. 17, no. 3, pp. 56–65, 2010.

[8] A. Chamseddine, T. Li, Y. Zhang, C. A. Rabbath, and D. Theilliol, "Flatness-based trajectory planning for a quadrotor unmanned aerial vehicle test-bed considering actuator and system constraints," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 920–925.

[9] R. Ritz, M. Hehn, S. Lupashin, and R. D'Andrea, "Quadrocopter performance benchmarking using optimal control," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 5179–5186.

[10] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010, pp. 1642–1648.

[11] D. Brescianini, M. Hehn, and R. D'Andrea, "Quadrocopter pole acrobatics," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 3472–3479.

[12] S. Tang, V. Wüest, and V. Kumar, "Aggressive flight with suspended payloads using vision-based control," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1152–1159, 2018.

[13] R. Ritz, M. W. Müller, M. Hehn, and R. D'Andrea, "Cooperative quadrocopter ball throwing and catching," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 4972–4978.

[14] M. Cutler and J. P. How, "Analysis and control of a variable-pitch quadrotor for agile flight," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 10, p. 101002, 2015.

[15] M. Ryll, H. H. Bülthoff, and P. R. Giordano, "A novel overactuated quadrotor unmanned aerial vehicle: Modeling, control, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 540–556, 2015.

[16] F. Kendoul, I. Fantoni, and R. Lozano, "Modeling and control of a small autonomous aircraft having two tilting rotors," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1297–1302, 2006.

[17] M. Ryll, D. Bicego, and A. Franchi, "Modeling and control of fast-hex: a fully-actuated by synchronized-tilting hexarotor," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[18] M. Zhao, T. Anzai, F. Shi, X. Chen, K. Okada, and M. Inaba, "Design, modeling, and control of an aerial robot dragon: A dual-rotor-embedded multilink robot with the ability of multi-degree-of-freedom aerial transformation," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1176–1183, 2018.

[19] Q. Delamare, P. R. Giordano, and A. Franchi, "Toward aerial physical locomotion: The contact-fly-contact problem," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1514–1521, 2018.

[20] K. Zhang, P. Chermprayong, T. Alhinai, R. Siddall, and M. Kovac, "Spidermav: Perching and stabilizing micro aerial vehicles with bio-inspired tensile anchoring systems," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6849–6854.

[21] K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Experiments in fast, autonomous, gps-denied quadrotor flight," *arXiv preprint arXiv:1806.07053*, 2018.

[22] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, "Low-latency visual odometry using event-based feature tracks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 16–23.

[23] I. Fantoni and R. Lozano, *Non-linear control for underactuated mechanical systems*. Springer Science & Business Media, 2002.

[24] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5774–5781.

[25] P. Foehn, D. Falanga, N. Kuppuswamy, R. Tedrake, and D. Scaramuzza, "Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload," in *Robotics: Science and Systems*, 2017, pp. 1–10.

[26] M. H. Dickinson, "Motor control: how dragonflies catch their prey," *Current Biology*, vol. 25, no. 6, pp. R232–R234, 2015.

[27] V. A. Tucker, A. E. Tucker, K. Akers, and J. H. Enderson, "Curved flight paths and sideways vision in peregrine falcons (falco peregrinus)," *Journal of Experimental Biology*, vol. 203, no. 24, pp. 3755–3763, 2000.

[28] F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *2006 IEEE Robotics & Automation Mag.*, vol. 13, no. 4, pp. 82–90, December 2006.

[29] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[30] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee, 2011, pp. 2688–2693.

[31] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1469–1475.

[32] M. W. Mueller and R. D'Andrea, "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 45–52.

[33] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[34] S. Bouabdallah and R. Y. Siegwart, "Full control of a quadrotor," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007: IROS 2007; Oct. 29, 2007-Nov. 2, 2007, San Diego, CA*. Ieee, 2007, pp. 153–158.

[35] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[36] M. W. Mueller and R. D'Andrea, "A model predictive controller for quadrocopter state interception," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 1383–1389.

[37] A. Mokhtari, N. K. M'Sirdi, K. Meghriche, and A. Belaidi, "Feedback linearization and linear observer for a quadrotor unmanned aerial vehicle," *Advanced Robotics*, vol. 20, no. 1, pp. 71–91, 2006.

[38] W. Jasim and D. Gu, "Integral backstepping controller for quadrotor path tracking," in *2015 International Conference on Advanced Robotics (ICAR)*. IEEE, 2015, pp. 593–598.

[39] G. Perozzi, D. Efimov, J.-M. Biannic, L. Planckaert, and P. Coton, "On sliding mode control design for uav using realistic aerodynamic coefficients," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 5403–5408.

