University of West Bohemia
Faculty of Applied Sciences

TRAJECTORY PLANNING AND TRACKING FOR UNMANNED AERIAL VEHICLES

Ing. Zdeněk Bouček

A dissertation submitted for the degree
of Doctor of Philosophy
in Cybernetics

Supervisor: Doc. Ing. Ondřej Straka, Ph.D.
Consultant: Ing. Miroslav Flídr, Ph.D.
Department of Cybernetics

Pilsen, 2024

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

PLÁNOVÁNÍ A SLEDOVÁNÍ TRAJEKTORIÍ PRO BEZPILOTNÍ LETOUNY

Ing. Zdeněk Bouček

disertační práce
k získání akademického titulu doktor
v oboru Kybernetika

Školitel: Doc. Ing. Ondřej Straka, Ph.D.
Konzultant specialista: Ing. Miroslav Flídr, Ph.D.
Katedra kybernetiky

Plzeň, 2024

Declaration

I hereby submit for review and defense my dissertation prepared at the end of my studies at the Faculty of Applied Sciences of the University of West Bohemia in Pilsen.

I hereby declare that I have compiled this work independently, using only the listed resources and literature.

In Pilsen, April 19, 2024

_____

Signature

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem disertační práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 19. 4. 2024

_____

Podpis

Acknowledgment

I would like to express my deepest gratitude to my consultant, Ing. Miroslav Flídr Ph.D., and my supervisor, Doc. Ing. Ondřej Straka Ph.D., for their invaluable guidance throughout my study and research work. Their mentorship has played a pivotal role in shaping the trajectory of my academic journey, and I am truly grateful for their dedication and insightful advice.

I am deeply grateful to the Department of Cybernetics and the Identification and Decision Making (IDM) research group for providing a warm and collaborative environment during my academic journey. The welcoming and friendly environment within the IDM group helped me feel like a valued member of the team. I appreciate this nurturing setting cultivated by all those involved, which facilitated my personal growth and successful completion of this thesis.

I would also like to extend a heartfelt thank you to my beloved wife and children for their love, understanding, and unwavering support. Their presence in my life has been a constant source of motivation and encouragement, and I am thankful for their patience and understanding during the writing of this thesis and throughout my academic journey.

I am truly thankful to my parents for their belief in my abilities and unshakable support. Their encouragement and sacrifices have been the foundation of my achievements, and I am forever indebted to them for their unconditional love and guidance.

Furthermore, I would like to acknowledge the support of the Technology Agency of the Czech Republic, programme National Competence Centres, project # TN 0200 0054 Bozek Vehicle Engineering National Competence Center.

Finally, I would like to express my gratitude for the computational resources provided by the e-INFRA CZ project (ID:90254), which is supported by the Ministry of Education, Youth, and Sports of the Czech Republic.

If one does not know to which port one is sailing, no wind is favorable.

—Seneca

Abstract

This thesis presents the development of a trajectory planning and tracking system for an Unmanned Aerial Vehicle (UAV). It is critical for every application that employs the fully autonomous UAV because the quality of trajectory can significantly affect the UAV mission. The system is based on two methods: a pseudospectral method (PSM) for efficient trajectory planning, and an Interpolating Control (IC) method for trajectory tracking.

In the first part of this thesis, the foundations are laid out, where the behavior of UAVs is described and the methods for path planning and trajectory planning are introduced. The formulation of the trajectory planning problem as an Optimal Control Problem (OCP) and its numerical solution using PSM are emphasized. Further, the trajectory tracking problem is presented and the state-of-the-art method of Model Predictive Control (MPC) is described along with its computationally less demanding alternative IC.

In the second part of the thesis, a system for adaptive PSM-based trajectory planning for UAVs is proposed that leverages the knowledge of the UAV behavior and its optimal path. A variant of IC is also proposed that can control the system along the reference trajectory. The performance of this method is compared with MPC in terms of computational time and control quality on the task of UAV trajectory tracking both in simulation and in the laboratory using a real UAV.

Keywords: Unmanned Aerial Vehicle, Drone, Trajectory Planning, Trajectory Tracking, Model Predictive Control, Interpolating Control, Optimal Control Problem, Pseudospectral Method.

Abstrakt

Tato práce se zabývá vývojem systému pro plánování a sledování trajektorií kvadrotorových bezpilotních letounů (UAV). Tento systém je klíčový pro jakoukoli aplikaci, která používá plně autonomní UAV, protože kvalita trajektorie může výrazně ovlivnit výsledek mise UAV. Systém je založen na dvou metodách: pseudospektrální metoda (PSM) pro efektivní plánování trajektorie a metoda interpolačního řízení (IC) pro sledování trajektorie.

V první části této práce jsou položeny základy, kde je popsáno chování UAV a jsou představeny metody pro plánování cesty a trajektorie. Důraz je kladen na formulaci problému plánování trajektorie jako úlohy optimálního řízení (OCP) a jeho numerické řešení pomocí PSM. Dále je prezentován problém sledování trajektorie a je popsána state-of-the-art metoda prediktivního řízení (MPC) spolu s její výpočetně méně náročnou alternativou IC.

V druhé části práce je navržen systém pro adaptivní plánování trajektorie UAV založený na PSM, který využívá znalost chování UAV a jeho optimální cesty. Je také navržena varianta IC, která dokáže řídit systém podle referenční trajektorie. Chování této metody je porovnáno s MPC z hlediska výpočetního času a kvality řízení na úloze sledování trajektorie pomocí UAV jak v simulaci, tak i v laboratoři na reálném UAV.

Klíčová slova: bezpilotní letoun, dron, plánování trajektorie, sledování trajektorie, prediktivní řízení, interpolační řízení, optimální řízení, pseudospektrální metoda.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

BVP	Boundary Value Problem.
BLDC	Brush-Less Direct Current.
C-space	Configuration Space.
ENU	East-North-Up.
EKF	Extended Kalman Filter.
eIC	extended version of Interpolating Control.
FDM	Finite Difference Method.
FEM	Finite Element Method.
FCU	Flight Control Unit.
IMU	Inertial Measurement Unit.
ISE	Integral Square Error.
IC	Interpolating Control.
LMPC	Linear Model Predictive Control.
LP	Linear Program.
LQR	Linear Quadratic Regulator.
LTI	linear time-invariant.
LTV	linear time-varying.
MWRM	Mean Weighted Residual Method.
MEMS	Microelectromechanical Systems.
MPC	Model Predictive Control.
MPT3	Multi-Parametric Toolbox 3.
MIMO	Multiple-Input Multiple-Output.
NMPC	Nonlinear Model Predictive Control.
NLP	Nonlinear Program.
OCP	Optimal Control Problem.
PD	Proportional-Derivative.
PID	Proportional-Integral-Derivative.

PSEM	Pseudospectral Elements Method.
PSM	Pseudospectral Method.
QP	Quadratic Program.
RRT	Rapidly-exploring Random Tree.
RRT*	Rapidly-exploring Random Tree*.
RMPC	Robust Model Predictive Control.
SAR	search and rescue.
SMPC	Stochastic Model Predictive Control.
SE(3)	three-dimensional Special Euclidean Group.
UAV	Unmanned Aerial Vehicle.
VC	Vertex Control.
VTOL	Vertical Take-Off and Landing.

List of Symbols

Sign	Description	Unit
$\mathcal{C}^N$	N-step controlled positively invariant set	
$\boldsymbol{a}$	acceleration	$\mathrm{m} \cdot \mathrm{s}^{-2}$
$\mathcal{C}^S$	additional S-step controlled positively invariant set	
$\boldsymbol{\Omega}^m$	additional positively invariant set	
$\boldsymbol{F}_A^B$	aerodynamic force in body coordinate frame	N
ρ	air density	$\mathrm{kg} \cdot \mathrm{m}^{-3}$
t_G	air temperature of pressure reference point	°C
$\dot{\boldsymbol{\omega}}$	angular acceleration in body frame	$\mathrm{rad} \cdot \mathrm{s}^{-2}$
$\boldsymbol{\omega}$	angular rate in body frame	$\mathrm{rad} \cdot \mathrm{s}^{-1}$
ϖ_i	angular rate of i-th rotor	$\mathrm{rad} \cdot \mathrm{s}^{-1}$
p_A	atmospheric pressure	Pa
$p_{A,k}$	atmospheric pressure at time instant k	Pa
p_G	atmospheric pressure reference	Pa
$\boldsymbol{q}_e$	attitude quaternion error	
$\vec{x}^B$, $\vec{y}^B$, $\vec{z}^B$	axes of body coordinate frame B	
$\vec{x}^L$, $\vec{y}^L$, $\vec{z}^L$	axes of local coordinate frame L	
ν	basis function	
τ_{F_i}	blade flapping torque around axis i	$\mathrm{N} \cdot \mathrm{m}$
B	body coordinate frame	
Φ	boundary cost function	
$\boldsymbol{D}$	coefficient of induced drag	
I_R	coefficient of rotor inertia	$\mathrm{kg} \cdot \mathrm{m}^{-2}$
k_β	coefficient of rotor stiffness	
$\boldsymbol{\tau}_R$	collective torque	$\mathrm{N} \cdot \mathrm{m}$
τ_{R_x}, τ_{R_y}, τ_{R_z}	collective torques around corresponding axes of body frame	$\mathrm{N} \cdot \mathrm{m}$
k_F	constant relating flapping to apparent wind velocity	
$\boldsymbol{u}$	control vector	
$\boldsymbol{u}_k$	control vector at the time instant k	
$\boldsymbol{u}(t)$	control vector at the time instant t	
$\boldsymbol{u}_k^N$	control vectors from the time instant k to N	
$\boldsymbol{x}^h$	decomposed state vector for high gain control law	

Sign	Description	Unit
$\boldsymbol{x}^l$	decomposed state vector for low gain control law	
$\boldsymbol{x}^m$	decomposed state vector for medium gain control law	
$\dot{\boldsymbol{q}}_e$	derivative of attitude quaternion error	
$\dot{\boldsymbol{\omega}}_r$	desired angular acceleration in body frame	$\mathrm{rad} \cdot \mathrm{s}^{-2}$
$\boldsymbol{\omega}_r$	desired angular rate in body frame	$\mathrm{rad} \cdot \mathrm{s}^{-1}$
$\boldsymbol{q}_r$	desired attitude quaternion	
$\boldsymbol{q}_{r'}$	desired attitude quaternion without desired heading	
$\boldsymbol{x}_r$	desired state vector	
$\boldsymbol{x}_{r,k}$	desired state vector at time instant k	
$\boldsymbol{\tau}_r$	desired torque on UAV's body in body frame	$\mathrm{N} \cdot \mathrm{m}$
$\boldsymbol{F}_r^B$	desired translational force inflicting UAV's body in body coordinate frame	N
$\boldsymbol{F}_r^L$	desired translational force inflicting UAV's body in local coordinate frame	N
$\ddot{\boldsymbol{r}}_r^L$	desired UAV acceleration in local coordinate frame L	$\mathrm{m} \cdot \mathrm{s}^{-2}$
$\boldsymbol{r}_r^L$	desired UAV position in local coordinate frame L	m
$\dot{\boldsymbol{r}}_r^L$	desired UAV velocity in local coordinate frame L	$\mathrm{m} \cdot \mathrm{s}^{-1}$
ψ_r	desired yaw angle	rad
$I_{xx},\ I_{yy},\ I_{zz}$	diagonal elements of matrix $\boldsymbol{I}$	$\mathrm{kg} \cdot \mathrm{m}^{-2}$
d_R	diameter of rotor	m
h_R	displacement of rotor along $\vec{z}^B$	m
$\dot{\omega}_x,\ \dot{\omega}_y,\ \dot{\omega}_z$	elements of angular acceleration corresponding to axes in body frame	$\mathrm{rad} \cdot \mathrm{s}^{-2}$
$\omega_x,\ \omega_y,\ \omega_z$	elements of angular rate corresponding to axes in body frame	$\mathrm{rad} \cdot \mathrm{s}^{-1}$
$q_x,\ q_y,\ q_z$	elements of imaginary part of quaternion	
$\dot{q}_x,\ \dot{q}_y,\ \dot{q}_z$	elements of imaginary part of quaternion derivative	
$\ddot{x}^B,\ \ddot{y}^B,\ \ddot{z}^B$	elements of UAV acceleration in body coordinate frame B	$\mathrm{m} \cdot \mathrm{s}^{-2}$
$\ddot{x}^L,\ \ddot{y}^L,\ \ddot{z}^L$	elements of UAV acceleration in local coordinate frame L	$\mathrm{m} \cdot \mathrm{s}^{-2}$
$x^B,\ y^B,\ z^B$	elements of UAV position in body coordinate frame B	m
$x^L,\ y^L,\ z^L$	elements of UAV position in local coordinate frame L	m
$\dot{x}^B,\ \dot{y}^B,\ \dot{z}^B$	elements of UAV speed in body coordinate frame B	$\mathrm{m} \cdot \mathrm{s}^{-1}$
$\dot{x}^L,\ \dot{y}^L,\ \dot{z}^L$	elements of UAV speed in local coordinate frame L	$\mathrm{m} \cdot \mathrm{s}^{-1}$
$\boldsymbol{p}_k$	error vector of attitude at time instant k	
$\phi,\ \theta,\ \psi$	Euler angles for roll, pitch, and yaw angle, respectively	rad
α_i	flap angle in axis i	rad
$\boldsymbol{F}$	force	N
$\boldsymbol{F}_T^B$	force of collective thrust in body coordinate frame	N
$\boldsymbol{s}_{goal}$	goal vertex of 3D grid for find-path algorithms	
g	gravitational constant	$\mathrm{m} \cdot \mathrm{s}^{-2}$
h_G	height of pressure reference point	m
$\boldsymbol{u}^h$	High gain control law for decomposed state vector $\boldsymbol{x}^h$	

Sign	Description	Unit
$\boldsymbol{q}_v$	imaginary part of quaternion	
$\dot{\boldsymbol{q}}_v$	imaginary part of quaternion derivative	
$\boldsymbol{F}_D^B$	induced drag in body coordinate frame	N
$\boldsymbol{\Omega}^h$	inner positively invariant set	
$\mathfrak{L}$	integral cost function	
$\boldsymbol{L}$	Lagrangian	
l	length of UAV's arm	m
h_A	local altitude	m
L	local coordinate frame	
$\boldsymbol{u}^l$	Low gain control law for decomposed state vector $\boldsymbol{x}^l$	
k_D	lumped drag coefficient	
$\boldsymbol{K}_D$	lumped drag coefficient matrix	
k_T	lumped thrust coefficient	
k_τ	lumped torque coefficient	
m	mass of UAV	kg
$\boldsymbol{F}^N$	matrix for inequavility defining the N-step controlled positively invariant set	
$\boldsymbol{F}^m$	matrix for inequavility defining the additional positively invariant controlled set	
$\boldsymbol{F}^u$	matrix for inequavility defining the control constraining set	
$\boldsymbol{F}^h$	matrix for inequavility defining the inner positively invariant set	
$\boldsymbol{F}^l$	matrix for inequavility defining the outter positively invariant set	
$\boldsymbol{F}^x$	matrix for inequavility defining the state constraining set	
$\boldsymbol{\Gamma}(\boldsymbol{q})$	matrix for description of dynamics with quaternion $\boldsymbol{q}$	
$\boldsymbol{B}$	matrix of control	
$\boldsymbol{A}$	matrix of dynamics	
$\boldsymbol{A}_c$	matrix of dynamics for closed-loop system	
$\boldsymbol{K}$	matrix with gain of controller	
$\boldsymbol{\Omega}_{\max}$	maximal positively invariant set	
$\boldsymbol{u}^m$	Medium gain control law for decomposed state vector $\boldsymbol{x}^m$	
J	optimality criterion	
$\boldsymbol{\Omega}^l$	outter positively invariant set	
θ_a	pitch angle measured by accelerometer	rad
$\boldsymbol{U}$	potential energy	J
$\boldsymbol{q}$	quaternion	
$\boldsymbol{q}_k$	quaternion at time instant k	
$\dot{\boldsymbol{q}}$	quaternion derivative	
$\boldsymbol{s}_{rand}$	random vertex for RRT* algorithm	

Sign	Description	Unit
ε	residual function	
ϕ_a	roll angle measured by accelerometer	rad
$\boldsymbol{R}_{\phi\theta\psi}$	rotation matrix	
$\boldsymbol{R}_{\phi}$, $\boldsymbol{R}_{\theta}$, $\boldsymbol{R}_{\psi}$	rotation matrix for rotation around x,y and z, respectively	
$\boldsymbol{F}_q$	rotational force	N
$\boldsymbol{T}_q$	rotational kinetic energy	J
τ_{I_z}	rotor inertia	$\mathrm{N}\cdot\mathrm{m}$
q_w	scalar part of quaternion	
q_w	scalar part of quaternion derivative	
$\mathcal{U}$	set constraining the control vector	
$\mathcal{W}$	set constraining the noise vector	
$\mathcal{X}$	set constraining the state vector	
$\mathcal{C}^{N(j)}$	simplex j of N-step controlled positively invariant set	
$\boldsymbol{\Omega}_i$	skew-symmetric matrix of dimmension i	
ρ_0	standard air density for 0°C	$\mathrm{kg}\cdot\mathrm{m}^{-3}$
$\boldsymbol{s}_{start}$	start vertex of 3D grid for find-path algorithms	
$\boldsymbol{x}$	state vector	
$\boldsymbol{x}_k$	state vector at the time instant k	
$\boldsymbol{x}(t)$	state vector at the time instant t	
$\boldsymbol{w}$	system noise	
t_A	temperature of air	°C
η	test function	
K_T	thrust coefficient	
F_i	thrust of i-th rotor	N
K_τ	torque coefficient	
τ_i	torque of i-th rotor	$\mathrm{N}\cdot\mathrm{m}$
$\boldsymbol{\tau}$	torque on UAV's body in body frame	$\mathrm{N}\cdot\mathrm{m}$
$\boldsymbol{F}^L$	translational force inflicting UAV's body in the local coordinate frame	N
$\boldsymbol{T}_L$	translational kinetic energy	J
$\mathcal{T}$	tree for the RRT* algorithm	
$\boldsymbol{a}^B$	UAV acceleratin in body coordinate frame B	$\mathrm{m}\cdot\mathrm{s}^{-2}$
$\boldsymbol{a}^L$	UAV acceleration in local coordinate frame L	$\mathrm{m}\cdot\mathrm{s}^{-2}$
$\boldsymbol{r}^B$	UAV position in body coordinate frame B	m
$\boldsymbol{r}^L$	UAV position in local coordinate frame L	m
$\dot{\boldsymbol{r}}^B$	UAV velocity in body coordinate frame B	$\mathrm{m}\cdot\mathrm{s}^{-1}$
$\dot{\boldsymbol{r}}^L$	UAV velocity in local coordinate frame L	$\mathrm{m}\cdot\mathrm{s}^{-1}$
$\boldsymbol{I}$	UAV's matrix of inertia	$\mathrm{kg}\cdot\mathrm{m}^{-2}$
$\boldsymbol{g}^N$	vector for inequavility defining the N-step controlled positively invariant set	

Sign	Description	Unit
$\boldsymbol{g}^m$	vector for inequavility defining the additional positively invariant controlled set	
$\boldsymbol{g}^u$	vector for inequavility defining the control constraining set	
$\boldsymbol{g}^h$	vector for inequavility defining the inner positively invariant set	
$\boldsymbol{g}^l$	vector for inequavility defining the outter positively invariant set	
$\boldsymbol{g}^x$	vector for inequavility defining the state constraining set	
$\boldsymbol{\lambda}$	vector of Lagrange multipliers	
$\boldsymbol{\xi}$	vector of position and quaternion	
$\boldsymbol{s}$	vertex of 3D grid for find-path algorithms	
$\boldsymbol{S}_{near}$	vertices for RRT* algorithm which are near some vertex $\boldsymbol{s}$	
$\boldsymbol{R}$	weight matrix of control	
$\boldsymbol{Q}$	weight matrix of state	
ψ_{GPS}	yaw angle based on data from GPS receiver	rad
ψ_m	yaw angle measured by magnetometer	rad

1 Introduction

Before 2010, fixed-wing aircraft and helicopters held significant dominance in the fields of small aircraft, such as aerial photography and aircraft sports [80]. Nevertheless, in the following years, there were substantial changes, and due to the ease-of-use, the multirotor helicopters, called multirotors, (e.g. quadrotors), gained popularity. In the beginning, the autopilots and other components were sold separately and further assembled by professionals. At the end of 2012, a company DJI (Dà-Jiāng Innovations) came with its commercially successful ready-to-fly quadrotor called *Phantom*. Thanks to its accessibility, low price, and reliability, it helped to raise awareness and provided a multirotor helicopter for amateurs. It caused a huge public interest, which resulted in the boom of new multirotors, autopilots, specialized accessories, employment in robotics, and the unshakable dominance of multirotors in the market of small aircraft.

The multirotors are helicopters that have three or more rotors. They consist of a fixed frame, which serves as a base for rotors (motor with fixed propeller), that are placed at the ends of frame arms. Besides their ease of use and simple design, their biggest advantage is the ability of Vertical Take-Off and Landing (VTOL). The most common type of multirotor is the quadrotor helicopter (see Figure 1.1). It is, as the name implies, composed of four independent rotors and it is popular because of a low number of components and the property that the moments of each other rotor counters.

A term, that is tightly coupled with multirotor helicopters, is Unmanned Aerial Vehicle (UAV) [80, 56], which is an aircraft without pilots on-board. The flight of UAV is controlled autonomously by Flight Control Unit (FCU) or by the remote control from a pilot on the ground or in another vehicle. Sometimes, UAVs are also called *drones*. There are several types of UAVs; the UAV airships, aircraft, VTOL (where standard helicopters and multirotor helicopters are included), and balloons. In this work, a multirotor UAV, namely a quadrotor UAV will be considered.

In the 20th century, there were several experiments with large-scale quadrotor helicopters, however, they were incomparable with standard helicopters in critical parameters such as payload, fly-time, safety, and others, which resulted in their abandoning. Nevertheless, since the 90s, there has been a big development in Microelectromechanical Systems (MEMS), namely in MEMS Inertial Measurement Units (IMUs). These IMU allowed the construction of such vehicles as quadrotor UAV. The MEMS-based IMU suffered from significant noise of measurement, but this problem was solved with the employment of microcontrollers that were at the same time sufficiently powerful and small for processing data and additionally, for the employment in FCU.

The rapid development resulted in an intensification of research in UAVs and their applications since 2005. Nowadays, UAVs penetrate various civil applications such as search and rescue (SAR), remote sensing, precision agriculture, construction and infrastructure inspection, delivery of goods, real-time surveillance, or providing a wireless coverage [88].

In the SAR, the UAVs can speed up search operations and they can help during nature-origin

Figure 1.1: Quadrotor helicopter

or man-made disasters with the transport of medicine, food, and water supply, and with rapid service recovery, where the swarm of UAVs can temporarily replace damaged communication network. They can also help with post-disaster analysis by monitoring the area for structural weakness and inspecting the functionality of critical water, communication, or transport infrastructure and power supply.

In precise agriculture, the UAVs serve in crop management and monitoring with e.g. weed and disease detection, irrigation scheduling, or pesticide spraying. By employment of UAVs, the efficiency of agriculture can be dramatically increased and also, the plant requirements can be significantly lowered.

The monitoring of infrastructure mentioned in the SAR is also important for the regular inspection of high voltage transmission lines for their distance from trees, buildings, or bad conductivity detection. Moreover, construction site inspections could minimize the injuries of personnel and damages caused by failures.

The delivery of goods could be revolutionized with UAV employment, mainly in the last-mile delivery, and it could significantly help in places without infrastructure. The UAVs play an even more crucial role in the healthcare field, where fast and safe deliveries are essential and the medicine, immunization, blood samples, or even replacement organs can be transported by a UAV.

Those applications have many common challenges [88]. There are many problems caused by legislation because the fully autonomous UAVs missions are highly prohibited all over the globe. Moreover, there are limitations in the employment of UAVs in crowded areas or high altitudes. The legislative restrictions can be overcome by exceptions or with the employment of complex air control systems similar to standard air traffic. On the other hand, there are several problems of a technical or scientific nature. The most common problem is the power supply, which is provided in the vast majority by Lithium-polymer batteries. Nowadays batteries provide up to tens of minutes of flight time, nevertheless, there is rapid research supported also due to electric cars and smartphones, where the performance of batteries is also essential. Moreover, the UAV applications require small and precise sensors, and standard or thermal vision cameras, which may face difficult conditions such as mechanical vibrations, fast-changing light terms, and detrimental weather. Other challenges are security or networking aspects.

Trajectory planning [59] is essential for every application that employs the fully autonomous UAV and therefore it will be the major topic of this work. The trajectory defines not only the path of a UAV but also its behavior along the path such as speed and attitude as functions of time. To accomplish the mission, the UAV must sometimes correctly operate with the accessories and therefore, the behavior of the accessory can also be appended to the trajectory. Before the start of the mission such as flight through given waypoints, covering a specific area, or payload transport between points, the trajectory should be defined according to the mission conditions and criteria considering the available energy, distance, speed limits, and other indicators.

