SAPIENZA
UNIVERSITÀ DI ROMA

Navigation of Quadrotor UAVs among Obstacles with Formal Safety Guarantees

Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Intelligenza Artificiale e Robotica

Candidate

Lorenzo D'Auria

ID number 1918917

Thesis Advisor

Prof. Marilena Vendittelli

Co-Advisor

Prof. Andrea Cristofaro

Academic Year 2020/2021

Navigation of Quadrotor UAVs among Obstacles with Formal Safety Guarantees
Master's thesis. Sapienza – University of Rome

© 2021 Lorenzo D'Auria. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: dauria.1918917@studenti.uniroma1.it

To the memory of Grant M. Imahara,
whose passion for robotics has inspired me since childhood

Abstract

The goal of this thesis is to develop a safe navigation algorithm for UAVs moving in an environment with obstacles. The method is divided in two main phases, the first is done offline, while the second is performed online. We start by considering a generic 3D path planning problem in which we want to compute a simple geometric safe path connecting the robot initial position to a goal point. Using a fast probabilistic approach we obtain a path, which is in general unfeasible due to the underactuated nature of UAVs. In the second phase, which is done online, the initial trajectory obtained in the previous phase is used as reference for an impedance-based controller. The key factor here is that the control considers also a virtual external force, assumed to be applied on the UAV, which pushes the robot away from the obstacles. This force is computed from the artificial potential fields generated by the obstacles detected around during the navigation. This allows to handle uncertainties in the obstacles layout while ensuring the robot safety. With this method we ensure a fast planning phase, since the actual path planning problem is trivial, together with a safe navigation in which the trajectory followed by the robot is smoothed with respect to the reference in accordance with the obstacles detected around and the system dynamics. With this approach we expect to perform well in many difficult conditions, such as when there are unexpected obstacles in the environment. In our implementation we focused on a quadrotor instead of a generic UAV, but the method is amenable to extension to other types of aircraft. Due to economical and practical reasons we tested our method only using PC simulations where there are not uncertainties in the quadrotor dynamics or in the obstacles detection. Anyway, looking at the simulation results, this method showed to work properly in many useful situations.

Contents

List of Figures

Chapter 1

Introduction

During the last decade the interest of companies and researchers in unmanned aerial vehicles (UAVs) has risen significantly. One of the most obvious reasons of this is the availability and the cost of the components required to develop this type of robots. Nowadays small and light electric motors can be easily found and the same goes for all the others components, such as the electronic speed controllers. Also the evolution of the batteries has contributed to this trend, they are light and powerful enough to keep these vehicles in flight for several minutes up to an hour. In particular, a type of UAV which has become very popular is the quadrotor, an helicopter-like robot with four rotors directed along the vertical axis. A pair of non adjacent rotors spins in the clockwise direction, while the other two spins counterclockwise, in this way, in addition to the altitude, the vehicle's attitude can be modified by changing the speeds of the rotors.

Following this trend, many companies started developing and selling a lot of products of this type. The fields of applications are a lot and goes from military to agricultural, going through security and the entertainment industry. With vehicles

Figure 1.1. Example of a popular quadrotor available on the market. Produced by DJI [4].

of this kind it is possible to explore environments that are hard or even impossible to be reached by other types of manned aerial vehicles. For example flying with a manned helicopter above crowded places at low altitude or between buildings it is dangerous for the pilot and the people around, while the use of an UAV is a safer and cheaper solution.

Going deeper in this analysis, the UAVs potentialities are exploited at best when the human pilot's commands are coordinated or substituted with a dedicated control software. In many quadrotors available on the market the pilot does not control directly the rotors speeds, since it would be almost impossible to fly stably. The user can only modify the desired attitude and altitude, then, through an attitude controller [3], the motor commands are computed so to reach and stabilize the aircraft in the desired configuration. On the other hand, there are some situations in which having a fully autonomous robot is preferable, for example when there is a significant delay between the the pilot command and the UAV actuation. In this cases the control strategy needs to be way more complex and may need additional sensors to estimate the vehicle position in the environment. The general strategy used in this situations is a hierarchical approach [6, 12] and consists in computing or receiving as input a desired trajectory to follow, then, using a closed loop position controller, the attitude required to track the given trajectory is computed. To conclude, the desired configuration is used as reference for the attitude controller which will compute the rotors commands that has to be applied.

Even if the position control of a quadrotor can be achieved without particular difficulties, a big problem remains in the preliminary phase of trajectory planning. If we want to have a fully autonomous robot we have to compute automatically also the trajectory that has to be followed. For example we may be interested in flying from a point of the environment to another, without specifying the trajectory in details. The computation of a trajectory of this kind can be done with many known planning algorithms as long as the environment is well known and there are not obstacles around. On the contrary, when we have to deal with an UAV moving in an environment populated with obstacles, the difficulty in developing a trajectory planning algorithm increases. In this case the trajectory chosen has to be safe, i.e. has to keep the robot far from the obstacles. Obtaining a geometric safe path in a known environment is not difficult, many path planning methods work fine, the problem is that the trajectory generated has to be also feasible for the UAV. Since most UAVs, and in particular quadcopters, are underactuated, not all path in configuration space are feasible. Unconstrained geometric trajectories may contain sharp turns, so the path needs to be smoothed [18], a process that can alter the safeness of the trajectory. A common problem that can arise during this process is that around corners the smoothed trajectory goes through an obstacle. In this case the only thing that can be done is that, when this problem is detected, the algorithm starts again from another safe geometric path, but there is no guarantee that sooner or later a safe feasible path will be found.

The method proposed in this thesis stems exactly from this limitation, we are interested in developing an autonomous navigation system that includes a fast path planning phase together with a trajectory tracking method that ensures the safety of the vehicle. To simplify this analysis we developed the algorithm only for quadrotors, but the extension of this method to other types of UAVs is straightforward. Another

important assumption that we made is that the robot configuration is always well known and not affected by error, same goes for the detection of the obstacles around the vehicle, in this way it will be possible to evaluate the effectiveness of the algorithm in ideal conditions. With the addition of other more strict hypotheses and using a control barrier function, we will make some considerations about the guarantee of flying within a safe distance from the initial trajectory.

Going more in the details of this method, we use artificial potential fields to compute a virtual repulsive force that pushes the robot away from the obstacles. So, given a generic safe path, the quadcopter simply tries to follow the path using a position controller, when one or more near obstacles are detected, the virtual external force is computed. To modify the UAV motion based on this force, we compute the commands using an impedance controller [26], instead of a standard position control, in which we take into account the repulsive force that pushes the UAV away from obstacles. Thanks to this method we make a simple and fast planning of a geometric, not necessarily feasible, trajectory that guides the UAV through the obstacles. Then, with an impedance-based controller, we track the reference trajectory as close as possible, accordingly to the UAV dynamics, while executing obstacle avoidance in reactive fashion.

Since a real prototype was not available, we use computer simulation to validate the approach. In this way we are able to work on an ideal environment with all the assumptions mentioned before. The framework chosen is CoppeliaSim, a simulation software specific for robotics experiments which allowed to setup an ad hoc scene for our tests. To handle in a simpler way the matrix operations, some MATLAB scripts have been developed to be used together with the CoppeliaSim code through the dedicated remote API functions. Various different scene configurations have been considered during the evaluation, the scope was to test the method's performance when dealing with different obstacles configuration and parameters settings.

In the following chapters all the necessary steps done to develop these algorithm will be described in depth. In particular, in Chapter 2 we will review the relevant

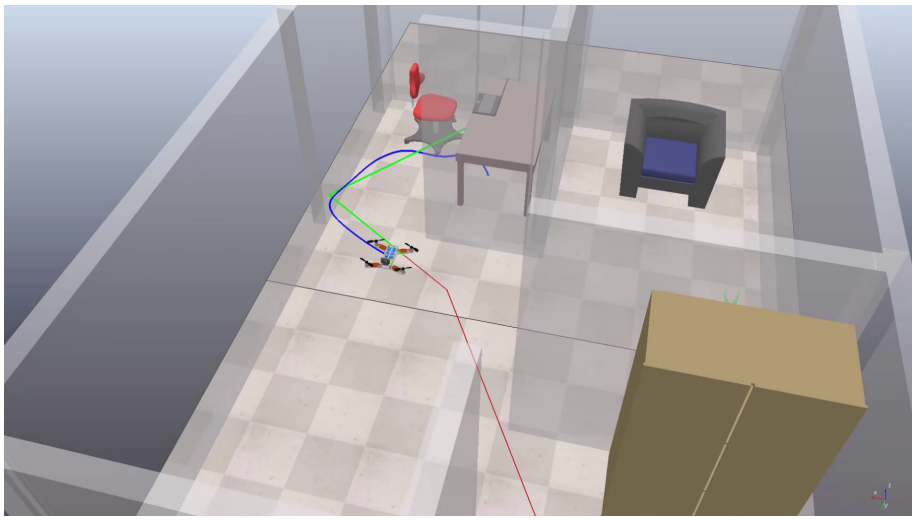

Figure 1.2. Rendering of a CoppeliaSim scene developed for validation.

literature on path and trajectory planning, reactive navigation, trajectory tracking and interaction control, focusing more on the concepts that have been used in this work. In Chapter 3 will be provided the mathematical model of the quadrotor dynamics together with the main control techniques that can be used to stabilize it. The main advantages and limitations will be highlighted. Then, in Chapter 4, the general path planning problem will be described. Some well known approaches to solve this problem will be reported and compared to give an overview of the typical resolution strategies. Moving on to Chapter 5, the specific problem taken into account in this thesis will be formalized and the proposed approach will be presented. Some safety considerations will also be reported. All the details about the simulations and their results can be found in Chapter 6. At the end, in Chapter 7, more general conclusions on the entire method developed and some future prospective are listed.

Chapter 2

Related work

In this chapter we give an overview of the literature on the topics relevant for this thesis. The analysis is divided in three main sections: path and trajectory planning in environments with obstacles, reactive navigation and trajectory tracking strategies, interaction control in presence of external disturbances.

2.1 Path and trajectory planning

Path and trajectory planning are two well studied topics in literature, the first consist in computing a sequence $\boldsymbol{q}(s)$ of safe configurations from a point to another of the environment, while the second returns a time-dependent sequence $\boldsymbol{q}(t)$. If the system dynamics allows it, a trajectory planning problem can be solved starting from a path planning problem to which a timing law is then added, this workaround is particularly useful because simplifies the computations. One generic and reliable path planning algorithm known in the literature is the one developed by Ó'Dúnlaing and Yap [17]. In this algorithm the generation of a roadmap based on the Voronoi diagram allows to solve a simple graph pathfinding problem with the guarantee of maximizing the distance from the obstacles. Another similar strategy is [23] by Sleumer and Tschichold-Gürmann, in this case a connectivity graph is obtained from the cell decomposition of the configuration space and then, as before, it is solved using a pathfining algorithm. In both these methods the main limitation is that the time complexity raises rapidly when dealing with high dimensional configuration spaces or with environments containing many complex obstacles.

An alternative category of algorithms is the one based on a probabilistic approach, which in general is faster. The well known Rapidly-exploring Random Trees (RRT) by LaValle et al. [11] is one the most famous representative of this category. This types of algorithm are only probabilistically complete, but, if a safe path can be found, the execution time will be short. Since in our method the goal is to have a fast offline path planning, the probabilistic-based method are the most suitable. In particular the algorithm chosen in our implementation is RRT-Connect, see Kuffner and LaValle [10], which is an heuristic-based variant of the RRT algorithm.

The path planning algorithm seen so far are very general, but they may have the disadvantage of generating unfeasible trajectories when dealing with underactuated robots, such as UAVs. If we want to ensure the feasibility of the trajectory we need

to use more sophisticated methods. One solution suitable for UAVs is based on the use of polynomials path segments optimization, see Richter, Bry, and Roy [19]. With this approach the computational time is way less than the one required for a purely sampling-based optimal kinodynamic method, while the trajectory feasibility is preserved. On the other hand it scarifies the guarantee of asymptotic convergence to the global optimum.

Another relevant solution is the one proposed by Liu et al. [14], in this case the use of a set of motion primitives, generated by solving an optimal control problem, has made it possible to compute safe and dynamically feasible trajectories. This method proved to be suitable also in case of fast online re-planning while the robot is moving. With a similar idea works also the method proposed by Zhou et al. [29]. In this case the motion primitives are taken in a discretized control space to obtain a safe and minimum time trajectory, then, using B-spline optimization, an additional smoothing is performed. The problem of trajectory planning has also been studied for multiple aerial robots moving in the same environment. For example Hönig et al. [8] proposed a three phase method in which is done a smoothing of the geometric trajectories generated from a roadmap.

In case of more complex scenarios, for example environments containing obstacles in motion, reduced field of view for the robot or motion uncertainties in general, some more sophisticated method are available. In [15] Liu et al. describe a search-based motion planning framework that uses motion primitives to obtain a dynamically feasible, collision-free and optimal trajectory in environments of this kind. The use of LPA* and graph pruning are at the core of this algorithm which is also suitable for planning motions of multiple UAVs. A different complex scenario has been considered instead in [13] by Liu et al. where is presented a method for planning aggressive trajectories that take into account narrow passages smaller than the quadrotor diameter. In this case a lower dimensional search is used as an heuristic to compute the final dynamically feasible trajectory through the generation of motion primitives.

2.2 Reactive navigation and trajectory tracking

A different way to achieve safe navigation is through reactive navigation. This approach does not require the computation of a safe trajectory since it works by simply reacting to the environment. The method introduced by Khatib in [9] falls into this category, it uses artificial potential fields to drive the robot motion towards the goal and away from the obstacles. This strategy has the big advantage of being capable to operate fully online using the information collected by the on-board sensors. A notable use of artificial potential fields in the context of UAVs was made by Zhou and Schwager [30], in this case they are used to compute a desired velocity vector that drives the robot in a desired position. The usage of reactive navigation to achieve obstacle avoidance can be found also in [16] by Muratet, Doncieux, and Meyer. In this work, inspired by the behavior of flying insects, the optical flow provided by a camera is used to control a rotary-wing UAV in a reactive fashion so to obtain a safe navigation in a urban environment.

On the contrary, if we want to follow a given trajectory, we have to use a trajectory

tracking algorithm, another deeply studied topic. The goal of trajectory tracking is to stabilize the robot movements along a given trajectory, possibly reducing the control effort. In the context of UAVs, and in particular for quadrotors, many well known tracking methods are available, first of all a simple cascade PID controller as described by Hoffmann, Waslander, and Tomlin in [6]. Over the years more sophisticated approaches were developed, a notable one is the controller in SE(3) developed by Lee, Leok, and McClamroch [12]. With this type of controller the small angles assumption, necessary in PID control, is not required and so the UAV can be stabilized starting from any attitude.

As anticipated our method is halfway between the reactive navigation and the trajectory tacking since we want to track a safe reference trajectory while being ready to react to potential contacts. To prove that our method guarantees a limited tracking error from the original geometric safe path we will use barrier functions, a powerful tool described by Ames et al. in [1]. In our case the safe set considered is a tube around the planned path. Barrier functions have been already used to prove the safety of UAV maneuvers, as can be seen in [28]. In this work the authors solve an optimization problem based on the safety barrier certificates to modify an initial trajectory computed ignoring collisions. The main differences with respect to this work is that in our case the barrier functions guided the development of the virtual repulsive force. This force is then used as an auxiliary control input in a reactive fashion and not inside an optimization problem. This simple solution speeds up the offline planning phase while being more flexible to changes of the environment during the online phase.

2.3 Interaction control

The implementation of interaction control on UAVs is a quite new field of research, it consist in controlling the aircraft in presence of external forces that can arise from collisions, wind disturbances or desired contacts. This type of control was initially developed for manipulators that has to be compliant with the environment, see [7]. The adaptation of interaction control to UAVs is quite straightforward and can be done with an admittance or an impedance controller as described in [26]. One of the first attempts to control an UAV in contact with the environment was made by Fumagalli et al. [5]. In this case the goal was to inspect the environment using a quadrotor equipped with a small manipulator touching the obstacles while being compliant to the contact forces. A similar task has been considered also by Ryll et al. [21], in this case the UAV is a Tilt-Hex and not a quadrotor, while the stability when dealing with contacts is guaranteed by an admittance controller, already adapted for UAVs in 2013 by Augugliaro and D'Andrea [2]. Very relevant for this thesis is also the work of Tognon, Alami, and Siciliano [24] where is described a control strategy based on admittance control to track a trajectory while a rope applies an external force to the UAV. To conclude, a method that uses interaction control in a way similar to what is done in this thesis is [20], by Ruggiero et al. In this case the goal was to control an UAV in presence of external force, estimated online, while being compliant to uncertainties in the vehicle dynamics.