[40] S. A. Al-Hiddabi, "Quadrotor control using feedback linearization with dynamic extension," in *2009 6th International Symposium on Mechatronics and its Applications (ISMA)*. IEEE, 2009, pp. 1–3.

[41] Z. Jia, J. Yu, Y. Mei, Y. Chen, Y. Shen, and X. Ai, "Integral backstepping sliding mode control for quadrotor helicopter under external uncertain disturbances," *Aerospace Science and Technology*, vol. 68, pp. 299–307, 2017.

[42] J. P. How, B. Behihke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE control systems*, vol. 28, no. 2, pp. 51–64, 2008.

[43] G. Hoffmann, S. Waslander, and C. Tomlin, "Quadrotor helicopter trajectory tracking control," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008, p. 7410.

[44] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, and J. Guichard, "Lqr control for a quadrotor using unit quaternions: Modeling and simulation," in *2013 International Conference on Electronics, Communications and Computing (CONIELECOMP)*. IEEE, 2013, pp. 172–178.

[45] G. Ganga and M. M. Dharmana, "Mpc controller for trajectory tracking control of quadcopter," in *2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. IEEE, 2017, pp. 1–6.

[46] H. Voos, "Nonlinear control of a quadrotor micro-uav using feedback-linearization," in *2009 IEEE International Conference on Mechatronics*. IEEE, 2009, pp. 1–6.

[47] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *None*, no. LSA-CONF-2005-003, 2005.

[48] S. Kumar and R. Gill, "Path following for quadrotors," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2017, pp. 2075–2081.

[49] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control," *Department of Mechanical and Process Engineering, ETHZ, Tech. Rep*, 2013.

[50] K. Sreenath and V. Kumar, "Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots," *rn*, vol. 1, no. r2, p. r3, 2013.

[51] T. Lee, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *2010 49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 5420–5425.

[52] P. Foehn and D. Scaramuzza, "Onboard State Dependent LQR for Agile Quadrotors," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6566–6572.

[53] H. J. Kim, D. H. Shim, and S. Sastry, "Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles," in *Proceedings of the 2002 American Control Conference*, vol. 5. IEEE, 2002, pp. 3576–3581.

[54] J. Yu, A. Jadbabaie, J. Primbs, and Y. Huang, "Comparison of nonlinear control design techniques on a model of the caltech ducted fan," *Automatica*, vol. 37, no. 12, pp. 1971–1978, 2001.

[55] A. Jadbabaie and J. Hauser, "On the stability of receding horizon control with a general terminal cost," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 674–678, 2005.

[56] P. Abbeel, *Apprenticeship learning and reinforcement learning with application to robotic control*. Stanford University, 2008.

[57] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[58] M. Hehn, R. Ritz, and R. D'Andrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, no. 1-2, pp. 69–88, 2012.

[59] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.

[60] J. Svacha, K. Mohta, and V. Kumar, "Improving quadrotor trajectory tracking by compensating for aerodynamic effects," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2017, pp. 860–866.

[61] J.-M. Kai, G. Allibert, M.-D. Hua, and T. Hamel, "Nonlinear feedback control of quadrotors exploiting first-order drag effects," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 8189–8195, 2017.

[62] M. Leahy and G. Saridis, "Compensation of unmodeled puma manipulator dynamics," in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4. IEEE, 1987, pp. 151–156.

[63] A. Piazzi and A. Visioli, "Global minimum-jerk trajectory planning of robot manipulators," *IEEE transactions on industrial electronics*, vol. 47, no. 1, pp. 140–149, 2000.

[64] T. Flash and N. Hogan, "The coordination of arm movements: an experimentally confirmed mathematical model," *Journal of neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

[65] M. J. Richardson and T. Flash, "Comparing smooth arm movements with the two-thirds power law and the related segmented-control hypothesis," *Journal of neuroscience*, vol. 22, no. 18, pp. 8201–8211, 2002.

[66] K. Bipin, V. Duggal, and K. M. Krishna, "Autonomous navigation of generic quadrocopter with minimum time trajectory planning and control," in *2014 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2014, pp. 66–71.

[67] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *Proc. of the 2013 IEEE International Conference on Robotics and Automation (ICRA)., Karlsruhe, Germany*, 2013.

[68] J. Yu, Z. Cai, and Y. Wang, "Minimum jerk trajectory generation of a quadrotor based on the differential flatness," in *2014 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*. IEEE, 2014, pp. 832–837.

[69] M. Hehn and R. D'Andrea, "Quadrocopter trajectory generation and control," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 1485–1491, 2011.

