Politecnico di Torino

Master of Science in
Mechatronic Engineering

— Master Thesis —

Learning Model Predictive Control for Optimal Path Planning of Quadrotors in Multi-Scenario Applications

Student
Lorenzo Calogero

Supervisors
Fabrizio Dabbene
Martina Mammarella

A.Y. 2021-2022

Abstract

The aim of this thesis is to develop a Learning Model Predictive Control (LMPC) framework for quadrotors; this thesis results to be one of the first comprehensive studies on LMPC applied to quadrotors in real-case scenarios.

LMPC is a novel control technique that autonomously learns to improve its performances, by collecting data coming from past executions of the control task. Given a certain task, the control algorithm makes the quadrotor to perform it repetitively; information coming from these repetitive iterations is progressively collected and employed by the controller to obtain better performances for the required task. This control method is especially versatile and useful for time-sensitive operations, in which the drone has to make fast and dexterous maneuvers in constrained and cluttered environments.

In this scenario, our specific goal is to implement a LMPC algorithm that pilots the quadrotor within a closed 3D race track, in which multiple types of obstacles can be inserted. The task of the controller is to autonomously find the trajectory achieving the minimum lap time, after multiple flights of the drone within the track.

The control algorithm is fully developed in MATLAB and is tested via several software-in-the-loop simulations, employing a complete dynamic model of the quadrotor.

The conducted simulations show that the LMPC algorithm successfully achieves the task of finding the optimal path for lap time minimization and also the additional task of avoiding the obstacles placed within the track: the control algorithm has learned to fly the quadrotor aggressively, adopting a flight style that exploits multiple driving tricks to optimize both the travelled distance and the time needed to complete a lap. The simulations not only demonstrate the correct functionality of the algorithm, but also empirically verify all the fundamental theorems of LMPC.

Table of contents

List of figures

List of symbols and abbreviations

List of symbols

Symbol	Meaning
	Chapter 2 (Quadrotor modelling)
$\boldsymbol{x}$	State variable
$\boldsymbol{u}$	Input variable
F_0	Base frame (fixed)
F_1	Body frame (moving)
$\boldsymbol{q}$	Generalized coordinates
$\boldsymbol{r}$	Position vector $(x, y, z)^T$ of the CM of the quadrotor / position generalized coordinates
$\boldsymbol{R}_1^0$	Rotation matrix from frame F_0 to F_1
$\boldsymbol{R}_a(\alpha)$	Elementary rotation matrix around axis a ($a = x, y, z$) by an angle α
$\boldsymbol{R}_{RPY}$	RPY rotation matrix
$\boldsymbol{\alpha}$	RPY orientation angles $(\phi, \theta, \psi)^T$ of the quadrotor / orientation generalized coordinates
$\boldsymbol{\omega}$	Angular velocity vector of the quadrotor
$\boldsymbol{T}_{RPY}$, $\boldsymbol{T}$	Transformation matrix from RPY angular rates $\dot{\boldsymbol{\alpha}}$ to angular velocity $\boldsymbol{\omega}$
$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Lagrangian function
$\mathcal{K}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Kinetic energy of the system
$\mathcal{P}(\boldsymbol{q})$	Potential energy of the system
$\boldsymbol{\mathcal{F}}$	Vector of generalized forces and torques

$\boldsymbol{f}_i$	Lift force generated by the i-th rotor of the quadrotor ($i = 1, 2, 3, 4$)
$\boldsymbol{\tau}_i$	Reaction torque applied on the quadrotor body by its i-th rotor ($i = 1, 2, 3, 4$)
$\boldsymbol{f}$	Cumulative thrust force
$\boldsymbol{\tau}$	Cumulative torque $(\tau_\phi, \tau_\theta, \tau_\psi)^T$, decomposed on the directions of RPY angles
$\boldsymbol{I}^1_{O_1}$	Inertia matrix of the quadrotor, computed wrt its CM (O_1) and expressed in the F_1 base
$\boldsymbol{C}(\boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}})$	Coriolis matrix
Chapter 2 (Frenet coordinates)	
$\boldsymbol{\gamma}$	Frenet reference curve
s	Signed curvilinear abscissa of $\boldsymbol{\gamma}$
d	Signed lateral distance from $\boldsymbol{\gamma}$
F_2	Frenet frame (moving)
ψ_1	Planar rotation angle between F_0 and F_1
ψ_2	Planar rotation angle between F_0 and F_2
ψ_e	Planar rotation angle between F_2 and F_1 (i.e. $\psi_e = \psi_1 - \psi_2$)
$\boldsymbol{r}^b_a$	Position of P wrt F_a, expressed in the base of F_b ($a, b = 0, 1, 2$)
$\boldsymbol{v}^b_a$	Time derivative of $\boldsymbol{r}^b_a$
$\boldsymbol{r}^c_{O_a,O_b}$	Position of O_a wrt F_b, expressed in the base of F_c ($a, b, c = 0, 1, 2$)
$\boldsymbol{v}^c_{O_a,O_b}$	Time derivative of $\boldsymbol{r}^c_{O_a,O_b}$
$\boldsymbol{\omega}^b_a$	Angular velocity of F_a wrt fixed frame F_0, expressed in the base of F_b ($a, b = 0, 1, 2$)
$\boldsymbol{R}^a_b$	Rotation matrix from F_a to F_b
$K(s)$	Curvature function of $\boldsymbol{\gamma}$
$\boldsymbol{S}(\boldsymbol{v})$	Skew-symmetric matrix of vector $\boldsymbol{v}$
$\boldsymbol{f}_Q$	Discrete-time dynamic model of the quadrotor in Frenet coordinates
L_{track}	Length of the track (from start to finish line)
W_{track}	Width of the track (from inner to outer border)
d_{lim}	Width of the track (from centerline to border)
Chapter 3 (NMPC)	
$\boldsymbol{X}_k$, $\boldsymbol{X}_{[0,N]\|k}$	State optimization variables $(\boldsymbol{x}_{0\|k}, \boldsymbol{x}_{1\|k}, ..., \boldsymbol{x}_{N\|k})$ for the NMPC control problem
$\boldsymbol{U}_k$, $\boldsymbol{U}_{[0,N-1]\|k}$	Input optimization variables $(\boldsymbol{u}_{0\|k}, \boldsymbol{u}_{1\|k}, ..., \boldsymbol{u}_{N-1\|k})$ for the NMPC control problem

J_{NMPC}	NMPC cost function
h	Stage cost function
$J_{[0,N-1]}$	Cost function without terminal cost, evaluated over $k = 0, 1, ..., N-1$
V	Terminal cost function
$\boldsymbol{x}_r$	Reference state
$\boldsymbol{u}_r$	Reference input associated to the reference state
$\mathcal{X}_F$	Terminal set
$\boldsymbol{X}_k^*$	Optimal predicted state trajectory $(\boldsymbol{x}_{0\|k}^*, \boldsymbol{x}_{1\|k}^*, ..., \boldsymbol{x}_{N\|k}^*)$ of the NMPC control problem
$\boldsymbol{U}_k^*$	Optimal predicted input sequence $(\boldsymbol{u}_{0\|k}^*, \boldsymbol{u}_{1\|k}^*, ..., \boldsymbol{u}_{N-1\|k}^*)$ for the NMPC control problem
J_{NMPC}^*	Optimal cost
$\boldsymbol{X}$	State trajectory $(\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_k, ...)$ generated by the NMPC algorithm
$\boldsymbol{U}$	State trajectory $(\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_k, ...)$ generated by the NMPC algorithm
$\boldsymbol{J}_{\boldsymbol{f},\boldsymbol{x}}, \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$	Jacobian matrix of $\boldsymbol{f}$ with respect to the state variables $\boldsymbol{x}$
$\boldsymbol{J}_{\boldsymbol{f},\boldsymbol{u}}, \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}$	Jacobian matrix of $\boldsymbol{f}$ with respect to the input variables $\boldsymbol{u}$
$\hat{\boldsymbol{f}}_Q$	Discrete-time dynamic model of the quadrotor in Frenet coordinates for curvature propagation
$\tilde{K}(s)$	Relaxed curvature function of the Frenet curve $\boldsymbol{\gamma}$
K_{rel}	Curvature relaxation coefficient
Chapter 4 (LMPC)	
$\boldsymbol{X}^j$	State trajectory $(\boldsymbol{x}_0^j, \boldsymbol{x}_1^j, ..., \boldsymbol{x}_k^j, ...)$ generated at iteration j
$\boldsymbol{U}^j$	Input sequence $(\boldsymbol{u}_0^j, \boldsymbol{u}_1^j, ..., \boldsymbol{u}_k^j, ...)$ generated at iteration j
$\boldsymbol{X}$	State optimization variables $(\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_k, ...)$ for the infinite-horizon control problem
$\boldsymbol{U}$	Input optimization variables $(\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_k, ...)$ for the infinite-horizon control problem
$\boldsymbol{x}_S$	Initial state of every trajectory (non-repetitive LMPC)
$\boldsymbol{x}_F$	Goal state of the control action
h	Stage cost function
$J_{[0,T]}$	Cost function without terminal cost, evaluated over $k = 0, 1, ..., T$
$\boldsymbol{X}^*$	Optimal state trajectory of the infinite-horizon control problem
$\boldsymbol{U}^*$	Optimal input sequence of the infinite-horizon control problem
SS^j	Sampled safe set at iteration j

$J^j_{[k,\infty]}(\boldsymbol{x}^j_k)$, J^j_k	Cost-to-go of the trajectory j at time k (i.e. starting from the state $\boldsymbol{x}^j_k$)
$J^j_{[0,\infty]}(\boldsymbol{x}^j_0)$, J^j_0	Iteration cost of iteration j
Q^j	Terminal cost function at iteration j
N	LMPC prediction horizon
J_{LMPC}	LMPC cost function
$\boldsymbol{X}_k$, $\boldsymbol{X}_{[0,N]\|k}$	State optimization variables $(\boldsymbol{x}_{0\|k}, \boldsymbol{x}_{1\|k}, ..., \boldsymbol{x}_{N\|k})$ for the LMPC control problem
$\boldsymbol{U}_k$, $\boldsymbol{U}_{[0,N-1]\|k}$	Input optimization variables $(\boldsymbol{u}_{0\|k}, \boldsymbol{u}_{1\|k}, ..., \boldsymbol{u}_{N-1\|k})$ for the LMPC control problem
$\boldsymbol{X}^{j^*}_k$	Optimal predicted state trajectory $(\boldsymbol{x}^{j^*}_{0\|k}, \boldsymbol{x}^{j^*}_{1\|k}, ..., \boldsymbol{x}^{j^*}_{N\|k})$ of the LMPC control problem
$\boldsymbol{U}^{j^*}_k$	Optimal predicted input sequence $(\boldsymbol{u}^{j^*}_{0\|k}, \boldsymbol{u}^{j^*}_{1\|k}, ..., \boldsymbol{u}^{j^*}_{N-1\|k})$ for the LMPC control problem
$J^{j^*}_{LMPC}$	Optimal cost
CS^j	Convex safe set at iteration j
$\boldsymbol{\lambda}$	Convex combination weighting scalars
$\boldsymbol{\alpha}$	Parameters vector of the max-affine interpolating function
P^j	Terminal cost barycentric function at iteration j
$P^{(l)}_j$	j-th partition of data indices at the l-th iteration of the convex piecewise-linear fitting algorithm
$\boldsymbol{p}_j$	j-th seed point for the generation of the initial partitions for the fitting algorithm
$\boldsymbol{\mu}_x$	Sample mean of the data points to interpolate
$\boldsymbol{\Sigma}_x$	Sample covariance of the data points to interpolate
$d_{o,l}(s)$, $d_{o,u}(s)$	Obstacle functions for the lower and upper bounds on the lateral distance d
$z_{o,l}(s)$, $z_{o,u}(s)$	Obstacle functions for the lower and upper bounds on the altitude z
$\tilde{d}_{o,l}(s)$, $\tilde{d}_{o,u}(s)$	Relaxed obstacle functions for the lower and upper bounds on the lateral distance d
$\tilde{z}_{o,l}(s)$, $\tilde{z}_{o,u}(s)$	Relaxed obstacle functions for the lower and upper bounds on the altitude z
$d_{o,rel}$, $z_{o,rel}$	Obstacles relaxation coefficients

List of abbreviations

Abbreviation	Meaning
CM	Center of mass
DoF	Degrees of freedom
ILC	Iterative Learning Control
LMPC	Learning Model Predictive Control
MINLP	Mixed-Integer Non-Linear Program/Programming
MPC	Model Predictive Control
NLP	Non-Linear Program/Programming
NMPC	Nonlinear Model Predictive Control
ODE	Ordinary Differential Equation
OP	Optimization Problem
QP	Quadratic Program/Programming
RPY	Roll-Pitch-Yaw
UAV	Unmanned Aerial Vehicle
wrt	With respect to
YALMIP	Yet Another Linear Matrix Inequalities Parser

1

Introduction

Unmanned Aerial Vehicles (UAVs), such as quadrotors, are aircrafts becoming nowadays increasingly more employed to assist humans in performing a wide range of time-sensitive tasks, typically in constrained outdoor and indoor environments; such tasks include logistics, transport, search, rescue, and monitoring. These time-sensitive operations require UAVs to make fast decisions and agile maneuvers in uncertain, cluttered, and dynamic environments, by intelligently exploiting environmental information to improve their performances over time.

In this scenario, the aim of this thesis is to investigate the use of Learning Model Predictive Control (LMPC) for quadrotors, which is a novel control technique that exploits past information, coming from previous iterations of the given control task, to autonomously improve its performance over time, while respecting system dynamics, environmental and actuation constraints.

It is worth remarking that, so far, very few studies have been developed on the application of LMPC to quadrotors; specifically, such studies are either limited only to 2D motion control [1] or employ just a simple kinematic model of the quadrotor as system to be controlled in software-in-the-loop simulations [8].
In this thesis, instead, we use LMPC to control the quadrotor motion within a closed 3D environment, having a limited height, horizontally delimited by a planar race track, and with the possibility of adding different kinds of obstacles therein; moreover, in the simu-

lations, we employ a complete dynamic model of the quadrotor (including aerodynamic effects) as system to be controlled.

The majority of the studies that combine quadrotors control with past data collection are actually based only on classic MPC, instead of LMPC. In these works, MPC is integrated with additional features, such as perception capabilities, data-driven refinement of the system dynamics, or neural networks based identification of the system model. However, such control methods are limited to reference tracking applications and, in general, exploit past information only to refine the quadrotor model, and not to improve the control performances.

The most relevant studies on LMPC apply this control method to ground vehicles [17] [3]. In these works, the vehicle task is to travel repetitively within a planar race track. During each lap, the LMPC algorithm collects the states of the generated trajectories and their corresponding costs. The vehicle learns from this "past" data to explore new ways to decrease the cost of its current trajectory, as long as it maintains the ability to reach one of the states collected during previous iterations.
This approach has the advantage of not requiring a reference trajectory; thus, it is especially versatile and useful during tasks where the desired trajectory is not known or is difficult to compute due to the system complexity. A clear example of such tasks is drone racing competitions, in which the UAV is required to autonomously perform agile and dexterous maneuvers to achieve a superior performance compared to human controlled aircrafts. Specifically, the control algorithm needs to find the trajectory achieving the minimum lap time after multiple flights of the drone within the race track.

In this thesis, we develop a LMPC framework for quadrotors control, that finds a good application in the context of drone racing and, in general, to any other scenario of constrained indoor activity.
Specifically, we implement a LMPC algorithm that controls the quadrotor motion within a closed 3D environment. This region of space, in which the quadrotor can move, has a limited vertical height and is delimited horizontally by a closed race track, having a predefined shape and a limited width. Moreover, within this space, multiple kinds of obstacles can be inserted at any point of the track, thus including the additional task, for the quadrotor, of avoiding such obstacles during its motion.
The quadrotor should move from the start to the finish line of the track, performing repetitive laps, staying within the bounds defined by the vertical and horizontal borders,

and avoiding the obstacles therein.
The states of the generated trajectories and the related costs are stored at every completed lap. With this data, collected across multiple successful iterations of the same task, the algorithm learns to explore new paths within the track to improve, at every iteration, the cost of the current trajectory followed by the quadrotor.
For our purposes, the cost of each trajectory will be quantified as the time needed by it to complete a lap; thus, the goal of the control algorithm is to obtain the optimal path that minimizes the lap time of the quadrotor.

The LMPC algorithm is developed starting from the construction of all the mathematical objects and tools required by LMPC theory. Through them, we formulate the LMPC optimization problem, which is the key element of the prediction phase of the algorithm; such optimization problem is then relaxed, to reduce its computational complexity and making it suitable for real-time applications. Finally, we formulate the complete LMPC algorithm. The functionality of the algorithm is tested by means of software-in-the-loop simulations, that use, as system to be controlled, a complete dynamic model of the quadrotor.

This thesis is organized as follows:

- In Chapter 2, the dynamic model of the quadrotor is derived. Staring from some fundamental notions on quadrotors flight, the modelling phase makes use of the Lagrangian approach to mechanics to derive the model of the quadrotor. Therefore, generalized coordinates for the quadrotor are defined, along with the computation of its kinetic and potential energy, to finally derive the system of Lagrange equations constituting the final model of the quadrotor.
  In this chapter, it is also taken into account the choice of a suitable coordinate system that allows to describe, in the most convenient way, the motion of the quadrotor within the track. For this purpose, the Frenet coordinate system is used.

- In Chapter 3, a first control approach based on classic MPC is formulated. This intermediate step is necessary, since, as explained in the next chapter, the LMPC algorithm needs to be initialized with a first feasible trajectory that allows the quadrotor to complete its first lap, before starting to be controlled by the proper LMPC. Such trajectory is generated by means of a basic reference tracking control method, which is chosen to be indeed MPC.

Therefore, a MPC algorithm for quadrotor trajectory tracking is developed. Starting from the general theory of Nonlinear MPC (NMPC), the MPC optimization problem and algorithm for the quadrotor are formulated, along with some relaxation techniques, that allow to reduce the computational complexity and provide robustness to infeasibility.

- In Chapter 4, the proper LMPC control algorithm for quadrotor autonomous and optimal path planning is formulated. Starting from general theoretical considerations on LMPC, all the LMPC mathematical objects and tools are constructed. The general LMPC optimization problem and algorithm are then formulated; to make such algorithm more suitable for real-time applications, relaxation procedures for the optimization problem are described and employed in the actual algorithm for quadrotors control.
  The general algorithm is then reformulated for the application to quadrotors; specifically, all the elements of the LMPC problem are readapted for the purpose of autonomously finding the path corresponding to the minimum lap time.
  The LMPC algorithm is then improved by adding the capability of avoiding obstacles within the track. This improved algorithm is meant to achieve both the task of finding the optimal path minimizing the lap time and the task of avoiding possible obstacles in the flight area.

- In Chapter 5, finally, we report all the results obtained through simulations of the MPC algorithm for quadrotor trajectory tracking and of the LMPC algorithms for quadrotor optimal path planning and obstacle avoidance. These algorithms are employed in software-in-the-loop simulations, in which they control the same quadrotor dynamic model on a certain number of different race tracks and in various conditions (including the addition of obstacles within the track).

2

Quadrotor modelling in Frenet coordinates

2.1. Introduction to systems modelling

The mathematical model of a real-world dynamical system is, in general, a system of ordinary differential equations (ODEs) that manages to fully describe the time evolution of the system, using a limited number of variables.
The *mathematical modelling of dynamical systems* is a branch of mathematical physics that plays a crucial role in automatic control: indeed, modelling is a mandatory step in control systems design, since every control method requires some knowledge of the mathematical equations governing the behaviour of the system to be controlled.

In a mathematical model (hereafter called just "model"), we identify two types of variables: *state variables* and *input variables*.

- State variables (denoted as $\boldsymbol{x} \in \mathbb{R}^n$) are the unknowns of the ODEs constituting the model; in general, they are internal quantities of the system that are able to fully describe its actual *state*.
  By solving the system of ODEs, we obtain the values $\boldsymbol{x}(t)$ of the state variables over time, which indeed represent the time evolution of the system.

For example, considering a rigid body that is free to move in the 3D space, the state variables of its model can be the position coordinates x, y, z (with respect to a fixed Cartesian frame), and a triplet of orientation angles ϕ, θ, ψ. Assuming that also the mass m of the body can change over time (like, for example, in a rocket, due to propellant consumption over time), m is another state variable of the system. The model will then be a system of seven ODEs in seven unknowns (i.e. the state variables).

Instead, considering an electrical circuit, the state variables are, in the general case, the voltage and the current of each circuit element. Therefore, if the circuit is composed by n nodes and b branches (b is also equal to the number of circuit elements, assuming to have only bipoles), its model will be a system of $2b$ differential-algebraic equations (DAEs) in $2b$ unknowns, where $n-1$ equations are given by the Kirchhoff current law, $b-n+1$ come from the Kirchhoff voltage law, and the last b are the constitutive equations of each component.

- Input variables (denoted as $\boldsymbol{u} \in \mathbb{R}^m$) represent external actions on the system that can influence its "free" evolution over time (i.e. when no inputs are present).
  Input variables can either be adjustable from the external (and so can be exploited as control inputs for the system), or cannot be controlled (in this case, they are called disturbance inputs).

  For the 3D rigid body, the input variables are all the external forces and torques applied on it.

  In the electrical circuit, instead, the input variables are the voltage/current of independent voltage/current sources.

For what concerns mechanical systems, models are typically subdivided in two categories: *kinematic models* and *dynamic models*.

- Kinematic models consider as input variables the relevant velocities of the system; therefore, the model contains only first-order ODEs.

  These models are well suited for simulation purposes and typically apply well to wheeled vehicles subject to non-holonomic constraints. However, they are non adequate for control purposes, since their inputs (i.e. the system velocities) are quantities that cannot be directly controlled from the external.

- Dynamic models, instead, consider as input variables the forces and torques applied

on the system (which, from Newton's second law, are related to the system accelerations); thus, the model contains second-order ODEs.

Dynamic models can be systematically derived using the Newtonian or the Lagrangian approach to mechanics. Moreover, these models, considering the "real" inputs of the systems, are well suited for control purposes, with the price of being much more complicated than kinematic ones.

As a consequence, for mechanical systems, in the following we will mainly focus our attention on dynamic models.

2.2. Notions on quadrotors flight

A *quadrotor* (or quadcopter) is a multi-rotor helicopter that is lifted and propelled by four rotors (i.e. electric motors + blades) in a cross configuration.

The propellers (i.e. the rotors) of a quadrotor are mounted such that the front and rear ones (1 and 3) rotate in the counter-clockwise direction, while the right and left ones (2 and 4) rotate clockwise, as shown in Figure 2.1.
As we will see in the next paragraphs, alternating the directions of rotation of the rotors is needed to nullify the yaw moment, that is generated by the reaction torques exerted by the motors on the quadrotor body.

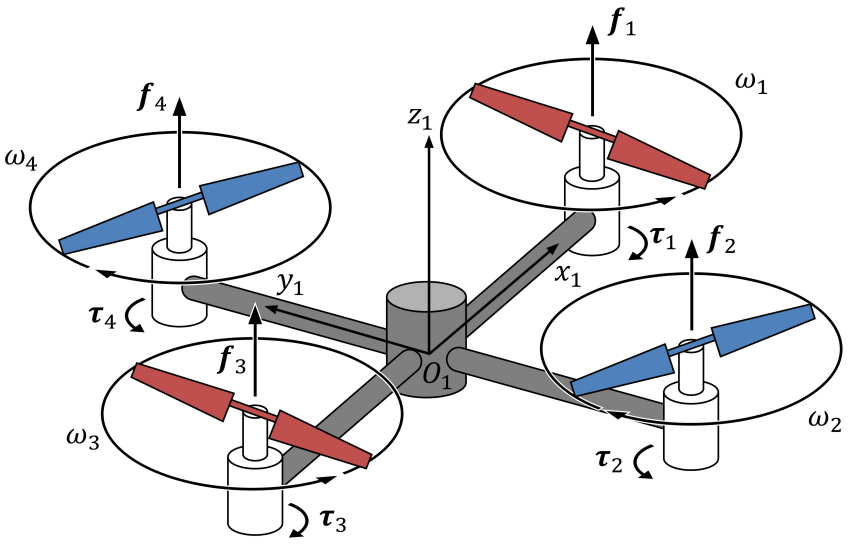


Figure 2.1. Quadrotor lift forces and reaction torques generated by the motors; the propellers rotating clockwise are depicted in blue, while those rotating counter-clockwise are depicted in red.

The position and orientation of a quadrotor can be controlled by adjusting the rotation speeds of the four motors. Specifically, each motor applies:

- a torque $\boldsymbol{\tau}$ on the rotor blades. This torque puts in rotation the blades in the same direction of it. The blades, hitting the air, have to overcome the drag force due to air friction. The air is also deflected by each blade towards the ground, generating a *lift force* $\boldsymbol{f}$ directed upwards; this force, if greater that the weight of the quadrotor, allows to lift it up from the ground;
- a *reaction torque* $-\boldsymbol{\tau}$ on the quadrotor body, as a consequence of Newton's third law, acting in the opposite direction with respect to the rotation of the blades. Since each motor applies an independent torque $\boldsymbol{\tau}_i$, with $i = 1, ..., 4$, if the sum $\sum_{i=1}^{4} \boldsymbol{\tau}_i \neq \mathbf{0}$, the quadrotor will start rotating around its z_1 axis (see Figure 2.1), generating a so-called yaw motion, which is something unwanted unless required by the pilot.
  By mounting the rotors such that they alternatively rotate in opposite directions, we obtain (when their rotation speeds are the same) that $\boldsymbol{\tau}_1 = \boldsymbol{\tau}_3 = -\boldsymbol{\tau}_2 = -\boldsymbol{\tau}_4$, making $\sum_{i=1}^{4} \boldsymbol{\tau}_i = \mathbf{0}$, and so nullifying the yaw motion.

The lift forces and reaction torques of each rotor are depicted in Figure 2.1.

The motion of the quadrotor, as for any rigid body moving in the 3D space, can be seen as the combination of two contributions: the *linear motion* of its center of mass (CM) and the *rotational motion* of the quadrotor around its CM.
The quadrotor has thus six degrees of freedom (DoF), three related to the linear motion (longitudinal, lateral, and vertical motions) and three related to the rotational motion (typically, roll, pitch, and yaw motions).

The control of these motions can be performed by adjusting the rotation speeds of the different motors of the quadrotor. Specifically:

- The *yaw motion* is the rotation of the quadrotor around its z_1 axis. As already explained, it is generated by the reaction torques of each motor: when the four rotor speeds are the same, the total reaction torque is null, and so no yaw motion is generated; if instead the four rotor speeds are not equal, the quadrotor will start to rotate (Figure 2.2a).
- The *roll motion* is the rotation of the quadrotor around its x_1 axis. It is generated by the difference of speed between the right and left rotors: if the left one (4) rotates

faster that the right one (2), then a positive roll motion is generated, since $f_4 > f_2$, and viceversa (Figure 2.2b).

- The *pitch motion* is the rotation of the quadrotor around its y_1 axis. It is generated by the difference of speed between the front and back rotors: if the back one (3) rotates faster that the front one (1), then a positive pitch motion is generated, since $f_3 > f_1$, and viceversa (Figure 2.2c).

- The *vertical motion* is the translation of the quadrotor along its z_1 axis. It is obtained by regulating the speed of all the rotors, which contribute to the total lift force that generates the vertical movement (Figure 2.2d).

- The *longitudinal motion* is the translation of the quadrotor along its x_1 axis, while the *lateral motion* is the translation along its y_1 axis. They are obtained by means of a combination of vertical motion and roll-pitch motions (Figure 2.2b-c).

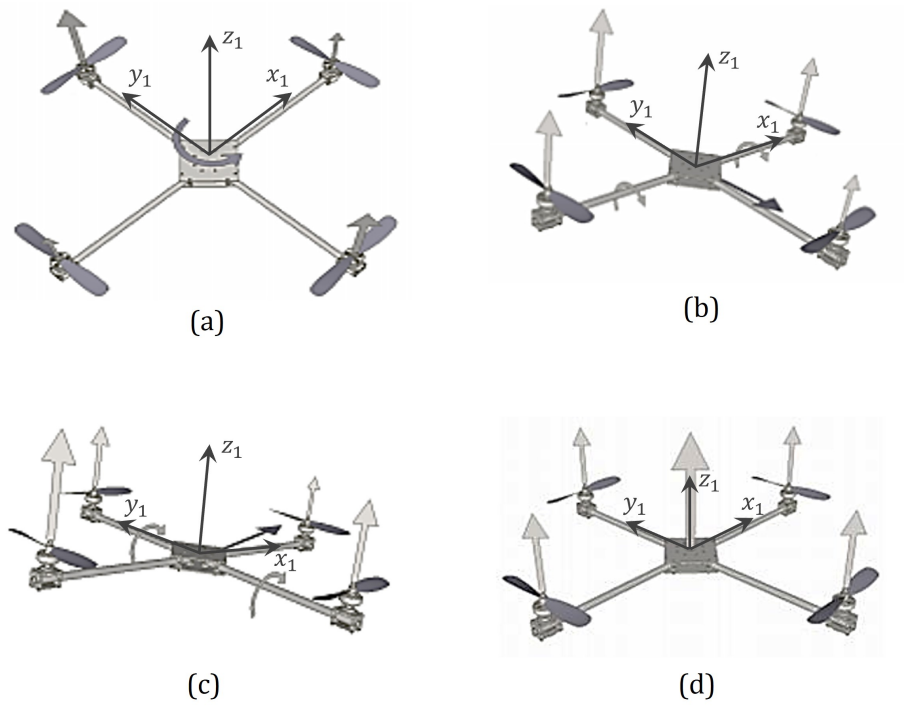


Figure 2.2. Quadrotor motions: (a) yaw motion, (b) roll and lateral motions, (c) pitch and longitudinal motions, (d) vertical motion.

2.3. Modelling problem setup

2.3.1. Quadrotor pose in the space

To derive the mathematical model of the quadrotor, we need at first to define the reference systems and the variables through which we will describe the *pose* (i.e. *position* and *orientation*) of the quadrotor in the space at any time instant.

Reference frames

We start considering the reference frames. As it is commonly done for any problem concerning the pose of a rigid body in the 3D space, we define two right-handed Cartesian frames:

- the *base frame* F_0 $(Oxyz)$, which is the frame representing the stationary space around the body; this frame is therefore a *fixed* one;
- the *body frame* F_1 $(O_1x_1y_1z_1)$, which is a frame attached to our body, which moves together with it and is centered in the CM of the body; this frame is therefore a *moving* one.

The frames are depicted in Figure 2.3.

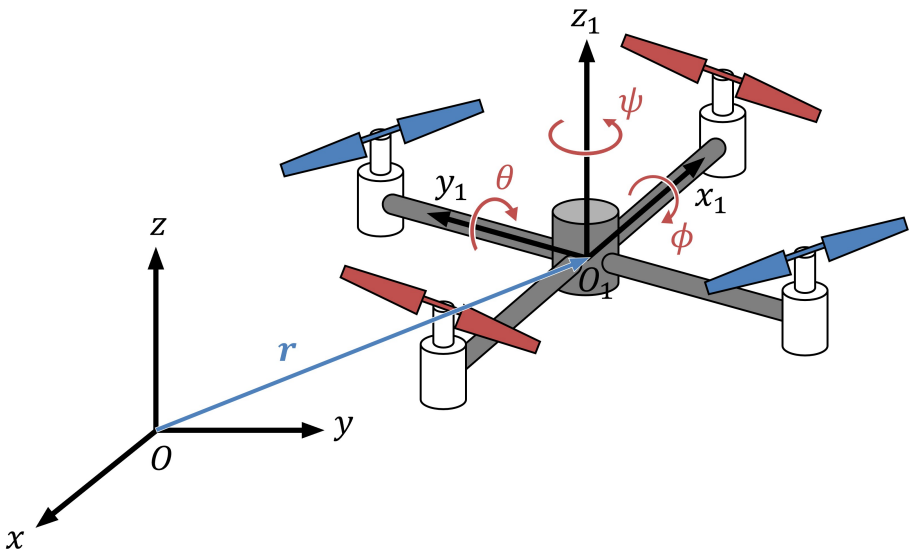


Figure 2.3. Quadrotor reference frames and generalized coordinates

2.3.2. Generalized coordinates

Now we need to define a set of variables $\boldsymbol{q}$, called *generalized coordinates*, allowing us to univocally describe any pose of the body.

Specifically:

- The *position* of the body is univocally defined by the vector $\boldsymbol{r} = \overrightarrow{OO_1}$, which is the position vector of the CM of the body with respect to (wrt) the base frame.
- The *orientation* of the body frame (and, thus, of the body itself) is univocally defined by a *rotation matrix* $\boldsymbol{R}$ wrt the fixed base frame.

Position

The vector $\boldsymbol{r}$ can be expressed in the base of F_0: $\boldsymbol{r} = (x, y, z)^T$. [1]

The components x, y, z of $\boldsymbol{r}$ are the first three generalized coordinates for the quadrotor.

Orientation

The rotation matrix $\boldsymbol{R} \equiv \boldsymbol{R}_{0\to 1} \equiv \boldsymbol{R}_1^0$ is defined as follows:

$$\boldsymbol{R}_1^0 = \begin{pmatrix} \hat{\boldsymbol{x}}_1 & \hat{\boldsymbol{y}}_1 & \hat{\boldsymbol{z}}_1 \end{pmatrix} \tag{2.1}$$

where $\hat{\boldsymbol{x}}_1$, $\hat{\boldsymbol{y}}_1$, $\hat{\boldsymbol{z}}_1$ are the versors of F_1 expressed in the base of F_0.

A rotation matrix can be always decomposed in a product of three matrices, each of them associated to an elementary rotation around one of the frame axes. These angles are called *Euler angles*. In particular, the following decomposition:

$$\boldsymbol{R}_1^0 = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) \equiv \boldsymbol{R}_{RPY}(\phi, \theta, \psi) \tag{2.2}$$

defines the triplet of angles $(\phi, \theta, \psi)^T \equiv \boldsymbol{\alpha}_{RPY} \equiv \boldsymbol{\alpha}$ called *RPY (roll-pitch-yaw) angles*.

The RPY angles describe an orientation that, from the pre-multiplication rule, can be constructed by:

[1] For notation clarity, in this chapter the superscript "0", indicating the F_0 base, will be omitted for vectors.

1) rotating the body frame around the x axis of the base frame by an angle ϕ (roll);

2) rotating the body frame around the y axis of the base frame by an angle θ (pitch);

3) rotating the body frame around the z axis of the base frame by an angle ψ (yaw).

Equivalently, the orientation, from the post-multiplication rule, can be constructed by:

1) rotating the body frame around its z_1 axis by an angle ψ (yaw);

2) rotating the body frame around its y_1 axis by an angle θ (pitch);

3) rotating the body frame around its x_1 axis by an angle ϕ (roll).

To ensure that it exists a biunivocal correspondence between RPY angles and rotation matrix, each angle is bounded in the following intervals:

$$\phi \in [-180°, 180°], \quad \theta \in [-90°, 90°], \quad \psi \in [-90°, 90°]$$

These bounds remove the ambiguity given by the 360°-periodicity of rotations.

The rotation matrix $\boldsymbol{R}_{RPY}(\phi, \theta, \psi)$ has the following expression [20]:

$$\boldsymbol{R}_{RPY}(\phi, \theta, \psi) = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \tag{2.3}$$

where $s_x \equiv \sin(x)$ and $c_x \equiv \cos(x)$.

From the rotation matrix, the RPY angles can be obtained as follows:

$$\phi = \text{atan2}(r_{32}, r_{33}), \quad \theta = \text{atan}\left(-\frac{r_{31}}{\sqrt{1 - r_{31}^2}}\right), \quad \psi = \text{atan2}(r_{21}, r_{11}) \tag{2.4}$$

For $\theta = \pm 90°$, $\boldsymbol{R}_{RPY}$ is singular and the expressions for ϕ and ψ become undefined. This phenomenon is called *singularity* or *gimbal lock* of RPY angles. It means that when $\theta = \pm 90°$, only the value of $\phi \mp \psi$ can be known exactly; therefore, in this case, multiple sequences of rotations (with different values of ϕ and ψ) give the same orientation/rotation matrix.

The RPY angles ϕ, θ, ψ, which are the components of $\boldsymbol{\alpha}$, are the final three generalized coordinates for the quadrotor.

In conclusion, we have that the generalized coordinates vector $\boldsymbol{q}$ is the following:

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{r} \\ \boldsymbol{\alpha} \end{pmatrix}, \quad \boldsymbol{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \boldsymbol{\alpha} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \tag{2.5}$$

The generalized coordinates of the quadrotor are depicted in Figure 2.3.

2.3.3. Kinematic quantities

Linear velocity

The linear velocity of the quadrotor wrt the base frame is equal to the derivative wrt time of the vector $\boldsymbol{r}$:

$$\frac{d}{dt}\boldsymbol{r} = \dot{\boldsymbol{r}} \equiv \boldsymbol{v} = \begin{pmatrix} \dot{x} & \dot{y} & \dot{z} \end{pmatrix}^T \tag{2.6}$$

We can express $\dot{\boldsymbol{r}}$ in the F_1 base, obtaining the vector:

$$\boldsymbol{v}^1 \equiv \begin{pmatrix} u & v & w \end{pmatrix}^T, \quad \dot{\boldsymbol{r}} = \boldsymbol{R}_1^0 \boldsymbol{v}^1 \tag{2.7}$$

Angular velocity

Considering the RPY angles as function of time, we obtain a time-varying rotation matrix describing the sequence of orientations assumed by the body in time.

Knowing the angular rates $\begin{pmatrix} \dot{\phi}, \dot{\theta}, \dot{\psi} \end{pmatrix}^T = \frac{d}{dt}(\phi, \theta, \psi)^T = \dot{\boldsymbol{\alpha}}$, it is possible to derive the value of the angular velocity vector $\boldsymbol{\omega}$ of the body (wrt the base frame) through the following expression [21]:

$$\boldsymbol{\omega} = \boldsymbol{T}_{RPY}(\boldsymbol{\alpha})\dot{\boldsymbol{\alpha}} \equiv \boldsymbol{T}(\boldsymbol{\alpha})\dot{\boldsymbol{\alpha}} \tag{2.8}$$

where:

$$\boldsymbol{T}_{RPY} = \begin{pmatrix} c_\theta c_\psi & -s_\psi & 0 \\ c_\theta s_\psi & c_\psi & 0 \\ -s_\theta & 0 & 1 \end{pmatrix}, \quad \boldsymbol{T}_{RPY}^{-1} = \begin{pmatrix} c_\psi / c_\theta & s_\psi / c_\theta & 0 \\ -s_\psi & c_\psi & 0 \\ t_\theta c_\psi & t_\theta s_\psi & 1 \end{pmatrix} \tag{2.9}$$

with $t_x \equiv \tan(x)$.

We can express $\boldsymbol{\omega}$ in the F_1 base, obtaining the vector:

$$\boldsymbol{\omega}^1 \equiv \begin{pmatrix} p & q & r \end{pmatrix}^T, \ \boldsymbol{\omega} = \boldsymbol{R}_1^0 \boldsymbol{\omega}^1 \quad \Rightarrow \quad \boldsymbol{\omega}^1 = \boldsymbol{R}_0^1 \boldsymbol{T} \dot{\boldsymbol{\alpha}} \equiv \boldsymbol{W} \dot{\boldsymbol{\alpha}} \tag{2.10}$$

where:

$$\boldsymbol{W} = \begin{pmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\phi c_\theta \end{pmatrix}, \quad \boldsymbol{W}^{-1} = \begin{pmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{pmatrix} \tag{2.11}$$

$\boldsymbol{T}$ and $\boldsymbol{W}$ are singular for $\theta = \pm 90°$ (gimbal lock).

Note 2.1 The fact that $\boldsymbol{T}_{RPY}$ is singular for $\theta = \pm 90°$ is the biggest issue caused by the gimbal lock of RPY angles.
When dealing only with orientations, gimbal lock is not a severe issue, since the ambiguity on the values of ϕ and ψ is limited to a single time instant and specific formulae exist to evaluate $\phi \mp \psi$.
However, gimbal lock also causes the singularity of the transformation matrix $\boldsymbol{T}_{RPY}$ from angular rates to angular velocity. Specifically, the inverse matrix $\boldsymbol{T}_{RPY}^{-1}$ will be implicitly embedded in the model equations, since it allows to compute the angular rates from the angular velocity vector:

$$\dot{\boldsymbol{\alpha}} = \boldsymbol{T}_{RPY}^{-1}(\boldsymbol{\alpha})\boldsymbol{\omega} \tag{2.12}$$

During the simulation, such angular rates will be numerically integrated to obtain the RPY angles values in time $\boldsymbol{\alpha}(t)$.
If during the simulation the system goes in the singular configuration $\theta = \pm 90°$, some elements of $\boldsymbol{T}_{RPY}^{-1}$ will diverge, making diverge also the angular rates and, in turn, leading to the computation of erroneous RPY angles; this error is irremediable and will affect the angles in all the next time instants.

Therefore, when using a model based on Euler angles, we have to ensure that the system does not go in the singular configuration.

Alternative models exploit *quaternions* instead of Euler angles to avoid the singularity, since the transformation matrix from quaternion rate to angular velocity does not become singular in the gimbal lock configuration.

2.4. Quadrotor kinematic model

The kinematic model of the quadrotor can be obtained in a simple way, by considering as system inputs the linear and angular velocities $\boldsymbol{v}^1$ and $\boldsymbol{\omega}^1$, expressed in the F_1 base, and the relations (2.7), (2.10):

$$\begin{cases} \dot{\boldsymbol{r}} = \boldsymbol{R}_1^0(\boldsymbol{\alpha})\boldsymbol{v}^1 \\ \dot{\boldsymbol{\alpha}} = \boldsymbol{W}(\boldsymbol{\alpha})^{-1}\boldsymbol{\omega}^1 \end{cases} \Rightarrow \begin{cases} \dot{x} = c_\theta c_\psi u - (c_\phi s_\psi + s_\phi s_\theta c_\psi)v + (s_\phi s_\psi + c_\phi s_\theta c_\psi)w \\ \dot{y} = c_\theta s_\psi u + (c_\phi c_\psi + s_\phi s_\theta s_\psi)v - (s_\phi c_\psi + c_\phi s_\theta s_\psi)w \\ \dot{z} = -s_\theta u + s_\phi c_\theta v + c_\phi c_\theta w \\ \dot{\phi} = p + s_\phi t_\theta q + c_\phi t_\theta r \\ \dot{\theta} = c_\phi q - s_\phi r \\ \dot{\psi} = \dfrac{1}{c_\theta}(s_\phi q + c_\phi r) \end{cases} \tag{2.13}$$

Defining the input vector $\boldsymbol{u}$:

$$\boldsymbol{u} = (u_1, u_2, u_3, u_4, u_5, u_6)^T \equiv \begin{pmatrix} \boldsymbol{v}^1 \\ \boldsymbol{\omega}^1 \end{pmatrix} = (u, v, w, p, q, r)^T \tag{2.14}$$

we can rewrite:

Quadrotor kinematic model

$$\begin{cases} \dot{x} = c_\theta c_\psi u_1 - (c_\phi s_\psi + s_\phi s_\theta c_\psi)u_2 + (s_\phi s_\psi + c_\phi s_\theta c_\psi)u_3 \\ \dot{y} = c_\theta s_\psi u_1 + (c_\phi c_\psi + s_\phi s_\theta s_\psi)u_2 - (s_\phi c_\psi + c_\phi s_\theta s_\psi)u_3 \\ \dot{z} = -s_\theta u_1 + s_\phi c_\theta u_2 + c_\phi c_\theta u_3 \\ \dot{\phi} = u_4 + s_\phi t_\theta u_5 + c_\phi t_\theta u_6 \\ \dot{\theta} = c_\phi u_5 - s_\phi u_6 \\ \dot{\psi} = \dfrac{1}{c_\theta}(s_\phi u_5 + c_\phi u_6) \end{cases} \tag{2.15}$$

2.5. Quadrotor dynamic model

2.5.1. Lagrange formulation

The dynamic model of a mechanical system, described by means of generalized coordinates, allows to relate the causes of motion (i.e. forces and torques applied on the system) to the time evolution of each generalized coordinate.

This model can be derived in a systematic way by means of the *Lagrangian approach to mechanics*.

In the Lagrange formulation of classical mechanics, given a system with n generalized coordinates $\boldsymbol{q} = (q_i)_{i=1}^n$, we introduce the *Lagrangian function*:

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathcal{K}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \mathcal{P}(\boldsymbol{q}) \tag{2.16}$$

where $\mathcal{K}$ and $\mathcal{P}$ respectively denote the total *kinetic energy* and *potential energy* of the system, as function of the generalized coordinates $\boldsymbol{q}$ and their derivatives wrt time $\dot{\boldsymbol{q}}$.

The dynamic model is a system of n second-order ODEs, called *Lagrange equations*, with the generalized coordinates $\boldsymbol{q}$ as unknowns, which are obtained by computing:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \mathcal{F}_i, \quad i = 1, ..., n \tag{2.17}$$

where $\boldsymbol{\mathcal{F}} = (\mathcal{F}_i)_{i=1}^n$ is the vector of *generalized forces and torques* applied on the system. In general, $\boldsymbol{\mathcal{F}} = \boldsymbol{F} + \boldsymbol{T}$, where $\boldsymbol{F}$ is the vector of generalized forces and $\boldsymbol{T}$ is the vector of generalized torques.

The system of ODEs (2.17) can be rewritten in compact form as:

$$\frac{d}{dt}\begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \dot{q}_1} & \cdots & \frac{\partial \mathcal{L}}{\partial \dot{q}_n} \end{pmatrix}^T - \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial q_1} & \cdots & \frac{\partial \mathcal{L}}{\partial q_n} \end{pmatrix}^T = \begin{pmatrix} \mathcal{F}_1 \\ \vdots \\ \mathcal{F}_n \end{pmatrix} \Rightarrow \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}} = \boldsymbol{\mathcal{F}} \tag{2.18}$$

We will now use the Lagrangian approach to obtain the dynamic model equations of the quadrotor, given the setup defined in § 2.3.

Generalized forces and torques

We start computing the generalized forces and torques acting on the quadrotor.

As reported in § 2.2, the angular velocity ω_i of each rotor $i = 1, ..., 4$ creates a force $\boldsymbol{f}_i$ in the direction of the rotor axis.
Moreover, the rotation of each motor applies a torque $\boldsymbol{\tau}_i$ on the quadrotor in the opposite sense wrt ω_i:

$$f_i = k\omega_i^2, \quad \tau_i = \pm(b\omega_i^2 + I_R\dot{\omega}_i) \tag{2.19}$$

where k is the lift coefficient, b is the drag coefficient (i.e. the air friction coefficient) and I_R is the moment of inertia of the rotor; in the rotor torque expression, we choose the $+$ if the blades rotate clockwise and the $-$ if the blades rotate counter-clockwise.

Typically, the latter contribution on $\boldsymbol{\tau}_i$ is neglected; in this way, we can relate rotor force and torque as follows:

$$\tau_i = \pm c f_i, \quad c = b/k \tag{2.20}$$

All rotor forces generate a cumulative thrust force $\boldsymbol{f}$ in the direction of the z_1 axis and applied in the CM of the quadrotor; the cumulative torque $\boldsymbol{\tau}$ on the body is reformulated as three torques $\boldsymbol{\tau}_\phi$, $\boldsymbol{\tau}_\theta$ and $\boldsymbol{\tau}_\psi$ acting in the direction of the corresponding RPY angles (Figure 2.4):

$$\boldsymbol{f}^1 = \begin{pmatrix} 0 \\ 0 \\ f \end{pmatrix}, \quad f = f_1 + f_2 + f_3 + f_4 \tag{2.21}$$

$$\boldsymbol{\tau} \equiv \begin{pmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix}, \quad \tau_\phi = l(f_4 - f_2), \quad \tau_\theta = l(f_3 - f_1)$$

$$\tau_\psi = \tau_1 + \tau_2 + \tau_3 + \tau_4 = c(-f_1 + f_2 - f_3 + f_4) \tag{2.22}$$

where l is the length of the arms of the quadrotor. The expressions of τ_ϕ, τ_θ and τ_ψ are derived by means of the considerations done in § 2.2.

Thus, the control distribution from the four rotors of the quadrotor is given by:

$$\begin{pmatrix} f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -l & 0 & l \\ -l & 0 & l & 0 \\ -c & c & -c & c \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} \tag{2.23}$$

So, if a required thrust and torque vector is given, one may solve for the rotor force vector using (2.23) [4].

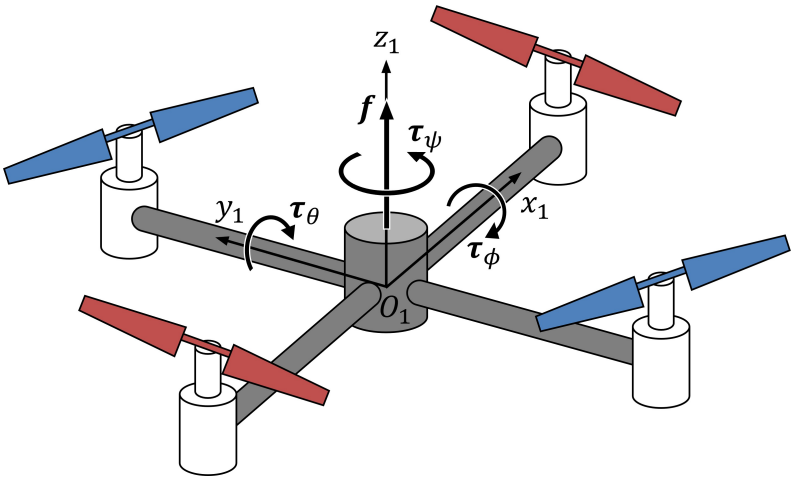


Figure 2.4. Quadrotor thrust force and torques

The generalized forces and torques $\boldsymbol{\mathcal{F}} = (\mathcal{F}_i)_{i=1}^n$ ($n = \dim(\boldsymbol{q}) = 6$) in Lagrangian mechanics are computed as follows:

$$\boldsymbol{F} = \sum_k \frac{\partial \boldsymbol{r}_k}{\partial \boldsymbol{q}} \boldsymbol{f}_k, \quad \boldsymbol{T} = \sum_k \frac{\partial \alpha_k}{\partial \boldsymbol{q}} \tau_k, \quad \boldsymbol{\mathcal{F}} = \boldsymbol{F} + \boldsymbol{T} \tag{2.24}$$

where $\{\boldsymbol{f}_k\}_k$ and $\{\boldsymbol{\tau}_k\}_k$ are all the forces and torques applied on the body, $\boldsymbol{r}_k$ is the position of the point of action of $\boldsymbol{f}_k$ on the body, and α_k is angle on which $\boldsymbol{\tau}_k$ acts.