The fundamental property of a trajectory is to be achievable by the given UAV, which means that during the planning, the properties of all UAVs that will fly along the trajectory must be considered. For the sake of safety and to reflect the changes in the environment, the adjustment of the trajectory should be possible during the mission. Before the start of trajectory planning, the properties of UAV and environment should be known. The environment can be described using a map, where the obstacles are registered. From the map, the knowledge about the environment can be extracted and stored in the data structure.

For trajectory planning, there are several distinct approaches, which usually employ graph-based algorithms for path-planning and the subsequent conversion of the path into the trajectory, methods based on potential fields, or model-based approach using a definition of the problem in the manner of Optimal Control Problem (OCP) and its solution [8, 91].

As has been mentioned, it is important to ensure that the UAV will accomplish flight along the given trajectory. To solve this problem, the trajectory tracking methods are used, which are usually implemented on-board in the FCU or some additional on-board computational system. These methods are capable of closed-loop control, which guarantees the robustness of the system, taking into account the future course of a reference trajectory. In the tracking problem, the complex dynamics of a specific UAV can be considered along with its constraints given by structural and physical properties.

The thesis presents methods for trajectory planning and tracking with a focus on UAV applications. The presentation is accompanied by fundamental knowledge about the behavior of quadrotor UAVs and their control. The goal of the thesis is to propose improved algorithms for both trajectory planning and tracking based on the described methods.

Chapter 2 describes UAVs and their behavior. First, the coordinate system and attitude of the UAV are described. Next, the mathematical model of UAV behavior is presented, including the incorporation of minor phenomena into the UAV model.

Chapter 3 is devoted to trajectory planning. Two distinct approaches to planning are presented. First, path planning methods are described, which require additional adjustments to transform them into trajectory. The second approach is based on describing trajectory planning using Optimal Control Problem (OCP) methodology and is supplemented with the presentation of numerical methods for the approximate solution of time-continuous nonlinear OCP.

The trajectory tracking problem is described in Chapter 4. Model Predictive Control (MPC), the state-of-the-art method for trajectory tracking, is presented. General areas of MPC are denoted, and its application for UAVs is introduced. Afterward, a computationally efficient alternative to the MPC called Interpolating Control (IC) is described.

Chapter 5 presents the objectives of the thesis. The goal is to design a trajectory planning system for a quadrotor UAV. The first subtask is to propose an effective method for UAV trajectory planning, building upon the methods described in Chapter 3. The method is made more efficient by obtaining a better initial guess of the solution based on the knowledge of the optimal path

and dynamics of the UAV. The second subtask is to develop a trajectory tracking method based on Interpolation Control introduced in Chapter 4, making it applicable for trajectory tracking using UAVs.

Chapter 6 focuses on trajectory planning for UAVs, building upon concepts from Chapter 3. This chapter proposes a design using Pseudospectral Method (PSM) and Pseudospectral Elements Method (PSEM) to create efficient, precise, and accurate trajectories. The approach includes utilizing graph-based path planning for accurate initial guesses and splitting the problem into smaller interconnected segments to reduce computation time and improve feasibility.

In Chapter 7, an IC variant is proposed to enable its use for UAV trajectory tracking. The process begins by presenting an IC variant for steering to an arbitrary setpoint and then generalizing it to arbitrary trajectory tracking. The performance of the proposed methods is compared with Model Predictive Control (MPC) in terms of computation time and quality. Additionally, the controller is tested for UAV both in simulation and in the laboratory to validate its effectiveness and applicability.

The source code produced during the work on this thesis is available at `https://github.com/zboucek/uav-tracking-planning`. Data and results generated from the source files and in the experiments are stored on Zenodo, the trajectory planning in [16] and the trajectory tracking in [15].

2 Unmanned Aerial Vehicle

This chapter describes UAV and its behavior. First, the UAV's position and attitude are defined. Further, the considered type of UAV is briefly described. Afterward, a mathematical model of the UAV is designed based on the forces and torques. The obtained model is then extended to include aerodynamic effects that can significantly influence the behavior of the UAV.

2.1 Coordinate System and Attitude

The position of a point in space is described by a coordinate vector, which represents a displacement of the point with respect to a given reference coordinate frame. A coordinate frame is a set of orthogonal axes that intersect at a point of origin. Frequently, there is a need to transform the point from one coordinate frame to another. A coordinate system is a set of frames and relations among them. It is in particular given by the application and it is also tightly coupled with the area of operation.

A direction of the frame coordinates and a rotation around the axes is in most frames given by the right-hand rule. In a 3D space, the thumb, first, and middle finger are pointing in an orthogonal way and in a direction of coordinates. The direction of the rotation around the axis is determined by the direction of the curled fingers when the hand is closed around the axis, with the thumb pointing in the direction of one of the axis coordinates.

Further, several types of coordinate frames will be denoted for the needs of this thesis. A local coordinate frame denoted as L, sometimes called a local navigation frame is suitable for the description of an environment, where the UAV operates. It is also used for the description of UAV's behavior (its translational and rotational motion). In the frame L, the curvature of the Earth's surface is neglected.

The convention used for the coordinate frame in this thesis is called East-North-Up (ENU) (see Figure 2.1), where the axis $\vec{z}^L$ is pointing in the opposite direction to the local gravitational field vector ("up") and $\vec{y}^L$ is pointing to the true north.

Another type of frame is called a body coordinate frame or a vehicle frame denoted as B. It has an origin in a UAV's center of mass and is aligned with the body of UAV. In most cases,

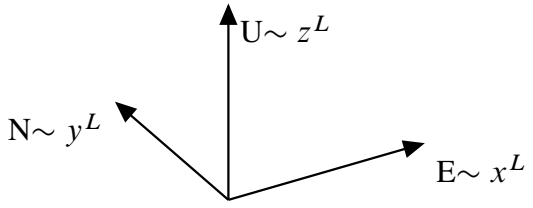


Figure 2.1: Earth-North-Up (ENU)

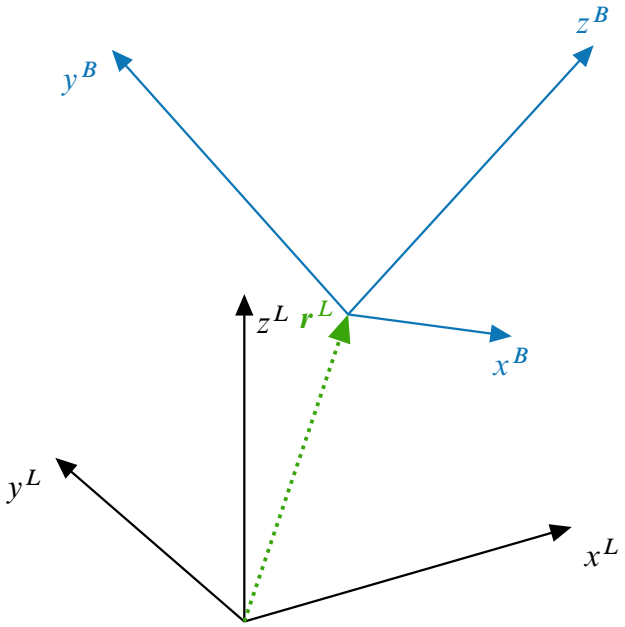


Figure 2.2: Coordinate system

the body coordinate frame is used for the description of a rotor position and consequently the influence of rotors on the UAV. Accessories that are on board of the UAV (sensors, cameras, payload, robotic arms, etc.) can be described in this frame as well. Frame B is composed of axes $\vec{x}^B$, $\vec{y}^B$, $\vec{z}^B$.

As has been said, the coordinate system of UAV (see Figure 2.2) is composed of the frames L and B. The position of vehicle is described in the frame L by the vector $\boldsymbol{r}^L = \left[x^L,\ y^L,\ z^L\right]^\mathsf{T}$. The origin of the frame B is at the coordinates $\boldsymbol{r}^L$ that is the position of UAV's center of mass. The frame B can be rotated with reference to the frame L and is aligned with the body of UAV; it means that this rotation is further denoted as UAV attitude.

There are several ways to represent the attitude [80, Chapter 5]. The most frequently used representation is based on the Euler angles. In 3D space, it is composed of 3 angles. Each angle corresponds to the rotation around the relevant axis. The main advantage of the Euler angles is in their clarity, simple operations, and compactness (only three elements are required to describe the full attitude). Unfortunately, there is a major drawback called the Gimbal Lock phenomenon [42]. It appears when a certain sequence of rotations is performed. As a result, two rotational axes become aligned and a rotation around one axis causes rotation around another one as well. In other words, one degree of freedom is lost. Further, in 3D the full rotation is dependent on the sequence of rotations around axes.

Another representation is a rotation matrix. In 3D space, it is a square matrix with 9 elements. This representation is beneficial for the full description of attitude without singularities and unambiguity. The disadvantage is the high number of elements.

The attitude representation, that will be used in this thesis, is called quaternion [35, 40]. It is a hyper-complex number with four components

$$\boldsymbol{q} = [q_w, \boldsymbol{q}_v^\mathsf{T}]^\mathsf{T},\ \boldsymbol{q}_v = [q_x,\ q_y,\ q_z]^\mathsf{T}, \tag{2.1}$$

where q_w is a scalar part of the quaternion and $\boldsymbol{q}_v$ contains the coefficients of the imaginary part. Using quaternions to describe attitude is advantageous because of the small number

of elements compared to the rotation matrix and unlike Euler angles, the singularity does not occur. In Appendix A, some useful relations and operations with quaternions are presented, and Appendix B describes conversions between quaternions and other attitude representations.

2.2 Mathematical Model

The considered type of UAV is a standard quadrotor UAV [80, 63, 98]; nevertheless, the model and its parts are easy to adapt for a different type of UAV. The quadrotor UAV is equipped with four rotors, which are composed of motors and propellers. The rotors are mounted at the end of each arm. In the overwhelming majority, the Brush-Less Direct Current (BLDC) motor is used. It is assumed that all rotors have the same parameters except for the direction of spinning.

There are two main construction configurations for a quadrotor (see Figure 2.3). The schemes differ in the direction of forward flight. The first is "+"-configuration (see Figure 2.3a) the forward flight is in the direction of rotors 1 and it is employed in this thesis. The second is "×"-configuration (see Figure 2.3b) where the direction of forward flight is between rotors 1 and 2.

The quadrotor UAV is considered a rigid body with the center of the UAV mass in the origin of the body frame B coordinates. The quadrotor UAV connection to the local and the body frame in "+"-configuration is depicted in Figure 2.4, where the rotor 1 is in the positive direction of the axis $\vec{x}^{B}$.

The behavior of UAV is described by two important equations. The first equation is Newton's second law of motion, which describes the translational motion

$$\boldsymbol{F} = m \cdot \boldsymbol{a}, \tag{2.2}$$

where $\boldsymbol{F}$, m, $\boldsymbol{a}$ are force, mass, and acceleration, respectively. The acceleration can be denoted also using the second derivative of position in the frame L as $\boldsymbol{a}^{L} = \ddot{\boldsymbol{r}}^{L} = \left[\ddot{x}^{L},\ \ddot{y}^{L},\ \ddot{z}^{L}\right]^{\mathsf{T}}$. For local frame L, Equation (2.2) is modified to

$$\boldsymbol{F}^{L} - m \cdot g \cdot [0,\ 0,\ 1]^{\mathsf{T}} = m \cdot \ddot{\boldsymbol{r}}^{L}, \tag{2.3}$$

where the force $\boldsymbol{F}$ is substituted with the sum of generalized force $\boldsymbol{F}^{L}$ and gravitational force $-m \cdot g \cdot [0,\ 0,\ 1]^{\mathsf{T}}$. The second equation, which describes the rotational motion, is called the Euler's equation of motion

$$\boldsymbol{\tau} = \boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}, \tag{2.4}$$

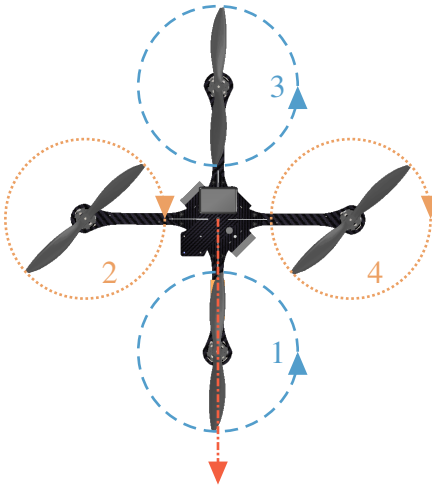
where $\boldsymbol{\tau}$, $\boldsymbol{I}$, $\boldsymbol{\omega}$, $\dot{\boldsymbol{\omega}}$ are vector of torque, the UAV's matrix of inertia, angular rate, and angular acceleration, respectively.

Because of the quaternion dynamics, the relation (A.17) is employed; this relation describes how the angular rate affects the attitude quaternion. The resulting equations of motion are as follows

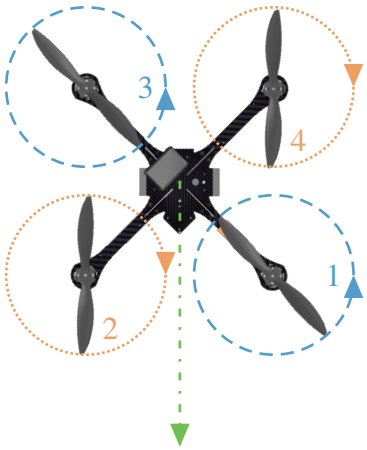
$$\ddot{\boldsymbol{r}}^{L} = -\, g\, [0,\ 0,\ 1]^{\mathsf{T}} + \frac{\boldsymbol{F}^{L}}{m}, \tag{2.5}$$

$$\dot{\boldsymbol{\omega}} = -\, \boldsymbol{I}^{-1}\, (\boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}) + \boldsymbol{I}^{-1}\boldsymbol{\tau}, \tag{2.6}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{\Gamma}(\boldsymbol{q})^{\mathsf{T}}\boldsymbol{\omega}, \tag{2.7}$$



(a) "+"-configuration



(b) "×"-configuration

Figure 2.3: Quadrotor UAV schematics with a direction of rotor rotation and a forward direction in "+" and "×"-configuration

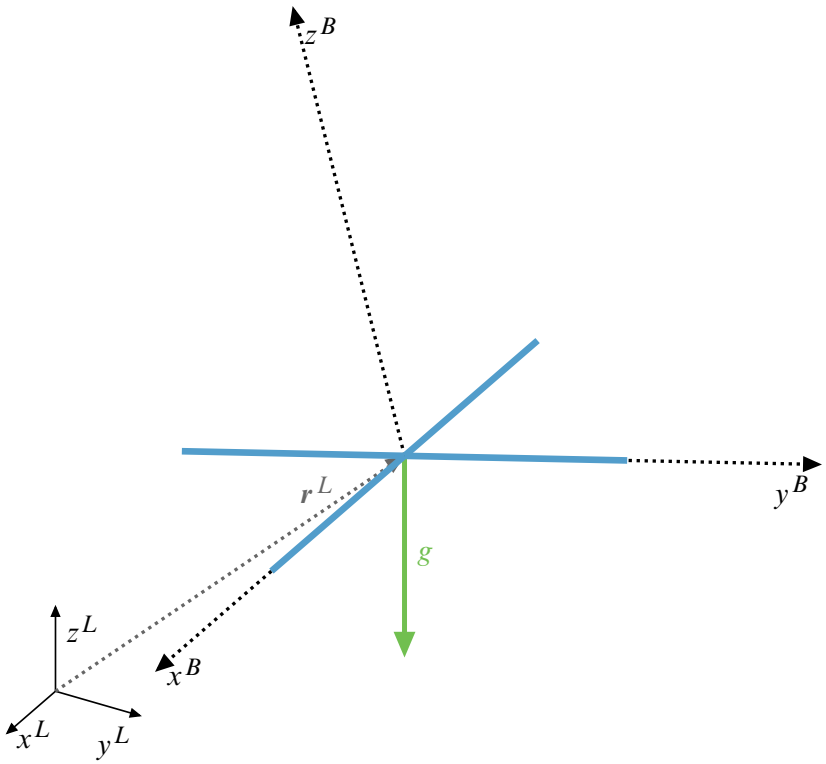


Figure 2.4: Quadrotor UAV in the local coordinate frame

where

$$\mathbf{\Gamma}(\boldsymbol{q}) = \begin{bmatrix} -q_x & q_w & q_z & -q_y \\ -q_y & -q_z & q_w & q_x \\ -q_z & q_y & -q_x & q_w \end{bmatrix}. \tag{2.8}$$

The stabilization control of UAV is discussed in Appendix C, which focuses on designing a nonlinear controller to stabilize the position of the UAV, utilizing the presented mathematical model as its foundation.

2.3 Forces and Torques

The behavior of the UAV is closely related to the forces and torques acting on the UAV body. The standard source of force and torque in the UAV's system is the rotor [63, 80]. The spinning of the rotor generates force, which is called a rotor thrust, it is parallel with the axis $\vec{z}^B$ and it can be described as

$$F_i = K_T \rho d_R^4 \left(\frac{\varpi_i}{2\pi}\right)^2 = k_T \varpi_i^2, \tag{2.9}$$

where i indicates the rotor number, K_T is a dimensionless thrust coefficient, which is considered the same for all rotors, ρ is air density in [kg / m^3], d_R is the diameter of the rotor in [m], ϖ_i is an angular rate of i-th rotor in [rad/s] and k_T is a lumped thrust coefficient. The air density can be calculated according to a local altitude h_A in [m] and a temperature t_A in [°C] as

$$\rho = \frac{273 p_A}{101325\,(273 + t_A)} \rho_0, \tag{2.10}$$

where $\rho_0 = 1.293$ kg / m^3 is the standard air density for 0°C and p_A is the atmospheric pressure that can be obtained as

$$p_A = 101325 \left(1 - 0.0065 \frac{h_A}{273 + t_A}\right)^{5.2561}. \tag{2.11}$$

The spinning of the rotor also generates a rotor torque, that is orthogonal to the axis $\vec{z}^B$. The direction of the torque is opposite to the rotor spinning direction and it can be described as

$$\tau_i = (-1)^{i+1} \frac{K_\tau \rho d_R^5}{4\pi^2} \varpi_i^2 = (-1)^{i+1} k_\tau \varpi_i^2, \tag{2.12}$$

where K_τ is the coefficient of the rotor's torque and k_τ is a lumped coefficient of the rotor's torque. For more insight, the procedure for the rotor coefficients K_T, K_τ calculation based on the geometry of the rotor is described in [80].

Now, the direct effect of rotors on the UAV's body can be described. First, the force that affects the UAV is called collective thrust $\boldsymbol{F}_T^B$ which is defined in the frame B and it is simply a vector with the sum of all rotor thrusts in axis $\vec{z}^B$

$$\boldsymbol{F}_T^B = \left[0,\, 0,\, \sum_{i=1}^{4} F_i\right]^\mathsf{T}. \tag{2.13}$$

Second, the torque is called collective torque $\boldsymbol{\tau}_R$ and it depends on the sum of rotor torques and proportionally on the differences between rotor thrusts

$$\boldsymbol{\tau}_R = \begin{bmatrix} \tau_{R_x} & \tau_{R_y} & \tau_{R_z} \end{bmatrix}^\mathsf{T}, \tag{2.14}$$

where τ_{R_x}, τ_{R_y}, τ_{R_z} are torques around $\vec{x}^B$, $\vec{y}^B$, $\vec{z}^B$, respectively. In the case of "×"-configuration, the torques can be described as

$$\boldsymbol{\tau}_R = \begin{bmatrix} \tau_{R_x} \\ \tau_{R_y} \\ \tau_{R_z} \end{bmatrix} = \begin{bmatrix} l\dfrac{K_T \rho d_R^4}{4\pi^2}\left(-\varpi_2^2 + \varpi_4^2\right) \\ l\dfrac{K_T \rho d_R^4}{4\pi^2}\left(-\varpi_1^2 + \varpi_3^2\right) \\ \sum_{i=1}^{4} \tau_i \end{bmatrix} = \begin{bmatrix} lk_T\left(-\varpi_2^2 + \varpi_4^2\right) \\ lk_T\left(-\varpi_1^2 + \varpi_3^2\right) \\ \sum_{i=1}^{4} \tau_i \end{bmatrix}, \tag{2.15}$$

where l is the length of UAV's arm.

Sometimes it can be beneficial to describe minor effects in the UAV dynamics, mainly in applications where high quality and precision are needed. Further, some of these effects will be presented. In applications with large and heavy UAVs, it can be advantageous to consider the inertia of rotor blades [62]. This feature can be reflected in the low-level control, such as rotor speed or attitude control. The rotor inertia τ_{I_z} of all rotors together act in the system of UAV as an additional torque around $\vec{z}^B$ and can be described by the following relation

$$\tau_{I_z} = I_R \sum_{i=1}^{4} (-1)^{i+1}\, \dot{\varpi}_i, \tag{2.16}$$

where I_R is the coefficient of rotor inertia and $\dot{\varpi}_i$ is the angular acceleration of i-th rotor.

Induced drag [46] is a force that acts against the vehicle's motion (see Figure 2.5). It is highly dependent on the parameter $\boldsymbol{D}^B$, which is affected by the UAV aerodynamics. The resulting force can be described in the body frame B as

$$\boldsymbol{F}_D^B = -\boldsymbol{D}^B \cdot \left(\dot{\boldsymbol{r}}^B\right)^2, \tag{2.17}$$

where $\dot{\boldsymbol{r}}^B$ is the speed in the body frame. Compensation of the induced drag is convenient in the position control of UAV.

Blade flapping [46] is an effect that occurs during the translational flight of the UAV (see Figure 2.6). It is caused by an imbalance in the relative speed of the rotor blade to the free stream. The advancing blade has a higher speed, thus it generates a higher lift. On the contrary, the retreating blade's lift is lowered. This effect can result in mechanical stress on the rotor.

For the quadrotor UAV, it can be described as

$$\tau_{F_x} = -4\left(k_\beta \cdot \alpha_y + \boldsymbol{F}_T^B \cdot \vec{z}^B \cdot h_R \cdot \sin\alpha_y\right) \approx -4\left(k_\beta + \boldsymbol{F}_T^B \cdot \vec{z}^B \cdot h_R\right) \cdot k_F \cdot \dot{y}^B, \tag{2.18}$$

$$\tau_{F_y} = -4\left(k_\beta \cdot \alpha_x + \boldsymbol{F}_T^B \cdot \vec{z}^B \cdot h_R \cdot \sin\alpha_x\right) \approx -4\left(k_\beta + \boldsymbol{F}_T^B \cdot \vec{z}^B \cdot h_R\right) \cdot k_F \cdot \dot{x}^B, \tag{2.19}$$

where τ_{F_x} and τ_{F_y} are torques around $\vec{x}^B$ and $\vec{y}^B$, respectively. A flap angle is described in each body axis as $\alpha_x = k_F \dot{x}^B$, where k_F is a constant relating the flapping to the apparent wind speed. A displacement of the rotor from the center of gravity along $\vec{z}^B$ is h_R and k_β is the stiffness of the rotor. It is possible to compensate for the resulting torque in attitude control. According to [49, 74] it is also possible to include the induced drag and the blade flapping effect into one lumped drag coefficient k_D and the result of aerodynamic effects leads to the force

$$\boldsymbol{F}_A^B = -\left(\sum_{i=1}^{4} F_i\right) \cdot \left(\boldsymbol{K}_D \dot{\boldsymbol{r}}^B\right)^\mathsf{T}, \tag{2.20}$$

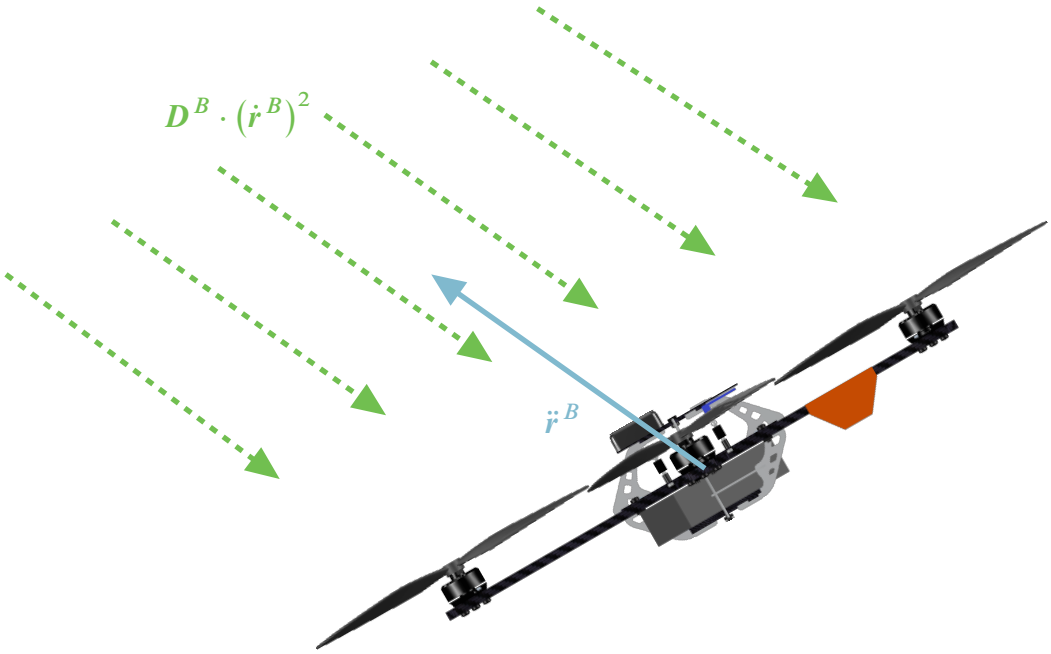


Figure 2.5: Induced drag

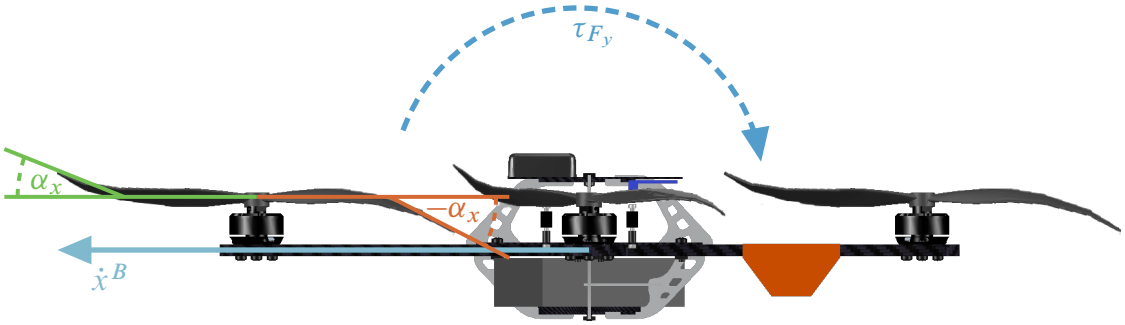


Figure 2.6: Blade flapping

where $\boldsymbol{K}_D = \mathrm{diag}\,(k_D, k_D, 0)$. The calculation of $\boldsymbol{K}_D$ is described in [74]. Because of this reduction, both the induced drag and the blade flapping can be compensated in the position control.