[70] K. J. Kyriakopoulos and G. N. Saridis, "Minimum jerk path generation," in *Proceedings 1988 IEEE International Conference on Robotics and Automation*. IEEE, 1988, pp. 364–369.

[71] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 2005, vol. 1.

[72] M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart, "Inversion based direct position control and trajectory following for micro aerial vehicles," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 2933–2939.

[73] J. Yu, Z. Cai, and Y. Wang, "Minimum jerk trajectory generation of a quadrotor based on the differential flatness," in *2014 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*. IEEE, 2014, pp. 832–837.

[74] F. Gao and S. Shen, "Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex qcqp," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 6354–6361.

[75] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 969–1002, 2015.

[76] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000.

[77] A. Howard, L. E. Parker, and G. S. Sukhatme, "Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 431–447, 2006.

[78] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[79] R. Spica, P. Robuffo Giordano, M. Ryll, H. Bülthoff, and A. Franchi, "An Open-Source Hardware/Software Architecture for Quadrotor UAVs," in *2nd Workshop on Research, Education and Development of Unmanned Aerial System*, Compiègne, France, Nov. 2013.

[80] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[81] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments," *Journal of Field Robotics*, vol. 28, no. 6, pp. 854–874, 2011.

[82] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 15–22.

[83] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor." in *Robotics: Science and Systems*, vol. 1. Citeseer, 2013.

[84] M.-D. Hua, N. Manerikar, T. Hamel, and C. Samson, "Attitude, linear velocity and depth estimation of a camera observing a planar target using continuous homography and inertial data," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1429–1435.

[85] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: introductory theory and examples," *International journal of control*, vol. 61, no. 6, pp. 1327–1361, 1995.

[86] P. Martin, R. M. Murray, and P. Rouchon, "Flat systems, equivalence and trajectory generation," 2003.

[87] M. J. Van Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 8, no. 11, pp. 995–1020, 1998.

[88] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *ASME international Mechanical Engineering Congress and Exposition*. Citeseer, 1995.

[89] M. Shomin and R. Hollis, "Differentially flat trajectory generation for a dynamically stable mobile robot," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 4467–4472.

[90] B. Yüksel, G. Buondonno, and A. Franchi, "Differential flatness and control of protocentric aerial manipulators with any number of arms and mixed rigid-/elastic-joints," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 561–566.

[91] M. Tognon and A. Franchi, "Dynamics, control, and estimation for aerial robots tethered by cables or bars," *IEEE Transactions on Robotics*, vol. 33, no. 4, pp. 834–845, 2017.

[92] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "A lie-backlund approach to equivalence and flatness of nonlinear systems," *IEEE Transactions on automatic control*, vol. 44, no. 5, pp. 922–937, 1999.

[93] B. Charlet, J. Lévine, and R. Marino, "On dynamic feedback linearization," *Systems & Control Letters*, vol. 13, no. 2, pp. 143–151, 1989.

[94] V. Chetverikov, "New flatness conditions for control systems," *IFAC Proceedings Volumes*, vol. 34, no. 6, pp. 191–196, 2001.

[95] J. Thomas, G. Loianno, J. Polin, K. Sreenath, and V. Kumar, "Toward Autonomous Avian-Inspired Grasping for Micro Aerial Vechicles," *Bioinspiration & Biomimetics*, vol. 9, no. 2, p. 025010, 2014.

[96] S. Formentin and M. Lovera, "Flatness-based control of a quadrotor helicopter via feedforward linearization," in *2011 50th IEEE Conference on Decision and*

Control and European Control Conference (CDC-ECC). IEEE, 2011, pp. 6171–6176.

[97] R. Rao, V. Kumar, and C. Taylor, "Visual Servoing of a UGV from a UAV using Differential Flatness," in *2003 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 1. IEEE, 2003, pp. 743–748.

[98] M. Tognon, S. S. Dash, and A. Franchi, "Observer-based control of position and tension for an aerial robot tethered to a moving platform," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 732–737, 2016.

[99] G. Allibert, E. Courtial, and F. Chaumette, "Predictive Control for Constrained Image-Based Visual Servoing," *IEEE Trans. on Robotics*, vol. 26, no. 5, pp. 933–939, 2010.

[100] J. De Doná, F. Suryawan, M. Seron, and J. Lévine, "A flatness-based iterative method for reference trajectory generation in constrained nmpc," in *Nonlinear Model Predictive Control*. Springer, 2009, pp. 325–333.

[101] M. Sheckells, G. Garimella, and M. Kobilarov, "Optimal Visual Servoing for Differentially Flat Underactuated Systems," in *2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2016.

[102] A. De Luca and G. Oriolo, "Trajectory planning and control for planar robots with passive last joint," *The International Journal of Robotics Research*, vol. 21, no. 5-6, pp. 575–590, 2002.