Performing the calculations, we obtain:

$$\boldsymbol{F} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}} \boldsymbol{f} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{q}} \boldsymbol{R}_1^0 \boldsymbol{f}^1 = \begin{pmatrix} \boldsymbol{I}_3 \\ \boldsymbol{O}_3 \end{pmatrix} \boldsymbol{R}_1^0 \boldsymbol{f}^1 = \begin{pmatrix} \boldsymbol{R}_1^0 \boldsymbol{f}^1 \\ \boldsymbol{0}_3 \end{pmatrix} \tag{2.25}$$

$$\boldsymbol{T} = \sum_k \frac{\partial \alpha_k}{\partial \boldsymbol{q}} \tau_k = \frac{\partial \phi}{\partial \boldsymbol{q}} \tau_\phi + \frac{\partial \theta}{\partial \boldsymbol{q}} \tau_\theta + \frac{\partial \psi}{\partial \boldsymbol{q}} \tau_\psi = \begin{pmatrix} \boldsymbol{0}_3 \\ \boldsymbol{\tau} \end{pmatrix} \tag{2.26}$$

In conclusion:

$$\boldsymbol{\mathcal{F}} = \begin{pmatrix} \boldsymbol{R}_1^0 \boldsymbol{f}^1 \\ \boldsymbol{\tau} \end{pmatrix} \tag{2.27}$$

Lagrangian function

We now compute the Lagrangian function (2.16).

The kinetic energy of the body is the sum of its two main components, translational and rotational energy:

$$\mathcal{K}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} m \dot{\boldsymbol{r}}^T \dot{\boldsymbol{r}} + \frac{1}{2} {\boldsymbol{\omega}^1}^T \boldsymbol{I}_{O_1}^1 \boldsymbol{\omega}^1 \tag{2.28}$$

where $\boldsymbol{I}^1_{O_1}$ is the *inertia matrix* of the quadrotor, computed wrt its CM (O_1) and expressed in the F_1 base.

The inertia matrix can be computed as:

$$\boldsymbol{I}^1_{O_1} = \begin{pmatrix} I_x & I_{xy} & I_{xz} \\ I_{xy} & I_y & I_{yz} \\ I_{xz} & I_{yz} & I_z \end{pmatrix} = \int_V \begin{pmatrix} y^2+z^2 & -xy & -xz \\ -xy & x^2+z^2 & -yz \\ -xz & -yz & x^2+y^2 \end{pmatrix} \rho(x,y,z)dV \tag{2.29}$$

where V is the spatial volume of the quadrotor and ρ its volume density.

Having the quadrotor a symmetric structure, with its four arms aligned along the x_1 and y_1 axes of the body frame, and assuming its volume density as uniform, the inertia matrix will be diagonal, with all the extra-diagonal terms equal to 0 and $I_x = I_y$:

$$\boldsymbol{I}^1_{O_1} = \begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix} \tag{2.30}$$

If we model the shape of the quadrotor as a union of several simple bodies with geometric symmetry, we can derive closed-form expressions for the moments of inertia I_x, I_y and I_z. For example, we can model the quadrotor central body as a cube (mass M_b, side length b), the quadrotor arms as thin rods (mass M_a, length l), and the rotors as point masses (mass m_r); the four arms are attached one to the center of each lateral face of the cubic central body; the rotors are attached at the end of each arm. Then, the moments of inertia are equal to:

$$I_z = \frac{1}{6}M_b b^2 + \frac{4}{3}M_a l^2 + M_a b^2 + 4m_r\left(\frac{b}{2}+l\right) \tag{2.31a}$$

$$I_x = I_y = \frac{1}{6}M_b b^2 + \frac{2}{3}M_a l^2 + \frac{1}{2}M_a b^2 + 2m_r\left(\frac{b}{2}+l\right) \tag{2.31b}$$

The potential energy has only one component related to gravity:

$$\mathcal{P}(\boldsymbol{q}) = mgz \tag{2.32}$$

Therefore, the Lagrangian is equal to:

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}m\dot{\boldsymbol{r}}^T\dot{\boldsymbol{r}} + \frac{1}{2}{\boldsymbol{\omega}^1}^T\boldsymbol{I}^1_{O_1}\boldsymbol{\omega}^1 - mgz \tag{2.33}$$

We want to express $\mathcal{L}$ as function only of the generalized coordinates $\boldsymbol{q} = (\boldsymbol{r}, \boldsymbol{\alpha})$ and their derivatives wrt time $\dot{\boldsymbol{q}}$:

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}m\dot{\boldsymbol{r}}^T\dot{\boldsymbol{r}} + \frac{1}{2}{\boldsymbol{\omega}^1}^T\boldsymbol{I}^1_{O_1}\boldsymbol{\omega}^1 - mgz =$$

$$
\begin{aligned}
&= \frac{1}{2} m \dot{\boldsymbol{r}}^T \dot{\boldsymbol{r}} + \frac{1}{2} (\boldsymbol{W}\dot{\boldsymbol{\alpha}})^T \boldsymbol{I}_{O_1}^1 (\boldsymbol{W}\dot{\boldsymbol{\alpha}}) - mgz \\
&= \frac{1}{2} m \dot{\boldsymbol{r}}^T \dot{\boldsymbol{r}} + \frac{1}{2} \dot{\boldsymbol{\alpha}}^T \underbrace{\boldsymbol{W}^T \boldsymbol{I}_{O_1}^1 \boldsymbol{W}}_{\equiv \boldsymbol{J}} \dot{\boldsymbol{\alpha}} - mgz = \\
&= \frac{1}{2} m \dot{\boldsymbol{r}}^T \dot{\boldsymbol{r}} + \frac{1}{2} \dot{\boldsymbol{\alpha}}^T \boldsymbol{J} \dot{\boldsymbol{\alpha}} - mgz
\end{aligned} \tag{2.34}
$$

where $\boldsymbol{J}$ has the following expression [9]:

$$
\boldsymbol{J} = \begin{pmatrix}
I_x & 0 & -I_x s_\theta \\
0 & I_y c_\phi^2 + I_z s_\phi^2 & (I_y - I_z) c_\phi s_\phi c_\theta \\
-I_x s_\theta & (I_y - I_z) c_\phi s_\phi c_\theta & I_x s_\theta^2 + I_y s_\phi^2 c_\theta^2 + I_z c_\phi^2 c_\theta^2
\end{pmatrix} \tag{2.35}
$$

Computing the derivatives

We compute now all the derivatives required by (2.18) [6]:

$$
\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}} = \left(\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{r}}}, \ \frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{\alpha}}} \right)^T \ \Rightarrow \ \frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{r}}} = m\dot{\boldsymbol{r}}, \quad \frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{\alpha}}} = \frac{1}{2} \left(\boldsymbol{J} + \boldsymbol{J}^T \right) \dot{\boldsymbol{\alpha}} = \boldsymbol{J}\dot{\boldsymbol{\alpha}} \tag{2.36}
$$

$$
\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}} = \left(\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{r}}}, \ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{\alpha}}} \right)^T \ \Rightarrow \ \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{r}}} = m\ddot{\boldsymbol{r}}, \quad \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{\alpha}}} = \boldsymbol{J}\ddot{\boldsymbol{\alpha}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{\alpha}} \tag{2.37}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{r}}, \ \frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} \right)^T \ \Rightarrow \quad & \frac{\partial \mathcal{L}}{\partial \boldsymbol{r}} = \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} = -mg\hat{\boldsymbol{z}}, \\
& \frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} = \frac{\partial}{\partial \boldsymbol{\alpha}} \left(\frac{1}{2} \dot{\boldsymbol{\alpha}}^T \boldsymbol{J} \dot{\boldsymbol{\alpha}} \right) = \frac{1}{2} \frac{\partial}{\partial \boldsymbol{\alpha}} \left(\dot{\boldsymbol{\alpha}}^T \boldsymbol{J} \right) \dot{\boldsymbol{\alpha}}
\end{aligned} \tag{2.38}
$$

Analytical computation of the angular terms yields [14]:

$$
\begin{aligned}
\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \phi} &= I_y \left(-\dot{\psi}\dot{\theta} c_\theta s_\phi^2 + \dot{\psi}\dot{\theta} c_\theta c_\phi^2 + \dot{\psi}^2 s_\phi c_\phi c_\theta^2 - \dot{\theta}^2 s_\phi c_\phi \right) + \\
&\quad + I_z \left(-\dot{\psi}^2 s_\phi c_\phi c_\theta^2 + \dot{\psi}\dot{\theta} c_\theta s_\phi^2 - \dot{\psi}\dot{\theta} c_\theta c_\phi^2 + \dot{\theta}^2 s_\phi c_\phi \right) && (2.39) \\
\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \theta} &= I_x \left(-\dot{\psi}\dot{\phi} c_\theta + \dot{\psi}^2 c_\theta s_\theta \right) + I_y \left(-\dot{\theta}\dot{\psi} s_\phi c_\phi s_\theta - \dot{\psi}^2 s_\phi^2 c_\theta s_\theta \right) + \\
&\quad + I_z \left(-\dot{\psi}^2 s_\theta c_\theta c_\phi^2 + \dot{\psi}\dot{\theta} s_\theta s_\phi c_\phi \right) && (2.40) \\
\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \psi} &= 0 && (2.41) \\
\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\phi}} &= I_x (\dot{\phi} - \dot{\psi} s_\theta) && (2.42)
\end{aligned}
$$

$$\frac{\partial \mathcal{L}(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \dot{\theta}} = \dot{\theta}\left(I_y c_\phi^2 + I_z s_\phi^2\right) + \dot{\psi}\left(I_y c_\phi s_\phi c_\theta - I_z c_\phi s_\phi c_\theta\right) \tag{2.43}$$

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \dot{\psi}} &= -\dot{\phi} I_x s_\theta^2 + \dot{\theta}\left(\left(I_y - I_z\right) c_\phi s_\phi c_\theta\right) + \\ &+\dot{\psi} I_x s_\theta^2 + \dot{\psi} I_y s_\phi^2 c_2\theta + \dot{\psi} I_z c_\phi^2 c_\theta^2\end{aligned} \tag{2.44}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \dot{\phi}}\right) = I_x(\ddot{\phi} - \ddot{\psi} s_\theta - \dot{\phi}\dot{\psi} c_\theta) \tag{2.45}$$

$$\begin{aligned}\frac{d}{dt}\left(\frac{\partial \mathcal{L}(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \dot{\theta}}\right) &= I_y\left(\ddot{\theta} c_\phi^2 - 2\dot{\theta}\dot{\phi} c_\phi s_\phi + \ddot{\psi} c_\phi s_\phi c_\theta - \dot{\psi}\dot{\phi} s_\phi^2 c_\theta + \right. \\ &\left. +\dot{\psi}\dot{\phi} c_\phi^2 c_\theta - \dot{\psi}\dot{\theta} c_\phi s_\phi s_\theta\right) + \\ &+I_z\left(\ddot{\theta} s_\phi^2 + 2\dot{\theta}\dot{\phi} c_\phi s_\phi - \ddot{\psi} c_\phi s_\phi c_\theta + \dot{\psi}\dot{\phi} s_\phi^2 c_\theta - \right. \\ &\left. -\dot{\psi}\dot{\phi} c_\phi^2 c_\theta + \dot{\psi}\dot{\theta} c_\phi s_\phi s_\theta\right)\end{aligned} \tag{2.46}$$

$$\begin{aligned}\frac{d}{dt}\left(\frac{\partial \mathcal{L}(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \dot{\psi}}\right) &= I_x\left(-\ddot{\phi} s_\theta - \dot{\phi}\dot{\theta} c_\theta + \ddot{\psi} s_\theta^2 + 2\dot{\psi}\dot{\theta} s_\theta c_\theta^2\right) + \\ &+I_y\left(\ddot{\theta} c_\phi s_\phi c_\theta - \dot{\theta}\dot{\phi} s_\phi^2 c_\theta + \dot{\theta}\dot{\phi} c_\phi^2 c_\theta - \dot{\theta}^2 c_\phi s_\phi s_\theta + \right. \\ &\left. +\ddot{\psi} s_\phi^2 c_2\theta + 2\dot{\psi}\dot{\phi} s_\phi c_\phi^2 c_2\theta - 2\dot{\psi}\dot{\theta} s_\phi^2 c_\theta s_\theta\right) + \\ &+I_z\left(-\ddot{\theta} c_\phi s_\phi c_\theta + \dot{\theta}\dot{\phi} s_\phi^2 c_\theta - \dot{\theta}\dot{\phi} c_\phi^2 c_\theta + \dot{\theta}^2 c_\phi s_\phi s_\theta + \right. \\ &\left. +\ddot{\psi} c_\phi^2 c_2\theta - 2\dot{\psi}\dot{\phi} c_\phi s_\phi^2 c_2\theta - 2\dot{\psi}\dot{\theta} c_\theta^2 s_\theta c^2\phi\right)\end{aligned} \tag{2.47}$$

Lagrange equations

Finally, we can write the final Lagrange equations of the quadrotor:

$$\begin{cases} m\ddot{\boldsymbol{r}} + mg\hat{\boldsymbol{z}} = \boldsymbol{R}_1^0 \boldsymbol{f}^1 \\ \boldsymbol{J}\ddot{\boldsymbol{\alpha}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{\alpha}} - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\alpha}}\left(\dot{\boldsymbol{\alpha}}^T \boldsymbol{J}\right)\dot{\boldsymbol{\alpha}} = \boldsymbol{\tau} \end{cases} \Rightarrow \begin{cases} m\ddot{\boldsymbol{r}} + mg\hat{\boldsymbol{z}} = \boldsymbol{R}_1^0 \boldsymbol{f}^1 \\ \boldsymbol{J}\ddot{\boldsymbol{\alpha}} + \underbrace{\left[\dot{\boldsymbol{J}} - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\alpha}}\left(\dot{\boldsymbol{\alpha}}^T \boldsymbol{J}\right)\right]}_{\equiv \boldsymbol{C}(\boldsymbol{\alpha},\dot{\boldsymbol{\alpha}})}\dot{\boldsymbol{\alpha}} = \boldsymbol{\tau} \end{cases} \tag{2.48}$$

The matrix $\boldsymbol{C}(\boldsymbol{\alpha},\dot{\boldsymbol{\alpha}})$ is called *Coriolis matrix* and has the following expression [9]:

$$\boldsymbol{C}(\boldsymbol{\alpha},\dot{\boldsymbol{\alpha}}) = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$
\begin{aligned}
c_{11} &= 0 \\
c_{12} &= (I_y - I_z)(\dot{\theta} c_\phi s_\phi + \dot{\psi} s_\phi^2 c_\theta) + (I_z - I_y)\dot{\psi} c_\phi^2 c_\theta - I_x \dot{\psi} c_\theta \\
c_{13} &= (I_z - I_y)\dot{\psi} c_\phi s_\phi c_\theta^2 \\
c_{21} &= (I_z - I_y)(\dot{\theta} c_\phi s_\phi + \dot{\psi} s_\phi c_\theta) + (I_y - I_z)\dot{\psi} c_\phi^2 c_\theta - I_x \dot{\psi} c_\theta \\
c_{22} &= (I_z - I_y)\dot{\phi} c_\phi c_\phi \\
c_{23} &= -I_x \dot{\psi} s_\theta c_\theta + I_y \dot{\psi} s_\phi^2 s_\theta c_\theta + I_z \dot{\psi} c_\phi^2 s_\theta c_\theta \\
c_{31} &= (I_y - I_z)\dot{\psi} c_\theta^2 s_\phi c_\phi - I_x \dot{\theta} c_\theta \\
c_{32} &= (I_z - I_y)(\dot{\theta} c_\phi s_\phi s_\theta + \dot{\phi} s_\phi^2 c_\theta) + (I_y - I_z)\dot{\phi} c_\phi^2 c_\theta + \\
&\quad + I_x \dot{\psi} s_\theta c_\theta - I_y \dot{\psi} s_\phi^2 c_\theta - I_z \dot{\psi} c_\phi^2 s_\theta c_\theta \\
c_{33} &= (I_y - I_z)\dot{\phi} c_\phi s_\phi c_\theta^2 - I_y \dot{\theta} s_\phi^2 c_\theta s_\theta - I_z \dot{\theta} c_\phi^2 c_\theta s_\theta + I_x \dot{\theta} c_\theta s_\theta
\end{aligned} \tag{2.49}
$$

In conclusion, the Lagrange equations of the quadrotor are the following:

$$
\begin{cases}
\ddot{\boldsymbol{r}} = -g\hat{\boldsymbol{z}} + \dfrac{1}{m}\boldsymbol{R}_1^0(\boldsymbol{\alpha})\boldsymbol{f}^1 \\
\ddot{\boldsymbol{\alpha}} = \boldsymbol{J}(\boldsymbol{\alpha})^{-1}\left(\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{\alpha}, \dot{\boldsymbol{\alpha}})\dot{\boldsymbol{\alpha}}\right)
\end{cases} \tag{2.50}
$$

We would like now to expand these equations, to obtain the six second-order ODEs representing the dynamic model of the quadrotor.
To do so, in order to simplify the model equations (specifically those regarding $\boldsymbol{\alpha}$), we perform the following approximation [1] [18]: we assume that the angles ϕ and θ are sufficiently small to let:

$$
\cos(\phi) \approx \cos(\theta) \approx 1, \quad \sin(\phi) \approx \sin(\theta) \approx 0
$$

This assumption holds if the quadrotor tends to stay close to the hovering configuration (i.e. $\phi = \theta = 0$) during its motion, i.e. it is not required to perform very fast turns during its motion.

With this approximation, (2.50) are expanded, obtaining:

$$\begin{cases} \ddot{x} = \dfrac{1}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi) f \\ \ddot{y} = \dfrac{1}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi) f \\ \ddot{z} = -g + \dfrac{1}{m} c_\phi c_\theta f \\ \ddot{\phi} = \dfrac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} + \dfrac{1}{I_x}\tau_\phi \\ \ddot{\theta} = \dfrac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \dfrac{1}{I_y}\tau_\theta \\ \ddot{\psi} = \dfrac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \dfrac{1}{I_z}\tau_\psi \end{cases} \tag{2.51}$$

Defining the input vector $\boldsymbol{u}$ as:

$$\boldsymbol{u} = (u_1, u_2, u_3, u_4)^T \equiv \begin{pmatrix} \boldsymbol{f}^1 \\ \boldsymbol{\tau} \end{pmatrix} = (f, \tau_\phi, \tau_\theta, \tau_\psi)^T \tag{2.52}$$

we can rewrite:

Quadrotor dynamic model

$$\begin{cases} \ddot{x} = \dfrac{1}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi) u_1 \\ \ddot{y} = \dfrac{1}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi) u_1 \\ \ddot{z} = -g + \dfrac{1}{m} c_\phi c_\theta u_1 \\ \ddot{\phi} = \dfrac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} + \dfrac{1}{I_x} u_2 \\ \ddot{\theta} = \dfrac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \dfrac{1}{I_y} u_3 \\ \ddot{\psi} = \dfrac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \dfrac{1}{I_z} u_4 \end{cases} \tag{2.53}$$

2.5.2. Aerodynamic effects

To enforce a more realistic behaviour of the quadrotor, we can include in the model some *aerodynamic effects* that act on the quadrotor during its motion. As pointed out

in [9], several aerodynamic effects have been studied, such as the dependance of the thrust force on the angle of attack, blade flipping and airflow disruptions.

The influence of aerodynamic effects is typically complicated and difficult to model; moreover, some of these effects have significant influence only for high velocities. Therefore, in our model, we will only include the effect due to the air drag force (i.e. the air resistance) on the quadrotor:

Quadrotor dynamic model (with aerodynamic effects)

$$\begin{cases} \ddot{x} = \frac{1}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi)u_1 - \frac{\beta_x}{m}\dot{x} \\ \ddot{y} = \frac{1}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi)u_1 - \frac{\beta_y}{m}\dot{y} \\ \ddot{z} = -g + \frac{1}{m}c_\phi c_\theta u_1 - \frac{\beta_z}{m}\dot{z} \\ \ddot{\phi} = \frac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} + \frac{1}{I_x}u_2 \\ \ddot{\theta} = \frac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \frac{1}{I_y}u_3 \\ \ddot{\psi} = \frac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \frac{1}{I_z}u_4 \end{cases} \tag{2.54}$$

where β_x, β_y and β_z are the drag force coefficients (i.e. the air friction coefficients) in the corresponding directions of the base frame.

2.6. Choosing the coordinate system

So far we have derived a dynamic model of the quadrotor where its position in the 3D space is described by classic Cartesian coordinates. Now, we have to consider whether this *coordinate system* is adequate for our needs or if we need to rewrite the model with another coordinate system.

This is specifically related to the task that we want to achieve. As described in the Introduction, we want the quadrotor to move on a bounded 3D space, limited horizontally by a closed planar race track, having a predefined shape and a limited width; it should move on the track departing from the start line and arriving to the finish one (being the track closed, the two lines coincide); also, during the motion, the quadrotor must not

exit from the track boundaries. Moreover, the quadrotor will be controlled by means of Model Predictive Control (MPC) algorithms.

This context poses two severe issues when using a Cartesian coordinate system to describe the position of the body:

- To describe the shape and the width of the track, a set of state constraints on the planar position (x, y) of the quadrotor has to be included in the MPC optimization problem. These constraints may be very difficult to write in a closed form using Cartesian coordinates and, in general, would be non-convex and nonlinear
- Being the track closed, in Cartesian coordinates the initial position and the goal end point of the motion can be coincident or very close to each other, causing the control system to not produce any motion or to generate an unwanted trajectory that does not travel along the whole track.

Both of these issues can be solved by transforming the quadrotor model from Cartesian coordinates to different ones, called Frenet coordinates.

2.7. Frenet coordinate system

The *Frenet coordinate system* is a reference system which univocally defines the position of a point P on the plane with respect to a predefined *reference curve* $\boldsymbol{\gamma}$ and by means of two coordinates.

Firstly, we define the reference curve $\boldsymbol{\gamma}(s) : \mathbb{R} \to \mathbb{R}^2$ on the xy plane, where s is the curvilinear abscissa of the curve. The position of P is then defined with respect to this curve through:

- the *signed curvilinear abscissa* s, which is the length of the curve $\boldsymbol{\gamma}$ from its origin to the orthogonal projection of P on $\boldsymbol{\gamma}$;
- the *signed distance* d from $\boldsymbol{\gamma}$, which is the lateral distance between P and its orthogonal projection on $\boldsymbol{\gamma}$.

On the reference curve $\boldsymbol{\gamma}$, we can construct a moving reference frame F_2 $(O_2 x_2 y_2)$, called *Frenet frame*, centered in the orthogonal projection of P on $\boldsymbol{\gamma}$ (O_2) and with axes the tangent versor $\hat{\boldsymbol{t}}_{\boldsymbol{\gamma}} \equiv \hat{\boldsymbol{x}}_2$ and the normal versor $\hat{\boldsymbol{n}}_{\boldsymbol{\gamma}} \equiv \hat{\boldsymbol{y}}_2$ of $\boldsymbol{\gamma}$ in O_2.

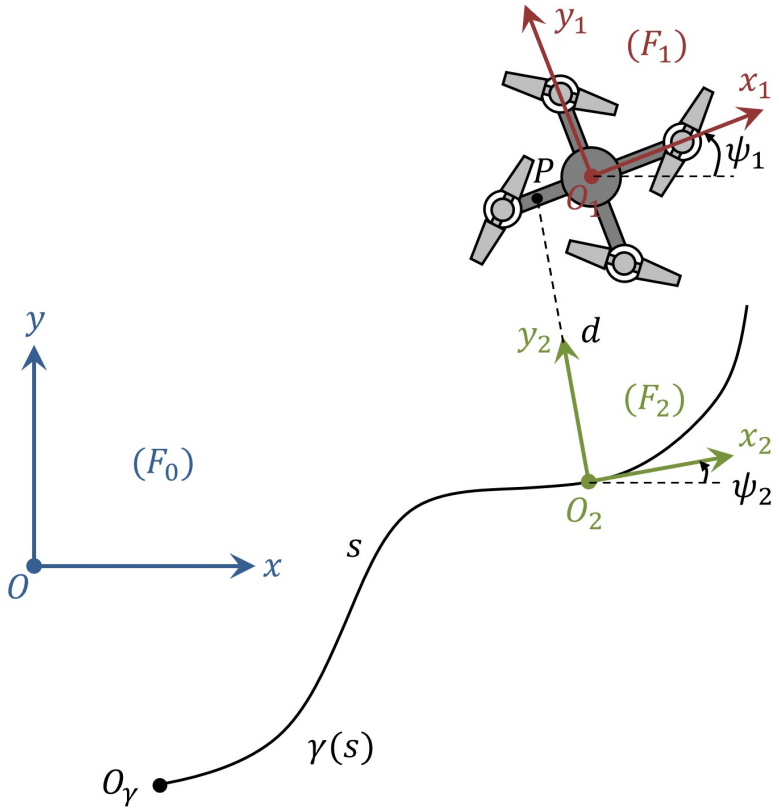


Figure 2.5. Cartesian (F_0), body (F_1), and Frenet (F_2) frames

Hereafter, we will consider the following frames (Figure 2.5):

- the fixed Cartesian frame F_0 (Oxy);
- the moving body frame F_1 $(O_1x_1y_1)$, which is the frame attached to our body, moving together with it and centered in the point O_1 (which is the body CM);
- the moving Frenet frame F_2 $(O_2x_2y_2)$.

We denote as ψ_1 the signed angle between the versors $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}_1$ and ψ_2 the signed angle between the versors $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}_2$ (also called *Frenet angle*).

2.8. Cartesian to Frenet conversion

Our objective is to derive a set of conversion equations that allow to transform a generic system model from Cartesian to Frenet coordinates. This means that, in our scenario, the planar position of the quadrotor will be expressed with the variables (s, d) instead of (x, y).

In particular, we start from the general case in which the model equations are referred to a generic point P attached to the quadrotor (which may not coincide with its CM).

Hereafter, the following notations will be used:

- The position of P wrt frame F_a, expressed in the base of frame F_b ($a, b = 0, 1, 2$, as stated previously), is denoted as:

$$\boldsymbol{r}_a^b \equiv \overrightarrow{O_a P}^{\,b} \tag{2.55}$$

  The derivative wrt time of this vector is denotes as:

$$\boldsymbol{v}_a^b \equiv \frac{d}{dt}\boldsymbol{r}_a^b \tag{2.56}$$

- The position of O_a (i.e origin of frame F_a) wrt frame F_b, expressed in the base of frame F_c, is denoted as:

$$\boldsymbol{r}_{O_a,O_b}^c \equiv \overrightarrow{O_a O_b}^{\,c} \tag{2.57}$$

  The derivative wrt time of this vector is denotes as:

$$\boldsymbol{v}_{O_a,O_b}^c \equiv \frac{d}{dt}\boldsymbol{r}_{O_a,O_b}^c \tag{2.58}$$

- The angular velocity of frame F_a wrt the fixed frame F_0, expressed in the base of frame F_b, is denoted as:

$$\boldsymbol{\omega}_a^b \tag{2.59}$$

- The rotation matrix from F_a to frame F_b is denoted as:

$$\boldsymbol{R}_b^a = \boldsymbol{R}_a^{b\,T} \tag{2.60}$$

  Specifically:

$$\boldsymbol{R}_0^1 = \begin{pmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{R}_0^2 = \begin{pmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{R}_1^2 = \begin{pmatrix} c_e & -s_e & 0 \\ s_e & c_e & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.61}$$

  where:

$$\sin(\psi_x) \equiv s_x, \quad \cos(\psi_x) \equiv c_x, \quad x = 1, 2, e \tag{2.62}$$

$$\psi_e \equiv \psi_1 - \psi_2 \tag{2.63}$$

Therefore, we have that:

$$\boldsymbol{r}_0^0 = \begin{pmatrix} x_P \\ y_P \\ 0 \end{pmatrix}, \quad \boldsymbol{r}_1^1 = \begin{pmatrix} l_1 \\ l_2 \\ 0 \end{pmatrix}, \quad \boldsymbol{r}_2^2 = \begin{pmatrix} 0 \\ d \\ 0 \end{pmatrix}, \quad \boldsymbol{r}_{O_1,O}^0 = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} \tag{2.64}$$

Being F_1, F_2 moving frames and F_0 a fixed frame, we can apply the fundamental law of kinematics on P between the couples of frames F_0, F_1 and F_0, F_2:

$$\begin{cases} \boldsymbol{v}_0 = \boldsymbol{v}_{O_1,O} + \boldsymbol{\omega}_1 \times \boldsymbol{r}_1 + \boldsymbol{v}_1 & (a) \\ \boldsymbol{v}_0 = \boldsymbol{v}_{O_2,O} + \boldsymbol{\omega}_2 \times \boldsymbol{r}_2 + \boldsymbol{v}_2 & (b) \end{cases} \tag{2.65}$$

Subtracting (b) from (a), we obtain the fundamental law of kinematics between the two moving frames F_1 and F_2:

$$\boldsymbol{v}_2 = \boldsymbol{v}_{O_1,O} - \boldsymbol{v}_{O_2,O} + \boldsymbol{\omega}_1 \times \boldsymbol{r}_1 - \boldsymbol{\omega}_2 \times \boldsymbol{r}_2 + \boldsymbol{v}_1 \tag{2.66}$$

We express (2.66) in the F_2 base:

$$\begin{aligned} \boldsymbol{v}_2^2 &= \boldsymbol{v}_{O_1,O}^2 - \boldsymbol{v}_{O_2,O}^2 + \boldsymbol{\omega}_1^2 \times \boldsymbol{r}_1^2 - \boldsymbol{\omega}_2^2 \times \boldsymbol{r}_2^2 + \boldsymbol{v}_1^2 \\ &= \boldsymbol{R}_0^2 \boldsymbol{v}_{O_1,O}^0 - \boldsymbol{v}_{O_2,O}^2 + \boldsymbol{\omega}_1^0 \times \boldsymbol{R}_1^2 \boldsymbol{r}_1^1 - \boldsymbol{\omega}_2^0 \times \boldsymbol{r}_2^2 + \boldsymbol{R}_1^2 \boldsymbol{v}_1^1 \end{aligned} \tag{2.67}$$

The expressions of each term of (2.67) are the following:

$$\boldsymbol{v}_2^2 = \begin{pmatrix} 0 \\ \dot{d} \\ 0 \end{pmatrix} \tag{2.68}$$

$$\boldsymbol{v}_{O_1,O}^2 = \boldsymbol{R}_0^2 \boldsymbol{v}_{O_1,O}^0 = \begin{pmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ 0 \end{pmatrix} = \begin{pmatrix} \dot{x}c_2 + \dot{y}s_2 \\ -\dot{x}s_2 + \dot{y}c_2 \\ 0 \end{pmatrix} \tag{2.69}$$

$$\boldsymbol{v}_{O_2,O}^2 = \begin{pmatrix} \dot{s} \\ 0 \\ 0 \end{pmatrix} \tag{2.70}$$

$$\boldsymbol{\omega}_1^0 = \begin{pmatrix} 0 \\ 0 \\ \dot{\psi}_1 \end{pmatrix}, \quad \boldsymbol{\omega}_2^0 = \begin{pmatrix} 0 \\ 0 \\ \dot{\psi}_2 \end{pmatrix} \tag{2.71}$$

$$\boldsymbol{r}_1^2 = \boldsymbol{R}_1^2 \boldsymbol{r}_1^1 = \begin{pmatrix} c_e & -s_e & 0 \\ s_e & c_e & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} l_1 \\ l_2 \\ 0 \end{pmatrix} = \begin{pmatrix} l_1 c_e - l_2 s_e \\ l_1 s_e + l_2 c_e \\ 0 \end{pmatrix} \tag{2.72}$$

$$\boldsymbol{r}_2^2 = \begin{pmatrix} 0 \\ d \\ 0 \end{pmatrix} \tag{2.73}$$

$$\boldsymbol{v}_1^2 = \boldsymbol{R}_1^2 \boldsymbol{v}_1^1 = \boldsymbol{R}_1^2 \, \boldsymbol{0} = \boldsymbol{0} \tag{2.74}$$

In the end, by plugging all the previous expressions into (2.67), we obtain:

$$\begin{pmatrix} 0 \\ \dot{d} \\ 0 \end{pmatrix} = \begin{pmatrix} \dot{x}c_2 + \dot{y}s_2 \\ -\dot{x}s_2 + \dot{y}c_2 \\ 0 \end{pmatrix} - \begin{pmatrix} \dot{s} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \dot{\psi}_1 \end{pmatrix} \times \begin{pmatrix} l_1 c_e - l_2 s_e \\ l_1 s_e + l_2 c_e \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \dot{\psi}_2 \end{pmatrix} \times \begin{pmatrix} 0 \\ d \\ 0 \end{pmatrix} \tag{2.75}$$

Performing the calculations, we obtain:

$$\begin{cases} \dot{s} = \dot{x}c_2 + \dot{y}s_2 + \dot{\psi}_1(-l_1 s_e - l_2 c_e) + \underbrace{\dot{\psi}_2}_{=K(s)\dot{s}} d \\ \dot{d} = -\dot{x}s_2 + \dot{y}c_2 + \dot{\psi}_1(l_1 c_e - l_2 s_e) \end{cases}$$
$$\Rightarrow \begin{cases} \dot{s} = \dfrac{1}{1 - K(s)d}\left(\dot{x}c_2 + \dot{y}s_2 - \dot{\psi}_1(l_1 s_e + l_2 c_e)\right) \\ \dot{d} = -\dot{x}s_2 + \dot{y}c_2 + \dot{\psi}_1(l_1 c_e - l_2 s_e) \end{cases} \tag{2.76}$$

Including in (2.76) also the expression:

$$\dot{\psi}_2 = K(s)\dot{s} \tag{2.77}$$

where $K(s)$ is the *curvature function* of $\boldsymbol{\gamma}$, and replacing $\dot{s}$ with its whole expression, we finally obtain:

Cartesian to Frenet conversion equations (general form)

$$\begin{cases} \dot{s} = \dfrac{1}{1 - K(s)d}\left(\dot{x}c_2 + \dot{y}s_2 - \dot{\psi}_1(l_1 s_e + l_2 c_e)\right) \\ \dot{d} = -\dot{x}s_2 + \dot{y}c_2 + \dot{\psi}_1(l_1 c_e - l_2 s_e) \\ \dot{\psi}_2 = \dfrac{K(s)}{1 - K(s)d}\left(\dot{x}c_2 + \dot{y}s_2 - \dot{\psi}_1(l_1 s_e + l_2 c_e)\right) \end{cases} \tag{2.78}$$

(2.78) is the set of conversion equations for a system model expressed in Cartesian coordinates to the equivalent one in Frenet coordinates.

We can simplify (2.78) by making P to coincide with the point O_1 (quadrotor CM):

Cartesian to Frenet conversion equations (simplified form)

$$\begin{cases} \dot{s} = \dfrac{1}{1 - K(s)d} \left(\dot{x} \cos \psi_2 + \dot{y} \sin \psi_2 \right) \\ \dot{d} = -\dot{x} \sin \psi_2 + \dot{y} \cos \psi_2 \\ \dot{\psi}_2 = \dfrac{K(s)}{1 - K(s)d} \left(\dot{x} \cos \psi_2 + \dot{y} \sin \psi_2 \right) \end{cases} \tag{2.79}$$

The obtained (2.78) and (2.79) coincide with the results reported in [12] and [3].

2.8.1. Model conversion

The actual coordinate conversion for the system model is performed by adding (2.79) to the model equations, thus adding three new system states: s, d, and ψ_2.

After the conversion to Frenet coordinates, the planar position of the body is determined by the states s and d, making x and y redundant. Therefore, if x and y do not appear in any other model equation, we can safely remove the equations for $\dot{x}$ and $\dot{y}$ (reducing the number of system states by two).

Note 2.2 A specific observation has to be made for the models of wheeled vehicles (e.g. unicycle, bicycle, car, etc.). For this kind of bodies, when we require them to track a reference trajectory (which is typically the Frenet curve $\boldsymbol{\gamma}$), one of the tracking requirements has also to be imposed also on the heading angle ψ_1.
In fact, for the quadrotor the heading/yaw angle ψ_1 is not relevant for tracking purposes, since the quadrotor is free of non-holonomic constraints on the direction of its velocity vector.
Instead, any wheeled vehicle is subject to non-holonomic constraints, which have the purpose of modelling the absence of lateral slippage of the wheels, meaning that the velocity vector at the wheels cannot have a component parallel to the wheel axle.

As a consequence, to accomplish trajectory tracking of wheeled vehicles, we must ensure that $\psi_1(t) \to \psi_2(t)$ for $t \to \infty$. For the sake of simplicity, we can rewrite the requirement as $\psi_1(t) - \psi_2(t) = \psi_e(t) \to 0$ for $t \to \infty$.
Therefore, it is convenient to replace, in the system model, the state ψ_2 with $\psi_e =$

$\psi_1 - \psi_2$. In this case, the equations (2.79) are reformulated as follows:

$$\dot{\psi}_2 = \dot{\psi}_1 - \dot{\psi}_e = K(s)\dot{s} \quad \Rightarrow \quad \dot{\psi}_e = \dot{\psi}_1 - K(s)\dot{s} \tag{2.80}$$

$$\Rightarrow \begin{cases} \dot{s} = \dfrac{1}{1 - K(s)d} \left(\dot{x}\cos(\psi_1 + \psi_e) + \dot{y}\sin(\psi_1 + \psi_e) \right) \\ \dot{d} = -\dot{x}\sin(\psi_1 + \psi_e) + \dot{y}\cos(\psi_1 + \psi_e) \\ \dot{\psi}_e = \dot{\psi}_1 - \dfrac{K(s)}{1 - K(s)d} \left(\dot{x}\cos(\psi_1 + \psi_e) + \dot{y}\sin(\psi_1 + \psi_e) \right) \end{cases} \tag{2.81}$$

Including (2.81) in the model equations adds the new states s, d, and ψ_e.

2.8.2. Alternative derivation

Instead of directly applying the fundamental law of kinematics (2.65), we can obtain the same results by differentiating the relations involving position vectors.

By looking at Figure 2.5, it holds:

$$\boldsymbol{r}_2 = \boldsymbol{r}_{O_1,O} - \boldsymbol{r}_{O_2,O} + \boldsymbol{r}_1 \tag{2.82}$$

Expressing it in the F_0 base:

$$\begin{aligned} \boldsymbol{r}_2^0 &= \boldsymbol{r}_{O_1,O}^0 - \boldsymbol{r}_{O_2,O}^0 + \boldsymbol{r}_1^0 \\ \boldsymbol{R}_2^0 \boldsymbol{r}_2^2 &= \boldsymbol{r}_{O_1,O}^0 - \boldsymbol{r}_{O_2,O}^0 + \boldsymbol{R}_1^0 \boldsymbol{r}_1^1 \end{aligned} \tag{2.83}$$

We now differentiate both sides of the equation:

$$\begin{aligned} \frac{d}{dt}\left(\boldsymbol{R}_2^0 \boldsymbol{r}_2^2\right) &= \frac{d}{dt}\left(\boldsymbol{r}_{O_1,O}^0 - \boldsymbol{r}_{O_2,O}^0 + \boldsymbol{R}_1^0 \boldsymbol{r}_1^1\right) \\ \boldsymbol{R}_2^0 \boldsymbol{v}_2^2 + \dot{\boldsymbol{R}}_2^0 \boldsymbol{r}_2^2 &= \boldsymbol{v}_{O_1,O}^0 - \boldsymbol{v}_{O_2,O}^0 + \boldsymbol{R}_1^0 \boldsymbol{v}_1^1 + \dot{\boldsymbol{R}}_1^0 \boldsymbol{r}_1^1 \end{aligned} \tag{2.84}$$

We recall that:

$$\boldsymbol{S}(\boldsymbol{\omega}_a^0) = \dot{\boldsymbol{R}}_a^0 \boldsymbol{R}_0^a \Rightarrow \dot{\boldsymbol{R}}_a^0 = \boldsymbol{S}(\boldsymbol{\omega}_a^0)\boldsymbol{R}_a^0 \tag{2.85}$$

where $\boldsymbol{\omega}_a^0$ is the angular velocity of frame F_a wrt the fixed frame F_0 and $\boldsymbol{S}$ denotes the skew-symmetric matrix:

$$\boldsymbol{S}(\boldsymbol{\omega}) = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \tag{2.86}$$

In particular, the matrix multiplication $(\boldsymbol{S}(\boldsymbol{\omega})(\cdot))$ is equivalent to the cross product $(\boldsymbol{\omega} \times \cdot)$; therefore, (2.84) becomes:

$$\begin{aligned}
&\boldsymbol{R}_2^0 \boldsymbol{v}_2^2 + \boldsymbol{\omega}_2^0 \times \boldsymbol{R}_2^0 \boldsymbol{r}_2^2 = \boldsymbol{v}_{O_1,O}^0 - \boldsymbol{v}_{O_2,O}^0 + \boldsymbol{R}_1^0 \boldsymbol{v}_1^1 + \boldsymbol{\omega}_1^0 \times \boldsymbol{R}_1^0 \boldsymbol{r}_1^1 \\
&\boldsymbol{v}_2^0 = \boldsymbol{v}_{O_1,O}^0 - \boldsymbol{v}_{O_2,O}^0 + \boldsymbol{\omega}_1^0 \times \boldsymbol{v}_1^0 - \boldsymbol{\omega}_2^0 \times \boldsymbol{v}_2^0 + \boldsymbol{v}_1^0 \\
&\Rightarrow \boldsymbol{v}_2 = \boldsymbol{v}_{O_1,O} - \boldsymbol{v}_{O_2,O} + \boldsymbol{\omega}_1 \times \boldsymbol{r}_1 - \boldsymbol{\omega}_2 \times \boldsymbol{r}_2 + \boldsymbol{v}_1
\end{aligned} \tag{2.87}$$

which is exactly the (2.66).

From this point, calculations are identical to those in the previous section.

2.9. Quadrotor dynamic model in Frenet coordinates

Recalling the dynamic model of the quadrotor in (2.54), we have that:

$$\begin{cases}
\ddot{x} = \dfrac{1}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi)u_1 - \dfrac{\beta_x}{m}\dot{x} \\
\ddot{y} = \dfrac{1}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi)u_1 - \dfrac{\beta_y}{m}\dot{y} \\
\ddot{z} = -g + \dfrac{1}{m}c_\phi c_\theta u_1 - \dfrac{\beta_z}{m}\dot{z} \\
\ddot{\phi} = \dfrac{I_y - I_z}{I_x}\dot{\theta}\dot{\psi} + \dfrac{1}{I_x}u_2 \\
\ddot{\theta} = \dfrac{I_z - I_x}{I_y}\dot{\phi}\dot{\psi} + \dfrac{1}{I_y}u_3 \\
\ddot{\psi} = \dfrac{I_x - I_y}{I_z}\dot{\phi}\dot{\theta} + \dfrac{1}{I_z}u_4
\end{cases} \tag{2.88}$$

By expressing it in *state-space form*, we obtain:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{\phi} = v_\phi \\ \dot{\theta} = v_\theta \\ \dot{\psi} = v_\psi \\ \dot{v}_x = \dfrac{1}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi)u_1 - \dfrac{\beta_x}{m} v_x \\ \dot{v}_y = \dfrac{1}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi)u_1 - \dfrac{\beta_y}{m} v_y \\ \dot{v}_z = -g + \dfrac{1}{m} c_\phi c_\theta u_1 - \dfrac{\beta_z}{m} v_z \\ \dot{v}_\phi = \dfrac{I_y - I_z}{I_x} v_\theta v_\psi + \dfrac{1}{I_x} u_2 \\ \dot{v}_\theta = \dfrac{I_z - I_x}{I_y} v_\phi v_\psi + \dfrac{1}{I_y} u_3 \\ \dot{v}_\psi = \dfrac{I_x - I_y}{I_z} v_\phi v_\theta + \dfrac{1}{I_z} u_4 \end{cases} \tag{2.89}$$

The states are $\boldsymbol{x} = (x, y, z, \phi, \theta, \psi, v_x, v_y, v_z, v_\phi, v_\theta, v_\psi)^T \in \mathbb{R}^{12} \Rightarrow n = 12$, while the inputs are $\boldsymbol{u} = (u_1, u_2, u_3, u_4)^T \in \mathbb{R}^4 \Rightarrow m = 4$.

To convert this model from Cartesian to Frenet coordinates, we recall equation (2.79):

$$\begin{cases} \dot{s} = \dfrac{1}{1 - K(s)d} \left(\dot{x}\, c_{\psi_2} + \dot{y}\, s_{\psi_2} \right) \\ \dot{d} = -\dot{x}\, s_{\psi_2} + \dot{y}\, c_{\psi_2} \\ \dot{\psi}_2 = \dfrac{K(s)}{1 - K(s)d} \left(\dot{x}\, c_{\psi_2} + \dot{y}\, s_{\psi_2} \right) \end{cases} \tag{2.90}$$

where, calling $\boldsymbol{\gamma}$ the Frenet reference curve: s is the signed curvilinear abscissa of $\boldsymbol{\gamma}$, which is the curve length from its origin to the orthogonal projection of the quadrotor CM on $\boldsymbol{\gamma}$; d the signed distance between the quadrotor CM and its orthogonal projection on $\boldsymbol{\gamma}$; ψ_2 is the Frenet angle between the $\hat{\boldsymbol{x}}_2$ versor of the Frenet frame (F_2) and the $\hat{\boldsymbol{x}}$ versor of the base frame (F_0) (see Figure 2.5); $K(s)$ is the curvature function of $\boldsymbol{\gamma}$.

The conversion is performed embedding the equations (2.79) into the model (2.89), obtaining:

State-space quadrotor dynamic model in Frenet coordinates

$$
\begin{cases}
\dot{z} = v_z \\
\dot{\phi} = v_\phi \\
\dot{\theta} = v_\theta \\
\dot{\psi} = v_\psi \\
\dot{v}_x = \dfrac{1}{m}(c_\phi s_\theta c_\psi + s_\phi s_\psi)u_1 - \dfrac{\beta_x}{m}v_x \\
\dot{v}_y = \dfrac{1}{m}(c_\phi s_\theta s_\psi - s_\phi c_\psi)u_1 - \dfrac{\beta_y}{m}v_y \\
\dot{v}_z = -g + \dfrac{1}{m}c_\phi c_\theta u_1 - \dfrac{\beta_z}{m}v_z \\
\dot{v}_\phi = \dfrac{I_y - I_z}{I_x}v_\theta v_\psi + \dfrac{1}{I_x}u_2 \\
\dot{v}_\theta = \dfrac{I_z - I_x}{I_y}v_\phi v_\psi + \dfrac{1}{I_y}u_3 \\
\dot{v}_\psi = \dfrac{I_x - I_y}{I_z}v_\phi v_\theta + \dfrac{1}{I_z}u_4 \\
\dot{s} = \dfrac{1}{1 - K(s)d}\left(v_x c_{\psi_2} + v_y s_{\psi_2}\right) \\
\dot{d} = -v_x s_{\psi_2} + v_y c_{\psi_2} \\
\dot{\psi}_2 = \dfrac{K(s)}{1 - K(s)d}\left(v_x c_{\psi_2} + v_y s_{\psi_2}\right)
\end{cases}
\tag{2.91}
$$

The states are $\boldsymbol{x} = (z, \phi, \theta, \psi, v_x, v_y, v_z, v_\phi, v_\theta, v_\psi, s, d, \psi_2)^T \in \mathbb{R}^{13} \Rightarrow n = 13$, while the inputs are $\boldsymbol{u} = (u_1, u_2, u_3, u_4)^T \in \mathbb{R}^4 \Rightarrow m = 4$.

We can notice that the states x and y have been removed, since now (as already pointed out in section 2.8.1), being in Frenet coordinates, the position on the xy plane is determined by means of the states s and d, making x and y redundant.
For instance, the state z has not been removed, since the Frenet coordinates, in this formulation, represent only the planar motion, so the state z is necessary to describe the full 3D motion of the quadrotor.

2.9.1. Model discretization

To control the quadrotor via MPC algorithms, the system model must be *discretized* (i.e. transformed from continuous-time to discrete-time). Several discretization methods can

be used; for the sake of simplicity, the *Forward Euler method* will be used.
Denoting the continuous-time Frenet model of the quadrotor (2.91) as:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_Q^{(CT)}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.92}$$

discretizing we obtain:

$$\begin{aligned} \dot{\boldsymbol{x}} &\approx \frac{\boldsymbol{x}(k+1) - \boldsymbol{x}(k)}{T} = \boldsymbol{f}_Q^{(CT)}(\boldsymbol{x}(k), \boldsymbol{u}(k)) \\ \Rightarrow \boldsymbol{x}(k+1) &= \boldsymbol{x}(k) + \boldsymbol{f}_Q^{(CT)}(\boldsymbol{x}(k), \boldsymbol{u}(k))T \equiv \boldsymbol{f}_Q^{(DT)}(\boldsymbol{x}(k), \boldsymbol{u}(k)) \equiv \boldsymbol{f}_Q(\boldsymbol{x}(k), \boldsymbol{u}(k)) \end{aligned} \tag{2.93}$$

where k is the discrete time, with time step T.

For compactness, the discrete time can be also written as a subscript:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_Q(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.94}$$

(2.94) denotes the discrete-time dynamic model of the quadrotor, expressed in Frenet coordinates.

2.9.2. Observations

We have introduced Frenet coordinates to make it simple to describe the track shape and width as state constraints in the MPC optimization problem.
Specifically, in the context of our problem, it is reasonable to choose $\boldsymbol{\gamma}$ as the centerline of the track. In this way, the shape of the track is described by its curvature $K(s)$, which is embedded in the system model, while the width of the track is defined by simple bound constraints on the state d (i.e. $-d_{lim} \leq d \leq d_{lim}$, where d_{lim} is the track width from centerline to border and $W_{track} = 2 \cdot d_{lim}$ is the track width from inner to outer border).

Frenet coordinates makes simpler also the formulation of tracking MPC problems, which are aimed at tracking a specific trajectory inside the track.
In particular, if we want the quadrotor to track the centerline, the MPC problem can be formulated as a tracking MPC with constant reference, where the reference state (limited to Frenet coordinates) is $(s_r, d_r) = (L_{track}, 0)$, where L_{track} is the total length of the track, from start to finish line (see Chapter 3 for more details).

The price of using the Frenet coordinate system is that any linear Cartesian model becomes nonlinear in Frenet coordinates. Therefore, in principle, in order to control the

quadrotor, a *Nonlinear Model Predictive Control* (NMPC) algorithm has to be used. However, as we will see in Chapter 3, we can control the quadrotor also through Linear MPC, by employing, in the MPC optimization problem, an affine time-variant (ATV) version of the nonlinear quadrotor model (see § 3.6.2 for more details).

Nonetheless, the real value of Frenet coordinates is the possibility to convert the non-convex and nonlinear Cartesian constraints on the track shape and width in simple bound constraints, which are convex and very well handled by all solvers.

3

Model Predictive Control for quadrotor trajectory tracking

3.1. Introduction to MPC and NMPC

Model predictive control (MPC) is an advanced and flexible control method that founds application in a wide range of fields, from industrial processes to power electronics devices. MPC relies on the knowledge of the dynamic model of the system to be controlled and provides numerous advantages with respect to other control techniques: it can be easily formulated for general MIMO systems, it allows to manage systematically states and inputs saturations, and, under certain conditions, it ensures the asymptotic stability of the closed-loop system.

MPC, differently from many other control methods, does not rely on a static control law (for example, in the form of a transfer function $C(s)$), but follows algorithmically some defined steps to generate the control inputs; for this reason, typically we denote MPC as a *control algorithm* rather than a control law.