Another important aspect that has to be considered when dealing with interac-

tions is the estimation of the external forces. A detailed analysis of this problem was done by Tomić and Haddadin [26] and showed that, using only proprioceptive sensors and the robot's dynamics, it is possible to obtain a good estimation of the external wrenches acting on an UAV. Further analysis on this topic, which anyway is not very relevant for this thesis, can be found in [27, 25]. The core idea consist in monitoring a residual vector expressing the differences between the dynamics predicted by the robot model and the one detected by the on-board sensors. It follows that the more accurate the dynamic model is, the more precise the external force estimation will be.

Chapter 3

Modeling and control

Before going deep in the details of the method developed, it is better to start by introducing the theory behind the UAVs control. To setup an effective control strategy we have to start by considering the dynamic model and the motion equations of an UAV, which obviously depend on the particular type of aerial vehicle we want to use. As anticipated in the previous chapters, in this thesis we evaluate our algorithm on a quadrotor and for this reason we will focus on modeling this type of aircraft. Then, using the model of the system, we will be able to present in details the theory behind some of the most common control strategies to achieve different tasks such as trajectory tracking.

3.1 Quadrotor mathematical model

The first important aspect that we will investigate is the dynamic model of an UAV, which describes the evolution of the robot state given the current state and the applied wrench. The dynamic model of a generic vertical take off and landing

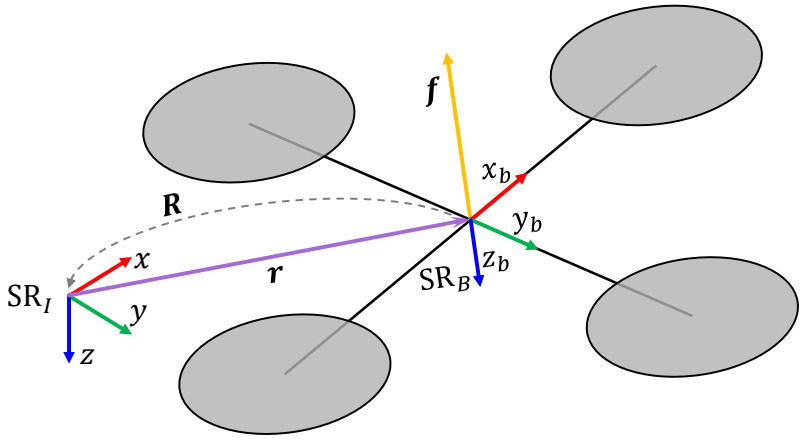

Figure 3.1. Quadrotor model graphical representation.

(VTOL) UAV can be described as

$$\begin{aligned} \mathcal{M}\ddot{\boldsymbol{r}} &= \mathcal{M}g\boldsymbol{e}_3 + \boldsymbol{R}\boldsymbol{f} + \boldsymbol{f}_e \\ \boldsymbol{\mathcal{I}}\dot{\boldsymbol{\omega}} &= -\mathcal{S}(\boldsymbol{\mathcal{I}}\boldsymbol{\omega})\boldsymbol{\omega} + \boldsymbol{m} + \boldsymbol{R}^{\mathsf{T}}\boldsymbol{m}_e \\ \dot{\boldsymbol{R}} &= \boldsymbol{R}\mathcal{S}(\boldsymbol{\omega}), \end{aligned} \tag{3.1}$$

where $\mathcal{M}$ is the robot mass, $\boldsymbol{r}$ is its position in the NED inertial frame, $\boldsymbol{\omega}$ is its angular velocity in the body frame, $\boldsymbol{R} \in \mathrm{SO}(3)$ is the rotation matrix that transforms body to inertial coordinates and $\boldsymbol{\mathcal{I}} \in \mathbb{R}^{3\times3}$ is the inertia matrix of the aircraft. The skew-symmetric matrix operator is denoted as $\mathcal{S}(\cdot)$, $\boldsymbol{e}_3$ is the z-axis unit vector and g is the acceleration of gravity. The propulsion force in the body frame is defined as $\boldsymbol{f}$, while the external force applied to the robot is $\boldsymbol{f}_e$. In a similar way $\boldsymbol{m}$ is the control torque in the body frame and $\boldsymbol{m}_e$ is the external torque. For simplicity we also assumed that the center of mass of the aircraft is located at the center of the frame and that the aerodynamic drag forces are negligible.

By introducing the control wrench as $\boldsymbol{\tau} = [\boldsymbol{f}^{\mathsf{T}}, \boldsymbol{m}^{\mathsf{T}}]^{\mathsf{T}}$ and the external wrench as $\boldsymbol{\tau_e} = [\boldsymbol{f}_e^{\mathsf{T}}, \boldsymbol{m}_e^{\mathsf{T}}]^{\mathsf{T}}$, we can rewritten the system dynamics in Lagrange form in terms of the generalized velocity $\boldsymbol{\nu} = [\dot{\boldsymbol{r}}^{\mathsf{T}}, \boldsymbol{\omega}^{\mathsf{T}}]^{\mathsf{T}}$ obtaining

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{G} = \boldsymbol{J}^{\mathsf{T}}\boldsymbol{\tau} + \boldsymbol{\tau}_e, \tag{3.2}$$

where

$$\boldsymbol{M} = \begin{bmatrix} \mathcal{M}\boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{\mathcal{I}} \end{bmatrix}, \qquad \boldsymbol{C}(\boldsymbol{\nu}) = \begin{bmatrix} \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & -\mathcal{S}(\boldsymbol{\mathcal{I}}\boldsymbol{\omega}) \end{bmatrix},$$
$$\boldsymbol{G} = -\begin{bmatrix} \mathcal{M}g\boldsymbol{e}_3 \\ \boldsymbol{0}_{3\times3} \end{bmatrix}, \qquad \boldsymbol{J} = \begin{bmatrix} \boldsymbol{R}^{\mathsf{T}} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix}.$$

Starting from the dynamic model just described we can also write the motion equations of the quadrotor in the form

$$\dot{\boldsymbol{\xi}} = \mathcal{F}(\boldsymbol{\xi}) + \mathcal{G}(\boldsymbol{\xi})\boldsymbol{u}, \tag{3.3}$$

where $\boldsymbol{\xi}$ is the current state and $\boldsymbol{u}$ is the control input. In our case the state variable $\boldsymbol{\xi} \in \mathbb{R}^{12}$ contains the robot position $\boldsymbol{r} = [x, y, z]^{\mathsf{T}}$, the roll-pitch-yaw (RPY) angles $\boldsymbol{\phi} = [\varphi, \vartheta, \psi]^{\mathsf{T}}$ of the body frame with respect to the inertial frame and the linear and angular velocities $\dot{\boldsymbol{r}}, \boldsymbol{\omega} = [p, q, r]^{\mathsf{T}}$ of the robot. The RPY angles describe the quadrotor attitude in a way equivalent to the rotation matrix $\boldsymbol{R}$, which in fact can be written in terms of these angles:

$$\boldsymbol{R} = \begin{bmatrix} c_\psi c_\vartheta & c_\psi s_\vartheta s_\varphi - s_\psi c_\varphi & c_\psi s_\vartheta c_\varphi + s_\psi s_\varphi \\ s_\psi c_\vartheta & s_\psi s_\vartheta s_\varphi + c_\psi c_\varphi & s_\psi s_\vartheta c_\varphi - s_\psi c_\varphi \\ -s_\vartheta & c_\vartheta s_\varphi & c_\vartheta c_\varphi \end{bmatrix},$$

where the notation c_α, s_α stand for $\cos(\alpha), \sin(\alpha)$ respectively.

For what concerns the input variable $\boldsymbol{u}$ we will consider four input commands: the thrust T, i.e. the opposite of the z component of the propulsion force $\boldsymbol{f} = [0, 0, -T]^{\mathsf{T}}$ and the three components of the control torque $\boldsymbol{m} = [\tau_\varphi, \tau_\vartheta, \tau_\psi]^{\mathsf{T}}$. This choice

is particularly convenient since these commands can be transformed into velocity commands of the rotors which in turn are achieved with a low level control of the current provided to each motor. Given the rotors angular velocities $\varpi_1, ...\varpi_4$, the thrust generated by each rotor can be modeled as $f_i = b\varpi_i^2$, where b is the thrust factor. In a similar way the drag torques of each propeller can be modeled as $\tau_{R,i} = d\varpi_i^2$, where d is the drag factor. Both b and d can be estimated with static thrust tests and depend on the rotor geometry and profile, on its disk area and radius and on the air density.

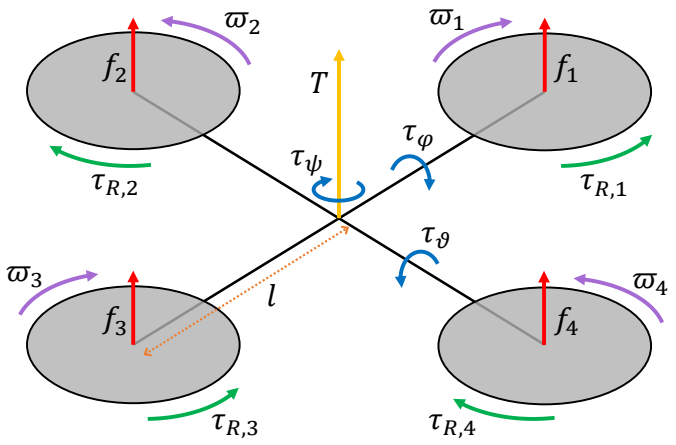

Figure 3.2. Visual representation of the forces and torques generated by each propeller.

Now, given the thrust and drag torque of each propeller, if we consider the arrangement shown in Fig. 3.2, the four input commands considered can be computed as

$$\begin{aligned} T &= f_1 + f_2 + f_3 + f_4 \\ \tau_\varphi &= l(f_2 - f_4) \\ \tau_\vartheta &= l(f_1 - f_3) \\ \tau_\psi &= -\tau_{R,1} + \tau_{R,2} - \tau_{R,3} + \tau_{R,4}, \end{aligned} \tag{3.4}$$

where l is the distance of the rotors from the quadrotor center of mass. So to summarize what said so far, the state variable $\boldsymbol{\xi}$ and the control input $\boldsymbol{u}$ are defined as

$$\begin{aligned} \boldsymbol{\xi} &= [x, y, z, \varphi, \vartheta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^\mathsf{T} \\ \boldsymbol{u} &= [T, \tau_\varphi, \tau_\vartheta, \tau_\psi]^\mathsf{T}. \end{aligned}$$

To obtain a simplified version of the motion equations, useful for developing a controller, we can make some assumptions. First we assume that the gyroscopic and aerodynamic effects of the rotors are negligible, then that the shape of the aircraft is symmetric and that there are no external disturbances. In addition, we will also consider the case when the roll and pitch angles are small, called small angle assumption, which implies that the rotational velocities $[p, q, r]^\mathsf{T} \approx [\dot{\varphi}, \dot{\vartheta}, \dot{\psi}]^\mathsf{T}$.

With these assumption we can describe the system dynamics as

$$\begin{aligned}
\dot{\boldsymbol{r}} &= [\dot{x}, \dot{y}, \dot{z}]^{\mathsf{T}} \\
\boldsymbol{\omega} &= [\dot{\varphi}, \dot{\vartheta}, \dot{\psi}]^{\mathsf{T}} \\
\ddot{x} &= -\big(\cos(\psi)\sin(\vartheta)\cos(\varphi) + \sin(\psi)\sin(\varphi)\big)\frac{T}{\mathcal{M}} = T_x\frac{T}{\mathcal{M}} \\
\ddot{y} &= -\big(\sin(\psi)\sin(\vartheta)\cos(\varphi) - \cos(\psi)\sin(\varphi)\big)\frac{T}{\mathcal{M}} = T_y\frac{T}{\mathcal{M}} \\
\ddot{z} &= g - \cos(\vartheta)\cos(\varphi)\frac{T}{\mathcal{M}} \\
\ddot{\varphi} &= \frac{\tau_\varphi}{I_x} \\
\ddot{\vartheta} &= \frac{\tau_\vartheta}{I_y} \\
\ddot{\psi} &= \frac{\tau_\psi}{I_z},
\end{aligned} \tag{3.5}$$

where $\boldsymbol{\mathcal{I}} = \mathrm{diag}\{I_x, I_y, I_z\}$.

3.2 Control problems and solutions

The simplest control problem we could be interested in solving is regulation, which consists in stabilizing the robot in a desired position. The only attitude in which a quadrotor can stay still is at hovering, i.e. when the roll and pitch angles are zero. This is an obvious consequence of the fact that all the propellers point up and so if the aircraft is tilted they generate an horizontal acceleration. The hovering position is an unstable equilibrium for the quadrotor and so we first need to use a closed loop control to stabilize the aircraft. More in general we can develop an attitude controller which stabilizes the quadrotor in a desired attitude, not necessarily in hovering. As described in [3], a very simple attitude control suitable for doing this is the following PD controller:

$$\begin{aligned}
\tau_\varphi &= k_{\varphi,p}(\varphi_d - \varphi) + k_{\varphi,d}(\dot{\varphi}_d - \dot{\varphi}) \\
\tau_\vartheta &= k_{\vartheta,p}(\vartheta_d - \vartheta) + k_{\vartheta,d}(\dot{\vartheta}_d - \dot{\vartheta}) \\
\tau_\psi &= k_{\psi,p}(\psi_d - \psi) + k_{\psi,d}(\dot{\psi}_d - \dot{\psi}),
\end{aligned} \tag{3.6}$$

where $k_{\{\varphi,\vartheta,\psi\},\{p,d\}} > 0$ are positive constants. Using only an attitude control we are not able to stabilize the quadrotor at a desired height, which is instead of our interest. To do so we need to define a PD height control using the (3.5). We obtain

$$T = \frac{\mathcal{M}}{\cos(\vartheta)\cos(\varphi)}\Big(g - \ddot{z}_d - k_{z,p}(z_d - z) - k_{z,d}(\dot{z}_d - \dot{z})\Big), \tag{3.7}$$

where $k_{z,p}, k_{z,d} > 0$ are positive constants. Now, to finally achieve the regulation task, we also need to add a controller to stabilize the motion in the horizontal plane. We start by assuming that we have control over the x and y dynamics and we stabilize the horizontal motion using a PD control acting on the virtual inputs

$U_x = \ddot{x}$ and $U_y = \ddot{y}$ as follows

$$\begin{aligned} U_x &= \ddot{x}_d + k_{x,d}(\dot{x}_d - \dot{x}) + k_{x,p}(x_d - x) \\ U_y &= \ddot{y}_d + k_{y,d}(\dot{y}_d - \dot{y}) + k_{y,p}(y_d - y), \end{aligned} \tag{3.8}$$

where $k_{\{x,y\},\{p,d\}} > 0$ are positive constants. Then, using again the simplified motion equations (3.5), we can compute the desired values of T_x and T_y

$$\begin{aligned} T_x &= \frac{\mathcal{M}}{T}\left(\ddot{x}_d + k_{x,d}(\dot{x}_d - \dot{x}) + k_{x,p}(x_d - x)\right) \\ T_y &= \frac{\mathcal{M}}{T}\left(\ddot{y}_d + k_{y,d}(\dot{y}_d - \dot{y}) + k_{y,p}(y_d - y)\right) \end{aligned} \tag{3.9}$$

and from them derive the desired roll and pitch angles needed to execute the horizontal motion required:

$$\begin{aligned} \varphi_d &= \arcsin\left(T_y \cos\psi - T_x \sin\psi\right), \\ \vartheta_d &= -\arcsin\left(\frac{T_x \cos\psi + T_y \sin\psi}{\cos\varphi_d}\right). \end{aligned}$$

With this type of controller, called hierarchical PD [6], we can stabilize the robot in a given configuration x_d, y_d, z_d, ψ_d by setting the desired velocities and accelerations to zero. Nevertheless this method is even more powerful because, by adding a timing law to the reference values, we are able to track a given trajectory. The desired trajectory will be defined as $x_d(t), y_d(t), z_d(t), \psi_d(t)$ and the reference velocities and accelerations will be in general non-zero.

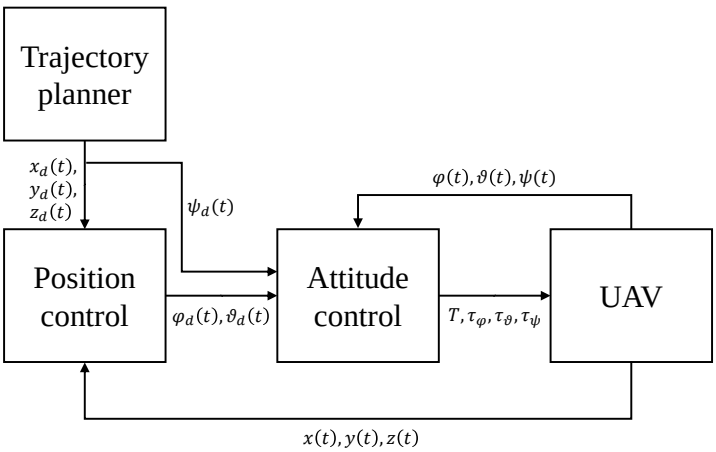

Figure 3.3. Block diagram of a hierarchical position controller.