[103] N. Guenard and T. Hamel, "A Practical Visual Servo Control for an Unmanned Aerial Vehicle," *IEEE Trans. on Robotics*, vol. 24, no. 2, pp. 331–340, 2008.

[104] M. Odelga, P. Stegagno, and H. H. Bülthoff, "Obstacle detection, tracking and avoidance for a teleoperated uav," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2984–2990.

[105] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a vtol uav on a moving platform using image-based visual servoing," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 971–976.

[106] J. Thomas, J. Welde, G. Loianno, K. Daniilidis, and V. Kumar, "Autonomous flight for detection, localization, and tracking of moving targets with a small quadrotor," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1762–1769, 2017.

[107] H. Seo, S. Kim, and H. J. Kim, "Aerial grasping of cylindrical object using visual servoing based on stochastic model predictive control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 6362–6368.

[108] D. Kragic, H. I. Christensen *et al.*, "Survey on visual servoing for manipulation," *Computational Vision and Active Perception Laboratory, Fiskartorpsv*, vol. 15, p. 2002, 2002.

[109] A. Cretual and F. Chaumette, "Application of motion-based visual servoing to target tracking," *The International Journal of Robotics Research*, vol. 20, no. 11, pp. 878–890, 2001.

[110] R. Mahony and S. Stramigioli, "A port-Hamiltonian approach to image-based visual servo control for dynamic systems," *Int. J. of Robotics Research*, vol. 31, no. 11, pp. 1303–1319, 2012.

[111] E. Zergeroglu, D. M. Dawson, M. S. de Querioz, and A. Behal, "Vision-based nonlinear tracking controllers with uncertain robot-camera parameters," *IEEE/ASME Trans. on Mechatronics*, vol. 6, no. 3, pp. 322–337, Sep 2001.

[112] R. Spica, A. Franchi, G. Oriolo, H. H. Bülthoff, and P. Robuffo Giordano, "Aerial Grasping of a Moving Target with a Quadrotor UAV," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vilamoura, Portugal, Oct. 2012, pp. 4985–4992.

[113] H. Hermes and G. Haynes, "On the nonlinear control problem with control appearing linearly," *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, vol. 1, no. 2, pp. 85–108, 1963.

[114] W. Van Loock, G. Pipeleers, and J. Swevers, "Time-optimal quadrotor flight," in *European Control Conference (ECC), 2013*. IEEE, 2013, pp. 1788–1792.

[115] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[116] H. J. Pesch, "Real-time computation of feedback controls for constrained optimal control problems. part 1: Neighbouring extremals," *Optimal Control Applications and Methods*, vol. 10, no. 2, pp. 129–145, 1989.

[117] ——, "Real-time computation of feedback controls for constrained optimal control problems. part 2: A correction method based on multiple shooting," *Optimal Control Applications and Methods*, vol. 10, no. 2, pp. 147–171, 1989.

[118] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.

[119] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[120] E. Polak, *Optimization: algorithms and consistent approximations*. Springer Science & Business Media, 2012, vol. 124.

[121] M. Wright, "The interior-point revolution in optimization: history, recent developments, and lasting consequences," *Bulletin of the American mathematical society*, vol. 42, no. 1, pp. 39–56, 2005.

[122] P. E. Gill, W. Murray, and M. H. Wright, "Practical optimization," 1981.

[123] C. T. Lawrence and A. L. Tits, "Nonlinear equality constraints in feasible sequential quadratic programming," *Optimization Methods and Software*, vol. 6, no. 4, pp. 265–282, 1996.

[124] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms 1. general considerations," *IMA Journal of Applied Mathematics*, vol. 6, no. 1, pp. 76–90, 1970.

[125] M. B. Milam, "Real-time Optimal Trajectory Generation for Constrained Dynamical Systems," Ph.D. dissertation, California Institute of Technology, 2003.

[126] C. D. Boor, *A practical guide to splines*. Springer-Verlag New York, 1978, vol. 27.

[127] G. Antonelli, E. Cataldi, F. Arrichiello, P. R. Giordano, S. Chiaverini, and A. Franchi, "Adaptive trajectory tracking for quadrotor mavs in presence of parameter uncertainties and external disturbances," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 1, pp. 248–254, 2018.

[128] P. Robuffo Giordano, Q. Delamare, and A. Franchi, "Trajectory generation for minimum closed-loop state sensitivity," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[129] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[130] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[131] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[132] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based Motion Planning for Aggressive Flight in SE (3)," *arXiv preprint arXiv:1710.02748*, 2017.

[133] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 2452–2458.