The aerodynamic interaction between neighboring rotors [89] also appears during the flight of UAV, but it is very problematic to model this behavior because of complex vortices. Other effects that are usually caused by the airflow through the rotors and by the changes of the airflow due to the environment are described in [47].

All mentioned forces and torques can be merged into the generalized force $\boldsymbol{F}^L$ and torque $\boldsymbol{\tau}$ that were mentioned in Section 2.2. Due to the description of forces in the body frame B, they must be rotated to the local coordinate frame L and thus $\boldsymbol{F}^L$ is defined as

$$\boldsymbol{F}^L = \boldsymbol{q} \otimes \left(\boldsymbol{F}_T^B + \boldsymbol{F}_A^B\right) \otimes \boldsymbol{q}^* = \boldsymbol{R}_{\phi\theta\psi}(\boldsymbol{q})^\mathsf{T} \left(\boldsymbol{F}_T^B + \boldsymbol{F}_A^B\right), \tag{2.21}$$

where $\boldsymbol{q}^*$ is the conjugate of quaternion. The torque $\boldsymbol{\tau}$ is simply the rotor inertia added to the collective torque

$$\boldsymbol{\tau} = \tau + [0,\, 0,\, \tau_{I_z}]^\mathsf{T}. \tag{2.22}$$

3 Path and Trajectory Planning for UAV

The goal of trajectory planning in robotics [59, 24] is to find a way how to get a robot from an initial pose to a final pose considering various constraints. The difference between a path and a trajectory is that the trajectory is a time-parametrized curve or a state trajectory of the robot; it contains not only its position, but also its orientation and their derivatives, etc. Each element of the trajectory is time-dependent, while the path is a function of position with a given direction from origin to destination and is time-independent.

The trajectory planning for UAVs is specific as they are usually not limited in movement within space, but they are extremely agile and their flight time is significantly limited. Therefore, it is advantageous to plan the trajectory of the UAV with available energy and time efficiency in mind.

This chapter will first introduce the concept of Configuration Space (C-space), which will be used to describe the UAV's area of operation. The C-space is a mathematical representation of the environment where the robot operates. It includes all possible configurations of the robot, including its position and orientation, and any obstacles present in the environment. Basic algorithms of both path and trajectory planning will afterward be presented. In particular, the Lazy Theta* algorithm will be described in the context of path planning. The trajectory planning problem will be formulated as an Optimal Control Problem (OCP) which numerical solution will also be discussed.

3.1 Configuration Space

In real-world applications of trajectory planning, it is crucial to represent the environment where the UAV moves. The environment can be described using the C-space. For ground navigation, only a 2D C-space is sufficient; however, for UAVs it is convenient to use a 3D C-space.

The 3D representation of space can be very demanding. In 2D, the environment is usually divided into $n \times n$ voxels; for each voxel, there is information about its occupancy. In 3D, such representation is problematic because the number of voxels steeply rises.

Some simplifications allow using the 2D C-space for 3D space. The first is called the height map and it additionally stores the height value for each voxel. This approach can be used for ground vehicles, but there is a problem for a UAV when the obstacle is above the ground. It is possible to store the next value in the second height map or to use even several height maps on top of each other. But even with a higher number of height maps, the C-space is still discretized in height, which is not suitable for precise trajectory planning or localization.

The mapping framework OctoMap [43] solves this problem by employing a structure called octrees storing the C-space. It allows to populate the C-space with more voxels but only in some limited space, which results in much fewer voxels in the whole space (see Figure 3.1).

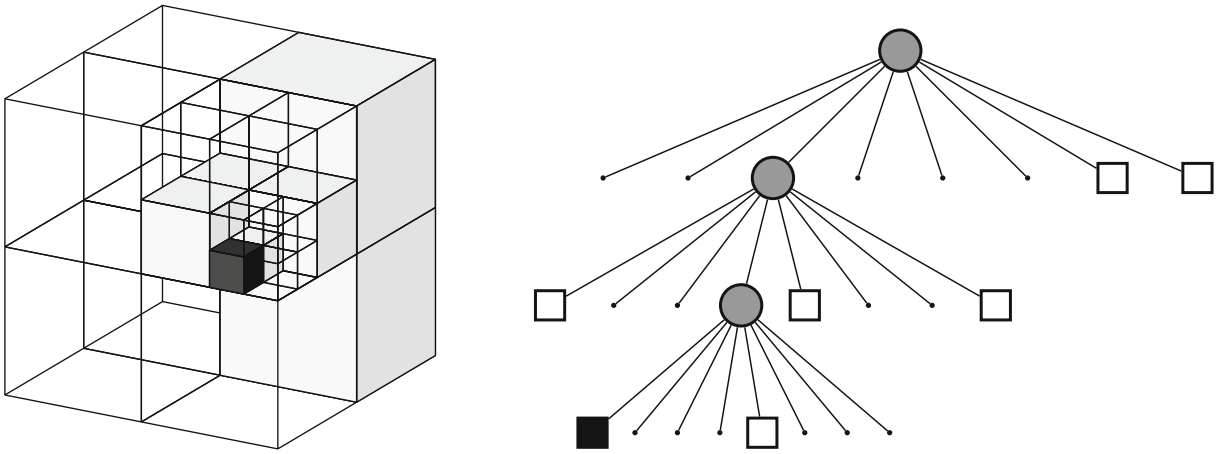


Figure 3.1: Octree which stores free (white) and occupied (black) cells. On the left, there is the volumetric model with the corresponding tree structure on the right. *Source:* [43]

3.2 Path Planning Algorithms

In this section, pathfinding algorithms [93, 96, 34, 7] will be presented that are applicable to 3D space and hence to quadrotor UAV path planning. These algorithms take into account the obstacles that were set in the C-space. Since only the obstacles are reflected, the resulting path must be smoothed and processed for feasibility according to the UAV dynamic model. After these adjustments, the trajectory can be obtained e.g. by the simulation of the UAV flight or derived for an approximate function.

First, the A* algorithm for optimal planning on a grid map will be presented. Next, the Theta* and Lazy Theta* algorithms will be introduced, which extend the A* algorithm with the ability to find a direct path between visible vertices. Then, the Rapidly-exploring Random Tree (RRT) algorithm and its variant Rapidly-exploring Random Tree* (RRT*) will be presented, which finds a path through the map based on randomly generated samples, i.e., it is not constrained by a grid, but the samples are generated freely according to a chosen probability distribution. Finally, the Artificial Potential Field method will be stated that represents the map using a sum of an attractive field to the goal and a repulsive field to the obstacles.

3.2.1 A*

Before presenting the A* algorithm, it is necessary to introduce its predecessor, the well-known Dijkstra's algorithm [29, 95]. Dijkstra's algorithm searches the shortest path to the goal; nevertheless, it considers only the *cost-to-here*, which is a cost estimate of the path from the start to the current vertex, for the evaluation of vertices and thus it is not very efficient.

The A* algorithm [27, 56] builds upon this foundation to improve efficiency and optimality. By incorporating heuristics and evaluating both *cost-to-here* and *cost-to-goal*, which is the cost estimate of a path from the current vertex to the goal given by heuristics, A* is able to make more informed decisions during pathfinding. By using the *cost-to-goal*, the number of states explored is reduced, resulting in improved convergence while ensuring solution optimality.

The A* operates over a grid, meaning that the resulting path is composed of vertices and edges of the gridmap. It results in an unrealistic-looking and non-smooth path for UAVs because it is determined purely by the gridmap. An example of how the A* algorithm finds a path through

a 2D gridmap is shown in Figure 3.2.

In Figure 3.2, it can be observed that the neighboring vertices of the start are evaluated, and then the vertex with the most favorable cost is further expanded and its neighbors are evaluated. However, the other expanded vertices remain in the priority queue and can be further expanded in the next iteration of the algorithm. Each expanded vertex is removed from the priority queue. If the algorithm reaches the goal, a path reconstruction is performed because information about the parent vertex is stored for each vertex.

Further, the costs, structures, and A* algorithm itself will be described in more detail. As was briefly mentioned, three values for each vertex in the A* are maintained:

- *cost-to-here*, a cost estimate of the path from the start to the current vertex
- *cost-to-goal* is the estimate of the path from the current vertex to goal, which is in A* given by heuristics
- *parent* of vertex employed in the extraction of the path from the start to the goal vertex after the termination of A*
- *cost-of-the-whole-path* is given as the sum of the *cost-to-here* and *cost-to-goal* and stands for the cost estimate of the whole path from the start to the goal

There are two main structures employed in A*:

- `open`, a priority queue with vertices for future expansion with the following functions:
  - `open.Insert` inserts a given vertex with its key, which is the *cost-of-the-whole-path*, into the queue
  - `open.Remove` removes a given vertex from the queue
  - `open.Pop` removes and returns the vertex with the smallest key
- `closed`, a set containing the vertices that have been already expanded

Further, the algorithm A* will be described; for completeness, the A* pseudocode is presented as Algorithm 2 in Appendix D.

1. The algorithm starts with an initialization where the following tasks are executed:

   For every vertex, the *cost-to-here* is set to infinity and the `parent` to `NULL` when it encounters the vertex for the first time.

   Further, the *cost-to-here* of the start vertex is set to zero and the `parent` to the start vertex itself, which provides an easy mechanism for the identification of the start.

   Structures `open` and `closed` are emptied and the start vertex is inserted into the `open` with its *cost-of-the-whole-path* as the priority key.

2. Subsequently, the A* algorithm launches the main loop:

   - In case the priority queue `open` is empty, the algorithm ends, because there are no more vertices to expand, and thus, the algorithm reports that there is no path.

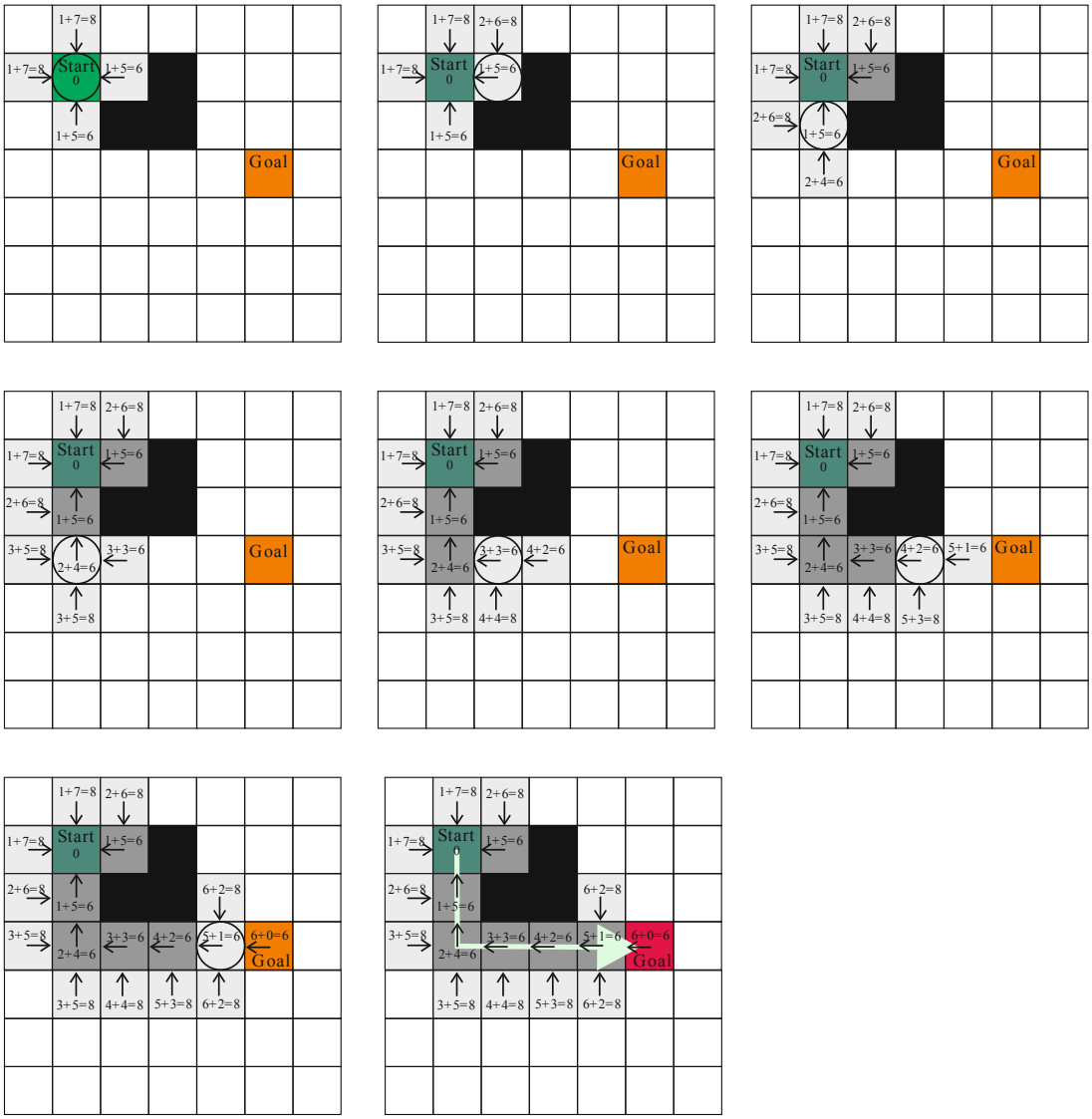
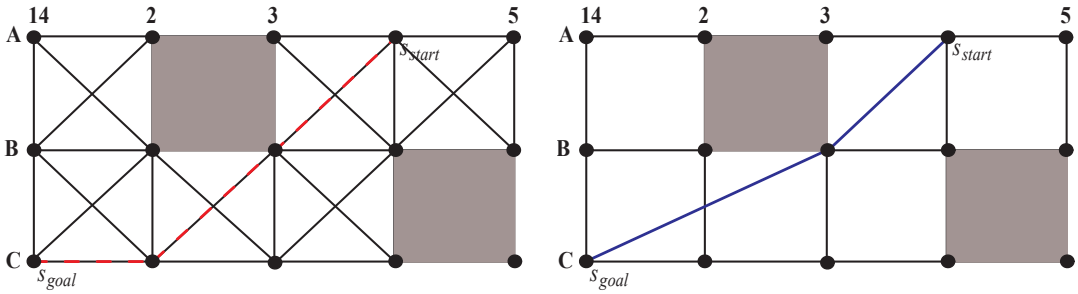


Figure 3.2: The start and goal vertices are labeled. The current vertex is marked with a circle, vertices in the priority queue `open` are light gray, and vertices in the set `closed` are dark gray, while obstacles are black. Transitions are possible in four directions (left, right, up, and down). In each visited vertex the direction to the parent vertex is denoted with an arrow. Each visited vertex contains the *cost-of-the-whole-path*, which is the sum of *cost-to-here* and *cost-to-goal*. For the cost function, the Manhattan distance is used. The found path can be tracked by following the connections marked with arrows. *Source:* [56]



(a) A* (b) Theta*

Figure 3.3: Path found by A* and Theta*. *Source:* [27]

- In case there are vertices in the queue `open`, A* searches the queue for the vertex with the smallest *cost-of-the-whole-path*; further, the check if the vertex is equal to the goal, is performed.
  - In case the equality with the goal is confirmed, A* has found the shortest path to the goal and it performs the path extraction by back-stepping from the goal to the start using information about the parents of vertices.
  - Unless the vertex equals the goal, it removes the vertex from `open` and inserts it into `closed`, while its unexpanded visible neighbors are expanded. The expansion is done because of future investigation of them by A* and it is executed as follows: it is tested if the current vertex's *cost-to-here* plus the cost to neighbor is smaller than the *cost-to-here* of neighbor, and if so, the neighbor's *cost-to-here* is set to the *cost-to-here* plus the cost from the current vertex to neighbor, the current vertex is set as the parent of neighbor and the neighbor is inserted into the `open` (in case it is already in `open`, only its *cost-of-the-whole-path* is updated). Further, the loop repeats.

3.2.2 Theta* and Lazy Theta*

Theta* algorithm [27, 70] is an extension of A* to allow off-grid movement, as pictured in Figure 3.3b. This leads to a smoother path containing fewer vertices when planning a path. Theta* tests whether two vertices are in a line of sight (no obstacle between them) even if they are not directly connected by a grid. The checks on the grid can be implemented with Bresenham's line drawing algorithm [19].

Theta* expands a vertex, but allows any already expanded vertex that is in sight to be its parent. Thus the algorithm develops two paths, a direct one as in A* and one between vertices in sight:

1. As in A*, Theta* considers the path from the start vertex to the current vertex and from the current vertex to its neighbor along a straight line

2. In the any-angle path, it considers the path from the start vertex to the parent of the current vertex and from the parent to the neighbor of the current vertex along a straight line.

The second path is considered in the case that the neighbor and the parent have line-of-sight, which is guaranteed to be no longer than the first path due to the triangle inequality. The *cost-to-here* and parent of a neighbor are updated only if the considered path is shorter than the shortest

path from the start vertex to the neighbor found so far. In Algorithm 3 in Appendix D, the pseudocode of COMPUTECOST function, which contains the changes compared to A*, is described.

The computational demands of Theta* can be addressed by a modification that allows a much smaller number of line-of-sight checks to be performed. In Lazy Theta* [70, 31], which is based on Theta*, the line-of-sight check occurs only once for each expanded vertex as opposed to Theta*, where the check was performed for each non-expanded visible neighbor of each expanded vertex. Theta* considers two paths of unexpanded visible neighbor, however, Lazy Theta* optimistically assumes, that the parent and neighbor have line-of-sight, and it prefers the second path. Therefore, it delays the line-of-sight check, which is performed immediately before expanding the vertex of the neighbor. In the case that the neighbor and its parent indeed have line-of-sight, the assumption is correct and the algorithm does not change the *cost-to-here* and the parent of the neighbor. Otherwise, they are altered according to the first path.

However, due to limited line-of-sight checks, the resulting Lazy Theta* path may be longer than that of Theta*. Therefore, the pros and cons must be considered according to the specific requirements of the application. In Algorithm 4 in Appendix D, the altered functions for Lazy Theta* are given.

3.2.3 Rapidly-exploring Random Trees

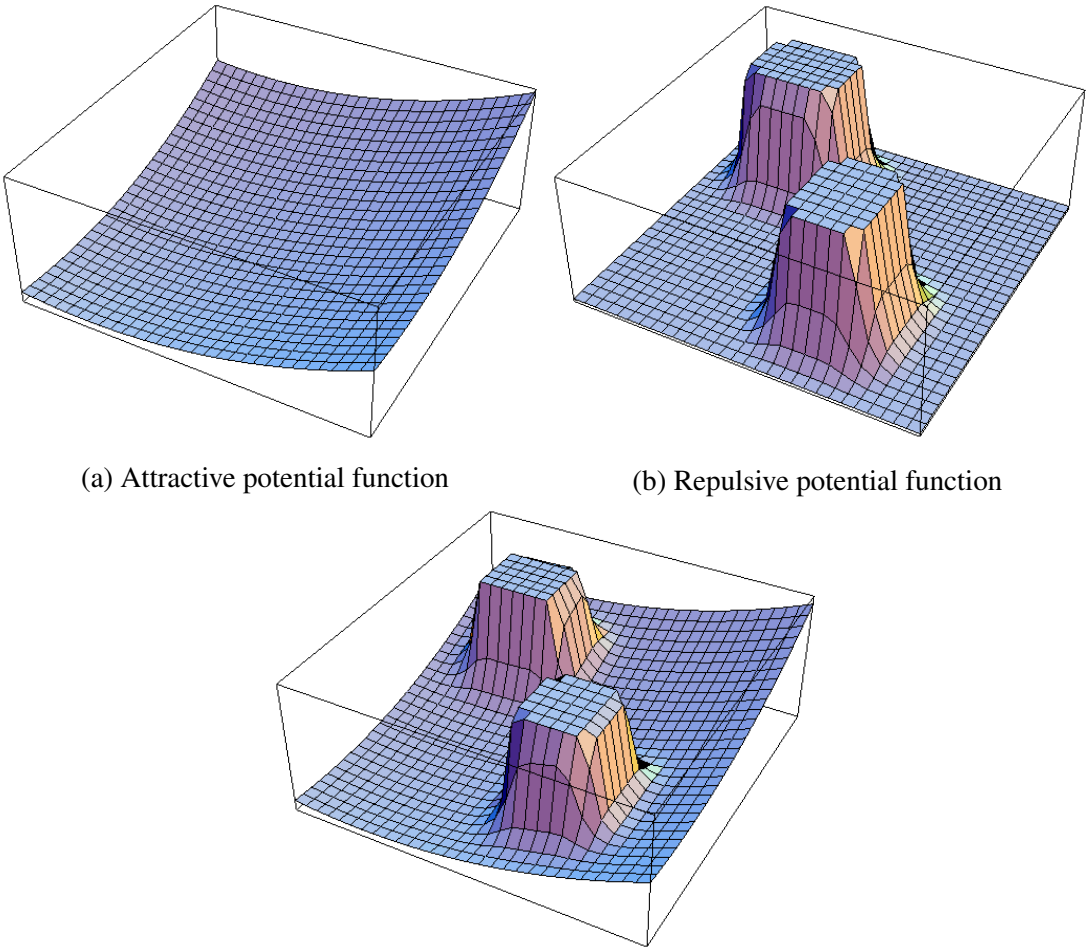
As opposed to the previously mentioned algorithms, RRT [50, 93] is an active sampling or Monte Carlo-based algorithm. The path is not searched at strictly given gridmap points, but it is sought using sampled particles. Thanks to the sampling, the RRT is applicable directly even in the continuous space. Sampling distribution is usually chosen based on e.g. the size of the area of operation. The RRT can adjust the sampled random particle according to given limitations keeping the reachability of the UAV. The algorithm is executed as follows.

1. At the start, the initial state is used as the first vertex and then a random vertex is sampled.
2. The vertex nearest to the new sampled random vertex is found based on given metrics.
3. The random vertex at this point only gives an idea about the direction where the next step should go. To ensure the reachability by the UAV, the control input factor is added considering the constraints of the problem using the cost function. As a result, the altered new reachable vertex is obtained and if it is located in the free space (outside the obstacles), it is added to the tree with the previously found nearest vertex as a parent. The algorithm repeats with the sampling of a new random vertex.

Due to the sampling, the RRT suffers from problematic convergence and sometimes unrealistically looking resulting path. Those problems are tackled in the extension called RRT* [50, 93], which provides procedures for processing the path. As a result, the RRT* provides the asymptotic optimal property as opposed to RRT [93]. In RRT*, the parent of the new reachable vertex is chosen as the nearest known vertex and the feature called *rewire* reconnects vertices in the tree to keep it dense and more compact, which usually results in the overall minimal cost of the path. The pseudocode of RRT* is presented in Algorithm 5 in Appendix D.