Another type of hierarchical position controller can be developed using the backstepping approach [3]. Without going into the details, with this approach one can stabilize the errors dynamics by considering the right Lyapunov functions. The resulting controller is very similar to the hierarchical PD control but with slightly different coefficients and arrangement of the terms in the formulas (3.6), (3.7) and (3.9).

In the development of the two controllers considered so far it has been used the quadrotor simplified model (3.5), but this means that if the small angle assumption does not hold, they will not work properly. A more sophisticated controller which overcomes this issue is the controller in SE(3) [12]. This controller works directly on the rotation matrix $\boldsymbol{R}$ stabilizing all the RPY angles together. By defining the attitude error vector as $\boldsymbol{e}_R = \frac{1}{2}\mathcal{S}^{-1}(\boldsymbol{R}_d^\mathsf{T}\boldsymbol{R} - \boldsymbol{R}^\mathsf{T}\boldsymbol{R}_d)$ and the angular velocity error vector as $\boldsymbol{e}_\omega = \boldsymbol{\omega} - \boldsymbol{R}^\mathsf{T}\boldsymbol{R}_d\boldsymbol{\omega}_d$, we can define the attitude controller simply as

$$\boldsymbol{m} = -k_R\boldsymbol{e}_R - k_\omega\boldsymbol{e}_\omega + \boldsymbol{\omega}\times\mathcal{I}\boldsymbol{\omega} - \mathcal{I}(\mathcal{S}(\boldsymbol{\omega})\boldsymbol{R}^\mathsf{T}\boldsymbol{R}_d\boldsymbol{\omega}_d - \boldsymbol{R}^\mathsf{T}\boldsymbol{R}_d\dot{\boldsymbol{\omega}}_d), \tag{3.10}$$

where $\boldsymbol{R}_d$ represents the desired attitude and k_R, k_ω are positive constants. Following also in this case the hierarchical approach, to control the robot position we have to stack a position controller before the attitude controller to compute the thrust command T and to pass the desired attitude $\boldsymbol{R}_d$ and the desired rotational velocity $\dot{\boldsymbol{\omega}}_d$ to the attitude controller. The thrust can be simply computed as

$$T = (k_p(\boldsymbol{r} - \boldsymbol{r}_d) + k_d(\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_d) + \mathcal{M}g\boldsymbol{e}_3 - \mathcal{M}\ddot{\boldsymbol{r}}_d)\boldsymbol{R}\boldsymbol{e}_3, \tag{3.11}$$

where k_p, k_d are positive constants. To compute the desired attitude we start by considering the vectors $\boldsymbol{b}_{1d}$, which is the desired direction of the x axis of the body frame, and $\boldsymbol{b}_{3d}$, which points toward the desired position and is computed as

$$\boldsymbol{b}_{3d} = \frac{-k_p(\boldsymbol{r} - \boldsymbol{r}_d) - k_d(\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_d) - \mathcal{M}g\boldsymbol{e}_3 + \mathcal{M}\ddot{\boldsymbol{r}}_d}{||- k_p(\boldsymbol{r} - \boldsymbol{r}_d) - k_d(\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_d) - \mathcal{M}g\boldsymbol{e}_3 + \mathcal{M}\ddot{\boldsymbol{r}}_d||}.$$

At this point the desired attitude and the desired rotational velocity are

$$\boldsymbol{R}_d = \begin{bmatrix} \boldsymbol{b}_{1d} & \boldsymbol{b}_{3d}\times\boldsymbol{b}_{1d} & \boldsymbol{b}_{3d} \end{bmatrix} \qquad \dot{\boldsymbol{\omega}}_d = \boldsymbol{R}_d^\mathsf{T}\dot{\boldsymbol{R}}_d$$

When executing a trajectory tracking task it can happen to have some external forces acting on the robot, such as wind disturbances. In this case the controllers seen so far become quite unreliable. A possible way to handle this situation is to switch to an interaction controller [26]. These type of controllers are made to purposefully interact with the environment taking into account the external forces and torques. In this thesis we will simply assume that the external wrench is known, we will not go into the details of the known strategies to estimate the forces and torques applied to an UAV.

The first type of interaction control we will consider is the admittance control (Fig. 3.4), which can be implemented around an existing position control loop. The admittance control computes a compensated reference trajectory $\boldsymbol{r}_c(t), \psi_c(t)$ so to satisfy the virtual dynamics

$$\boldsymbol{M}_a\begin{bmatrix}\ddot{\boldsymbol{r}} - \ddot{\boldsymbol{r}}_d \\ \ddot{\psi} - \ddot{\psi}_d\end{bmatrix} + \boldsymbol{D}_a\begin{bmatrix}\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_d \\ \dot{\psi} - \dot{\psi}_d\end{bmatrix} + \boldsymbol{K}_a\begin{bmatrix}\boldsymbol{r} - \boldsymbol{r}_d \\ \psi - \psi_d\end{bmatrix} = \begin{bmatrix}\boldsymbol{f}_e \\ \tau_{\psi,e}\end{bmatrix}, \tag{3.12}$$

where $\boldsymbol{M}_a \in \mathbb{R}^{4\times4}$ is the virtual inertia matrix, $\boldsymbol{D}_a \in \mathbb{R}^{4\times4}$ is the virtual damping gain matrix and $\boldsymbol{K}_a \in \mathbb{R}^{4\times4}$ is the virtual spring gain matrix, these matrices must be positive definite. The reference trajectory, computed as

$$\begin{bmatrix}\ddot{\boldsymbol{r}}_c \\ \ddot{\psi}_c\end{bmatrix} = \begin{bmatrix}\ddot{\boldsymbol{r}}_d \\ \ddot{\psi}_d\end{bmatrix} - \boldsymbol{M}_a^{-1}\left(\boldsymbol{D}_a\begin{bmatrix}\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_d \\ \dot{\psi} - \dot{\psi}_d\end{bmatrix} + \boldsymbol{K}_a\begin{bmatrix}\boldsymbol{r} - \boldsymbol{r}_d \\ \psi - \psi_d\end{bmatrix} - \begin{bmatrix}\boldsymbol{f}_e \\ \tau_{\psi,e}\end{bmatrix}\right), \tag{3.13}$$

is then given as input to the position control. It has to be mentioned that this strategy has the disadvantage of begin quite difficult to be tuned since uses three different controllers in cascade.

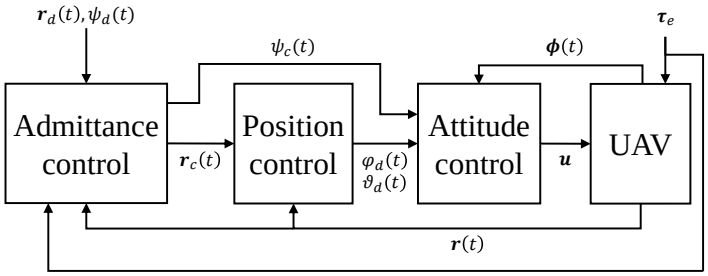

Figure 3.4. Block diagram of the admittance control for an underactuated UAV.

An alternative interaction control that can be used is the impedance control that computes directly the input commands from the state error so to satisfy the virtual dynamics

$$\boldsymbol{M}_v(\dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_d) + \boldsymbol{D}_v(\boldsymbol{\nu} - \boldsymbol{\nu}_d) + \boldsymbol{K}_v(\boldsymbol{x} - \boldsymbol{x}_d) = \boldsymbol{\tau}_e, \tag{3.14}$$

where $\boldsymbol{x} = [\boldsymbol{r}, \boldsymbol{\phi}]^\mathsf{T}$ and $\boldsymbol{x}_d = [\boldsymbol{r}_d, \boldsymbol{\phi}_d]^\mathsf{T}$ are the robot state and desired state, $\boldsymbol{M}_v \in \mathbb{R}^{6\times6}$ is the apparent inertia matrix, $\boldsymbol{D}_v \in \mathbb{R}^{6\times6}$ is the desired damping matrix and $\boldsymbol{K}_v \in \mathbb{R}^{6\times6}$ is the desired stiffness matrix, these matrices must be positive definite. The control wrench is computed by inserting (3.14) into (3.2) obtaining

$$\begin{aligned}\boldsymbol{J}^\mathsf{T}\boldsymbol{\tau} = \boldsymbol{M}\boldsymbol{M}_v^{-1}(\boldsymbol{M}_v\dot{\boldsymbol{\nu}}_d + \boldsymbol{D}_v(\boldsymbol{\nu}_d - \boldsymbol{\nu}) + \boldsymbol{K}_v(\boldsymbol{x}_d - \boldsymbol{x}))+ \\ + (\boldsymbol{M}\boldsymbol{M}_v^{-1} - \boldsymbol{I}_{6\times6})\boldsymbol{\tau}_e + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{G}.\end{aligned} \tag{3.15}$$

This control cannot be directly applied if the UAV is underactuated, as in the case of a quadrotor. Under this circumstance what we can do is something similar to the previous controllers, that is to use a cascaded control structure [25] (Fig. 3.5), with

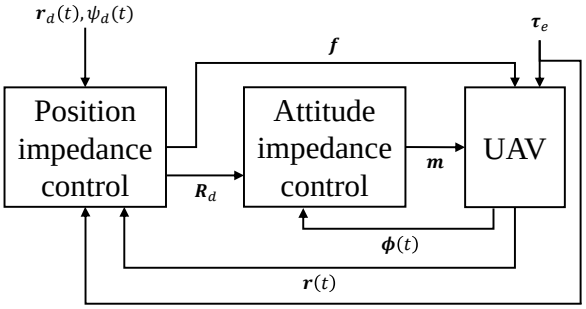

Figure 3.5. Block diagram of the impedance control for an underactuated UAV.

a position and attitude impedance controllers defined as

$$\begin{aligned}\boldsymbol{R}_d\boldsymbol{f} &= \mathcal{M}\ddot{\boldsymbol{r}}_d + (\mathcal{M}\mathcal{M}_v^{-1} - 1)\boldsymbol{f}_e - \mathcal{M}\mathcal{M}_v^{-1}(\boldsymbol{D}_1(\dot{\boldsymbol{r}} - \dot{\boldsymbol{r}}_d) + \boldsymbol{K}_1(\boldsymbol{r} - \boldsymbol{r}_d)) - \mathcal{M}g\boldsymbol{e}_3 \\ \boldsymbol{m} &= \boldsymbol{\mathcal{I}}\dot{\boldsymbol{\omega}}_d + (\boldsymbol{\mathcal{I}}\boldsymbol{\mathcal{I}}_v^{-1} - \boldsymbol{I}_{3\times 3})\boldsymbol{m}_e - \boldsymbol{\mathcal{I}}\boldsymbol{\mathcal{I}}_v^{-1}(\boldsymbol{D}_2(\boldsymbol{\omega} - \boldsymbol{\omega}_d) + \boldsymbol{K}_2(\boldsymbol{\phi} - \boldsymbol{\phi}_d)) - \mathcal{S}(\boldsymbol{\mathcal{I}}\boldsymbol{\omega})\boldsymbol{\omega},\end{aligned}$$

where $\boldsymbol{M}_v = \text{blockdiag}\{\mathcal{M}_v\boldsymbol{I}_{3\times 3}, \boldsymbol{\mathcal{I}}_v\}$, $\boldsymbol{D}_v = \text{blockdiag}\{\boldsymbol{D}_1, \boldsymbol{D}_2\}$ and $\boldsymbol{K}_v = \text{blockdiag}\{\boldsymbol{K}_1, \boldsymbol{K}_2\}$.

Chapter 4

Planning and navigation

In the controllers presented in the previous chapter we always assumed to have as input a desired cartesian trajectory, but how can we obtain it? When dealing with very simple motions we can define it by manually writing a function $\boldsymbol{r}_d(t) : \mathbb{R} \to \mathbb{R}^3$ which specifies the desired robot position at a given time t, but it is obvious that this is not a general solution. In this chapter we introduce the trajectory planning problem together with some strategies that can be used to solve it. In particular two main general approaches will be considered: fully offline path planning and online reactive navigation. We will see that the algorithm proposed in this thesis works halfway between these two methods.

4.1 Preliminary notions

To formalize the general problem of a mobile robot $\mathcal{B}$ that has to move in a environment with obstacles, it is important to start by introducing some basic notions. We start by introducing the workspace $\mathcal{W}$, which is the space inside which the robot can move and it is usually $\mathbb{R}^2$ or $\mathbb{R}^3$. Inside this workspace let us assume to have m known fixed obstacles $\mathcal{O}_1, ..., \mathcal{O}_m$ with which the robot has to avoid collisions. Then we introduce the concept of configuration space $\mathcal{C}$, whose dimension n depends on the degrees of freedom of the robot, which allows to uniquely describe the robot configuration as a point $\boldsymbol{q} \in \mathcal{C}$. The subset of configurations that cause a collision between the robot $\mathcal{B}$ and a generic obstacle $\mathcal{O}_i$ is called $\mathcal{C}$-obstacle and is defined as

$$\mathcal{CO}_i = \{\boldsymbol{q} \in \mathcal{C} : \mathcal{B}(\boldsymbol{q}) \cap \mathcal{O}_i \neq \emptyset\},$$

while the complement of all the $\mathcal{C}$-obstacles is the free configuration space

$$\mathcal{C}_{\text{free}} = \mathcal{C} - \bigcup_{i=1}^{m} \mathcal{CO}_i = \{\boldsymbol{q} \in \mathcal{C} : \mathcal{B}(\boldsymbol{q}) \cap \left(\bigcup_{i=1}^{m} \mathcal{O}_i \right) = \emptyset\},$$

which contains all the possible robot configurations that do not cause a collision. Using this notions the trajectory planning problem can be stated as finding a trajectory $\boldsymbol{q}(t) : \mathbb{R} \to \mathcal{C}$, for $t \in [t_i, t_f]$, that specifies the robot motion from a starting point $\boldsymbol{q}_i = \boldsymbol{q}(t_i)$ to a goal point $\boldsymbol{q}_f = \boldsymbol{q}(t_f)$ and it is safe, i.e. all its points are in $\mathcal{C}_{\text{free}}$. To simplify this problem it can be convenient to divide the trajectory

into a geometric path $\boldsymbol{q}(s) : \mathbb{R} \to \mathcal{C}$ and a timing law $s = s(t)$, where s is a parameter varying from $s_i = s(t_i)$ to $s_f = s(t_f)$. Taking advantage of this simplification, from now on we will focus only on finding a safe path, cutting out from our analysis the timing law definition which is trivial and independent from the obstacle avoidance requirement.

In the specific case of an UAV, the workspace is clearly three-dimensional $\mathcal{W} = \mathbb{R}^3$ and the configuration space has dimension $n = 6$, since is defined as $\mathcal{C} = \mathbb{R}^3 \times \mathrm{SO}(3)$. This entails that we can express the robot configuration as

$$\boldsymbol{q} = [x, y, z, \varphi, \vartheta, \psi]^\mathsf{T},$$

where x, y, z describe the cartesian position in the inertial frame and φ, ϑ, ψ are the RPY angles.

4.2 Path planning

When dealing with a priori known environments, it is possible to plan a safe path for the robot completely offline. The different strategies available in the literature [22] can be subdivided in two main categories: non-probabilistic and probabilistic based methods.

4.2.1 Non-probabilistic methods

For what concerns the non-probabilistic methods, many solutions have been studied. A very famous one is [17] and works by defining a roadmap $\mathcal{R}$ from the generalized Voronoi diagram of $\mathcal{C}_{\text{free}}$. Without going to much in the details, the Voronoi diagram is defined as

$$\mathcal{R} = \mathcal{V}(\mathcal{C}_{\text{free}}) = \{\boldsymbol{q} \in \mathcal{C}_{\text{free}} : \mathrm{card}(N(\boldsymbol{q})) > 1\}, \tag{4.1}$$

where $N(\boldsymbol{q})$ is the neighbors of $\boldsymbol{q}$. Given the roadmap, the algorithm proceeds to connect $\boldsymbol{q}_i$ and $\boldsymbol{q}_f$ to $\mathcal{R}$ via retraction so to build a connected graph. To conclude, using a simple graph pathfinding algorithm as A^*, the final safe path is returned.