At each time step k (in which the system will be at state $\boldsymbol{x}_k$), the MPC control algorithm, knowing the system model, simulates (i.e. predicts) how the system states would change by applying any possible input sequence over the next N time steps (where N is called *prediction horizon*). Among all the trajectories generated by these input sequences, it

chooses the "best" one according to some optimization criterion; typically, the optimal predicted state trajectory and input sequence are chosen as those minimizing a certain cost function.

The control algorithm, then, applies to the system only the first input of the predicted sequence, making the system evolve to the next state $\boldsymbol{x}_{k+1}$; this strategy is called *receding horizon control*.

We see that, at each time step, the MPC control algorithm follows two steps:

- *prediction*, in which it solves an online optimization problem (which is also called *control problem*), knowing the current state of the system, obtaining an optimal predicted state trajectory and input sequence;
- *control*, in which it applies to the system the first of the optimal predicted inputs, obtaining the next state.

Then, the algorithm is iterated until the full closed-loop state trajectory is generated.

Since the MPC prediction step relies on solving an optimization problem, we can include in it some *constraints* on the states and inputs values; in this way, through MPC, we can systematically manage the trade-off between performance and control effort.

MPC denotes a whole class of different algorithms, based on the same principle of prediction and control, but applied to different types of systems and with peculiar properties.
The most general MPC algorithm is *Nonlinear Model Predictive Control* (NMPC), which, as the name suggests, can be used to control generic nonlinear MIMO systems, and allows to specify nonlinear constraints in the optimization problem.

In the following sections, we will focus our attention on the most general NMPC framework.

3.2. NMPC theoretical formulation

In this section, we provide all the fundamental concepts that are required to formulate the NMPC control problem.

Consider the nonlinear discrete-time system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{3.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ are the system states and $\boldsymbol{u} \in \mathbb{R}^m$ are the system inputs; the generic nonlinear function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ is assumed to be continuous.

To formulate the NMPC optimization problem (OP), we need to define the following entities:

- the *optimization variables*;
- the *cost function* (i.e. the function to be optimized);
- the *constraints* on the optimization variables.

3.2.1. Optimization variables

Since in the prediction phase the NMPC algorithm needs to simulate the system states evolution over all the possible input sequences, the system equations (3.1) have to be embedded inside the optimization problem for each time instant of the prediction horizon.

To include these equations, we can formulate the NMPC problem in 2 forms: *explicit prediction form* and *implicit prediction form* [10].

- The explicit prediction form expresses the predicted states as function of the given initial state and the predicted inputs. The expressions of the predicted states are therefore expanded using the system equations, making appear (in the cost function and in the constraints) only the initial state and the predicted inputs.
  Therefore, the optimization variables for this problem are only the predicted inputs.
- The implicit prediction form, instead, considers as optimization variables both the predicted states and the predicted inputs, inserting the system equations as equality constraints of the optimization problem for each time instant of the prediction horizon.
  Although this formulation yields to a larger number of optimization variables, the optimization problem has a better structure and a lot of sparsity, which are two properties that are well exploited by solvers.

For our needs, we will use the implicit prediction form, meaning that the system equations will be inserted as equality constraints for the predicted states and predicted inputs optimization variables.

The optimization variables will then be the predicted states and predicted inputs over the prediction horizon. For a generic time instant k, they are denoted as follows:

$$\{\boldsymbol{x}_{t|k}\}_{t=0}^{N}, \quad \{\boldsymbol{u}_{t|k}\}_{t=0}^{N-1} \tag{3.2}$$

where N is the prediction horizon.

The notation "$t|k$" has the following meaning: t is the time instant of the optimization variable over the prediction horizon $t \in [0, 1, ..., N]$; k, instead, denotes the global time instant in which the system is (meaning that the current state of the system is $\boldsymbol{x}_k$).

Note 3.1 This implies that $\boldsymbol{x}_k = \boldsymbol{x}_{0|k}$, since the current state of the system is used as initial state to simulate all the possible state trajectories over the prediction horizon.

We denote the predicted state trajectory and the predicted input sequence as:

$$\boldsymbol{X}_{[0,N]|k} \equiv \boldsymbol{X}_k = (\boldsymbol{x}_{t|k})_{t=0}^{N} = (\boldsymbol{x}_{0|k}, \boldsymbol{x}_{1|k}, ..., \boldsymbol{x}_{N|k}) \tag{3.3a}$$

$$\boldsymbol{U}_{[0,N-1]|k} \equiv \boldsymbol{U}_k = (\boldsymbol{u}_{t|k})_{t=0}^{N-1} = (\boldsymbol{u}_{0|k}, \boldsymbol{u}_{1|k}, ..., \boldsymbol{u}_{N-1|k}) \tag{3.3b}$$

3.2.2. Cost function

The general expression of the cost function is the following:

$$J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = J_{[0,N-1]}(\boldsymbol{X}_k, \boldsymbol{U}_k) + V(\boldsymbol{x}_{N|k}) \equiv \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + V(\boldsymbol{x}_{N|k}) \tag{3.4}$$

J_{NMPC} is the *cost function*, h is the *stage cost function*, and V is the *terminal cost function*. V is not mandatory in the definition of the cost function, but it required when we want to ensure the asymptotic stability of the NMPC algorithm (see § 3.4); $J_{[0,N-1]}$ represents the NMPC cost function without terminal cost.

We see that the function J_{NMPC} to be optimized depends on all the predicted states and predicted inputs over the prediction horizon $(\boldsymbol{X}_k, \boldsymbol{U}_k)$.

The cost function is constructed with the purpose of formulating a *tracking NMPC problem*. This means that the solution of the optimization problem leads to a predicted

input sequence that makes the state trajectory to pursue a reference state $\boldsymbol{x}_r$, that will be eventually reached for $k \to \infty$.

To ensure that the NMPC problem accomplishes the reference tracking, we must satisfy the following requisites [5]:

1) It must exist an input $\boldsymbol{u}_r$ such that the couple $(\boldsymbol{x}_r, \boldsymbol{u}_r)$ is an equilibrium point of the open-loop system (3.1):
$$\boldsymbol{x}_r = \boldsymbol{f}(\boldsymbol{x}_r, \boldsymbol{u}_r) \tag{3.5}$$

2) The stage cost function $h(\boldsymbol{x}, \boldsymbol{u})$ should penalize the distance of an arbitrary state $\boldsymbol{x}$ to $\boldsymbol{x}_r$.
In addition, it is often desired to penalize also the input $\boldsymbol{u}$. This can be useful for computational reasons, since optimization problems may be easier to solve if the input variable is also penalized; moreover, penalizing $\boldsymbol{u}$ allows to tune the control signal effort, thus avoiding the generation of input signals requiring a high expense of energy to be applied.
To ensure that h penalizes the distance from $\boldsymbol{x}_r$ for all $\boldsymbol{x} \in \mathbb{R}$, we require h to be positive definite in $(\boldsymbol{x}_r, \boldsymbol{u}_r)$:
$$h(\boldsymbol{x}_r, \boldsymbol{u}_r) = 0, \quad h(\boldsymbol{x}, \boldsymbol{u}) > 0, \quad \forall (\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^n \times \mathbb{R}^m \setminus \{(\boldsymbol{x}_r, \boldsymbol{u}_r)\} \tag{3.6}$$
Sometimes, the input $\boldsymbol{u}_r$ cannot be easily determined; in these cases, the function h is defined as positive definite in $(\boldsymbol{x}_r, \boldsymbol{0})$. In this way, the input $\boldsymbol{u}$ is penalized considering its distance from $\boldsymbol{0}$ (and not from $\boldsymbol{u}_r$).
Since the penalization of $\boldsymbol{u}$ is not a mandatory condition for the tracking NMPC to work, the algorithm will still accomplish the tracking of $\boldsymbol{x}_r$ also with this cost function.

Without loss of generality, all the previous considerations can be extended to a time-variant state $\boldsymbol{x}_r(k)$, which is commonly denoted as reference trajectory.

A typical choice for the cost function is the following:
$$J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) = \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} \tag{3.7}$$
where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are diagonal matrices.

The cost function is quadratic and satisfies (3.6) to ensure reference tracking. By tuning the weight matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, it is possible to regulate the trade-off between the speed of convergence of the system to the reference state $\boldsymbol{x}_r$ and the amplitude of the input signal (i.e. the control effort). Specifically, the higher is $\|\boldsymbol{Q}\|$ (where $\|\cdot\|$ denotes an induced matrix norm), the faster the system will converge to the reference state (typically causing an increase in amplitude of the input signal); instead, the higher is $\|\boldsymbol{R}\|$, the lower will be the amplitude of the input signal (typically causing a reduction of the convergence speed).

3.2.3. Constraints

As already mentioned in § 3.1, since NMPC relies on solving an optimization problem, this one can be formulated as a *constrained optimization problem*, meaning that we can impose to find the optimal predicted states and inputs within specific subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$.

This means that, in the formulation of the NMPC problem, we can include constraints in the following form:

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \quad t = 0, 1, ..., N-1 \tag{3.8a}$$

$$\boldsymbol{u}_{t|k} \in \mathcal{U}, \quad t = 0, 1, ..., N-1 \tag{3.8b}$$

$$\boldsymbol{x}_{N|k} \in \mathcal{X}_F \tag{3.8c}$$

where $\mathcal{X} \subset \mathbb{R}^n$, $\mathcal{U} \subset \mathbb{R}^m$, and $\mathcal{X}_F \subset \mathbb{R}^n$ are closed sets.

Typically, constraints (3.8a) and (3.8b) are expressed in the form of nonlinear inequality constraints:

$$\boldsymbol{c}_x(\boldsymbol{x}_{t|k}) \leq \boldsymbol{0}, \quad t = 0, 1, ..., N-1 \tag{3.9a}$$

$$\boldsymbol{c}_u(\boldsymbol{u}_{t|k}) \leq \boldsymbol{0}, \quad t = 0, 1, ..., N-1 \tag{3.9b}$$

We see that the constraint (3.8c) regarding the *terminal state* $\boldsymbol{x}_{N|k}$ is defined separately. This constraint, called *terminal constraint*, is not mandatory in the formulation of the NMPC problem, but is required when we want to ensure asymptotic stability and recursive feasibility of the NMPC algorithm (see § 3.4).

3.3. NMPC optimization problem

In this section, by using all the concepts that have been described above, we provide the formulation of the *NMPC control problem*, which, as already mentioned, takes the form of an optimization problem (more specifically, a finite-horizon constrained optimal control problem):

NMPC optimization problem

$$(\boldsymbol{X}_k^*, \boldsymbol{U}_k^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k}{\arg\min}\, J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k)$$

$$J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = J_{[0,N-1]}(\boldsymbol{X}_k, \boldsymbol{U}_k) + V(\boldsymbol{x}_{N|k}) =$$

$$= \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + V(\boldsymbol{x}_{N|k}) \tag{3.10a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \quad t = 0, 1, ..., N-1 \tag{3.10b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k \tag{3.10c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \quad \boldsymbol{u}_{t|k} \in \mathcal{U}, \quad t = 0, 1, ..., N-1 \tag{3.10d}$$

$$\boldsymbol{x}_{N|k} \in \mathcal{X}_F \tag{3.10e}$$

The problem optimization variables are:

$$\boldsymbol{X}_k = (\boldsymbol{x}_{0|k}, \boldsymbol{x}_{1|k}, ..., \boldsymbol{x}_{N|k})$$
$$\boldsymbol{U}_k = (\boldsymbol{u}_{0|k}, \boldsymbol{u}_{1|k}, ..., \boldsymbol{u}_{N-1|k}) \tag{3.11}$$

representing the predicted state trajectory and the predicted input sequence, as in (3.3).

(3.10b) and (3.10c) represent respectively the system equations and the initial condition; (3.10d) are the states and inputs constraints; (3.10e) is the terminal constraint.

The most basic version of the problem does not include the terminal cost function V and the terminal constraint (3.10e):

$$(\boldsymbol{X}_k^*, \boldsymbol{U}_k^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k}{\arg\min}\, J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k)$$

$$J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = J_{[0,N-1]}(\boldsymbol{X}_k, \boldsymbol{U}_k) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) \tag{3.12a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \quad t = 0, 1, ..., N-1 \tag{3.12b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k \tag{3.12c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \quad \boldsymbol{u}_{t|k} \in \mathcal{U}, \quad t = 0, 1, ..., N-1 \tag{3.12d}$$

The optimal predicted state trajectory and the optimal predicted input sequence, which are indeed the solutions of the optimization problem (3.10), are denoted as:

$$\begin{aligned} \boldsymbol{X}_k^* &= (\boldsymbol{x}_{0|k}^*, \boldsymbol{x}_{1|k}^*, ..., \boldsymbol{x}_{N|k}^*) \\ \boldsymbol{U}_k^* &= (\boldsymbol{u}_{0|k}^*, \boldsymbol{u}_{1|k}^*, ..., \boldsymbol{u}_{N-1|k}^*) \end{aligned} \tag{3.13}$$

while the optimal value of the cost function is denoted as $J_{NMPC|k}^*$; this optimal cost can be also expressed specifying the initial state $\boldsymbol{x}_k$: $J_{NMPC}^*(\boldsymbol{x}_k)$.

The control algorithm consists in computing, at time k, the input sequence $\boldsymbol{U}_k^*$ and applying to system (3.1) only the first element of it:

$$\boldsymbol{u}_k \doteq \boldsymbol{u}_{0|k}^* \tag{3.14}$$

Then, the next state $\boldsymbol{x}_{k+1}$ is computed and is used to initialize the new optimization problem at time $k+1$: $\boldsymbol{x}_{0|k+1} \doteq \boldsymbol{x}_{k+1}$.

The control strategy, as already mentioned in § 3.1, is called *receding horizon* control.

At the end of the algorithm, the obtained closed-loop trajectory and input sequence are denoted as:

$$\begin{aligned} \boldsymbol{X} &= (\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_k, ...) \\ \boldsymbol{U} &= (\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_k, ...) \end{aligned} \tag{3.15}$$

Note 3.2 In practical applications, as we will see in the following, the closed-loop trajectory has a finite time duration T, which is the time instant at which the NMPC algorithm is terminated (the algorithm is terminated when the state $\boldsymbol{x}_T$ satisfies a certain exit condition, that will be formulated in § 3.5):

$$\begin{aligned} \boldsymbol{X} &= (\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_k, ..., \boldsymbol{x}_T) \\ \boldsymbol{U} &= (\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_k, ..., \boldsymbol{u}_{T-1}) \end{aligned} \tag{3.16}$$

We can state that (3.10) and (3.14) fully describe the *NMPC control algorithm*; together with (3.1), they fully describe also the closed-loop system (i.e. the system controlled by the NMPC algorithm).

3.4. NMPC properties

After having shown the formulation of the NMPC optimization problem, we now provide some fundamental theorems associated to NMPC.
Specifically, we will prove, as briefly mentioned in the previos sections, how to ensure asymptotic stability and recursive feasibility of the NMPC algorithm, by including in the control problem a suitable terminal constraint and terminal cost function.

3.4.1. Recursive feasibility

The NMPC problem is *recursively feasible* if, for all the feasible initial states $\boldsymbol{x}_0$, the NMPC problem is also feasible for any other state $\boldsymbol{x}_k$, $k \geq 1$, of the closed-loop trajectory.

Typically, NMPC problems, even if they are feasible at $k = 0$, may become infeasible at one of the next time instants, since the set of initial states leading to a feasible closed-loop trajectory (i.e. recursive feasibility) is in general a subset of the one containing the feasible initial states for open-loop prediction [7].

Assumption 3.1 The terminal set $\mathcal{X}_F$ of (3.10e) is a *control invariant set* for the system (3.1). Specifically, $\mathcal{X}_F$ is a control invariant set for (3.1) if, for $\boldsymbol{x}_k \in \mathcal{X}_F$, it always exists an input $\boldsymbol{u}'$ such that:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}') \in \mathcal{X}_F \tag{3.17}$$

Since, for (3.5), we have imposed that $\boldsymbol{x}_r$ is an equilibrium point of (3.1), then the smallest control invariant set containing $\boldsymbol{x}_r$ is:

$$\mathcal{X}_F = \{\boldsymbol{x}_r\} \tag{3.18}$$

since, with $\boldsymbol{u}' = \boldsymbol{u}_r$, $\boldsymbol{f}(\boldsymbol{x}_r, \boldsymbol{u}') = \boldsymbol{x}_r \in \mathcal{X}_F$.

Note 3.3 Granting recursive feasibility means that the set of feasible initial states becomes a control invariant set for system (3.1) [7]; this set will also coincide with the one containing the initial states leading to a feasible closed-loop trajectory.

Theorem 3.1 NMPC recursive feasibility

Consider the following NMPC problem:

$$(\boldsymbol{X}_k^*, \boldsymbol{U}_k^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k}{\arg\min}\, J_{_{NMPC}}(\boldsymbol{X}_k, \boldsymbol{U}_k)$$

$$J_{_{NMPC}}(\boldsymbol{X}_k, \boldsymbol{U}_k) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) \tag{3.19a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \quad t = 0, 1, ..., N-1 \tag{3.19b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k \tag{3.19c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \quad \boldsymbol{u}_{t|k} \in \mathcal{U}, \quad t = 0, 1, ..., N-1 \tag{3.19d}$$

$$\boldsymbol{x}_{N|k} \in \mathcal{X}_{_F} \tag{3.19e}$$

where the terminal constraint (3.19e) has been inserted.
The system (3.1) is controlled by the NMPC algorithm (3.19) and (3.14).

If Assumption 3.1 holds, then the NMPC is recursively feasible for all $k \geq 0$ [7] [5].

Proof By definition of recursive feasibility, we assume that the initial state $\boldsymbol{x}_0$ is feasible, i.e. it exists a feasible optimal predicted state trajectory and input sequence:

$$\begin{aligned} \boldsymbol{X}_0^* &= (\boldsymbol{x}_{0|0}^*, \boldsymbol{x}_{1|0}^*, ..., \boldsymbol{x}_{N|0}^*) \\ \boldsymbol{U}_0^* &= (\boldsymbol{u}_{0|0}^*, \boldsymbol{u}_{1|0}^*, ..., \boldsymbol{u}_{N-1|0}^*) \end{aligned} \tag{3.20}$$

Assume now that at time k the NMPC (3.19) and (3.14) is feasible; this means that it exists a feasible optimal predicted state trajectory and input sequence:

$$\begin{aligned} \boldsymbol{X}_k^* &= (\boldsymbol{x}_{0|k}^*, \boldsymbol{x}_{1|k}^*, ..., \boldsymbol{x}_{N|k}^*) \\ \boldsymbol{U}_k^* &= (\boldsymbol{u}_{0|k}^*, \boldsymbol{u}_{1|k}^*, ..., \boldsymbol{u}_{N-1|k}^*) \end{aligned} \tag{3.21}$$

Since the terminal constraint (3.19e) is present in the NMPC problem (3.19), the terminal state $\boldsymbol{x}_{N|k}^*$ will be inside $\mathcal{X}_{_F}$.

For Assumption 3.1, $\mathcal{X}_{_F}$ is a control invariant set for system (3.1); therefore, it exists an input $\boldsymbol{u}'$ such that:

$$\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}_{N|k}^*, \boldsymbol{u}') \in \mathcal{X}_{_F} \tag{3.22}$$

For the receding horizon control (3.14), we have that:

$$\boldsymbol{u}_k \doteq \boldsymbol{u}^*_{0|k} \tag{3.23}$$

Therefore, for Note 3.1, at time $k+1$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{f}(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) = \boldsymbol{x}^*_{1|k} \tag{3.24}$$

At time $k+1$, the input sequence:

$$(\boldsymbol{u}^*_{1|k},\ \ \boldsymbol{u}^*_{2|k},\ \ ...,\ \ \boldsymbol{u}^*_{N-1|k},\ \ \boldsymbol{u}') \tag{3.25}$$

and the related state trajectory:

$$(\boldsymbol{x}^*_{1|k},\ \ \boldsymbol{x}^*_{2|k},\ \ ...,\ \ \boldsymbol{x}^*_{N-1|k},\ \ \boldsymbol{x}^*_{N|k},\ \ \boldsymbol{x}') \tag{3.26}$$

satisfy input and state constrains (3.19b)-(3.19e). Therefore, (3.25)-(3.26) is a feasible solution for the NMPC (3.19) and (3.14) at time $k+1$.

We showed that:

- with $\boldsymbol{x}_0$ belonging to the feasible initial states, the NMPC is feasible at time $k=0$;
- if the NMPC is feasible at time k, then the NMPC is feasible at time $k+1$.

Thus, we conclude by induction that the NMPC in (3.19) and (3.14) is feasible $\forall k \geq 0$ [7]. ■

3.4.2. Asymptotic stability

Theorem 3.2 NMPC asymptotic stability

Consider the following NMPC problem:

$$(\boldsymbol{X}^*_k, \boldsymbol{U}^*_k) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k}{\arg\min}\, J_{_{NMPC}}(\boldsymbol{X}_k, \boldsymbol{U}_k)$$

$$J_{_{NMPC}}(\boldsymbol{X}_k, \boldsymbol{U}_k) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + V(\boldsymbol{x}_{N|k}) \tag{3.27a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}),\ \ t = 0, 1, ..., N-1 \tag{3.27b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k \tag{3.27c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \;\; \boldsymbol{u}_{t|k} \in \mathcal{U}, \;\; t = 0, 1, ..., N-1 \tag{3.27d}$$

$$\boldsymbol{x}_{N|k} \in \mathcal{X}_F \tag{3.27e}$$

where both the terminal cost function V and the terminal constraint (3.27e) have been inserted.
The system (3.1) is controlled by the NMPC algorithm (3.27) and (3.14).

If:

- the stage cost function h is positive definite in $(\boldsymbol{x}_r, \boldsymbol{u}_r)$,
- the terminal set $\mathcal{X}_F$ is a control invariant set for system (3.1) and contains $\boldsymbol{x}_r$,
- the terminal cost function V is a Lyapunov function in $\boldsymbol{x}_r$, defined on the terminal set $\mathcal{X}_F$, and such that, for $\boldsymbol{x}_k, \boldsymbol{x}_{k+1} \in \mathcal{X}_F$:

$$V(\boldsymbol{x}_{k+1}) - V(\boldsymbol{x}_k) \leq -h(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{3.28}$$

then $\boldsymbol{x}_r$ is an asymptotically stable equilibrium point of the closed-loop system (3.1), (3.27) and (3.14) [7] [5].

Proof To prove the asymptotic stability of $\boldsymbol{x}_r$, we can show, according to *Lyapunov's direct method*, that the optimal cost $J^*_{NMPC}(\boldsymbol{x})$ is a Lyapunov function for the equilibrium point $\boldsymbol{x}_r$ of the closed-loop system (3.1), (3.27) and (3.14).
Specifically, we have to show that:

- $\boldsymbol{x}_r$ is an equilibrium point of the closed-loop system;
- $J^*_{NMPC}(\boldsymbol{x})$ is positive definite in $\boldsymbol{x}_r$;
- $J^*_{NMPC}(\boldsymbol{x}_{k+1}) - J^*_{NMPC}(\boldsymbol{x}_k) < 0$, where $\boldsymbol{X} = (\boldsymbol{x}_0, ..., \boldsymbol{x}_k, \boldsymbol{x}_{k+1}, ...)$ is the closed-loop trajectory obtained via the LMPC control algorithm (3.27) and (3.14); equivalently, $J^*_{NMPC}(\boldsymbol{x})$ decreases along the closed-loop trajectory $\boldsymbol{X}$.

To prove that $\boldsymbol{x}_r$ is an equilibrium point of the closed-loop system, we have to show that, if $\boldsymbol{x}_k = \boldsymbol{x}_r$, the NMPC algorithm generates the optimal input $\boldsymbol{u}^*_{0|k} = \boldsymbol{u}_r$ and so $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_r, \boldsymbol{u}_r) = \boldsymbol{x}_r$.

By observing the problem (3.27), we see that, if $\boldsymbol{x}_k = \boldsymbol{x}_{0|k} = \boldsymbol{x}_r$, the input sequence $\boldsymbol{U}_k = (\boldsymbol{u}_r, ..., \boldsymbol{u}_r)$ and the related state trajectory $\boldsymbol{U}_k = (\boldsymbol{x}_r, ..., \boldsymbol{x}_r)$ give a cost function value $J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = 0$. This holds since:

- $\boldsymbol{X}_k$ and $\boldsymbol{U}_k$ satisfy all the system equations (3.27b);
- from (3.6), h is positive definite in $(\boldsymbol{x}_r, \boldsymbol{u}_r) \Rightarrow h(\boldsymbol{x}_r, \boldsymbol{u}_r) = 0$;
- for hypothesis, V is a Lyapunov function in $\boldsymbol{x}_r$, which means that it is positive definite in $\boldsymbol{x}_r \Rightarrow V(\boldsymbol{x}_r) = 0$.

Being 0 the lowest value that can be reached by J_{NMPC}, we conclude that, if $\boldsymbol{x}_k = \boldsymbol{x}_{0|k} = \boldsymbol{x}_r$, then $\boldsymbol{u}^*_{0|k} = \boldsymbol{u}_r$.
Therefore, $\boldsymbol{x}_r$ is an equilibrium point of the closed-loop system.

For the positive definiteness of h and V, $J^*_{NMPC}(\boldsymbol{x}) > 0$, $\forall \boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{x}_r\}$, and, for what we have shown in the previous paragraph, $J^*_{NMPC}(\boldsymbol{x}_r) = 0$, so $J^*_{LMPC}(\boldsymbol{x})$ is positive definite in $\boldsymbol{x}_r$.

Now, we need to show that $J^*_{NMPC}(\boldsymbol{x})$ is decreasing along the closed-loop trajectory $\boldsymbol{X}$.

For (3.14) and (3.27c), we have that $\boldsymbol{x}_{k+1} = \boldsymbol{x}^*_{1|k}$.

Given the optimal predicted input sequence and the related state trajectory in (3.13), the optimal cost is given by:

$$\begin{aligned} J^*_{NMPC}(\boldsymbol{x}_k) &= \min_{\boldsymbol{X}_k, \boldsymbol{U}_k} \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + V(\boldsymbol{x}_{N|k}) \\ &= \sum_{t=0}^{N-1} h(\boldsymbol{x}^*_{t|k}, \boldsymbol{u}^*_{t|k}) + V(\boldsymbol{x}^*_{N|k}) = \\ &= h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) + \sum_{t=1}^{N-1} h(\boldsymbol{x}^*_{t|k}, \boldsymbol{u}^*_{t|k}) + V(\boldsymbol{x}^*_{N|k}) \end{aligned} \tag{3.29}$$

Recalling the feasible trajectory (3.26) and input sequence (3.25), used to prove Theorem 3.1, we can compute the suboptimal cost $J'_{NMPC}(\boldsymbol{x}^*_{1|k})$ associated to this trajectory:

$$J'_{NMPC}(\boldsymbol{x}^*_{1|k}) = \sum_{t=1}^{N-1} h(\boldsymbol{x}^*_{t|k}, \boldsymbol{u}^*_{t|k}) + h(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}') + V(\boldsymbol{x}') \tag{3.30}$$

Being this cost suboptimal, it is for sure higher than the optimal cost $J^*_{NMPC}(\boldsymbol{x}^*_{1|k})$, associated to the optimal predicted state trajectory at time $k+1$:

$$J'_{NMPC}(\boldsymbol{x}^*_{1|k}) \geq J^*_{NMPC}(\boldsymbol{x}^*_{1|k}) \tag{3.31}$$

Then, by substituting (3.30) and (3.31) in (3.29), we obtain:

$$\begin{aligned} J^*_{NMPC}(\boldsymbol{x}_k) &= h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) + \sum_{t=1}^{N-1} h(\boldsymbol{x}^*_{t|k}, \boldsymbol{u}^*_{t|k}) + V(\boldsymbol{x}^*_{N|k}) \geq \\ &\geq h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) + J^*_{NMPC}(\boldsymbol{x}^*_{1|k}) - h(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}') - V(\boldsymbol{x}') + V(\boldsymbol{x}^*_{N|k}) \\ \Rightarrow J^*_{NMPC}(\boldsymbol{x}^*_{1|k}) - J^*_{NMPC}(\boldsymbol{x}_k) &\leq h(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}') - h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) + V(\boldsymbol{x}') - V(\boldsymbol{x}^*_{N|k}) \end{aligned} \tag{3.32}$$

Since $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}')$, $\boldsymbol{x}', \boldsymbol{x}^*_{N|k} \in \mathcal{X}_F$ and for the hypothesis on V, we have that:

$$V(\boldsymbol{x}') - V(\boldsymbol{x}^*_{N|k}) \leq -h(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}') \tag{3.33}$$

Therefore:

$$\begin{aligned} &J^*_{NMPC}(\boldsymbol{x}^*_{1|k}) - J^*_{NMPC}(\boldsymbol{x}_k) \leq \\ &\leq h(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}^*_{N|k}) - h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) + V(\boldsymbol{x}') - V(\boldsymbol{x}^*_{N|k}) \leq \\ &\leq -h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) \end{aligned} \tag{3.34}$$

Finally, we conclude that the optimal cost is a decreasing Lyapunov function along the closed-loop trajectory:

$$J^*_{NMPC}(\boldsymbol{x}_{k+1}) - J^*_{NMPC}(\boldsymbol{x}_k) \leq -h(\boldsymbol{x}_k, \boldsymbol{u}_k) < 0 \tag{3.35}$$

Therefore, we conclude that $\boldsymbol{x}_r$ is asymptotically stable [7] [13]. ■

Note 3.4 From Theorem 3.2, we can derive an useful corollary, that still ensures asymptotic stability while not requiring the inclusion of the terminal cost function V

Specifically, given the NMPC problem (3.27), if:

- the stage cost function h is positive definite in $(\boldsymbol{x}_r, \boldsymbol{u}_r)$,
- the terminal set $\mathcal{X}_F = \{\boldsymbol{x}_r\}$,

then $\boldsymbol{x}_r$ is an asymptotically stable equilibrium point of the closed-loop system (3.1), (3.27) and (3.14).

Since the terminal cost function is not present, it is equivalent to say that $V(\boldsymbol{x}) = 0$. Moreover, given (3.32) and noticing that, being $\mathcal{X}_F = \{\boldsymbol{x}_r\}$ and $\boldsymbol{x}^*_{N|k}, \boldsymbol{x}' \in \mathcal{X}_F$ $\Rightarrow \boldsymbol{x}^*_{N|k} = \boldsymbol{x}' = \boldsymbol{x}_r$, we have that:

$$\begin{aligned}
&J^*_{NMPC}(\boldsymbol{x}^*_{1|k}) - J^*_{NMPC}(\boldsymbol{x}_k) \leq \\
&\leq h(\boldsymbol{x}^*_{N|k}, \boldsymbol{u}^*_{N|k}) - h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) + V(\boldsymbol{x}') - V(\boldsymbol{x}^*_{N|k}) = \\
&= h(\boldsymbol{x}_r, \boldsymbol{u}_r) - h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) = \\
&= h(\boldsymbol{x}^*_{0|k}, \boldsymbol{u}^*_{0|k}) < 0 \\
&\Rightarrow J^*_{NMPC}(\boldsymbol{x}_{k+1}) - J^*_{NMPC}(\boldsymbol{x}_k) < 0
\end{aligned} \tag{3.36}$$

from which we conclude that $\boldsymbol{x}_r$ is asymptotically stable.

The reduction of the terminal set $\mathcal{X}_F$ to the single state $\boldsymbol{x}_r$ has the consequence, however, to reduce as well the set of initial feasible states $\boldsymbol{x}_0$ [7]. Therefore, it is necessary to increase the prediction horizon N to enlarge the feasible set.

Note 3.5 Theorem 3.1 provides a sufficient condition to ensure the recursive feasibility of the NMPC problem. The inclusion of the terminal constraint, however, can be avoided when a sufficiently large prediction horizon N is chosen. Indeed, it can be proved that if $N \geq \overline{N}$, the optimization problem is recursively feasible [2]. $\overline{N}$ is called *determinedness index* and is difficult to compute, so typically its value is guessed by trial and error.

3.5. NMPC algorithm

Finally, we can write the proper LMPC algorithm as follows:

Algorithm 3.1 Nonlinear Model Predictive Control

Inputs: $\boldsymbol{x}_0$; N (prediction horizon)
Outputs: $\boldsymbol{X}$; $\boldsymbol{U}$

(1) Store the initial state $\boldsymbol{x}_0$ in $\boldsymbol{X}$

(2) **For** $k = 0, 1, ...$:

(2.1) • **If** the exit condition on $\boldsymbol{x}_k$ is satisfied, **break** the cycle (at $k = T$);
• otherwise, continue

(2.2) Initialize the NMPC optimization problem (3.10) with $\boldsymbol{x}_k$

(2.3) Solve the NMPC optimization problem, obtaining the optimal predicted state trajectory $\boldsymbol{X}_k^* = (\boldsymbol{x}_{0|k}^*, \boldsymbol{x}_{1|k}^*, ..., \boldsymbol{x}_{N|k}^*)$ and optimal predicted input sequence $\boldsymbol{U}_k^* = (\boldsymbol{u}_{0|k}^*, \boldsymbol{u}_{1|k}^*, ..., \boldsymbol{u}_{N-1|k}^*)$

(2.4) Apply the input $\boldsymbol{u}_k \doteq \boldsymbol{u}_{0|k}^*$ to the system (3.1), as in (3.14), obtaining the next state $\boldsymbol{x}_{k+1}$

(2.5) Store $\boldsymbol{x}_{k+1}$ and $\boldsymbol{u}_k$ in $\boldsymbol{X}$ and $\boldsymbol{U}$

(3) Return the closed-loop state trajectory and input sequence:

$$\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_T)$$
$$\boldsymbol{U} = (\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{T-1})$$

Note 3.6 As already mentioned in Note 3.2, the algorithm requires an exit condition to obtain a finite-time trajectory. The exit condition at point (2.1) of the algorithm can be chosen to determine whether the state $\boldsymbol{x}_k$ is sufficiently close to the reference state $\boldsymbol{x}_r$. Therefore, it can be expressed as follows:

$$|\boldsymbol{x}_k - \boldsymbol{x}_r| \overset{?}{\leq} \delta \tag{3.37}$$

where δ is set by the user.

3.6. NMPC relaxation

In this section, we show some useful methods to relax the NMPC constraints. These relaxations make the problem more resistant to infeasibility and easier to solve, reducing in this way the online computation time.

3.6.1. Slack variables

When implementing a NMPC algorithm that uses the base version of the optimization problem (3.12) (i.e. no terminal constraint and terminal cost), as explained in Note 3.5 the only way to ensure recursive feasibility is to set a sufficiently high prediction horizon N. However, since high prediction horizons increase the number of optimization variables and, in turn, the online computation time, we would like to find an alternative way to prevent infeasibility while keeping N low.

The major cause of infeasibility of the optimization problem (3.12) is the impossibility to satisfy together the system equations constraints (3.12b) and the hard state constraints (3.8a)/(3.9a).
In this cases, it is often introduced a new optimization variable $\boldsymbol{e} \in \mathbb{R}^n$, called *slack variable*, in order to soften the state constraints (3.9a) and prevent the infeasibility of the optimization problem:

$$\boldsymbol{c}_x(\boldsymbol{x}_{t|k}) \leq \boldsymbol{e}, \quad t = 0, 1, ..., N-1 \tag{3.38}$$

If the state constraint is a linear inequality, then it can be rewritten as:

$$\begin{aligned} &\boldsymbol{A}_x \boldsymbol{x}_{t|k} \leq \boldsymbol{b}_x + \boldsymbol{e} \\ &\boldsymbol{A}_x (\boldsymbol{x}_{t|k} - \boldsymbol{A}_x^{-1} \boldsymbol{e}) \leq \boldsymbol{b}_x \\ &\boldsymbol{x}_{t|k} \in \mathcal{X} \oplus \boldsymbol{A}_x^{-1} \boldsymbol{e}, \quad t = 0, 1, ..., N-1 \end{aligned} \tag{3.39}$$

Adding the slack variable, the constraint is softened, permitting the system states to go slightly outside the set $\mathcal{X}$, preventing the infeasibility of the current optimization problem and allowing the algorithm to continue with the next iterations.

Being the slack variable an additional optimization variable of the problem, it must be inserted in the cost function, with a reference value $\boldsymbol{e}_r = \boldsymbol{0}$ and in general, for simplicity, with a quadratic term:

$$J(\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{e}) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \boldsymbol{e}^T \boldsymbol{K} \boldsymbol{e} \tag{3.40}$$

In this case, the higher is $\|\boldsymbol{K}\|$, the lower will be the softening of the constraints (i.e. the higher is $\|\boldsymbol{K}\|$, the higher $\boldsymbol{e}$ will tend to be very close to $\boldsymbol{0}$, meaning that the system will be allowed to exit from the state constraint by very little).

3.6.2. Nonlinear to affine time-variant system equations

Since we have formulated our NMPC problem in implicit prediction form, the system equations (3.1) are inserted in the optimization problem as equality constraints (3.10b).

Being such constraints nonlinear, the whole problem will be a NLP (Non-Linear Program), thus requiring a NLP solver. These solvers are typically very slow (since they have to deal with a huge range of possible OPs) and their performance is strongly dependent on the problem type, formulation, and complexity.
For our specific case, increasing the prediction horizon N will slow down the solver, since the number of optimization variables will be higher; moreover, this will slow it down even more since, the higher is N, the higher is the number of nonlinear equality constraints associated to the system equations.

To deal with this issue, a possible approach is to *linearize*, at each time instant k, the system equations around the current *operating point* of the system (i.e. the current state $\boldsymbol{x}_k$ and the previous input $\boldsymbol{u}_{k-1}$). In this way, the NMPC problem is solved using the linearized system equations at time k; such equations will be then updated at time $k+1$ with the new state $\boldsymbol{x}_{k+1}$ and the input $\boldsymbol{u}_k$.
The state update will be still computed using the nonlinear system equations, i.e. $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$.

For the linearization at time k we use the previous input $\boldsymbol{u}_{k-1}$, since the input $\boldsymbol{u}_k$ is still to be computed by the NMPC optimization problem at time k (i.e. $\boldsymbol{u}_k = \boldsymbol{u}^*_{0|k}$).

Being the operating point $(\boldsymbol{x}_k, \boldsymbol{u}_{k-1})$ not necessarily an equilibrium point of the system, the linearization provides a set of *Affine Time-Variant* (ATV) equations, as shown in the following.

Denoting as $(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1})$ the operating point, the linearization (neglecting higher order terms to achieve linearity) is done as follows:

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) &\approx \boldsymbol{f}(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1}) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1})\,(\boldsymbol{x}_k - \overline{\boldsymbol{x}}_k) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1})\,(\boldsymbol{u}_k - \overline{\boldsymbol{u}}_{k-1}) \equiv \\ &\equiv \boldsymbol{f}(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1}) + \boldsymbol{A}_k(\boldsymbol{x}_k - \overline{\boldsymbol{x}}_k) + \boldsymbol{B}_k(\boldsymbol{u}_k - \overline{\boldsymbol{u}}_{k-1}) \end{aligned} \tag{3.41}$$

where:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \equiv \boldsymbol{J}_{\boldsymbol{f},\boldsymbol{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \equiv \boldsymbol{A} \tag{3.42a}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \equiv \boldsymbol{J}_{\boldsymbol{f},\boldsymbol{u}} = \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \dots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \dots & \frac{\partial f_n}{\partial u_m} \end{pmatrix} \equiv \boldsymbol{B} \tag{3.42b}$$

are the Jacobian matrices of $\boldsymbol{f}$ wrt $\boldsymbol{x}$ and $\boldsymbol{u}$.

The ATV model can be then written as:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1}) + \boldsymbol{A}_k(\boldsymbol{x}_k - \overline{\boldsymbol{x}}_k) + \boldsymbol{B}_k(\boldsymbol{u}_k - \overline{\boldsymbol{u}}_{k-1}) = \\ &= \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \underbrace{\boldsymbol{f}(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1}) - \boldsymbol{A}_k\overline{\boldsymbol{x}}_k - \boldsymbol{B}_k\overline{\boldsymbol{u}}_{k-1}}_{\equiv \boldsymbol{c}_k} = \\ &= \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \boldsymbol{c}_k \end{aligned} \tag{3.43}$$

We see that this model is *time-variant*, since its matrices $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ depend on the discrete time k (specifically, they depend on the current state and previous input of the system), and *affine*, since the extra term $\boldsymbol{c}_k$ is present ($\boldsymbol{c}_k$ would be equal to $\boldsymbol{0}$ if the operating point $(\overline{\boldsymbol{x}}_k, \overline{\boldsymbol{u}}_{k-1})$ is also an equilibrium point of the system).

With this approximation, the nonlinear equality constraints (3.10b) become linear:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{A}_k\boldsymbol{x}_{t|k} + \boldsymbol{B}_k\boldsymbol{u}_{t|k} + \boldsymbol{c}_k, \quad t = 0, 1, ..., N-1 \tag{3.44}$$

This means that, if the cost function (3.10a) is quadratic as in (3.7), the NMPC problem becomes a QP (Quadratic Program), meaning that now it requires just a QP solver. These solvers are much quicker that NP solvers and their performance is reliable and not significantly altered by the problem complexity [1].

3.7. NMPC for quadrotors

We now apply all the NMPC concepts that have been described until now for the purpose of controlling the quadrotor described in Chapter 2.

Let's consider the nonlinear discrete-time dynamic model of the quadrotor in Frenet coordinates (2.94):

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_Q(\boldsymbol{x}_k, \boldsymbol{u}_k, K(s_k)) \tag{3.45}$$

in which we make explicit the dependence on the curvature function $K(s)$ for a reason that will be explained in § 3.7.2.

The goal is to formulate a basic NMPC algorithm (3.12) and (3.14) (i.e. with no terminal constraint and terminal cost) that makes the quadrotor to *track a predefined trajectory* within the race track.
For the observations made in § 2.9.2, this task can be achieved by formulating a tracking NMPC problem with constant reference if we want, for example, to track the centerline or, in general, any other trajectory that keeps a constant lateral distance d from the centerline. We may want also, for a reason that will be explained in Note 4.8 of Chapter 4, to generate a trajectory whose lateral distance d oscillates within the range $[-d_{lim}, d_{lim}]$, where d_{lim} is the track width from centerline to border.
The NMPC problem can be then relaxed as shown in § 3.6, to obtain a Linear MPC version of it, so to make it faster to be solved and resistant to infeasibility.

In the following, we construct every element of the NMPC problem, to eventually derive the final control problem and algorithm.

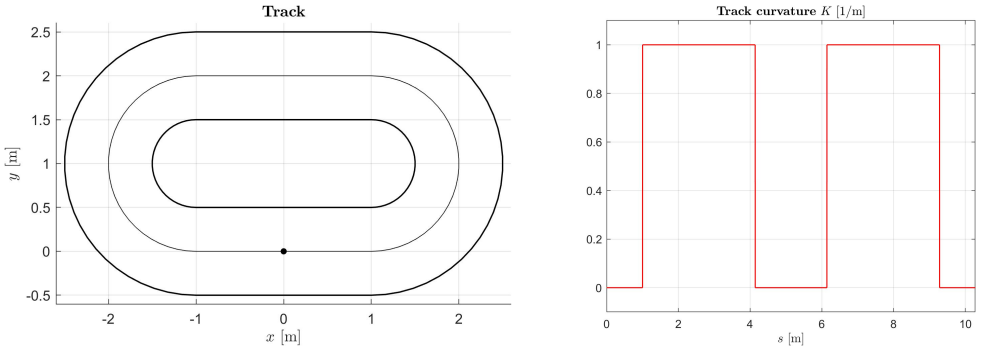
3.7.1. Track definition

As already mentioned in § 2.9.2, the race track is defined by means of three elements:

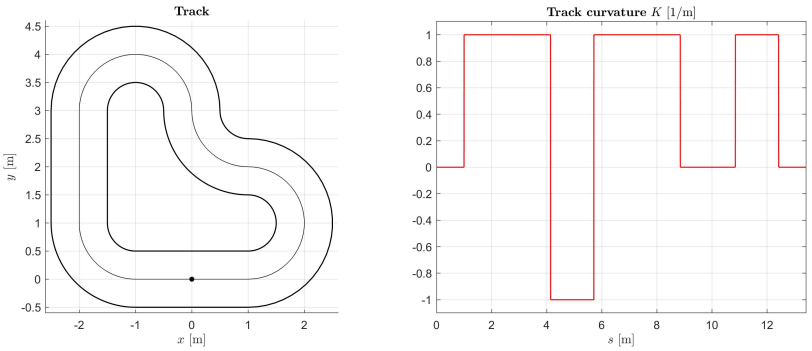
- the Frenet curve $\boldsymbol{\gamma}$, which is set to be the centerline of the track;
- the curvature function of $\boldsymbol{\gamma}$, $K(s)$, which is embedded in the quadrotor model (2.94);
- an upper-lower bound constraint on the state variable d, defining the width of the track, and, so, its lateral boundaries.

For what concerns the choice of $K(s)$, which defines the shape of the track, we consider it as belonging to the set of *constant piecewise functions*: this means that the track will be composed by a sequence of straight lines and circular curves, of any length and angle.

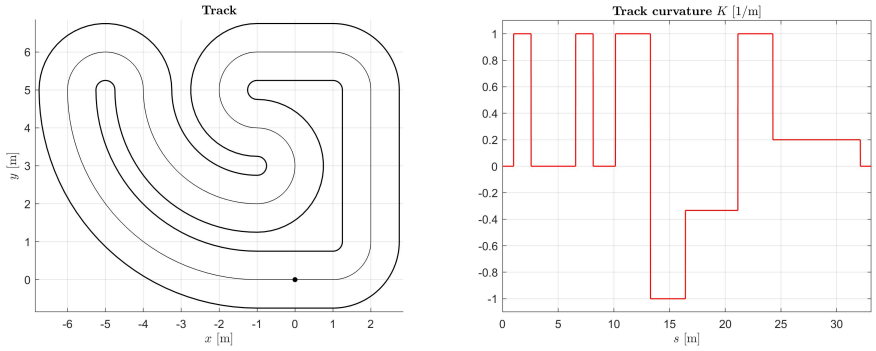
In Figure 3.1 are depicted three possible tracks having a constant piecewise curvature $K(s)$; specifically, these three tracks will be used in the simulations to test the algorithm (see Chapter 5).



(a) Track 1



(b) Track 2



(c) Track 3

Figure 3.1. Examples of tracks with a constant piecewise curvature function. These three tracks will be used in the simulations to test the algorithm.

3.7.2. Curvature propagation and relaxation

In the equations (2.91), we see that it appears the analytical curvature function $K(s)$ of the Frenet curve $\boldsymbol{\gamma}$. The expression of this function cannot be easily inserted in the system equations constraints (3.12b) of the NMPC problem, since:

- as previously said, $K(s)$ is typically a constant piecewise function;

- functions with complicated expressions inside the NMPC constraints are not supported by many solvers;
- if we want to use the ATV model, the nonlinear model equations need to be differentiated, therefore $K(s)$ should be at least differentiable once, which is not the case of constant piecewise functions.

Therefore, as suggested also in [3], we need to deal with $K(s)$ in a different way, as explained in the following.

The curvature $K(s)$ in the model equations will be considered as a *constant* parameter K, transforming (2.94) as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_Q(\boldsymbol{x}_k, \boldsymbol{u}_k, K(s_k)) \quad \rightarrow \quad \boldsymbol{x}_{k+1} = \hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k, \boldsymbol{u}_k, K) \tag{3.46}$$

At each time instant k, the value of the current curvature $K(s_k)$ will be passed to the NMPC optimization problem, which will set it as the constant value for K in the model equations constraints.
This means that the NMPC problem will compute the optimal predicted states and inputs assuming that, over its prediction horizon, the curvature is constant to its initial value at $s_k = s_{0|k}$. This action is denoted as *curvature propagation*.

This approach surely solves the issue of embedding the analytical $K(s)$ in the NMPC constraints. However, now a new problem arises. Let's assume that the curvature abruptly changes at a certain point of the track, for example during the transition from a straight line to a curve: the NMPC will "notice" the presence of the curve only at the end of the straight line, since we have imposed the curvature as constant in each NMPC optimization problem.
This issue eliminates all the advantages of having a predictive control strategy, since the prediction is falsified by imposing the constant curvature.

A possible solution to this problem (which leads to good results, as shown in Chapter 5), is denoted as *curvature relaxation* and consists in evaluating the current curvature $K = K(s_k)$ using a relaxed curvature function $\tilde{K}(s)$.

This relaxed function $\tilde{K}$ connects the constant piecewise segments of the original function K with third-order polynomials, ensuring the continuity of the derivatives in the junction points (Figure 3.2).

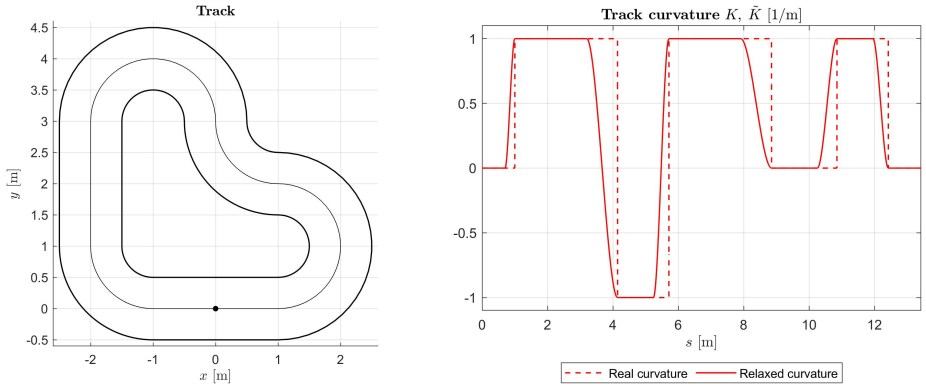


Figure 3.2. Track 2 with relaxed curvature function ($K_{rel} = 30\%$)

This connection using polynomials is particularly useful to reduce the steepness of vertical edges of stepwise curvatures: the gradual change of curvature in the relaxed function allows the NMPC to better predict the future change of curvature, even if the curvature is still considered constant in each optimization problem.

To quantify how much the curvature is relaxed, we define a parameter K_{rel}, called *curvature relaxation coefficient*, that is equal to the percentage of each step that has been "replaced" by the junction polynomial; in Figure 3.2, $K_{rel} = 30\%$, since, after the relaxation, the length of each step is reduced by 30%.

3.7.3. Cost function

For our NMPC problem, we use a quadratic cost function as the one shown in (3.7). However, to ensure a better behaviour of the quadrotor under control, an additional quadratic term is added to it:

$$J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \sum_{t=1}^{N-1} h'(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}, \boldsymbol{x}_{t-1|k}, \boldsymbol{u}_{t-1|k}) +$$

$$+ K_1 e_1^2 + K_2 e_2^2 =$$

$$= \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} + \tag{3.47a}$$

$$+ \sum_{t=1}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k})^T \boldsymbol{Q}_\Delta (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k}) + (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k})^T \boldsymbol{R}_\Delta (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k}) + \tag{3.47b}$$

$$+ K_1 e_1^2 + K_2 e_2^2 \tag{3.47c}$$

As already explained in § 3.2.2, the term (3.47a) of the cost function satisfies (3.6) to

ensure that the quadrotor tracks the reference state.