3.2.4 Artificial Potential Field Method

Artificial Potential Field Method [23, 93, 22] was proposed in [54] and it has been widely used due to its low computational complexity [93]. It is based on assigning a potential function



(a) Attractive potential function

(b) Repulsive potential function

(c) Total potential function

Figure 3.4: Potential functions. *Source:* [23]

to the free space and simulating the robot as a particle that reacts to force due to the potential field, which can be denoted as

$$U_t(\boldsymbol{x}) = U_a(\boldsymbol{x}) + U_r(\boldsymbol{x}), \tag{3.1}$$

where $U_t(\boldsymbol{x})$, $U_a(\boldsymbol{x})$, and $U_r(\boldsymbol{x})$ are the total, attractive, and repulsive potential functions at state $\boldsymbol{x}$, respectively. The attractive potential causes motion towards the goal (see Figure 3.4a). On the contrary, the repulsive potential results in avoiding the obstacles (see Figure 3.4b). Visualization of the total potential function is depicted in Figure 3.4c.

The motion of a robot is given by a total virtual force, which can be described as a negative gradient of the total potential function

$$f_t(\boldsymbol{x}) = -\nabla U_t(\boldsymbol{x}) = -\nabla U_a(\boldsymbol{x}) - \nabla U_r(\boldsymbol{x}) = f_a(\boldsymbol{x}) + \sum_{i=1} f_{r,i}(\boldsymbol{x}), \tag{3.2}$$

where $f_a(\boldsymbol{x})$ is an attractive virtual force and $f_{r,i}(\boldsymbol{x})$ is a restrictive virtual force of the i-th obstacle.

The application of the method for UAV was described in [22] where it was transformed to the OCP and the constraints were handled using slack variables.

3.2.5 Conclusion of Path Planning

Section 3.2 presented various methods for path planning. The first set of methods operates over the gridmap. The A* algorithm is capable of finding an optimal path, but the path between vertices is limited to direct neighbors, which can result in a nonsmooth path. Therefore, Theta*, a method that builds upon A*, was presented. Theta* is capable of finding a path between visible vertices, making it an improvement over A*. A variant of Theta* known as Lazy Theta* was also introduced, which restricts the number of line-of-sight checks. Theta* and LT* are particularly useful for UAV path planning as they can find a direct path with a small number of vertices.

Additionally, the section introduces the RRT* algorithm, which can find a path through the map based on randomly generated samples. The algorithm generates samples freely based on a chosen probability distribution, resulting in more realistic paths. Unlike the RRT, the RRT* solves the convergence problem by rewiring the tree of vertices.

Additionally, the Artificial Potential Field method was presented, which represents the map using a sum of an attractive field towards the goal and a repulsive field away from the obstacles. The movement of the UAV is determined by a virtual force, which can be defined as the negative gradient of the total potential field function.

3.3 Trajectory Planning via Optimal Control Problem

For robots with slower dynamics, the path itself can be used as a reference for the motion commander. For UAVs, however, this is problematic in terms of their considerable agility, but also inertia, where the UAV must exert considerable energy when reaching a waypoint without a more precise trajectory plan.

The path can be interlaced with a spline and the trajectory can be constructed based on kinematic constraints, but this may not be a good approximation of the UAV dynamics [34]. For the reasons mentioned above, the trajectory planning problem will be described in this thesis as OCP, where the behavior of the UAV can be fully described using differential equations.

According to the taxonomy presented in [93], OCP is a mathematical model-based method, which searches for the optimal trajectory according to the objective function considering the environment (kinematic constraints) and dynamic constraints. In other words, the optimal trajectory is the solution to OCP. Unlike path planning methods, which often require post-processing of the path, the trajectory found by the OCP inherently satisfies the requirements given by the UAV physical properties contained in the dynamics of the UAV model, and the state and control constraints. Moreover, the trajectory is optimal according to the criterion.

The optimal trajectory can be sought as the solution to OCP [1, 3, 8, 79] which is known to be solved by Boundary Value Problem (BVP) [11]. In the OCP, a time-continuous control trajectory is sought that minimizes the optimality criterion. First, a general OCP satisfying the nonlinear optimality criterion for a nonlinear continuous-time system with nonlinear constraints will be described. Further, several methods that provide a numerical solution to the OCP

will be presented. The OCP is in this case formulated in the following manner

$$J\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t_0,t_f\right) = \Phi\left(\boldsymbol{x}(t_0),\boldsymbol{x}(t_f),t_0,t_f\right) + \int_{t_0}^{t_f} \mathfrak{L}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t\right) dt, \tag{3.3}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}(t) = f\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t\right), \tag{3.4}$$

$$0 \leq h\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t\right), \tag{3.5}$$

$$\boldsymbol{x}(t) \in \mathfrak{X}, \tag{3.6}$$

$$\boldsymbol{u}(t) \in \mathfrak{U}. \tag{3.7}$$

The optimality criterion (3.3) is composed of the cost function Φ associated with the initial and final state, and the integral cost function $\mathfrak{L}$, where $x(t)$ and $u(t)$ are the state and control at time instant t, respectively. The beginning of the trajectory is at the time instant t_0 and it ends at t_f.

The solution to the OCP can be found using Hamiltonian canonical equations, where the Hamiltonian is defined as

$$H(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{\lambda}(t),t) = \mathfrak{L}(\boldsymbol{x}(t),\boldsymbol{u}(t),t) + \boldsymbol{\lambda}^{\mathsf{T}}\boldsymbol{f}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t\right) \tag{3.8}$$

with $\boldsymbol{\lambda}$ denoting a vector of the Lagrange multipliers. The Hamilton canonical equations are then given as

$$\dot{\boldsymbol{x}}(t) = \left(\frac{\partial H}{\partial \boldsymbol{\lambda}}\right)^{\mathsf{T}} = f(\boldsymbol{x}(t),\boldsymbol{u}(t),t), \text{with initial condition } \boldsymbol{x}(t_0), \tag{3.9}$$

$$\dot{\boldsymbol{\lambda}}(t) = -\left(\frac{\partial H}{\partial \boldsymbol{x}}\right)^{\mathsf{T}} = -\left(\frac{\partial \mathfrak{L}}{\partial \boldsymbol{x}}\right)^{\mathsf{T}} - \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)^{\mathsf{T}} \boldsymbol{\lambda}, \tag{3.10}$$

$$0 = \left(\frac{\partial H}{\partial \boldsymbol{u}}\right)^{\mathsf{T}} \tag{3.11}$$

with the terminal condition for Equation (3.10) given by the transversality condition [11, 3]

$$\left[\left(\frac{\partial \Phi}{\partial \boldsymbol{x}} - \boldsymbol{\lambda}^{\mathsf{T}}\right) d\boldsymbol{x}\left(t_f\right) + \left(\frac{\partial \Phi}{\partial t} + H\right) dt_f\right]\Bigg|_{t=t_f} = 0. \tag{3.12}$$

These equations pose a BVP. After finding the extremal control trajectory, a sufficient condition is to check whether a minimum or maximum is found.

If the constraints for the control trajectory are involved, the optimal trajectory is sought using the Pontryagin minimum principle [79, 3] in eq. (3.13). Equation (3.11) is substituted according to the Pontryagin minimum principle with equation

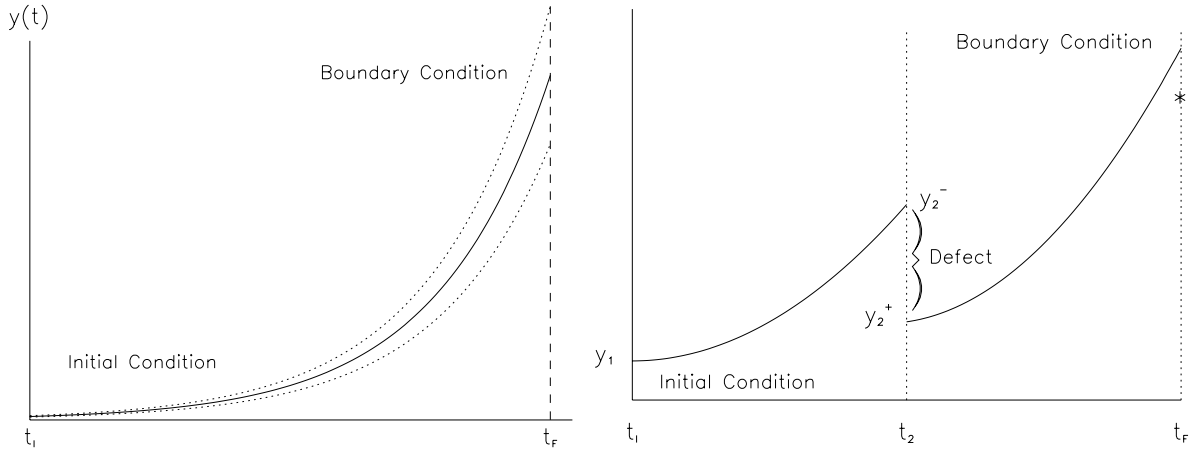
$$\boldsymbol{u}^*(t) = \underset{\boldsymbol{u}(t)\in\mathfrak{U}\subset\mathsf{R}^m}{\operatorname{argmin}} \; H\left(\boldsymbol{x}^*(t),\boldsymbol{u}(t),\boldsymbol{\lambda}^*(t),t\right), \tag{3.13}$$

which is a necessary condition.

Employment of inequality trajectory constraints [8] given by

$$0 \leq \boldsymbol{h}(\boldsymbol{x}(t),\boldsymbol{u}(t),t) \tag{3.14}$$

can be very difficult. The inequality constraints may be active ($0 = h$) or inactive ($0 < h$), which may lead to difficulties because the number of intervals with active and inactive constraints is not a priori known. At the junction points, where the status of the constraint changes, the control, and Lagrange multipliers trajectories are discontinuous. That can cause the two-point BVP becomes a multi-point BVP. Because of the difficulty of obtaining a direct analytical solution, it may be advantageous to use methods that seek the solution numerically.



(a) Single-shooting method

(b) Multiple-shooting method

Figure 3.5: Single and multiple shooting method. *Source:* [8]

3.3.1 Direct and Indirect Methods

Numerical methods that are suitable for determining the OCP solutions are presented in this section. There are many ways to classify this vast and well-studied field; however, the following classification is tailored to the specific needs of this thesis. There are two major classes of numerical approaches, *direct* and *indirect methods*. The class of indirect methods contains indirect shooting methods, where the BVP is solved based on the guess of the initial condition and the subsequent check of the solution validity. On the contrary, the direct methods are based on an approximation of functions and the transcription of the problem to the Nonlinear Program (NLP), and therefore they are sometimes called *transcription methods*.

In the OCP, the terminal point of Lagrange multipliers $\boldsymbol{\lambda}(t_f)$ is given by the transversality condition (3.12) and their dynamics is described by Equation (3.10). For the state dynamics (3.9), an initial condition $\boldsymbol{x}\ (t_0)$ is given, but to solve the OCP the Lagrange multipliers trajectory must be obtained. Lagrange multipliers have only a terminal boundary condition (3.12) and dynamics (3.10), therefore their starting point is unknown.

The indirect shooting methods [8, 45, 44, 52] guess the unknown initial Lagrange multipliers $\boldsymbol{\lambda}(t_0)$. After guessing, the problem is propagated using Equation (3.10) with initial guess and conditions towards the terminal point. If the terminal point of the solution is not equal to the one given by the transversality condition, the guess is slightly changed. The main drawback is that even a small change of guess may cause a large change at the end-point, which results in the uncertain convergence of the method.

The shooting methods can be further divided into single and multiple shooting methods (see Figure 3.5). The single-shooting method (see Figure 3.5a) corresponds to the previous general description of the shooting methods. The multiple-shooting method is based on the breaking of the OCP into several intervals, which is illustrated in Figure 3.5b. Subsequently, the OCP is solved separately for each interval and the continuity of the solution is enforced by employing constraints, which are called *defect* constraints. Separating the OCP into several smaller problems can significantly help with the convergence of the solution [8].

On the other hand, there are direct methods, which are based on the approximation of nonlinear functions describing the OCP. Using the approximation of nonlinear functions, it is easier to evaluate the integral in the optimality criterion and the derivative in the dynamics of the UAV.

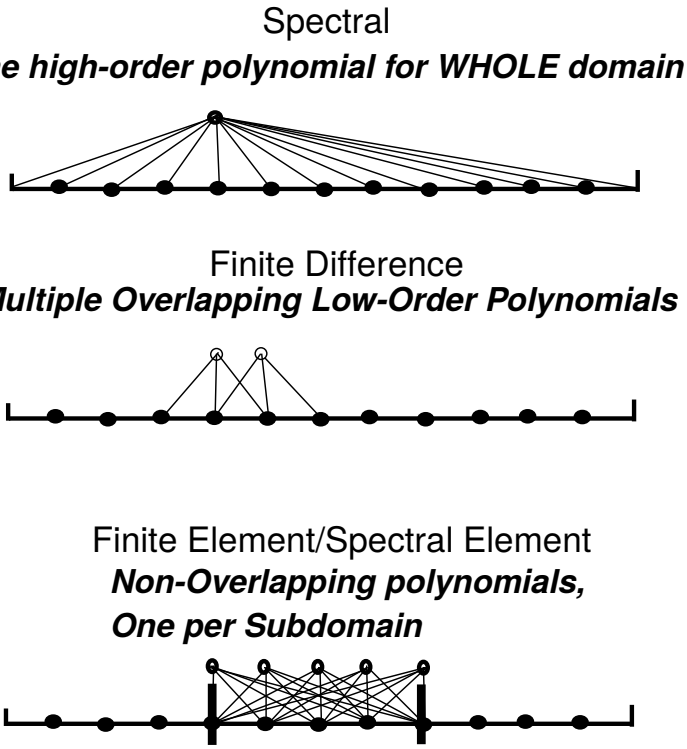


Figure 3.6: Subdomains in Spectral, Finite Difference, and Finite Element Methods. *Source:* [17]

Employing the approximation of functions, and their integrals and derivatives, it is possible to transcribe the OCP into the NLP, which is afterward solved by an NLP solver. The direct methods can be divided into spectral methods, Finite Element Method (FEM), and Finite Difference Method (FDM). These methods differ in the domain of approximation (see Figure 3.6). The spectral method approximates function by a single high-degree polynomial. On the other hand, in FEM and FDM, the approximations are composed of a large number of low-degree polynomials.

Thanks to the simplicity of FEM and FDM, they are easily implementable; however, very accurate results require usually a large number of functions, which causes high-dimensionality of the problem. On the other hand, the spectral method can deliver a much more accurate solution than other direct methods with comparable complexity [17]. In addition, there is a method that combines both approaches, where the problem is approximated by several high-degree polynomials. This method is referred to as the Spectral Element Method or *h-p* element method (*h* is a degree of polynomial and *p* is a number of segments/polynomials) in the literature [17, 51, 94].

3.3.2 Spectral Method in Optimal Control Problem

The spectral method is a special case of the Mean Weighted Residual Methods (MWRMs) [17]. The MWRMs use an approximation of the unknown function $f(t)$, which can be expressed as

$$f(t) \approx f_N(t) = \sum_{n=0}^{N} a_n \nu_n(t) \tag{3.15}$$

and it is composed of the basis functions $v_n(t)$, $n = 0, \ldots, N$, and spectral coefficients a_n. When it is substituted into the equation

$$Gf = g(t), \tag{3.16}$$

where G is a differential or integral operator and g is the approximated function, the result is the residual function

$$\varepsilon(t; a_0, a_1, \ldots, a_N) = Gf_N - g, \tag{3.17}$$

where f_N is the approximation of function g. By minimizing the residual function, the MWRM determines the coefficients a_n of the basis functions. If the basis function $v_n(t)$ already satisfies the boundary conditions, the MWRM determines the spectral coefficients a_n by imposing $N + 1$ conditions

$$\langle \eta_i, \varepsilon\,(t; a_0, a_1, \ldots, a_N)\rangle = 0,\ i = 0, 1, \ldots, N \tag{3.18}$$

for some suitable test function $\eta_i(t)$, where the inner product is defined by

$$\langle u, v\rangle = \int_a^b \rho(t)u(t)v(t)dt \tag{3.19}$$

with a given non-negative weight function $\rho(t)$ and arbitrary two functions $u(t)$ and $v(t)$.

The name of the spectral methods is given by the fact that the set of basis functions usually consists of the Fourier series or Chebyshev polynomial. Spectral methods contain Galerkin and Petrov-Galerkin methods. The Galerkin method [17] is a special case of the MWRM, where the basis and test functions are equal, i.e.

$$v_i(t) = \eta_i(t).$$

The Petrov-Galerkin method is the case where the equality of the functions is not preserved.

There is no simple guide for choosing the best basis function set. Usually, the sets are composed of functions, that are easy to evaluate, e.g. trigonometric functions and polynomials [17]. A very important property of the basis function is its completeness, i.e. $a_N = 0$ with $N \to \infty$. The property, which may improve the convergence of the solution is the orthogonality of the basis, which can be described as $\langle v_i, v_j\rangle = 0, i \neq j$. A suitable choice of the basis and therefore test functions is often given by the boundary conditions of the problem. A guide for the choice of the most suitable function according to boundary conditions is depicted in Figure 3.7. A detailed description of various basis functions is written in [17].

According to [17], an analysis of the Galerkin method convergence is a non-trivial problem, however, few hints exist. For example, the convergence can be observed through spectral coefficients, which should decrease with N, if N is very high. Also, the convergence can be analyzed on the function with similar properties as the investigated function, because it should have a similar convergence of the approximation error. The convergence is tightly coupled with the singularities of the approximated function. If the coefficients are calculated numerically, the error will most likely not decrease after some N, but it will stay around a non-zero value. This effect is caused by the precision of the computer.

Since the special case of the Galerkin method will be studied in the following section in more detail, the Galerkin method itself will be demonstrated on a simple example from [17] with a solution to the differential equation

$$\frac{\partial^2 f}{\partial t^2} - \frac{1}{2}f(t) = -\frac{3}{2}\cos(t) - \frac{9}{2}\cos(2t) \tag{3.20}$$

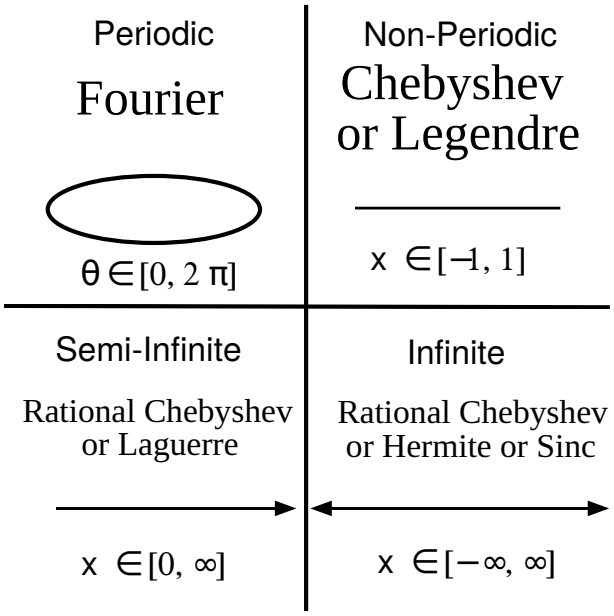


Figure 3.7: Guide for choosing a suitable basis function according to the domain of the problem. *Source:* [17]

with periodic boundary conditions.

Because of the boundary conditions periodicity, the basis function will be chosen as a Fourier cosine series without constant. The form of solution is

$$f_2(t) = a_1 \cos(t) + a_2 \cos(2t). \tag{3.21}$$

It is possible to rewrite the differential equation into form $Gf_2 = g$, where

$$G = \frac{\partial^2}{\partial t^2} - \frac{1}{2}, \qquad g(t) = -\frac{3}{2}\cos(t) - \frac{9}{2}\cos(2t) \tag{3.22}$$

The matrix form of inner product (3.19) for the relations from (3.22) can be denoted as

$$\begin{bmatrix} \langle \cos(t), G\cos(t)\rangle & \langle \cos(t), G\cos(2t)\rangle \\ \langle \cos(2t), G\cos(t)\rangle & \langle \cos(2t), G\cos(2t)\rangle \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \langle \cos(t), g\rangle \\ \langle \cos(2t), g\rangle \end{bmatrix}, \tag{3.23}$$

where

$$\langle g, h\rangle = \int_{-\pi}^{\pi} g(t)h(t)dt \tag{3.24}$$

for any two functions $g(t)$ and $h(t)$, where the weight function was chosen as $\rho(t) = 1$. Elements of the matrix (3.23) are given as

$$G\cos(nt) = \frac{\partial^2 \cos(nt)}{\partial t^2} - \frac{1}{2}\cos(nt) = -\left(n^2 + \frac{1}{2}\right)\cos(nt), \ \forall n$$

for the given set of basis functions the inner product is

$$\langle \cos(mt), \cos(nt)\rangle = \pi\delta_{mn}, \ \forall m, n > 0, \tag{3.25}$$

where $\delta_{mn} = \begin{cases} 1, & m = n \\ 0, & m \neq n \end{cases}$ is the Kronecker delta. After the substitution into Equation (3.23), the resulting matrix form is

$$\begin{bmatrix} -\frac{3}{2} & 0 \\ 0 & -\frac{9}{2} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -\frac{3}{2} \\ -\frac{9}{2} \end{bmatrix}. \tag{3.26}$$

Finally, the spectral coefficients are

$$a_1 = 1,\ a_2 = 1 \tag{3.27}$$

and the solution is

$$f_2(t) = \cos(t) + \cos(2t). \tag{3.28}$$

The solution acquired by the Galerkin method is exact because of the appropriate choice of the basis functions.

3.3.3 Pseudospectral Methods

In the Galerkin method, it is necessary to evaluate the inner product, which is given in the form of the integral (3.19). That can be calculated analytically or using a method for numerical calculation of integral. Nevertheless, the integral calculation can be avoided by employing the Pseudospectral Method (PSM) [17, 91, 84] which uses the spectral function only as a basis function, but the test function is composed of a set of collocation points. It means that the function is tested at these points only. In comparison to the Galerkin method, the solution is found much faster at the cost of a slightly bigger error of approximation. For the completeness, the test function is given as

$$\eta_i(t) = \delta(t - t_i), \tag{3.29}$$

where t_i is the i-th collocation point. When collocation points are not chosen properly, Runge's phenomenon [94] can occur, where the approximation error can increase significantly as the number of points increases. By using optimal collocation points, Runge's phenomenon can be avoided. Each basis function has optimal locations of collocation points, which usually depend on the boundaries of the problem (if they are included in the approximation) and their quantity. The locations of the points significantly influence the convergence of the solution. In Figure 3.8, an approximation of function $f(t) = 1/(1 + 16t^2)$ is shown. With the correct choice of the collocation points, the convergence of approximation is granted. In the case of equidistant points, the solution converges in the center but diverges near the boundaries.

The problem of the suitable basis function set is the same as in the Galerkin method. Thus the same rules from Section 3.3.2 can be applied and the same hints can also be applied for the convergence of the method. The problem can be transformed into the domain of basis function [84]. Generally used basis functions for PSM are the Chebyshev and Legendre polynomials.