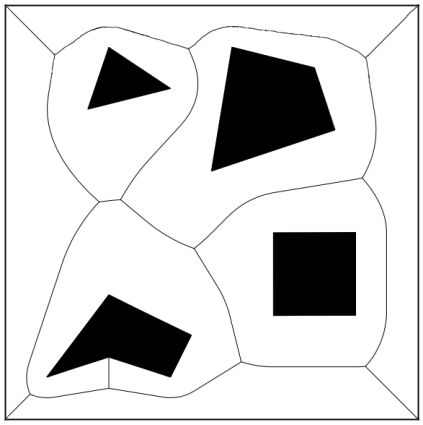

Figure 4.1. Example of Voronoi diagram in 2D. Adapted from [22].

This method has the big advantage of maximizing the safety since the Voronoi diagram is composed by the farthest points from the obstacles. In addition this method is complete and multiple-query.

Another method available is the exact cell decomposition [23], which uses the decomposition of the environment in cells to create a connectivity graph where each cell is represented as a node. Similarly to the previous algorithm, the graph built is then used similarly to a roadmap to reduce the initial problem to a graph pathfinding problem. This algorithm, as the previous, is complete and multiple-query, but is more flexible for example when dealing with nonholonomic constraints. The two algorithms presented have anyway the disadvantage that the time complexity increases with the number of nodes in the graph, which means that in high dimensional environments or with many obstacles the time required to compute a solution can be a limitation.

4.2.2 Probabilistic methods

Differently from the previous category, the probabilistic methods are not complete, but they are faster. This type of algorithms basically work by repeatedly choosing with a certain probability distribution a point $\boldsymbol{q}_{\text{rand}}$ and using it to grow a tree graph to connect $\boldsymbol{q}_i$ to $\boldsymbol{q}_f$. A simple and popular algorithm of this kind is Rapidly-exploring Random Trees (RRT) [11] which starts growing a tree $\mathcal{T}_s$ from $\boldsymbol{q}_i$. In each iteration a $\boldsymbol{q}_{\text{rand}}$ is extracted with uniform probability distribution form $\mathcal{C}$, then the nearest node $\boldsymbol{q}_{\text{near}}$ is identified and used to chose a $\boldsymbol{q}_{\text{new}}$ at distance δ to $\boldsymbol{q}_{\text{near}}$ in the direction of $\boldsymbol{q}_{\text{rand}}$. If the segment connecting $\boldsymbol{q}_{\text{near}}$ to $\boldsymbol{q}_{\text{new}}$ is safe, $\boldsymbol{q}_{\text{new}}$ is added to $\mathcal{T}_s$. Checking if the new arc is safe requires very little computational power and this is the reason why this algorithm is faster then the previous ones. The final safe path can be found backtracking from a node close enough to $\boldsymbol{q}_f$ to the root. It is clear that the number of iterations necessary to find a solution is not fixed, therefore it can happen that the space exploration done does not allow to find a solution path even if it exist, this algorithm is in fact only probabilistically complete and single-query.

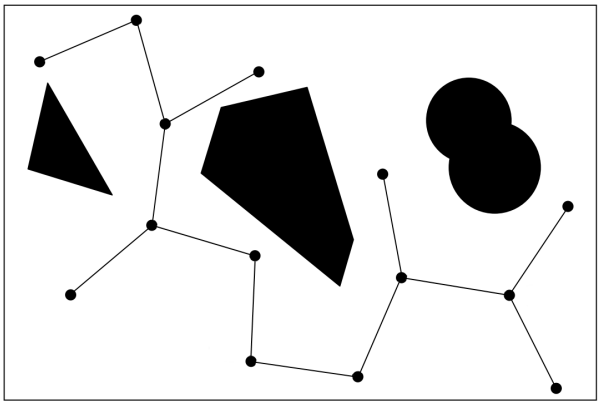

Figure 4.2. Example of a 2D random tree generated by RRT. Adapted from [22].

All the path planning algorithms seen so far have the problem that the final path may not be smooth since it was generated from a graph. This can be an issue, especially when dealing with underactuated robots such as quadrators. In fact this

types of paths, containing sharp turns, are in general unfeasible for these robots. This means that we know in advice that a robot of this kind will never be able to follow exactly the given trajectory. An intuitive way to overcome this limitation is to apply a smoothing process [18] on the solution path. Without focusing too much on its implementation, this process consists in substituting the sharp turns with smooth curves and then check if the new path is still safe. If the response is negative it means that one of the substitutions creates a collision and we can only try to repeat the process starting from a new solution path. The success of this strategy obviously deepens on the path planning algorithm chosen and the complexity of the environment, but in general it can slow down a lot the path computation and it does not give any guarantee on the completeness of the procedure.

4.3 Reactive navigation

A different approach that can be used to navigate in obstacle-rich environments is the reactive navigation, which consist in collecting the information about the environment online and exploit it to avoid collisions with a reaction strategy. A quite reliable one is [9], which is based on the computation of an artificial potential field $U_t(\boldsymbol{q})$ composed of an attractive potential $U_a(\boldsymbol{q})$ generated by $\boldsymbol{q}_f$ and multiple repulsive potentials $U_{r,i}(\boldsymbol{q})$ generated by the obstacles. Then, given $U_t(\boldsymbol{q})$, the robot has simply to follow the negative gradient $\boldsymbol{f}_t(\boldsymbol{q}) = -\nabla U_t(\boldsymbol{q})$. Going more in details, the first step is to define the attractive potential based on the goal configuration, for example as

$$U_a(\boldsymbol{q}) = \frac{1}{2} k_a ||\boldsymbol{q}_f - \boldsymbol{q}||^2, \tag{4.2}$$

where k_a is a positive constant. The negative gradient of this potential will then be

$$\boldsymbol{f}_a(\boldsymbol{q}) = -\nabla U_a(\boldsymbol{q}) = k_a(\boldsymbol{q}_f - \boldsymbol{q}). \tag{4.3}$$

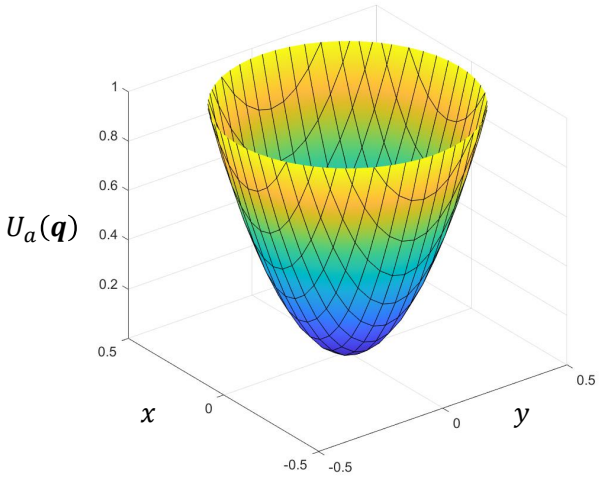

Figure 4.3. Attractive potential plot with $\boldsymbol{q} = [x, y]^\mathsf{T}$, $\boldsymbol{q}_f = [0, 0]^\mathsf{T}$ and k_a=10.

Then we have to define the repulsive potential produced by a generic convex $\mathcal{C}$-obstacle $\mathcal{CO}_i$, we can use

$$U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{1}{2} k_r \left(\frac{1}{\eta_i(\boldsymbol{q})} - \frac{1}{\eta_{0,i}} \right)^2 & \text{for } \eta_i(\boldsymbol{q}) \leq \eta_{0,i} \\ 0 & \text{for } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases}, \tag{4.4}$$

where k_r is a positive constant, $\eta_i(\boldsymbol{q})$ is the clearance, i.e. the minimum distance from $\mathcal{CO}_i$, and $\eta_{0,i}$ is the range of influence of $\mathcal{CO}_i$. In this way its negative gradient will be

$$\boldsymbol{f}_{r,i}(\boldsymbol{q}) = -\nabla U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{k_r}{\eta_i^2(\boldsymbol{q})} \left(\frac{1}{\eta_i(\boldsymbol{q})} - \frac{1}{\eta_{0,i}} \right) \nabla \eta_i(\boldsymbol{q}) & \text{for } \eta_i(\boldsymbol{q}) \leq \eta_{0,i} \\ 0 & \text{for } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases}. \tag{4.5}$$

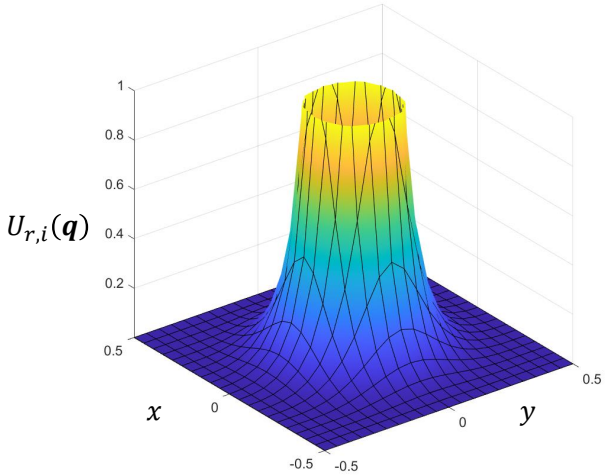

Figure 4.4. Repulsive potential plot with $\boldsymbol{q} = [x, y]^\mathsf{T}$, a point obstacle at $[0, 0]^\mathsf{T}$, $k_r = 0.1$ and $\eta_0 = 0.5$.

To conclude, the total potential and its negative gradient will simply be

$$U_t(\boldsymbol{q}) = U_a(\boldsymbol{q}) + \sum_{i=1}^{m} U_{r,i}(\boldsymbol{q}) \tag{4.6}$$

$$\boldsymbol{f}_t(\boldsymbol{q}) = -\nabla U_t(\boldsymbol{q}) = \boldsymbol{f}_a(\boldsymbol{q}) + \sum_{i=1}^{m} \boldsymbol{f}_{r,i}(\boldsymbol{q}) \tag{4.7}$$

At this point, to make the robot follow the negative gradient $\boldsymbol{f}_t(\boldsymbol{q})$, we can use it as the reference generalized velocity, but also other choices are possible. In any case this method doesn't guarantee to follow the shortest path nor to reach the goal since there may be local minima in the potential function. On the other hand it works entirely online and gives the possibility of collecting the information about the obstacles positions with sensors while exploring the environment.

Chapter 5

Problem formulation and proposed approach

The specific problem that we will consider in this thesis is a quadrotor that has to move in a workspace $\mathcal{W} = \mathbb{R}^3$ from a starting point $\boldsymbol{r}_i \in \mathcal{W}$ to a goal point $\boldsymbol{r}_f \in \mathcal{W}$ while avoiding collisions with the fixed obstacles $\mathcal{O}_1, ..., \mathcal{O}_m$ present in the environment. In other words we want the quadrotor to execute a trajectory $\boldsymbol{q}(t)$ such that

$$\boldsymbol{q}(t) \in \mathcal{C}_{\text{free}} \quad \forall t \in [t_i, t_f],$$
$$\text{with } \boldsymbol{q}(t_i) = \boldsymbol{q}_i = [\boldsymbol{r}_i^\mathsf{T}, 0, 0, \psi_i]^\mathsf{T} \text{ and } \boldsymbol{q}(t_f) = \boldsymbol{q}_f = [\boldsymbol{r}_f^\mathsf{T}, 0, 0, \psi_f]^\mathsf{T}.$$

It is important to note that we are interested in a motion of the quadrotor from an hovering configuration to another where the yaw angle has to go from ψ_i to ψ_f. For the sake of simplicity we will consider the robot body as a point so that the configuration space investigated is just $\mathcal{C} = \mathbb{R}^3$. This simplification means assuming that the RPY angles motion does not influence the obstacle avoidance and this is convenient when dealing with collision checking because it allows to focus only on the robot cartesian trajectory $\boldsymbol{r}(t)$. Since the yaw angle motion is decoupled from the cartesian motion, we will assume that its trajectory is already defined as $\psi_d(t)$ for $t \in [t_i, t_f]$ with $\psi_d(t_i) = \psi_i$ and $\psi_d(t_f) = \psi_f$.

The challenging part of this problem is to preserve the vehicle integrity by remaining away from all the obstacles, even the unexpected ones, while minimizing the overall execution time. As already discussed in the previous chapter, planning a feasible trajectory for a quadrotor is not straightforward due to its underactuated nature and this means that the planning phase can require a non-indifferent amount of time. The method proposed in this thesis is based on the idea that the smoothing required to get a feasible trajectory from a simple geometric path can be done online by exploiting the underactuated nature of the UAVs. This approach may save some time in the preliminary planning phase while guaranteeing good performances with a reduced computational effort even when the environment changes.

The control strategy that will be presented starts with an offline computation of a safe geometrical path $\boldsymbol{r}_r(s)$ from $\boldsymbol{r}_i$ to $\boldsymbol{r}_f$ using the fast and not optimal RRT-Connect algorithm, which will be described later in Section 5.1. Then a timing law is added to obtain a reference trajectory $\boldsymbol{r}_r(t)$ that the UAV will try to follow. At

this point the planning switches in an online phase running during the tracking of the given initial trajectory. With the augmented information collected online about the near obstacles, a virtual repulsive force $\boldsymbol{f}_r(\boldsymbol{r})$ is computed to be used by the controller. Since the trajectory tracking is done with an impedance controller with inputs $\boldsymbol{r}_r(t)$ and $\boldsymbol{f}_r(\boldsymbol{r})$, the final motion will take into account changes in the environment and the quadrotor dynamics. The output of the impedance controller is a force command $\boldsymbol{f}_d$ which is then converted by a PD attitude controller to thrust and torque commands which are in turn used to control the quadrotor.

5.1 Trajectory planning

In the first step of the algorithm we use an offline path planner to compute a safe geometrical trajectory $\boldsymbol{r}_r(t)$ that will be used as reference in the online phase. As anticipated in Chapter 4, we can divide a trajectory into a path and a timing law to work easily on the obstacle avoidance. To plan a safe path it is convenient to look within the class of probabilistic planners because they are faster. In our implementation we chose the RRT-Connect algorithm [10], which is fast enough and reliable in computing paths with few segments. This algorithm is a variant of bidirectional RRT and its logic can be summarized as follows.

Algorithm 1: RRT Connect

Data: Obstacles $\mathcal{O}_1, ..., \mathcal{O}_m$, starting point $\boldsymbol{r}_i$, goal point $\boldsymbol{r}_f$
Result: Safe geometric path $\boldsymbol{r}(s)$ from $\boldsymbol{r}_i$ to $\boldsymbol{r}_f$
Initialize the tree $\mathcal{T}_a$ with $\boldsymbol{r}_i$ and the tree $\mathcal{T}_b$ with $\boldsymbol{r}_f$;
for $k = 1$ **to** $k_{\max}$ **do**
  $\boldsymbol{r}_{\text{rand}} \leftarrow$ random point;
  if *EXTEND*$(\mathcal{T}_a, \boldsymbol{r}_{\text{rand}})$ *is not trapped* **then**
    if *CONNECT*$(\mathcal{T}_b, \boldsymbol{r}_{\text{new}})$ *is reached* **then**
      return path stored in $\mathcal{T}_a$ and $\mathcal{T}_b$
    end
  end
  Swap $\mathcal{T}_a$ and $\mathcal{T}_b$
end
return failure

The value of the parameter $k_{\max}$ has to be chosen high enough to find a solution since this algorithm is only probabilistically complete. The more complex the obstacles pattern is, the higher $k_{\max}$ has to be set if we want to increase the probability of finding a solution. On the other hand if $k_{\max}$ is set too high the execution time could become very large. The function $EXTEND(\mathcal{T}_a, \boldsymbol{r}_{\text{rand}})$ finds, if exist, a point $\boldsymbol{r}_{\text{new}}$ in the direction of $\boldsymbol{r}_{\text{rand}}$ that can be connected to $\mathcal{T}_a$. The function $CONNECT(\mathcal{T}_b, \boldsymbol{r}_{\text{new}})$ iterates the $EXTEND$ operation until $\boldsymbol{r}_{\text{new}}$ or an obstacle is reached.

If the algorithm does not return failure, we have computed a safe path $\boldsymbol{r}_r(s)$. Now, to obtain the final reference trajectory, we need to add a timing law. Without loss of generality we will consider a constant speed timing law that allows to cover

the path in a given time t_d. This timing law is defined as

$$s(t) = \left(\frac{s_f - s_i}{t_d}\right) t. \tag{5.1}$$

It needs to be specified that the trajectory computed in this phase is in general unfeasible since it can contain sharp turns at constant speed. To obtain a good behavior of the trajectory tracking controller is useful to fix the desired acceleration always to zero, in this way we avoid the formation of peaks arising from the derivation of the non-continuous desired velocity $\dot{\boldsymbol{r}}_r(t)$. Let us stress that such simple workaround works particularly well in this case because we are considering a constant speed trajectory, whereas with other type of trajectories the peaks in $\ddot{\boldsymbol{r}}_r(t)$ should be removed in a different way.