The term (3.47b) is very helpful to adjust the behaviour of the quadrotor, since it penalizes the variation (i.e. the discrete-time derivative) of the system states and inputs. By tuning the weight matrices $\boldsymbol{Q}_\Delta$ and $\boldsymbol{R}_\Delta$, it is possible to force the quadrotor to follow a smoother trajectory, without abrupt changes in its velocity and accelerations. Specifically, the higher is $\|\boldsymbol{Q}_\Delta\|$, the lower will be the difference between consecutive states (leading to smoother but slower trajectories); the higher is $\|\boldsymbol{R}_\Delta\|$, the lower will be the difference between consecutive inputs (preventing abrupt changes in the trajectories at the price of reducing speed and maneuverability).

The term (3.47c), finally, is related to two slack variables $e_1, e_2 \in \mathbb{R}$ that will be inserted in the problem constraints (see next section).

Reference state

For the purpose of trajectory tracking, the reference state $\boldsymbol{x}_r$ will be the following:

$$\boldsymbol{x}_r = \begin{pmatrix} z_r & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & L_{track} & d_r & \star \end{pmatrix} \tag{3.48}$$

In $\boldsymbol{x}_r$, the states denoted with "$\star$" are associated to a weight equal to 0 in the matrix $\boldsymbol{Q}$, so their value is not needed to be specified. Such states are then not penalized by the cost function.

We see that the reference values z_r and L_{track} define a trajectory with constant altitude and directed towards the finish line of the track.
Regarding the reference value for d_r, it can be either chosen constant, if we want to track the centerline or any other curve that keeps a constant lateral distance from it, or variable, if we want the trajectory to oscillate; specifically, in the latter case, d_r is defined as:

$$d_r(s_k) = c_o \cdot \frac{d_u - d_l}{2} \cdot \sin\left(n_o \cdot 2\pi \frac{s_k}{L_{track}}\right) + \frac{d_u + d_l}{2} \tag{3.49}$$

where s_k is the current position of the quadrotor, d_l and d_u are respectively the lower and upper bounds of the oscillation, n_o is the number of oscillations over the track length, and $c_o \in [-1, 1]$ is a coefficient for tuning the amplitude and direction of the oscillations.

The value $\psi_r = 45°$ ensures that the quadrotor will move in cross configuration; the values $\phi_r = \theta_r = 0°$ ensure that the quadrotor will not have high tilt angles, that, if too

large, may enter in the gimbal lock configuration (see Note 3.8).
The other values (related to Cartesian velocities and angular rates) are set to 0 to ensure that $\boldsymbol{x}_r$ is an equilibrium point of the open-loop system as in (3.5).

3.7.4. Constraints

The following constraints will be added to the NMPC problem:

- ATV model equations constraints (implicit prediction form);
- track boundaries, i.e. bounds on the lateral distance d;
- bounds on the altitude z;
- bounds on other state variables, in order to force the desired quadrotor behaviour during its motion.

The constraints will be expressed as follows:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{A}_k \boldsymbol{x}_{t|k} + \boldsymbol{B}_k \boldsymbol{u}_{t|k} + \boldsymbol{c}_k, \quad t = 0, 1, ..., N-1 \tag{3.50a}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k \tag{3.50b}$$

$$-d_{lim} - e_1 \leq d_{t|k} \leq d_{lim} + e_1, \quad t = 0, 1, ..., N-1 \tag{3.50c}$$

$$z_{(m)} - e_2 \leq d_{t|k} \leq z_{(M)} + e_2, \quad t = 0, 1, ..., N-1 \tag{3.50d}$$

$$v_x^{(m)} \leq v_{x,t|k} \leq v_x^{(M)}, \quad t = 0, 1, ..., N-1 \tag{3.50e}$$

$$v_y^{(m)} \leq v_{y,t|k} \leq v_y^{(M)}, \quad t = 0, 1, ..., N-1 \tag{3.50f}$$

$$v_z^{(m)} \leq v_{z,t|k} \leq v_z^{(M)}, \quad t = 0, 1, ..., N-1 \tag{3.50g}$$

$$e_1 \geq 0 \tag{3.50h}$$

$$e_2 \geq 0 \tag{3.50i}$$

where (m) stands for minimum value and (M) stands for maximum value.

Note 3.7 The ATV model for the quadrotor is computed using the new model (3.46), $\boldsymbol{x}_{k+1} = \hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k, \boldsymbol{u}_k, K)$, having the curvature expressed as a constant parameter K, independent from s. Therefore:

$$\boldsymbol{A}_k = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{x}}(\boldsymbol{x}_k, \boldsymbol{u}_{k-1}, K) \tag{3.51}$$

$$\boldsymbol{B}_k = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{u}}(\boldsymbol{x}_k, \boldsymbol{u}_{k-1}, K) \tag{3.52}$$

$$\boldsymbol{c}_k = \hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k, \boldsymbol{u}_{k-1}, K) - \boldsymbol{A}_k \boldsymbol{x}_k - \boldsymbol{B}_k \boldsymbol{u}_{k-1} \tag{3.53}$$

Constraints (3.50a)-(3.50b) are those associated to the ATV model equations.

Constraint (3.50c) is associated to the track boundaries/width ($W_{track} = 2 \cdot d_{lim}$); thus, it sets lower and upper bounds on the lateral distance d of the quadrotor. Together with the function $K(s)$, already embedded in the model, the race track is now fully described.
We see that the slack variable e_1, introduced in the cost function (3.47c), is used here to soften the constraint. This allows the quadrotor to slightly go outside the bounds, preventing possible infeasibility when the quadrotor approaches too closely the track borders.

Constraint (3.50d) sets lower and upper bounds on the altitude z of the quadrotor. The slack variable e_2 softens the constraint; this allows the quadrotor to slightly go outside the bounds, preventing possible infeasibility when the quadrotor approaches too closely the vertical borders.

Constraints (3.50e)-(3.50g) set lower and upper bounds on the Cartesian velocities of the quadrotor (wrt the base frame). These allow to both regulate the quadrotor average speed and to avoid that it gains too much speed, possibly making infeasible the next iterations of the NMPC.

Constraints (3.50h)-(3.50i), finally, set the slack variables e_1 and e_2 as non-negative, which is necessary to soften correctly constraints (3.50c)-(3.50d).

Note 3.8 Additional constraints may be added to set lower and upper bounds on the roll and pitch angles ϕ and θ (also called "tilt" angles) assumed by the quadrotor during its motion.

These constraints are particularly important if we want to nullify the risk of gimbal lock: as explained in Note 2.1, the dynamic model of the quadrotor suffers from gimbal lock/singularity when the pitch angle θ approaches $\pm 90°$; specifically, the transformation matrix $\boldsymbol{T}_{RPY}$ becomes singular, meaning that the numerical integration of the model equations will give erroneous RPY angles, since some elements of $\boldsymbol{T}_{RPY}^{-1}$ will diverge.
For this reason, commercial quadrotors are typically equipped with a security system

that turns off the quadrotor when the tilt angles exceed a threshold value. Additional constraints on ϕ and θ, therefore, take into account these threshold values.

In our MPC algorithm, we will not include such constraints, since it has been observed that, in all simulations (see Chapter 5), the quadrotor always stays far away from the gimbal lock configuration.

Note 3.9 It is important to notice that, now, being all the constraints linear equalities or inequalities and being the cost function quadratic, the optimization problem is a QP, meaning that it is now a Linear MPC problem. Therefore, from now on, we will use the term MPC instead of NMPC.

3.7.5. Optimization problem

We can now write the MPC optimization problem for quadrotor trajectory tracking:

MPC optimization problem for quadrotor trajectory tracking

$$(\boldsymbol{X}_k^*, \boldsymbol{U}_k^*, e_1^*, e_2^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2}{\arg\min} \; J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2)$$

$$J_{NMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2) = \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} +$$

$$+ \sum_{t=1}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k})^T \boldsymbol{Q}_\Delta (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k}) + (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k})^T \boldsymbol{R}_\Delta (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k}) +$$

$$+ K_1 e_1^2 + K_2 e_2^2 \tag{3.54a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{A}_k \boldsymbol{x}_{t|k} + \boldsymbol{B}_k \boldsymbol{u}_{t|k} + \boldsymbol{c}_k, \;\; t = 0, 1, ..., N-1 \tag{3.54b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k \tag{3.54c}$$

$$-d_{lim} - e_1 \leq d_{t|k} \leq d_{lim} + e_1, \;\; t = 0, 1, ..., N-1 \tag{3.54d}$$

$$z_{(m)} - e_2 \leq z_{t|k} \leq z_{(M)} + e_2, \;\; t = 0, 1, ..., N-1 \tag{3.54e}$$

$$v_x^{(m)} \leq v_{x,t|k} \leq v_x^{(M)}, \;\; t = 0, 1, ..., N-1 \tag{3.54f}$$

$$v_y^{(m)} \leq v_{y,t|k} \leq v_y^{(M)}, \;\; t = 0, 1, ..., N-1 \tag{3.54g}$$

$$v_z^{(m)} \leq v_{z,t|k} \leq v_z^{(M)}, \;\; t = 0, 1, ..., N-1 \tag{3.54h}$$

$$e_1 \geq 0 \tag{3.54i}$$

$$e_2 \geq 0 \tag{3.54j}$$

with:

$$\boldsymbol{A}_k = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{x}}(\boldsymbol{x}_k, \boldsymbol{u}_{k-1}, \tilde{K}(s_k)) \tag{3.54k}$$

$$\boldsymbol{B}_k = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{u}}(\boldsymbol{x}_k, \boldsymbol{u}_{k-1}, \tilde{K}(s_k)) \tag{3.54l}$$

$$\boldsymbol{c}_k = \hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k, \boldsymbol{u}_{k-1}, \tilde{K}(s_k)) - \boldsymbol{A}_k \boldsymbol{x}_k - \boldsymbol{B}_k \boldsymbol{u}_{k-1} \tag{3.54m}$$

3.7.6. Algorithm

The complete MPC algorithm for the quadrotor is composed by the MPC problem (3.54) and the receding horizon control law (3.14).

Note 3.10 It is worth noticing that, even though we have used in the MPC problem (3.54) the ATV model of the quadrotor, the next state $\boldsymbol{x}_{k+1}$ is obtained applying the optimal control input $\boldsymbol{u}_k = \boldsymbol{u}^*_{0|k}$ to the complete nonlinear model of the quadrotor (2.94):

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, K(s_k)) \tag{3.55}$$

Algorithm 3.2 MPC for quadrotor trajectory tracking

Inputs: $\boldsymbol{x}_0$; N (prediction horizon)
Outputs: $\boldsymbol{X}$; $\boldsymbol{U}$

(1) Store the initial state $\boldsymbol{x}_0$ in $\boldsymbol{X}$

(2) **For** $k = 0, 1, ...$:

(2.1) • **If** the exit condition on $\boldsymbol{x}_k$ is satisfied, **break** the cycle (at $k = T$);
• otherwise, continue

(2.2) Initialize the MPC optimization problem (3.54) with $\boldsymbol{x}_k$ and $\boldsymbol{u}_{k-1}$ (**if** $k = 0$, set $\boldsymbol{u}_{k-1} \doteq \boldsymbol{0}$)

(2.3) Solve the MPC optimization problem, obtaining the optimal predicted state trajectory $\boldsymbol{X}^*_k = (\boldsymbol{x}^*_{0|k}, \boldsymbol{x}^*_{1|k}, ..., \boldsymbol{x}^*_{N|k})$ and optimal predicted input

sequence $\boldsymbol{U}_k^* = (\boldsymbol{u}_{0|k}^*, \boldsymbol{u}_{1|k}^*, ..., \boldsymbol{u}_{N-1|k}^*)$

(2.4) Apply the input $\boldsymbol{u}_k \doteq \boldsymbol{u}_{0|k}^*$ to the system (2.94), as in (3.14), obtaining the next state $\boldsymbol{x}_{k+1}$

(2.5) Store $\boldsymbol{x}_{k+1}$ and $\boldsymbol{u}_k$ in $\boldsymbol{X}$ and $\boldsymbol{U}$

(3) Return the closed-loop state trajectory and input sequence:

$$\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_T)$$
$$\boldsymbol{U} = (\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{T-1})$$

As done in Note 3.6, we need to define the exit condition that stops the algorithm as soon as the current state $\boldsymbol{x}_k$ is sufficiently close to the reference state $\boldsymbol{x}_r$. For our needs, the exit condition is when the quadrotor has crossed the finish line; therefore, the algorithm should stop as soon as the value of the curvilinear abscissa s_k is greater or equal to the track length L_{track}:

$$s_k \overset{?}{\geq} L_{track} \tag{3.56}$$

4

Learning Model Predictive Control for quadrotor autonomous and optimal path planning

4.1. Introduction to LMPC

Learning Model Predictive Control (LMPC) was conceived to combine the features of classical Model Predictive Control to other control strategies that are able to learn from previous iterations of the control algorithm, improving in this way their closed-loop performance. One of these control methods is called Iterative Learning Control (ILC), in which the system to control always starts from the same initial condition and the controller objective is to track a given reference. The key aspect of ILC is that the control algorithm is run multiple times and, at each iteration, information from past iterations is used by the controller to *learn* from its previous results, improving in this way its current performance [15] [3]. Such control strategies are very well suited for systems performing iterative tasks.

However, the main limitation of ILC and other analogous control methods, as also mentioned in the Introduction, is that they are limited to reference tracking applications, where, in general, the main goal is to minimize the difference between the reference and

the output signal (i.e. the tracking error); moreover, the reference signal is known in advance and does not change at each iteration.
These limitations are overcome by LMPC, which is indeed configured as a reference-free iterative control algorithm, which learns from previous iterations to improve its performance over time. Specifically, the LMPC algorithm ensures that:

- defining as j-th iteration cost the objective function evaluated for the j-th closed-loop system trajectory, the j-th iteration cost does not increase compared to the $(j-1)$-th iteration cost;
- state and input constraints are satisfied at iteration j if they were satisfied at iteration $j-1$ (recursive feasibility);
- the final goal state is an asymptotically stable equilibrium point for the closed-loop system;
- as the number of iterations goes to infinity, the system converges to a steady-state trajectory that is the optimal solution of the corresponding infinite-horizon control problem.

4.2. LMPC theoretical formulation

In this section, we provide all the fundamental concepts that are required to formulate the LMPC control problem [15].

Consider the discrete-time system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{4.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ are the system states and $\boldsymbol{u} \in \mathbb{R}^m$ are the system inputs.
We assume that the generic nonlinear function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ is continuous and that the states and inputs of the system are subject to the constraints:

$$\boldsymbol{x}_k \in \mathcal{X}, \quad \boldsymbol{u}_k \in \mathcal{U}, \quad \forall k \geq 0 \tag{4.2}$$

where $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{U} \subset \mathbb{R}^m$ are closed sets.

At each iteration of the LMPC algorithm, a new state trajectory is generated; for the j-th iteration, we denote the vectors containing the inputs applied to the system (4.1)

and the corresponding states evolution (i.e. the state trajectory) as:

$$\boldsymbol{X}^j = (\boldsymbol{x}_0^j, \boldsymbol{x}_1^j, ..., \boldsymbol{x}_k^j, ...) \tag{4.3a}$$

$$\boldsymbol{U}^j = (\boldsymbol{u}_0^j, \boldsymbol{u}_1^j, ..., \boldsymbol{u}_k^j, ...) \tag{4.3b}$$

In (4.3), $\boldsymbol{x}_k^j$ and $\boldsymbol{u}_k^j$ denote the system states and the control inputs at time k of the j-th iteration.

We initially assume that, at every iteration, the closed-loop trajectories start from the same initial state:

$$\boldsymbol{x}_0^j = \boldsymbol{x}_S, \quad \forall j \geq 0 \tag{4.4}$$

The goal is to design a control algorithm which solves the following infinite-horizon optimal control problem at each iteration:

$$(\boldsymbol{X}^*, \boldsymbol{U}^*) = \arg\min_{\boldsymbol{X}, \boldsymbol{U}} J_{[0,\infty]}(\boldsymbol{X}, \boldsymbol{U})$$

$$J_{[0,\infty]}(\boldsymbol{X}, \boldsymbol{U}) = \sum_{k=0}^{\infty} h(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{4.5a}$$

subject to:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \forall k \geq 0 \tag{4.5b}$$

$$\boldsymbol{x}_0 = \boldsymbol{x}_S \tag{4.5c}$$

$$\boldsymbol{x}_k \in \mathcal{X}, \quad \boldsymbol{u}_k \in \mathcal{U}, \quad \forall k \geq 0 \tag{4.5d}$$

where:

$$\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_k, ...)$$

$$\boldsymbol{U} = (\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_k, ...) \tag{4.6}$$

are the optimization variables, (4.5b) and (4.5c) represent the system equations and the initial condition, and (4.5d) are the states and inputs constraints.
We assume that the function $h(\boldsymbol{x}, \boldsymbol{u})$ in (4.5a), called *stage cost function*, is continuous and it satisfies:

$$h(\boldsymbol{x}_F, \boldsymbol{0}) = 0, \quad h(\boldsymbol{x}, \boldsymbol{u}) > 0, \quad \forall (\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}^n \times \mathbb{R}^m \setminus \{(\boldsymbol{x}_F, \boldsymbol{0})\} \tag{4.7}$$

meaning that h is positive definite with center in $(\boldsymbol{x}_F, \boldsymbol{0})$.
$\boldsymbol{x}_F$ is the *goal state* towards which the control algorithm should drive the system, i.e. $\lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}_F$.

Moreover, the goal state $\boldsymbol{x}_F$ is assumed to be an equilibrium point of system (4.1):

$$\boldsymbol{x}_F = \boldsymbol{f}(\boldsymbol{x}_F, \boldsymbol{0}) \tag{4.8}$$

We also assume that a local optimal solution to problem (4.5) exists and it is denoted as:

$$\begin{aligned} \boldsymbol{X}^* &= (\boldsymbol{x}_0^*, \boldsymbol{x}_1^*, ..., \boldsymbol{x}_k^*, ...) \\ \boldsymbol{U}^* &= (\boldsymbol{u}_0^*, \boldsymbol{u}_1^*, ..., \boldsymbol{u}_k^*, ...) \end{aligned} \tag{4.9}$$

Note 4.1 By assumption, the stage cost function h in (4.7) is continuous, strictly positive and zero in $\boldsymbol{x}_F$. Thus, an optimal solution to (4.5) will converge to the goal state $\boldsymbol{x}_F$, i.e. $\lim_{k\to\infty} \boldsymbol{x}_k^* = \boldsymbol{x}_F$.

Note 4.2 In practical applications, as we will see in the following, each iteration has a finite time duration T_j:

$$\begin{aligned} \boldsymbol{X}^j &= (\boldsymbol{x}_0^j, \boldsymbol{x}_1^j, ..., \boldsymbol{x}_k^j, ..., \boldsymbol{x}_{T_j}^j) \\ \boldsymbol{U}^j &= (\boldsymbol{u}_0^j, \boldsymbol{u}_1^j, ..., \boldsymbol{u}_k^j, ..., \boldsymbol{u}_{T_j-1}^j) \end{aligned} \tag{4.10}$$

Nonetheless, in literature it is typically adopted, for the sake of simplicity, an infinite time formulation at each iteration, that is what we have used in the previous equations.

In the next sections, we provide the definitions of sampled safe set, iteration cost and terminal cost function. All of these concepts will be used in the following to formulate the LMPC algorithm, granting its stability and recursive feasibility [15].

4.2.1. Sampled safe set

We define the *sampled safe set* SS^j at iteration j as:

$$SS^j = \left\{ \bigcup_{i \in G^j} \bigcup_{k=0}^{\infty} \boldsymbol{x}_k^i \right\} \tag{4.11}$$

where

$$G^j = \left\{ i \in [0, j] : \lim_{k\to\infty} \boldsymbol{x}_k^i = \boldsymbol{x}_F \right\} \tag{4.12}$$

SS^j is, therefore, a set containing all the state trajectories at iteration i, with $i \in G^j$. Specifically, G^j in (4.12) is the set of indices i associated to successful iterations of the algorithm (i.e. iterations for which the related trajectory converged to $\boldsymbol{x}_F$ for $k \to \infty$).

In conclusion, SS^j stores all the states $\boldsymbol{x}_k$, with $k \in [0, \infty]$, composing the state trajectories generated by successful iterations of the LMPC algorithm.

From (4.12) we see that $G^i \subseteq G^j$ for $i \leq j$, which implies that:

$$SS^i \subseteq SS^j, \quad \forall i \leq j \tag{4.13}$$

4.2.2. Iteration cost and terminal cost function

Let's consider the closed-loop state trajectory (4.3a) and input sequence (4.3b) associated to the j-th iteration of the algorithm. Considering a time instant $k \in [0, \infty]$, we define the *cost-to-go* of the trajectory j at time k as:

$$J^j_{[k,\infty]}(\boldsymbol{x}^j_k) = \sum_{t=k}^{\infty} h(\boldsymbol{x}^j_t, \boldsymbol{u}^j_t) \tag{4.14}$$

where h is the stage cost function of (4.5). The cost-to-go can be equivalently denoted also as J^j_k and $J^j(\boldsymbol{x}^j_k)$.

The cost-to-go of the trajectory at time k is therefore the sum of all the stage costs associated to the part of the trajectory on $[k, \infty]$, so starting from the state $\boldsymbol{x}^j_k$ (associated to the time k).

We then define the *iteration cost* as:

$$J^j_{[0,\infty]}(\boldsymbol{x}^j_0) \equiv J^j_0 \equiv J^j(\boldsymbol{x}^j_0) = \sum_{k=0}^{\infty} h(\boldsymbol{x}^j_k, \boldsymbol{u}^j_k) \tag{4.15}$$

The iteration cost is nothing but the cost-to-go of the related trajectory at the initial time $k = 0$.

J^j_0 quantifies the control algorithm performance at each j-th iteration.

Note 4.3 In (4.14) and (4.15), $\boldsymbol{x}^j_k$ and $\boldsymbol{u}^j_k$ are the realized states and inputs of the trajectory at the j-th iteration, as defined in (4.3).

The computation and a graphical representation of the cost-to-go and iteration cost are depicted in Figure 4.1. Specifically, considering for simplicity the finite-time states

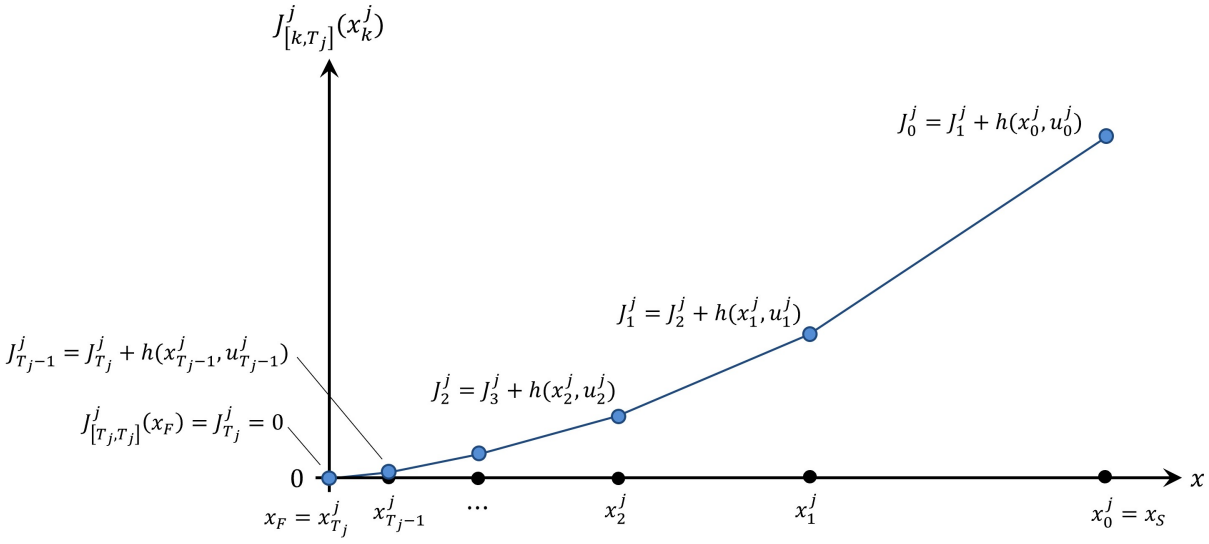


Figure 4.1. Computation and graphical representation of the cost-to-go function $J^j_{[k,T_j]}(x^j_k)$ and the iteration cost J^j_0

trajectory and inputs sequence in (4.10), with $x \in \mathbb{R}$ and $u \in \mathbb{R}$, we can compute each cost-to-go J^j_k in an iterative way by noticing that:

- By definition of cost-to-go (4.14), $J^j_{T_j} = J^j_{[T_j,T_j]}(x^j_{T_j}) = h(x^j_{T_j}, u^j_{T_j}) = h(x_F, 0)$; from (4.7), $h(x_F, 0) = 0 = J^j_{T_j}$;
- From (4.14), $J^j_k = J^j_{k+1} + h(x^j_k, u^j_k)$; for (4.7), being h positive definite with center in $(x_F, 0)$, $J^j_k > J^j_{k+1}$, $\forall k \in [0, T_j - 1]$, meaning that also $J^j_{[k,T_j]}(x^j_k)$ is a (discrete) positive definite function, with center in x_F.

This means that the iteration cost J^j_0 can be computed summing the stage costs "backwards", starting from $x_F = x^j_{T_j}$ (for which $J^j_{T_j} = 0$) till reaching $x_S = x^j_0$ (which is associated J^j_0).

Finally, we define the *terminal cost function* $Q^j(\boldsymbol{x})$, defined over the SS^j, as:

$$Q^j(\boldsymbol{x}) = \begin{cases} \min\limits_{(i,k)\in F^j(\boldsymbol{x})} J^i_{[k,\infty]}(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in SS^j \\ +\infty & \text{if } \boldsymbol{x} \notin SS^j \end{cases} \tag{4.16}$$

where

$$F^j(\boldsymbol{x}) = \left\{(i,k) \in [0,j] \times [0,\infty] : \boldsymbol{x} = \boldsymbol{x}^i_k \text{ for } \boldsymbol{x}^i_k \in SS^j\right\} \tag{4.17}$$

The terminal cost function assigns, to every point in the sampled safe set SS^j, the minimum cost-to-go along the trajectories in SS^j. Specifically, give the state $\boldsymbol{x}$, the set

$F^j(\boldsymbol{x})$ selects all the states belonging to a certain trajectory (from iteration 0 up to j) that are equal to $\boldsymbol{x}$. Then, the function $Q^j(\boldsymbol{x})$ returns the minimum cost-to-go of the trajectories selected by $F^j(\boldsymbol{x})$, starting from $\boldsymbol{x}$.

We can equivalently write that:

$$\forall \boldsymbol{x} \in SS^j, \;\; Q^j(\boldsymbol{x}) = J^{i^*}_{[k^*,\infty]}(\boldsymbol{x}) = \sum_{t=k^*}^{\infty} h(\boldsymbol{x}_t^{i^*}, \boldsymbol{u}_t^{i^*}) \tag{4.18}$$

where (i^*, k^*) are the iteration number and time instant associated to the trajectory giving the minimum cost-to-go, i.e.:

$$(i^*, k^*) = \underset{(i,k)\in F^j(\boldsymbol{x})}{\arg\min} \; J^i_{[k,\infty]}(\boldsymbol{x}) \tag{4.19}$$

Note 4.4 It is worth noticing that, in (4.18), $\boldsymbol{x}^{i^*}_{k^*} = \boldsymbol{x}$.

Note 4.5 If it exists only one trajectory i for which, at time k, $\boldsymbol{x}^i_k = \boldsymbol{x}$ (i.e. $F^j(\boldsymbol{x}) = \{(i,k)\}$), then $Q^j(\boldsymbol{x})$ simply becomes $Q^j(\boldsymbol{x}) = J^i_{[k,\infty]}(\boldsymbol{x}^i_k)$, which is the cost-to-go of the i-th trajectory at time k (so starting from $\boldsymbol{x}^i_k$).

4.3. LMPC optimization problem

In this section, by using all the concepts that have been described above, we provide the formulation of the LMPC control problem, which, being in the class of MPC control methods, takes the form of an optimization problem (OP).

Specifically, the LMPC tries to compute a solution to the infinite-horizon optimal control problem (4.5) by solving, at time k of iteration j, the following finite-horizon constrained optimal control problem:

LMPC optimization problem

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k}{\arg\min} \, J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k)$$

$$J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = J_{[0,N-1]}(\boldsymbol{X}_k, \boldsymbol{U}_k) + Q^{j-1}(\boldsymbol{x}_{N|k}) =$$

$$= \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + Q^{j-1}(\boldsymbol{x}_{N|k}) \tag{4.20a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \;\; t = 0, 1, ..., N-1 \tag{4.20b}$$
$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.20c}$$
$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \;\; \boldsymbol{u}_{t|k} \in \mathcal{U}, \;\; t = 0, 1, ..., N-1 \tag{4.20d}$$
$$\boldsymbol{x}_{N|k} \in SS^{j-1} \tag{4.20e}$$

The problem optimization variables are:

$$\begin{aligned} \boldsymbol{X}_k &\equiv \boldsymbol{X}_{[0,N]|k} = (\boldsymbol{x}_{0|k}, \boldsymbol{x}_{1|k}, ..., \boldsymbol{x}_{N|k}) \\ \boldsymbol{U}_k &\equiv \boldsymbol{U}_{[0,N-1]|k} = (\boldsymbol{u}_{0|k}, \boldsymbol{u}_{1|k}, ..., \boldsymbol{u}_{N-1|k}) \end{aligned} \tag{4.21}$$

representing the predicted state trajectory and the predicted input sequence.

(4.20b) and (4.20c) represent respectively the system equations and the initial condition; (4.20d) are the states and inputs constraints; (4.20e) is a *terminal constraint* that forces the terminal state $\boldsymbol{x}_{N|k}$ into the sampled safe set SS^{j-1} at the previous iteration, as defined in (4.11).

We also see that the function $Q^{j-1}(\boldsymbol{x})$ is indeed used as terminal cost function for the problem (4.20).

The optimal predicted state trajectory of (4.20) and the related optimal predicted input sequence are denoted as:

$$\begin{aligned} \boldsymbol{X}_k^{j^*} &= (\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{x}_{1|k}^{j^*}, ..., \boldsymbol{x}_{N|k}^{j^*}) \\ \boldsymbol{U}_k^{j^*} &= (\boldsymbol{u}_{0|k}^{j^*}, \boldsymbol{u}_{1|k}^{j^*}, ..., \boldsymbol{u}_{N-1|k}^{j^*}) \end{aligned} \tag{4.22}$$

while the optimal value of the cost function is denoted as $J_{LMPC|k}^{j^*}$; this optimal cost can be also expressed specifying the initial state $\boldsymbol{x}_k^j$: $J_{LMPC}^{j^*}(\boldsymbol{x}_k^j)$.

The control algorithm consists in computing, at time k and iteration j, the input sequence $\boldsymbol{U}_k^{j^*}$ and applying to system (4.1) only the 1st element of it:

$$\boldsymbol{u}_k^j \doteq \boldsymbol{u}_{0|k}^{j^*} \tag{4.23}$$

Then, the next state $\boldsymbol{x}_{k+1}^j$ is computed and is used to initialize the new optimization problem at time $k+1$ of iteration j: $\boldsymbol{x}_{0|k+1} \doteq \boldsymbol{x}_{k+1}^j$.

We see that the LMPC features a *receding horizon* control strategy.

4.4. LMPC properties

After having shown the formulation of the LMPC optimization problem, we now provide four fundamental theorems associated to LMPC [15].
Specifically, we will prove, as briefly mentioned in § 4.1, that the LMPC algorithm is guaranteed to be recursively feasible, asymptotically stable, and such that the iteration cost is non-increasing between two successive iterations; moreover, we will also show that, if the LMPC algorithm converges to a steady-state trajectory at iteration $j \to \infty$, then this state trajectory is the solution of the infinite-horizon optimal control problem (4.5).

4.4.1. Recursive feasibility

Assumption 4.1 At iteration $j = 1$ we assume that $SS^{j-1} = SS^0$ is a non-empty set, containing a trajectory $\boldsymbol{X}^0$ that is feasible and convergent to $\boldsymbol{x}_F$.

Typically, this first trajectory is obtained through a basic reference tracking control method (e.g. classic MPC, PID control, etc.).

Theorem 4.1 LMPC recursive feasibility

Consider the system (4.1) controlled by the LMPC control algorithm (4.20) and (4.23). Let SS^j be the sampled safe set at iteration j as defined in (4.11); let Assumption 4.1 hold.

Then, the LMPC is feasible for all $k \geq 0$ and at every iteration $j \geq 1$ [15].

Proof The proof follows from standard MPC arguments. By assumption, SS^0 is non-empty. From (4.13) we have that $SS^0 \subseteq SS^{j-1}$, $\forall j \geq 1$, and consequently SS^{j-1} is a non-empty set. In particular, there exists a feasible trajectory $\boldsymbol{X}^0 \in SS^0 \subseteq SS^{j-1}$. From (4.4) we know that $\boldsymbol{x}_0^j = \boldsymbol{x}_S$, $\forall j \geq 0$. At time $k = 0$ of the j-th iteration the N-steps trajectory:

$$\boldsymbol{X}^0_{[0,N]} = (\boldsymbol{x}^0_0,\ \boldsymbol{x}^0_1,\ ...,\ \boldsymbol{x}^0_N) \in SS^{j-1} \tag{4.24}$$

and the related input sequence:

$$(\boldsymbol{u}^0_0,\ \boldsymbol{u}^0_1,\ ...,\ \boldsymbol{u}^0_{N-1}) \tag{4.25}$$

satisfy input and state constrains (4.20d). Therefore (4.24)-(4.25) is a feasible solution to the LMPC (4.20) and (4.23) at $k = 0$ of the j-th iteration.

Assume that at time k of the j-th iteration the LMPC (4.20) and (4.23) is feasible and let $\boldsymbol{X}_k^{j^*}$ and $\boldsymbol{U}_k^{j^*}$ be the optimal trajectory and input sequence, as defined in (4.22). From (4.20c) and (4.23), the realized state and input at time k of the j-th iteration are given by:

$$\begin{aligned} \boldsymbol{x}_{0|k}^{j^*} &= \boldsymbol{x}_k^j \\ \boldsymbol{u}_k^j &= \boldsymbol{u}_{0|k}^{j^*} \end{aligned} \tag{4.26}$$

The terminal constraint (4.20e) enforces $\boldsymbol{x}_{N|k}^{j^*} \in SS^{j-1}$ and, from (4.18):

$$Q^{j-1}(\boldsymbol{x}_{N|k}^{j^*}) = J_{[k^*,\infty]}^{i^*}(\boldsymbol{x}_{N|k}^{j^*}) = \sum_{t=k^*}^{\infty} h(\boldsymbol{x}_t^{i^*}, \boldsymbol{u}_t^{i^*}) \tag{4.27}$$

Note that $\boldsymbol{x}_{k^*+1}^{i^*} = \boldsymbol{f}\left(\boldsymbol{x}_{k^*}^{i^*}, \boldsymbol{u}_{k^*}^{i^*}\right)$ and, by the definition of $Q^j(\boldsymbol{x})$ and $F^j(\boldsymbol{x})$ in (4.16)-(4.17), $\boldsymbol{x}_{k^*}^{i^*} = \boldsymbol{x}_{N|k}^{j^*}$. Since the state update in (4.1) and (4.20a) are assumed identical, we have that:

$$\boldsymbol{x}_{k+1}^j = \boldsymbol{x}_{1|k}^{j^*} \tag{4.28}$$

At time $k+1$ of the j-th iteration, the input sequence:

$$(\boldsymbol{u}_{1|k}^{j^*},\ \boldsymbol{u}_{2|k}^{j^*},\ ...,\ \boldsymbol{u}_{N-1|k}^{j^*},\ \boldsymbol{u}_{k^*}^{i^*}) \tag{4.29}$$

and the related state trajectory:

$$(\boldsymbol{x}_{1|k}^{j^*},\ \boldsymbol{x}_{2|k}^{j^*},\ ...,\ \boldsymbol{x}_{N-1|k}^{j^*},\ \boldsymbol{x}_{k^*}^{i^*},\ \boldsymbol{x}_{k^*+1}^{i^*}) \tag{4.30}$$

satisfy input and state constrains (4.20b)-(4.20e). Therefore, (4.29)-(4.30) is a feasible solution for the LMPC (4.20) and (4.23) at time $k+1$.

We showed that at the j-th iteration, $\forall j \geq 1$:

- the LMPC is feasible at time $k = 0$;
- if the LMPC is feasible at time k, then the LMPC is feasible at time $k+1$.

Thus, we conclude by induction that the LMPC in (4.20) and (4.23) is feasible $\forall j \geq 1$ and $k \geq 0$ [15]. ■

4.4.2. Asymptotic stability

Theorem 4.2 LMPC asymptotic stability

Consider the system (4.1) controlled by the LMPC control algorithm (4.20) and (4.23). Let SS^j be the sampled safe set at iteration j as defined in (4.11); let Assumption 4.1 hold.

Then, the equilibrium point $\boldsymbol{x}_F$ is asymptotically stable for the closed-loop system (4.1), (4.20) and (4.23) at every iteration $j \geq 1$ [15].

Proof To prove the asymptotic stability of $\boldsymbol{x}_F$, we can show, according to *Lyapunov's direct method*, that the optimal cost $J^{j^*}_{LMPC}(\boldsymbol{x})$ is a Lyapunov function for the equilibrium point $\boldsymbol{x}_F$ of the closed-loop system (4.20) and (4.23).
Specifically, we have to show that:

- $J^{j^*}_{LMPC}(\boldsymbol{x})$ is positive definite with center in $\boldsymbol{x}_F$;
- $J^{j^*}_{LMPC}(\boldsymbol{x}^j_{k+1}) - J^{j^*}_{LMPC}(\boldsymbol{x}^j_k) < 0$, where $\boldsymbol{X}^j = (\boldsymbol{x}^j_0, ..., \boldsymbol{x}^j_k, \boldsymbol{x}^j_{k+1}, ...)$ is the j-th closed-loop trajectory obtained via the LMPC control algorithm (4.20) and (4.23); equivalently, $J^{j^*}_{LMPC}(\boldsymbol{x})$ decreases along the closed-loop trajectory $\boldsymbol{X}^j$.

From (4.7), $J^{j^*}_{LMPC}(\boldsymbol{x}) > 0$, $\forall \boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{x}_F\}$ and $J^{j^*}_{LMPC}(\boldsymbol{x}_F) = 0$, so $J^{j^*}_{LMPC}(\boldsymbol{x})$ is positive definite with center in $\boldsymbol{x}_F$.

Now, we need to show that $J^{j^*}_{LMPC}(\boldsymbol{x})$ is decreasing along the closed-loop trajectory.

From (4.28) we have $\boldsymbol{x}^j_{k+1} = \boldsymbol{x}^{j^*}_{1|k}$, which implies that:

$$J^{j^*}_{LMPC}(\boldsymbol{x}^j_{k+1}) = J^{j^*}_{LMPC}(\boldsymbol{x}^{j^*}_{1|k}) \tag{4.31}$$

Given the optimal predicted input sequence and the related state trajectory in (4.22) and exploiting (4.31) and (4.20c), the optimal cost is given by:

$$J^{j^*}_{LMPC}(\boldsymbol{x}^j_k) = \min_{\boldsymbol{X}_k, \boldsymbol{U}_k} \left[\sum_{t=0}^{N-1} h\left(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}\right) + Q^{j-1}\left(\boldsymbol{x}_{N|k}\right) \right] =$$
$$= h\left(\boldsymbol{x}^{j^*}_{0|k}, \boldsymbol{u}^{j^*}_{0|k}\right) + \sum_{t=1}^{N-1} h\left(\boldsymbol{x}^{j^*}_{t|k}, \boldsymbol{u}^{j^*}_{t|k}\right) + Q^{j-1}\left(\boldsymbol{x}^{j^*}_{N|k}\right) =$$

$$
\begin{aligned}
&= h\left(\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{u}_{0|k}^{j^*}\right) + \sum_{t=1}^{N-1} h\left(\boldsymbol{x}_{t|k}^{j^*}, \boldsymbol{u}_{t|k}^{j^*}\right) + J_{[k^*,\infty]}^{i^*}\left(\boldsymbol{x}_{N|k}^{j^*}\right) = \\
&= h\left(\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{u}_{0|k}^{j^*}\right) + \sum_{t=1}^{N-1} h\left(\boldsymbol{x}_{t|k}^{j^*}, \boldsymbol{u}_{t|k}^{j^*}\right) + \sum_{t=k^*}^{\infty} h\left(\boldsymbol{x}_t^{i^*}, \boldsymbol{u}_t^{i^*}\right) = \\
&= h\left(\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{u}_{0|k}^{j^*}\right) + \sum_{t=1}^{N-1} h\left(\boldsymbol{x}_{t|k}^{j^*}, \boldsymbol{u}_{t|k}^{j^*}\right) + h\left(\boldsymbol{x}_{k^*}^{i^*}, \boldsymbol{u}_{k^*}^{i^*}\right) + Q^{j-1}\left(\boldsymbol{x}_{k^*+1}^{i^*}\right) \geq \\
&\geq h\left(\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{u}_{0|k}^{j^*}\right) + J_{LMPC}^{j^*}(\boldsymbol{x}_{1|k}^{j^*}) = \\
&= h\left(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j\right) + J_{LMPC}^{j^*}(\boldsymbol{x}_{k+1}^j)
\end{aligned} \tag{4.32}
$$

where (i^*, t^*) is defined in (4.19).

Finally, we conclude that the optimal cost is a decreasing Lyapunov function along the closed-loop trajectory:

$$
\begin{aligned}
&J_{LMPC}^{j^*}(\boldsymbol{x}_{k+1}^j) - J_{LMPC}^{j^*}(\boldsymbol{x}_k^j) \leq -h\left(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j\right) < 0 \\
&\forall \boldsymbol{x}_k^j \in \mathbb{R}^n \setminus \{\boldsymbol{x}_F\}, \forall \boldsymbol{u}_k^j \in \mathbb{R}^m \setminus \{\boldsymbol{0}\}
\end{aligned} \tag{4.33}
$$

Therefore, we conclude that $\boldsymbol{x}_F$ is asymptotically stable [15]. ■

4.4.3. Non-increasing iteration cost

Theorem 4.3 LMPC non-increasing iteration cost

Consider the system (4.1) controlled by the LMPC control algorithm (4.20) and (4.23). Let SS^j be the sampled safe set at iteration j as defined in (4.11); let Assumption 4.1 hold.

Then, the iteration cost J_0^j does not increase with the iteration j [15].

Proof First, we find a lower bound on the j-th iteration cost J_0^j, $\forall j > 0$. Consider the realized state and input sequences (4.3) at the j-th iteration, which, at time k, $\forall k \geq 0$, collects the first elements of the optimal state and of the optimal input sequences generated by the LMPC algorithm (4.20) and (4.23). By the definition of the iteration cost in (4.15), we have:

$$
J_{[0,\infty]}^{j-1}(\boldsymbol{x}_S) = \sum_{k=0}^{\infty} h\left(\boldsymbol{x}_k^{j-1}, \boldsymbol{u}_k^{j-1}\right) =
$$

$$
\begin{aligned}
&= \sum_{k=0}^{N-1} h\left(\boldsymbol{x}_k^{j-1}, \boldsymbol{u}_k^{j-1}\right) + \sum_{k=N}^{\infty} h\left(\boldsymbol{x}_k^{j-1}, \boldsymbol{u}_k^{j-1}\right) \geq \\
&\geq \sum_{k=0}^{N-1} h\left(\boldsymbol{x}_k^{j-1}, \boldsymbol{u}_k^{j-1}\right) + Q^{j-1}\left(\boldsymbol{x}_N^{j-1}\right) \geq \\
&\geq \min_{\boldsymbol{X}_{[0,N]}, \boldsymbol{U}_{[0,N]}} \left[\sum_{k=0}^{N-1} h\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) + Q^{j-1}\left(\boldsymbol{x}_N\right)\right] = \\
&= J_{LMPC}^{j^*}(\boldsymbol{x}_0^j)
\end{aligned} \tag{4.34}
$$

Then we notice that, at the j-th iteration, the optimal cost of the LMPC (4.20) and (4.23) at $k = 0$, $J_{LMPC}^{j^*}(\boldsymbol{x}_0^j)$, can be upper bounded along the realized trajectory (4.3a) using (4.33):

$$
\begin{aligned}
J_{LMPC}^{j^*}(\boldsymbol{x}_0^j) &\geq h\left(\boldsymbol{x}_0^j, \boldsymbol{u}_0^j\right) + J_{LMPC}^{j^*}(\boldsymbol{x}_1^j) \geq \\
&\geq h\left(\boldsymbol{x}_0^j, \boldsymbol{u}_0^j\right) + h\left(\boldsymbol{x}_1^j, \boldsymbol{u}_1^j\right) + J_{LMPC}^{j^*}(\boldsymbol{x}_2^j) \geq ... \geq \\
&\geq \lim_{k\to\infty} \left[\sum_{t=0}^{k-1} h\left(\boldsymbol{x}_t^j, \boldsymbol{u}_t^j\right) + J_{LMPC}^{j^*}(\boldsymbol{x}_k^j)\right]
\end{aligned} \tag{4.35}
$$

From Theorem 1.2, $\boldsymbol{x}_F$ is asymptotically stable for the closed loop system (4.20) and (4.23) (i.e. $\lim_{k\to\infty} \boldsymbol{x}_k^j = \boldsymbol{x}_F$), thus by continuity of $h(\boldsymbol{x}, \boldsymbol{u})$:

$$
\lim_{k\to\infty} J_{LMPC}^{j^*}(\boldsymbol{x}_k^j) = J_{LMPC}^{j^*}(\boldsymbol{x}_F) = 0 \tag{4.36}
$$

From the previous equations:

$$
J_{LMPC}^{j^*}(\boldsymbol{x}_0^j) \geq \sum_{k=0}^{\infty} h\left(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j\right) = J_{[0,\infty]}^{j}(\boldsymbol{x}_S) \tag{4.37}
$$

and, finally:

$$
J_{[0,\infty]}^{j-1}(\boldsymbol{x}_S) \geq J_{LMPC}^{j^*}(\boldsymbol{x}_0^j) \geq J_{[0,\infty]}^{j}(\boldsymbol{x}_S) \tag{4.38}
$$

Therefore, we conclude that the iteration cost is non-increasing [15]. ■

4.4.4. Convergence to the solution of the infinite-horizon optimal control problem

Assumption 4.2 The problem (4.5) is strictly convex.

Theorem 4.4 LMPC convergence to the solution of the infinite-horizon optimal control problem

Consider the system (4.1) controlled by the LMPC control algorithm (4.20) and (4.23). Let SS^j be the sampled safe set at iteration j as defined in (4.11); let Assumption 4.1 and Assumption 4.2 hold; assume also that the LMPC algorithm converges to the steady-state input sequence $\boldsymbol{U}^\infty = \lim_{j\to\infty} \boldsymbol{U}^j$ and state trajectory $\boldsymbol{X}^\infty = \lim_{j\to\infty} \boldsymbol{X}^j$ for iteration $j \to \infty$.

Then, $\boldsymbol{U}^\infty$ (and so $\boldsymbol{X}^\infty$) is the solution of the infinite-horizon optimal control problem (4.5) [15].

Proof We refer the reader to [15]. ■

4.5. LMPC algorithm

Finally, we can write the proper LMPC algorithm as follows:

Algorithm 4.1 Learning Model Predictive Control

Inputs: $\boldsymbol{x}_S$; N_{iter} (number of LMPC iterations); N (LMPC prediction horizon)
Outputs: $\boldsymbol{X}^0, \boldsymbol{X}^1, ..., \boldsymbol{X}^{N_{iter}}$; $\boldsymbol{U}^0, \boldsymbol{U}^1, ..., \boldsymbol{U}^{N_{iter}}$

(1) Generate the first feasible trajectory $\boldsymbol{X}^0 = (\boldsymbol{x}_0^0, \boldsymbol{x}_1^0, ..., \boldsymbol{x}_{T_0}^0)$ using a basic reference tracking control method, with initial state $\boldsymbol{x}_0^0 = \boldsymbol{x}_S$

(2) Construct the initial sampled safe set $SS^0 = \{\boldsymbol{X}^0\}$

(3) **For** $j = 1, 2, ..., N_{iter}$:

  (2.1) Set the initial state $\boldsymbol{x}_0^j = \boldsymbol{x}_S$ and store it in $\boldsymbol{X}^j$

  (2.2) **For** $k = 0, 1, ...$:

(3.1) • **If** the exit condition on $\boldsymbol{x}_k^j$ is satisfied, **break** the cycle (at $k = T_j$);

• otherwise, continue

(3.2) Initialize the LMPC optimization problem (4.20) with $\boldsymbol{x}_k^j$

(3.3) Solve the LMPC optimization problem, obtaining the optimal predicted state trajectory $\boldsymbol{X}_k^{j^*} = (\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{x}_{1|k}^{j^*}, ..., \boldsymbol{x}_{N|k}^{j^*})$ and optimal predicted input sequence $\boldsymbol{U}_k^{j^*} = (\boldsymbol{u}_{0|k}^{j^*}, \boldsymbol{u}_{1|k}^{j^*}, ..., \boldsymbol{u}_{N-1|k}^{j^*})$

(3.4) Apply the input $\boldsymbol{u}_k^j \doteq \boldsymbol{u}_{0|k}^{j^*}$ to the system (4.1), as in (4.23), obtaining the next state $\boldsymbol{x}_{k+1}^j$

(3.5) Store $\boldsymbol{x}_{k+1}^j$ and $\boldsymbol{u}_k^j$ in $\boldsymbol{X}^j$ and $\boldsymbol{U}^j$

(2.3) Augment the sampled safe set with the new trajectory $\boldsymbol{X}^j = (\boldsymbol{x}_0^j, \boldsymbol{x}_1^j, ..., \boldsymbol{x}_{T_j}^j)$:

$$SS^j = SS^{j-1} \cup \{\boldsymbol{X}^j\}$$

(4) **Return** all the generated trajectories and input sequences:

$$\boldsymbol{X}^0, \boldsymbol{X}^1, ..., \boldsymbol{X}^{N_{iter}}$$
$$\boldsymbol{U}^0, \boldsymbol{U}^1, ..., \boldsymbol{U}^{N_{iter}}$$

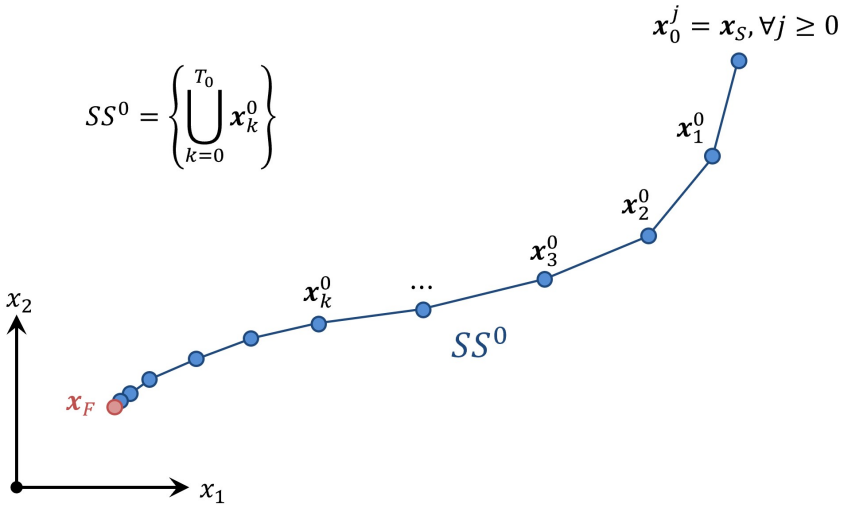
Note 4.6 As for the NMPC of Chapter 3, the exit condition at point (3.1) of the algorithm can be chosen to determine whether the state $\boldsymbol{x}_k^j$ is sufficiently close to the goal state $\boldsymbol{x}_F$. Therefore, it can be expressed as follows:

$$|\boldsymbol{x}_k^j - \boldsymbol{x}_F| \overset{?}{\leq} \delta \tag{4.39}$$

where δ is set by the user.