The PSM will be presented using an example from [17], the differential equation

$$\frac{\partial^2 f}{\partial t^2} - \frac{1}{2} f(t) = -\frac{3}{2}\cos(t) - \frac{9}{2}\cos(2t) \tag{3.30}$$

with periodic boundary conditions and it will employ the Chebyshev polynomials as a set of basis functions. The collocation points for $N = 9$ are defined as

$$t_k = \cos\left(\frac{\pi k}{N-1}\right), k = 0, \ldots, N-1 \tag{3.31}$$

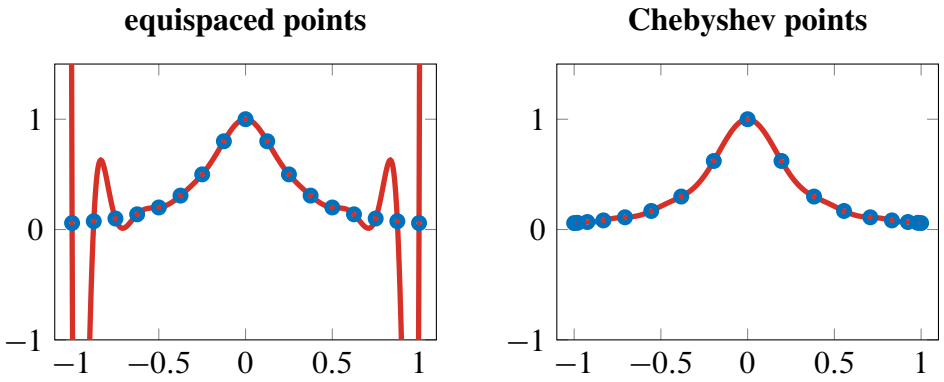


Figure 3.8: Equidistant and Chebyshev collocation points interpolation of $f(t) = 1/(1 + 16t^2)$

and the right-hand side of the equation is given as

$$g(t) = -\frac{3}{2}\cos(t) - \frac{9}{2}\cos(2t). \tag{3.32}$$

The equation can be rewritten as

$$\boldsymbol{D}_9^2 \boldsymbol{f}_9 - \frac{1}{2}\boldsymbol{f}_9 = g(\boldsymbol{t}), \tag{3.33}$$

where $\boldsymbol{D}_9^2$ is a difference matrix for the second derivative with 9 collocation points. The solution is afterward acquired as

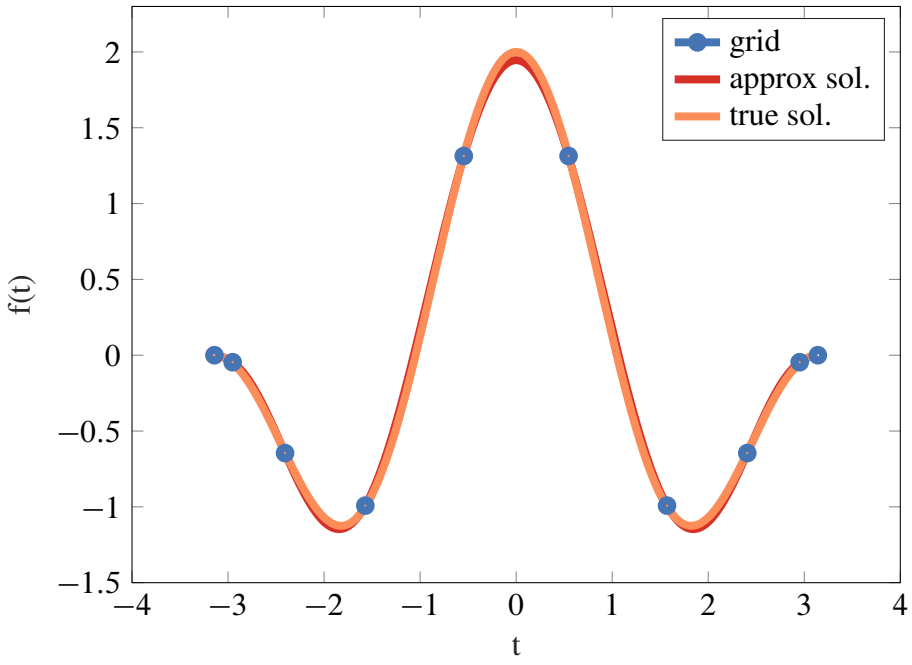
$$\boldsymbol{f}_9 = \left(\boldsymbol{D}_9^2 - \frac{1}{2}\boldsymbol{I}\right)^{-1} g(\boldsymbol{t}), \tag{3.34}$$

where $\boldsymbol{f}_9$ is a vector of values at the collocation points. The continuous solution (the so-called *modal* representation) can be obtained by fitting $\boldsymbol{f}_9$ with a polynomial. The solution is depicted in Figure 3.9a along with the exact solution. The approximation error is shown in Figure 3.9b, where almost zero error can be observed at the collocation points with an increase far from them. Rapid fluctuations of the error are caused by the fact that the test function fits the approximation at the collocation points. In Figure 3.9b, the approximation is more precise near the boundaries, which is an important property of the Chebyshev polynomial.

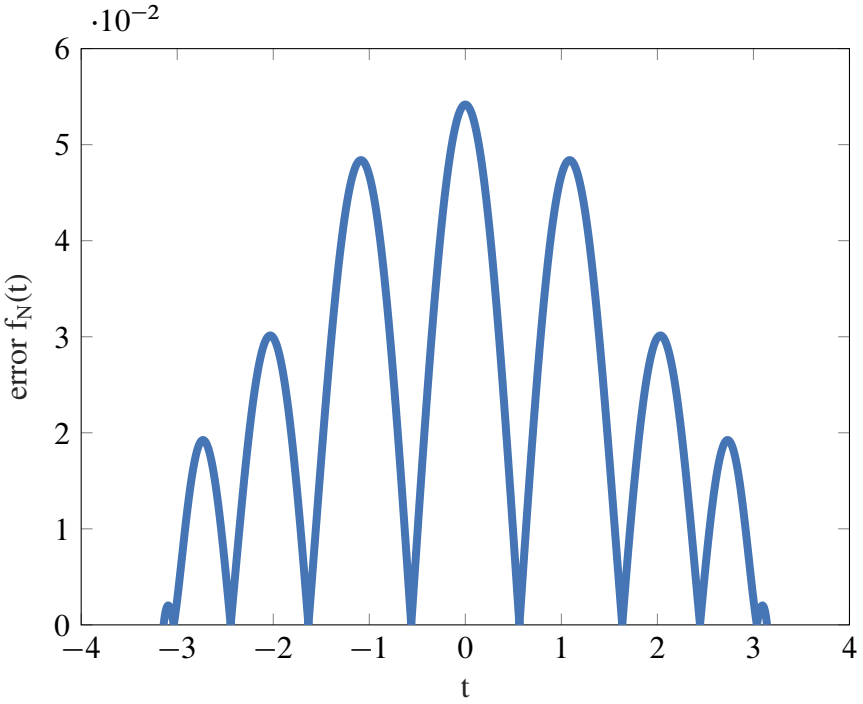
3.3.4 Transcription of OCP to NLP employing Pseudospectral Methods

Thanks to the approximation using the PSM, the OCP can be transcribed to the NLP. That is possible because the OCP is evaluated only at the collocation points; between them, the approximation is considered. The criterion is evaluated using integral weights from the PSM. The derivative of the state at the collocation points is evaluated using a difference matrix.

First, the NLP will be presented. As opposed to the OCP, variables are real numbers, the dimension of the problem is finite and it is described using algebraic equations [52]. A general



(a) Solution



(b) Error of solution

Figure 3.9: Solution to the differential equation by PSM with Chebyshev polynomial and its error

NLP can be denoted as

$$\min_{\boldsymbol{z}} f\left(\boldsymbol{z}\right), \tag{3.35}$$

$$g\left(\boldsymbol{z}\right) \leq 0, \tag{3.36}$$

$$h\left(\boldsymbol{z}\right) = 0, \tag{3.37}$$

$$\boldsymbol{z}^{-} \leq \boldsymbol{z} \leq \boldsymbol{z}^{+}, \tag{3.38}$$

where $\boldsymbol{z}$ is a vector of variables; g and h are inequality and equality constraints, respectively. The upper and lower bounds of $\boldsymbol{z}$ are given by vectors $\boldsymbol{z}^{+}$ and $\boldsymbol{z}^{-}$, respectively.

The PSM OCP with variable boundary points, equality constraints, inequality constraints, and nonlinear dynamics approximated by a polynomial of N-th degree can be described as

$$\boldsymbol{z} = \left[t_0, t_N, \boldsymbol{x}_0^{\mathsf{T}}, \ldots, \boldsymbol{x}_N^{\mathsf{T}}, \boldsymbol{u}_0^{\mathsf{T}}, \ldots, \boldsymbol{u}_N^{\mathsf{T}}\right]^{\mathsf{T}}, \tag{3.39}$$

$$\min_{\boldsymbol{z}} \Phi\left(\boldsymbol{x}_0, \boldsymbol{x}_N, t_0, t_N\right) + \sum_{k=0}^{N} w_k \mathfrak{L}\left(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k\right), \tag{3.40}$$

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} = \boldsymbol{D}_N \begin{bmatrix} \boldsymbol{x}_0 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} - \begin{bmatrix} f\left(\boldsymbol{x}_0, \boldsymbol{u}_0, t_0\right) \\ \vdots \\ f\left(\boldsymbol{x}_N, \boldsymbol{u}_N, t_N\right) \end{bmatrix}, \tag{3.41}$$

$$g\left(\boldsymbol{z}\right) \leq 0, \tag{3.42}$$

$$h\left(\boldsymbol{z}\right) = 0, \tag{3.43}$$

$$\boldsymbol{z}^{-} \leq \boldsymbol{z} \leq \boldsymbol{z}^{+}, \tag{3.44}$$

where $\boldsymbol{z}$ includes the boundary time points, state vector, and control vector for N collocation points. The integral weights are denoted as w_k and $\boldsymbol{D}_N$ is the difference matrix for the N-th degree approximation. The constraints in the transcribed OCP are tested only at the collocation points, so it can be convenient to test if the constraints are not violated outside the collocation points. If a serious violation is found, the so-called *mesh-refinement* can be applied. It means that the previous solution is used as an initial guess for the NLP solver and the OCP is solved with a higher-degree approximation (i.e. with more collocation points), which results in a more precise approximation of the system behavior, but also more frequent checks of constraints.

The general procedure for obtaining a solution to the OCP will be briefly described. First, the problem is described in the manner of the OCP (3.3)-(3.7). The number of collocation points N is chosen and the initial guess for the state trajectory $\boldsymbol{x}(t)$ and the control trajectory $\boldsymbol{u}(t)$ is chosen. Further, the interval of end-time is chosen. According to N, the difference matrix $\boldsymbol{D}_N$ and the integral weights w_k are generated. The initial guess is refined to the mesh of N collocation points. Afterward, the OCP is transcribed into the NLP (3.39)-(3.44). The resulting solution to the NLP is provided by the NLP solver. The solution is extracted from the NLP variable $\boldsymbol{z}$ and the error of the solution is tested. The error of the solution can be evaluated e.g. by the discretization error or by testing the constraints outside the collocation points. If the error is too high, then N is increased, the current solution is employed in place of the initial guess and the solution to the problem starts over with the generation of new collocation points.

3.3.5 Spectral Element Method

As mentioned in Section 3.3.1, it may be advantageous to search for a solution using the Spectral Element Method [17, 51, 94], which combines the best of both worlds; the FEM and Spectral Method. The problem is divided into several interrelated domains as in FEM. However, high-degree approximations are used in these domains as in spectral methods (see Figure 3.6). The Spectral Element Method may be advantageous in terms of further convergence improvement.

To ensure the solution remains continuous across domains, a patching is employed. It is necessary to verify that the state $\boldsymbol{x}(t)$ and its derivatives $\frac{d\boldsymbol{x}(t)}{dt}$ are equal between neighboring domains. For the OCP, the patching can be achieved by imposing additional equality constraints:

$$\boldsymbol{x}^{(i)}\left(t_f^{(i)}\right) = \boldsymbol{x}^{(i+1)}\left(t_0^{(i+1)}\right), \tag{3.45}$$

$$f\left(\boldsymbol{x}^{(i)}\left(t_f^{(i)}\right), \boldsymbol{u}^{(i)}\left(t_f^{(i)}\right), t_f^{(i)}\right) = f\left(\boldsymbol{x}^{(i+1)}\left(t_0^{(i+1)}\right), \boldsymbol{u}^{(i+1)}\left(t_0^{(i+1)}\right), t_0^{(i+1)}\right), \tag{3.46}$$

where superscripts (i) and $(i+1)$ denote i-th and $i+1$-th domains. Both integral Galerkin methods and collocation pseudospectral methods can be used in this way.

In [17], it is noted that it is advantageous to solve the problem in two phases; first, solve several uncoupled smaller problems, and in the second phase link the individual solutions together into a single continuous global domain.

3.4 Analysis and Conclusion of Trajectory Planning

In this chapter, various methods involved in trajectory planning were presented. These methods were divided into two general classes: path planning and OCP-based methods. Path planning methods are advantageous due to their utilization of the map, which can be reflected during tree building, and their low computational demands. The strength of OCP-based methods lies in their model-based approach, where dynamics is described using differential equations and the optimality of the sought trajectory is ensured through the OCP.

The usefulness of the methods is in particular given by the specifics of the application. In the UAV field, the computationally cheap solutions are usually preferred for the sake of keeping the autonomy of UAV, as most of the calculations can be performed on-board. In general, for autonomous flight mission planning, path planning algorithms such as Lazy Theta* for the discretely described map or RRT* coping with the continuous space should be the first choice.

However, the OCP-based approach, which is more computationally demanding than the path planning methods, can directly incorporate the complex dynamics of the UAV. This can be particularly important in applications where the UAV must be steered with extreme precision, such as fast flights with collision avoidance or when managing tools that are on-board the UAV, where precise movements are required for successful operation, or even including the behavior of these tools into the OCP.

4 Trajectory Tracking for UAV

In Chapter 3, the trajectory and its planning were presented. In this chapter, the problem of trajectory tracking will be discussed. First, the problem will be outlined. Further, the state-of-the-art method, called Model Predictive Control (MPC) will be presented. Finally, an alternative approach to the MPC, the Interpolating Control (IC), will be described in detail.

The main goal of the UAV trajectory tracking is to follow a given trajectory with a high precision reflecting the constraints. The constraints can be designed to take into account the physical attributes of the UAV or the restrictions imposed by the task. These constraints can be given e.g. as limited rotor speed, or constrained orientation (in some applications it is desired to avoid a large tilt of the UAV from the hover state for the sake of preserving the stability of UAV), speed limits to sustain safety rules, or limited position to prevent the UAV to fly to a restricted area or altitude. The trajectory tracking controller is generally implemented directly on board of the UAV, where a control code is running usually on the processor in the loop with a given frequency, so it is advantageous to consider the discrete model of the behavior. Considering the mentioned attributes, the trajectory tracking problem will be denoted in a manner of closed-loop Optimal Control Problem (OCP) [1, 55] for discrete-time linear time-varying (LTV) systems with linear constraints and a quadratic criterion for evaluation of the control quality.

The optimization problem is in this case formulated as

$$J\left(\boldsymbol{x}_0, \boldsymbol{u}_0^{M-1}\right) = \left(\boldsymbol{x}_M - \boldsymbol{x}_{r,M}\right)^{\mathsf{T}} \boldsymbol{Q}_M \left(\boldsymbol{x}_M - \boldsymbol{x}_{r,M}\right) + \sum_{k=0}^{M-1} \left[\left(\boldsymbol{x}_k - \boldsymbol{x}_{r,k}\right)^{\mathsf{T}} \boldsymbol{Q}_k \left(\boldsymbol{x}_k - \boldsymbol{x}_{r,k}\right) + \boldsymbol{u}_k^{\mathsf{T}} \boldsymbol{R}_k \boldsymbol{u}_k\right], \tag{4.1}$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k,\ k = 0, 1, 2, \ldots, M, \tag{4.2}$$

$$\boldsymbol{x}_k \in \mathcal{X},\ \mathcal{X} = \{\boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}^x \boldsymbol{x} \leq \boldsymbol{g}^x\},\ k = 0, 1, 2, \ldots, M, \tag{4.3}$$

$$\boldsymbol{u}_k \in \mathcal{U},\ \mathcal{U} = \{\boldsymbol{u} \in \mathcal{R}^m : \boldsymbol{F}^u \boldsymbol{u} \leq \boldsymbol{g}^u\},\ k = 0, 1, 2, \ldots, M-1, \tag{4.4}$$

where a long control horizon $M \gg 0$ is considered. The system is controlled along the given trajectory $\boldsymbol{x}_{r,k}$. The weighting matrices $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ of the quadratic cost function (4.1) are known symmetric positive semidefinite and positive definite, respectively. The quantities $\boldsymbol{x}_k \in \mathcal{R}^n$ and $\boldsymbol{u}_k \in \mathcal{R}^m$ are a state and control vector at time instant k, respectively.

The optimization constraints (4.2)-(4.4) are given by the linear system dynamics of the UAV, the state space and feasible control action. State space is constrained by linear inequality with matrix $\boldsymbol{F}^x$ and vector $\boldsymbol{g}^x$; control constraint inequality is defined using matrix $\boldsymbol{F}^u$ and vector $\boldsymbol{g}^u$.

A solution to this OCP is a closed-loop control strategy that minimizes the optimality criterion and steers the system by respecting the constraints at the same time. Unfortunately, the closed-form solution is hard to obtain due to the very long control horizon. The computational

and storage demands are also prohibitive. This necessitates the employment of some suitable approximation of the OCP that makes the problem more tractable.

Many feasible solutions to the OCP (4.1)-(4.4) are based on the employment of the standard Linear Quadratic Regulator (Linear Quadratic Regulator (LQR)) law [1]. This control law is optimal for the OCP given only by relations (4.1)-(4.2). The LQR law is determined by solving the Bellman optimization recursion that leads to the Riccati recursive relation

$$\boldsymbol{P}_k = \boldsymbol{A}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{A}_k + \boldsymbol{Q}_k - \left(\boldsymbol{A}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{B}_k\right)\left(\boldsymbol{B}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{B}_k + \boldsymbol{R}_k\right)^{-1}\left(\boldsymbol{A}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{B}_k\right)^\mathsf{T}, \quad (4.5)$$

where $\boldsymbol{P}_k \in \Re^{n\times n}$ is a symmetric positive semidefinite matrix with a terminal boundary condition $\boldsymbol{P}_N = \boldsymbol{Q}_N$. The LQR law is given in the form of the state-feedback controller

$$\boldsymbol{u}_k(\boldsymbol{x}_k) = \boldsymbol{K}_k\boldsymbol{x}_k \quad (4.6)$$

with state feedback gain $\boldsymbol{K}_k \in \Re^{n\times m}$

$$\boldsymbol{K}_k = -\left(\boldsymbol{B}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{B}_k + \boldsymbol{R}_k\right)^{-1}\left(\boldsymbol{A}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{B}_k\right)^\mathsf{T}. \quad (4.7)$$

For the control along the trajectory $\boldsymbol{x}_{r,k}$, the LQR law is according to [1] given by

$$\boldsymbol{u}_k(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}) = \boldsymbol{K}_k\boldsymbol{x}_k + \boldsymbol{L}_k, \quad (4.8)$$

where the compensation for the tracking of the trajectory $\boldsymbol{L}_k$ is described as

$$\boldsymbol{L}_k = -\left(\boldsymbol{B}_k^\mathsf{T}\boldsymbol{P}_{k+1}\boldsymbol{B}_k + \boldsymbol{R}_k\right)^{-1}\boldsymbol{B}_k^\mathsf{T}\boldsymbol{b}_{k+1}, \quad (4.9)$$

where $\boldsymbol{P}_k$ is the same as for the stabilization to the origin and

$$\boldsymbol{b}_k = \left(\boldsymbol{A}_k^\mathsf{T} + \boldsymbol{K}_k\boldsymbol{B}_k^\mathsf{T}\right)\boldsymbol{b}_{k+1} - \boldsymbol{Q}_k\boldsymbol{x}_{r,k}, \quad \boldsymbol{b}_M = 0. \quad (4.10)$$

In the case of the linear time-invariant (LTI) system and criterion with constant weight matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$, it is possible to calculate the gain $\boldsymbol{K}$ and compensation $\boldsymbol{L}_k$ in advance according to recursive Equation (4.9). In this case, the control law is given as follows

$$\boldsymbol{u}_k(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}^{k+M}) = \boldsymbol{K}\boldsymbol{x}_k + \bar{\boldsymbol{L}}\boldsymbol{x}_{r,k}^{k+M}, \quad (4.11)$$

$$\bar{\boldsymbol{L}}\boldsymbol{x}_{r,k}^{k+M} = \sum_{i=0}^{M}\bar{\boldsymbol{L}}_i \cdot \boldsymbol{x}_{r,k+i}, \quad (4.12)$$

$$\bar{\boldsymbol{L}}_i = -\left(\boldsymbol{B}^\mathsf{T}\boldsymbol{P}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\boldsymbol{B}^\mathsf{T} \cdot \left(\left(\boldsymbol{A}^\mathsf{T} + \boldsymbol{K}\boldsymbol{B}^\mathsf{T}\right)^{(i-1)} - \boldsymbol{Q}\right), \quad (4.13)$$

$$\boldsymbol{K} = -\left(\boldsymbol{B}^\mathsf{T}\boldsymbol{P}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{P}\boldsymbol{B}\right)^\mathsf{T}. \quad (4.14)$$

In the special case of control to a constant setpoint $\boldsymbol{x}_r$, the LQR has following form

$$\boldsymbol{u}_k(\boldsymbol{x}_k, \boldsymbol{x}_r) = \boldsymbol{K}\boldsymbol{x}_k + \bar{\boldsymbol{L}}\boldsymbol{x}_r, \quad (4.15)$$

$$\bar{\boldsymbol{L}} = -\left(\boldsymbol{B}^\mathsf{T}\boldsymbol{P}\boldsymbol{B} + \boldsymbol{R}\right)^{-1}\left[\boldsymbol{B}^\mathsf{T} \cdot \sum_{i=1}^{M}\left(\left(\boldsymbol{A}^\mathsf{T} + \boldsymbol{K}\boldsymbol{B}^\mathsf{T}\right)^{(i-1)} - \boldsymbol{Q}\right)\right]. \quad (4.16)$$

4.1 Model Predictive Control

MPC [12, 66] reduces the complexity of the constrained OCP by solving the OCP over a much shorter control horizon and employs a receding horizon policy, which means that at each time instant only the control $\boldsymbol{u}_k$, which is given as a solution to a particular OCP at the time instant k, is applied. As the name implies the MPC is a model-based control methodology. The MPC is the state-of-the-art method for the trajectory tracking problem because it inherently uses prediction for the acquisition of control strategy and at the same time it can take into account given constraints. Therefore, it can reflect the future of the desired trajectory, not only the request at the current time-instant, and control the system smoothly.

Sometimes, a prediction horizon and a control horizon can be considered separately in model predictive control. The prediction horizon refers to the period over which the future behavior of the system is predicted. The control horizon refers to the period in which the control inputs are directly optimized as decision variables. If the control horizon is shorter than the prediction horizon, then the control inputs are typically held constant after the control horizon ends, while predictions continue over the remaining prediction window. If the horizons are not explicitly described, a common assumption is that they are set to be equal in length.

In this section, the various types of the MPC will briefly be outlined. Further, several key areas regarding the MPC, their specific properties and requirements will be discussed. Finally, the state-of-the-art trajectory tracking application, where the MPC plays a crucial role, will be presented.

The history of MPC [30, 12, 82] reaches back to the end of the 1970s. In the beginning, the model of the control system was described explicitly by impulse response (in *Model Predictive Heuristic Control* [81], also known as *Model Algorithmic Control*) or step response (in *Dynamic Matrix Control* [26]). The input-output characteristics are still frequently used in industrial applications, nevertheless, in the UAV field, the state-space model is a standard for the MPC design and therefore it will be used in this section. The MPC has quickly become popular, particularly in the chemical industry [33], because of the inherent reflecting of system constraints, direct support of Multiple-Input Multiple-Output (MIMO) systems, and straightforward design of controller parameters. In the past, it used to be employed only in systems with slow dynamics due to the necessity of solving the problem online at each time instant. Over the years, it has been investigated from various perspectives by both academics and industry, and it has become a standard in constrained control. With current computational power, the MPC is applicable even to fast processes such as quadrotor UAVs [73].

4.1.1 Types of MPC

This part presents several types of MPC and discusses specific approaches to their solution. The Linear Model Predictive Control (LMPC) is a type of MPC, where a deterministic discrete-time LTV system and a quadratic criterion are assumed. The description of the LMPC is similar

to the OCP (4.1)-(4.4) with a reformulated criterion, that is at each time instant given as

$$J\left(\boldsymbol{x}_k, \boldsymbol{u}_k^{k+N-1}, N\right) = (\boldsymbol{x}_{k+N} - \boldsymbol{x}_{r,k+N})^\mathsf{T} \boldsymbol{Q}_{k+N} (\boldsymbol{x}_{k+N} - \boldsymbol{x}_{r,k+N}) + \sum_{l=k}^{k+N-1} \left[(\boldsymbol{x}_l - \boldsymbol{x}_{r,l})^\mathsf{T} \boldsymbol{Q}_l (\boldsymbol{x}_l - \boldsymbol{x}_{r,l}) + \boldsymbol{u}_l^\mathsf{T} \boldsymbol{R}_l \boldsymbol{u}_l\right], \tag{4.17}$$

$$\text{s.t.} \quad \boldsymbol{x}_{l+1} = \boldsymbol{A}_l \boldsymbol{x}_l + \boldsymbol{B}_l \boldsymbol{u}_l,\ l = k, k+1, \ldots, k+N, \tag{4.18}$$

$$\boldsymbol{x}_l \in \mathfrak{X},\ l = k, k+1, \ldots, k+N, \tag{4.19}$$

$$\boldsymbol{u}_l \in \mathfrak{U},\ l = k, k+1, \ldots, k+N-1, \tag{4.20}$$

where N is the length of the receding horizon. Constraints (4.2)-(4.4) are the same as for OCP.

The solution to the LMPC can be obtained using a transcription of the problem to a Quadratic Program (QP) [12] which can be solved by a QP solver. There are two main forms used for this transformation, the sparse and dense [57], which can be acquired either analytically or using software such as CVXGEN [65] or YALMIP [61].