5.2 Repulsive force computation

In the online phase of the algorithm the obstacles are used to compute a virtual repulsive force $\boldsymbol{f}_r(\boldsymbol{r})$ used by the impedance controller. The computation of this force can be done using an artificial potential field similar to the one presented in [9] and Section 4.3. The only difference is that in this case we will consider only the repulsive potentials generated by the obstacles with the same range of influence. In particular, if we consider the generic obstacle $\mathcal{O}_i$, the artificial repulsive potential and force generated by it are

$$U_{r,i}(\boldsymbol{r}) = \begin{cases} \frac{k_r}{2}\left(\frac{1}{\eta_i(\boldsymbol{r})} - \frac{1}{\eta_0}\right)^2 & \text{for } \eta_i(\boldsymbol{r}) \leq \eta_0 \\ 0 & \text{for } \eta_i(\boldsymbol{r}) > \eta_0 \end{cases}, \tag{5.2}$$

$$\boldsymbol{f}_{r,i}(\boldsymbol{r}) = -\nabla U_{r,i}(\boldsymbol{r}) = \begin{cases} \frac{k_r}{\eta_i^2(\boldsymbol{r})}\left(\frac{1}{\eta_i(\boldsymbol{r})} - \frac{1}{\eta_0}\right)\nabla\eta_i(\boldsymbol{r}) & \text{for } \eta_i(\boldsymbol{r}) \leq \eta_0 \\ \boldsymbol{0} & \text{for } \eta_i(\boldsymbol{r}) > \eta_0 \end{cases}, \tag{5.3}$$

where k_r is a positive constant, $\eta_i(\boldsymbol{r})$ is the minimum distance between $\boldsymbol{r}$ and the obstacle borders and $\eta_0 > 0$ is the fixed range of influence. To join all the obstacles contributions the overall repulsive force is simply computed with the vector sum of all the forces $\boldsymbol{f}_{r,i}$

$$\boldsymbol{f}_r(\boldsymbol{r}) = \sum_{i=1}^{m} \boldsymbol{f}_{r,i}(\boldsymbol{r}). \tag{5.4}$$

The main disadvantage of using the potential (5.2) is that it does not account for the speed of approach towards the obstacle. This can be a problem for example when the trajectory passes parallel to an obstacle. If the robot velocity is perpendicular to the vector of relative position with respect to the obstacle, there is no risk to have a collision and so we would like to have $\boldsymbol{f}_r = \boldsymbol{0}$. In addition the function (5.2) makes the potential raise too fast when approaching an obstacle. To solve these issues we can use a polynomial function in the potential definition and add a scaling factor on $\boldsymbol{f}_{r,i}$ based on the UAV velocity component directed towards the obstacle $\mathcal{O}_i$. For

example we can define

$$U_{r,i}(\boldsymbol{r}) = \begin{cases} -\frac{k_r}{3}\left(\eta_i(\boldsymbol{r}) - \eta_0\right)^3 & \text{for } \eta_i(\boldsymbol{r}) \leq \eta_0 \\ 0 & \text{for } \eta_i(\boldsymbol{r}) > \eta_0 \end{cases}, \tag{5.5}$$

$$\boldsymbol{f}_{r,i}(\boldsymbol{r}) = -\nabla U_{r,i}(\boldsymbol{r}) = \begin{cases} k_r\left(\eta_i(\boldsymbol{r}) - \eta_0\right)^2 \nabla\eta_i(\boldsymbol{r}) & \text{for } \eta_i(\boldsymbol{r}) \leq \eta_0 \\ \boldsymbol{0} & \text{for } \eta_i(\boldsymbol{r}) > \eta_0 \end{cases}, \tag{5.6}$$

where k_r is a positive constants. Then the total repulsive force, scaled using the robot velocity $\dot{\boldsymbol{r}}$, will be computed as

$$\boldsymbol{f}_r(\boldsymbol{r}) = \sum_{i=1}^{m} \max\left\{\frac{\nabla\eta_i(\boldsymbol{r})^\mathsf{T} \cdot \dot{\boldsymbol{r}}}{\eta_i(\boldsymbol{r})}, 0\right\} \cdot \boldsymbol{f}_{r,i}(\boldsymbol{r}). \tag{5.7}$$

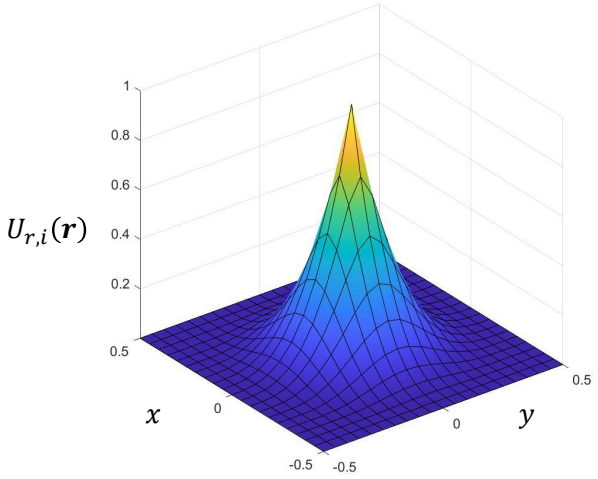

Figure 5.1. Repulsive potential plot using (5.5) with $\boldsymbol{r} = [x, y]^\mathsf{T}$, a point obstacle at $[0, 0]^\mathsf{T}$, $k_r = 24$ and $\eta_0 = 0.5$.

5.3 Impedance control

To control the robot motion we have to track the reference trajectory $\boldsymbol{r}_r(t)$ while taking into account the artificial repulsive force $\boldsymbol{f}_r(\boldsymbol{r})$. We showed in Chapter 3 that an impedance control [26] can be used to obtain the desired closed loop dynamics (3.14) in the presence of an external wrench $\boldsymbol{\tau}_e$. In this case, however, we want to consider $-\boldsymbol{f}_r$ as the external force and no external torque. Therefore, using only the upper half of the (3.15) we obtain that the virtual input force $\boldsymbol{f}_d$, expressed in the inertial frame, is computed as

$$\boldsymbol{f}_d = \mathcal{M}\boldsymbol{M}_v^{-1}(\boldsymbol{M}_v\ddot{\boldsymbol{r}}_d + \boldsymbol{D}_v(\dot{\boldsymbol{r}}_d - \dot{\boldsymbol{r}}) + \boldsymbol{K}_v(\boldsymbol{r}_d - \boldsymbol{r})) - (\mathcal{M}\boldsymbol{M}_v^{-1} - \boldsymbol{I}_{3\times3})\boldsymbol{f}_r - \mathcal{M}g\boldsymbol{e}_3, \tag{5.8}$$

where in this case $\boldsymbol{M}_v, \boldsymbol{D}_v, \boldsymbol{K}_v \in \mathbb{R}^{3\times3}$. With this controller there is also the possibility of compensating a real external force $\boldsymbol{f}_e$ acting on the robot while moving

away from the obstacles. To achieve this goal, it is sufficient to sum the two external forces obtaining

$$\begin{aligned} \boldsymbol{f}_d = \mathcal{M}\boldsymbol{M}_v^{-1}(\boldsymbol{M}_v\ddot{\boldsymbol{r}}_d + \boldsymbol{D}_v(\dot{\boldsymbol{r}}_d - \dot{\boldsymbol{r}}) + \boldsymbol{K}_v(\boldsymbol{r}_d - \boldsymbol{r})) + \\ + (\mathcal{M}\boldsymbol{M}_v^{-1} - \boldsymbol{I}_{3\times 3})(\boldsymbol{f}_e - \boldsymbol{f}_r) - \mathcal{M}g\boldsymbol{e}_3. \end{aligned} \tag{5.9}$$

5.4 Attitude control

To conclude, since the quadrotor is controlled by specifying the thrust and torque commands $\boldsymbol{u}$, we have to derive them from the virtual input force $\boldsymbol{f}_d$ by considering $\boldsymbol{f}_d = \mathcal{M}\ddot{\boldsymbol{r}}$. In this way, similarly to what was done for the hierarchical PD control [6], using the quadrotor simplified model (3.5) we obtain

$$T = -\frac{\boldsymbol{f}_d^\top \boldsymbol{e}_3}{\cos\vartheta\cos\varphi}, \tag{5.10}$$

$$T_x = \frac{\boldsymbol{f}_d^\top \boldsymbol{e}_1}{T}, \tag{5.11}$$

$$T_y = \frac{\boldsymbol{f}_d^\top \boldsymbol{e}_2}{T}, \tag{5.12}$$

where $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ are the axes unit vectors. As already seen in Section 3.2, it follows that the roll and pitch angles required to obtain T_x and T_y can be computed as

$$\begin{aligned} \varphi_d &= \arcsin\left(\frac{\boldsymbol{U}_y}{T}\cos\psi - T_x\sin\psi\right) \\ \vartheta_d &= -\arcsin\left(\frac{T_x\cos\psi + T_y\sin\psi}{\cos\varphi_d}\right). \end{aligned}$$

Practically speaking, we have transformed the virtual input force $\boldsymbol{f}_d$ into a thrust command and a desired attitude. Following again a hierarchical approach, we can finally compute the torque commands from the desired attitude with a PD attitude

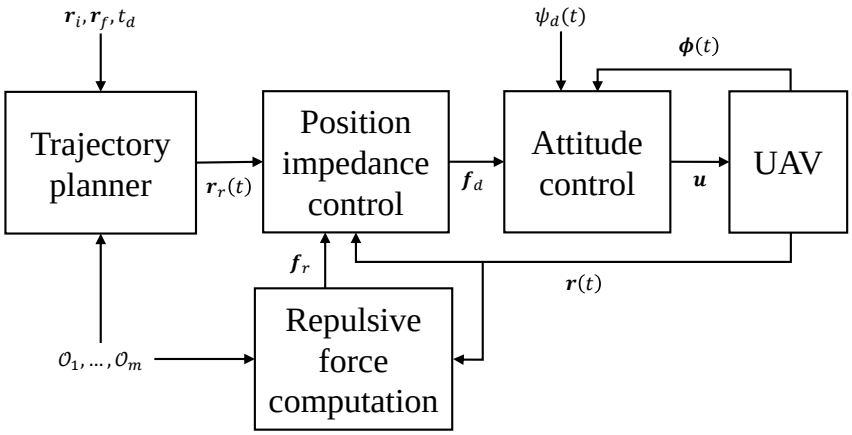

Figure 5.2. Block diagram of the proposed method.

controller as the one described in [3]

$$\tau_\varphi = k_{\varphi,p}(\varphi_d - \varphi) + k_{\varphi,d}(\dot{\varphi}_d - \dot{\varphi})$$
$$\tau_\vartheta = k_{\vartheta,p}(\vartheta_d - \vartheta) + k_{\vartheta,d}(\dot{\vartheta}_d - \dot{\vartheta})$$
$$\tau_\psi = k_{\psi,p}(\psi_d - \psi) + k_{\psi,d}(\dot{\psi}_d - \dot{\psi}),$$

where it is important to remark that the trajectory $\psi_d(t)$ was given in advance and is assumed to be independent of the obstacle avoidance requirement.

5.5 Safety conditions

Now that the control strategy has been described in details, we can make some considerations about the safety of this method. In particular we are going to discuss the conditions under which the quadrotor always keeps a limited distance from the initial safe trajectory $\boldsymbol{r}_r(t)$. By defining the state error as

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_d - \boldsymbol{r} \\ \dot{\boldsymbol{r}}_d - \dot{\boldsymbol{r}} \end{bmatrix},$$

we can rewrite the closed loop dynamics (3.14) of the impedance control in the form $\dot{\boldsymbol{z}} = f(\boldsymbol{z}) + g(\boldsymbol{z})\boldsymbol{\sigma}$, for $\boldsymbol{\sigma} \in \Theta \subset \mathbb{R}^6$, as

$$\dot{\boldsymbol{z}} = \begin{bmatrix} \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \\ -\boldsymbol{M}_v^{-1}\boldsymbol{K}_v & -\boldsymbol{M}_v^{-1}\boldsymbol{D}_v \end{bmatrix} \boldsymbol{z} + \begin{bmatrix} \boldsymbol{0}_{3\times3} \\ \boldsymbol{M}_v^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{f}_v \end{bmatrix}, \tag{5.13}$$

where $\boldsymbol{f}_v$ is the virtual external force applied to the quadrotor.

To ensure safety through control barrier functions [1] we have to consider a safe set $\mathcal{C}_0$ representing the state space region near the reference trajectory and show that it is an invariant set for the system (5.13). To do this we will consider the smooth function $h : \mathbb{R}^6 \to \mathbb{R}$ defined as

$$h(\boldsymbol{z}) = D_s^2 - ||\boldsymbol{z}_1||^2 = D_s^2 - \boldsymbol{z}_1^\mathsf{T}\boldsymbol{z}_1, \tag{5.14}$$

where $D_s > 0$ is the minimum distance between $\boldsymbol{r}_r(t)$ and the obstacles. Using this function we can define the safe set $\mathcal{C}_0$ as the largest tubular region around $\boldsymbol{r}_r(t)$ that does not collide with the obstacles. In mathematical form we have

$$\mathcal{C}_0 = \{\boldsymbol{z} \in \mathbb{R}^6 | h(\boldsymbol{z}) \geq 0\}. \tag{5.15}$$

Using the theory described in [1] we can prove that $\mathcal{C}_0$ is invariant if h is a control barrier function, in this case an exponential control barrier function. This happens if there exist a row vector $\boldsymbol{K}_\alpha = [\alpha_1, \alpha_2]$ such that $\forall \boldsymbol{z} \in \mathrm{Int}(\mathcal{C}_0)$

$$\sup_{\boldsymbol{\sigma}\in\Theta} \left(L_f^2 h(\boldsymbol{z}) + L_g L_f h(\boldsymbol{z})\boldsymbol{\sigma} \right) \geq -\boldsymbol{K}_\alpha \boldsymbol{\eta}(\boldsymbol{z}), \tag{5.16}$$

and $h(\boldsymbol{z}(t)) \geq \boldsymbol{C}e^{(\boldsymbol{A}-\boldsymbol{B}\boldsymbol{K}_\alpha)t}\boldsymbol{\eta}(\boldsymbol{z}_0) \geq 0$ whenever $h(\boldsymbol{z}_0) \geq 0$, where

$$\boldsymbol{\eta}(\boldsymbol{z}) = \begin{bmatrix} h(\boldsymbol{z}) \\ L_f h(\boldsymbol{z}) \end{bmatrix},$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad \boldsymbol{C} = [1, 0].$$

By explicit computation we get

$$\boldsymbol{\eta}(\boldsymbol{z}) = \begin{bmatrix} D_s^2 - ||\boldsymbol{z}_1||^2 \\ -2\boldsymbol{z}_1^\mathsf{T}\dot{\boldsymbol{z}}_1 \end{bmatrix}$$

and the (5.16) as

$$-2(\dot{\boldsymbol{z}}_1^\mathsf{T}\dot{\boldsymbol{z}}_1 + \boldsymbol{z}_1^\mathsf{T}\dot{\boldsymbol{z}}_2) \geq -\boldsymbol{K}_\alpha\boldsymbol{\eta}(\boldsymbol{z}). \tag{5.17}$$

After further algebraic manipulations, we can rearrange the terms of (5.17) as

$$\begin{aligned} &-2\dot{\boldsymbol{z}}_1^\mathsf{T}\dot{\boldsymbol{z}}_1 - 2\boldsymbol{z}_1^\mathsf{T}(-\boldsymbol{M}_v^{-1}\boldsymbol{K}_v\boldsymbol{z}_1 - \boldsymbol{M}_v^{-1}\boldsymbol{D}_v\dot{\boldsymbol{z}}_1 + \boldsymbol{M}_v^{-1}\boldsymbol{f}_v) \geq \\ &\qquad\qquad - \alpha_1(D_s^2 - ||\boldsymbol{z}_1||^2) + 2\alpha_2(\boldsymbol{z}_1^\mathsf{T}\dot{\boldsymbol{z}}_1) \\ &\alpha_1 D_s^2 - 2(||\boldsymbol{z}_2||^2 + \boldsymbol{z}_1^\mathsf{T}\boldsymbol{M}_v^{-1}\boldsymbol{f}_v) + \\ &\qquad + (2\boldsymbol{z}_1^\mathsf{T}\boldsymbol{M}_v^{-1}\boldsymbol{K}_v - \alpha_1\boldsymbol{z}_1^\mathsf{T})\boldsymbol{z}_1 + 2(\boldsymbol{z}_1^\mathsf{T}\boldsymbol{M}_v^{-1}\boldsymbol{D}_v - \alpha_2\boldsymbol{z}_1^\mathsf{T})\dot{\boldsymbol{z}}_1 \geq 0. \end{aligned}$$