[134] A. Raemaekers, "Design of a model predictive controller to control uavs," *Technische Universiteit Eindhoven*, 2007.

[135] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[136] R. M. Murray, "Optimization-based control," *California Institute of Technology, CA*, 2009.

[137] S. A. Homsi, A. Sherikov, D. Dimitrov, and P.-B. Wieber, "A Hierarchical Approach to Minimum Time Control of Industrial Robots," in *2016 IEEE Int. Conf. on Robotics and Automation*, Stockholm, Sweden, May 2016, pp. 16–21.

[138] R. Findeisen and F. Allgöwer, "An introduction to nonlinear model predictive control," in *21st Benelux Meeting on Systems and Control*, vol. 11. Technische Universiteit Eindhoven Veldhoven Eindhoven, The Netherlands, 2002, pp. 119–141.

[139] A. Boeuf, J. Cortés, R. Alami, and T. Siméon, "Planning agile motions for quadrotors in constrained environments," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2014, pp. 218–223.

[140] N. Faiz, S. Agrawal, and R. Murray, "Differentially flat systems with inequality constraints: An approach to real-time feasible trajectory generation," *Journal of Guidance, Control, and Dynamics*, vol. 24, no. 2, pp. 219–227, 2001.

[141] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[142] V. Grabe, M. Riedel, H. H. Bülthoff, P. R. Giordano, and A. Franchi, "The TeleKyb Framework for a Modular and Extendible ROS-based Quadrotor Control," in *2013 European Conference on Mobile Robots (ECMR)*. IEEE, 2013, pp. 19–25.

[143] T. Lee, M. Leokyand, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *49th IEEE Conf. on Decision and Control*, Atlanta, GA, Dec. 2010, pp. 5420–5425.

[144] M. Geisert and N. Mansard, "Trajectory generation for quadrotor based systems using numerical optimal control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2958–2964.

[145] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *2000 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.

[146] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[147] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2872–2879.

[148] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 42–49.

[149] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 477–483.

[150] S. Tang and V. Kumar, "Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2216–2222.

[151] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[152] P. M. Bouffard and S. L. Waslander, "A hybrid randomized/nonlinear programming technique for small aerial vehicle trajectory planning in 3d," *Planning, Perception and Navigation for Intelligent Vehicles (PPNIV)*, vol. 63, 2009.

[153] L. Bascetta, I. M. Arrieta, and M. Prandini, "Flat-rrt*: A sampling-based optimal trajectory planner for differentially flat vehicles with constrained dynamics," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6965–6970, 2017.

[154] R. Allen and M. Pavone, "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 1374.

[155] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps," in *WAFR 2016*, 2016.

[156] L. Van den Broeck, M. Diehl, and J. Swevers, "Model predictive control for time-optimal point-to-point motion control," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 2458–2463, 2011.

[157] S. Spedicato and G. Notarstefano, "Minimum-time trajectory generation for quadrotors in constrained environments," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1335–1344, 2018.

[158] H. Jabbari, G. Oriolo, and H. Bolandi, "Dynamic IBVS Control of an Underactuated UAV," in *2012 IEEE Int. Conf. on Robotics and Biomimetics*, Guangzhou, China, Dec. 2012, pp. 1158–1163.

[159] H. Jabbari and G. Oriolo and H. Bolandi, "An Adaptative Scheme for IBVS of an Underactuated UAV," *Int. J. of Robotics Research*, vol. 29, no. 1, pp. 92–104, 2014.

[160] J. Thomas, G. Loianno, J. Polin, K. Sreenath, and V. Kumar, "Toward autonomous avian-inspired grasping for micro aerial vehicles," *Bioinspiration & Biomimetics*, vol. 9, no. 2, p. 025010, 2014.

[161] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck, "Image-Based Visual Servo Control of the Translation Kinematics of a Quadrotor Aerial Vehicle," *IEEE Trans. on Robotics*, vol. 25, no. 3, pp. 743–749, 2009.

[162] F. Chaumette, "Image moments: a general and useful set of features for visual servoing," *IEEE Transactions on Robotics*, vol. 20, no. 4, pp. 713–723, 2004.

[163] G. Fink, H. Xie, A. F. Lynch, and M. Jagersand, "Nonlinear dynamic image-based visual servoing of a quadrotor," *Journal of unmanned vehicle systems*, vol. 3, no. 1, pp. 1–21, 2015.

[164] R. Mebarki, V. Lippiello, and B. Siciliano, "Nonlinear visual control of unmanned aerial vehicles in gps-denied environments," *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 1004–1017, 2015.

[165] T. Hamel and R. Mahony, "Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 2, pp. 187–198, 2002.