Nonlinear Model Predictive Control (NMPC) is a methodology that can deal with nonlinear deterministic problems. There are two main descriptions of NMPC. The first is based on the discrete-time and the second on the continuous-time model. The former can be denoted as

$$J\left(\boldsymbol{x}_k, \boldsymbol{u}_k^{k+N-1}, N\right) = \Phi\left(\boldsymbol{x}_{k+N}, \boldsymbol{x}_{r,k+N}, N\right) + \sum_{l=k}^{k+N-1} \mathfrak{L}\left(\boldsymbol{x}_l, \boldsymbol{x}_{r,l}, \boldsymbol{u}_l, l\right), \tag{4.21}$$

$$\text{s.t.} \quad \boldsymbol{x}_{l+1} = f\left(\boldsymbol{x}_l, \boldsymbol{u}_l, l\right),\ l = k, k+1, \ldots, k+N, \tag{4.22}$$

$$\boldsymbol{x}_l \in \mathfrak{X},\ l = k, k+1, \ldots, k+N, \tag{4.23}$$

$$\boldsymbol{u}_l \in \mathfrak{U},\ l = k, k+1, \ldots, k+N-1, \tag{4.24}$$

where $\mathfrak{L}$ and Φ are the cost and the terminal cost function, respectively. The constraints for state and control vectors are given by sets $\mathfrak{X}$ and $\mathfrak{U}$, respectively. A solution to the NMPC can be acquired by the NLP solver which usually solves the problem using sequential QP or by Bellman optimization recursion.

In some UAV applications, the NMPC with a continuous-time model is employed. It can be stated as follows

$$J\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t_f\right) = \Phi\left(\boldsymbol{x}(t_f), \boldsymbol{x}_r(t_f), t_f\right) + \int_{t_s}^{t_f} \mathfrak{L}\left(\boldsymbol{x}(t), \boldsymbol{x}_r(t), \boldsymbol{u}(t), t\right) dt, \tag{4.25}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}(t) = f\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right),\ t \in \left\langle t_s, t_f \right\rangle, \tag{4.26}$$

$$\boldsymbol{x}(t) \in \mathfrak{X},\ t \in \left\langle t_s, t_f \right\rangle, \tag{4.27}$$

$$\boldsymbol{u}(t) \in \mathfrak{U},\ t \in \left\langle t_s, t_f \right\rangle, \tag{4.28}$$

where t_s is the current time instant and t_f is the end-time of the horizon. It is specific in dealing with the receding horizon, which is given by the interval $t \in \langle t_s, t_f \rangle$. Using the analogy of discrete system, $t_s = kT, t_f = (k+N)T$ and $T = \dfrac{t_f - t_s}{N}$, where T is the sampling period. The continuous-time NMPC can be solved using numerical methods for the continuous-time OCP, which were presented in Section 3.3.

The possibility of the MPC employment in systems with uncertainty has been also investigated. There are two distinct classes of MPC which consider the uncertainty in the model.

In the Robust Model Predictive Control (RMPC), the state and the control constraints have to be satisfied by the controlled system for *all* realizations of uncertainty. A general model with a disturbance can be denoted as

$$\boldsymbol{x}_{l+1} = f\left(\boldsymbol{x}_l, \boldsymbol{u}_l, \boldsymbol{w}_l\right), \; l = k, k+1, \ldots, k+N, \tag{4.29}$$

where $\boldsymbol{w}_l \in \Re^n$ is the disturbance that is assumed to lie in a compact subset $\mathfrak{W}$ of $\Re^n$.

One of the approaches in the RMPC is to incorporate the disturbance sequence, denoted as $\boldsymbol{w}_k^{k+N}$, along with the control sequence $\boldsymbol{u}_k^{k+N}$. This introduces a change in the optimization criterion, given by

$$J\left(\boldsymbol{x}_k, \boldsymbol{w}_k^{k+N-1}\boldsymbol{u}_k^{k+N}, N\right) = \max_{\boldsymbol{w}\in\mathfrak{W}} \left\{ \Phi\left(\boldsymbol{x}_{k+N}, \boldsymbol{x}_{r,k+N}, N\right) + \sum_{l=k}^{k+N-1} \mathfrak{L}\left(\boldsymbol{x}_l, \boldsymbol{x}_{r,l}, \boldsymbol{u}_l, l\right) \right\}. \tag{4.30}$$

By incorporating the disturbance sequence and changing the criterion, it becomes a min-max problem which was discussed in [53, 78]. Another option is to use the criterion for the deterministic system (4.21) and find a robust invariant set to ensure the recursive feasibility [36, 64, 78]. Compared to the aforementioned methods that are based on open-loop solutions to RMPC, the Tube-based RMPC (see [67]) makes sure that all the closed-loop trajectories lie in a tube that satisfies the constraints [66].

Unlike the RMPC, in the Stochastic Model Predictive Control (SMPC) [66, 39], it is assumed that the disturbance process is stochastic; it is not necessarily bounded and the constraints are softened (which means that not all realizations of disturbance satisfy the constraints). The criterion for the SMPC can be obtained by modifying the criterion for NMPC as

$$J\left(\boldsymbol{x}_k, \boldsymbol{u}_k^{k+N-1}, N\right) = \mathsf{E}\left\{ \Phi\left(\boldsymbol{x}_{k+N}, \boldsymbol{x}_{r,k+N}, N\right) + \sum_{l=k}^{k+N-1} \mathfrak{L}\left(\boldsymbol{x}_l, \boldsymbol{x}_{r,l}, \boldsymbol{u}_l, l\right) \right\}. \tag{4.31}$$

Solutions to the aforementioned MPC problems are called implicit, because they require solving the problems at each time instant. In addition, it is also possible to obtain an explicit MPC solution [12, 41] for the entire state-space (or a desired subset). The solution can be acquired using multi-parametric programming.

Multi-parametric programming transforms the MPC optimization problem into a multi-parametric quadratic program (mp-QP) by treating the system state as a parameter. The solution to the mp-QP provides an explicit, piecewise affine state feedback control law that can be evaluated efficiently online, without the need to solve an optimization problem. Implicit and explicit solutions yield the same results and are equivalent mathematically. The primary benefit of explicit MPC is the faster and simpler evaluation of the piecewise affine control function compared to solving the implicit MPC online. Therefore, explicit MPC is ideal for deployment on low-computational platforms.

However, there are drawbacks in computational demands (nearly impossible in higher dimensions), in memory demands because of storing the whole solution in a control device, and in a time-consuming search for an adequate control strategy based on the current state-space coordinates for its use in control loop. Moreover, it is important to mention that there is a significant increase of multi-parametric problem dimensionality with free trajectory tracking (the solution depends not only on the state $\boldsymbol{x}_k$ but rather on the whole reference trajectory $\boldsymbol{x}_{r,k}^{k+N}$).

4.1.2 MPC Application in the UAV Field

State-of-the-art research of the MPC in the UAV field will now be presented, with an emphasis on trajectory tracking. Trajectory tracking for UAV was investigated and also experimentally tested e.g. in [5, 4, 49, 48, 69, 77]. The standard LMPC for LTI system with quadratic criterion was utilized in [5, 49].

In [5], a problem with control vector constraints is investigated. The control is considered in the form of a desired attitude, which is employed in the control cascade as a setpoint for the onboard attitude controller. The constraints are handled a priori by tuning the weight $\boldsymbol{R}$ and a safety mechanism is implemented, which adjusts the control vector to meet the constraints at the cost of losing the optimality. Since the unconstrained MPC is employed, it can be solved analytically with significantly lower computational demands. The resulting MPC was compared to a standard Proportional-Integral-Derivative (PID) controller and delivered much better results.

In [49], the performance of the LMPC and the continuous-time NMPC are compared. The linear model has been obtained by linearization at the hover state. The NMPC applies the same criterion as the standard LMPC. The control strategy contains the required attitude and collective thrust. An external disturbance (wind) is estimated by the Extended Kalman Filter (EKF) and compensated in both controllers. The LMPC was transcribed into the form of QP and implemented using CVXGEN [65] code generator framework. The NMPC was implemented using ACADO toolbox [45, 44]. Both controllers were running on the Core-i7 processor and 8GB of RAM and they were evaluated in several scenarios. The NMPC showed slightly better performance, disturbance rejection, and also a computational effort when controlling the Asctec NEO hexarotor UAV.

In [69], the possibility of implementing NMPC directly in an embedded UAV system was explored. The reference trajectory was predicted based on the information on the position and velocity of the virtual target. The prediction was performed for 6 steps, which with a frequency of 100Hz made the prediction horizon 0.06s. The control horizon was one step. For efficiency, the unit norm of the UAV's Manhattan distance to the target and the control strategy were considered in the cost function. The NMPC was compared with the PID controller using the Parrot Mambo Minidrone UAV with only onboard sensors involved. Despite the limitations of the NMPC platform, it achieved comparable and sometimes slightly better results than PID.

Thanks to the ability to easily extend MPCs with additional constraints, some MPCs add collision avoidance capability. The LMPC was designed with a simple LTI model of the UAV with state constraints and quadratic criterion in [4]. This LMPC reassured that the reference UAV trajectory is feasible by processing the MPC control strategy through the LTI model. Afterward, the output of the model was used as a reference for the nonlinear three-dimensional Special Euclidean Group (SE(3)) attitude and position controller similar to the controller presented in Appendix C (with the difference that the rotational matrix was chosen as an attitude representation). Moreover, the output of the model was sent to other UAVs, where it was harnessed for a simple collision avoidance algorithm. If there was a risk of a collision, the UAV would have been ordered to climb according to its priority in the swarm. The LMPC problem in the form of the QP was solved by the CVXGEN [65]. The whole tracking system was operating directly and separately in each UAV using onboard computers with 5th generation Core-i7 processor and 8GB of RAM. The controller has been proven in real-time applications with 4 DJI F550 UAVs at the same time.

The continuous-time NMPC for the UAV collision avoidance problem was presented in [48]. The control strategy was considered in the form of the desired attitude, which was then applied

by the attitude controller. Therefore, only a simplified attitude model was considered directly in NMPC. The criterion of the NMPC was quadratic supplemented by the collision cost, which involved the distance to other UAVs. It was possible to consider the priority of the UAVs in the cost function such that the UAV will only perform an evasive maneuver in front of more prioritized UAV. The NMPC employed a simple model of other UAVs considering their position and speed. The estimation of other UAVs motion was obtained by the EKF and uncertainty of estimated positions and speeds were further reflected in the MPC through the collision cost. The method was implemented using the ACADO toolbox and it was tested in a series of simulations with six UAVs and experimentally with two AscTec NEO hexarotor UAVs.

In [77], the NMPC was proposed that controls both the position and attitude of the drone. Thus, the control strategy was in the form of individual rotor thrusts. The NMPC was able to avoid static elliptical spheres on a reference trajectory. To minimize the deviation from the reference trajectory, the cost function was augmented with the integral of the control error in the cost function. NMPC was solved using the CaSADI framework [2] by a multiple-shooting method, however, only simulation experiments were performed.

In [97], the procedure of incorporating MPC in training a neural network in deep reinforcement learning method to control a drone based on onboard sensor data as observations have been described. The MPC was involved in trajectory tracking and also minimized the deviation from the neural network policy in the objective function. Data acquisition and final tests were performed on a simulated drone. The resulting policy was able to steer the drone to the target along a non-collision path and even outperformed the original MPC in most scenarios. The involvement of the MPC during learning minimized collisions to zero and thus increased the safety of the whole procedure.

4.2 Interpolating Control

Another promising methodology applicable to the OCP is Interpolating Control (IC) [72, 68, 14, 87], which applies to similar problems as the MPC. The huge advantage of IC is that the control action is obtained by solving a very simple Linear Program (LP). Unfortunately, IC in its standard form does not apply to the trajectory tracking problem as it can only stabilize the system to zero. Nevertheless, it serves as the foundation for the trajectory tracking capable algorithm proposed in Chapter 7.

The IC is based on the interpolation between a couple or several state-feedback gain control laws designed without consideration of the inherent constraints. Usually, the interpolation is performed between two independent state-feedback controllers. One of the controllers has a higher gain, but it operates within constraints in the smaller region. The second controller has a lower gain, which enables it to operate within a larger region. However, this often results in poorer performance. Using the invariant set theory [10], it is ensured that the constraints are not violated. Thanks to the fusion of several control laws with known positively invariant sets, the IC applies to the same type of problems as the MPC. Knowing the positively invariant set for each control law, the IC searches only an optimal coefficient for the interpolation between control laws as a solution to a simple LP.

The positively invariant sets are utilized in the design of the interpolating control and will therefore be described. The system presented at the beginning of this chapter will be considered in the description. Since the system (4.2) and the constraints (4.3) and (4.4) are considered linear, the sets will be defined in the form of polyhedra.

4.2.1 Invariant Sets

Prior to defining the positively invariant set, it is essential to describe the closed-loop system in the following form

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_{c_k}\boldsymbol{x}_k, \tag{4.32}$$

where $\boldsymbol{A}_{c_k}$ denotes the matrix of dynamics for the closed-loop system, given by

$$\boldsymbol{A}_{c_k} = \boldsymbol{A}_k + \boldsymbol{B}_k\boldsymbol{K}, \tag{4.33}$$

where $\boldsymbol{K}$ is the gain of the state-feedback controller such as LQR in Equation (4.6).

According to [10, p. 121] it is assumed that the system (4.32) is defined on a proper open set

$$\mathcal{O} \subseteq \mathcal{R}^n, \tag{4.34}$$

where n is a dimension of state-space, and that there is a globally defined solution (i.e., $\forall k \geq 0$) for every initial condition $\boldsymbol{x}_0 \in \mathcal{O}$.

Furthermore, the state is constrained as in Equation (4.3) and the control is constrained by Equation (4.4), which can be for the case of the state-feedback linear controller denoted also as $\boldsymbol{u}_k = \boldsymbol{K}\boldsymbol{x}_k \in \mathcal{U}$.

Positively Invariant Set $\boldsymbol{\Omega} \subseteq \mathcal{O}$ is said to be positively invariant w.r.t. Equation (4.32) if and only if $\forall \boldsymbol{x}_k \in \boldsymbol{\Omega}$ with initial condition $\boldsymbol{x}_0 \in \boldsymbol{\Omega}$ is globally defined and such that $\boldsymbol{x}_k \in \boldsymbol{\Omega}$ for $k > 0$. This implies that once the state $\boldsymbol{x}_k$ reaches $\boldsymbol{\Omega}$, it will remain within $\boldsymbol{\Omega}$ while satisfying the state and control constraints. The term *positively* refers to the fact that only forward evolutions of system (4.32) are considered.

The state-feedback gain controllers operate within a limited region given by constraints which can be described using positively invariant sets $\boldsymbol{\Omega}$. The acquisition of set is identical for each state-feedback gain controller; therefore, it will be described below only for the controller with the positively invariant set $\boldsymbol{\Omega}$ described by a polytope with the half-space representation as

$$\boldsymbol{\Omega} = \{\boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}\boldsymbol{x} \leq \boldsymbol{g}\}. \tag{4.35}$$

If matrix $\boldsymbol{F}$ and vector $\boldsymbol{g}$ are found such that $\boldsymbol{\Omega}$ reaches maximal volume within the constraints of the controlled system, then it is called the maximal positively invariant set $\boldsymbol{\Omega}_{\max}$, where

$$\boldsymbol{\Omega} \subseteq \boldsymbol{\Omega}_{\max}, \ \forall \boldsymbol{\Omega}. \tag{4.36}$$

The algorithm for the acquisition of $\boldsymbol{\Omega}$ and $\boldsymbol{\Omega}_{\max}$ is provided in Appendix E.

Controlled Positively Invariant Set According to the definition in [10, p. 123], $\mathcal{C} \subseteq \mathcal{X}$ is considered a controlled positively invariant if there exists a control $\boldsymbol{u}_k \in \mathcal{U}$ such that, $\forall \boldsymbol{x}_0 \in \mathcal{C}$, the condition $\boldsymbol{x}_k \in \mathcal{C}$ holds for all $k \geq 0$. In other words, the set $\mathcal{C}$ is not sought for the closed-loop system for a given controller as it was performed in the case of set $\boldsymbol{\Omega}$, but for control actions given by set $\mathcal{U}$.

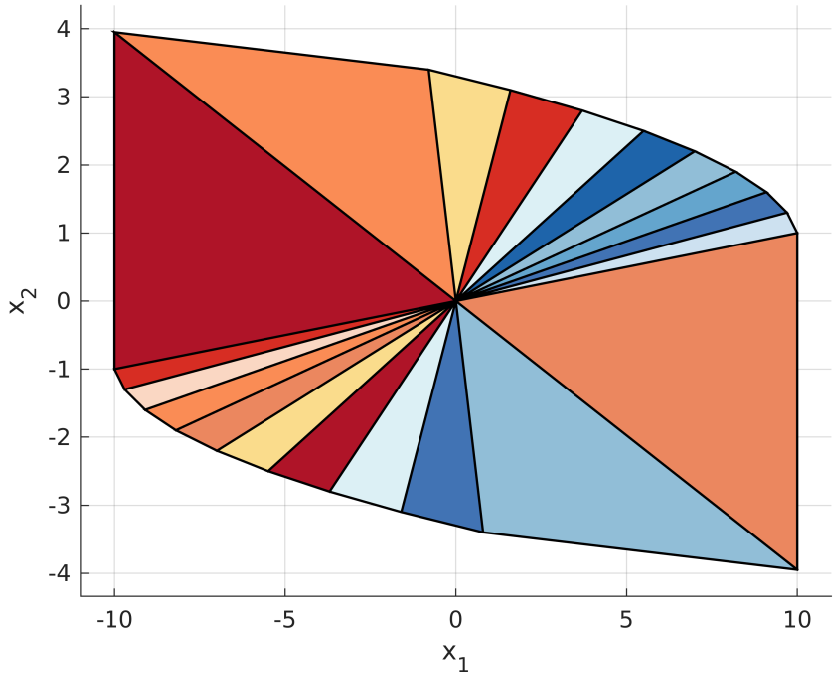


Figure 4.1: Partition of the VC state space

N-step controlled positively invariant set $\mathcal{C}^N$ is a special case of the previously described controlled invariant set $\mathcal{C}$. Set $\mathcal{C}^N$ describes the state-space from which the system can be steered to $\boldsymbol{\Omega}$ in no more than N steps using an admissible control; it is denoted as

$$\mathcal{C}^N = \left\{ \boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}^N \boldsymbol{x} \leq \boldsymbol{g}^N \right\}. \tag{4.37}$$

The set $\mathcal{C}^N$ is calculated by an iterative algorithm (see Algorithm 7 in Appendix E), which is initialized with $\boldsymbol{\Omega}$ and further expanded within the constraints of the OCP.

4.2.2 Vertex Control

Vertex Control (VC) [38, 72] is often used as a low-gain controller in IC for its ability to inherently consider the constraints on a given set; therefore, it will be described below.

The VC is usually defined on the N-step controlled positively invariant set $\mathcal{C}^N$ defined in Section 4.2.1, because it covers state-space, from which it is possible to control the system within the constraints. In the standard VC, the admissible control is determined for all vertices of $\mathcal{C}^N$ and subsequently, the control law is calculated in the form of a piece-wise affine function with zero control in origin.

The set $\mathcal{C}^N$ is in the VC divided into j simplices $\mathcal{C}^{N(j)}$, each with $n+1$ vertices $\boldsymbol{x}_1^{(j)}$, $\boldsymbol{x}_2^{(j)}, \ldots, \boldsymbol{x}_{n+1}^{(j)}$, where n is the dimension of state space. An example of the $\mathcal{C}^N$ division is depicted in Figure 4.1 for $n = 2$. Each simplex represents a part of N-step controlled positively invariant set $\mathcal{C}^N$ over which a relevant control function will be defined. This division of $\mathcal{C}^N$ into simplices is important for the use of VC because it is necessary to find out in which simplex the current state of the system $\boldsymbol{x}_k$ is and to use the appropriate control function accordingly.

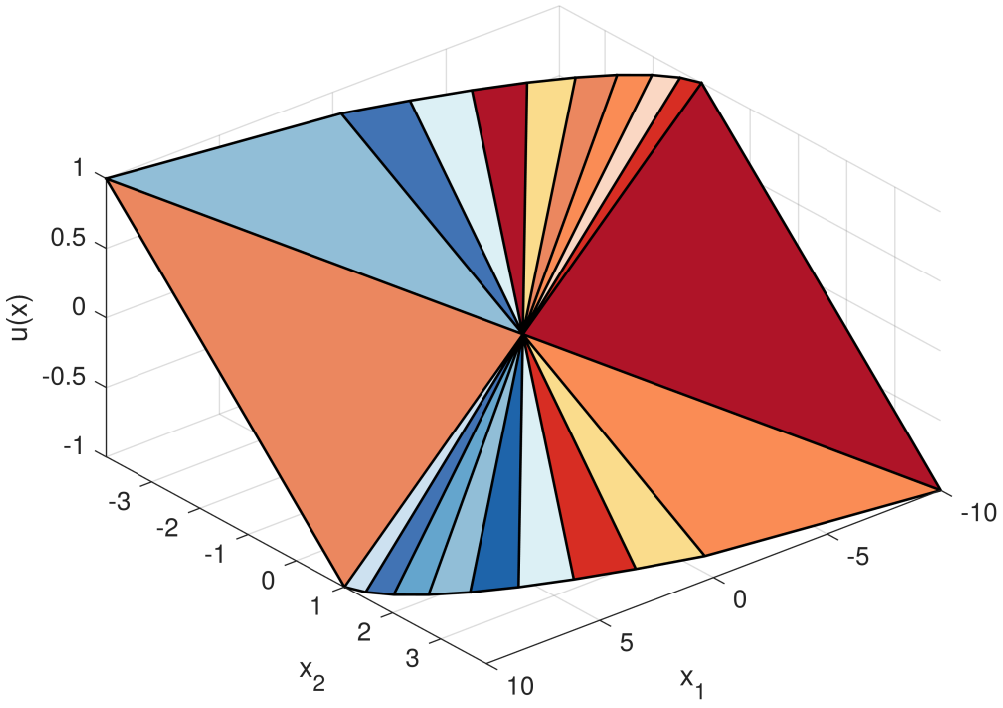


Figure 4.2: Explicit control law of the VC

The linear control law is defined for the simplex $\mathcal{C}^{N^{(j)}}$ as

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{K}^{(j)}\boldsymbol{x},\ \boldsymbol{x} \in \mathcal{C}^{N^{(j)}}, \tag{4.38}$$

where $\boldsymbol{K}^{(j)}$ is obtained using

$$\boldsymbol{K}^{(j)} = \boldsymbol{U}^{(j)}\left(\boldsymbol{V}^{(j)}\right)^{-1} \tag{4.39}$$

with the matrix of vertices $\boldsymbol{V}^{(j)} = \left[\boldsymbol{v}_1^{(j)}, \boldsymbol{v}_2^{(j)}, \ldots, \boldsymbol{v}_n^{(j)}\right]$ which generates $\mathcal{C}^{N^{(j)}}$ and matrix of admissible control at the corresponding vertices $\boldsymbol{U}^{(j)} = \left[\boldsymbol{u}_1^{(j)}, \boldsymbol{u}_2^{(j)}, \ldots, \boldsymbol{u}_n^{(j)}\right]$.

The admissible control $\boldsymbol{u}_i^{(j)}$ [71] at the vertex $\boldsymbol{v}_i^{(j)}$ of the simplex $\mathcal{C}^{N^{(j)}}$ can be determined as a solution to the following LP

$$J(\boldsymbol{u}^{(j)}) = \|\boldsymbol{u}^{(j)}\|_p, \tag{4.40}$$

$$\text{s.t.}\quad \boldsymbol{F}^N\left(\boldsymbol{A}\boldsymbol{v}^{(j)} + \boldsymbol{B}\boldsymbol{u}^{(j)}\right) \leq \boldsymbol{g}^N, \tag{4.41}$$

$$\boldsymbol{F}^u\boldsymbol{u}^{(j)} \leq \boldsymbol{g}^u, \tag{4.42}$$

where the criterion (4.40) with the p-norm, $p = 1$ or $p = +\infty$, is maximized reflecting the constraints (4.41) and (4.42). The resulting VC law for a system with the constrained control $-1 \leq \boldsymbol{u} \leq 1$ is depicted in Figure 4.2.

In [72, Section 3.4], it was proven, that the VC guarantees recursive feasibility for all initial states $\boldsymbol{x}_0 \in \mathcal{C}^{N^{(j)}}$. Moreover, for the LTV system with linear constraints of state and control, the control law is asymptotically stabilizing for all initial states $\boldsymbol{x}_0 \in \mathcal{C}^N$.