Now, the assumption that $\boldsymbol{M}_v = \mathcal{M}_v\boldsymbol{I}_{3\times3}$, $\boldsymbol{K}_v = \text{diag}\{K_{v,1}, K_{v,2}, K_{v,3}\}$ and $\boldsymbol{D}_v = \text{diag}\{D_{v,1}, D_{v,2}, D_{v,3}\}$ is an acceptable restriction for the controller behavior since it means to have the same virtual mass in all directions of motion and to avoid reactions along one axis as a response to motions along the others. Using this assumption, the third term is non-negative if

$$\alpha_1 \leq \frac{2}{\mathcal{M}_v} \cdot \frac{K_{v,1}z_{1,1}^2 + K_{v,2}z_{1,2}^2 + K_{v,3}z_{1,3}^2}{||\boldsymbol{z}_1||^2}, \tag{5.18}$$

which is verified if the following stronger condition holds:

$$\alpha_1 \leq \frac{2}{\mathcal{M}_v} \cdot \min\{K_{v,1}, K_{v,2}, K_{v,3}\}.$$

In a similar way the fourth term is non-negative if

$$\alpha_2 \leq \frac{1}{\mathcal{M}_v} \cdot \frac{D_{v,1}z_{1,1}\dot{z}_{1,1} + D_{v,2}z_{1,2}\dot{z}_{1,2} + D_{v,3}z_{1,3}\dot{z}_{1,3}}{\boldsymbol{z}_1^\mathsf{T}\dot{\boldsymbol{z}}_1}. \tag{5.19}$$

To verify this inequality we can introduce another acceptable restriction on the controller behavior by assuming that $\boldsymbol{D}_v = D_{v,c}\boldsymbol{I}_{3\times3}$, which means having the same damping gain in all directions. In this way the (5.19) is satisfied if

$$\alpha_2 \leq \frac{D_{v,c}}{\mathcal{M}_v}.$$

At this point we can focus on the sign of the first two terms to ensure that h is an exponential control barrier function. A simple solution is to consider the virtual external force

$$\boldsymbol{f}_v = -k_{v,p}\boldsymbol{z}_1 - k_{v,d}\frac{||\boldsymbol{z}_2||^2}{||\boldsymbol{z}_1||^2}\boldsymbol{z}_1, \tag{5.20}$$

where $k_{v,p}, k_{v,d}$ are positive constants. We will have that the first two terms are non-negative if

$$||\boldsymbol{z}_2||^2\left(\frac{k_{v,d}}{\mathcal{M}_v} - 1\right) + \frac{k_{v,p}}{\mathcal{M}_v}||\boldsymbol{z}_1||^2 \geq -\frac{\alpha_1}{2}D_s^2. \tag{5.21}$$

This condition is necessarily verified as long as $k_{v,d} \geq \mathcal{M}_v$ and $\alpha_1 \geq 0$, which is consistent with the requirement (5.18). To summarize, with this analysis we have shown that there exist suitable conditions under which the algorithm proposed guarantees safety. It is important to remark that these conditions were derived in a very conservative way and so it is possible that safety may be guaranteed also with less strict assumptions on the controller design. It is worth to mention that the virtual external force $\boldsymbol{f}_v$, even if it may seem quite different from the force $-\boldsymbol{f}_r$ considered in Section 5.2 and 5.3, it is instead strongly correlated with the virtual repulsive force used in our method. In fact we have that $\boldsymbol{f}_v$ depends on the error from the reference trajectory $\boldsymbol{z}_1$, while $\boldsymbol{f}_r$ depends on the distance from the obstacles $\eta_i(\boldsymbol{r})$. Intuitively it is reasonable to have a correlation between these two quantities since in general the further we get from $\boldsymbol{r}_r$, the closer to an obstacles we get. On the other hand proving this formally in a general way is not straightforward, what we can do is instead to consider a particular case.

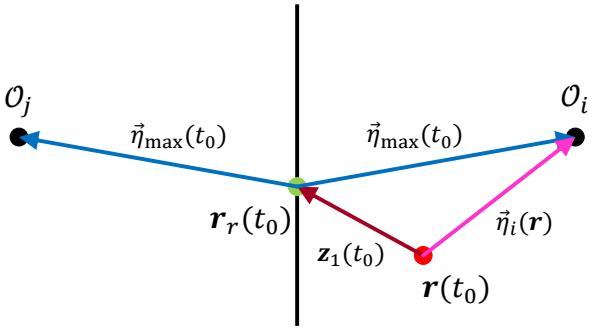

Figure 5.3. Graphical representation of the correlation between $\boldsymbol{z}_1$ and $\vec{\eta}_i$.

Let us assume to have computed the reference trajectory from the Voronoi diagram, thus at time t_0 we known that the reference point $\boldsymbol{r}_r(t_0)$ will be at the maximum distance possible $\eta_{\max}(t_0)$ from the nearest obstacles. Assuming that from the actual robot position $\boldsymbol{r}(t_0)$ the robot detects only one fixed point obstacle $\mathcal{O}_i$, the distance from it will be $\eta_i(\boldsymbol{r}(t_0))$. Now if we consider the vectors $\vec{\eta}_i, \vec{\eta}_{\max}$ representing these distances as sketched in Fig. 5.3, we have that $\boldsymbol{z}_1 = \vec{\eta}_i - \vec{\eta}_{\max}$. Making the substitutions in the (5.20) and rearranging the terms we obtain that

$$-\boldsymbol{f}_v(\boldsymbol{r}(t)) = (\vec{\eta}_i(\boldsymbol{r}) - \vec{\eta}_{\max}(t)) \left(k_{v,p} + k_{v,d} \frac{||\dot{\boldsymbol{r}}_d - \dot{\boldsymbol{r}}||^2}{||\vec{\eta}_i(\boldsymbol{r}) - \vec{\eta}_{\max}(t)||^2} \right). \tag{5.22}$$

This repulsive force corresponds, even if with some minor differences, to the repulsive force (5.6) used in our method. In particular in this case we have that the obstacle range of influence is $\eta_{\max}(t)$, so it is not fixed and is derived from the Voronoi diagram. Then we have that the repulsive force depends linearly instead of quadratically on the difference between the distance from the obstacle and the range of influence. To conclude also the velocity scaling is present. The main difference is that in this case it is done by adding a scalar quantity to the coefficient $k_{v,p}$, while in (5.7) it is done with a multiplication.

Chapter 6

Simulation results

The validation of the proposed method has been done using a PC simulation developed with CoppeliaSim (v. 4.2.0), a simulation software well known in the field of robotics. In this chapter we describe the experimental setup used in the simulations and the different scenes considered. Then, by analyzing the data collected, we will make some considerations about the performance of our method highlighting the advantages with respect to a standard trajectory tracking controller.

6.1 Simulation setup

The CoppeliaSim software works using scenes representing the environment in which the experiment runs. In this case all the simulation scenes contain a symmetric quadrotor, with the plausible physical characteristics shown in Table 6.1, and some obstacles of different shapes. The task that we will consider in each scene is always the one described in Chapter 5, namely the navigation of the quadrotor from a point of the environment to another while avoiding the obstacles.

Table 6.1. Physical parameters

Physical property	Value
Mass	0.5 kg
Moment of inertia (around x, y)	1.8×10^{-3} kg·m^2
Moment of inertia (around z)	3.3×10^{-3} kg·m^2
Size	$35 \times 30 \times 4$ cm

Since we are interested in validating the method in nominal conditions, the simulation contains some simplifications with respect to the real world. First of all the aerodynamics is ignored and the quadrotor motion is computed only by considering the force and torque applied to the robot. The actuation is done using directly the input commands $\boldsymbol{u} = [T, \tau_\varphi, \tau_\vartheta, \tau_\psi]^\mathsf{T}$, in this way the low level control of the rotors torque can be ignored. Then, to reduce the computational complexity of collision checking, the quadrotor body is assumed to be a uniform parallelepiped-like convex shape. For what concerns the obstacles, they are defined as non-respondable objects without any dynamics, this means that they can be traversed by the robot without affecting its motion. The number of obstacles, their position and their

dimensions vary from scene to scene as will be discussed later. From an aesthetic point of view, the scenes are created trying to reproduce plausible indoor and outdoor working scenarios using simple 3D objects. The color of each obstacles changes when a collision happens, in this way it is easier to observe when a contact occurs.

The experiments runs are saved as mp4 videos, while all the relevant physical quantities are saved in a csv file and will be reported as plots. In all the experiments done the variables stored are the reference trajectory $\boldsymbol{r}_r(t)$, the actual trajectory $\boldsymbol{r}(t)$, the input commands $\boldsymbol{u}$, the repulsive force $\boldsymbol{f}_r$ and the distance $\eta_{\min}$ from the quadrotor body to the nearest obstacle. It has to be remarked that the distance $\eta_{\min}$ is different from the distance $\eta_i(\boldsymbol{r})$ used for the repulsive force computation since the latter is computed from the quadrotor center. By monitoring if the distance $\eta_{\min}$ reaches the value zero, we are able to identify when a contact happens.

Going more in the details of the simulation logic, the quadrotor body has associated a child script containing all the Lua code needed for controlling the robot motion. By changing the values of some variables of the script we can select which control strategy to use from the ones available. It is important to remark that the robot state used as feedback in all the controllers developed is not affected by error since it is obtained directly from the scene state. The tuning of the controllers has been done so to allow a meaningful comparison between the different strategies, therefore their behavior is very similar in terms of response time and control effort in absence of external forces. The details of the main software components implemented inside this script are reported in the sections below.

Trajectory planner

The first operation performed during the simulation is always the computation of a safe path from the robot initial position to the goal with the RRT-Connect algorithm [10] described in Section 5.1. This computation is done using the OMPL plugin of CoppeliaSim that simply required the start and goal positions, the obstacles to avoid and the state space region in which the path has to be contained. These parameters must be modified according to the scene characteristics. Given the solution path, the constant speed reference trajectory is then generated using again the functions of the OMPL plugin. After this operation we enter in the loop phase of the simulation during which the reference trajectory value at a given time-step can be obtained calling the `get_ref_traj()` function. Inside this function is also defined the yaw angle reference trajectory $\psi_r(t)$, which in our experiments is always fixed to zero.

Hierarchical PD controller

The first control strategy available for trajectory tracking is the simple hierarchical PD control [6] presented in Section 3.2. In particular, by setting the variable `control_type=1`, the control will be obtained using the PD control (3.8) and the attitude control (3.6) in cascade. This control strategy is for sure the simplest and obviously it can not take into account the distance from the obstacles during navigation.

Controller in SE(3)

The second controller developed is the [12], see (3.10) and (3.11), it is more general since works directly in SE(3), but also in this case it ignores the obstacles during navigation. To use this controller we have to set `control_type=2`. Due to the complexity of the vectors and matrices operations required for this control, most of the computations have been implemented in a MATLAB script connected to CoppeliaSim via remote API calls. In particular the input of the MATLAB script are the reference trajectory and the robot state, while the output are the quadrotor commands $\boldsymbol{u}$. This solution slows down a bit the simulation speed, but simplifies a lot the Lua code.

Impedance controller

The last type of control available is used when `control_type=3` and is the impedance control (5.9) [26]. Likewise for the controller in SE(3), it is implemented in MATLAB to simplify the Lua code. The difference in this case is that the MATLAB script receives as input also the external forces $\boldsymbol{f}_e$ and $\boldsymbol{f}_r$ and returns as output the virtual input force $\boldsymbol{f}_d$. This force is then converted into thrust and torque commands inside the Lua script as described in Section 5.4. It has to be remarked that the attitude controller used is exactly the same of the hierarchical PD control.

Repulsive force computation

The last important software component implemented is the repulsive force computation. With the value of the parameter `force_type` we can choose to not produce any repulsive force, to compute it using (5.3) and (5.4) or using (5.6) and (5.7). The value of η_0 and k_r of each method are chosen so to maximize the distance $\eta_{\min}$ during the trajectory tracking, further considerations about their values can be found in the next sections. The minimum distances from the obstacles $\eta_i(\boldsymbol{r})$ are computed at each time-step from the scene state using the `checkDistance()` function available in CoppeliaSim.

6.2 Experiments and results

After having presented the experimental setup, we can now discuss the results obtained by our method in three different CoppeliaSim scenes. All the scenes have been created to represent some realistic scenarios in which safe navigation is challenging. For each scene will be discussed the main advantages and disadvantages in the use of a repulsive force with respect to the standard trajectory tracking controllers. In order to have a meaningful comparison between the different methods, we will use the same reference trajectory in each scene. Since the safe trajectory obtained using a simple probabilistic planner is in general unfeasible, it is reasonable to expect that the tracking task will not be executed at best by the standard controllers.

6.2.1 Scene 1

The first scene that we will consider represents the inside of an office, it contains 14 obstacles, in particular 8 walls and 6 furnishings as shown in Fig. 6.1. The quadrotor starts from the floor of a room under a table and has to reach the center of another room. In this scene we will consider the control in SE(3) with respect to our method with standard repulsive force (SRF) or with velocity-based repulsive force (VRF). The hierarchical PD control and the impedance control without any repulsive force are ignored in this scene since their behavior is almost the same as the control in SE(3), which is instead more general.

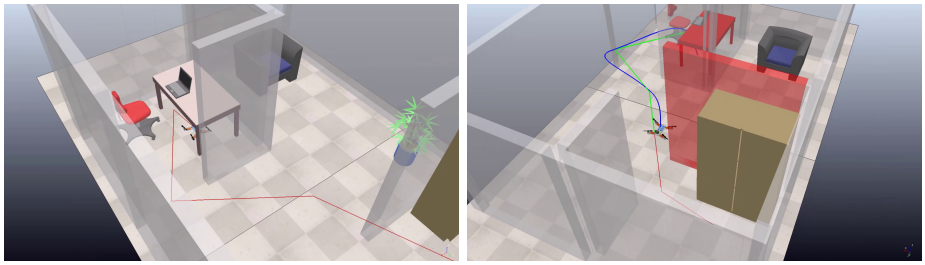

Figure 6.1. Graphical rendering of scene 1 during execution with control in SE(3).

By making a comparison between the tracking errors (Fig. 6.2) we can notice that for the control in SE(3) the error raises in the point of discontinuity of the reference trajectory, while the other two methods sometimes deviate from the reference trajectory to avoid some obstacles. The amplitudes of the error peaks are anyway similar and the same goes for the cumulative errors. What is more relevant to consider is instead the minimum distance from the obstacles $\eta_{\min}$. From Fig. 6.3 we can instantly notice that the two contacts occurred while using the control in SE(3) are avoided with our method. In addition also the dangerous passage done by the standard control after around 10 seconds is substituted with a safest trajectory farther from the wall. From these plots is difficult to highlight some differences

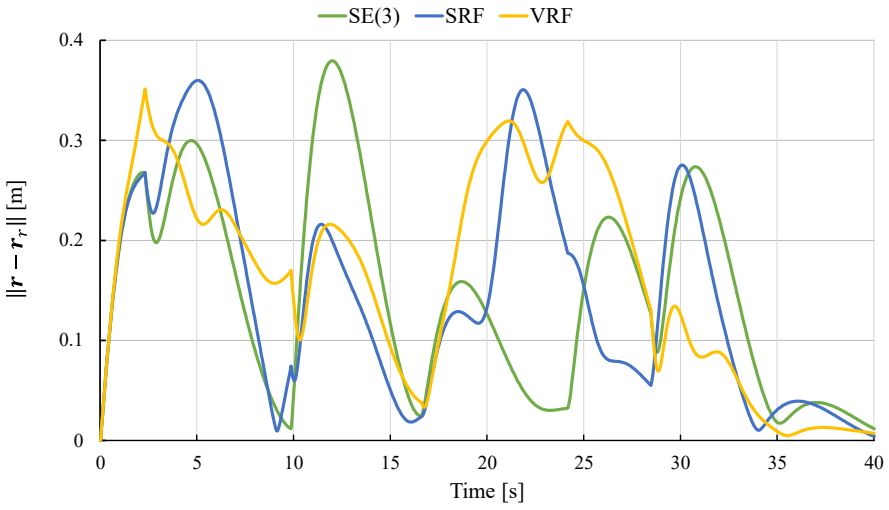

Figure 6.2. Scene 1 tracking error plot comparison.

between the two types of repulsive force since in some points one performs better and in others happens the opposite, but if we look at Fig. 6.4 it is a different story. The SRF reveals a more aggressive behavior with short peaks of higher amplitude with respect to the VRF, in fact the latter is smoothed by the obstacles approaching speed. For sure the choice of the parameters η_0 and k_r can make a difference, but is important to remark that each type of repulsive force has different needs to perform well. For example the SRF, that raises rapidly when approaching an obstacle, has to use small η_0 so to react only when very close to contact. In addition, if η_0 is too large, when the robot is close to many obstacles the total repulsive force can become huge since the approaching velocity is not taken into account. On the contrary, when using the VRF we do not have this problem. By setting an higher value of η_0 we simply ensure to react softly and in time when a contact can occur. The aggressive behavior of the SRF is also confirmed by looking at the control effort. Even if the

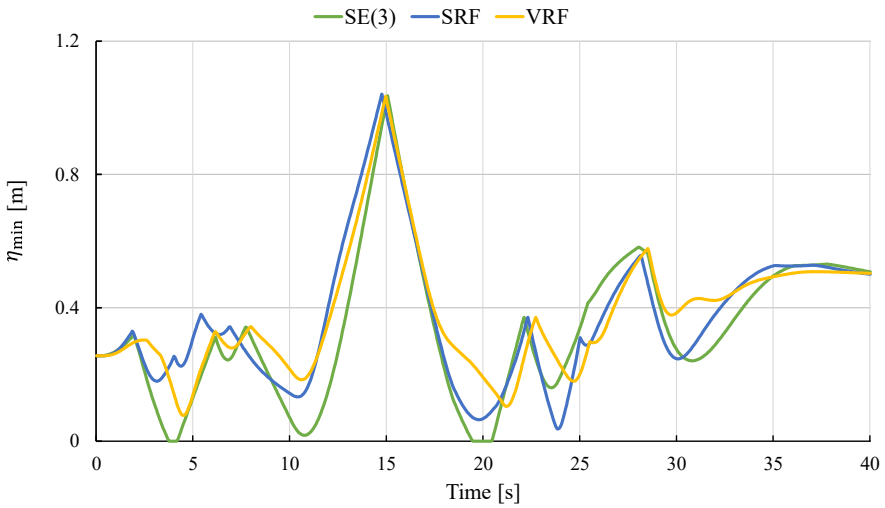

Figure 6.3. Scene 1 minimum distance from the obstacles plot comparison.