[166] ——, "Image based visual servo control for a class of aerial robotic systems," *Automatica*, vol. 43, no. 11, pp. 1975–1983, 2007.

[167] P. Serra, R. Cunha, T. Hamel, D. Cabecinhas, and C. Silvestre, "Landing on a moving target using image-based visual servo control," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 2179–2184.

[168] R. Ozawa and F. Chaumette, "Dynamic visual servoing with image moments for a quadrotor using a virtual spring approach," in *2011 IEEE Int. Conf. on Robotics and Automation*, 2011, pp. 5670–5676.

[169] R. Fleurmond and V. Cadenat, "Handling visual features losses during a coordinated vision-based task with a dual-arm robotic system," in *2016 European Control Conference (ECC)*. IEEE, 2016, pp. 684–689.

[170] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," in *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics, Shanghai, China*, 2017, pp. 11–13.

[171] C. Teuliere, L. Eck, and E. Marchand, "Chasing a moving target from a flying uav," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 4929–4934.

[172] N. J. Cowan, J. D. Weingarten, and D. E. Koditschek, "Visual servoing via navigation functions," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 521–533, 2002.

[173] Y. Mezouar and F. Chaumette, "Path planning for robust image-based control," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 534–549, 2002.

[174] E. Marchand and G. D. Hager, "Dynamic sensor planning in visual servoing," in *1998 IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, 1998, pp. 1988–1993.

[175] N. Mansard and F. Chaumette, "A new redundancy formalism for avoidance in visual servoing," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 468–474.

[176] D. Folio and V. Cadenat, "A controller to avoid both occlusions and obstacles during a vision-based navigation task in a cluttered environment," in *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05.* IEEE, 2005, pp. 3898–3903.

[177] G. Allibert, E. Courtial, and Y. Touré, "A flat model predictive controller for trajectory tracking in image based visual servoing," *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 993–998, 2007.

[178] R. Rao, V. Kumar, and C. Taylor, "Visual servoing of a ugv from a uav using differential flatness," in *Proceedings. 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1. IEEE, 2003, pp. 743–748.

[179] D. J. Agravante, G. Claudio, F. Spindler, and F. Chaumette, "Visual servoing in an optimization framework for the whole-body control of humanoid robots," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 608–615, 2017.

[180] D. Nicolis, M. Palumbo, A. M. Zanchettin, and P. Rocco, "Occlusion-free visual servoing for the shared autonomy teleoperation of dual-arm robots," *IEEE Robotics and Automation Letters*, 2018.

[181] M. Kazemi, K. K. Gupta, and M. Mehrandezh, "Randomized kinodynamic planning for robust visual servoing," *IEEE Transactions on Robotics*, vol. 29, no. 5, pp. 1197–1211, 2013.

[182] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: Perception-aware model predictive control for quadrotors," *arXiv preprint arXiv:1804.04811*, 2018.

[183] W. Ding, M. R. Ganesh, R. N. Severinghaus, J. J. Corso, and D. Panagou, "Real-time model predictive control for keeping a quadrotor visible on the camera field-of-view of a ground robot," in *2016 American control conference (ACC)*, 2016, pp. 2259–2264.

[184] J. Chen, T. Liu, and S. Shen, "Tracking a moving target in cluttered environments using a quadrotor," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 446–453.

[185] G. Costante, C. Forster, J. Delmerico, P. Valigi, and D. Scaramuzza, "Perception-aware path planning," *arXiv preprint arXiv:1605.04151*, 2016.

[186] R. Pepy, M. Kieffer, and E. Walter, "Reliable robust path planning with application to mobile robots," *International Journal of Applied Mathematics and Computer Science*, vol. 19, no. 3, pp. 413–424, 2009.

[187] B. Charrow, S. Liu, V. Kumar, and N. Michael, "Information-theoretic mapping using cauchy-schwarz quadratic mutual information," in *2015 IEEE Int. Conf. on Robotics and Automation*, May 2015, pp. 4791–4798.

[188] L. Heng, A. Gotovos, A. Krause, and M. Pollefeys, "Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments," in *2015 IEEE Int. Conf. on Robotics and Automation*, May 2015, pp. 1071–1078.

[189] M. W. Achtelik, S. Lynen, S. Weiss, M. Chli, and R. Siegwart, "Motion- and uncertainty-aware path planning for micro aerial vehicles," *Journal of Field Robotics*, vol. 31, no. 4, pp. 676–698, 2014.

[190] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular Vision for Long-term Micro Aerial Vehicle State Estimation: A Compendium," *J. of Field Robotics*, vol. 30, no. 5, pp. 803–831, 2013.