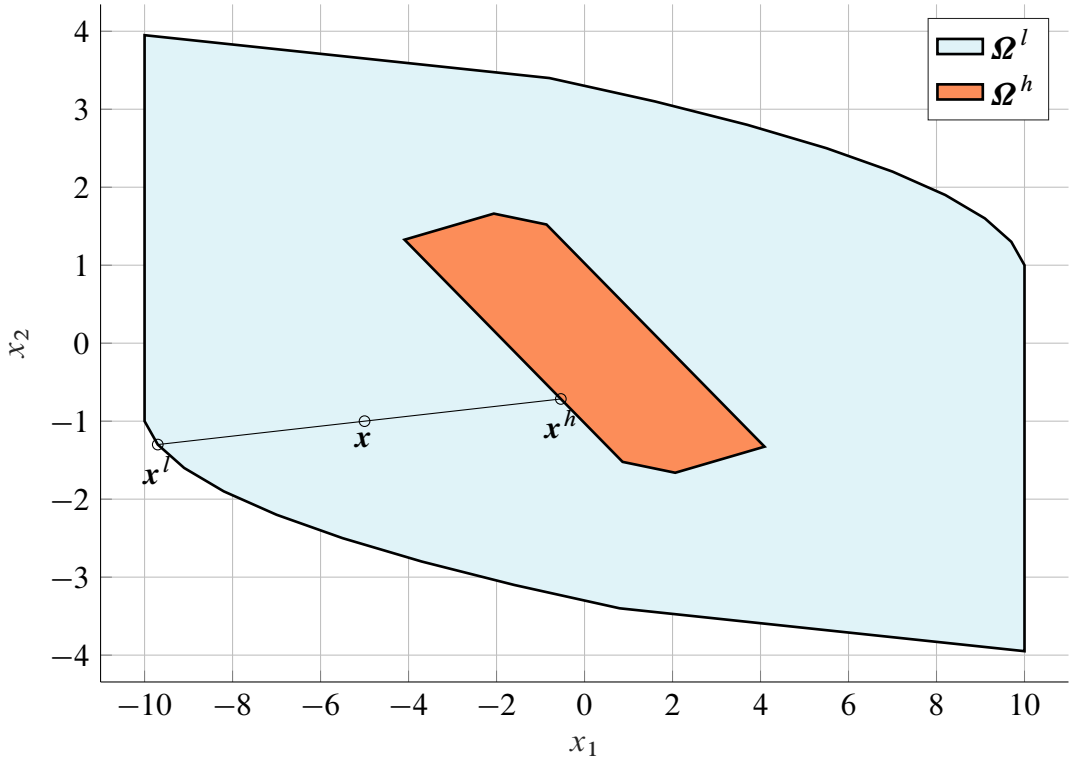


Figure 4.3: Example of state decomposition

4.2.3 State and Control Decomposition

For the interpolation, the principle of state decomposition (see Figure 4.3) is employed, which can be denoted as

$$\boldsymbol{x} = c\boldsymbol{x}^l + (1-c)\,\boldsymbol{x}^h, \tag{4.43}$$

where $\boldsymbol{x}$ is the state vector, c is the interpolating coefficient, $c \in [0, 1]$, and $\boldsymbol{x}^h$ and $\boldsymbol{x}^l$ is the state vector for a high-gain and a low-gain controller, respectively. It is possible to reflect the decomposed state from Equation (4.43) to the interpolating control law as follows

$$\boldsymbol{u}(\boldsymbol{x}) = c\boldsymbol{u}^l\left(\boldsymbol{x}^l\right) + (1-c)\,\boldsymbol{u}^h\left(\boldsymbol{x}^h\right), \tag{4.44}$$

where $\boldsymbol{u}^h(\boldsymbol{x}^h)$ is the high-gain control law for $\boldsymbol{x}^h$ and $\boldsymbol{u}^l(\boldsymbol{x}^l)$ is the low-gain control law for $\boldsymbol{x}^l$. The high-gain controller is more aggressive compared to the low-gain controller. It is designed to achieve faster control response and more significant changes in the system's behavior. However, this behavior comes at a cost. Because of the higher gain, the controller's invariant set has a smaller volume compared to the invariant set for the low-gain controller. In contrast, the low-gain controller is designed to cover a much larger volume of state space.

Within this thesis, the N-step controlled positively invariant set will be denoted also by $\boldsymbol{\Omega}^l$. This notation is because not only the VC will be used as a low-gain controller $\boldsymbol{u}^l$, but sometimes it may be replaced by another controller that may not fully cover $\mathcal{C}^N$ within the constraints.

Both controllers could be utilized by simply switching between them based on the current state of the system. However, this switching would introduce discontinuity in the control, and in the region $\boldsymbol{\Omega}^l \setminus \boldsymbol{\Omega}^h$, it would lead to slower convergence of the system state towards the desired region of the state space.

4.2.4 Implicit Interpolating Control

As has been said, the IC depends on finding the optimal interpolation coefficient c^*, which determines the decomposition of the state (4.43) and subsequently the resulting control (4.44). The optimal ratio c^* can be acquired by minimizing the criterion $J(\boldsymbol{x}^l, c)$ in the following NLP

$$J(\boldsymbol{x}^l, c) = c, \tag{4.45}$$
$$\text{s.t.} \quad \boldsymbol{F}^l \boldsymbol{x}^l \leq \boldsymbol{g}^l, \tag{4.46}$$
$$\boldsymbol{F}^h \boldsymbol{x}^h \leq \boldsymbol{g}^h, \tag{4.47}$$
$$c\boldsymbol{x}^l + (1-c)\,\boldsymbol{x}^h = \boldsymbol{x}, \tag{4.48}$$
$$0 \leq c \leq 1, \tag{4.49}$$

where $\boldsymbol{x}$ is the current state vector. Using an auxiliary variable $\boldsymbol{r}^l = c \cdot \boldsymbol{x}^l$ the problem becomes linear

$$J(\boldsymbol{r}^l, c) = c, \tag{4.50}$$
$$\text{s.t.} \quad \boldsymbol{F}^l \boldsymbol{r}^l \leq c\boldsymbol{g}^l, \tag{4.51}$$
$$\boldsymbol{F}^h \left(\boldsymbol{x} - \boldsymbol{r}^l\right) \leq (1-c)\left(\boldsymbol{g}^h\right), \tag{4.52}$$
$$0 \leq c \leq 1, \tag{4.53}$$

where c^* is found by minimizing the criterion $J(\boldsymbol{x}^l, c)$ or $J(\boldsymbol{r}^l, c)$. To keep the interpolating coefficient optimal and to ensure that the constraints are not violated, the LP must be solved at each time instant k. Feasibility of LP solution and closed-loop stability of IC was closely studied in [72]. Briefly, if controllers $\boldsymbol{u}^h$ and $\boldsymbol{u}^l$ are stabilizing and $\boldsymbol{x}_0 \in \boldsymbol{\Omega}^l$, then the IC control law is asymptotically stabilizing $\forall \boldsymbol{x} \in \boldsymbol{\Omega}^l$ for the control to origin.

After obtaining a solution to the LP, the decomposed control laws are calculated as follows

$$\boldsymbol{u}^l = \boldsymbol{K}^l \boldsymbol{r}^l, \tag{4.54}$$
$$\boldsymbol{u}^h = \boldsymbol{K}^h \boldsymbol{r}^h, \tag{4.55}$$

where $\boldsymbol{r}^h = \boldsymbol{x} - \boldsymbol{r}^l$. If the VC is employed the gain is given by $\boldsymbol{K}^l = \boldsymbol{K}^{(j)}$, where j is index of simplex, where $\boldsymbol{x}^l$ is located.

Looking into the LP (4.50)–(4.53) it is clear that if $\boldsymbol{x}_k \in \boldsymbol{\Omega}^h$, $c = 0$, the IC behaves like the high-gain controller $\boldsymbol{u}^h$. If $\boldsymbol{x}$ is on the boundary of $\boldsymbol{\Omega}^l$, that is $\boldsymbol{F}^l \boldsymbol{x} = \boldsymbol{g}^l$, then $c = 1$ and the IC behaves like the low-gain controller $\boldsymbol{u}^l$.

In case the set $\boldsymbol{\Omega}^l \setminus \boldsymbol{\Omega}^h$ is large, the performance of the IC may be low. However, the performance can be usually improved by adding an intermediate set

$$\boldsymbol{\Omega}^m = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{F}^m \boldsymbol{x} \leq \boldsymbol{g}^m\}, \tag{4.56}$$
$$\boldsymbol{\Omega}^l \subset \boldsymbol{\Omega}^m \subset \boldsymbol{\Omega}^h, \tag{4.57}$$

where another state-feedback controller is defined. This state decomposition is presented in Figure 4.4. In case $\boldsymbol{u}$ is LQR, $\boldsymbol{u}^m$ can be obtained for example with an increase in the weight $\boldsymbol{R}$. The set $\boldsymbol{\Omega}^m$ is calculated in the same way as $\boldsymbol{\Omega}^h$ with Algorithm 6. In this extended version

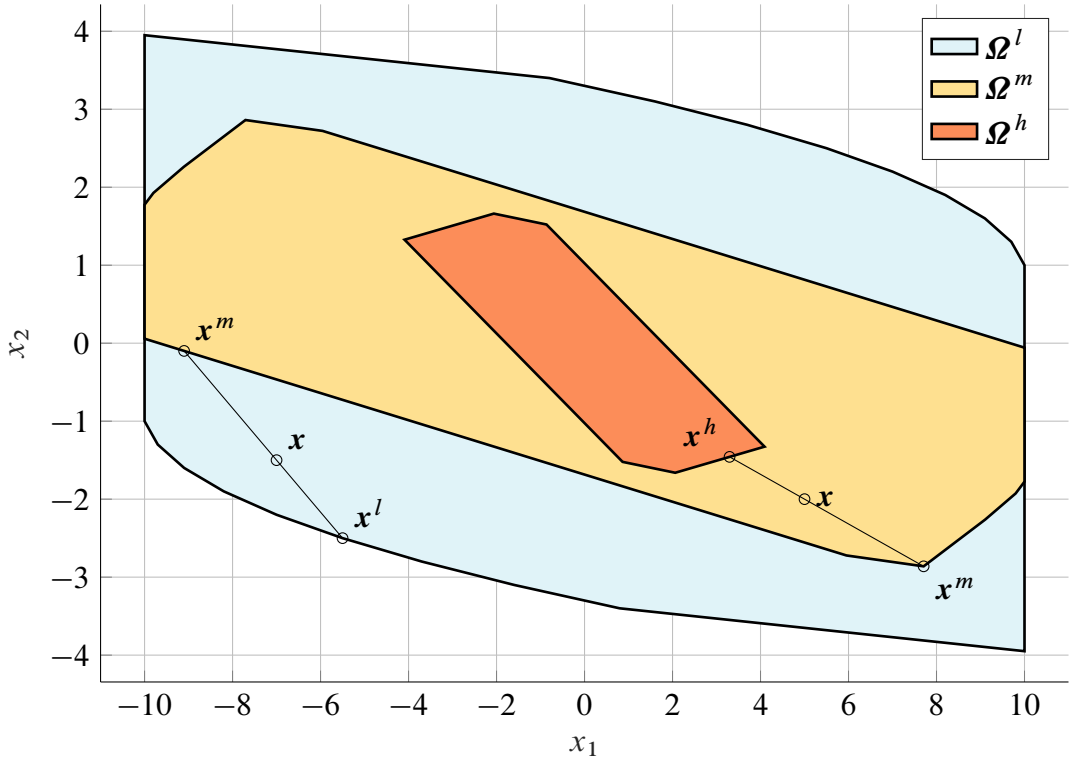


Figure 4.4: Example of state decomposition with additional region Ω^m or $\mathcal{C}^S$

of Interpolating Control (eIC), there are two different LPs. If $\boldsymbol{x} \in \Omega^l \setminus \Omega^m$, the interpolation is done between $\boldsymbol{u}^l$ and $\boldsymbol{u}^m$ and the IC control law is in form

$$\boldsymbol{u}(\boldsymbol{x}) = c\boldsymbol{u}^l\left(\boldsymbol{x}^l\right) + (1-c)\,\boldsymbol{u}^m\left(\boldsymbol{x}^m\right),\ \boldsymbol{x} \in \Omega^l \setminus \Omega^m. \tag{4.58}$$

If the $\boldsymbol{x} \in \Omega^m$, the interpolation is performed for both $\boldsymbol{u}^m$ and $\boldsymbol{u}^h$, which results in control law

$$\boldsymbol{u}(\boldsymbol{x}) = c\boldsymbol{u}^m\left(\boldsymbol{x}^m\right) + (1-c)\,\boldsymbol{u}^h\left(\boldsymbol{x}^h\right),\ \boldsymbol{x} \in \Omega^m. \tag{4.59}$$

4.2.5 Explicit Interpolating Control

The advantage of using explicit IC [72] is that the LP is solved beforehand, removing the requirement for the LP solver to be available on the target control platform. In addition, the computational demands of the control loop can be reduced. The explicit solution to IC is obtained using a multi-parametric LP solver [41, 12] with $\boldsymbol{x}$ as a parametric variable. By solving the LP (4.50)–(4.53), a state-space partition of Ω^l is acquired. In the set Ω^h, the control law is given by the high-gain controller $\boldsymbol{u}^h$ and in each i-th simplex of partition $\Omega^l \setminus \Omega^h$ a linear control law function is defined as

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{M}_i\boldsymbol{x} + \boldsymbol{n}_i, \tag{4.60}$$

where $\boldsymbol{M}_i \in \Re^{m\times n}$ and $\boldsymbol{n}_i \in \Re^m$ are defined as

$$\begin{bmatrix}\boldsymbol{M}_i & \boldsymbol{n}_i\end{bmatrix} = \begin{bmatrix}u_1^{(i)} & u_2^{(i)} & \dots & u_{n+1}^{(i)}\end{bmatrix}\begin{bmatrix}V_1^{(i)} & V_2^{(i)} & \dots & V_{n+1}^{(i)} \\ 1 & 1 & \dots & 1\end{bmatrix}, \tag{4.61}$$

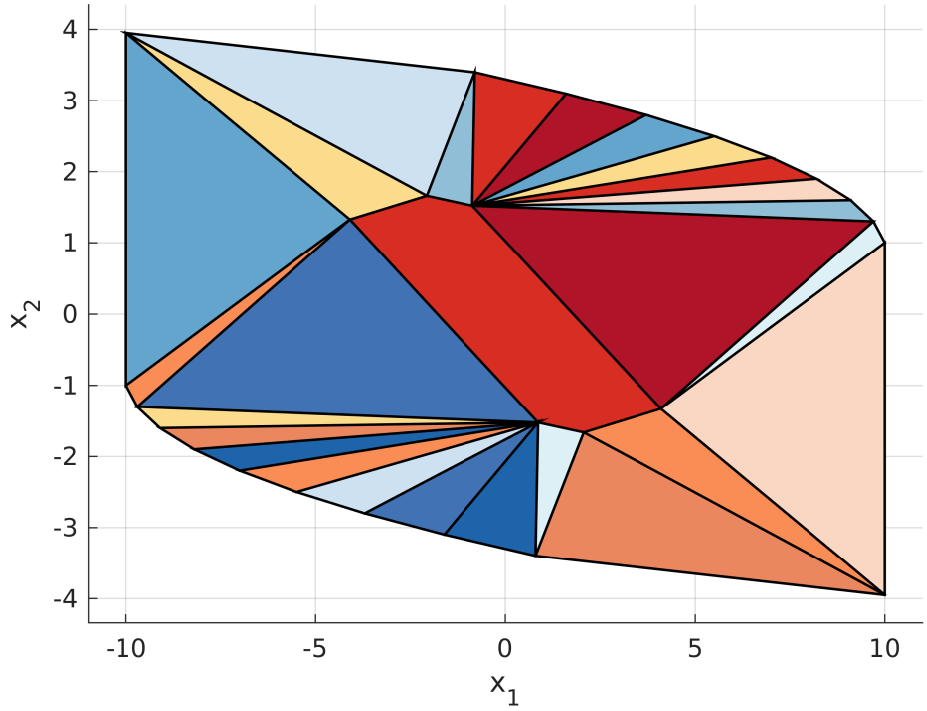


Figure 4.5: Partition of explicit solution for the IC

where $V_j^{(i)}$ is the $j-th$ vertex of the i-th simplex. The control value is given by the vertex. If $V_j^{(i)} \in \boldsymbol{\Omega}^h$, it is given by $\boldsymbol{u}^h$, if it is on the boundary of $\boldsymbol{\Omega}^l$ (i.e. $\boldsymbol{F}^l\boldsymbol{x} = \boldsymbol{g}^l$), it is given by $\boldsymbol{u}^l$. An example of such partition for the 2D system is depicted in Figure 4.5.

In the case of eIC, the explicit solution is obtained similarly as in IC, except that the solution for two domains is obtained. First, a solution for $\boldsymbol{\Omega}^l \setminus \boldsymbol{\Omega}^h$ is found, which is then complemented by the solution for $\boldsymbol{\Omega}^m$.

5 Dissertation Goals

In the previous chapters, the dissertation has laid out the theoretical foundation by discussing fundamental methods and procedures for both trajectory planning and tracking, as well as providing essential information about UAV behavior and control. The dissertation contribution will be built upon this foundation. Real-world UAV applications, such as intelligent agriculture, autonomous inspections in the industry, search-and-rescue missions, and others, employ trajectory planning as an essential component. Therefore, this will be the focus of the dissertation.

1. **Design of the Pseudospectral Method for Efficient Trajectory Planning for Unmanned Aerial Vehicle**

   As stated in Section 3.4, the Pseudospectral Methods are advantageous for UAV trajectory planning formulated as an Optimal Control Problem (OCP) because they allow for the direct employment of the UAV behavioral model and produce high-quality trajectories with considerably fast solution convergence. However, the convergence rate strongly depends on the initial guess of the solution and the number of collocation points. In practice, a simple initial guess, such as a constant or linear trajectory, is often used, leading to poor convergence.

   There is no straightforward general rule for selecting a sufficient number of collocation points, as this depends on the nonlinear dynamics and constraints of the problem. If the approximation error is large due to an insufficient number of collocation points, their number can be adjusted at each subsequent iteration. This, however, comes at the cost of increased computation time.

   The first goal of the dissertation is to design a Pseudospectral Method for UAV trajectory planning that is computationally efficient and provides high-quality results. This will be achieved through:

   (a) Acquiring a better initial guess using graph-based path planning methods and splitting the problem according to the initial path into several smaller interconnected segments.

   (b) Analysing a suitable version of the initial state and control trajectory guess based on UAV dynamics.

2. **Solution to the Trajectory Tracking Problem by Means of Interpolating Control**

   In Chapter 4, it was mentioned that Model Predictive Control (MPC, see Section 4.1) is widely used in state-of-the-art methods for trajectory tracking. However, as presented in Section 4.1.2, there are challenges associated with the computational demands of MPC when applied to fast-dynamic systems such as UAVs. These challenges often require compromises to be made, such as shortening the controlled or prediction horizon, increasing the sampling period, or utilizing suboptimal solutions. A more computationally

efficient alternative to MPC, called Interpolating Control (IC), was described in Section 4.2. The current IC method is applicable for stabilizing the system towards the origin.

The second goal of the dissertation thesis is to design a trajectory tracking method based on Interpolating Control (IC) that can be implemented for the UAV. This will involve:

(a) Adapting the IC currently suitable for the stabilization of the system to the origin and to include trajectory tracking features.

(b) Considering the future development of the reference trajectory directly in the control law.

(c) Testing the developed controller in a laboratory setting, where its performance will be evaluated by tracking a trajectory with a UAV.

6 Trajectory Planning with Path Processing for UAV

Building upon the concepts introduced in Chapter 3, this chapter delves into the specific challenges and proposed solutions surrounding trajectory planning for UAVs. As previously established, trajectory planning goes beyond path planning, as it involves a time-parametrized state and control variables of the UAV, including position, orientation, and their derivatives, together with a time-parametrized curve of control inputs. In the context of UAVs, this planning must account for the agility of these vehicles and the time constraints related to their flight durations.

As noted in Chapter 3, efficient trajectory planning for UAVs presents a unique set of challenges. The Pseudospectral Method (PSM) discussed in Section 3.4 provides a promising approach to address these challenges by offering fast convergence and high-quality trajectories when solving the Optimal Control Problem (OCP). However, the method's performance can significantly improve with an accurate initial guess and an optimal number of collocation points.

The challenge arises in determining these crucial factors given the nonlinear dynamics and constraints of the system. Simple initial guesses, such as constant or linear trajectories, can lead to poor convergence, necessitating a higher number of collocation points and more iterations of replanning. This increase, in turn, prolongs the computation time and undermines the efficiency of the method.

This thesis explores ways to enhance the efficiency of the trajectory planning process involving a PSM and Pseudospectral Elements Method (PSEM) while maintaining high precision and accuracy. The goal is to determine if a less computationally demanding approach can be used. Even though the trajectory is being planned, the initial path will be very important for building the starting point for this process. This goal is pursued through:

- The use of graph-based path planning methods to yield a more accurate initial guess and splitting the problem according to the initial path, into smaller interconnected problems to reduce computational time in Section 6.2.

- Evaluating the impact of initial state and control trajectory guesses on UAV trajectory tracking, tailored to the specific dynamics of the vehicle in Section 6.3.

6.1 Pseudospectral Methods for UAV Trajectory Planning

In this section, the fundamentals for both PSM and PSEM are presented. Both methods leverage approximation by Chebyshev polynomials of the first kind. The relations used for both methods are adapted from [91]. Given PSEM being an extension of PSM as it concurrently solves

multiple PSM problems on interconnected segments, the common principles will be described for the PSM.

6.1.1 Pseudospectral Method

As mentioned, the core of the PSM is the use of Chebyshev polynomial approximation on Gauss-Lobatto collocation points, which are defined as

$$\xi_j = \cos\left(\frac{\pi j}{N}\right), j = 0, 1, \ldots, N, \tag{6.1}$$

where N is the number of collocation points of the optimal grid for approximation by Chebyshev polynomial with degree $N-1$. These points are non-uniformly spaced and clustered towards both ends of the interval, which aids in capturing high-frequency dynamics near the start and end of the trajectory. Given the use of the Chebyshev polynomial, the OCP needs to be rescaled to the interval $\xi_j \in [-1, 1]$ because the OCP is described on the interval $\boldsymbol{t} \in \left[t_0, t_f\right]$. The grid can be rescaled as

$$t_i = \frac{1}{2}\left(t_f - t_0\right)\xi_j + \frac{1}{2}\left(t_0 + t_f\right), \tag{6.2}$$

where t_f is a terminal time of OCP, which can be fixed or varying and t_0 is an initial time of OCP, usually given as $t_0 = 0$.

For the approximation of the entire OCP, it is essential to describe the approximation of inherent derivatives, enabling the UAV dynamics to be approximated by Chebyshev polynomial. A relation between a polynomial and its derivative is linear. Thus, it can be calculated using simple matrix-vector multiplication by a differentiation matrix $\boldsymbol{D}_N$ which is built as

$$(D_N)_{00} = \frac{2N^2+1}{6}, \quad (D_N)_{NN} = -(D_N)_{00}, \tag{6.3}$$

$$(D_N)_{ii} = -\frac{\xi_i}{2(1-\xi_i^2)}, \quad i = 1, \ldots, N, \tag{6.4}$$

$$(D_N)_{ij} = \frac{c_i}{c_j}\cdot\frac{(-1)^{i+j}}{(\xi_i - \xi_j)}, \quad i \neq j,\ i, j = 0, \ldots, N, \tag{6.5}$$

where $(D_N)_{ij}$ denotes the element of matrix $\boldsymbol{D}_N$ in the i-th row and j-th column. $c_0 = c_N = 2$ and $c_i = 1$ for $i = 1, \ldots, N-1$. Similarly to collocation points, the differentiation matrix must also be scaled for interval $\boldsymbol{t} \in \left[t_0, t_f\right]$ as

$$\boldsymbol{D}_N = \frac{2\boldsymbol{D}_N^{[-1,1]}}{\left(t_f - t_0\right)}, \tag{6.6}$$

where $\boldsymbol{D}_N^{[-1,1]}$ is the differentiation matrix for the original interval $\xi_j \in [-1, 1]$ and $\boldsymbol{D}_N$ is the scaled differentiation matrix for interval $\boldsymbol{t} \in \left[t_0, t_f\right]$.

Besides the derivative approximation, an approximation of the integral is also required for a full description of the objective function. The Clenshaw-Curtis quadrature, by utilizing the special properties of Chebyshev points related to discrete cosine transformation, enables these weights to be effectively calculated, ensuring a reasonable approximation of the integral

across the given interval. The integral weights $\boldsymbol{w}_N^{[-1,1]}$ are acquired using Algorithm 1 and scaled for interval $\boldsymbol{t} \in \left[t_0, t_f\right]$ as

$$\boldsymbol{w}_N = \frac{1}{2}\left(t_f - t_0\right) \cdot \boldsymbol{w}_N^{[-1,1]}, \tag{6.7}$$

where $\boldsymbol{w}_N$ is the vector of weights for interval $\boldsymbol{t} \in \left[t_0, t_f\right]$ and $\boldsymbol{w}_N^{[-1,1]}$ is the scaled vector of weights for interval $\boldsymbol{\xi} \in [-1, 1]$.