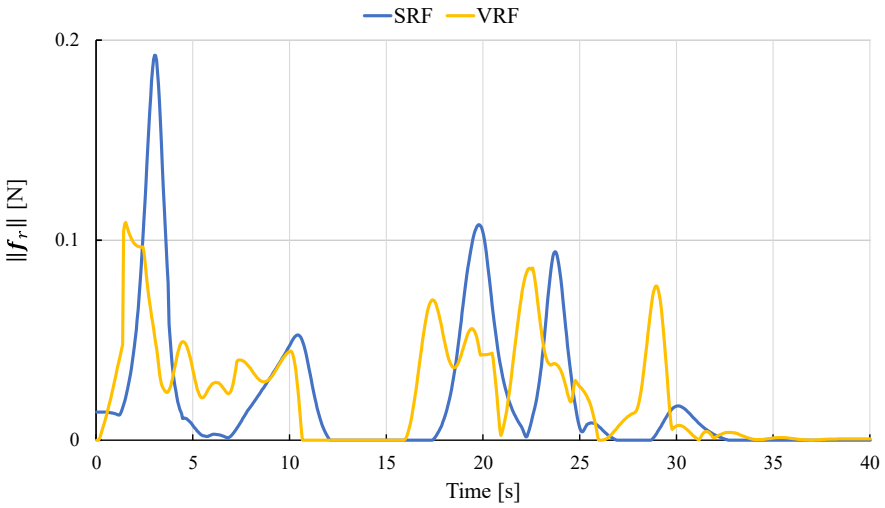

Figure 6.4. Scene 1 repulsive force plot comparison.

overall effort is quite similar between the three methods, we can notice from Fig. 6.5 that the thrust command has some higher peaks when using the SRF and the same goes for the torque commands.

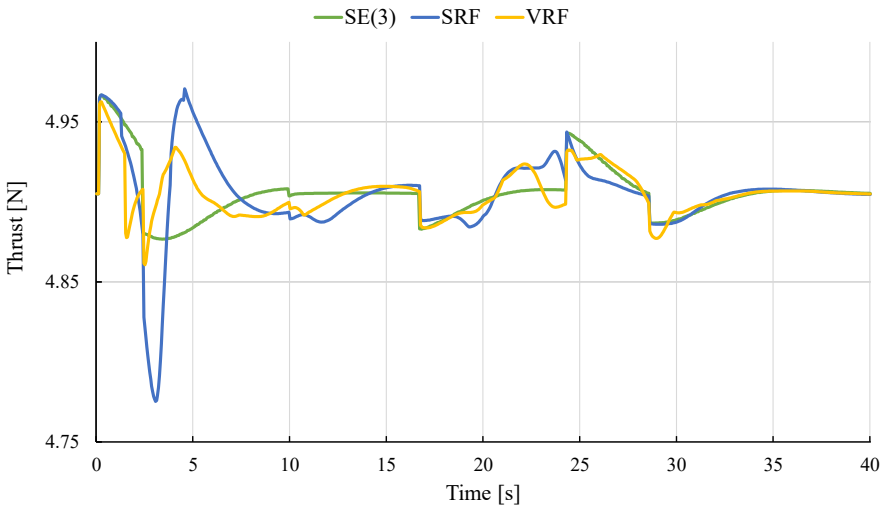

Figure 6.5. Scene 1 thrust command plot comparison.

Focusing on the safety considerations done in Section 5.5, we can notice from the plot of the control barrier function $h(\boldsymbol{z})$ (Fig. 6.6) that with both types of control the quadrotor exits from the safe set multiple times. The difference is that with our method the robot moves away from the reference trajectory to avoid the obstacles. In fact, even if staying inside the small volume of $\mathcal{C}_0$ is safe, the repulsive force used tends to push the robot even further to stay away from the obstacles. This can be seen looking at the contacts that are avoided after 4 and 20 seconds. On the contrary, with the control in SE(3) the robot exits from the safe set only when it crosses the points of discontinuity of the geometric trajectory. For example after 11 seconds, due to the execution of a sharp turn, the robot exits from the safe set

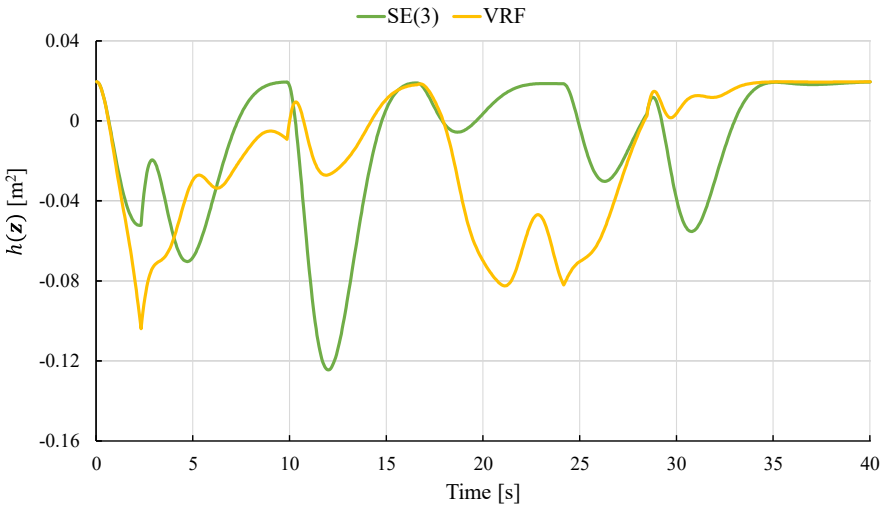

Figure 6.6. Scene 1 control barrier function plot comparison.

and passes close to a wall. With our method, instead, the repulsive force pushes the quadrotor away from the wall and consequently closer to the initial safe trajectory.

6.2.2 Scene 2

The second scene (Fig. 6.7) represent an outdoor environment with 15 obstacles of various types. Also in this case the quadrotor starts from the floor and has to pass inside the hole of an obstacle. In addition at the beginning of the simulation, from second 6 to 8, a disturbance force is applied to the UAV making the trajectory tracking task more difficult. In this scene we are going to highlight the differences between the control in SE(3), the standard impedance control (IMP) that reacts only to real external forces and ignores the repulsive one and our method with the velocity-based repulsive force (VRF).

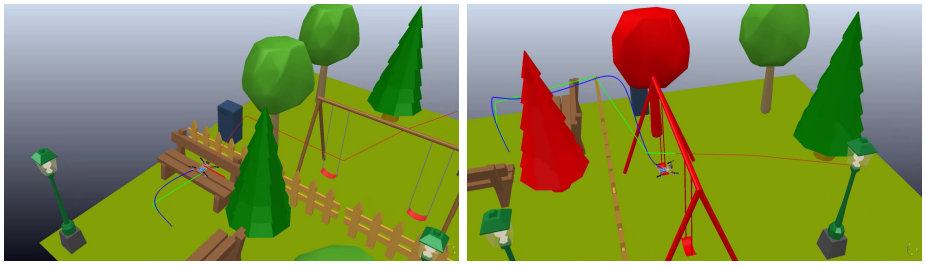

Figure 6.7. Graphical rendering of scene 2 during execution with control in SE(3).

From Fig. 6.8 we can instantly notice how similar the impedance control and the control in SE(3) are in absence of disturbances, while a big difference is visible when the external force is applied. As expected the standard impedance control and our method react to the disturbance keeping limited the error from the reference trajectory. On the other hand, looking at Fig. 6.9, we can conclude that, even if the impedance control avoided the first collision after 8 seconds, the other two happened anyway. Instead with our method and the more reliable VRF the quadrotor navigates

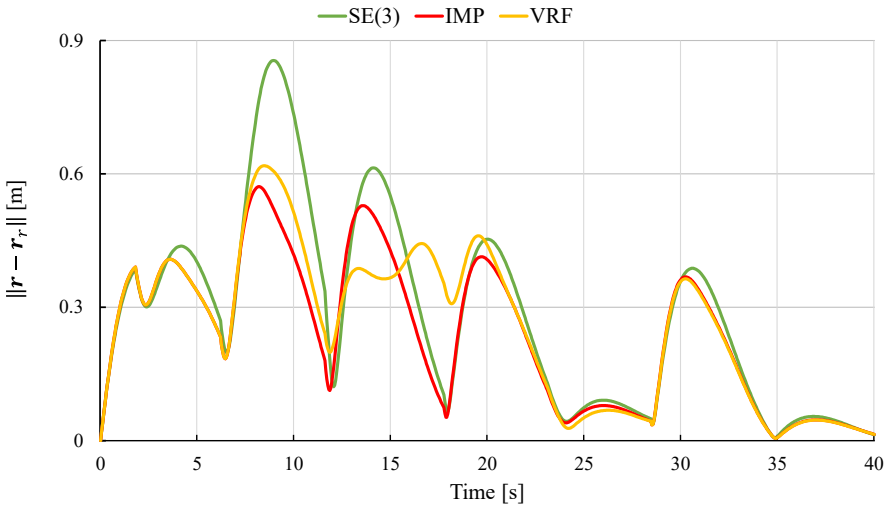

Figure 6.8. Scene 2 tracking error plot comparison.

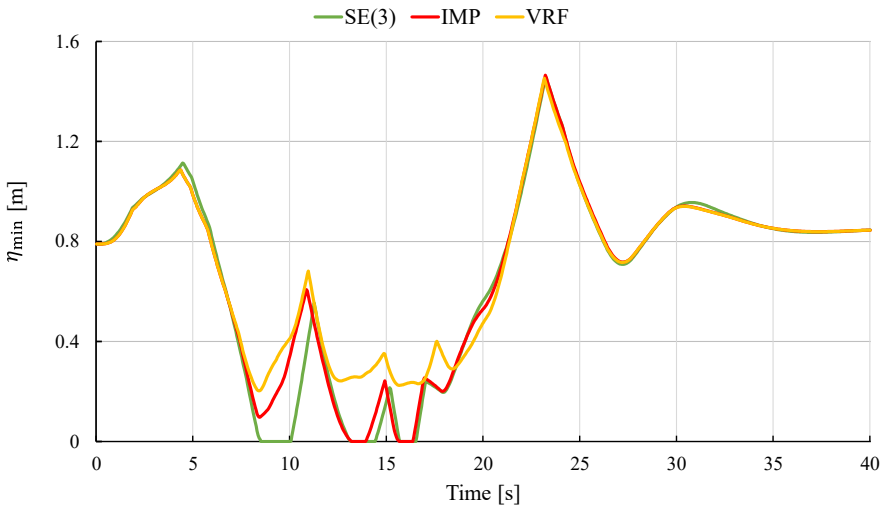

Figure 6.9. Scene 2 minimum distance from the obstacles plot comparison.

safely. For what concerns the control effort there are not significant differences between these strategies and so we do not report the plots here. The same goes for the evolution of the repulsive force which is similar to Scene 1.

6.2.3 Scene 3

The third scene considered (Fig. 6.10) is very similar to the previous one, in particular it uses the same obstacles but with a slightly modified arrangement. The quadrotor has different start and goal points and there are not external disturbances, but the main variation is that in this case we have an additional moving obstacle that is not taken into account in the planning phase and can only be detected online. The obstacle motion is chosen so to interfere with the reference trajectory after 5 seconds from the start. This choice was done to try to reach the limits of our method since keeping the quadrotor safe in this scenario is very challenging. Given the difficulty of this scene, in this case we will consider only the most promising velocity-based repulsive force with respect to the standard control in SE(3) to focus, among other things, on the control effort required to prevent the contact with these type of obstacles.

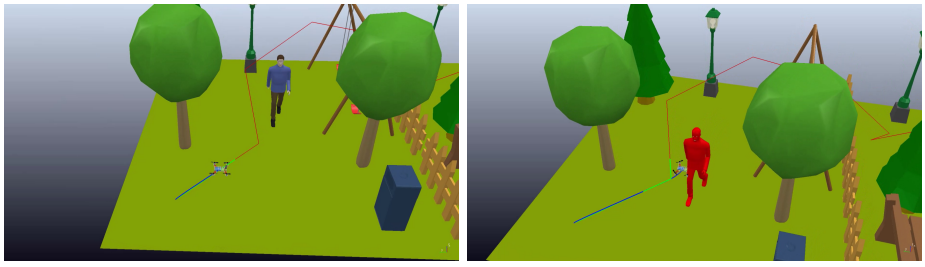

Figure 6.10. Graphical rendering of scene 3 during execution with control in SE(3).

Note that, since our method was initially developed for fixed obstacles, a little modification was required to deal with moving obstacles. In particular we had to

modify the (5.7) so to take into account not only the robot linear velocity, but also the obstacles velocities $\boldsymbol{v}_{\text{ob,i}}$ so to make the control more aware of the potentiality of a collision. The computation of the total repulsive force (5.7) has been therefore substituted with

$$\boldsymbol{f}_r(\boldsymbol{r}) = \sum_{i=1}^{m} \max\left\{ \frac{\nabla\eta_i(\boldsymbol{r})^{\mathsf{T}} \cdot (\dot{\boldsymbol{r}} - k_m \boldsymbol{v}_{\text{ob,i}})}{\eta_i(\boldsymbol{r})}, 0 \right\} \cdot \boldsymbol{f}_{r,i}(\boldsymbol{r}), \tag{6.1}$$

where k_m is a positive constant. The introduction of the constant k_m allows to make a more precise tuning of the repulsive force characteristics, since using only the parameters k_r and η_0 it is almost impossible to find a behavior that works well for both fixed and moving obstacles. The main problem is that we would like to have an aggressive response for the moving obstacles to react in time, but with fixed obstacles this choice makes overestimate the threat of a collision. With the introduction of k_m it is possible to balance these two aspects.

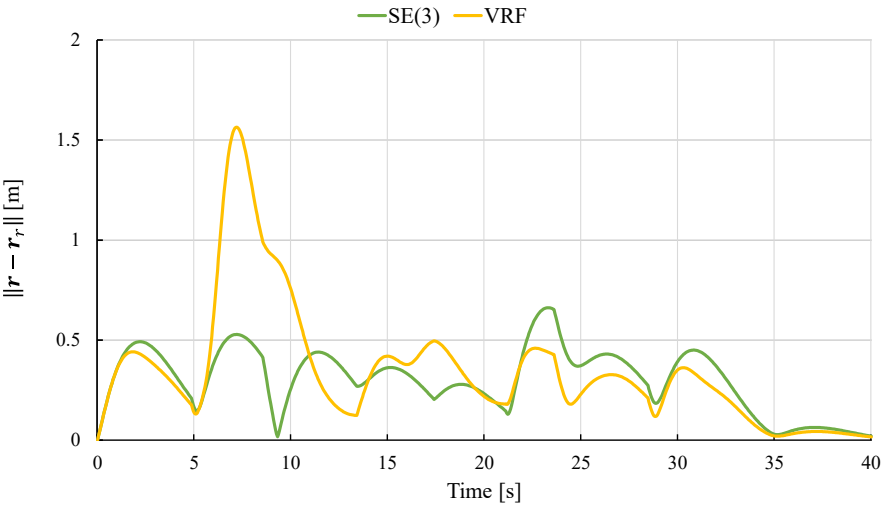

Figure 6.11. Scene 3 tracking error plot comparison.