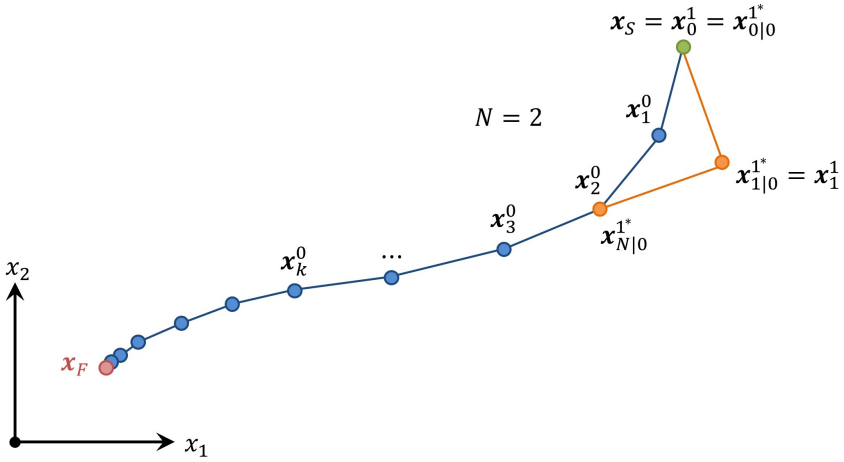
In the following, it is reported a sequence of figures (Figures 4.2a-e) that depict a generic execution of the LMPC algorithm with $N = 2$. Specifically:

- In Figure 4.2a, the first feasible trajectory $\boldsymbol{X}^0$ has been generated and, from it, it has been constructed the initial sampled safe set SS^0, as in Assumption 4.1.



(a) First feasible trajectory $\boldsymbol{X}^0$ and initial sampled safe set SS^0

- In Figure 4.2b, the iteration $j = 1$ of the LMPC algorithm is initiated.
  The first state of the generated trajectory is $\boldsymbol{x}_0^1 = \boldsymbol{x}_S$, as in (4.4), and it is used to initialize the LMPC optimization problem at time $k = 0$.
  The optimization problem is solved, obtaining the optimal predicted state trajectory $\boldsymbol{X}_0^{1^*} = (\boldsymbol{x}_{0|0}^{1^*}, \boldsymbol{x}_{1|0}^{1^*}, \boldsymbol{x}_{N|0}^{1^*}) = (\boldsymbol{x}_0^1, \boldsymbol{x}_{1|0}^{1^*}, \boldsymbol{x}_{N|0}^{1^*})$, which is composed by 3 states, since $N = 2$.
  It is worth noticing that the terminal state $\boldsymbol{x}_{N|0}^{1^*}$ coincides with one of the states in SS^0, satisfying the terminal constraint (4.20e).
  For the receding horizon control law in (4.23), the next state of the generated trajectory will be $\boldsymbol{x}_1^1 = \boldsymbol{x}_{1|0}^{1^*}$.



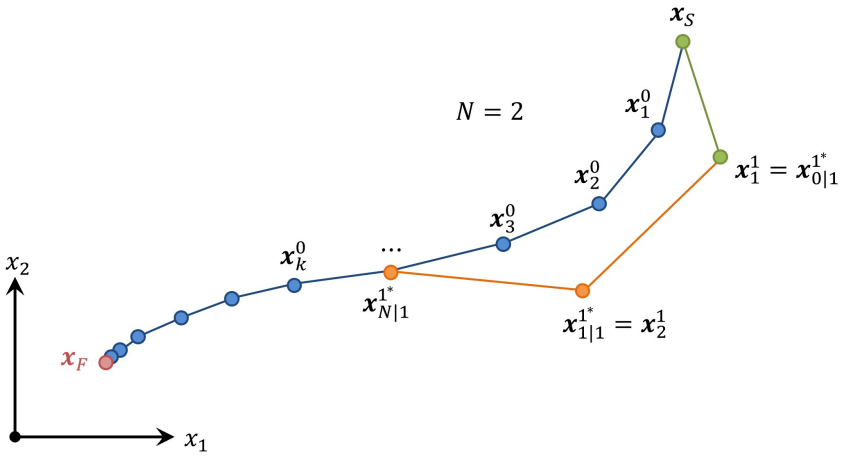
(b) Iteration 1, $k = 0$: optimal predicted state trajectory (in orange), generated state trajectory (in green)

- In Figure 4.2c, the state $\boldsymbol{x}_1^1$ of the generated trajectory is used to initialize the LMPC

optimization problem at time $k = 1$.

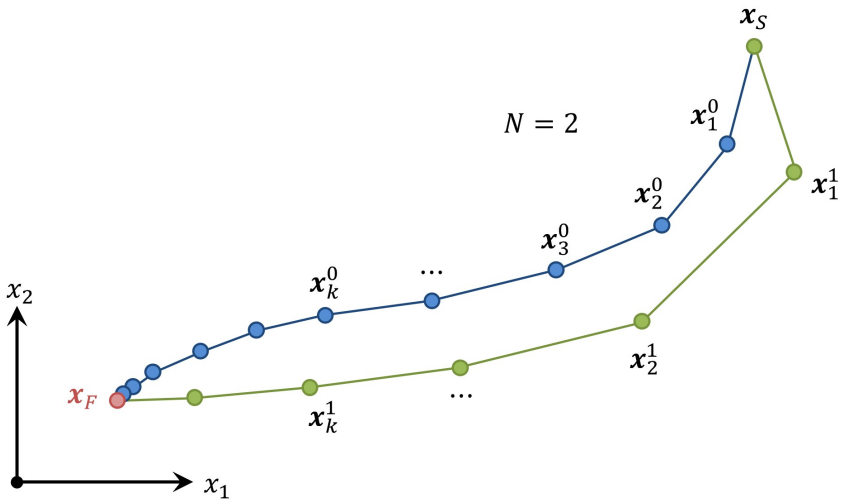
The optimization problem is solved, obtaining the optimal predicted state trajectory $\boldsymbol{X}_1^{1^*} = (\boldsymbol{x}_1^1, \boldsymbol{x}_{1|1}^{1^*}, \boldsymbol{x}_{N|1}^{1^*})$.

The next state of the generated trajectory will be $\boldsymbol{x}_2^1 = \boldsymbol{x}_{1|1}^{1^*}$.



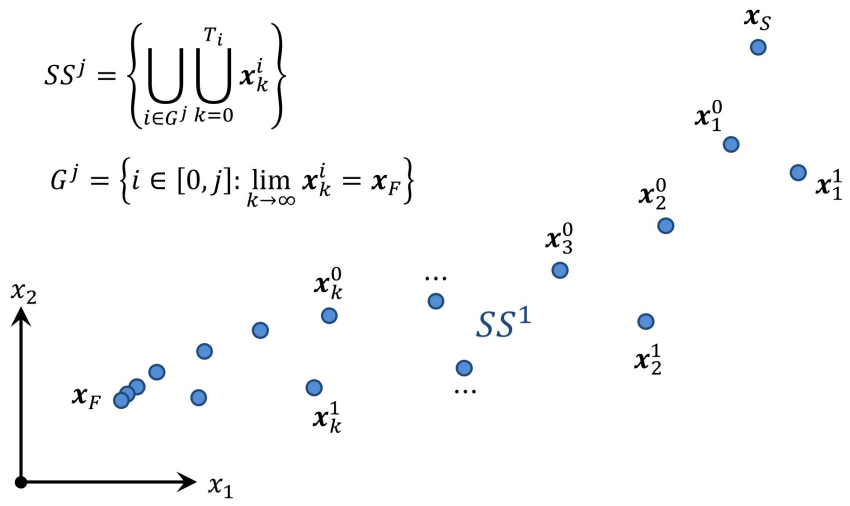
(c) Iteration 1, $k = 1$: optimal predicted state trajectory (in orange), generated state trajectory (in green)

- In Figure 4.2d, at time $k = T_1$ of the iteration $j = 1$, the state $\boldsymbol{x}_{T_1}^1$ of the generated trajectory satisfies the exit condition of the algorithm (i.e. it is sufficiently close to the goal state $\boldsymbol{x}_F$, as reported in Note 4.6); therefore, the current iteration is ended, returning the generated trajectory $\boldsymbol{X}^1 = (\boldsymbol{x}_0^1, \boldsymbol{x}_1^1, ..., \boldsymbol{x}_{T_1}^1)$.



(d) Iteration 1, $k = T_1$: generated state trajectory (in green); end of the iteration

- In Figure 4.2e, before staring the next iteration, the sampled safe set is augmented by including in it the states of the new trajectory $\boldsymbol{X}^1$, obtaining $SS^1 = SS^0 \cup \{\boldsymbol{X}^1\}$.



(e) Augmented sampled safe set SS^1

Figure 4.2. LMPC algorithm representation, with $N = 2$

- The algorithm continues initiating the iteration $j = 2$ and repeating the above operations, until $j > N_{iter}$.

4.6. Sampled safe set and terminal cost relaxation

In this section, we provide two possible ways to relax the terminal constraint and terminal cost function of (4.20), so to make the LMPC control problem easier to solve; in this way, we can reduce the computation time needed to solve the optimization problem, making it more suitable for real-time applications.

The first approach consists in:

- relaxing the discrete sampled safe set SS^j to its convex hull, obtaining a continuous convex set CS^j;
- approximating the discrete terminal cost function $Q^j(\boldsymbol{x})$ with a continuous convex function $P^j(\boldsymbol{x})$, defined on the convex safe set CS^j.

These relaxations are computed exploiting convex combinations and barycentric approximations.

4.6.1. Convex safe set

From its definition (4.11), the sampled safe set SS^j is a collection of discrete states, making it a discrete set. Therefore, the LMPC optimization problem (4.20) ends up being a Mixed-Integer Non-Linear Program (MINLP), which is hard to solve and may not be suitable for real-time applications (since there is no guarantee that it will be solved in real-time).

Therefore, we need to find a way to *relax* the terminal constraint (4.20e), that is transforming the SS^j from a discrete to a continuous set, so to make (4.20) a Non-Linear Program (NLP), which is easier to solve.

The most convenient approach is to relax the sampled safe set to its *convex hull* [16] [3]. The convex hull of SS^j is called *convex safe set* CS^j (Figure 4.3).

The convex hull of a discrete set of points can be obtained by computing the convex combination of all its elements:

$$\begin{aligned} CS^j = \text{conv}(SS^j) = \text{conv}\left(\left\{\bigcup_{i\in G^j}\bigcup_{k=0}^{\infty} \boldsymbol{x}_k^i\right\}\right) = \\ = \left\{\sum_{i=1}^{|SS^j|} \lambda_i \boldsymbol{x}_i : \quad \lambda_i \geq 0, \quad \sum_{i=1}^{|SS^j|} \lambda_i = 1, \quad \boldsymbol{x} \in SS^j\right\} \end{aligned} \tag{4.40}$$

where $|\cdot|$ denotes the set cardinality operator.

More concisely:

$$CS^j = \text{conv}(SS^j) = (\boldsymbol{X}^0, ..., \boldsymbol{X}^j)\boldsymbol{\lambda}^T \tag{4.41}$$

where

$$\boldsymbol{\lambda} = (\lambda_0^0, \lambda_1^0, ..., \lambda_{T_0}^0, ..., \lambda_0^j, \lambda_1^j, ..., \lambda_{T_j}^j), \quad \boldsymbol{\lambda} \geq 0, \quad ||\boldsymbol{\lambda}||_1 = 1 \tag{4.42}$$

Equations (4.40), (4.41) and (4.42) represent the *barycentric approximation* of SS^j: any state in the convex safe set can be written as a convex combination of the points in the sampled safe set; each component of $\boldsymbol{\lambda}$ is a positive weighting scalar for each element of SS^j [8].

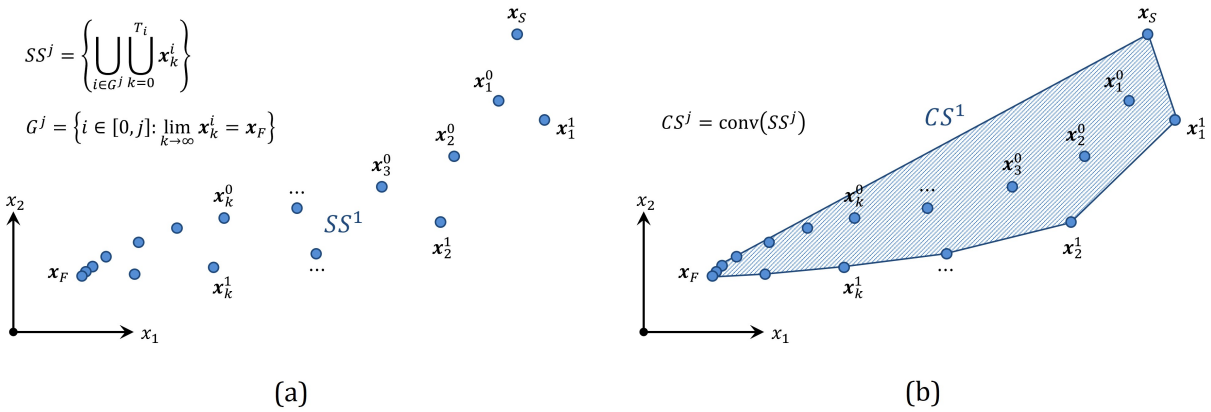


Figure 4.3. Sampled safe set SS^j (a) and convex safe set CS^j (b), with $j = 1$

4.6.2. Terminal cost barycentric function

Relaxing the sampled safe set to its convex hull requires also to adapt the terminal cost function $Q^j(\boldsymbol{x})$ accordingly. Therefore, we introduce a barycentric approximation also for Q^j:

$$P^j(\boldsymbol{x}) = \begin{cases} p^{j^*}(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in CS^j \\ +\infty & \text{if } \boldsymbol{x} \notin CS^j \end{cases} \tag{4.43}$$

where

$$p^{j^*}(\boldsymbol{x}) = \min_{\lambda_k \geq 0, k \in [0,\infty]} \sum_{i=0}^{j} \sum_{k=0}^{\infty} \lambda_k^i J_{[k,\infty]}^j(\boldsymbol{x}_k^i) \tag{4.44a}$$

subject to:

$$\sum_{i=0}^{j} \sum_{k=0}^{\infty} \lambda_k^i = 1 \tag{4.44b}$$

$$\sum_{i=0}^{j} \sum_{k=0}^{\infty} \lambda_k^i \boldsymbol{x}_k^i = \boldsymbol{x} \tag{4.44c}$$

$P^j(\boldsymbol{x})$ is called *terminal cost barycentric function*; as defined in (4.43) and (4.44), it represents a convex approximation of $Q^j(\boldsymbol{x})$, which assigns to every point in CS^j the minimum cost-to-go along all the possible trajectories in CS^j.

More concisely, assuming (as in Note 4.5) that in SS^j there are no states in common to multiple trajectories (except for $\boldsymbol{x}_S$):

$$P^j(\boldsymbol{x}) = \text{conv}(Q^j(\boldsymbol{x})) = \min_{\boldsymbol{\lambda} \geq 0} \left(J_{[0,T_0]}^0(\boldsymbol{x}_0^0), J_{[1,T_0]}^0(\boldsymbol{x}_1^0), ..., J_{[0,T_j]}^j(\boldsymbol{x}_0^j), ... \right) \boldsymbol{\lambda}^T \tag{4.45}$$

where

$$\boldsymbol{\lambda} = (\lambda_0^0, \lambda_1^0, ..., \lambda_{T_0}^0, ..., \lambda_0^j, \lambda_1^j, ..., \lambda_{T_j}^j), \quad \boldsymbol{\lambda} \geq 0, \quad ||\boldsymbol{\lambda}||_1 = 1, \quad (\boldsymbol{X}^0, ..., \boldsymbol{X}^j)\boldsymbol{\lambda}^T = \boldsymbol{x} \tag{4.46}$$

4.6.3. Relaxed LMPC optimization problem (first version)

From the relaxations shown in § 4.6.1 and § 4.6.2, we can formulate a first version of the relaxed LMPC optimization problem:

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}, \boldsymbol{\lambda}^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{\lambda}}{\arg\min}\, J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{\lambda})$$

$$J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{\lambda}) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + P^{j-1}(\boldsymbol{x}_{N|k}) \tag{4.47a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \;\; t = 0, 1, ..., N-1 \tag{4.47b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.47c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \;\; \boldsymbol{u}_{t|k} \in \mathcal{U}, \;\; t = 0, 1, ..., N-1 \tag{4.47d}$$

$$\boldsymbol{x}_{N|k} \in CS^{j-1} \tag{4.47e}$$

$$\boldsymbol{\lambda} \geq 0, \;\; ||\boldsymbol{\lambda}||_1 = 1 \tag{4.47f}$$

Equivalently:

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}, \boldsymbol{\lambda}^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{\lambda}}{\arg\min}\, J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{\lambda})$$

$$J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, \boldsymbol{\lambda}) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) +$$

$$+ \left(J_{[0,T_0]}^0(\boldsymbol{x}_0^0), J_{[1,T_0]}^0(\boldsymbol{x}_1^0), ..., J_{[0,T_{j-1}]}^{j-1}(\boldsymbol{x}_0^{j-1}), ... \right) \boldsymbol{\lambda}^T \tag{4.48a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \;\; t = 0, 1, ..., N-1 \tag{4.48b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.48c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \;\; \boldsymbol{u}_{t|k} \in \mathcal{U}, \;\; t = 0, 1, ..., N-1 \tag{4.48d}$$

$$\boldsymbol{x}_{N|k} = (\boldsymbol{X}^0, ..., \boldsymbol{X}^{j-1})\boldsymbol{\lambda}^T \tag{4.48e}$$

$$\boldsymbol{\lambda} \geq 0, \;\; ||\boldsymbol{\lambda}||_1 = 1 \tag{4.48f}$$

With respect to (4.20), the relaxations have introduced the following changes:

- The coefficients $\boldsymbol{\lambda}$ of the convex combination become additional optimization variables;
- Constraint (4.20e) becomes an equality constraint (4.48e) that, according to (4.41), forces the terminal state into the convex safe set CS^{j-1};
- The terminal cost function becomes the argument of the optimization problem in (4.45).

We now describe the second relaxation approach, which in part uses the results of the first approach, yielding in the end to a simpler optimization problem, involving a lower number of optimization variables.

This second approach consists in:

- expressing the terminal constraint through the set of linear inequalities defining the convex safe set CS^j;
- approximating the discrete terminal cost function $Q^j(\boldsymbol{x})$ with a convex piecewise-linear interpolating function $\tilde{P}^j(\boldsymbol{x})$, over the convex safe set CS^j.

4.6.4. Convex safe set linear constraints

The convex safe set CS^j, being the convex hull of discrete states in $SS^j \subset \mathbb{R}^n$, is a *convex polytope*. Specifically, a polytope is the generalization in n dimensions of the three-dimensional polyhedron.

CS^j can be defined through the convex combination of the discrete states as in (4.40); such a definition is called vertex representation (or V-representation).

For the sake of LMPC relaxation, however, CS^j can be also defined in alternative way, that proves to be more useful. It is called half-space representation (or H-representation). In H-representation, the convex polytope is seen as the intersection of a finite number of subspaces. Therefore, it can be described via a set of linear inequalities:

$$\begin{array}{cccccc} a_{11}x_1 + & a_{12}x_2 + \cdots + & a_{1n}x_n \leq b_1 \\ a_{21}x_1 + & a_{22}x_2 + \cdots + & a_{2n}x_n \leq b_2 \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \tag{4.49}$$

which can be written in a compact matrix form:

$$\boldsymbol{A}_{CS}\boldsymbol{x} \leq \boldsymbol{b}_{CS}, \quad \boldsymbol{A}_{CS} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \quad \boldsymbol{b}_{CS} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{4.50}$$

(4.50) is a linear inequality constraint, which can be easily inserted in the relaxed LMPC optimization problem as terminal constraint.

4.6.5. Convex piecewise-linear fitting of the terminal cost function

At iteration j, the sampled safe set SS^j contains the following states:

$$SS^j = \{\boldsymbol{X}^0, ..., \boldsymbol{X}^j\} = \{\boldsymbol{x}_0^0,\ \boldsymbol{x}_1^0,\ ...,\ \boldsymbol{x}_{T_j}^j\} \tag{4.51}$$

The terminal cost function $Q^i(\boldsymbol{x})$, if (as in Note 4.5) there are no states in common to multiple trajectories (except for $\boldsymbol{x}_S$), when applied to every state in SS^j gives:

$$\begin{aligned} Q^j(SS^j) &= \{Q^j(\boldsymbol{x}_0^0),\ Q^j(\boldsymbol{x}_1^0),\ ...,\ Q^j(\boldsymbol{x}_{T_j}^j)\} = \\ &= \{J_{[0,T_0]}^0(\boldsymbol{x}_0^0), J_{[1,T_0]}^0(\boldsymbol{x}_1^0), ..., J_{[T_j,T_j]}^j(\boldsymbol{x}_{T_j}^j)\} \equiv \\ &\equiv \{J_0^0, J_1^0, ..., J_{T_j}^j\} \equiv \\ &\equiv \{Q_0^0,\ Q_0^1,\ ...,\ Q_{T_j}^j\} \end{aligned} \tag{4.52}$$

We then consider the problem of fitting the obtained data:

$$(\boldsymbol{x}_0^0, Q_0^0), ..., (\boldsymbol{x}_{T_j}^j, Q_{T_j}^j) \equiv (\boldsymbol{x}_1, Q_1), ..., (\boldsymbol{x}_m, Q_m) \in \mathbb{R}^n \times \mathbb{R}, \ \ m = |SS^j| \tag{4.53}$$

with a *convex piecewise-linear function* $f : \mathbb{R}^n \to \mathbb{R}$ from some set $\mathcal{F}$ of candidate functions. With a least-squares fitting criterion, we obtain the problem:

$$\min_{f \in \mathcal{F}} \sqrt{\frac{J(f)}{m}}, \quad J(f) = \sum_{i=1}^{m} \left(f(\boldsymbol{x}_i) - Q_i\right)^2 \tag{4.54}$$

We call $\sqrt{\frac{J(f)}{m}}$ the RMSE index (root-mean-square error) of the function f fitting the data. The convex piecewise-linear fitting problem (4.54) is to find the function f, from

the given family $\mathcal{F}$ of convex piecewise-linear functions, that gives the best (smallest) RMSE value with the given data.

Our main interest is in the case when n (i.e. the state dimension) is relatively small, while m (i.e. the number of data points) can be relatively large (e.g. 10^4 or more). The methods that will be described in the following, however, work for any values of n and m.

Several special cases of the convex piecewise-linear fitting problems (4.54) can be solved exactly.

When $\mathcal{F}$ is the set of *affine functions*, i.e. $f(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x}+b$, the problem (4.54) reduces to an ordinary linear least-squares problem in the function parameters $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ and can be analytically solved. This approach is called *parametric* convex piecewise-linear fitting.

A less trivial case is, by converse, the *non-parametric* convex piecewise-linear fitting, in which $\mathcal{F}$ is the set of all piecewise-linear functions from $\mathbb{R}^n$ to $\mathbb{R}$, with no other constraint on the form of f.

Of course, not all data can be fit well (i.e. with small RMSE) with a convex piecewise-linear function. For example, if the data are samples from a function that has a strong concave curvature, then no convex function can fit it well.

Max-affine functions

For our purposes, we will consider a parametric fitting problem, in which the candidate functions are parametrized by a finite-dimensional vector of coefficients $\boldsymbol{\alpha}$.
We choose as $\mathcal{F}$ the set of functions on $\mathbb{R}^n$ with the form:

$$f(\boldsymbol{x}) = \max\left(\boldsymbol{a}_1^T\boldsymbol{x} + b_1, ..., \boldsymbol{a}_k^T\boldsymbol{x} + b_k\right) \tag{4.55}$$

We refer to a function of this form as a *max-affine* function with k terms. Such functions are parameterized by the coefficients vector:

$$\boldsymbol{\alpha} = (\boldsymbol{a}_1, ..., \boldsymbol{a}_k, b_1, ..., b_k) \in \mathbb{R}^{k(n+1)} \tag{4.56}$$

Max-affine functions can be visualized as an intersection of k planes in the $\mathbb{R}^n \times \mathbb{R}$ space, of which we take the envelope (which corresponds to the max operation).

Indeed, any convex piecewise-linear function on $\mathbb{R}^n$ can be expressed as a max-affine function, for some k, so this form is in a sense universal.
Our interest, however, is in the case when the number of terms k is relatively small, In this case the max-affine representation (4.55) is *compact*, in the sense that the number of parameters needed to describe f is much smaller than the number of parameters in the original data set (i.e. $m(n+1)$). The methods that will be described in the following, however, do not require k to be small.

With max-affine functions, the fitting problem (4.54) reduces to the nonlinear least-squares problem:

$$\min_{\boldsymbol{\alpha}} \sqrt{\frac{J(\boldsymbol{\alpha})}{m}}, \quad J(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \left(\max_{j=1,\dots,k} \left(\boldsymbol{a}_j^T \boldsymbol{x}_i + b_j \right) - Q_i \right)^2 \tag{4.57}$$

Fitting procedure

In this section, we present an algorithm that solves approximately the k-term max-affine fitting problem (4.57) [11].

The heuristic idea behind this algorithm is the following:

- create k partitions of the data points $\{(\boldsymbol{x}_1, Q_1), ..., (\boldsymbol{x}_m, Q_m)\}$;
- on each partition $j = 1, ..., k$, solve analytically the linear least-square fitting problem associated to the j-th affine functions composing f, obtaining a first estimate of the coefficients $\boldsymbol{a}_j$ and b_j;
- update each partition on the base of the current coefficients value;
- iterate the algorithm until the coefficients converge to a certain value.

The algorithm, therefore, alternates between partitioning the data and carrying out least-squares fits to update the coefficients.

Let $P_j^{(l)}$ for $j = 1, ..., k$, be a partition of the data indices $\{1, ..., m\}$ at the l-th iteration of the algorithm:

$$\begin{aligned} & P_j^{(l)} \subseteq \{1, ..., m\} \\ & \bigcup_j P_j^{(l)} = \{1, ..., m\}, \quad P_i^{(l)} \cap P_j^{(l)} = \emptyset \quad \text{for } i \neq j \end{aligned} \tag{4.58}$$

The generation of the initial partitions $P_j^{(0)}$ will be analyzed in the next section.

Denoting with $\boldsymbol{a}_j^{(l)}$ and $b_j^{(l)}$ the values of the parameters at the l-th iteration of the algorithm, we generate the next values, $\boldsymbol{a}_j^{(l+1)}$ and $b_j^{(l+1)}$, from the current partition $P_j^{(l)}$, as explained in the following paragraphs.

For each $j = 1, ..., k$, we carry out a least-squares fit of $\boldsymbol{a}_j^T \boldsymbol{x}_i + b_j$ to Q_i, using only the data points with $i \in P_j^{(l)}$. In other words, we take $\boldsymbol{a}_j^{(l+1)}$ and $b_j^{(l+1)}$ as values of $\boldsymbol{a}$ and b that minimize

$$\sum_{i \in P_j^{(l)}} \left(\boldsymbol{a}^T \boldsymbol{x}_i + b - Q_i \right)^2 \tag{4.59}$$

which is indeed the linear least-square problem associated to the j-th affine function composing f, restricted to only the data in the j-th partition.

Such problem can be solved analytically, and the pair $(\boldsymbol{a}, b)$ that minimizes (4.59) is given by:

$$\begin{pmatrix} \boldsymbol{a}_j^{(l+1)} \\ b_j^{(l+1)} \end{pmatrix} = \begin{pmatrix} \sum \boldsymbol{x}_i \boldsymbol{x}_i^T & \sum \boldsymbol{x}_i \\ \sum \boldsymbol{x}_i^T & \left| P_j^{(l)} \right| \end{pmatrix}^{-1} \begin{pmatrix} \sum Q_i \boldsymbol{x}_i \\ \sum Q_i \end{pmatrix} \tag{4.60}$$

where the sums are over $i \in P_j^{(l)}$.

Using the new values of the coefficients, we update the partition to obtain $P_j^{(l+1)}$, by assigning i to $P_j^{(l+1)}$ if:

$$f^{(l+1)}(\boldsymbol{x}_i) = \max_{s=1,\dots,k} \left(\boldsymbol{a}_s^{(l+1)T} \boldsymbol{x}_i + b_s^{(l+1)} \right) = \boldsymbol{a}_j^{(l+1)T} \boldsymbol{x}_i + b_j^{(l+1)} \tag{4.61}$$

This means that $P_j^{(l+1)}$ is the set of indices i for which the affine function $\boldsymbol{a}_j^{(l+1)T} \boldsymbol{x}_i + b_j^{(l+1)}$ is the maximum for the data point $\boldsymbol{x}_i$.

During this step of the algorithm, one or more of the sets $P_j^{(l+1)}$ may become empty. The simplest approach is to drop these empty sets from the partitions, and continue with a smaller value of k.

This algorithm is iterated until one of the following conditions is met:

- Convergence has been reached, which equivalently occurs when:
  - the coefficients vector $\boldsymbol{\alpha}^{(l)}$ has converged to a steady-state value: $\boldsymbol{\alpha}^{(l+1)} = \boldsymbol{\alpha}^{(l)}$, $\forall l \geq l'$;

- the partitions $P_j^{(l)}$ do not change anymore after a certain iteration: $P_j^{(l+1)} = P_j^{(l)}$, $\forall l \geq l',\ \forall j \in \{1, ..., k\}$.

- The maximum number of iterations has been reached: $l > N_{iter}$.

Generation of the initial partitions

The choice of the initial partitions $P_j^{(0)}$ is a crucial step, since it directly affects the quality of the fitting given by the max-affine interpolating function generated by the algorithm. A bad selection of the initial partitions may lead the algorithm to not converge, or to converge to a piecewise-linear approximation that poorly fits the data (even when the data shows a good convex curvature).

We provide in the following a general approach to generate initial partitions that ensure, in most cases, a good interpolation of the data points [11].

Each partition $P_j^{(0)}$, with $j = 1, ..., k$, is generated starting from k randomly chosen points $\boldsymbol{p}_1, ..., \boldsymbol{p}_k \in \mathbb{R}^n$, which are called *seed points*.
Then, $P_j^{(0)}$ is defined as the set of indices i associated to the data points $\boldsymbol{x}_i$ that are closest to $\boldsymbol{p}_j$:

$$P_j^0 = \{i : |\boldsymbol{x}_i - \boldsymbol{p}_j| < |\boldsymbol{x}_i - \boldsymbol{p}_s|, \forall s \neq j\}, \quad j = 1, ..., k \tag{4.62}$$

Specifically, the indices in the partitions $P_j^{(0)}$ define the *Voronoi sets* of data points $\boldsymbol{x}_i$ associated to the seed points $\boldsymbol{p}_j$.

With this method, we generate initial partitions that tessellate the whole subset of $\mathbb{R}^n$ containing the data points $\{\boldsymbol{x}_i\}_{i=1}^m$. Such partitions are very well suited for max-affine interpolation, since max-affine functions are the envelope of k intersecting planes in $\mathbb{R}^n$ and the fitting procedure locally interpolates the data in each partition with one of the planes composing the function.

The Voronoi sets associated to the initial partitions, which tessellate the set of data points, are depicted in Figure 4.6, showing the case of 20 randomly generated partitions.

The seed points $\boldsymbol{p}_j$ should be randomly generated according to some probability distribution that matches the shape of the data points $\boldsymbol{x}_i$. A possible choice is to generate them from a normal distribution with the sample mean $\boldsymbol{\mu}_x$ and the sample covariance

$\boldsymbol{\Sigma}_x$ of the data points:

$$\boldsymbol{\mu}_x = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i, \quad \boldsymbol{\Sigma}_x = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^T \tag{4.63}$$

Note 4.7 Since the seed points are generated from a normal distribution, it may happen that some of them are chosen outside the set of data points and enough far from it such that the related partition is empty. As done in the fitting algorithm, the simplest approach is to drop these empty partitions and continue with a smaller value of k.

This method for generating initial partitions can be reformulated to more specific procedure that applies only to 1D data (i.e. $n = 1$) and provides in most cases better results.

Instead of generating randomly the initial partitions $P_j^{(0)}$ from seed points, the partitions are chosen as equally spaced intervals on the *ordered* set (i.e. from lowest to highest value) of data points $\{x_1, ..., x_m\} \subset \mathbb{R}$:

$$\begin{aligned} P_j^{(0)} &= \left\{1 + (j-1)\left\lfloor \frac{m}{k} \right\rfloor, ..., j\left\lfloor \frac{m}{k} \right\rfloor\right\}, \quad \text{for } j = 1, ..., k-1 \\ P_k^{(0)} &= \left\{1 + k\left\lfloor \frac{m}{k} \right\rfloor, ..., m\right\} \end{aligned} \tag{4.64}$$

With this approach, we generate, as well as in the general method, initial partitions that cover the whole subset of $\mathbb{R}$ containing the data points $\{x_i\}_{i=1}^m$. This method applies well only to 1D data for the trivial reason that $\mathbb{R}$ is much easier to subdivide in bounded and connected subsets rather than $\mathbb{R}^n$, for which instead generating the partitions randomly in more convenient.

Fitting algorithm

The full algorithm for convex piecewise-linear fitting can be written as follows:

Algorithm 4.2 Convex piecewise-linear fitting

Inputs: $\{(\boldsymbol{x}_1, Q_1), ..., (\boldsymbol{x}_m, Q_m)\}$; k; N_{iter} (max n. of iterations)
Outputs: $\boldsymbol{\alpha}$

(1) Compute the sample mean $\boldsymbol{\mu}_x$ and sample covariance $\boldsymbol{\Sigma}_x$ of the data points

$\{\boldsymbol{x}_i\}_{i=1}^m$ as in (4.63)

(2) Generate the seed points $\boldsymbol{p}_j$, $j = 1, ..., k$, from the distribution $\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$

(3) Compute the initial partitions $P_j^{(0)}$, $j = 1, ..., k$, as in (4.62)

(4) Remove empty partitions from $P_j^{(0)}$, $j = 1, ..., k$, and update the value of k

(5) **For** $l = 0, ..., N_{iter} - 1$:

(2.1) Compute $\boldsymbol{a}_j^{(l+1)}$ and $b_j^{(l+1)}$, $j = 1, ..., k$, as in (4.60)

(2.2) Update the partitions to $P_j^{(l+1)}$, $j = 1, ..., k$, as in (4.61)

(2.3) Remove empty partitions from $P_j^{(l+1)}$, $j = 1, ..., k$, and update the value of k

(2.4)
- **If** $P_j^{(l+1)} = P_j^{(l)}$, **return**:

$$\boldsymbol{\alpha} = \left(\boldsymbol{a}_1^{(l)}, ..., \boldsymbol{a}_k^{(l)}, b_1^{(l)}, ..., b_k^{(l)}\right)$$

- otherwise, continue

(6) **Return**:

$$\boldsymbol{\alpha} = \left(\boldsymbol{a}_1^{(N_{iter})}, ..., \boldsymbol{a}_k^{(N_{iter})}, b_1^{(N_{iter})}, ..., b_k^{(N_{iter})}\right)$$

Examples

In this section, we will show two examples of application of the convex piecewise-linear fitting algorithm.
We will use it to fit a set of 1D and 2D data points, obtained from 2 convex functions, specifically:

$$f_1 : \mathbb{R} \to \mathbb{R}, \quad f_1(x) = y = 0.1x^2 \tag{4.65a}$$

$$f_2 : \mathbb{R}^2 \to \mathbb{R}, \quad f_2(\boldsymbol{x}) = y = 0.1(x_1^2 + x_2^2) \tag{4.65b}$$

For function 1, data points are $\{(x_i, y_i)\}_i = \{(x_i, f_1(x_i))\}_i$, $\{x_i\}_i = [-5 : 0.5 : 5]$.
The algorithm is set up with $k = 10$, $N_{iter} = 10$; the initial partitions are generated by setting up the MATLAB random number generator with the `'default'` seed.
The algorithm returns $\boldsymbol{\alpha}$ after only 1 iteration:

$$\boldsymbol{\alpha} = \left((a_j)_j, (b_j)_j\right)$$

$$(a_j)_j^T = \begin{pmatrix} 0.35 \\ 0.95 \\ 0.65 \\ 0.1 \\ -0.8 \\ -0.3 \end{pmatrix}, \quad (b_j)_j^T = \begin{pmatrix} -0.3 \\ -2.25 \\ -1.02 \\ -0.00833 \\ -1.55 \\ -0.175 \end{pmatrix}$$

$(a_j)_j$ and $(b_j)_j$ have $6 < k$ elements, so 4 empty partitions have been generated and removed.

Figure 4.4 depicts the max-affine interpolating function with respect to data points:

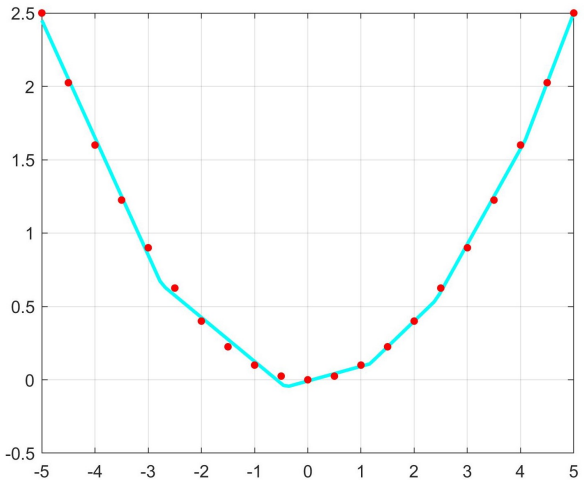


Figure 4.4. Convex piecewise-linear fit of a 1D convex function. Data points are depicted in red, while the max-affine interpolating function is depicted in cyan.

We see that the interpolating function is the envelope of 6 lines, having angular coefficient a_j and intercept b_j, for $j = 1, ..., 6$.

For function 2, data points are $\{(\boldsymbol{x}_i, y_i)\}_i = \{(\boldsymbol{x}_i, f_2(\boldsymbol{x}_i))\}_i$, $\{\boldsymbol{x}_i\}_i = [-5 : 0.5 : 5] \times [-5 : 0.5 : 5]$.
The algorithm is set up with $k = 20$, $N_{iter} = 50$; the initial partitions are generated by setting up the MATLAB random number generator with the `'default'` seed.
The algorithm returns $\boldsymbol{\alpha}$ after 16 iterations:

$$\boldsymbol{\alpha} = \left((\boldsymbol{a}_j)_j, (b_j)_j\right)$$

$$(\boldsymbol{a}_j)_j^T = \begin{pmatrix} 0.688 & 0.246 \\ 0.75 & -0.578 \\ -0.702 & -0.299 \\ 0.439 & 0.827 \\ 0.196 & 0.342 \\ -0.757 & 0.79 \\ -0.351 & 0.354 \\ 0.31 & -0.246 \\ -0.863 & 0.208 \\ 0.0964 & -0.777 \\ -0.65 & -0.8 \\ -0.147 & 0.829 \\ -0.161 & -0.18 \\ 0.837 & -0.159 \\ 0.908 & 0.724 \\ 0.75 & -0.9 \end{pmatrix}, \quad (b_j)_j^T = \begin{pmatrix} -1.21 \\ -2.15 \\ -1.31 \\ -2.08 \\ -0.29 \\ -2.87 \\ -0.506 \\ -0.295 \\ -1.86 \\ -1.37 \\ -2.47 \\ -1.66 \\ -0.0165 \\ -1.72 \\ -3.27 \\ -3.34 \end{pmatrix}$$

$(\boldsymbol{a}_j)_j$ and $(b_j)_j$ have $16 < k$ elements, so 4 empty partitions have been generated and removed.

We see that the interpolating function is the envelope of 16 planes, having coefficients $(\boldsymbol{a}_j^T, b_j)$, for $j = 1, ..., 16$.

Figure 4.5 depicts the interpolating max-affine function with respect to data points:

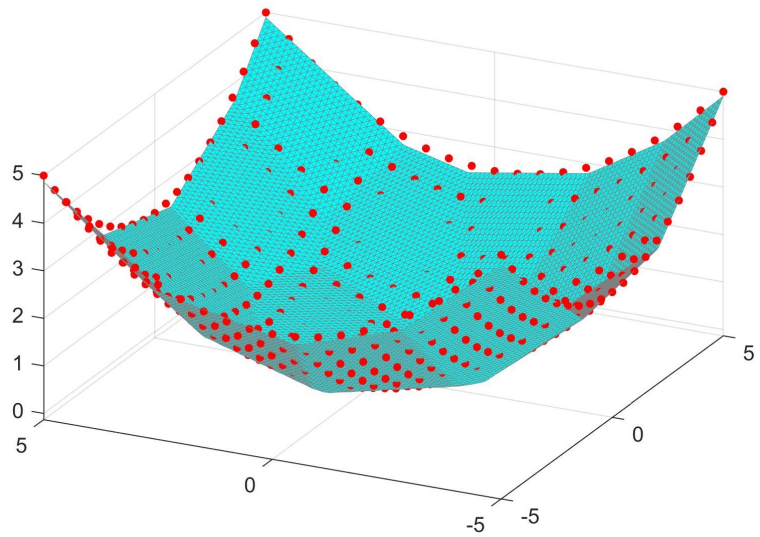


Figure 4.5. Convex piecewise-linear fit of a 2D convex function. Data points are depicted in red, while the max-affine interpolating function is depicted in cyan.

Figure 4.6 depicts instead the Voronoi sets associated to the initial partitions:

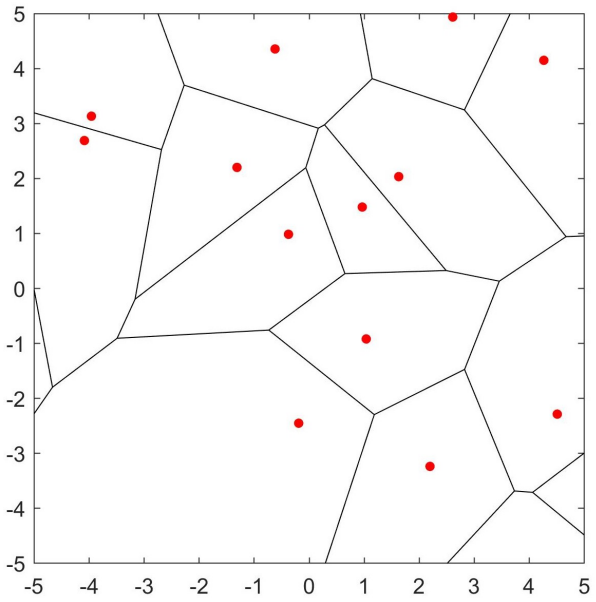


Figure 4.6. Voronoi sets of the initial partitions. Seed points are depicted in red, while the boundaries of each set are marked with black lines.

As pointed out in Note 4.7, in Figure 4.6 we see only 16 out of 20 generated partitions; this means that 4 partitions have been generated outside the subset of data points in $\mathbb{R}^2$ (i.e. $[-5,5] \times [-5,5]$). Indeed, we have seen that $(\boldsymbol{a}_j)_j$ and $(b_j)_j$ have only 16 elements, meaning that the 4 empty partitions have been removed from $P_j^{(0)}$, $j = 1, ..., 20$.

4.6.6. Relaxed LMPC optimization problem (second version)

From the relaxations shown in § 4.6.4 and § 4.6.5, we can formulate a second version of the relaxed LMPC optimization problem:

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k}{\arg\min}\, J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k)$$

$$J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \tilde{P}^{j-1}(\boldsymbol{x}_{N|k}) \tag{4.66a}$$

$$\tilde{P}^{j-1}(\boldsymbol{x}) = \max\left({\boldsymbol{a}_1^{j-1}}^T \boldsymbol{x} + b_1^{j-1}, ..., {\boldsymbol{a}_K^{j-1}}^T \boldsymbol{x} + b_K^{j-1} \right) \tag{4.66b}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \quad t = 0, 1, ..., N-1 \tag{4.66c}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.66d}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \quad \boldsymbol{u}_{t|k} \in \mathcal{U}, \quad t = 0, 1, ..., N-1 \tag{4.66e}$$

$$\boldsymbol{x}_{N|k} \in CS^{j-1} \tag{4.66f}$$

Equivalently, as noted in [11], by recalling that (4.66b) is equivalent to:

$$\begin{aligned}
&\tilde{P}^{j-1}(\boldsymbol{x}) = \min_{c} c \\
&\text{subject to:} \\
&c \geq {\boldsymbol{a}_s^{j-1}}^T \boldsymbol{x} + b_s^{j-1}, \quad s = 1, ..., K
\end{aligned} \tag{4.67}$$

the relaxed optimization problem can be rewritten as:

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}, c^*) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k, c}{\arg\min}\, J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, c)$$

$$J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, c) = \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + c \tag{4.68a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{f}(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}), \quad t = 0, 1, ..., N-1 \tag{4.68b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.68c}$$

$$\boldsymbol{x}_{t|k} \in \mathcal{X}, \quad \boldsymbol{u}_{t|k} \in \mathcal{U}, \quad t = 0, 1, ..., N-1 \tag{4.68d}$$

$$\boldsymbol{A}_{CS}^{j-1} \boldsymbol{x}_{N|k} \leq \boldsymbol{b}_{CS}^{j-1} \tag{4.68e}$$

$$c \geq {\boldsymbol{a}_s^{j-1}}^T \boldsymbol{x}_{N|k} + b_s^{j-1}, \quad s = 1, ..., K \tag{4.68f}$$

With respect to (4.20), the relaxations have introduced the following changes:

- Constraint (4.20e) becomes a linear inequality constraint (4.68e) as in (4.50), that forces the terminal state into the convex safe set CS^{j-1};
- The terminal cost function becomes the argument c of the optimization problem in (4.67); c then becomes an additional optimization variable. Therefore, also the constraints (4.68f) are added to the problem.

4.7. Repetitive LMPC

In (4.4), we assumed that, at each iteration of the LMPC algorithm, the initial conditions of the system are unchanged (i.e. $\boldsymbol{x}_S = \boldsymbol{x}_0^j$, $\forall j \geq 0$).

LMPC, however, is specifically suited for systems performing repetitive tasks.
In a repetitive framework, the initial conditions at the j-th iteration are, in general,

function of the final state of the previous iteration $j-1$; in the most simple case, it holds:

$$\boldsymbol{x}_0^{j+1} = \boldsymbol{x}_{T_j}^j, \quad \forall j \geq 0 \tag{4.69}$$

(4.69) replaces (4.4) when implementing a repetitive LMPC algorithm.

4.8. LMPC for quadrotors

We now apply all the LMPC concepts that have been described until now for the purpose of controlling the quadrotor described in Chapter 2.

As already done in Chapter 3, we consider, as system to be controlled, the nonlinear discrete-time dynamic model of the quadrotor in Frenet coordinates (2.94):

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_Q(\boldsymbol{x}_k, \boldsymbol{u}_k, K(s_k)) \tag{4.70}$$

When we have used MPC, our objective was to make the quadrotor to track a specific trajectory within the race track.
Now, using LMPC, the goal is to formulate a relaxed LMPC algorithm (second version) (4.68) and (4.23) that pilots the quadrotor around the track and that *autonomously learns* which is the best path to follow to *minimize the lap time*.
This means that we want the algorithm to perform an *autonomous and optimal path planning*.

In the following, we will construct every element of the LMPC problem, to eventually derive the final control algorithm.

We start saying that all the considerations regarding track definition and curvature propagation and relaxation hold also for the LMPC algorithm; therefore, we refer the reader to § 3.7.1 and § 3.7.2.

4.8.1. Safe set

Initialization

As required by Assumption 4.1, the LMPC has to be initialized with a SS^0 containing a first feasible trajectory $\boldsymbol{X}^0$ converging to the goal state $\boldsymbol{x}_F$.

We will generate this trajectory by means of the *MPC algorithm for quadrotor trajectory tracking* that we have developed in Chapter 3. Specifically, the first trajectory $\boldsymbol{X}^0$ will start from the initial state $\boldsymbol{x}_S = \boldsymbol{x}_0^0$ and will travel across the whole track until crossing the finish line, corresponding to the goal curvilinear abscissa $s_F = L_{track}$.

Since this trajectory has to be feasible wrt LMPC constraints, we must ensure that such constraints are a subset of those in the MPC optimization problem.

Construction

In the theoretical formulation, the SS^j contains the whole state variables $\boldsymbol{x}_k^j$ constituting the generated trajectories $\boldsymbol{X^0}, ..., \boldsymbol{X}^j$. However, when the number of states n of the system is large, this operation can become quite costly.
Specifically, when we compute the convex hull CS^j of SS^j, to obtain the linear inequality constraints $\boldsymbol{A}_{CS}^j$, $\boldsymbol{b}_{CS}^j$ to relax the LMPC optimization problem, the larger is n, the higher is the computational time needed to compute such constraints. Moreover, the higher is n, the higher will be the dimensions of $\boldsymbol{A}_{CS}^j$ and $\boldsymbol{b}_{CS}^j$, which, if too big, can slow down too much the solver due to the high number of constraints to be satisfied.

Therefore, it is convenient to store in the SS^j only *some* of the states composing $\boldsymbol{x}_k^j$.

For our purposes, we will store only two states: the curvilinear abscissa s and the lateral distance d. Since the LMPC has the goal of optimizing the planar trajectory (i.e. the one on the xy plane) to minimize the lap time, s and d are the only states needed by the algorithm to find, trough autonomous learning, such optimal path on the track.

Note 4.8 The SS^j will contain the states (s, d) composing each LMPC trajectory up to iteration j; these points are used, in the relaxed framework, to construct the max-affine interpolating function $\tilde{P}^j(\boldsymbol{x})$, which approximates the terminal cost barycentric function. This means that, to obtain a good first terminal cost function $\tilde{P}^0(\boldsymbol{x})$, which is used in the optimization problem at iteration 1, it is necessary that the first trajectory $\boldsymbol{X}^0$ "explores" as many states (s, d) as possible while travelling on the track.

Therefore, the MPC is set up to generate a $\boldsymbol{X}^0$ that oscillates in $[-d_{lim}, d_{lim}]$ while travelling from $s = 0$ to $s = L_{track}$. In this way, the first trajectory will sweep a good amount of states within the set $[0, L_{track}] \times [-d_{lim}, d_{lim}]$, providing a good first

terminal cost function $\tilde{P}^0(\boldsymbol{x})$.