[191] H. Michalska and D. Q. Mayne, "Robust receding horizon control of constrained nonlinear systems," *IEEE transactions on automatic control*, vol. 38, no. 11, pp. 1623–1633, 1993.

[192] N. M. Patrikalakis and T. Maekawa, *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer, 2009.

[193] S. G. Johnson, "The nlopt nonlinear-optimization package." http://ab-initio.mit.edu/nlopt.

[194] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[195] R. T. Fomena and F. Chaumette, "Improvements on visual servoing from spherical targets using a spherical projection model," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 874–886, 2009.

[196] R. Spica, P. R. Giordano, and F. Chaumette, "Active structure from motion for spherical and cylindrical targets," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5434–5440.

[197] W. Squire and G. Trapp, "Using complex variables to estimate derivatives of real functions," *Siam Review*, vol. 40, no. 1, pp. 110–112, 1998.

[198] R. Abreu, D. Stich, and J. Morales, "On the generalization of the complex step method," *Journal of Computational and Applied Mathematics*, vol. 241, pp. 84–102, 2013.

[199] J. Martins, P. Sturdza, and J. Alonso, "The connection between the complex-step derivative approximation and algorithmic differentiation," in *39th Aerospace Sciences Meeting and Exhibit*, 2001, p. 921.

[200] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[201] M. Rafieisakhaei, S. Chakravorty, and P. Kumar, "Belief space planning simplified: Trajectory-optimized lqg (t-lqg)," *arXiv preprint arXiv:1608.03013*, 2016.

[202] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 723–730.

[203] J. P. Gonzalez and A. Stentz, "Planning with uncertainty in position an optimal and efficient planner," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 2435–2442.

[204] B. Davis, I. Karamouzas, and S. J. Guy, "C-opt: Coverage-aware trajectory optimization under uncertainty," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1020–1027, 2016.

[205] J. Van Den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using differential dynamic programming in belief space," in *Robotics Research*. Springer, 2017, pp. 473–490.

[206] A. Lambert and D. Gruyer, "Safe path planning in an uncertain-configuration space," in *2003 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2003, pp. 4185–4190.

[207] Z. Zhang and D. Scaramuzza, "Perception-aware receding horizon navigation for mavs," Tech. Rep., 2018.

[208] S. Candido and S. Hutchinson, "Minimum uncertainty robot navigation using information-guided pomdp planning," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 6102–6108.

[209] K. Hausman, J. Preiss, G. S. Sukhatme, and S. Weiss, "Observability-aware trajectory optimization for self-calibration with application to uavs," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1770–1777, 2017.

[210] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *International Journal of Control*, vol. 61, no. 6, pp. 1327–1361, 1995.

[211] C. P. Tang, "Differential flatness-based kinematic and dynamic control of a differentially driven wheeled mobile robot," in *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2009, pp. 2267–2272.

[212] G. G. Rigatos, "Derivative-free kalman filtering for autonomous navigation of unmanned ground vehicles," in *2012 1st International Conference on Systems and Computer Science (ICSCS)*. IEEE, 2012, pp. 1–6.

[213] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[214] W. Van Loock, G. Pipeleers, and J. Swevers, "B-spline parameterized optimal motion trajectories for robotic systems with guaranteed constraint satisfaction," *Mechanical Sciences*, vol. 6, no. 2, p. 163, 2015.

[215] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Transactions on Control Systems Technology*, 2017.

[216] H. Prautzsch, W. Boehm, and M. Paluszny, *Bézier and B-spline techniques*. Springer Science & Business Media, 2013.

[217] C. Louembet, F. Cazaurang, and A. Zolghadri, "Motion planning for flat systems using positive b-splines: An lmi approach," *Automatica*, vol. 46, no. 8, pp. 1305–1309, 2010.

[218] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[219] E. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics and Automation Magazine*, vol. 12, no. 4, pp. 40–52, December 2005.

[220] S. Ramasamy, G. Wu, and K. Sreenath, "Dynamically feasible motion planning through partial differential flatness." in *Robotics: Science and Systems*, 2014.

[221] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-time collision avoidance with whole body motion control for humanoid robots," in *2007*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2007, pp. 2053–2058.

[222] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3948–3955.

[223] N. Crombez, E. M. Mouaddib, G. Caron, and F. Chaumette, "Visual servoing with photometric gaussian mixtures as dense feature," *IEEE Transactions on Robotics*, 2018.

[224] F. Suryawan, J. De Doná, and M. Seron, "On splines and polynomial tools for constrained motion planning," in *2010 18th Mediterranean Conference on Control & Automation (MED)*. IEEE, 2010, pp. 939–944.

[225] L. Biagiotti and C. Melchiorri, *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008.

[226] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 2012.