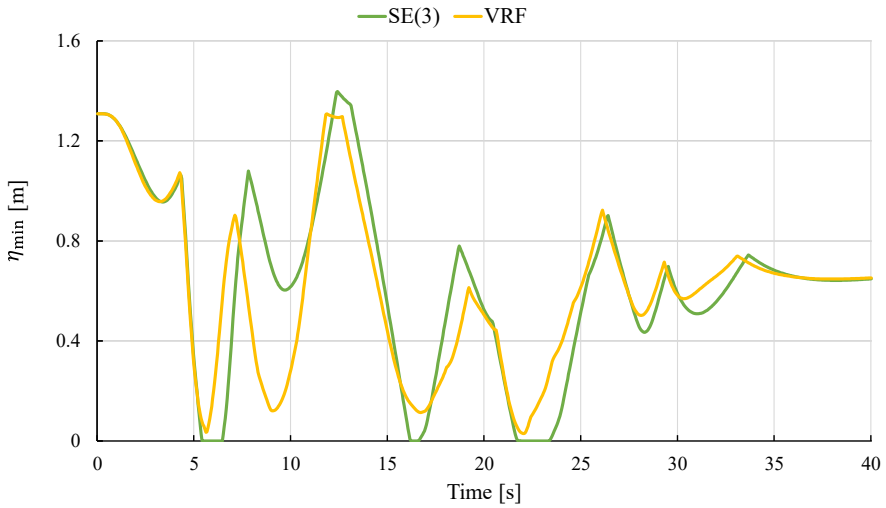

Figure 6.12. Scene 3 minimum distance from the obstacles plot comparison.

Now, looking at the experiments results, we can see from Fig. 6.11 that, to avoid the contact with the moving obstacle, the robot has to deviate a lot from the reference trajectory, while with the other fixed obstacles the behavior is very similar to the previous scenes. This is due to the fact that the quadrotor maneuver necessary to prevent the collision needs to be very aggressive. In any case we can notice from Fig. 6.12 that with our method the three collisions happened with the standard controller have been all avoided. On the other hand the contact with the moving obstacle has been averted only for few centimeters.

As an additional proof that the maneuver executed by the quadrotor to avoid the moving obstacle is very aggressive, we can look at Fig. 6.13. The intensity of the repulsive force required to move the robot away from the moving obstacle was almost three times the one required for the fixed obstacles, but obviously can be even higher if the approaching velocity raises. It has to be remarked that the presence of peaks of this intensity in the repulsive force tend to destabilize the control. This because

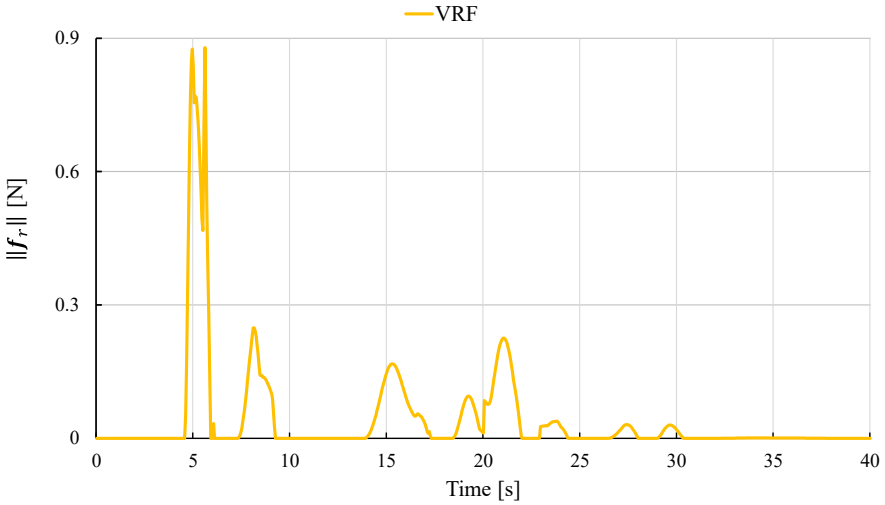

Figure 6.13. Scene 3 repulsive force plot.

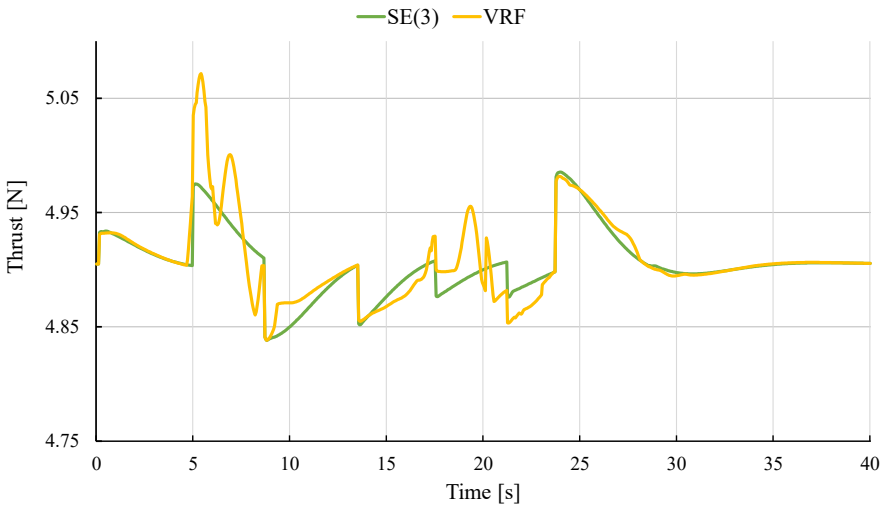

Figure 6.14. Scene 3 thrust command plot comparison.

they produce violent motions that can drive the aircraft very far from the reference trajectory and even towards other obstacles. The fast approaching speed toward another obstacle causes again peaks in the repulsive force making the final robot trajectory almost unpredictable. In this specific case the large deviation from the reference trajectory did not cause such a problem, even the thrust peaks (Fig. 6.14) were acceptable, but the risk was present. In fact if we look again at Fig. 6.12 we can notice how after around 9 seconds the quadrotor gets close to an obstacle that for the standard control in SE(3) was not even a threat. This dangerous behavior can be accentuated also by the choice of the repulsive potential. For example in this scene the use of the standard repulsive force (5.3), that raises up to infinity when approaching an obstacle, is enough to make the aircraft lose control and crash. So we can conclude that a smooth-rising repulsive force is always preferable.

Chapter 7

Conclusion

In this thesis we have developed and validated an innovative method for the safe navigation of UAVs in obstacle-rich environments. The algorithm has been implemented and tested only on a simulated quadrotor, but the adaptation to other type of UAVs is quite straightforward. The two phase strategy developed has the advantage of reducing the computational complexity both in the offline planning phase and in the online reactive navigation even after modifications of the environment. The use of a simple probabilistic planner to compute a non-feasible trajectory simplifies the path planning operation, while the use of an impedance control and a virtual repulsive force allows to smooth the reference trajectory and react to deviations from the safe region with a minimal computational cost. An additional advantage of this method is the possibility of adaptation to partially unknown or mutating environment only using on-board sensors.

With the results presented in Chapter 6 it was possible to prove the reliability of our method. In all the scenes tested the collisions happened using standard controllers have been always avoided by our controller with a comparable control effort. In particular, as predicted in the considerations done in Section 5.5, the use of a velocity-based repulsive force demonstrated to be the preferable option to obtain smoother and safer robot motions. This is mainly due to the importance of the approaching speed when predicting the potentiality of a collision. It is intuitive that an obstacle from which the UAV is moving away is not a threat, while an obstacle in the direction of motion, even if farther, is more dangerous.

Another key factor to obtain good performances turned out to be the choice of the parameters in the computation of the virtual repulsive force. In the specific, the value of the obstacles range of influence η_0 has to be chosen large enough to ensure a reaction in time when the UAV is close to a collision, but small enough to ignore the threat of a contact with objects too far from the robot trajectory, so to not destabilize too much the control. In addition, from the experiments emerged that the use of a velocity-based repulsive force allows to set an higher value of η_0 without compromising the control stability. This is due to the fact that taking into account the obstacles approaching speed mitigates the aggressiveness of the reactions to avoid them. Only after having selected a good value for η_0, we can focus on the choice of k_r. This parameter changes the aggressiveness of the reaction since defines how fast the repulsive force raises when approaching an obstacle.

Besides the tuning of these parameters, a more radical modification that may improve the performance of our method concerns the computation of the repulsive force. The options are many, the simplest one is the use of different values of η_0 and k_r for each obstacle, or even only for fixed and moving obstacles, as highlighted in Section 6.2.3. Also the use of other types of repulsive potential fields can make a big difference, in this thesis we considered only the (5.2) and the (5.5), but the possibilities are almost unlimited. Going even further an attitude impedance control could be added to the position impedance control allowing to handle external wrenches. In this way it would be possible to make the obstacles modify directly the UAV attitude, in addition to the position, through the computation of a virtual repulsive wrench $\boldsymbol{\tau}_r \in \mathbb{R}^6$. A solution of this kind could lead to the execution of very aggressive maneuvers to avoid collisions, included the ones that drive the UAV in vertical poses.

Obviously also the values of the gain matrices of the impedance control make a big difference. The desired stiffness matrix $\boldsymbol{K}_v$ is correlated with the importance that we want to give to the reference trajectory, the more close to $\mathbf{0}_{3\times3}$ is set, the less attracted to the reference trajectory the robot is. For example in a quite unknown environment it could be useful to rely more on the repulsive force than the initial reference trajectory. The desired damping matrix $\boldsymbol{D}_v$ has a similar behavior but on the reference velocity. Higher values inside this matrix lead to sharper turns in the point of discontinuity of the geometric path. On the other hand, to do this, they generates violent robot motions, an aspect that has to be considered if there are bounds on the actuators torque. The virtual mass $\mathcal{M}_v$ is instead related with the impact that the repulsive force has on the UAV motion. With big values of this parameter even a weak repulsive force entails a large motion, while the contrary happens with small values. The choice of this parameter has to be balanced with the repulsive force definition, in particular with the choice of k_r value, since it has a similar effect on the UAV behavior.

For what concerns the evaluation of our method, it would be interesting to make some additional experiments. Clearly the most important would be a test on a real quadrotor, with the online phase based on the detection of the obstacles made by a real sensor, such as a LIDAR, to understand if the control is reliable in a real partially unknown environment. Another interesting analysis could be the comparison with the safe navigation strategies available in the literature, such as [19], [29] or [14]. It would be crucial to understand if the reduced computational complexity of our method allows anyway to reach performances comparable to these methods and if, in some particular conditions, our approach works even better. From a more theoretical prospective, a precious improvement that could be done concerns the safety guarantees, since the proof described in Section 5.5 is quite far from proving the safety of our method in general conditions. The choice of less conservative assumptions or the use of alternative techniques, in place of control barrier functions, may allow to finally prove the safety of our control.

In conclusion the method described in this thesis showed to have great potential and, even if there is room of improvement, the core idea remains very valuable for UAVs and maybe also for other types of mobile robots. In general in unstructured environment, the most common ones, the implementation of reactive navigation strategies is crucial to handle at best unexpected situations. The minimal com-

putational complexity required combined with the use of on-board sensors make this method particularly suitable for low-budged multirotors operating in this type of environments, for example in the fields of security or video making. From the industries point of view, the main advantage is given by the fact that many of the UAVs available on the market already have on board some obstacle detection sensors and so the implementation of our method could be done through a simple software update. In addition, with some adaptations, the online phase of our method may be used also as a flight assistance feature that keeps the aircraft safe even if the pilot is not particularly skilled.

Bibliography

[1] Aaron D Ames et al. “Control barrier functions: Theory and applications”. In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 3420–3431.

[2] Federico Augugliaro and Raffaello D’Andrea. “Admittance control for physical human-quadrocopter interaction”. In: *2013 European Control Conference (ECC)*. IEEE. 2013, pp. 1805–1810.

[3] İ Can Dikmen, Aydemir Arisoy, and Hakan Temeltas. “Attitude control of a quadrotor”. In: *2009 4th International Conference on Recent Advances in Space Technologies*. IEEE. 2009, pp. 722–727.

[4] *DJI Mavic 2 Pro*. Available: `https://store.dji.com/it/product/mavic-2`. Oct. 2021.

[5] Matteo Fumagalli et al. “Modeling and control of a flying robot for contact inspection”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 3532–3537.

[6] Gabriel Hoffmann, Steven Waslander, and Claire Tomlin. “Quadrotor helicopter trajectory tracking control”. In: *AIAA guidance, navigation and control conference and exhibit*. 2008, p. 7410.

[7] Neville Hogan. “Impedance control: An approach to manipulation: Part I—Theory”. In: (1985).

[8] Wolfgang Hönig et al. “Trajectory planning for quadrotor swarms”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 856–869.

[9] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[10] James J Kuffner and Steven M LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.

[11] Steven M LaValle et al. “Rapidly-exploring random trees: A new tool for path planning”. In: (1998).

[12] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. “Geometric tracking control of a quadrotor UAV on SE (3)”. In: *49th IEEE conference on decision and control (CDC)*. IEEE. 2010, pp. 5420–5425.

[13] Sikang Liu et al. “Search-based motion planning for aggressive flight in se (3)”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2439–2446.

[14] Sikang Liu et al. “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.

[15] Sikang Liu et al. “Towards search-based motion planning for micro aerial vehicles”. In: *arXiv preprint arXiv:1810.03071* (2018).

[16] Laurent Muratet, Stéphane Doncieux, and Jean-Arcady Meyer. “A biomimetic reactive navigation system using the optical flow for a rotary-wing UAV in urban environment”. In: *Proceedings of the International Session on Robotics* (2004), pp. 2262–2270.

[17] Colm Ó’Dúnlaing and Chee K Yap. “A “retraction” method for planning the motion of a disc”. In: *Journal of Algorithms* 6.1 (1985), pp. 104–111.

[18] Abhijeet Ravankar et al. “Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges”. In: *Sensors* 18.9 (2018), p. 3170.

[19] Charles Richter, Adam Bry, and Nicholas Roy. “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments”. In: *Robotics research*. Springer, 2016, pp. 649–666.

[20] Fabio Ruggiero et al. “Impedance control of VToL UAVs with a momentum-based external generalized forces estimator”. In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 2093–2099.

[21] Markus Ryll et al. “6D interaction control with aerial robots: The flying end-effector paradigm”. In: *The International Journal of Robotics Research* 38.9 (2019), pp. 1045–1062.

[22] Bruno Siciliano et al. “Modelling, planning and control”. In: *Advanced Textbooks in Control and Signal Processing. Springer,* (2009).

[23] Nora Sleumer and Nadine Tschichold-Gürmann. “Exact cell decomposition of arrangements used for path planning in robotics”. In: *Technical Report/ETH Zurich, Department of Computer Science* 329 (1999).

[24] Marco Tognon, Rachid Alami, and Bruno Siciliano. “Physical human-robot interaction with a tethered aerial vehicle: Application to a force-based human guiding problem”. In: *IEEE Transactions on Robotics* 37.3 (2021), pp. 723–734.

[25] Teodor Tomić. “Model-based control of flying robots for robust interaction under wind influence”. PhD thesis. Hannover: Institutionelles Repositorium der Leibniz Universität Hannover, 2018.

[26] Teodor Tomić and Sami Haddadin. “A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots”. In: *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2014, pp. 4197–4204.

[27] Teodor Tomić et al. “Simultaneous contact and aerodynamic force estimation (s-cafe) for aerial robots”. In: *The International Journal of Robotics Research* 39.6 (2020), pp. 688–728.

[28] Li Wang, Aaron D Ames, and Magnus Egerstedt. “Safe certificate-based maneuvers for teams of quadrotors using differential flatness”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3293–3298.

[29] Boyu Zhou et al. “Robust and efficient quadrotor trajectory generation for fast autonomous flight”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3529–3536.

[30] Dingjiang Zhou and Mac Schwager. “Vector field following for quadrotors using differential flatness”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 6567–6572.