In general, if the choice of $\boldsymbol{X}^0$ is good, $\tilde{P}^0(\boldsymbol{x})$ may be used as terminal cost function also for iterations $j \geq 1$, i.e.:

$$\tilde{P}^j(\boldsymbol{x}) \doteq \tilde{P}^0(\boldsymbol{x}), \quad \forall j \geq 1 \tag{4.71}$$

Recomputing $\tilde{P}^j(\boldsymbol{x})$ at every iteration, with the additional points in SS^j, would not give any improvement in the interpolating function; by contrary, too many over-imposed data points (which come out when the trajectories tend to converge to the optimal one) might provide a worst interpolation.

4.8.2. Cost function

For our LMPC optimization problem, the stage cost function h will be the following:

$$h(\boldsymbol{x}_k, \boldsymbol{u}_k) = (\boldsymbol{x}_k - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}_k - \boldsymbol{x}_F) \tag{4.72}$$

The goal state will be:

$$\boldsymbol{x}_F = \begin{pmatrix} \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & 1.2 \cdot L_{track} & \star & \star \end{pmatrix} \tag{4.73}$$

in which we recall that the states denoted with "$\star$" are associated to a weight equal to 0 in the matrix $\boldsymbol{P}$, so their value is not needed to be specified.
This means that only the state s (i.e. the 11^{th} state) is penalized.

This stage cost function satisfies (4.7), but most importantly it fits well with our task of generating the path minimizing the lap time.
Indeed, this stage cost function penalizes only the distance from the finish line, without any penalty on the other system states. This means that the LMPC control algorithm will try to fly the quadrotor towards the finish line as quickly as possible and, being no penalties on the other system states, it is free to make the quadrotor to follow any possible path inside the race track, as long as it goes towards the finish line. This possibility to move along any trajectory enables the learning capability of LMPC.

Note 4.9 We see that in $\boldsymbol{x}_F$ the value of L_{track} is multiplied by a coefficient > 1. This is done to avoid that the quadrotor decelerates when approaching the

finish line. The coefficient moves virtually the finish line ahead of its real position, meaning that the quadrotor will cross the true finish line without slowing down.

However, to ensure a better behaviour of the quadrotor under control, additional terms are added to the complete LMPC cost function:

$$
\begin{aligned}
&J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c) = \\
&= \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \sum_{t=0}^{N-1} h'(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \sum_{t=1}^{N-1} h''(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}, \boldsymbol{x}_{t-1|k}, \boldsymbol{u}_{t-1|k}) + \\
&+ K_1 e_1^2 + K_2 e_2^2 + K_3 e_3^2 + c = \\
&= \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F) + && \text{(4.74a)} \\
&+ \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} + && \text{(4.74b)} \\
&+ \sum_{t=1}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k})^T \boldsymbol{Q}_\Delta (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k}) + (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k})^T \boldsymbol{R}_\Delta (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k}) + \\
& && \text{(4.74c)} \\
&+ K_1 e_1^2 + K_2 e_2^2 + K_3 e_3^2 + c && \text{(4.74d)}
\end{aligned}
$$

The cost-to-go, iteration cost, and terminal cost function are computed using only the stage cost function h:

$$
\begin{aligned}
J^j_{[k,\infty]}(\boldsymbol{x}^j_k) &= \sum_{t=k}^{\infty} h(\boldsymbol{x}^j_t, \boldsymbol{u}^j_t) = \sum_{t=k}^{\infty} (\boldsymbol{x}^j_t - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}^j_t - \boldsymbol{x}_F) \\
J^j_{[0,\infty]}(\boldsymbol{x}^j_0) &= J^j_0 = \sum_{k=0}^{\infty} h(\boldsymbol{x}^j_k, \boldsymbol{u}^j_k) = \sum_{k=0}^{\infty} (\boldsymbol{x}^j_k - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}^j_k - \boldsymbol{x}_F)
\end{aligned} \tag{4.75}
$$

The term (4.74a) is the usual one with the stage cost function h.

The term (4.74b) is similar to the one used in the MPC for quadrotor trajectory tracking and has the purpose to take into account all the other relevant states that are not present in the stage cost function h (such as the altitude z, the yaw angle ψ, and the lateral distance d). Specifically, the additional stage cost function h' will partially shape the final trajectory according to the reference state $\boldsymbol{x}_r$, acting as a "guide" for the LMPC algorithm; it also penalizes the amplitude of the input signal. In the following, we will define the value for $\boldsymbol{x}_r$.

The term (4.74c) was also used in the MPC for quadrotor trajectory tracking (§ 3.7.3)

and has the purpose to penalize the variation of the system states and inputs, forcing the quadrotor to follow a smoother trajectory, without abrupt changes in its velocity and accelerations.

The term (4.74d), finally, contains the terms related to three slack variables $e_1, e_2, e_3 \in \mathbb{R}$, that will be inserted in the problem constraints (see next section), and the additional optimization variable c that acts as terminal cost in the relaxed LMPC optimization problem (4.68).

Reference state

We will use the following value for $\boldsymbol{x}_r$:

$$\boldsymbol{x}_r = \begin{pmatrix} z_r & \star & \star & 45° & \star & \star & \star & \star & \star & \star & \star & d_r(s_k) & \star \end{pmatrix} \tag{4.76}$$

where the function $d_r(s_k)$ is:

$$d_r(s_k) = \begin{cases} c_{d_r} \cdot d_{lim} & \text{if } K(s_k) = 0 \\ \text{sign}(K(s_k)) \cdot c_{d_r} \cdot d_{lim} & \text{if } K(s_k) \neq 0 \end{cases} \tag{4.77}$$

where $c_{d_r} \in [0, 1]$.

The reference values z_r and $\psi_r = 45°$ suggest the quadrotor to keep an altitude near z_r and to fly in a cross configuration.

The variable value of d_r, instead, is fundamental to help the LMPC algorithm to find the optimal path minimizing the lap time.
The value of d_r is set to $c_{d_r} \cdot d_{lim}$ when the curvature of the track, at the current location s_k of the quadrotor, is $K(s_k) \geq 0$; the value, instead, is set to $-c_{d_r} \cdot d_{lim}$ when the curvature is $K(s_k) < 0$; the coefficient c_{d_r} is used to tune the reference value.
This means that the quadrotor will tend to stay close to the inner border of the track when it is on a straight line or on a left curve (wrt the Frenet frame); instead, it will tend to stay close to the outer border of the track when it is on a right curve. Qualitatively, this is how, in a closed track, a pilot would conduct a vehicle to efficiency "cut the curves" to reach the finish line in the shortest amount of time.
Therefore, the value of $d_r = d_r(s_k)$ suggests the LMPC to pilot in this way the quadrotor, reaching more easily the optimal path minimizing the lap time.

Note 4.10 The terms (4.74a) and (4.74b) of the cost function can be combined to a unique stage cost function as follows:

$$\sum_{t=0}^{N-1}(\boldsymbol{x}_{t|k}-(\boldsymbol{x}_r+\boldsymbol{x}_F))^T(\boldsymbol{Q}+\boldsymbol{P})(\boldsymbol{x}_{t|k}-(\boldsymbol{x}_r+\boldsymbol{x}_F))+\boldsymbol{u}_{t|k}^T\boldsymbol{R}\boldsymbol{u}_{t|k} \tag{4.78}$$

where:

$$\boldsymbol{x}_r+\boldsymbol{x}_F=\begin{pmatrix} z_r & \star & \star & 45^\circ & \star & \star & \star & \star & \star & \star & 1.2\cdot L_{track} & d_r(s_k) & \star \end{pmatrix} \tag{4.79}$$

4.8.3. Constraints

The following constraints will be added to the LMPC optimization problem:

- ATV model equations constraints (implicit prediction form);
- track boundaries, i.e. bounds on the lateral distance d;
- bounds on the altitude z;
- the constraints required by the relaxed LMPC optimization problem (second version) (4.68).

The constraints will be expressed as follows:

$$\boldsymbol{x}_{t+1|k}=\boldsymbol{A}_k^j\boldsymbol{x}_{t|k}+\boldsymbol{B}_k^j\boldsymbol{u}_{t|k}+\boldsymbol{c}_k^j,\quad t=0,1,...,N-1 \tag{4.80a}$$
$$\boldsymbol{x}_{0|k}=\boldsymbol{x}_k^j \tag{4.80b}$$
$$-d_{lim}-e_1\leq d_{t|k}\leq d_{lim}+e_1,\quad t=0,1,...,N-1 \tag{4.80c}$$
$$z_{(m)}-e_2\leq z_{t|k}\leq z_{(M)}+e_2,\quad t=0,1,...,N-1 \tag{4.80d}$$
$$\boldsymbol{A}_{CS}^{j-1}\boldsymbol{x}_{N|k}\leq\boldsymbol{b}_{CS}^{j-1}+e_3 \tag{4.80e}$$
$$c\geq\boldsymbol{a}_s^{j-1^T}\boldsymbol{x}_{N|k}+b_s^{j-1},\quad s=1,...,K \tag{4.80f}$$
$$e_1\geq 0 \tag{4.80g}$$
$$e_2\geq 0 \tag{4.80h}$$

where (m) stands for minimum value and (M) stands for maximum value.

As already pointed out in Note 3.7, the ATV model for the quadrotor is computed using the new model (3.46), $\boldsymbol{x}_{k+1}=\hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k,\boldsymbol{u}_k,K)$, having the curvature expressed as

a constant parameter K, independent from s (curvature propagation). Therefore:

$$\boldsymbol{A}_k^j = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{x}}(\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, K) \tag{4.81}$$

$$\boldsymbol{B}_k^j = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{u}}(\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, K) \tag{4.82}$$

$$\boldsymbol{c}_k^j = \hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, K) - \boldsymbol{A}_k^j \boldsymbol{x}_k^j - \boldsymbol{B}_k^j \boldsymbol{u}_{k-1}^j \tag{4.83}$$

Constraints (4.80a)-(4.80b) are associated to the ATV model equations.

Constraint (4.80c) is associated to the track boundaries/width; thus, it sets lower and upper bounds on the lateral distance d of the quadrotor. The slack variable e_1 softens the constraint; this allows the quadrotor to slightly go outside the bounds, preventing possible infeasibility when the quadrotor approaches too closely the track borders.

Constraint (4.80d) sets lower and upper bounds on the altitude z of the quadrotor, since in LMPC there is not a reference altitude (expect for the indicative value z_r in $\boldsymbol{x}_r$). The slack variable e_2 softens the constraint; this allows the quadrotor to slightly go outside the bounds, preventing possible infeasibility when the quadrotor approaches too closely the vertical borders.

Constraint (4.80e) is the terminal constraint forcing the terminal state $\boldsymbol{x}_{N|k}$ in the convex safe set CS^{j-1}. The slack variable e_3 softens the constraint.

Constraint (4.80f) is the one associated to the auxiliary optimization problem (4.67) that defines the value of the barycentric terminal cost function $\tilde{P}(\boldsymbol{x})$ (which is, as we know, interpolated by a max-affine function) evaluated in the terminal state $\boldsymbol{x}_{N|k}$.

Constraints (4.80g)-(4.80h), finally, set the slack variables e_1 and e_2 as non-negative, which is necessary to soften correctly constraints (4.80c)-(4.80d).

Note 4.11 We see that the LMPC constraints (4.80c)-(4.80d) are a subset of MPC constraints (3.50c)-(3.50h) except if $z_{(m)} > 0$. In that case, the first points of the first trajectory will be infeasible only for the altitude constraint (since the quadrotor starts at $z_0 = 0$); this, however, is not a problem, since typically, choosing a value of N not too small, such states of SS^j will unlikely be chosen as the terminal state at the beginning of the next iteration.

Note 4.12 It is important to notice that, now, being all the constraints linear equalities or inequalities and being the cost function quadratic, the optimization problem is a QP, meaning that the LMPC is linear.

4.8.4. Optimization problem

We can now write the LMPC optimization problem for quadrotor optimal path planning:

LMPC optimization problem for quadrotor optimal path planning

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}, e_1^{j^*}, e_2^{j^*}, e_3^{j^*}, c^{j^*}) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c}{\arg\min} J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c)$$

$$\begin{aligned} J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c) = \\ = \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F) + \\ + \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} + \\ + \sum_{t=1}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k})^T \boldsymbol{Q}_\Delta (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k}) + (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k})^T \boldsymbol{R}_\Delta (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k}) + \\ + K_1 e_1^2 + K_2 e_2^2 + K_3 e_3^2 + c \end{aligned} \tag{4.84a}$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{A}_k^j \boldsymbol{x}_{t|k} + \boldsymbol{B}_k^j \boldsymbol{u}_{t|k} + \boldsymbol{c}_k^j, \quad t = 0, 1, ..., N-1 \tag{4.84b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.84c}$$

$$-d_{lim} - e_1 \leq d_{t|k} \leq d_{lim} + e_1, \quad t = 0, 1, ..., N-1 \tag{4.84d}$$

$$z_{(m)} - e_2 \leq z_{t|k} \leq z_{(M)} + e_2, \quad t = 0, 1, ..., N-1 \tag{4.84e}$$

$$\boldsymbol{A}_{CS}^{j-1} \boldsymbol{x}_{N|k} \leq \boldsymbol{b}_{CS}^{j-1} + e_3 \tag{4.84f}$$

$$c \geq \boldsymbol{a}_s^{j-1^T} \boldsymbol{x}_{N|k} + b_s^{j-1}, \quad s = 1, ..., k_{fit} \tag{4.84g}$$

$$e_1 \geq 0 \tag{4.84h}$$

$$e_2 \geq 0 \tag{4.84i}$$

with:

$$\boldsymbol{A}_k^j = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{x}} (\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, \tilde{K}(s_k^j)) \tag{4.84j}$$

$$\boldsymbol{B}_k^j = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{u}}(\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, \tilde{K}(s_k^j)) \tag{4.84k}$$

$$\boldsymbol{c}_k^j = \hat{\boldsymbol{f}}_Q(\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, \tilde{K}(s_k^j)) - \boldsymbol{A}_k^j \boldsymbol{x}_k^j - \boldsymbol{B}_k^j \boldsymbol{u}_{k-1}^j \tag{4.84l}$$

4.8.5. Algorithm

The complete LMPC algorithm for quadrotor optimal path planning is composed by the optimization problem (4.84) and the receding horizon control law (4.23).

As already mentioned in Note 3.10, it is worth noticing that, even though we have used in the LMPC optimization problem (4.84) the ATV model of the quadrotor, the next state $\boldsymbol{x}_{k+1}^j$ is obtained applying the optimal control input $\boldsymbol{u}_k^j = \boldsymbol{u}_{0|k}^{j^*}$ to the complete nonlinear model of the quadrotor (2.94):

$$\boldsymbol{x}_{k+1}^j = \boldsymbol{f}_Q(\boldsymbol{x}_k^j, \boldsymbol{u}_k^j, K(s_k^j)) \tag{4.85}$$

Note 4.13 Being in Frenet coordinates, it is necessary to modify the expression (4.69). In fact, when the quadrotor crosses the finish line (so at the end of each lap), the state variables s (curvilinear abscissa) and ψ_2 (Frenet angle) have to be reset from $s = L_{track}$ to 0 and from $\psi_2 = 360°$ to 0. Therefore, (4.69) becomes:

$$\boldsymbol{x}_0^{j+1} = \boldsymbol{x}_{T_j}^j - L_{track}\, \boldsymbol{e}_{11} - 360°\, \boldsymbol{e}_{13}, \quad \forall j \geq 0 \tag{4.86}$$

where $\boldsymbol{e}_k$ is the k-th vector of the canonical base of $\mathbb{R}^n$ (with $n = 13$, i.e. the number of states of the quadrotor model):

$$\boldsymbol{e}_k = (0, ..., 0, \underbrace{1}_{k\text{-th element}}, 0, ..., 0)^T \in \mathbb{R}^n \tag{4.87}$$

Algorithm 4.3 LMPC for quadrotor optimal path planning

Inputs: $\boldsymbol{x}_S$; N_{iter} (number of LMPC iterations); N (LMPC prediction horizon)
Outputs: $\boldsymbol{X}^0, \boldsymbol{X}^1, ..., \boldsymbol{X}^{N_{iter}}$; $\boldsymbol{U}^0, \boldsymbol{U}^1, ..., \boldsymbol{U}^{N_{iter}}$

(1) Generate the first feasible trajectory $\boldsymbol{X}^0 = (\boldsymbol{x}_0^0, \boldsymbol{x}_1^0, ..., \boldsymbol{x}_{T_0}^0)$ using the MPC algorithm for quadrotor trajectory tracking (Algorithm 3.2), with initial state $\boldsymbol{x}_0^0 = \boldsymbol{x}_S$

(2) Construct the initial sampled safe set $SS^0 = \{\boldsymbol{X}^0\}$

(3) Using Algorithm 4.2, perform the convex piecewise-linear fitting of the data points $SS^0 = \{\boldsymbol{x}_0^0, ..., \boldsymbol{x}_{T_0}^0\}$ and $Q^0(SS^0) = \{J_0^0, ..., J_{T_0}^0\}$, obtaining the coefficients $\boldsymbol{\alpha}^0 = (\boldsymbol{a}_s^0, b_s^0)_{s=1}^{k_{fit}}$ of the max-affine terminal cost barycentric function $\tilde{P}^0(\boldsymbol{x})$

(4) **For** $j = 1, 2, ..., N_{iter}$:

(4.1) • **If** the LMPC is repetitive, set the initial state $\boldsymbol{x}_0^j = \boldsymbol{x}_{T_{j-1}}^{j-1} - L_{track}\, \boldsymbol{e}_{11} - 2\pi\, \boldsymbol{e}_{13}$, as in (4.86), and store it in $\boldsymbol{X}^j$;

• otherwise (the LMPC is non-repetitive), set the initial state $\boldsymbol{x}_0^j = \boldsymbol{x}_S$, as in (4.4), and store it in $\boldsymbol{X}^j$

(4.2) Compute, from SS^{j-1}, the convex safe set CS^{j-1} and its related linear constraints $\boldsymbol{A}_{CS}^{j-1}$, $\boldsymbol{b}_{CS}^{j-1}$

(4.3) **For** $k = 0, 1, ...$:

(4.3.1) • **If** the exit condition on $\boldsymbol{x}_k^j$ is satisfied, **break** the cycle (at $k = T_j$);

• otherwise, continue

(4.3.2) Initialize the LMPC optimization problem (4.84) with $\boldsymbol{x}_k^j$, $\boldsymbol{u}_{k-1}^j$ (**if** $k = 0$, set $\boldsymbol{u}_{k-1}^j \doteq \boldsymbol{0}$), $\boldsymbol{A}_{CS}^{j-1}$, $\boldsymbol{b}_{CS}^{j-1}$, and $\boldsymbol{\alpha}^0$

(4.3.3) Solve the LMPC optimization problem, obtaining the optimal predicted state trajectory $\boldsymbol{X}_k^{j^*} = (\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{x}_{1|k}^{j^*}, ..., \boldsymbol{x}_{N|k}^{j^*})$ and optimal predicted input sequence $\boldsymbol{U}_k^{j^*} = (\boldsymbol{u}_{0|k}^{j^*}, \boldsymbol{u}_{1|k}^{j^*}, ..., \boldsymbol{u}_{N-1|k}^{j^*})$

(4.3.4) Apply the input $\boldsymbol{u}_k^j \doteq \boldsymbol{u}_{0|k}^{j^*}$ to the system (2.94), as in (4.23), obtaining the next state $\boldsymbol{x}_{k+1}^j$

(4.3.5) Store $\boldsymbol{x}_{k+1}^j$ and $\boldsymbol{u}_k^j$ in $\boldsymbol{X}^j$ and $\boldsymbol{U}^j$

(4.4) Augment the sampled safe set with the new trajectory $\boldsymbol{X}^j = (\boldsymbol{x}_0^j, \boldsymbol{x}_1^j, ..., \boldsymbol{x}_{T_j}^j)$:

$$SS^j = SS^{j-1} \cup \{\boldsymbol{X}^j\}$$

(5) **Return** all the generated trajectories and input sequences:

$$\boldsymbol{X}^0, \boldsymbol{X}^1, ..., \boldsymbol{X}^{N_{iter}}$$
$$\boldsymbol{U}^0, \boldsymbol{U}^1, ..., \boldsymbol{U}^{N_{iter}}$$

As done in Note 4.6, we need to define the exit condition that stops the algorithm as soon as the current state $\boldsymbol{x}_k$ is sufficiently close to the goal state $\boldsymbol{x}_F$. For our needs, the exit condition is when the quadrotor has crossed the finish line; therefore, the algorithm should stop as soon as the value of the curvilinear abscissa s_k is greater or equal to the track length L_{track}:

$$s_k \overset{?}{\geq} L_{track} \tag{4.88}$$

4.9. LMPC with obstacle avoidance

In the previous sections, we have developed the complete LMPC algorithm for quadrotor optimal path planning, which is able to autonomously find the optimal trajectory minimizing the lap time.

We now implement an additional feature for this algorithm, which is the possibility to include, within the track, different kinds of obstacles. Therefore, the control algorithm, along with finding the optimal path for lap time minimization, must also ensure that the quadrotor avoids these obstacles while flying on the track.

4.9.1. Obstacles definition

Three types of obstacles can be included within the flight area:

- a *horizontal narrowing* of the track: over a certain interval $s = [s_i, s_o]$, the value of the bounds on the lateral distance d, defined by constraint (4.84d), are restricted to:

$$-d_{lim} < d_{o,l} \leq d_{t|k} \leq d_{o,u} < d_{lim}, \quad t = 0, 1, ..., N-1 \tag{4.89}$$

  where $d_{o,l}$ and $d_{o,u}$ define respectively the new lower (i.e. outer) and upper (i.e. inner) bounds on the lateral distance given by the obstacle.

  The obstacle is depicted in Figure 4.7:

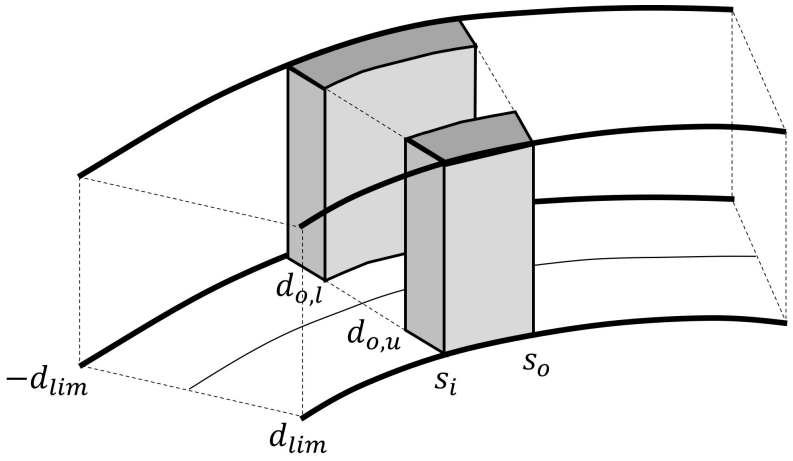


Figure 4.7. Horizontal narrowing obstacle, placed within a generic part of the track

For this obstacle, we can define two function $d_{o,l}(s)$ and $d_{o,u}(s)$ which describe respectively the value of the lower and upper bound on d over the whole track; specifically, they are equal to:

$$d_{o,l}(s) = \begin{cases} d_{o,l} & \text{if } s \in [s_i, s_o] \\ -d_{lim} & \text{if } s \notin [s_i, s_o] \end{cases}, \quad d_{o,u}(s) = \begin{cases} d_{o,u} & \text{if } s \in [s_i, s_o] \\ d_{lim} & \text{if } s \notin [s_i, s_o] \end{cases} \tag{4.90}$$

If multiple obstacles of this kind are present, in (4.90) will be added multiple cases, one for each obstacle;

- a *vertical narrowing* of the track: over a certain interval $s = [s_i, s_o]$, the value of the bounds on the altitude z, defined by constraint (4.84e), are restricted to:

$$z_{(m)} < z_{o,l} \leq z_{t|k} \leq z_{o,u} < z_{(M)}, \quad t = 0, 1, ..., N-1 \tag{4.91}$$

where $z_{o,l}$ and $z_{o,u}$ define respectively the new lower and upper bounds on the altitude given by the obstacle.

The obstacle is depicted in Figure 4.8:

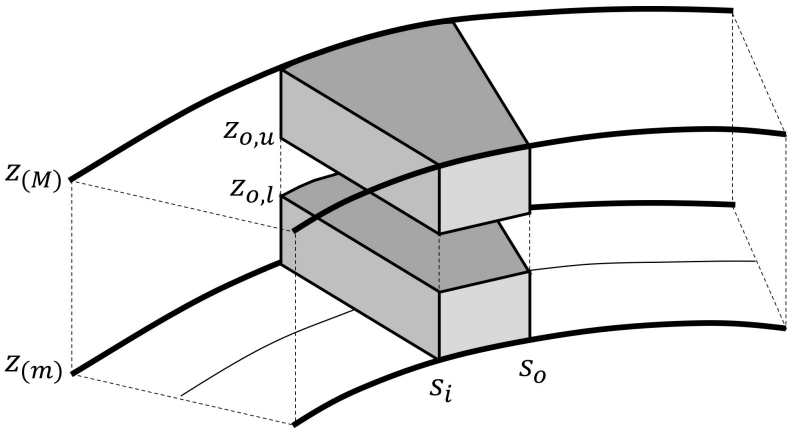


Figure 4.8. Vertical narrowing obstacle, placed within a generic part of the track

For this obstacle, we can define two function $z_{o,l}(s)$ and $z_{o,u}(s)$ which describe respectively the value of the lower and upper bound on z over the whole track; specifically, they are equal to:

$$z_{o,l}(s) = \begin{cases} z_{o,l} & \text{if } s \in [s_i, s_o] \\ z_{(m)} & \text{if } s \notin [s_i, s_o] \end{cases}, \quad z_{o,u}(s) = \begin{cases} z_{o,u} & \text{if } s \in [s_i, s_o] \\ z_{(M)} & \text{if } s \notin [s_i, s_o] \end{cases} \tag{4.92}$$

If multiple obstacles of this kind are present, in (4.92) will be added multiple cases, one for each obstacle;

- a *"square ring"* obstacle, which is the result of the union of a horizontal and a vertical narrowing of the track: over a certain interval $s = [s_i, s_o]$, the values of the bounds on both the lateral distance d and the altitude z, defined by constraints (4.84d) and (4.84e), are restricted to:

$$\begin{aligned} -d_{lim} < d_{o,l} \leq d_{t|k} \leq d_{o,u} < d_{lim}, \quad t = 0, 1, ..., N-1 \\ z_{(m)} < z_{o,l} \leq z_{t|k} \leq z_{o,u} < z_{(M)}, \quad t = 0, 1, ..., N-1 \end{aligned} \tag{4.93}$$

where $d_{o,l}$ and $d_{o,u}$ define respectively the new lower (i.e. outer) and upper (i.e. inner) bounds on the lateral distance given by the obstacle, while $z_{o,l}$ and $z_{o,u}$ define respectively the new lower and upper bounds on the altitude given by the obstacle.

The obstacle is depicted in Figure 4.9:

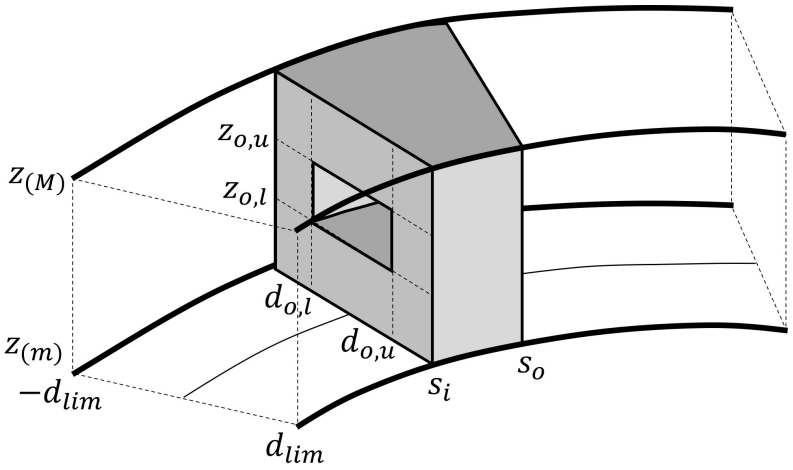


Figure 4.9. Square ring obstacle, placed within a generic part of the track

For this obstacle, we can define two couples of functions $d_{o,l}(s)$, $d_{o,u}(s)$ and $z_{o,l}(s)$, $z_{o,u}(s)$ which describe respectively the value of the lower and upper bounds on d and z over the whole track; specifically, they are equal to (4.90) and (4.92).

If multiple obstacles of this kind are present, in (4.90) and (4.92) will be added multiple cases, one for each obstacle.

In Figure 4.10a is depicted a track containing 3 obstacles: 1 vertical narrowing, 1 horizontal narrowing, and 1 square ring. Specifically, this track will be used in the simulations to test the algorithm (see Chapter 5).

4.9.2. Obstacles implementation

To make the LMPC optimization problem aware of a possible obstacle ahead, the first idea would be to insert the functions $d_{o,l}(s)$, $d_{o,u}(s)$ and $z_{o,l}(s)$, $z_{o,u}(s)$ inside the constraints on d (4.84d) and z (4.84e); in this way, according to the current value of the curvilinear abscissa s_k, the related constraints will force the quadrotor either within the obstacle borders or within the track borders.
However, this approach leads to the same problems that we had when trying to directly insert the curvature function $K(s)$ in the LMPC optimization problem.

Therefore, we deal with this problem in the same way as we did for the curvature in § 3.7.2:

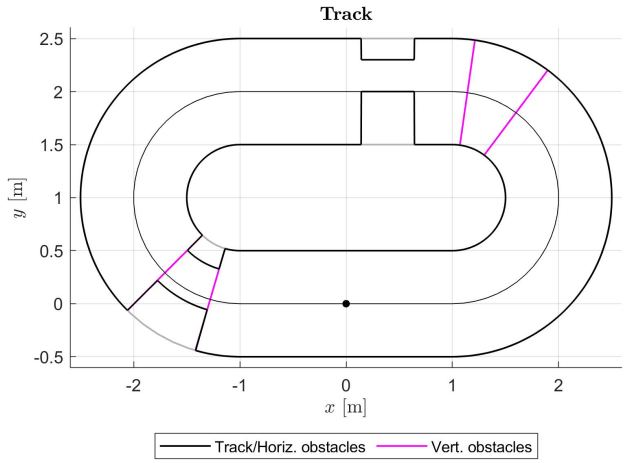
- At each time instant k, the value of the current bounds on d and z, given by $d_{o,l}(s_k)$, $d_{o,u}(s_k)$ and $z_{o,l}(s_k)$, $z_{o,u}(s_k)$, will be passed to the LMPC optimization problem, which will set them as constant values $d_{o,l}$, $d_{o,u}$ and $z_{o,l}$, $z_{o,u}$ in constraints (4.84d) and (4.84e).
  This means that the LMPC optimization problem will compute the optimal predicted states and inputs assuming that, over its prediction horizon, the bounds on d and z are constant to their initial values at $s_k = s_{0|k}$.

- The approach at the previous point solves the issue of embedding the analytical functions $d_{o,l}(s)$, $d_{o,u}(s)$ and $z_{o,l}(s)$, $z_{o,u}(s)$ in the LMPC constraints. However, we now have the problem that an abrupt change in the bounds on d and z will be noticed by LMPC only right at the beginning of the obstacle, since we have imposed the bounds as constant in each LMPC optimization problem. This issue may cause the quadrotor to "crash" on the obstacles, since, being these noticed only when the quadrotor arrives near them, the control algorithm is not able to find a feasible sequence of inputs that menages to successfully avoid such obstacles.

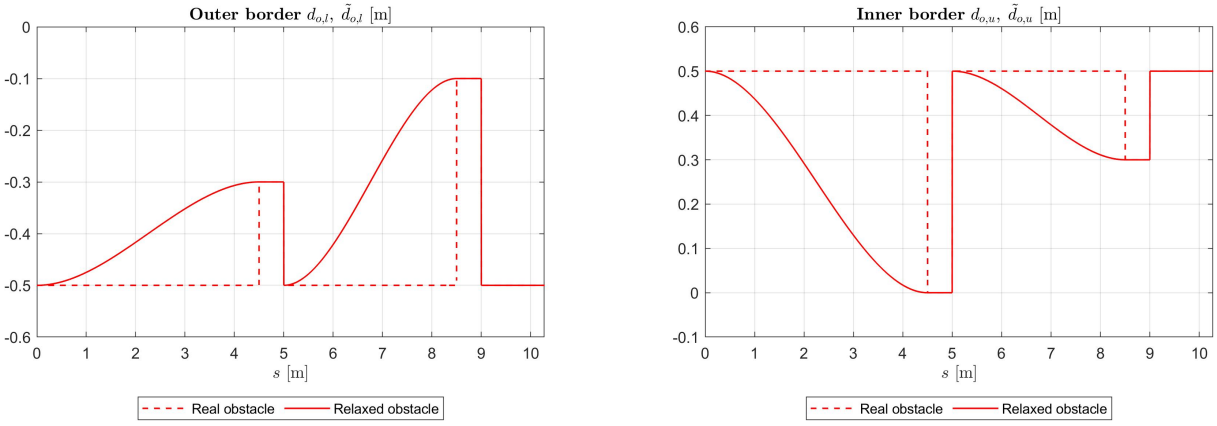
  A possible solution to this problem consists in evaluating the bounds on d and z using relaxed functions $\tilde{d}_{o,l}(s)$, $\tilde{d}_{o,u}(s)$ and $\tilde{z}_{o,l}(s)$, $\tilde{z}_{o,u}(s)$. Such relaxed functions are realized by connecting the constant piecewise segments with 3^{rd}-order polynomials,

ensuring the continuity of the derivatives in the junction points (Figures 4.10b-c). This connection using polynomials allows to reduce the steepness of the *rising* vertical edges of the stepwise bounds given by the obstacles: the gradual change of the bounds in the relaxed functions allows the LMPC to start avoiding in advance the obstacles.

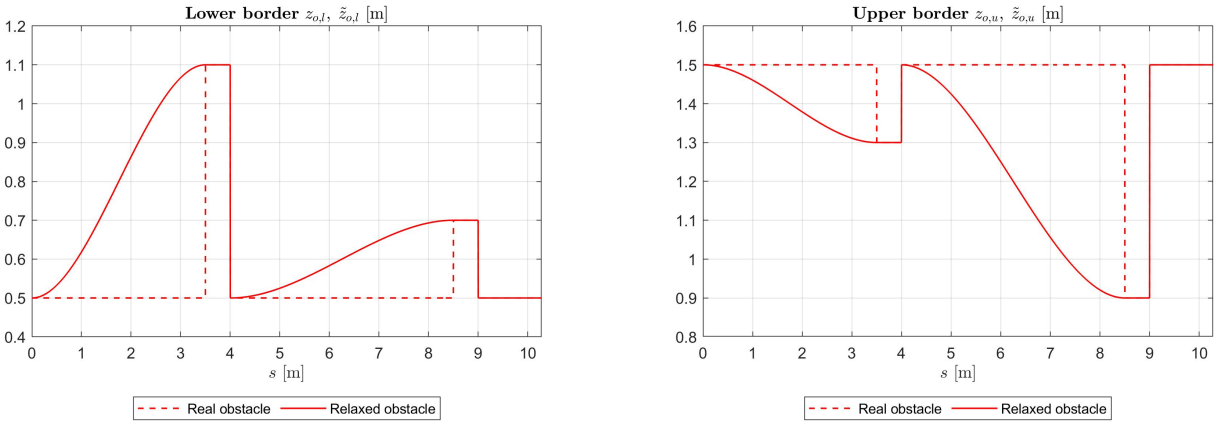
To quantify how much the bounds are relaxed, we define the parameters $d_{o,rel}$ and $z_{o,rel}$, called *obstacles relaxation coefficients*. To ensure the best obstacle avoidance capability, such coefficients should be set very close to 100%; in Figures 4.10b-c, the values of $d_{o,rel}$ and $z_{o,rel}$ are near 100%.



(a) Track with 3 obstacles: 1 vertical narrowing, 1 horizontal narrowing, and 1 square ring



(b) Real and relaxed horiz. obstacles functions $d_{o,l}(s)$, $d_{o,u}(s)$ and $\tilde{d}_{o,l}(s)$, $\tilde{d}_{o,u}(s)$ ($d_{o,rel} = 99.9\%$)



(c) Real and relaxed vertical obstacles functions $z_{o,l}(s)$, $z_{o,u}(s)$ and $\tilde{z}_{o,l}(s)$, $\tilde{z}_{o,u}(s)$ ($z_{o,rel} = 99.9\%$)

Figure 4.10. Track with obstacles and related obstacle functions (real and relaxed)

4.9.3. Safe set

As for the standard LMPC algorithm, the sampled safe set is initialized to SS^0 by means of the MPC algorithm for quadrotor trajectory tracking, which generates the first feasible trajectory $\boldsymbol{X}^0$.

Moreover, in SS^j are stored only some of the states composing each point $\boldsymbol{x}_k^j$ of the trajectories. Specifically, as for standard LMPC, only two states are stored: the curvilinear abscissa s and the lateral distance d.

The observations in Note 4.8 also apply for the LMPC with obstacle avoidance: the MPC is set up to generate a $\boldsymbol{X}^0$ that oscillates within the local bounds $[d_{o,l}(s_k), d_{o,u}(s_k)]$ while travelling from $s = 0$ to $s = L_{track}$; in this way, the first trajectory will sweep a good amount of states within the set $[0, L_{track}] \times [-d_{lim}, d_{lim}]$, providing a good first terminal cost function $\tilde{P}^0(\boldsymbol{x})$.
Also here, if the choice of $\boldsymbol{X}^0$ is good, $\tilde{P}^0(\boldsymbol{x})$ can be used as terminal cost function also for iterations $j \geq 1$.

4.9.4. Cost function

For the LMPC optimization problem with obstacle avoidance, the cost function is very similar to that of standard LMPC.

The stage cost function h is:

$$h(\boldsymbol{x}_k, \boldsymbol{u}_k) = (\boldsymbol{x}_k - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}_k - \boldsymbol{x}_F) \tag{4.94}$$

with goal state:

$$\boldsymbol{x}_F = \begin{pmatrix} \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & 1.2 \cdot L_{track} & \star & \star \end{pmatrix} \tag{4.95}$$

in which we recall that the states denoted with "$\star$" are associated to a weight equal to 0 in the matrix $\boldsymbol{P}$, so their value is not needed to be specified.

To ensure a better behaviour of the quadrotor under control, the complete cost function features the following additional terms:

$$\begin{aligned}
&J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c) = \\
&= \sum_{t=0}^{N-1} h(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \sum_{t=0}^{N-1} h'(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}) + \sum_{t=1}^{N-1} h''(\boldsymbol{x}_{t|k}, \boldsymbol{u}_{t|k}, \boldsymbol{x}_{t-1|k}, \boldsymbol{u}_{t-1|k}) + \\
&+ K_1 e_1^2 + K_2 e_2^2 + K_3 e_3^2 + c = \\
&= \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F) + && \text{(4.96a)} \\
&+ \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} + && \text{(4.96b)} \\
&+ \sum_{t=1}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k})^T \boldsymbol{Q}_\Delta (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k}) + (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k})^T \boldsymbol{R}_\Delta (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k}) + && \text{(4.96c)} \\
&+ K_1 e_1^2 + K_2 e_2^2 + K_3 e_3^2 + c && \text{(4.96d)}
\end{aligned}$$

The cost-to-go, iteration cost, and terminal cost function are computed using only the stage cost function h:

$$\begin{aligned}
J^j_{[k,\infty]}(\boldsymbol{x}^j_k) &= \sum_{t=k}^{\infty} h(\boldsymbol{x}^j_t, \boldsymbol{u}^j_t) = \sum_{t=k}^{\infty} (\boldsymbol{x}^j_t - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}^j_t - \boldsymbol{x}_F) \\
J^j_{[0,\infty]}(\boldsymbol{x}^j_0) &= J^j_0 = \sum_{k=0}^{\infty} h(\boldsymbol{x}^j_k, \boldsymbol{u}^j_k) = \sum_{k=0}^{\infty} (\boldsymbol{x}^j_k - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}^j_k - \boldsymbol{x}_F)
\end{aligned} \tag{4.97}$$

Reference state

We will use the following value for $\boldsymbol{x}_r$:

$$\boldsymbol{x}_r = \begin{pmatrix} z_r(s_k) & \star & \star & 45° & \star & \star & \star & \star & \star & \star & \star & d_r(s_k) & \star \end{pmatrix} \tag{4.98}$$

The function $z_r(s_k)$ is equal to:

$$z_r(s_k) = \frac{\tilde{z}_{o,u}(s_k) + \tilde{z}_{o,l}(s_k)}{2} \tag{4.99}$$

The function $d_r(s_k)$ is equal to:

$$d_r(s_k) = \begin{cases} c_{d_r} \cdot \tilde{d}_{o,u}(s_k) & \text{if } K(s_k) \geq 0 \\ c_{d_r} \cdot \tilde{d}_{o,l}(s_k) & \text{if } K(s_k) < 0 \end{cases} \tag{4.100}$$

where $c_{d_r} \in (0, 1)$.

The variable value of z_r suggests the quadrotor to attain, at each time instant k, an altitude near the local mean value of the bounds on z, given by the relaxed functions $\tilde{z}_{o,l}(s_k)$ and $\tilde{z}_{o,u}(s_k)$. In this way, the quadrotor should avoid more easily the vertical obstacles on its path.

The variable value of d_r, instead, helps the LMPC algorithm in finding the optimal path minimizing the lap time: the quadrotor will tend to stay close to the inner border of the track/obstacle (according to $\tilde{d}_{o,u}(s_k)$) when it is on a straight line or on a left curve; instead, it will tend to stay close to the outer border of the track/obstacle (according to $\tilde{d}_{o,l}(s_k)$) when it is on a right curve. Qualitatively, this is how, in a closed track, a pilot would conduct a vehicle to efficiency "cut the curves" to reach the finish line in the shortest amount of time.

4.9.5. Constraints

The following constraints are added to the LMPC optimization problem with obstacle avoidance:

- ATV model equations constraints (implicit prediction form);
- track boundaries, i.e. bounds on the lateral distance d (including obstacles);
- bounds on the altitude z (including obstacles);
- the constraints required by the relaxed LMPC optimization problem (second version) (4.68).

The constraints will be expressed as follows:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{A}_k^j \boldsymbol{x}_{t|k} + \boldsymbol{B}_k^j \boldsymbol{u}_{t|k} + \boldsymbol{c}_k^j, \quad t = 0, 1, ..., N-1 \tag{4.101a}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.101b}$$

$$\tilde{d}_{o,l}(s_k^j) - e_1 \leq d_{t|k} \leq \tilde{d}_{o,u}(s_k^j) + e_1, \quad t = 0, 1, ..., N-1 \tag{4.101c}$$

$$\tilde{z}_{o,l}(s_k^j) - e_2 \leq z_{t|k} \leq \tilde{z}_{o,u}(s_k^j) + e_2, \quad t = 0, 1, ..., N-1 \tag{4.101d}$$

$$\boldsymbol{A}_{CS}^{j-1} \boldsymbol{x}_{N|k} \leq \boldsymbol{b}_{CS}^{j-1} + e_3 \tag{4.101e}$$

$$c \geq \boldsymbol{a}_s^{j-1^T} \boldsymbol{x}_{N|k} + b_s^{j-1}, \quad s = 1, ..., K \tag{4.101f}$$

$$e_1 \geq 0 \tag{4.101g}$$

$$e_2 \geq 0 \tag{4.101h}$$

With respect to standard LMPC, only constraints 4.101c and 4.101d.

Constraint (4.101c) is associated to the track boundaries/width; thus, it sets lower and upper bounds on the lateral distance d of the quadrotor, also taking into account the horizontal obstacles, through the relaxed functions $\tilde{d}_{o,l}(s)$ and $\tilde{d}_{o,u}(s)$. The slack variable e_1 softens the constraint; this allows the quadrotor to slightly go outside the bounds, preventing possible infeasibility when the quadrotor approaches too closely the track borders.

Constraint (4.101d) sets lower and upper bounds on the altitude z of the quadrotor, also taking into account the vertical obstacles, through the relaxed functions $\tilde{z}_{o,l}(s)$ and $\tilde{z}_{o,u}(s)$. The slack variable e_2 softens the constraint; this allows the quadrotor to slightly go outside the bounds, preventing possible infeasibility when the quadrotor approaches too closely the vertical borders.

4.9.6. Optimization problem

We can now write the LMPC optimization problem for quadrotor optimal path planning and obstacle avoidance:

LMPC optimization problem for quadrotor optimal path planning and obstacle avoidance

$$(\boldsymbol{X}_k^{j^*}, \boldsymbol{U}_k^{j^*}, e_1^{j^*}, e_2^{j^*}, e_3^{j^*}, c^{j^*}) = \underset{\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c}{\arg\min} J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c)$$

$$
\begin{aligned}
&J_{LMPC}(\boldsymbol{X}_k, \boldsymbol{U}_k, e_1, e_2, e_3, c) = \\
&= \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F)^T \boldsymbol{P} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_F) + \\
&+ \sum_{t=0}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r)^T \boldsymbol{Q} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_r) + \boldsymbol{u}_{t|k}^T \boldsymbol{R} \boldsymbol{u}_{t|k} + \\
&+ \sum_{t=1}^{N-1} (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k})^T \boldsymbol{Q}_\Delta (\boldsymbol{x}_{t|k} - \boldsymbol{x}_{t-1|k}) + (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k})^T \boldsymbol{R}_\Delta (\boldsymbol{u}_{t|k} - \boldsymbol{u}_{t-1|k}) + \\
&+ K_1 e_1^2 + K_2 e_2^2 + K_3 e_3^2 + c
\end{aligned} \tag{4.102a}
$$

subject to:

$$\boldsymbol{x}_{t+1|k} = \boldsymbol{A}_k^j \boldsymbol{x}_{t|k} + \boldsymbol{B}_k^j \boldsymbol{u}_{t|k} + \boldsymbol{c}_k^j, \quad t = 0, 1, ..., N-1 \tag{4.102b}$$

$$\boldsymbol{x}_{0|k} = \boldsymbol{x}_k^j \tag{4.102c}$$

$$\tilde{d}_{o,l}(s_k^j) - e_1 \leq d_{t|k} \leq \tilde{d}_{o,u}(s_k^j) + e_1, \quad t = 0, 1, ..., N-1 \tag{4.102d}$$

$$\tilde{z}_{o,l}(s_k^j) - e_2 \leq z_{t|k} \leq \tilde{z}_{o,u}(s_k^j) + e_2, \quad t = 0, 1, ..., N-1 \tag{4.102e}$$

$$\boldsymbol{A}_{CS}^{j-1} \boldsymbol{x}_{N|k} \leq \boldsymbol{b}_{CS}^{j-1} + e_3 \tag{4.102f}$$

$$c \geq \boldsymbol{a}_s^{j-1^T} \boldsymbol{x}_{N|k} + b_s^{j-1}, \quad s = 1, ..., k_{fit} \tag{4.102g}$$

$$e_1 \geq 0 \tag{4.102h}$$

$$e_2 \geq 0 \tag{4.102i}$$

with:

$$\boldsymbol{A}_k^j = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{x}} (\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, \tilde{K}(s_k^j)) \tag{4.102j}$$

$$\boldsymbol{B}_k^j = \frac{\partial \hat{\boldsymbol{f}}_Q}{\partial \boldsymbol{u}} (\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, \tilde{K}(s_k^j)) \tag{4.102k}$$

$$\boldsymbol{c}_k^j = \hat{\boldsymbol{f}}_Q (\boldsymbol{x}_k^j, \boldsymbol{u}_{k-1}^j, \tilde{K}(s_k^j)) - \boldsymbol{A}_k^j \boldsymbol{x}_k^j - \boldsymbol{B}_k^j \boldsymbol{u}_{k-1}^j \tag{4.102l}$$

4.9.7. Algorithm

The complete LMPC algorithm for quadrotor optimal path planning and obstacle avoidance is composed by the optimization problem (4.102) and the receding horizon control law (4.23).

Algorithm 4.4 LMPC for quadrotor optimal path planning and obstacle avoidance

Inputs: $\boldsymbol{x}_S$; N_{iter} (number of LMPC iterations); N (LMPC prediction horizon)
Outputs: $\boldsymbol{X}^0, \boldsymbol{X}^1, ..., \boldsymbol{X}^{N_{iter}}$, $\boldsymbol{U}^0, \boldsymbol{U}^1, ..., \boldsymbol{U}^{N_{iter}}$

(1) Generate the first feasible trajectory $\boldsymbol{X}^0 = (\boldsymbol{x}_0^0, \boldsymbol{x}_1^0, ..., \boldsymbol{x}_{T_0}^0)$ using the MPC algorithm for quadrotor trajectory tracking (Algorithm 3.2), with initial state $\boldsymbol{x}_0^0 = \boldsymbol{x}_S$

(2) Construct the initial sampled safe set $SS^0 = \{\boldsymbol{X}^0\}$

(3) Using Algorithm 4.2, perform the convex piecewise-linear fitting of the data points $SS^0 = \{\boldsymbol{x}_0^0, ..., \boldsymbol{x}_{T_0}^0\}$ and $Q^0(SS^0) = \{J_0^0, ..., J_{T_0}^0\}$, obtaining the coefficients $\boldsymbol{\alpha}^0 = (\boldsymbol{a}_s^0, b_s^0)_{s=1}^{k_{fit}}$ of the max-affine terminal cost barycentric function $\tilde{P}^0(\boldsymbol{x})$

(4) **For** $j = 1, 2, ..., N_{iter}$:

(4.1) Set the initial state $\boldsymbol{x}_0^j = \boldsymbol{x}_{T_{j-1}}^{j-1} - L_{track}\, \boldsymbol{e}_{11} - 2\pi\, \boldsymbol{e}_{13}$, as in (4.86), and store it in $\boldsymbol{X}^j$ (repetitive LMPC)

(4.2) Compute, from SS^{j-1}, the convex safe set CS^{j-1} and its related linear constraints $\boldsymbol{A}_{CS}^{j-1}$, $\boldsymbol{b}_{CS}^{j-1}$

(4.3) **For** $k = 0, 1, ...$:

(4.3.1) • **If** the exit condition on $\boldsymbol{x}_k^j$ is satisfied, **break** the cycle (at $k = T_j$);
• otherwise, continue

(4.3.2) Initialize the LMPC optimization problem (4.102) with $\boldsymbol{x}_k^j$, $\boldsymbol{u}_{k-1}^j$ (**if** $k = 0$, set $\boldsymbol{u}_{k-1}^j \doteq \boldsymbol{0}$), $\boldsymbol{A}_{CS}^{j-1}$, $\boldsymbol{b}_{CS}^{j-1}$, and $\boldsymbol{\alpha}^0$

(4.3.3) Solve the LMPC optimization problem, obtaining the optimal predicted state trajectory $\boldsymbol{X}_k^{j^*} = (\boldsymbol{x}_{0|k}^{j^*}, \boldsymbol{x}_{1|k}^{j^*}, ..., \boldsymbol{x}_{N|k}^{j^*})$ and optimal predicted input sequence $\boldsymbol{U}_k^{j^*} = (\boldsymbol{u}_{0|k}^{j^*}, \boldsymbol{u}_{1|k}^{j^*}, ..., \boldsymbol{u}_{N-1|k}^{j^*})$

(4.3.4) Apply the input $\boldsymbol{u}_k^j \doteq \boldsymbol{u}_{0|k}^{j^*}$ to the system (2.94), as in (4.23), obtaining the next state $\boldsymbol{x}_{k+1}^j$

(4.3.5) Store $\boldsymbol{x}_{k+1}^j$ and $\boldsymbol{u}_k^j$ in $\boldsymbol{X}^j$ and $\boldsymbol{U}^j$

(4.4) Augment the sampled safe set with the new trajectory $\boldsymbol{X}^j = (\boldsymbol{x}_0^j, \boldsymbol{x}_1^j, ..., \boldsymbol{x}_{T_j}^j)$:

$$SS^j = SS^{j-1} \cup \{\boldsymbol{X}^j\}$$

(5) **Return** all the generated trajectories and input sequences:

$$\boldsymbol{X}^0, \boldsymbol{X}^1, ..., \boldsymbol{X}^{N_{iter}}$$
$$\boldsymbol{U}^0, \boldsymbol{U}^1, ..., \boldsymbol{U}^{N_{iter}}$$

The exit condition, as for standard LMPC, is:

$$s_k \overset{?}{\geq} L_{track} \tag{4.103}$$

5

Simulations and results

5.1. Introduction

In this chapter, we report the results of the simulations that have been conducted to test the MPC algorithm for quadrotor trajectory tracking and the LMPC algorithms for quadrotor optimal path planning and obstacle avoidance.
All of these algorithms are employed in software-in-the-loop simulations, in which they are used to control the same quadrotor dynamic model, on a certain number of different race tracks and in various conditions (including the addition of obstacles within the track).