[227] J. N. Lyness, "Numerical algorithms based on the theory of complex variable," in *Proceedings of the 1967 22nd national conference*. ACM, 1967, pp. 125–133.

[228] J. Martins, I. Kroo, and J. Alonso, "An automated method for sensitivity analysis using complex variables," in *38th aerospace sciences meeting and exhibit*, 2000, p. 689.

[229] K.-L. Lai and J. Crassidis, "Extensions of the first and second complex-step derivative approximations," *Journal of Computational and Applied Mathematics*, vol. 219, no. 1, pp. 276–293, 2008.

[230] R. Bagley, "On fourier differentiation-a numerical tool for implicit functions," *International Journal of Applied Mathematics*, vol. 19, no. 3, p. 255, 2006.

[231] L. B. Rall and G. F. Corliss, "An introduction to automatic differentiation," *Computational Differentiation: Techniques, Applications, and Tools*, vol. 89, 1996.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Contributions à la génération de trajectoires optimales et réactives basées vision pour un quadrirotor UAV

Mots clés : contraintes visuelles, optimisation nonlinéaire, navigation aérienne agile, quadrirotor

Résumé : La vision représente un des plus importants signaux en robotique. Une caméra monoculaire peut fournir de riches informations visuelles à une fréquence raisonnable pouvant être utilisées pour la commande, l'estimation d'état ou la navigation dans des environnements inconnus par exemple. Il est cependant nécessaire de respecter des contraintes visuelles spécifiques telles que la visibilité de mesures images et les occultations durant le mouvement afin de garder certaines cibles visuelles dans le champ de vision. Les quadrirotors sont dotés de capacités de mouvement très réactives du fait de leur structure compacte et de la configuration des moteurs. De plus, la vision par une caméra embarquée (fixe) va subir des rotations dues au sous-actionnement du système. Dans cette thèse nous voulons bénéficier de l'agilité du quadrirotor pour réaliser plusieurs tâches de navigation basées vision. Nous supposons que l'estimation d'état repose uniquement sur la fusion capteurs d'une centrale inertielle (IMU) et d'une caméra monoculaire qui fournit des estimations de pose précises. Les contraintes visuelles sont donc critiques et difficiles dans un tel contexte. Dans cette thèse nous exploitons l'optimisation numérique pour générer des trajectoires faisables satisfaisant un certain nombre de contraintes d'état, d'entrées et visuelles nonlinéaires. A l'aide la *platitude différentielle* et de la paramétrisation par des B-splines nous proposons une stratégie de replanification performante inspirée de la commande prédictive pour générer des trajectoires *lisses* et agiles. Enfin, nous présentons un algorithme de planification en temps minimum qui supporte des pertes de visibilité intermittentes afin de naviguer dans des environnements encombrés plus vastes. Cette contribution porte l'incertitude de l'estimation d'état au niveau de la planification pour produire des trajectoires robustes et sûres. Les développements théoriques discutés dans cette thèse sont corroborés par des simulations et expériences en utilisant un quadrirotor. Les résultats reportés montrent l'efficacité des techniques proposées.

Title : Contributions to optimal and reactive vision-based trajectory generation for a quadrotor UAV

Keywords : visibility constraints, nonlinear optimization, agile aerial navigation, quadrotor UAV

Abstract : Vision constitutes one of the most important cues in robotics. A single monocular camera can provide rich visual information at a reasonable rate that can be used as a feedback for control, state estimation of mobile robots or safe navigation in unknown environments for instance. However, it is necessary to satisfy particular visual constraints on the image such as visibility and occlusion constraints during motion to keep some visual targets visible. Quadrotors are endowed with very reactive motion capabilities due to their compact structure and motor configuration. Moreover, vision from a (fixed) on-board camera will suffer from rotation motions due to the system underactuation. In this thesis, we want to benefit from the system aggressiveness to perform several vision-based navigation tasks. We assume state estimation relies solely on sensor fusion of an onboard inertial measurement unit (IMU) and a monocular camera that provides reliable pose estimates. Therefore, visual constraints are challenging and critical in this context. In this thesis we exploit numerical optimization to design feasible trajectories satisfying several state, input and visual nonlinear constraints. With the help of *differential flatness* and B-spline parametrization we will propose an efficient replanning strategy inspired form Model Predictive Control to generate *smooth* and agile trajectories. Finally, we propose a minimum-time planning algorithm that handles intermittent visibility losses in order to navigate in larger cluttered environments. This contribution brings state estimation uncertainty at the planning stage to produce robust and safe trajectories. All the theoretical developments discussed in this thesis are corroborated by simulations and experiments run by using a quadrotor UAV. The reported results show the effectiveness of proposed techniques.