5.1.1. Software implementation

All the algorithms presented in Chapters 3 and 4 have been implemented in MATLAB®.

For what concerns the MATLAB implementation of the MPC and LMPC optimization problems, it has been used *YALMIP* (which stands for "Yet Another Linear Matrix Inequalities Parser"), which is a third-party MATLAB toolbox that provides custom syntax and commands to formulate OPs in a very straightforward way: using YALMIP syntax, the user can define symbolic optimization variables, to be used to write the mathe-

matical expressions of cost functions and constraints; YALMIP, then, uses an internal parser to convert the optimization problem, formulated by the user in YALMIP syntax, in low-level code to be provided to the chosen solver.

YALMIP has also the possibility to be integrated with many external solvers, the majority of which can be installed and interfaced through another third-party toolbox, called *OPTI*.

The YALMIP and OPTI toolboxes are free and available online, together with references and tutorials:

- YALMIP: `yalmip.github.io`
- OPTI: `github.com/jonathancurrie/OPTI`

The full MATLAB code implementing all the algorithms (MPC, LMPC, and LMPC with obstacle avoidance) is available in the following GitHub repository:

`github.com/lorenzocalogero/LMPC_quadrotors`

5.1.2. Quadrotor model

In all simulations, the MPC and LMPC algorithms are used to control the quadrotor dynamic model (2.94); the parameters of this model are set to be always the same in every simulation, so to make every algorithm to virtually control the same quadrotor. For what concerns mass and dimensions, we set up the model parameters considering the common specifications of a racing quadrotor; the moments of inertia I_x, I_y and I_z of the quadrotor have been estimated using (2.31):

Mass: $m = 0.5$ kg
Dimensions: 20 cm $\times$ 20 cm $\times$ 6 cm
Moments of inertia: $I_x = I_y = 6.3 \cdot 10^{-3}$ kg m^2, $I_z = 1.2 \cdot 10^{-2}$ kg m^2

Other model parameters are the following:

Discrete time interval: $T = 0.2$ s
Drag force coefficients: $\beta_x = \beta_y = \beta_z = 0.25$ kg s^{-1}
Acceleration of gravity: $g = 9.81$ m s^{-2}

5.1.3. Race tracks

The race tracks that are used in the simulations are reported below (Figure 5.1).
In Table 5.1 are reported, for each track, the values of L_{track} (track length from start to finish line) and d_{lim} (track width from centerline to border).

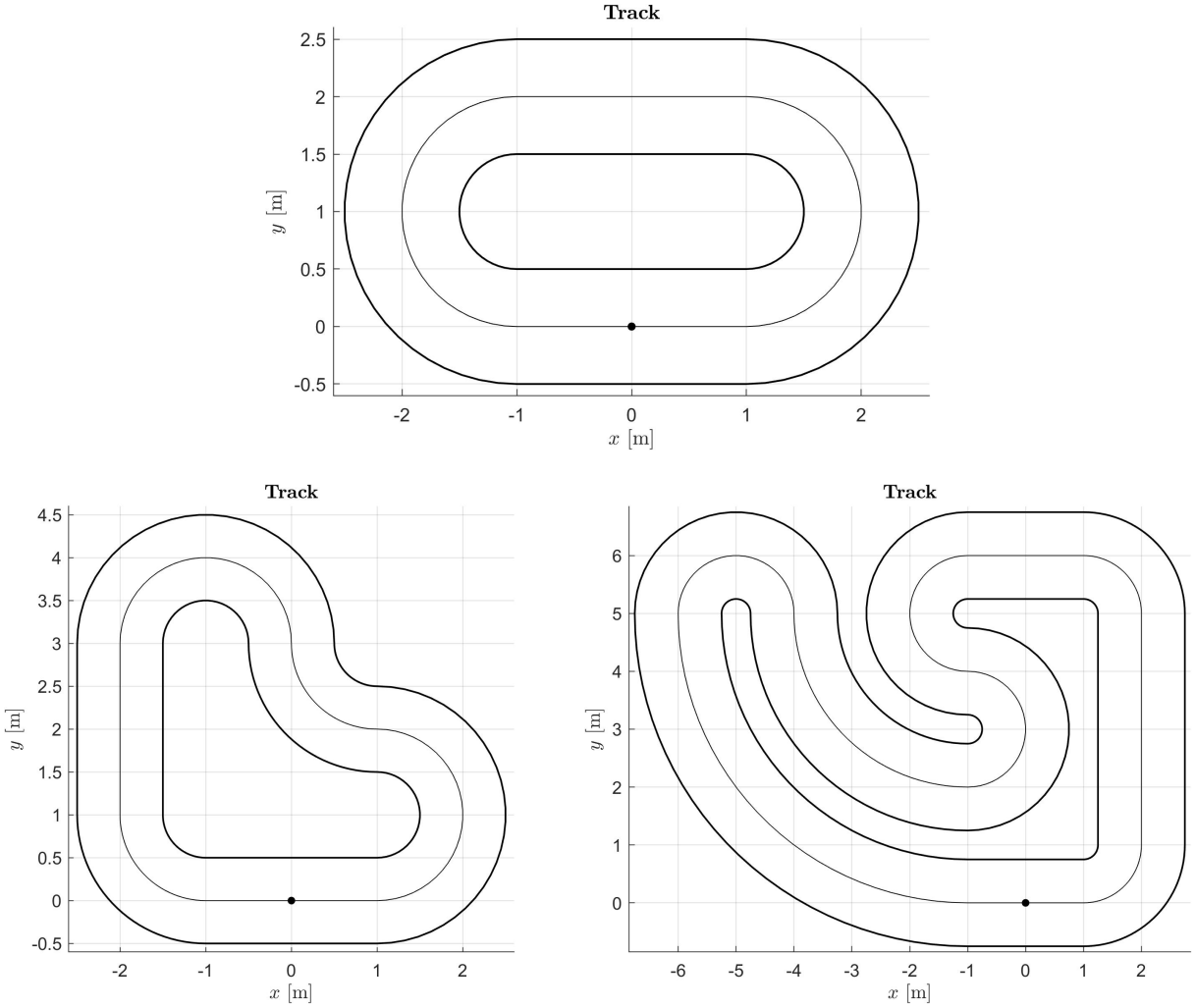


Figure 5.1. Tracks used in the simulations

Table 5.1. Race tracks data

Track	L_{track} [m]	d_{lim} [m]
1	10.2832	0.5
2	13.4248	0.5
3	33.1327	0.75

The curvature function $K(s)$ of each tracks is reported in Figure 3.1.

5.2. MPC simulations

In this section, we report the results of the simulations testing the MPC algorithm for quadrotor trajectory tracking, developed in § 3.7.

Each simulation employs the MPC algorithm to pilot the quadrotor within the race track, performing one of the following tasks:

1) Trajectory tracking with constant lateral distance d, meaning that the reference trajectory has a constant reference value for d:

$$d_r = \text{const.} \tag{5.1}$$

2) Trajectory tracking with oscillating lateral distance d, meaning that the reference trajectory has the following variable reference value for d:

$$d_r(s_k) = c_o \cdot d_{lim} \cdot \sin\left(n_o \cdot 2\pi \frac{s_k}{L_{track}}\right) \tag{5.2}$$

where n_o is the number of oscillations over the track length and $c_o \in [-1, 1]$ is a coefficient for tuning the amplitude and direction of the oscillations.

Six simulations have been carried out:

- Simulation 1: track 1, trajectory tracking with constant lateral distance;
- Simulation 2: track 2, trajectory tracking with constant lateral distance;
- Simulation 3: track 3, trajectory tracking with constant lateral distance;
- Simulation 4: track 1, trajectory tracking with oscillating lateral distance;
- Simulation 5: track 2, trajectory tracking with oscillating lateral distance;
- Simulation 6: track 3, trajectory tracking with oscillating lateral distance.

For each simulation, the following plots are reported:

- the trajectory on the xy plane (i.e. on the track);
- the track real and relaxed curvature functions $K(s)$ and $\tilde{K}(s)$;

- the altitude $z(t)$;
- the Frenet coordinates $s(t)$ and $d(t)$;
- the RPY angles $\phi(t)$, $\theta(t)$, and $\psi(t)$;
- the Cartesian velocities $v_x(t)$, $v_y(t)$, and $v_z(t)$.

5.2.1. Algorithm setup

In the following, we report the values of all the relevant parameters that have been used to set up the MPC algorithm and its related optimization problem in each simulation. If a parameter is reported symbolically, it means that its value depends on the simulation; its numerical value for each simulation is reported in Table 5.2.

Initial state:

$$\boldsymbol{x}_S = \boldsymbol{x}_0 = \begin{pmatrix} 0 & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & 0 & d_0 & 0 \end{pmatrix}$$

Reference state:

$$\boldsymbol{x}_r = \begin{pmatrix} 1\,\mathrm{m} & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & L_{track} & d_r & \star \end{pmatrix}$$

Weight matrices:

$$\begin{aligned}
\boldsymbol{Q} &= \mathrm{diag}\begin{pmatrix} 100 & 1 & 1 & 10 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 1 & 1 \cdot 10^3 & 0 \end{pmatrix} \\
\boldsymbol{R} &= \mathrm{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix} \\
\boldsymbol{Q}_\Delta &= \frac{1}{T^2}\mathrm{diag}\begin{pmatrix} 0 & 0 & 0 & 0 & 100 & 100 & 10 & 0.01 & 0.01 & 100 & 0 & 0 & 0 \end{pmatrix} \\
\boldsymbol{R}_\Delta &= \frac{1}{T^2}\mathrm{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix} \\
K_1 &= 1 \cdot 10^3 \\
K_2 &= 1 \cdot 10^3
\end{aligned}$$

Constraints:

$$\begin{aligned}
z_{(m)} &= 0, & z_{(M)} &= 1.5\ \mathrm{m} \\
v_x^{(m)} &= -0.5/\sqrt{2}\ \mathrm{m\,s^{-1}}, & v_x^{(M)} &= 0.5/\sqrt{2}\ \mathrm{m\,s^{-1}} \\
v_y^{(m)} &= -0.5/\sqrt{2}\ \mathrm{m\,s^{-1}}, & v_y^{(M)} &= 0.5/\sqrt{2}\ \mathrm{m\,s^{-1}}
\end{aligned}$$

$$v_z^{(m)} = -0.5\ \mathrm{m\,s^{-1}}, \qquad v_z^{(M)} = 0.5\ \mathrm{m\,s^{-1}}$$

Other parameters:

Prediction horizon: $N_{MPC} = 10$

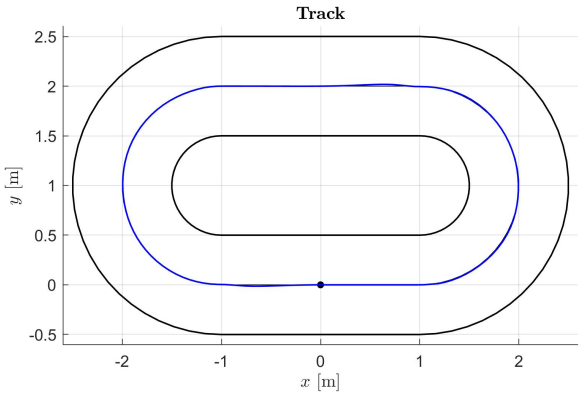
Curvature relaxation coefficient: $K_{rel} = 0.1$

Simulation-dependent parameters:

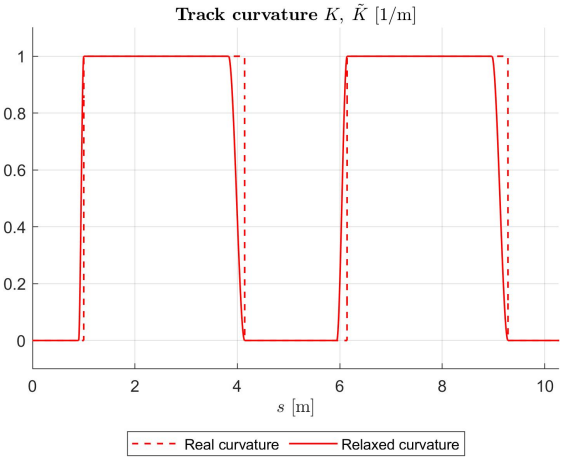
Table 5.2. Simulation-dependent parameters (MPC)

Parameter	**Simulation**					
	1	2	3	4	5	6
d_0 [m]	0	−0.25	0.375	0	0	0
d_r [m]	0	−0.25	0.375	Task 2	Task 2	Task 2
n_o	/	/	/	1	4	4
c_o	/	/	/	0.9	0.5	−0.5
L_{track} [m]	Track 1	Track 2	Track 3	Track 1	Track 2	Track 3
d_{lim} [m]	Track 1	Track 2	Track 3	Track 1	Track 2	Track 3

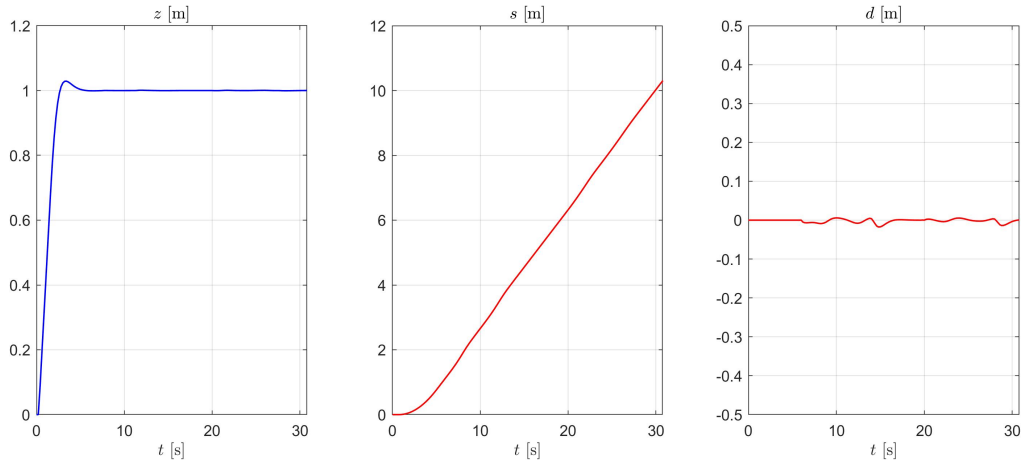
5.2.2. Simulation 1: track 1, constant lateral distance



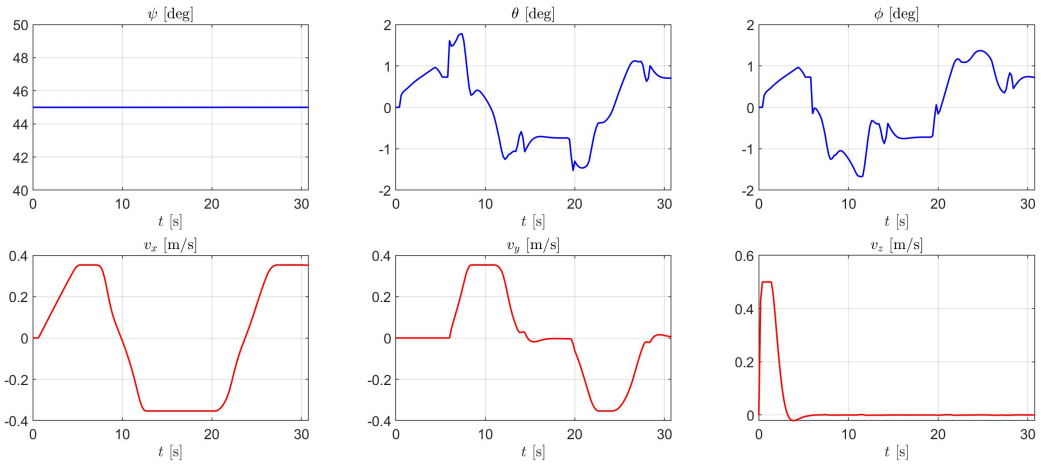
(a) Planar trajectory on the track



(b) Real and relaxed curvature of the track



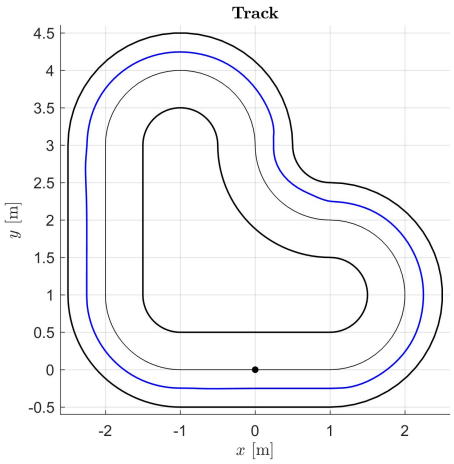
(c) Altitude z and Frenet coordinates (curvilinear abscissa s, lateral distance d)



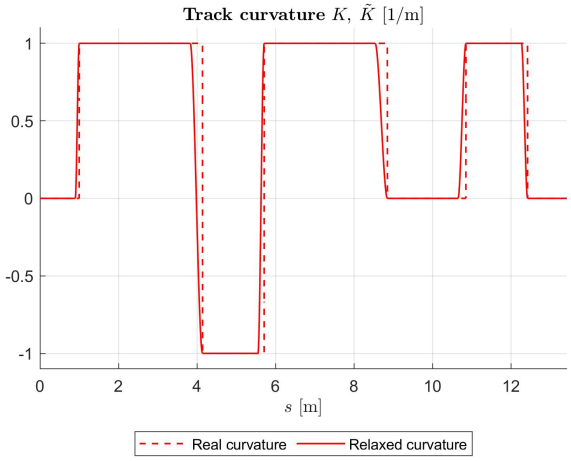
(d) RPY angles ψ, θ, ϕ (first row) and Cartesian velocities v_x, v_y, v_z (second row)

Figure 5.2. Simulation 1: track 1, trajectory tracking with constant lateral distance ($d_r = 0$)

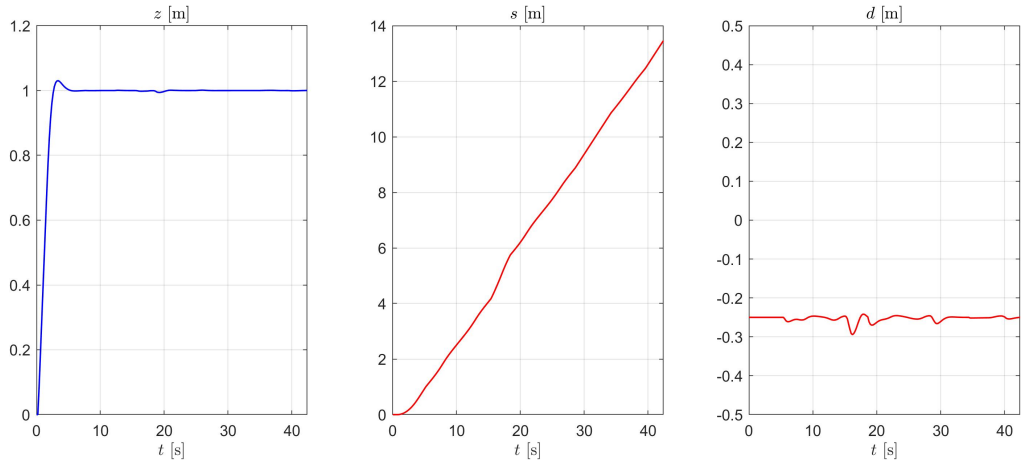
5.2.3. Simulation 2: track 2, constant lateral distance



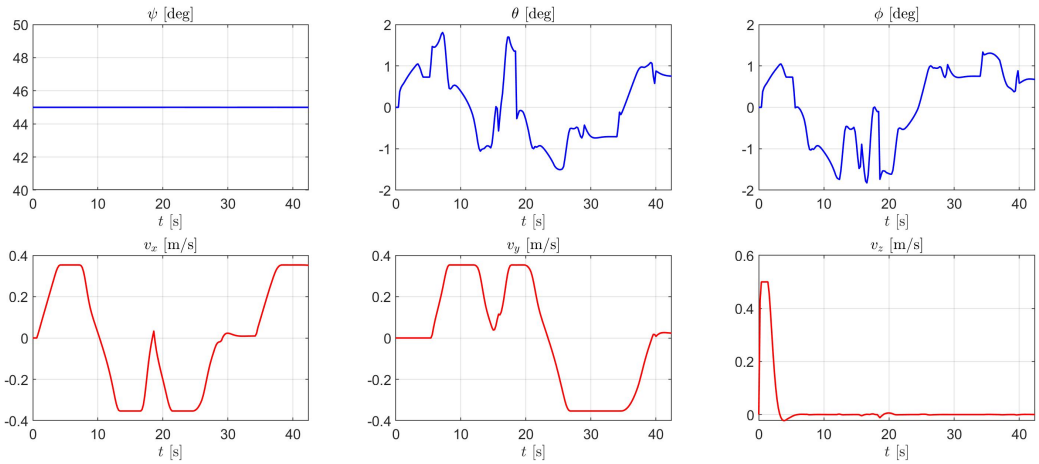
(a) Planar trajectory on the track



(b) Real and relaxed curvature of the track



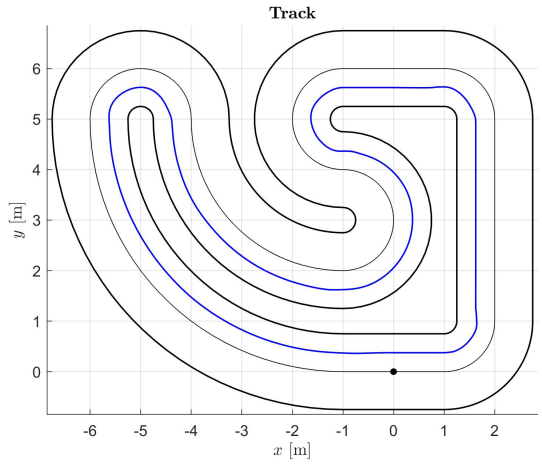
(c) Altitude z and Frenet coordinates (curvilinear abscissa s, lateral distance d)



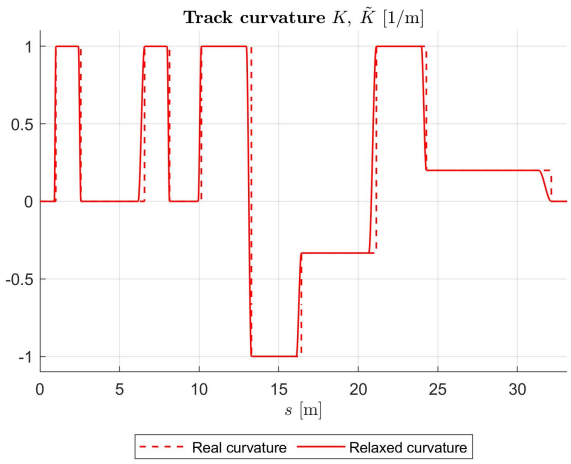
(d) RPY angles ψ, θ, ϕ (first row) and Cartesian velocities v_x, v_y, v_z (second row)

Figure 5.3. Simulation 2: track 2, trajectory tracking with constant lateral distance ($d_r = -0.25$ m)

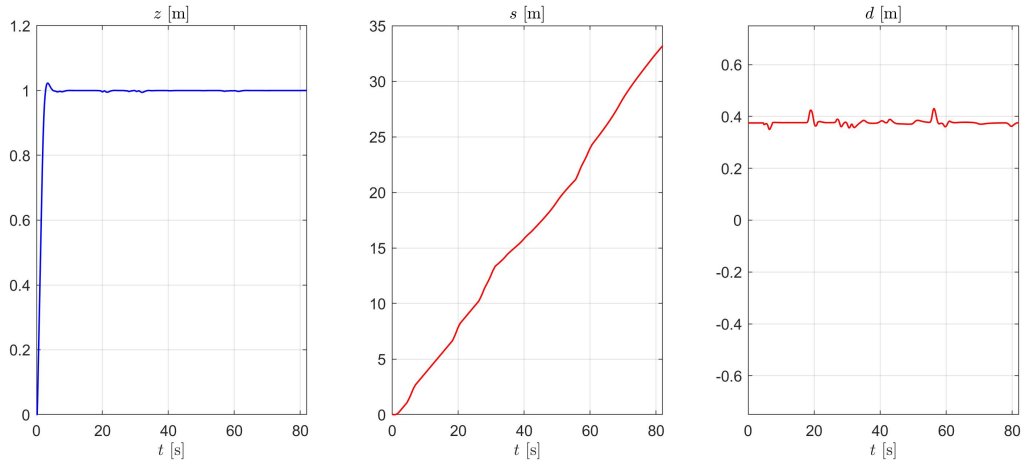
5.2.4. Simulation 3: track 3, constant lateral distance



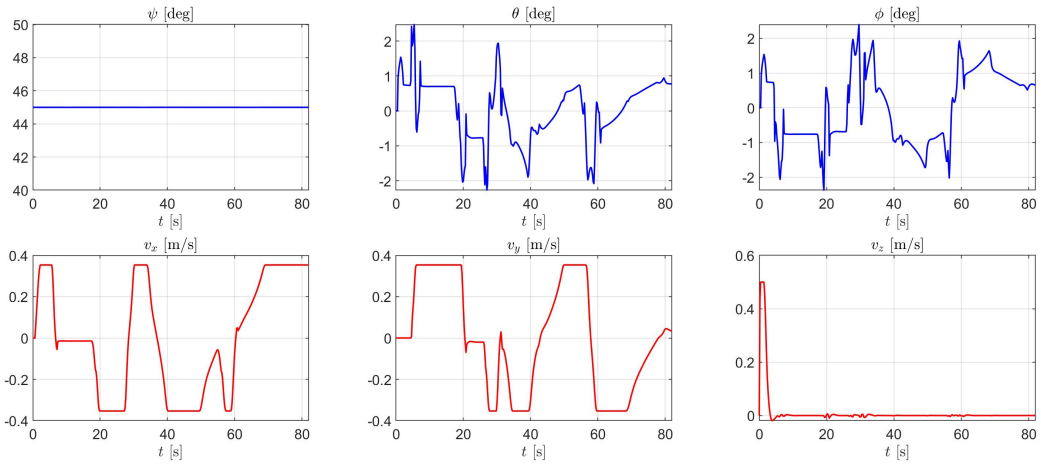
(a) Planar trajectory on the track



(b) Real and relaxed curvature of the track



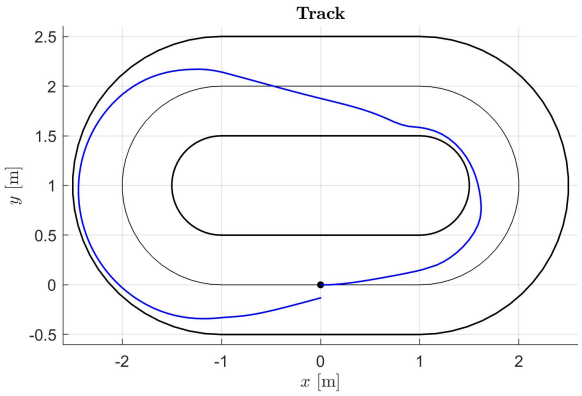
(c) Altitude z and Frenet coordinates (curvilinear abscissa s, lateral distance d)



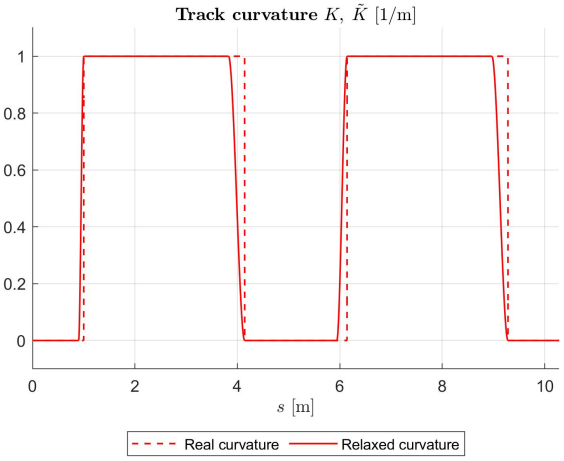
(d) RPY angles ψ, θ, ϕ (first row) and Cartesian velocities v_x, v_y, v_z (second row)

Figure 5.4. Simulation 3: track 3, trajectory tracking with constant lateral distance ($d_r = 0.375$ m)

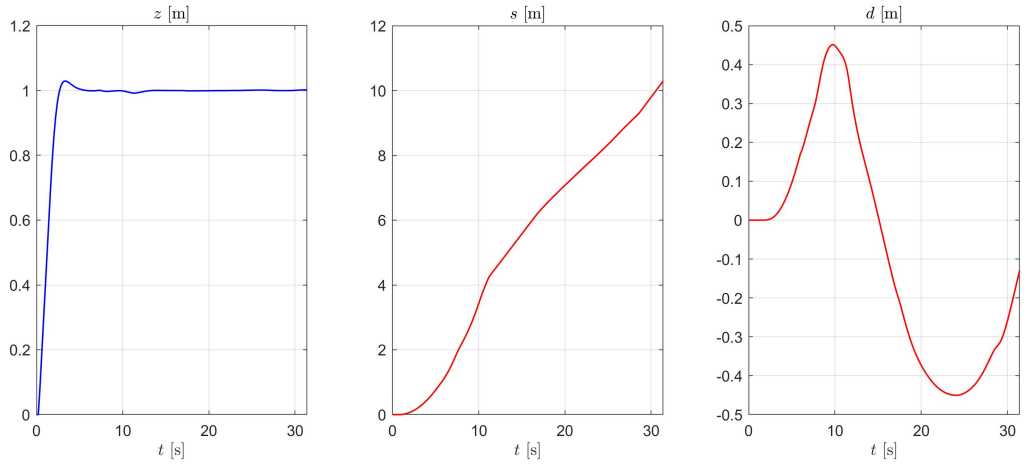
5.2.5. Simulation 4: track 1, oscillating lateral distance



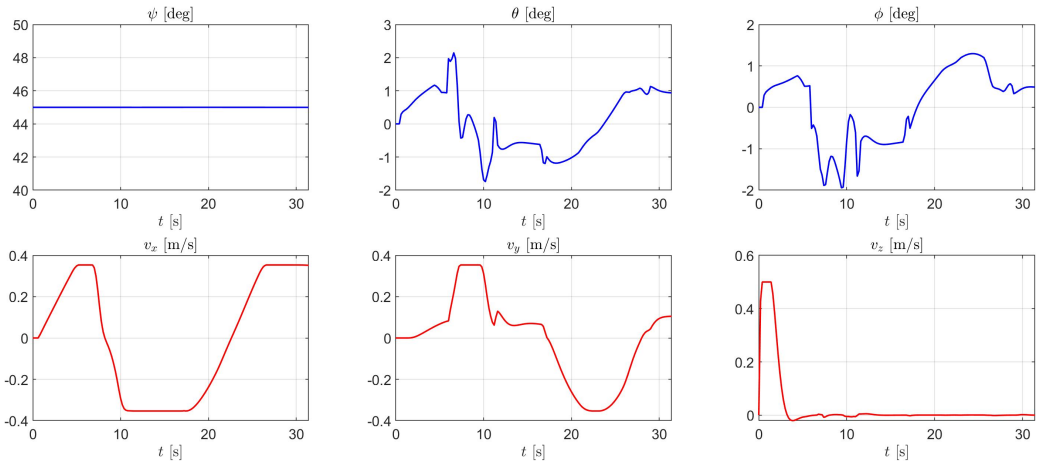
(a) Planar trajectory on the track



(b) Real and relaxed curvature of the track



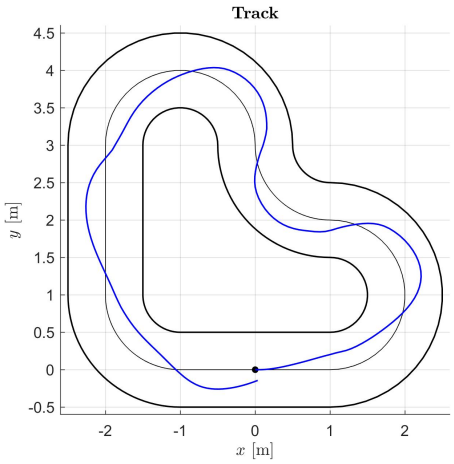
(c) Altitude z and Frenet coordinates (curvilinear abscissa s, lateral distance d)



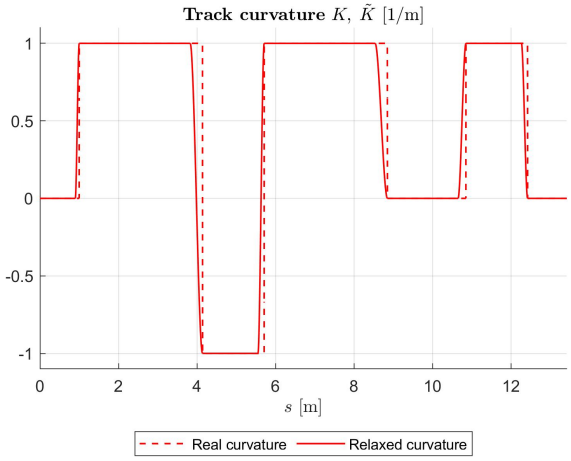
(d) RPY angles ψ, θ, ϕ (first row) and Cartesian velocities v_x, v_y, v_z (second row)

Figure 5.5. Simulation 4: track 1, trajectory tracking with oscillating lateral distance ($n_o = 1$, $c_o = 0.9$)

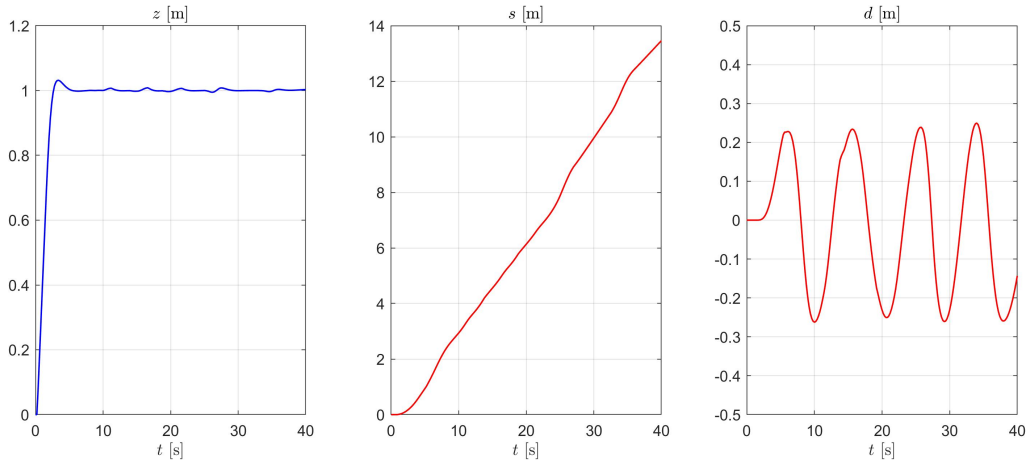
5.2.6. Simulation 5: track 2, oscillating lateral distance



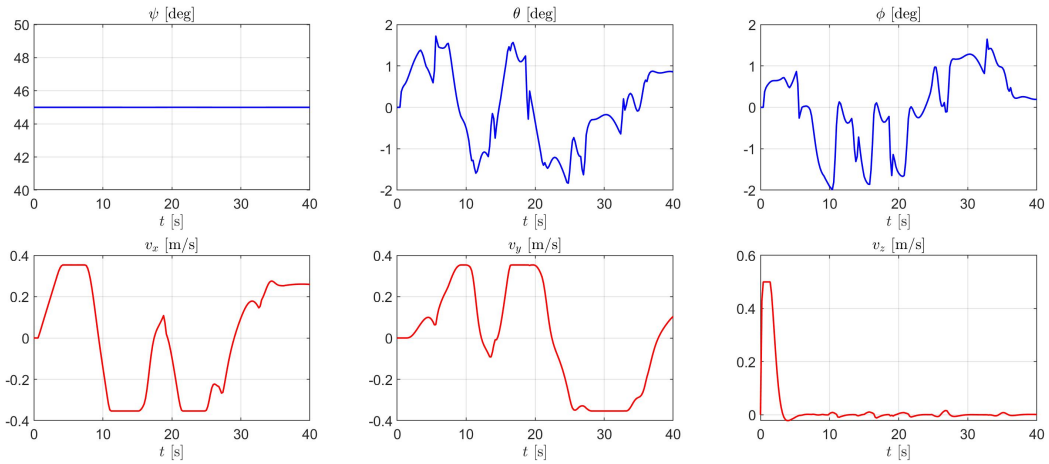
(a) Planar trajectory on the track



(b) Real and relaxed curvature of the track



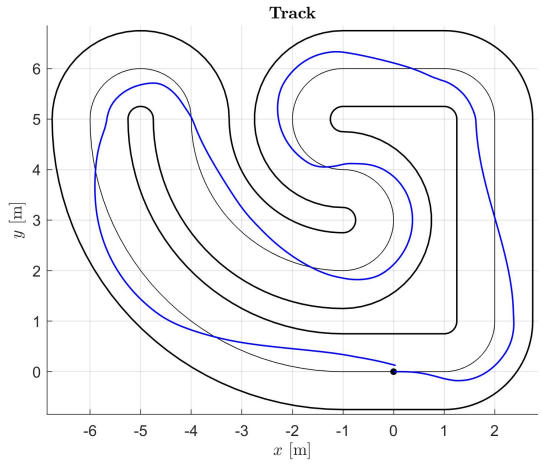
(c) Altitude z and Frenet coordinates (curvilinear abscissa s, lateral distance d)



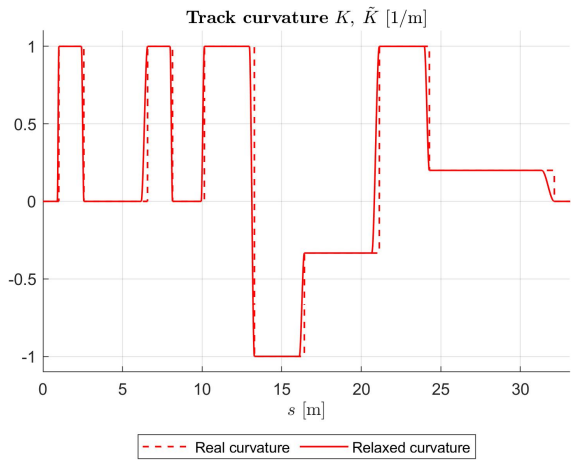
(d) RPY angles ψ, θ, ϕ (first row) and Cartesian velocities v_x, v_y, v_z (second row)

Figure 5.6. Simulation 5: track 2, trajectory tracking with oscillating lateral distance ($n_o = 4$, $c_o = 0.5$)

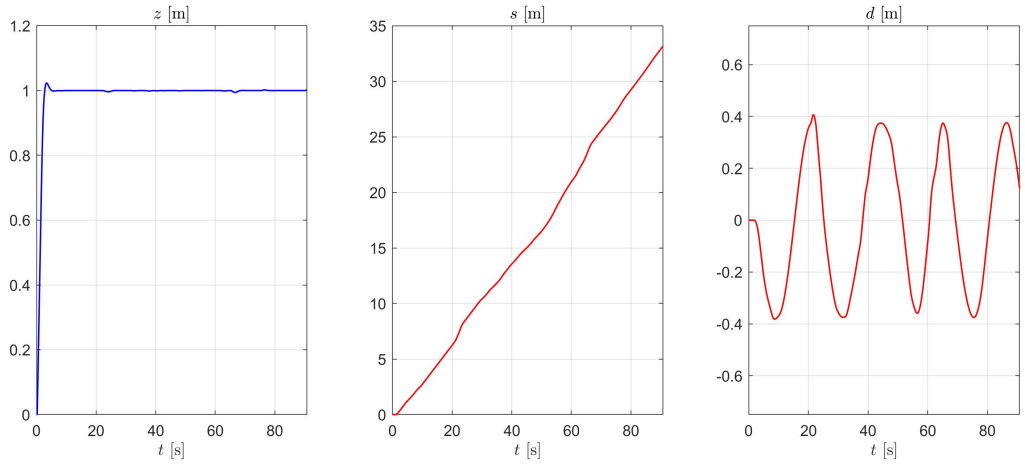
5.2.7. Simulation 6: track 3, oscillating lateral distance



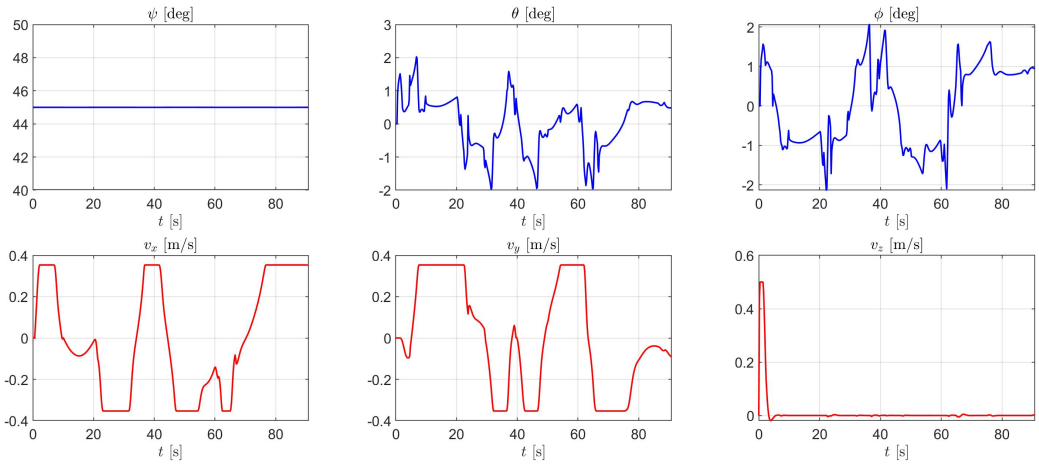
(a) Planar trajectory on the track



(b) Real and relaxed curvature of the track



(c) Altitude z and Frenet coordinates (curvilinear abscissa s, lateral distance d)



(d) RPY angles ψ, θ, ϕ (first row) and Cartesian velocities v_x, v_y, v_z (second row)

Figure 5.7. Simulation 6: track 3, trajectory tracking with oscill. lateral distance ($n_o = 4$, $c_o = -0.5$)

5.3. LMPC simulations

In this section, we report the results of the simulations testing the standard LMPC algorithm for quadrotor optimal path planning (i.e. without obstacle avoidance), developed in § 4.8.

Each simulation employs the LMPC algorithm to pilot the quadrotor within the race track, performing one of the following tasks:

1) Non-repetitive LMPC for optimal path generation;

2) Repetitive LMPC for optimal path generation.

Five simulations have been carried out:

- Simulation 1: track 1, non-repetitive LMPC;
- Simulation 2: track 1, repetitive LMPC;
- Simulation 3: track 1, repetitive LMPC, initialized using a different MPC trajectory wrt simulation 2;
- Simulation 4: track 2, repetitive LMPC;
- Simulation 5: track 3, repetitive LMPC.

For each simulation, the following plots are reported:

- the MPC and LMPC trajectories on the xy plane (i.e. on the track);
- the altitude z of each trajectory, plotted wrt s;
- the iteration cost J_0^j of each j-th trajectory;
- the time required by each trajectory to complete a lap (i.e. the lap time);
- the planar velocity profile of each trajectory, plotted wrt s.

5.3.1. Algorithm setup

In the following, we report the values of all the relevant parameters that have been used to set up the LMPC algorithm and its related optimization problem in each simulation. If a parameter is reported symbolically, it means that its value depends on the simulation; its numerical value for each simulation is reported in Table 5.3.

Initial state:

$$\boldsymbol{x}_S = \begin{pmatrix} 0 & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

MPC reference state:

$$\boldsymbol{x}_r^{(MPC)} = \begin{pmatrix} 1\,\text{m} & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & L_{track} & d_r^{(MPC)}(s_k) & \star \end{pmatrix}$$

$$d_r^{(MPC)}(s_k) = c_o \cdot d_{lim} \cdot \sin\left(n_o \cdot 2\pi \frac{s_k}{L_{track}}\right)$$

LMPC goal state and auxiliary reference state:

$$\boldsymbol{x}_F = \begin{pmatrix} \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & 1.2 \cdot L_{track} & \star & \star \end{pmatrix}$$
$$\boldsymbol{x}_r = \begin{pmatrix} 1\,\text{m} & \star & \star & 45° & \star & \star & \star & \star & \star & \star & \star & d_r(s_k) & \star \end{pmatrix}$$

$$d_r(s_k) = \begin{cases} c_{d_r} \cdot d_{lim} & \text{if } K(s_k) = 0 \\ \text{sign}(K(s_k)) \cdot c_{d_r} \cdot d_{lim} & \text{if } K(s_k) \neq 0 \end{cases}$$

MPC weight matrices:

$$\boldsymbol{Q} = \text{diag}\begin{pmatrix} 100 & 1 & 1 & 10 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 1 & 1 \cdot 10^3 & 0 \end{pmatrix}$$
$$\boldsymbol{R} = \text{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$\boldsymbol{Q}_\Delta = \frac{1}{T^2}\text{diag}\begin{pmatrix} 0 & 0 & 0 & 0 & 100 & 100 & 10 & 0.01 & 0.01 & 100 & 0 & 0 & 0 \end{pmatrix}$$
$$\boldsymbol{R}_\Delta = \frac{1}{T^2}\text{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$K_1 = 1 \cdot 10^3$$
$$K_2 = 1 \cdot 10^3$$

LMPC weight matrices:

$$\boldsymbol{P} = \text{diag}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & P_s & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q} = \mathrm{diag}\begin{pmatrix} 1 \cdot 10^5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \cdot 10^5 & 0 \end{pmatrix}$$
$$\boldsymbol{R} = \mathrm{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$\boldsymbol{Q}_\Delta = \frac{1}{T^2}\mathrm{diag}\begin{pmatrix} 10 & 0.01 & 0.01 & 10 & 5 \cdot 10^4 & 5 \cdot 10^4 & 1 \cdot 10^4 & 0.01 & 0.01 & 1 \cdot 10^3 & 10 & 0.01 & 0 \end{pmatrix}$$
$$\boldsymbol{R}_\Delta = \frac{1}{T^2}\mathrm{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$K_1 = 1 \cdot 10^8$$
$$K_2 = 1 \cdot 10^8$$
$$K_3 = 1 \cdot 10^3$$

MPC constraints:

$$\begin{aligned} z_{(m)} &= 0, & z_{(M)} &= 1.5 \text{ m} \\ v_x^{(m)} &= -0.5/\sqrt{2} \text{ m s}^{-1}, & v_x^{(M)} &= 0.5/\sqrt{2} \text{ m s}^{-1} \\ v_y^{(m)} &= -0.5/\sqrt{2} \text{ m s}^{-1}, & v_y^{(M)} &= 0.5/\sqrt{2} \text{ m s}^{-1} \\ v_z^{(m)} &= -0.5 \text{ m s}^{-1}, & v_z^{(M)} &= 0.5 \text{ m s}^{-1} \end{aligned}$$

LMPC constraints:

$$z_{(m)} = \begin{cases} 0 & \text{if non-repetitive LMPC} \\ 0.5 \text{ m} & \text{if repetitive LMPC} \end{cases}, \qquad z_{(M)} = 1.5 \text{ m}$$

Other parameters:

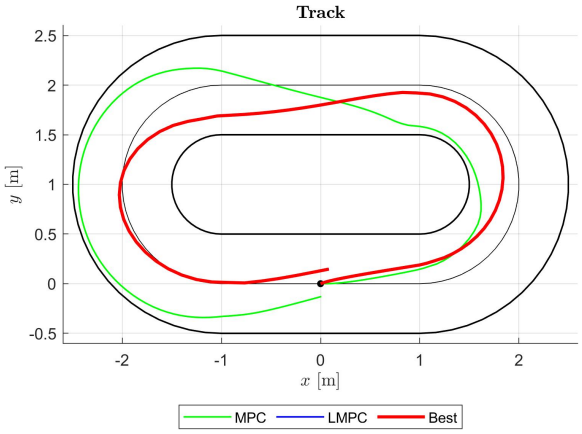
MPC prediction horizon: $N_{MPC} = 10$
LMPC prediction horizon: $N_{LMPC} = 10$
Number of LMPC iterations: N_{iter}
Curvature relaxation coefficient: $K_{rel} = 0.1$

Simulation-dependant parameters:

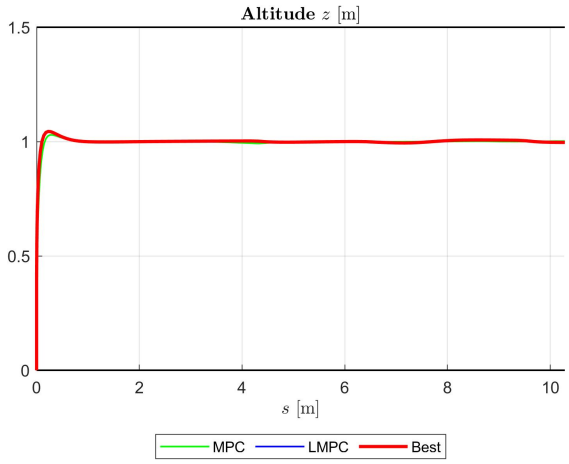
Table 5.3. Simulation-dependent parameters (LMPC)

Parameter	**Simulation**				
	1	2	3	4	5
n_o	1	1	4	4	4
c_o	0.9	0.9	0.5	0.5	0.5
c_{d_r}	0.6	0.6	0.6	0.6	0.4
P_s	15	15	15	10	1
N_{iter}	5	7	7	6	5
L_{track} [m]	Track 1	Track 1	Track 1	Track 2	Track 3
d_{lim} [m]	Track 1	Track 1	Track 1	Track 2	Track 3
$K(s)$ [m^{-1}]	Track 1	Track 1	Track 1	Track 2	Track 3

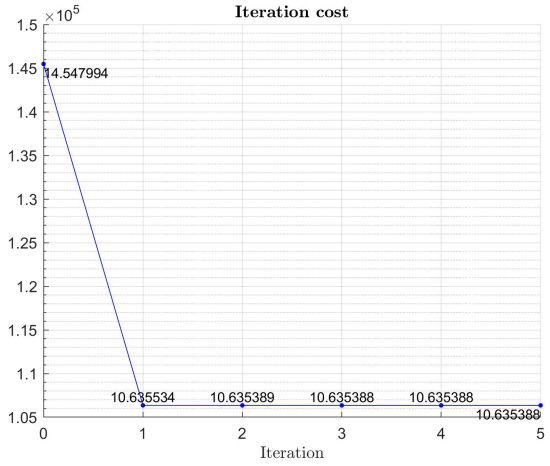
5.3.2. Simulation 1: track 1, non-repetitive LMPC



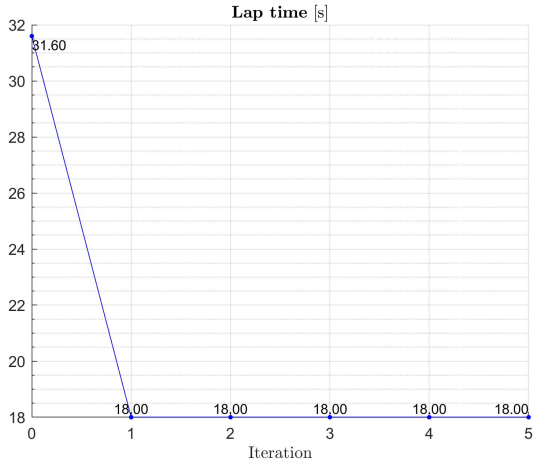
(a) Planar trajectories on the track



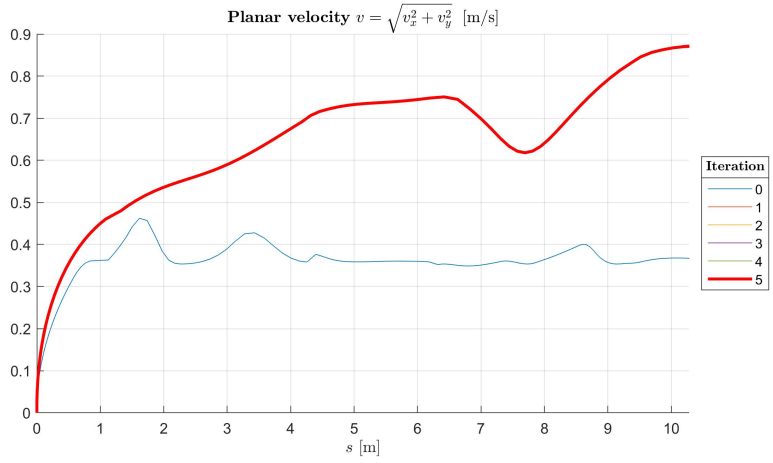
(b) Altitude z



(c) Iteration cost J_0^j



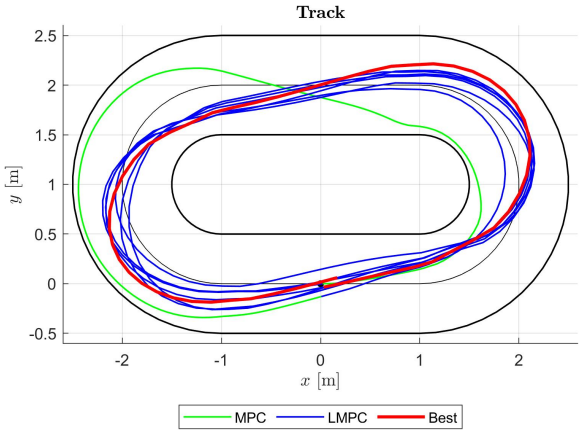
(d) Lap time



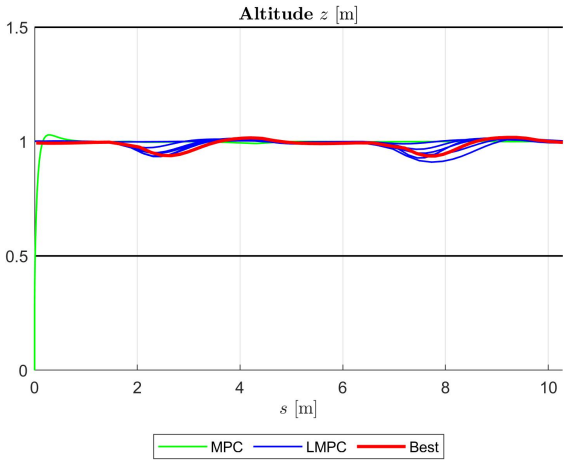
(e) Planar velocity profile $v = \sqrt{v_x^2 + v_y^2}$

Figure 5.8. Simulation 1: track 1, non-repetitive LMPC

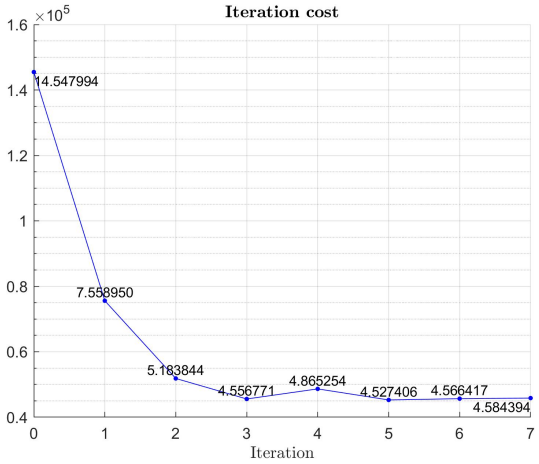
5.3.3. Simulation 2: track 1, repetitive LMPC (1)



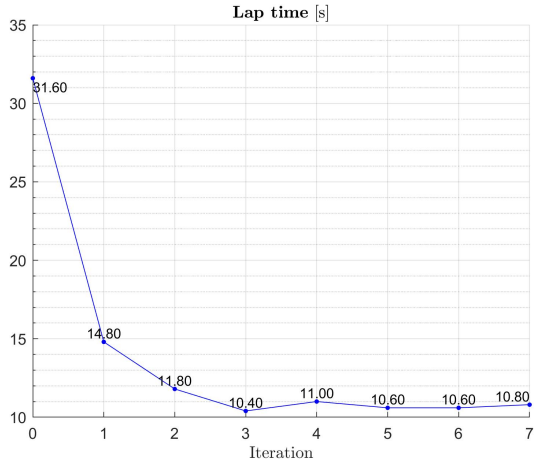
(a) Planar trajectories on the track



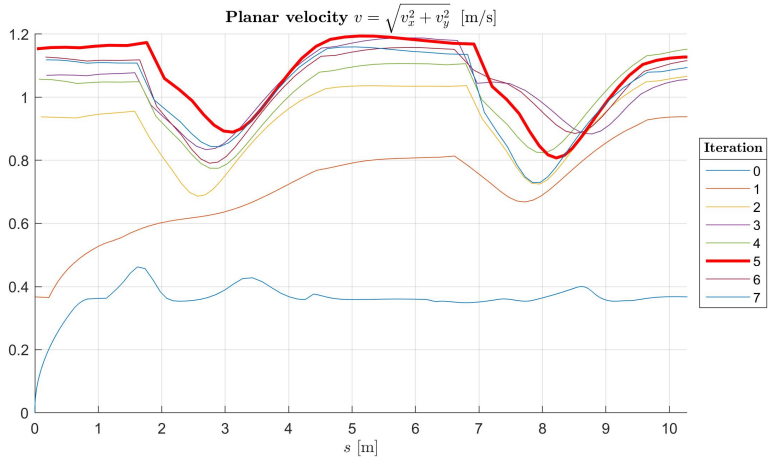
(b) Altitude z



(c) Iteration cost J_0^j



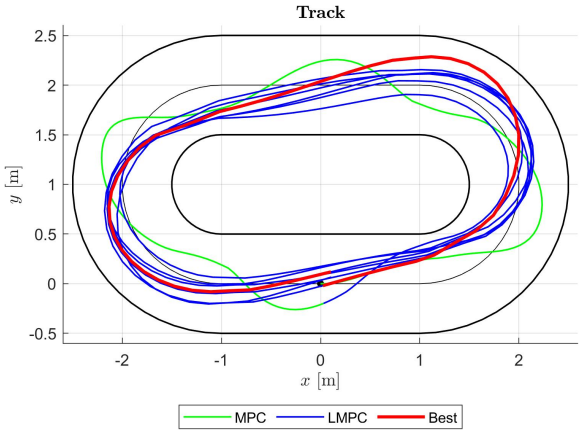
(d) Lap time



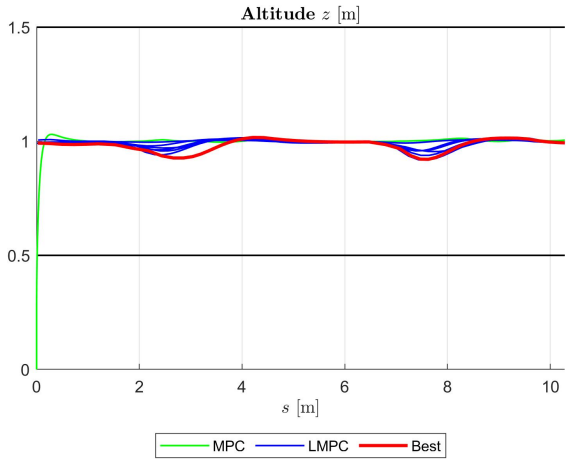
(e) Planar velocity profile $v = \sqrt{v_x^2 + v_y^2}$

Figure 5.9. Simulation 2: track 1, repetitive LMPC

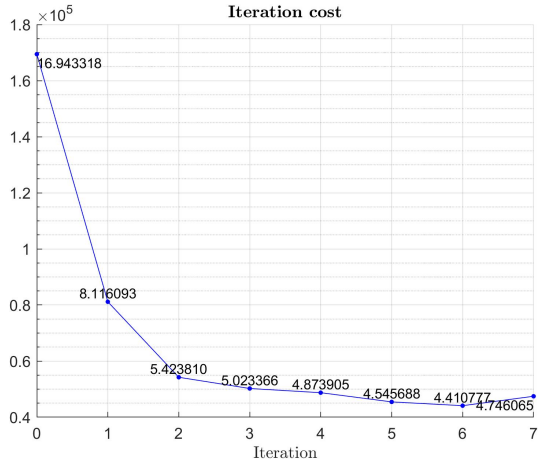
5.3.4. Simulation 3: track 1, repetitive LMPC (2)



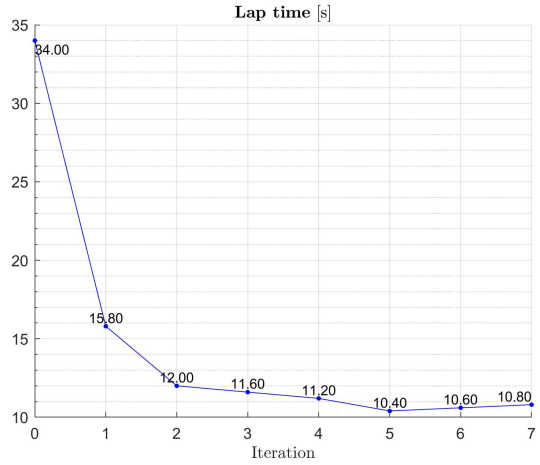
(a) Planar trajectories on the track



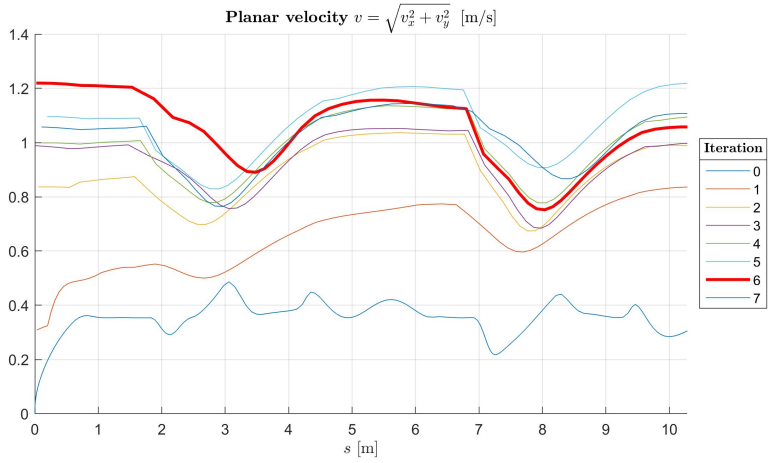
(b) Altitude z



(c) Iteration cost J_0^j



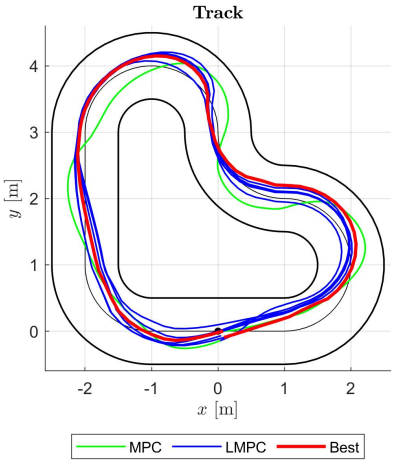
(d) Lap time



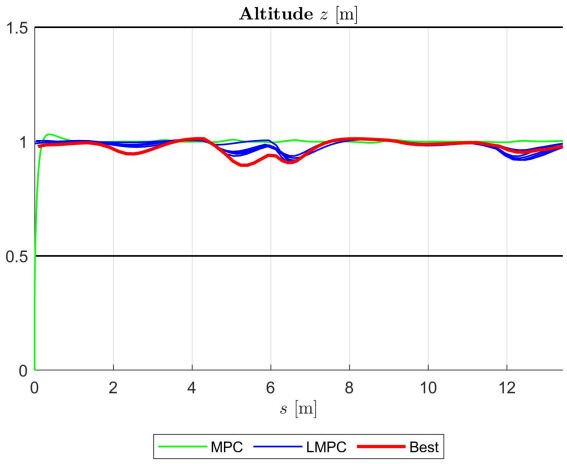
(e) Planar velocity profile $v = \sqrt{v_x^2 + v_y^2}$

Figure 5.10. Simulation 3: track 1, repetitive LMPC, initialized using a different MPC trajectory wrt simulation 2

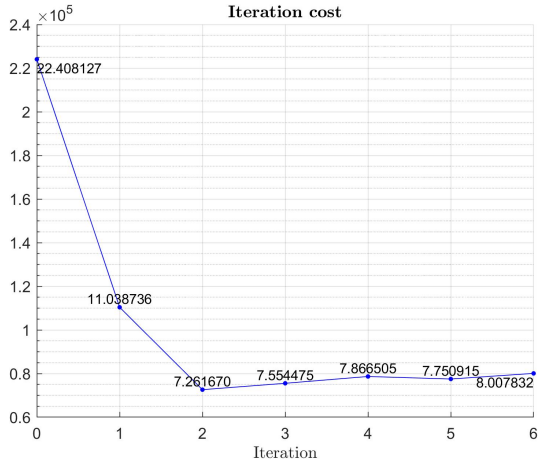
5.3.5. Simulation 4: track 2, repetitive LMPC



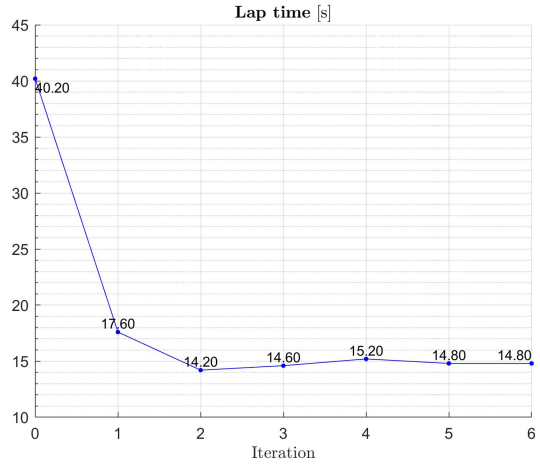
(a) Planar trajectories on the track



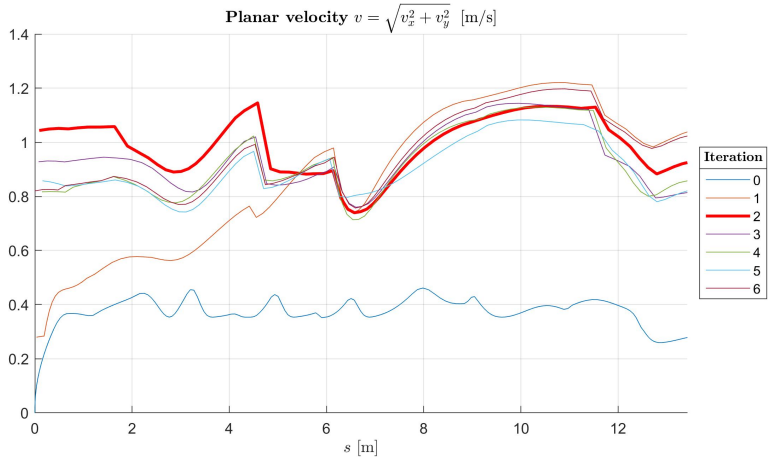
(b) Altitude z



(c) Iteration cost J_0^j



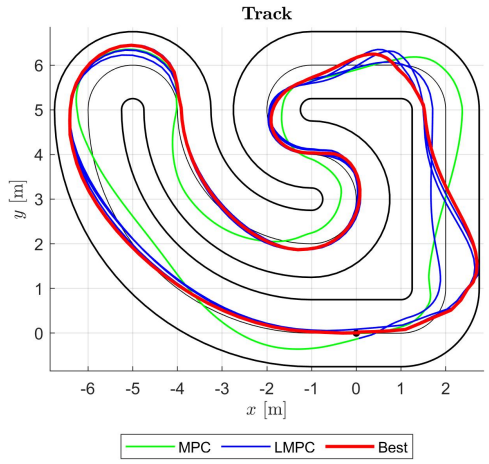
(d) Lap time



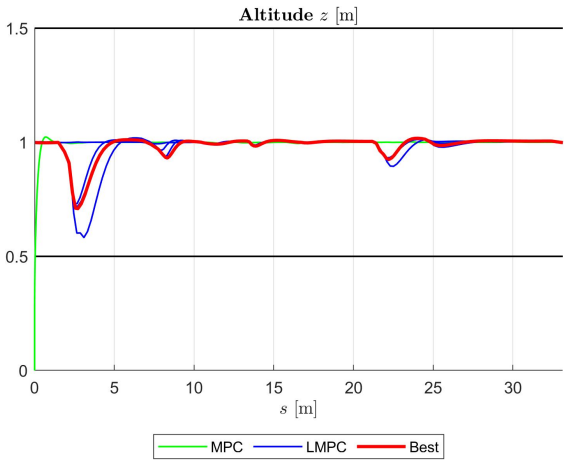
(e) Planar velocity profile $v = \sqrt{v_x^2 + v_y^2}$

Figure 5.11. Simulation 4: track 2, repetitive LMPC

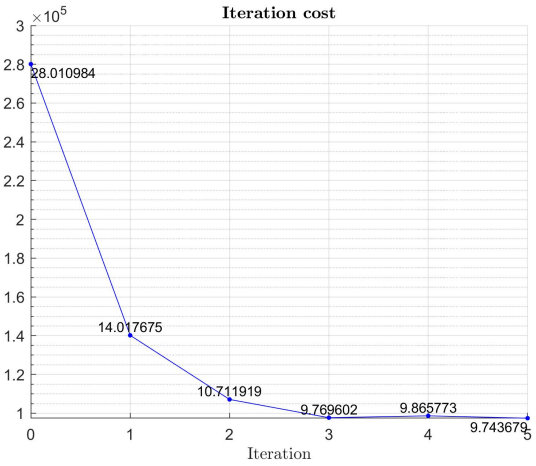
5.3.6. Simulation 5: track 3, repetitive LMPC



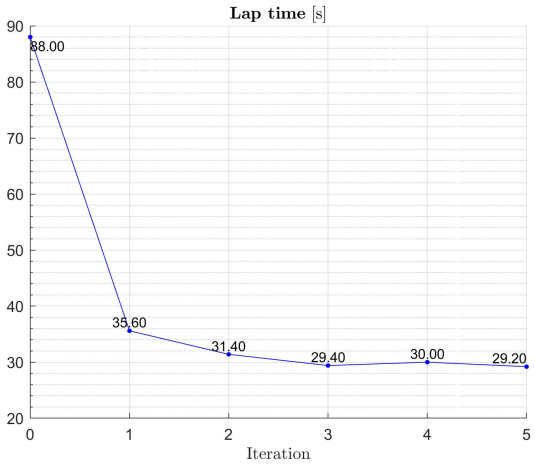
(a) Planar trajectories on the track



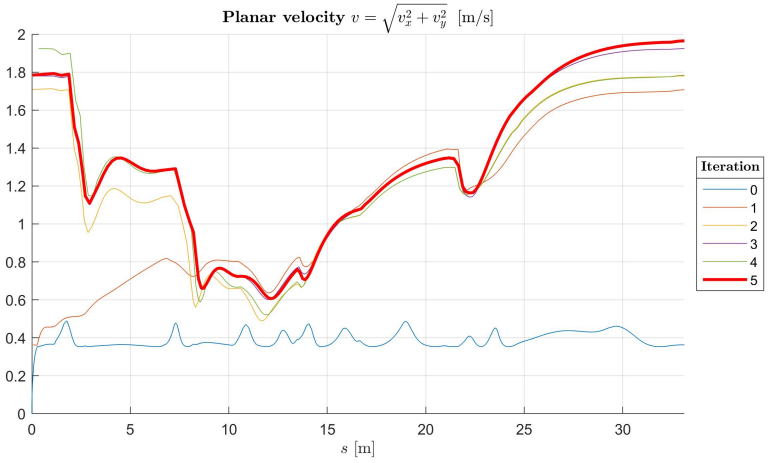
(b) Altitude z



(c) Iteration cost J_0^j



(d) Lap time



(e) Planar velocity profile $v = \sqrt{v_x^2 + v_y^2}$

Figure 5.12. Simulation 5: track 3, repetitive LMPC

5.4. LMPC with obstacle avoidance simulations

In this section, we report the results of the simulations testing the LMPC algorithm for quadrotor optimal path planning and obstacle avoidance, developed in § 4.9.

Each simulation employs the LMPC algorithm with obstacle avoidance to pilot the quadrotor within the race track, performing the following task:

- Repetitive LMPC for optimal path generation and obstacle avoidance.

One simulation has been carried out:

- Simulation 1: track 1, repetitive LMPC with obstacle avoidance, 3 obstacles (1 vertical narrowing, 1 horizontal narrowing, 1 square ring).

For each simulation, the following plots are reported:

- the MPC and LMPC trajectories on the xy plane (i.e. on the track);
- the altitude z of each trajectory, plotted wrt s;
- the iteration cost J_0^j of each j-th trajectory;
- the time required by each trajectory to complete a lap (i.e. the lap time);
- the planar velocity profile of each trajectory, plotted wrt s.

5.4.1. Algorithm setup

In the following, we report the values of all the relevant parameters that have been used to set up the LMPC algorithm with obstacle avoidance and its related optimization problem.

Initial state:

$$\boldsymbol{x}_S = \begin{pmatrix} 0 & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

MPC reference state:

$$\boldsymbol{x}_r^{(MPC)} = \begin{pmatrix} z_r^{(MPC)}(s_k) & 0 & 0 & 45° & 0 & 0 & 0 & 0 & 0 & 0 & L_{track} & d_r^{(MPC)}(s_k) & \star \end{pmatrix}$$

$$z_r^{(MPC)}(s_k) = \frac{\tilde{z}_{o,u}(s_k) + \tilde{z}_{o,l}(s_k)}{2}$$
$$d_r^{(MPC)}(s_k) = 0.5 \cdot \frac{\tilde{d}_{o,u}(s_k) - \tilde{d}_{o,l}(s_k)}{2} \cdot \sin\left(4 \cdot 2\pi \frac{s_k}{L_{track}}\right) + \frac{\tilde{d}_{o,u}(s_k) + \tilde{d}_{o,l}(s_k)}{2}$$

LMPC goal state and auxiliary reference state:

$$\boldsymbol{x}_F = \begin{pmatrix} \star & \star & \star & \star & \star & \star & \star & \star & \star & \star & 1.2 \cdot L_{track} & \star & \star \end{pmatrix}$$
$$\boldsymbol{x}_r = \begin{pmatrix} z_r(s_k) & \star & \star & 45° & \star & \star & \star & \star & \star & \star & \star & d_r(s_k) & \star \end{pmatrix}$$

$$z_r(s_k) = \frac{\tilde{z}_{o,u}(s_k) + \tilde{z}_{o,l}(s_k)}{2}$$
$$d_r(s_k) = \begin{cases} 0.6 \cdot \tilde{d}_{o,u}(s_k) & \text{if } K(s_k) \geq 0 \\ 0.6 \cdot \tilde{d}_{o,l}(s_k) & \text{if } K(s_k) < 0 \end{cases}$$

MPC weight matrices:

$$\boldsymbol{Q} = \text{diag}\begin{pmatrix} 100 & 1 & 1 & 10 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 1 & 1 \cdot 10^3 & 0 \end{pmatrix}$$
$$\boldsymbol{R} = \text{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$\boldsymbol{Q}_\Delta = \frac{1}{T^2}\text{diag}\begin{pmatrix} 0 & 0 & 0 & 0 & 100 & 100 & 10 & 0.01 & 0.01 & 100 & 0 & 0 & 0 \end{pmatrix}$$
$$\boldsymbol{R}_\Delta = \frac{1}{T^2}\text{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$K_1 = 1 \cdot 10^3$$
$$K_2 = 1 \cdot 10^3$$

LMPC weight matrices:

$$\boldsymbol{P} = \text{diag}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 \end{pmatrix}$$
$$\boldsymbol{Q} = \text{diag}\begin{pmatrix} 1 \cdot 10^5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \cdot 10^5 & 0 \end{pmatrix}$$
$$\boldsymbol{R} = \text{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$\boldsymbol{Q}_\Delta = \frac{1}{T^2}\text{diag}\begin{pmatrix} 10 & 0.01 & 0.01 & 10 & 5 \cdot 10^4 & 5 \cdot 10^4 & 1 \cdot 10^3 & 0.01 & 0.01 & 1 \cdot 10^3 & 10 & 0.01 & 0 \end{pmatrix}$$
$$\boldsymbol{R}_\Delta = \frac{1}{T^2}\text{diag}\begin{pmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{pmatrix}$$
$$K_1 = 1 \cdot 10^8$$
$$K_2 = 1 \cdot 10^8$$
$$K_3 = 1 \cdot 10^3$$

MPC constraints:

$$
\begin{aligned}
z_{(m)} &= 0, & z_{(M)} &= 1.5\ \mathrm{m} \\
v_x^{(m)} &= -0.5/\sqrt{2}\ \mathrm{m\,s^{-1}}, & v_x^{(M)} &= 0.5/\sqrt{2}\ \mathrm{m\,s^{-1}} \\
v_y^{(m)} &= -0.5/\sqrt{2}\ \mathrm{m\,s^{-1}}, & v_y^{(M)} &= 0.5/\sqrt{2}\ \mathrm{m\,s^{-1}} \\
v_z^{(m)} &= -0.5\ \mathrm{m\,s^{-1}}, & v_z^{(M)} &= 0.5\ \mathrm{m\,s^{-1}}
\end{aligned}
$$

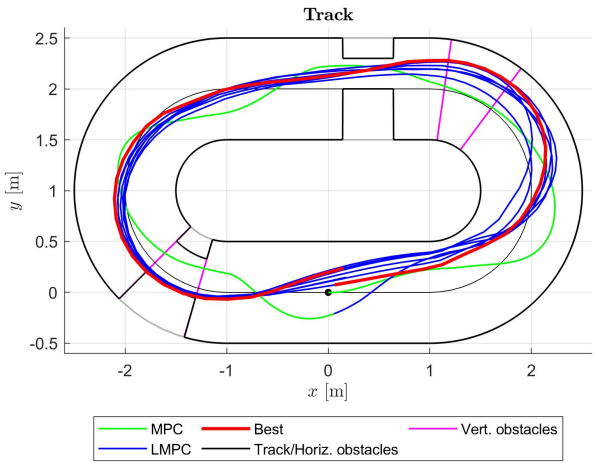
LMPC constraints:

$$
z_{(m)} = 0.5\ \mathrm{m}, \qquad z_{(M)} = 1.5\ \mathrm{m}
$$

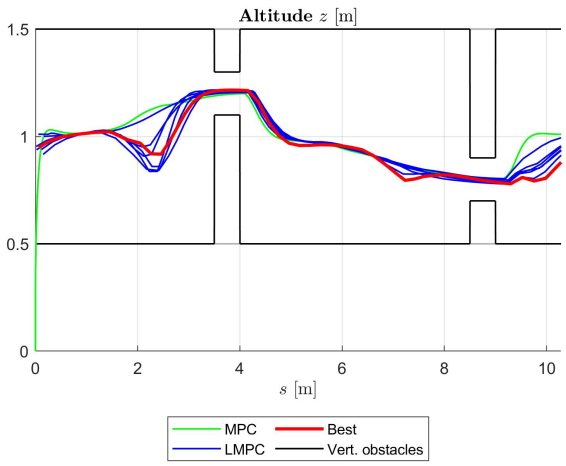
Other parameters:

MPC prediction horizon: $N_{MPC} = 10$
LMPC prediction horizon: $N_{LMPC} = 10$
Number of LMPC iterations: $N_{iter} = 7$
Curvature relaxation coefficient: $K_{rel} = 0.1$
Obstacle relaxation coefficients: $d_{o,rel} = 0.999$, $z_{o,rel} = 0.999$

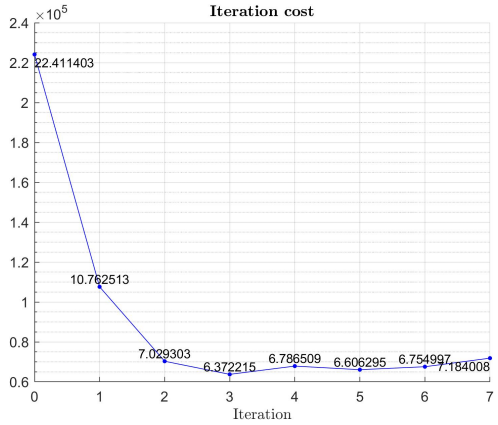
5.4.2. Simulation 1: track 1, repetitive LMPC with obstacle avoidance, 3 obstacles



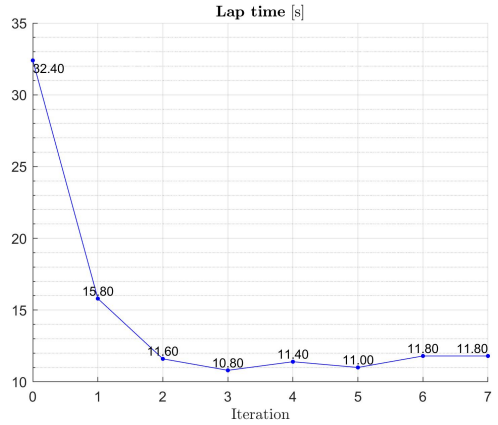
(a) Planar trajectories on the track



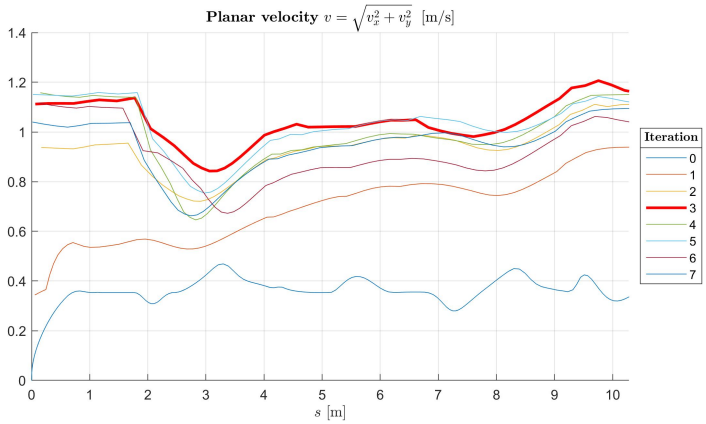
(b) Altitude z



(c) Iteration cost J_0^j



(d) Lap time

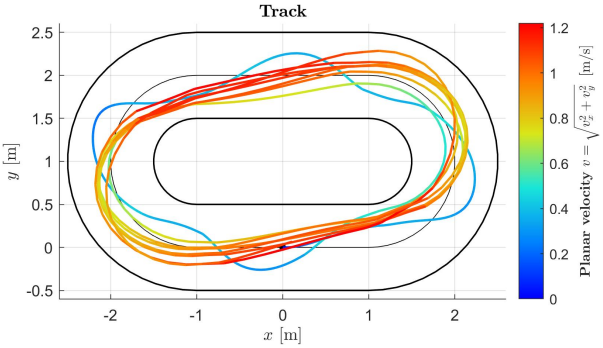


(e) Planar velocity profile $v = \sqrt{v_x^2 + v_y^2}$

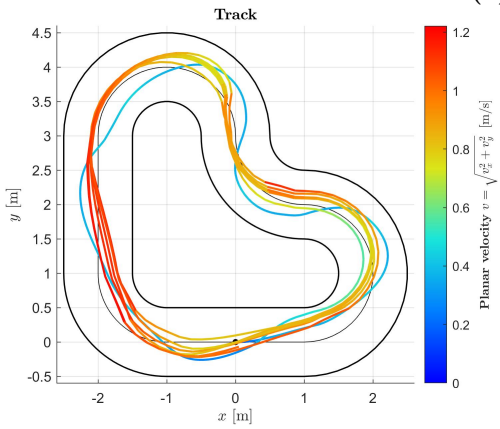
Figure 5.13. Simulation 1: track 1, repetitive LMPC with obstacle avoidance, 3 obstacles (1 vertical narrowing, 1 horizontal narrowing, 1 square ring)

5.5. Additional results

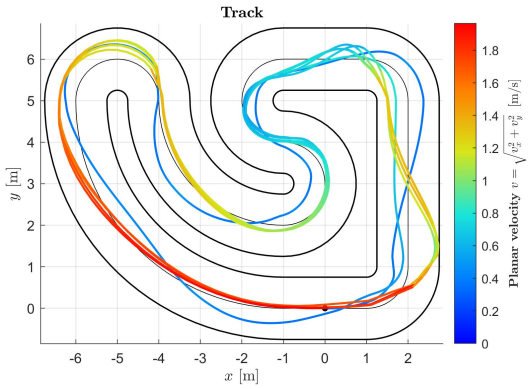
Velocity comparison (LMPC)



(a) Simulation 3



(b) Simulation 4



(c) Simulation 5

Figure 5.14. LMPC algorithm: trajectories overlaid by their coloured velocity profile

Velocity comparison (LMPC with obstacle avoidance)

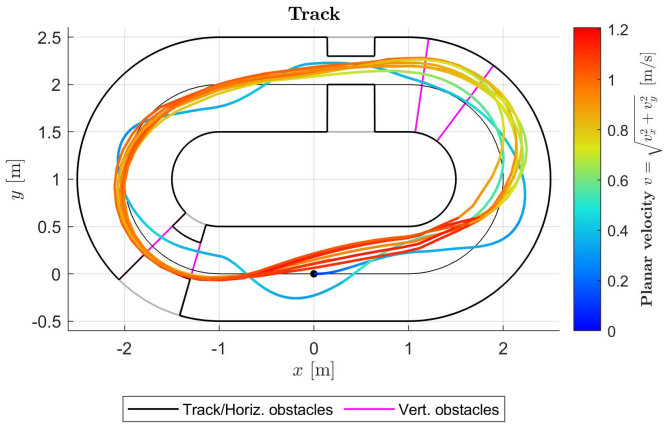


Figure 5.15. LMPC algorithm with obstacle avoidance: trajectories overlaid by their coloured velocity profile

Non-repetitive LMPC: iteration cost at each iteration

In Table 5.4, we report the iteration cost J_0^j at each iteration j with reference to non-repetitive LMPC in simulation 1 (§ 5.3.2).

Table 5.4. Non-repetitive LMPC: iteration cost J_0^j at each iteration j

Iteration (j)	J_0^j
0	$1.4547994418 \cdot 10^5$
1	$1.0635534475 \cdot 10^5$
2	$1.0635388610 \cdot 10^5$
3	$1.0635388432 \cdot 10^5$
4	$1.0635388432 \cdot 10^5$
5	$1.0635388432 \cdot 10^5$

6

Conclusions

In this thesis, it has been developed a control framework for quadrotors based on Learning Model Predictive Control (LMPC).
The LMPC algorithm has been realized to control the quadrotor motion within a closed 3D environment, with a limited height and horizontally delimited by a closed race track. The quadrotor performs repetitive laps of the track, staying within the bounds defined by the vertical and horizontal borders. The states of the generated trajectories and the related costs are stored at every completed lap; with this historical data, the algorithm learns to explore new paths within the track to improve, at every iteration, the cost of the trajectory followed by the quadrotor. Specifically, the cost of each trajectory is quantified as the time needed by it to complete a lap; therefore, the goal of the control algorithm is to obtain the optimal path that minimizes the lap time of the quadrotor.
In addition, the LMPC algorithm has been improved by adding to it the capability of avoiding obstacles placed within the track. In this way, the algorithm achieves both the task of finding the optimal path minimizing the lap time and the task of avoiding possible obstacles within the flight area.

The conducted simulations, that have been reported in Chapter 5, show that the LMPC algorithm successfully achieves the task of finding the optimal path for lap time minimization.
From Figures 5.14a-c, we can see that the control algorithm has learned to fly the

quadrotor aggressively, adopting a flight style that exploits several driving tricks to optimize both the travelled distance and the time needed to complete a lap. Specifically, the quadrotor tends to stay close to the border of the track having locally the shortest length, reducing in this way the local length of the trajectory; it travels diagonally along straight paths to take more easily a possible curve ahead; it accelerates when on a straight path and decelerates when on a curve.

From the simulations, we can also state that the LMPC algorithm successfully achieves the additional task of avoiding possible obstacles within the track. Moreover, from Figure 5.14d, we also see that, even when the quadrotor has to avoid multiple and closely spaced obstacles, the task of finding the optimal path is still achieved: even in presence of obstacles, the quadrotor still adopts an aggressive flight style that optimizes both the travelled distance and the lap time.

For what concerns simulations involving the non-repetitive LMPC algorithm, very meaningful results have been obtained, that not only demonstrate the correct functionality of the algorithm, but also empirically verify the theoretical properties of LMPC, shown in § 4.4:

- In every simulation, the LMPC algorithm is recursively feasible (as in Theorem 4.1) and the goal state $\boldsymbol{x}_F$ proves to be an asymptotically stable equilibrium point of the closed-loop system, since all generated trajectories tend to converge to it (as in Theorem 4.2).
- The iteration cost J_0^j, associated to each generated trajectory, is monotonically non-increasing with the iteration number j (as in Theorem 4.3). This can be noticed by observing Table 5.4, which reports the iteration cost at each iteration, for the simulation involving non-repetitive LMPC (simulation 1, § 5.3.2).

  Specifically, it is interesting to observe how the decrease rate of the iteration cost in Table 5.4 is comparable with the one reported in Table 1 of [16]. Indeed, both in work [16] and in this theses, the LMPC control problem is QP: in [16], the system equations are linear, state and input constraints are linear inequalities, and the LMPC stage cost function is quadratic; in this theses, the system equations are ATV (affine time-variant), state and input constraints are linear inequalities, and the LMPC stage cost function is quadratic. Thus, in both cases, the LMPC problem is QP, making it reasonable to have very similar decrease rates of the iteration cost.
- As the iteration number j increases, the generated trajectories tend to converge to

a steady-state trajectory (as in Theorem 4.4). This steady-state trajectory is the solution of the infinite-horizon optimal control problem associated to the LMPC problem.

Very meaningful results have been obtained also in the simulations involving the repetitive LMPC algorithm (with and without obstacle avoidance):

- In every simulation, the LMPC algorithm is recursively feasible (as in Theorem 4.1) and the goal state $\boldsymbol{x}_F$ proves to be an asymptotically stable equilibrium point of the closed-loop system, since all generated trajectories tend to converge to it (as in Theorem 4.2).
- The iteration cost J_0^j, associated to each generated trajectory, decreases during the first iterations and then settles within a limited range of values.

  The fact that the iteration cost is not perfectly monotonically non-increasing (as stated in Theorem 4.3) can be explained by noticing that here the LMPC algorithm is repetitive (while all the theorems of § 4.4 refer to the non-repetitive case) and it is implemented in its relaxed version.
- As the iteration number j increases, the generated trajectories tend to stay close to a closed steady-state trajectory (as in Theorem 4.4).
- Changing the first feasible trajectory (as in simulations 2 and 3, § 5.3.3 and § 5.3.4), generated through MPC, does not change the final shape of the best trajectory generated by LMPC, which is coherent with Theorem 4.4.

It is interesting to observe how the best trajectories obtained with repetitive LMPC have a lower cost than those obtained with non-repetitive LMPC.
This is reasonable, since, with repetitive LMPC, the final velocity of the quadrotor, when it crosses the finish line, is conserved at the beginning of the next iteration; thus, the initial acceleration phase, that is required in non-repetitive LMPC, here is not needed, meaning that the initial states of the repetitive trajectory will have already a lower cost with respect to those of the non-repetitive one.
Moreover, being the initial condition always different, the algorithm can explore more easily new paths to improve the iteration cost; such paths are hardly attainable by the non-repetitive LMPC, since the quadrotor always starts with null velocity and in the same position.

Finally, being the track a closed circuit, repetitive LMPC is best suited for generating the optimal path, since, typically, the best trajectory is closed as well and features a quasi-continuous transition between its initial and final state (i.e. when crossing the finish line).

6.1. Future developments

To improve even further the performance of the LMPC algorithms and to add additional features, here are listed some possible future works that can be carried out:

- *Alternative relaxation of the LMPC optimization problem.* The LMPC optimization problem can be implemented using the first relaxation approach shown in § 4.6.3. This method uses analytical convex combination to compute both the convex safe set and the terminal cost barycentric function, by including, as additional optimization variables, the vector $\boldsymbol{\lambda}$ of positive weighting scalars associated to the convex combination.

  This relaxation ensures to obtain the best possible continuous approximation $P^j(\boldsymbol{x})$ of the discrete terminal cost function $Q^j(\boldsymbol{x})$. This method, in fact, does not rely on an interpolation procedure (like the second relaxation approach), which is highly dependent on the seed points used to initialize the fitting algorithm (which are chosen randomly) and does not provide good results when the shape of the data points is not sufficiently convex.

  The price of using the first relaxation approach is that the number of involved optimization variables steadily increases with the number of iterations; indeed, $\dim(\boldsymbol{\lambda}) = |SS^j|$, where $|\cdot|$ denotes the set cardinality operator (i.e. the number of elements of the set), meaning that the more trajectories are stored in SS^j, the more $\boldsymbol{\lambda}$ optimization variables will be present in the OP.
  Such higher number of optimization variables may slow down the solver and cause more problems of infeasibility, with the advantage, nonetheless, of obtaining significantly better and reliable performances for the LMPC algorithm.

- *Alternative definition of the track.* We have defined the track as an arbitrary sequence of straight lines and circular curves, of any length and angle, and having limited width and height. Therefore, to describe the track boundaries within the LMPC optimization problem, we have employed Frenet coordinates, which make possible to

define the track borders as simple bound constraints on the state variables d and z, at the price of including the Frenet conversion equations in the system model.

The adoption of Frenet coordinates requires to make a series of precautions and approximations to be able to implement them within the LMPC optimization problem; specifically, these are curvature propagation and relaxation and, for LMPC with obstacle avoidance, the propagation and relaxation of the obstacles functions (which have been treated in § 3.7 and § 4.9).

To avoid using Frenet coordinates, the track can be described in an alternative way. This alternative approach consists in defining several waypoints, within the 3D space, and connecting them with segments; the union of these segments represents the centerline of the track. Around each segment, a generalized cylinder defines the local boundaries of the track; thus, the union of these cylinders represents the whole track borders.
An example of this kind of track is depicted in Figure 6.1:

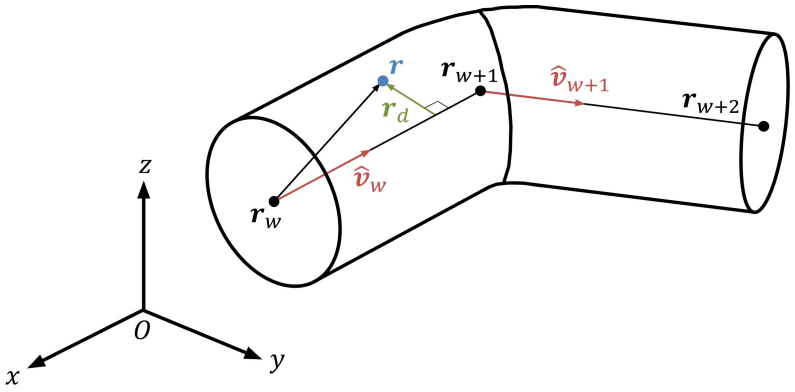


Figure 6.1. Alternative track

To represent the track within the LMPC optimization problem, a single linear inequality constraint is required for each segment.

Specifically, let's denote as $\boldsymbol{r}_w$ a generic waypoint of the track; successive waypoints will be denoted as $\boldsymbol{r}_{w+1}$, $\boldsymbol{r}_{w+2}$, etc.
Denoting as $\boldsymbol{r}$ the position of the quadrotor wrt the fixed Cartesian frame $Oxyz$, the position of the quadrotor from waypoint $\boldsymbol{r}_w$ is equal to $\boldsymbol{r} - \boldsymbol{r}_w$.
The versor defining the direction from waypoint $\boldsymbol{r}_w$ to $\boldsymbol{r}_{w+1}$ is denoted as $\hat{\boldsymbol{v}}_w$ and is equal to $\hat{\boldsymbol{v}}_w = \frac{\boldsymbol{r}_{w+1} - \boldsymbol{r}_w}{|\boldsymbol{r}_{w+1} - \boldsymbol{r}_w|}$.
We finally define the vector $\boldsymbol{r}_d$, which is the difference between $\boldsymbol{r} - \boldsymbol{r}_w$ and the projection of $\boldsymbol{r} - \boldsymbol{r}_w$ onto $\hat{\boldsymbol{v}}_w$. Recalling that the orthogonal projection operator/matrix

associated to a versor $\boldsymbol{u}$ is $\boldsymbol{P} = \boldsymbol{u}\boldsymbol{u}^T$, we can write $\boldsymbol{r}_d$ as:

$$\boldsymbol{r}_d = \left(\boldsymbol{I} - \hat{\boldsymbol{v}}_w \hat{\boldsymbol{v}}_w^T\right)(\boldsymbol{r} - \boldsymbol{r}_w) \tag{6.1}$$

This means that each segment is associated to a bound constraint on $\boldsymbol{r}_d$, which defines the local cross section of the track. Specifically, we can set the constraint as:

$$-\boldsymbol{r}_{lim} \leq \boldsymbol{r}_d \leq \boldsymbol{r}_{lim} \tag{6.2}$$

obtaining a linear inequality constraint; in this case, the cross sections of the track will be rectangles.

Bibliography

[1] M. Abhishek, "Learning Model Predictive Control with Application to Quadcopter Trajectory Tracking," KTH Royal Institute of Technology, 2020.

[2] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. New York: Cambridge University Press, 2017.

[3] L. Brunke, R. Seifried, H. Werner, and F. Borrelli, "Learning Model Predictive Control for Competitive Autonomous Racing," Hamburg University of Technology, 2018.

[4] A. Das, F. Lewis, and K. Subbarao, "Dynamic Inversion with Zero-Dynamics Stabilisation for Quadrotor Control," *IET Control Theory & Applications*, vol. 3, no. 3, pp. 303–314, Mar. 2009.

[5] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control. Theory and Algorithms*, 2nd ed. Switzerland: Springer, 2017.

[6] K. C. Haiyun, "Matrix Calculus," Simon Fraser University, 2014. [Online]. Available: `www.sfu.ca/~haiyunc/notes/matrix_calculus.pdf`

[7] C. James and M. Zeilinger, "Theory in Model Predictive Control: Constraint Satisfaction and Stability," Automatic Control Laboratory, EPFL, 2011. [Online]. Available: `uiam.sk/pc11/data/workshops/mpc/MPC_PC11_Lecture1.pdf`

[8] G. Li, A. Tunchez, and G. Loianno, "Learning Model Predictive Control for Quadrotors," *Proceedings of the 2022 IEEE International Conference on Robotics and Automation*, preprint, May 2022.

[9] T. Luukkonen, "Modelling and Control of Quadcopter," Aalto University, 2011.

[10] J. Löfberg, "Model Predictive Control - Basics," *YALMIP*, Sep. 16, 2016. yalmip.github.io/example/standardmpc

[11] A. Magnani and S. P. Boyd, "Convex Piecewise-Linear Fitting," *Optimization and Engineering*, vol. 10, no. 1, pp. 1–17, Mar. 2008.

[12] P. Morin and C. Samson, "Motion Control of Wheeled Mobile Robots," in *Springer Handbook of Robotics*, Springer, 2008, pp. 799–826.

[13] G. Pannocchia, "Model Predictive Control: Stability and Robustness," University of Pisa, 2012. [Online]. Available: centropiaggio.unipi.it/sites/default/files/course/material/3_MPCcourse_stability-robustness.pdf

[14] G. V. Raffo, "Modelado y Control de un Helicóptero Quadrotor," University of Seville, 2007.

[15] U. Rosolia and F. Borrelli, "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, Jul. 2018.

[16] U. Rosolia and F. Borrelli, "Learning Model Predictive Control for Iterative Tasks: A Computationally Efficient Approach for Linear System," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3142–3147, Jul. 2017.

[17] U. Rosolia and F. Borrelli, "Learning How to Autonomously Race a Car: a Predictive Control Approach," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2713–2719, Nov. 2020.

[18] F. Sabatino, "Quadrotor Control: Modelling, Nonlinear Control Design, and Simulation," KTH Royal Institute of Technology, 2015.

[19] N. Satoshi, "Quadrotor Dynamical Model with Euler-Lagrange Approach," Tokyo Institute of Technology, 2013.

[20] "Euler Angles," *Wikipedia*. en.wikipedia.org/wiki/Euler_angles

[21] "Kinematics," in *Robot Dynamics*, ETH Zurich, 2016. [Online]. Available: ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/documents/RobotDynamics2016/KinematicsSingleBody.pdf