Algorithm 1 Clenshaw-Curtis Weights

procedure CLENCURT(N)
  $\theta_j \leftarrow \frac{\pi j}{N}$, $j = 0, 1, \ldots, N$
  $\boldsymbol{w}_N^{[-1,1]} \leftarrow \mathbf{0}_{1\times(N+1)}$
  $i \leftarrow [1, \ldots, N-1]$
  $\boldsymbol{v} \leftarrow \mathbf{1}_{(N-1)\times 1}$
  if $\mathrm{mod}(N, 2) = 0$ **then**
    $w_0^{[-1,1]} \leftarrow \frac{1}{N^2-1}$
    $w_N^{[-1,1]} \leftarrow w_0^{[-1,1]}$
    for $k \leftarrow 1$ **to** $N/2$ **do**
      $\boldsymbol{v} \leftarrow \boldsymbol{v} - \frac{2\cos(2k\theta_i)}{4k^2-1}$
    end for
    $\boldsymbol{v} \leftarrow \boldsymbol{v} - \frac{\cos(N\theta_i)}{N^2-1}$
  else
    $w_0^{[-1,1]} \leftarrow \frac{1}{N^2}$
    $w_N^{[-1,1]} \leftarrow w_0^{[-1,1]}$
    for $k \leftarrow 1$ **to** $(N-1)/2$ **do**
      $\boldsymbol{v} \leftarrow \boldsymbol{v} - \frac{2\cos(2k\theta_i)}{4k^2-1}$
    end for
  end if
  $\boldsymbol{w}_i^{[-1,1]} \leftarrow 2\boldsymbol{v}/N$
  return $\boldsymbol{w}_N^{[-1,1]}$
end procedure

6.1.2 Pseudospectral Elements Method

As highlighted above, the PSEM is based on the principle of solving several PSM problems, corresponding to different, yet interconnected, segments. The grid with collocation points is characterized as follows.

$$c_j^{(k)} = \cos\left(\frac{\pi j}{N}\right), j = 0, 1, \ldots, N,\ k = 0, 1, \ldots, M-1, \tag{6.8}$$

$$t_k^{(k)} = \frac{1}{2}\left(t_f^{(k)} - t_0^{(k)}\right) c_j^{(k)} + \frac{1}{2}\left(t_0^{(k)} + t_f^{(k)}\right), \tag{6.9}$$

where k is the index of the segment, M is the total number of segments, and the interconnected segments satisfy continuity conditions

$$t_f^{(k)} = t_0^{(k+1)},\ \boldsymbol{x}_f^{(k)} = \boldsymbol{x}_0^{(k+1)},\ \dot{\boldsymbol{x}}_f^{(k)} = \dot{\boldsymbol{x}}_0^{(k+1)},\ \boldsymbol{u}_f^{(k)} = \boldsymbol{u}_0^{(k+1)}. \tag{6.10}$$

6.1.3 Solution Error and Mesh Refinement Scheme

Solving complex problems using PSM and PSEM is typically an iterative process to ensure sufficient solution accuracy. It may be beneficial to first find an approximate solution with a smaller number of collocation points. Although this solution lacks the required accuracy, it can provide useful information to increase the number of points and search for a more accurate solution. This section describes the evaluation of the solution and the iterative mesh refinement method used to increase accuracy.

To evaluate the accuracy of the PSM, the discretization error $\epsilon_d(t)$ is computed. This is accomplished by finding the difference between the derivative of the state polynomial and the nonlinear dynamics function, which are both evaluated at the state and control polynomials. The discretization error can be expressed as

$$\epsilon_d(t) = \frac{dp_{\boldsymbol{x}}(t)}{dt} - f(p(\boldsymbol{x}(t)), p(\boldsymbol{u}(t)), t), \tag{6.11}$$

where $p(\boldsymbol{x}(t))$ is state polynomial approximation trajectory, $p(\boldsymbol{u}(t))$ is control polynomial approximation trajectory, and $\frac{dp_{\boldsymbol{x}}(t)}{dt}$ is the derivative of state trajectory polynomial approximation.

The absolute discretization error $\epsilon_{a,i}$ is evaluated between i-th and $i+1$-th collocation point as

$$\epsilon_{a,i} = \int_{t_i}^{t_{i+1}} |\epsilon_d(t)| dt, \tag{6.12}$$

where the integral is evaluated numerically over each interval using composite Simpson's rule with a default number of points $N_{simp} = 10$. This implies that eight additional points are sampled between each pair of collocation points.

The relative error $\epsilon_{r,i}$ is calculated as the absolute error divided by the average value of the corresponding state variable in the interval $[t_i, t_{i+1}]$ in points given by composite Simpson's rule as

$$\epsilon_{r,i} = \frac{\epsilon_{a,i}}{\frac{1}{N_{simp}} \sum_{t_l=t_i}^{t_{i+1}} p_{\boldsymbol{x}}(t_l)}. \tag{6.13}$$

In the case of both, the absolute and the relative error, the maximum value of the error for each collocation point is sought as a maximum error value. The maximum absolute error $\epsilon_{a_{\max},i}$ is given as

$$\epsilon_{a_{\max},i} = \max \epsilon_{a,i} \tag{6.14}$$

and for the maximum relative error $\epsilon_{r_{\max},i}$ as

$$\epsilon_{r_{\max},i} = \max \epsilon_{r,i}. \tag{6.15}$$

The PSM seeks the solution iteratively by increasing the degree of the polynomial until the given error tolerance is reached. The polynomial degree increase is determined by the formula

$$P_k = \max\left(\lceil \ln(N_k, \epsilon_{a_{\max}}/\epsilon) \rceil, 3\right), \tag{6.16}$$

where P_k represents the increase in collocation points number, N_k is the original number of collocation points, $\epsilon_{a_{\max}}$ is the maximum absolute discretization error on interval $t \in < t_0, t_f >$, ϵ is the error tolerance, $\lceil \cdot \rceil$ denotes the ceiling function and the operator $\ln(\cdot, \cdot)$ is the logarithm function with base as the second argument.

The PSM incorporates a p-refinement scheme, that employs the past solution as an initial guess for the successive iteration. This enhances the convergence rate and reduces the computational cost of solving the NLP problem.

The PSEM scheme differs significantly from the PSM when the solution accuracy is insufficient. Instead of merely increasing the degree of the polynomial as in PSM, the PSEM allows for the division of the segment into two or more sections, based on the relative error, to modify the segment and enhance accuracy.

If the maximum relative error $\epsilon_{r_{\max}}$ within a segment exceeds the *rel_tol* value, the segment will be divided. For a split into two segments, the location with the highest relative deflection $\Delta\epsilon_{r_{\max},i}$ is pinpointed, and the segment is divided at that point. The relative deflection is given as

$$\Delta\epsilon_{r_{\max},i} = \epsilon_{r_{\max},i+1} - \epsilon_{r_{\max},i}. \tag{6.17}$$

If there are multiple splits, the segment is divided at each collocation point that exceeds *rel_tol*.

If the relative error for each point in the segment is below the threshold, the degree of the polynomial is increased according to relation (6.16).

The collocation points from the original segment are proportionally divided among the new segments. If a new segment has less or equal to *p_min* points, which refers to the minimum polynomial degree per segment, the number of points is increased to *p_min*+1.

The updated mesh with the refined collocation points is determined first. Then, polynomial fitting is used to fit the state and control trajectories to the new collocation points. This propagates the solution to the refined mesh.

After propagating the trajectories, they are incorporated as an initial guess in the next iteration of searching for the optimal solution.

6.2 Improvement of Initial Guess through Graph-Based Path Planning Methods

In this section, an approach is proposed to improve the solution provided by the PSM and PSEM through the use of the graph-based path planning method Lazy Theta* (LT*) for the initial guess acquisition. The quality of the initial guess for trajectory planning directly affects the convergence of the resulting solution. Traditional methods that use a simple initial guess, such as a linear or constant path, can lead to poorly converging results. To address this issue, the use of LT* is suggested to obtain an informed initial guess.

Papers [99] and [83] both investigate trajectory planning for UAVs, but their approaches to generating initial guesses for the trajectory planning process were different. Paper [99] utilized an online topological path planning approach to generate a comprehensive set of distinctive paths that guide the optimization process. Paper [83] used the RRT* algorithm to generate an initial route and then constructed a trajectory consisting of a sequence of polynomial spline segments to follow that route. In contrast, our proposed approach is focused on using LT* for generating the initial guess.

The choice of LT* over RRT* is motivated by its ability to find paths between line-of-sight nodes on a grid map, which can result in more direct paths with fewer waypoints, as mentioned in Section 3.2.2. This can be advantageous for UAVs that can move in any direction, as it may lead to a more efficient trajectory planning process.

In the proposed approach, the path generated by LT* is processed in multiple ways to form an enhanced initial guess for the PSM and PSEM. This not only includes a guess of the position in the trajectory and its segmentation but also contemplates a variant that processes the path according to the UAV's constraints. This is done in order to gain an initial guess of other state trajectory elements as well as the control trajectory.

6.2.1 LT* Path Generation

The LT* algorithm works with a grid map consisting of obstacles and free spaces in the environment. This map is formulated in 2D or 3D, with identified occupied cells falling within obstacle boundaries. Once the start and goal positions of the UAV trajectory are entered into LT*, the algorithm searches for an optimal path on the grid, where "optimal" refers to minimizing the path length, as given by the Euclidean distance between waypoints, while avoiding collisions.

The algorithm searches for a sequence of directly connected grid points that are within the line of sight between the start and end points. The line-of-sight inspection is executed using Bresenham's algorithm, which works both for 3D[1] and 2D[2] grid and generates the line of sight between two points.

The waypoints created by the path serve as an informed starting point for developing an initial guess of the trajectory while taking into account the constraints and dynamics of the UAV. Examples of paths are shown in Figure 6.1. The occasional intersection of the path with the corners of obstacles is due to the specific working of the LT* algorithm in conjunction with gridmap representations and the definition of obstacles within. To avoid passing through obstacles directly, the line-of-sight check is determined. In order to prevent the path from grazing obstacle corners, obstacles should not be represented as single cells within the gridmap, but rather as a collection of cells representing their vertices. However, these minor inconveniences are inconsequential. The initial path will only be used to obtain an initial guess of the trajectory. The problem of trajectory planning will be defined in a way that ensures the optimal trajectory completely avoids obstacles.

6.2.2 Time Parameterization

The LT* algorithm assigns a set of waypoints $\boldsymbol{S}$ along the path as the locations for the trajectory segments

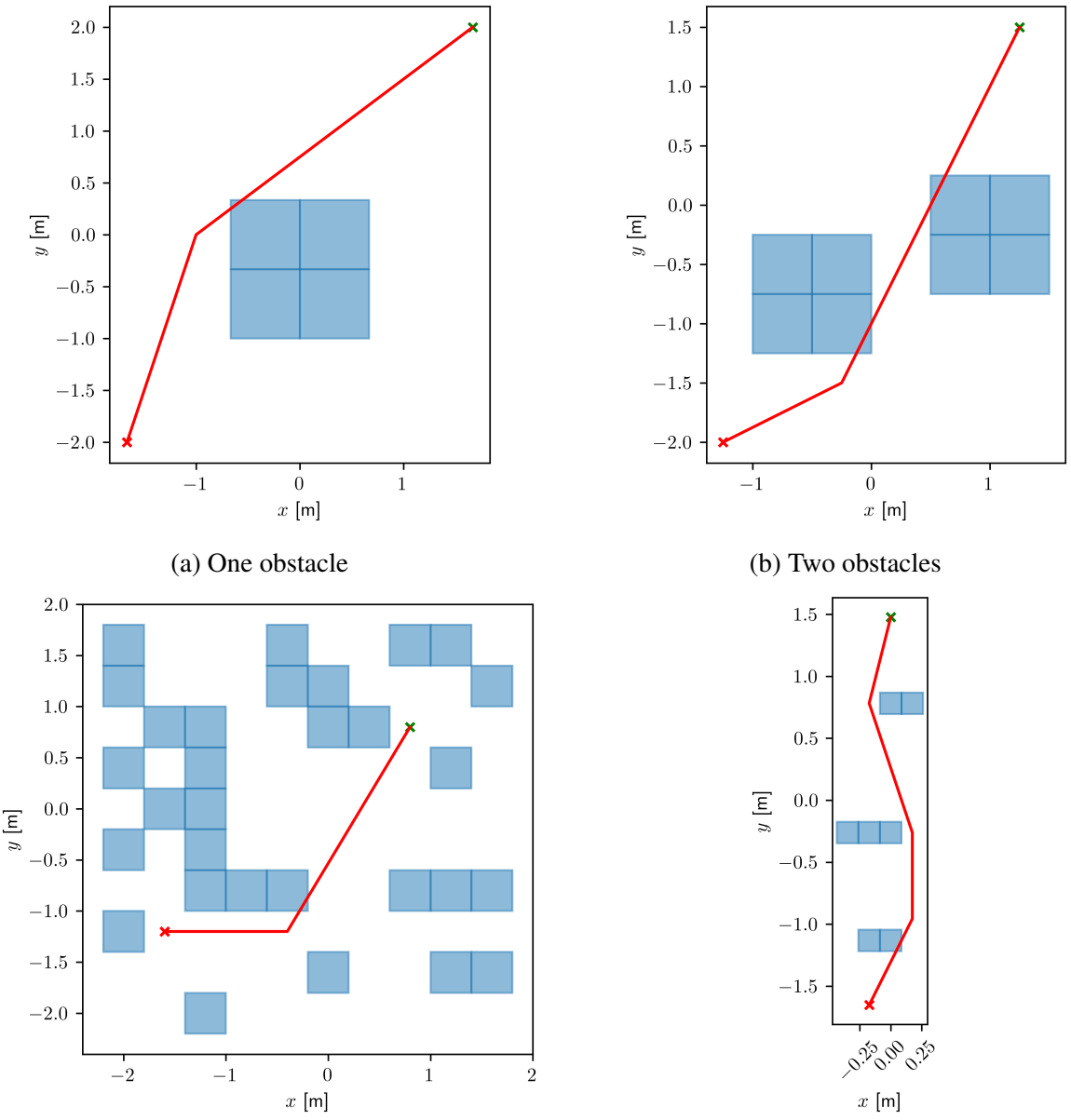
$$\boldsymbol{S} = [\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_M], \boldsymbol{s}_0 = \boldsymbol{s}_{start} = \boldsymbol{r}_0^L, \boldsymbol{s}_M = \boldsymbol{s}_{goal} = \boldsymbol{r}_f^L, \tag{6.18}$$

where M is the total number of segments. The starting point $\boldsymbol{s}_{start}$ marks the initiation of the first segment, the intermediate waypoints $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{M-1}$ indicate the nodes between segments, and the final $\boldsymbol{s}_{goal}$ waypoint terminates the trajectory at the desired goal position $\boldsymbol{r}_f^L$. If there is no defined path, the algorithm only uses the start and goal positions based on boundary conditions $\boldsymbol{r}_0^L$ and $\boldsymbol{r}_f^L$.

The distances between successive waypoints $\Delta \boldsymbol{s}_i$ are calculated as

$$\Delta \boldsymbol{s}_i = |\boldsymbol{s}_{i+1} - \boldsymbol{s}_i|, i = 0, \ldots, M-1. \tag{6.19}$$

[1] 3D Bresenham: `geeksforgeeks.org/bresenhams-algorithm-for-3-d-line-drawing`
[2] 2D Bresenham: `geeksforgeeks.org/chain-code-for-2d-line`



(a) One obstacle

(b) Two obstacles

(c) Random columns

(d) Walls

Figure 6.1: LT* Path Generation

If the path is two-dimensional, the $\vec{z}^L$-axis element is completed based on the boundary condition, where the movement is assumed to be linear in the context of the segment knot points. The $\vec{z}^L$ coordinates are then given as

$$z^L(t) = z_0^L + \frac{t - t_0}{t_f - t_0} \cdot \left(z_f^L - z_0^L\right). \tag{6.20}$$

Furthermore, the time required for the UAV to travel between each pair of waypoints, considering its maximum velocity constraint, is computed as

$$\Delta t_f^{(i)} = \max\left(\frac{\Delta \boldsymbol{s}_i}{\dot{\boldsymbol{r}}_{\max}^L}\right), i = 0, \ldots, M-1, \tag{6.21}$$

where the longest time required to reach the next waypoint along each axis, and $\dot{\boldsymbol{r}}_{\max}^L$ is the maximum velocity of UAV in $\vec{x}^L$, $\vec{y}^L$, and $\vec{z}^L$ directions. Adding all the intervals provides an optimistic total time as

$$t_f = \sum_{i=0}^{M-1} \Delta t_f^{(i)}. \tag{6.22}$$

Nevertheless, the optimal trajectory is expected to require a longer time frame, which must be adjusted by the NLP solver.

The time grid is generated between segments based on the number of Chebyshev Gauss-Lobatto points per segment defined by the input parameter. For single-segment initialization, points are sampled between the initial and final times.

In multi-segment cases, the grid is constructed sequentially for each segment span by using the allocated Gauss-Lobatto points. Moreover, PSEM can be also initialized with a single-segment initial guess.

6.2.3 State and Control Initial Guess

The initial guesses for state and control trajectories are created hierarchically, with each level building upon the previous one. This approach enables progressively increasing complexity and accuracy. The level names reflect the last calculated component in the initial guess. The velocity level includes a guess of the position in time based on the path and a guess of the velocity. In angular rate control, the guess calculates all components of the state and control trajectory, including the initial guess of collective thrust and torque. Table 6.1 shows the construction of each component's initial guess. Further details will be provided in the subsequent sections.

Within each level, guesses can be refined by imposing constraints on the lower and upper bounds and boundary conditions for both the state and control trajectories. The state and control trajectories are developed using the UAV's behavior, described in Chapter 2. Quaternion operations, as described in Appendix A, and SE(3) stabilizing controller relations, detailed in Appendix C, are also used. The single-domain trajectory estimate is derived from the multi-segment estimate, which is evaluated at single-domain collocated points.

6.2.3.1 State Initial Guess

The position, velocity, orientation, and angular rate are initialized to their corresponding terminal boundary values of $\boldsymbol{x}_f$. Furthermore, the control trajectory, composed of thrust and torque, is set

Table 6.1: Summary of initial guess construction for state and control trajectories (separated by horizontal line)

Component	Method	Inputs	Outputs	Purpose
Simple	Straight line interpolation	$\boldsymbol{r}_0^L, \boldsymbol{r}_f^L$	$\boldsymbol{r}^L(t)$	Basic path planning
Position	Spline	LT* path waypoints	$\boldsymbol{r}^L(t)$	Smooth path following
Velocity	Differentiation of position	$\boldsymbol{r}^L(t)$	$\dot{\boldsymbol{r}}^L(t)$	Smooth velocity profile
Orientation	Quaternion curve	$\dot{\boldsymbol{r}}^L(t), \mathbf{F}_r^L(t)$	$\boldsymbol{q}(t)$	Align with expected forces
Angular rate	Quaternion derivative	$\boldsymbol{q}(t)$	$\boldsymbol{\omega}(t)$	Follow orientation changes
Thrust	Quaternion rotation of force	$\boldsymbol{q}(t), \mathbf{F}_r^L(t)$	$T(t)$	Align with movement and orientation
Torque	Dynamic equation	$\boldsymbol{\omega}(t), \dot{\boldsymbol{\omega}}(t)$	$\boldsymbol{\tau}(t)$	Achieve desired angular motion

to the final control values $\boldsymbol{u}_f$ for all time steps. That is given by the definition of a problem where the boundary state and control conditions are equal except for the position. Nonetheless, this level of guess is not used, as a fundamental path plan is created based on the boundary conditions, namely the initial and final positions.

The simplest initial guess is a straight linear interpolation between the initial and final state and control. The endpoint time is set to $t_f = \frac{1}{2}\left(t_{f_{\max}} + t_{f_{\min}}\right)$.

At the position level, the LT* path waypoints are fitted by the polynomial or cubic spline, with the latter being chosen due to its smoothness. These curves are parameterized by the previously established time grid. At this level, the position guess $\boldsymbol{r}^L$ closely follows the LT* path over time, leaving velocity, orientation, and rates are set according to the terminal constraint.

The velocity level is calculated through the differentiation of the position trajectory

$$\dot{\boldsymbol{r}}^L = \frac{d\boldsymbol{r}^L}{dt}, \tag{6.23}$$

resulting in derivative curves that represent the initial velocity guess. This method ensures that the velocity guess avoids sudden changes by inheriting the smoothness of the parent position trajectory. The orientation and angular rate retain their endpoint values.

The orientation level guess begins by calculating an acceleration trajectory from the velocity guess via differentiation as

$$\ddot{\boldsymbol{r}}^L = \frac{d\dot{\boldsymbol{r}}^L}{dt}. \tag{6.24}$$

The total expected force over time, denoted by $\boldsymbol{F}_r^L$, is determined using the gravitational constant g and mass m as

$$\boldsymbol{F}_r^L = m \cdot \left(\ddot{\boldsymbol{r}}^L + g\vec{z}^L\right). \tag{6.25}$$

This force vector is then utilized to generate an orientation quaternion curve utilizing UAV

relations derived from Equation (C.8) for the SE(3) controller as

$$\boldsymbol{q} = \frac{1}{\sqrt{2\left(1 + \frac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} \cdot \frac{\boldsymbol{F}_r^L}{\|\boldsymbol{F}_r^L\|}\right)}} \begin{bmatrix} 1 + \frac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} \cdot \frac{\boldsymbol{F}_r^L}{\|\boldsymbol{F}_r^L\|} \\ \frac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} \times \frac{\boldsymbol{F}_r^L}{\|\boldsymbol{F}_r^L\|} \end{bmatrix}, \tag{6.26}$$

where $\frac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} = [0, 0, 1]^\mathsf{T}$. The angular rates continue to rely on the terminal constraint.

The angular rate level guess is based on the quaternion guess. The quaternion derivative is acquired from the quaternion trajectory as

$$\dot{\boldsymbol{q}} = \frac{d\boldsymbol{q}}{dt} \tag{6.27}$$

and subsequently the angular rate nodes are computed using Equation (A.19) as

$$\boldsymbol{\omega} = 2\boldsymbol{\Gamma}(\boldsymbol{q})\dot{\boldsymbol{q}} = -2\boldsymbol{\Gamma}(\dot{\boldsymbol{q}})\boldsymbol{q}. \tag{6.28}$$

The resulting angular rate nodes are used to construct the angular rate trajectory.

6.2.3.2 Control Initial Guess

The thrust level uses the aforementioned quaternion curve in conjunction with the expected force vector. The thrust T is obtained through quaternion rotation of expected force to the body frame, where the thrust corresponds to the $\vec{z}^B$-axis value as

$$\boldsymbol{F}_r^B = \boldsymbol{q}^{-1} \otimes \boldsymbol{F}_r^L \otimes \boldsymbol{q}, T = \boldsymbol{F}_{r,z}^B. \tag{6.29}$$

Once the nodes are computed, the thrust trajectory is acquired by fitting the curve. This thrust trajectory aligns with the UAV's movement in the local frame and orientation provided by the quaternion trajectory.

Finally, the torque guess is based on the complete rotational dynamics equation for the UAV described in Equation (2.6). The inputs consist of the known inertia matrix $\boldsymbol{I}$, the angular rate trajectory derived from the quaternion trajectory, and the angular acceleration profile obtained as the derivative of the angular rate trajectory. The formula for calculation of the torque for initial guess is

$$\boldsymbol{\tau} = -\boldsymbol{I}\dot{\boldsymbol{\omega}} + (\boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}). \tag{6.30}$$

Combining these terms using the dynamics formula produces a torque trajectory that aligns with the desired angular rate and orientation.

6.3 Numerical Illustrations

The practical part of this chapter follows the theoretical discussion in Sections 6.1 and 6.2, which explored Pseudospectral Methods (PSM) and Pseudospectral Elements Methods (PSEM), and construction of the initial guess. This section introduces the practical problem of UAV trajectory planning.

Several scenarios were designed for testing, including flying around a simple obstacle, navigating around columns, and flying in a space with randomly placed obstacles. These virtual environments were rescaled to fit the laboratory space, allowing for the feasibility of testing these scenarios in a limited area.

This chapter discusses the practical application of theoretical concepts of PSM and PSEM, with a focus on the role of the initial guess in the trajectory planning process and its impact on solution convergence. The presented insights and results aim to demonstrate the effectiveness and potential limitations of these methods in real UAV navigation scenarios.

6.3.1 Problem Formulation

The problem of UAV trajectory planning is described in detail in this section. It includes a description of the UAV parameters and the associated constraints, including those for collision avoidance with static obstacles. The obstacle constraints are derived from a list specific to each virtual environment. Additionally, the objective function, crucial for finding the optimal trajectory, is described. This function is essential in guiding the UAV through different scenarios while adhering to physical constraints and avoiding obstacles to achieve navigational goals.

6.3.1.1 UAV dynamics

The UAV model described in Chapter 2 remains unchanged as it sufficiently captures the key UAV dynamics necessary for trajectory optimization. The UAV state vector, denoted as $\boldsymbol{x}(t)$, is represented by its position $\boldsymbol{r}^L$, velocity $\dot{\boldsymbol{r}}^L$, orientation quaternion $\boldsymbol{q}$, and angular rate $\boldsymbol{\omega}$, fully describing the aircraft's motion in 3D space. The state vector is given as

$$\boldsymbol{x}(t) = \left[\boldsymbol{r}^L(t)^\mathsf{T}, \dot{\boldsymbol{r}}^L(t)^\mathsf{T}, \boldsymbol{q}(t)^\mathsf{T}, \boldsymbol{\omega}(t)^\mathsf{T}\right]^\mathsf{T}. \tag{6.31}$$

The UAV is actuated by the collective thrust $\boldsymbol{F}_T^B$, which provides a lift for altitude control, and collective torque $\boldsymbol{\tau}_R$, which allows agile maneuvering and attitude changes. The control input is given as

$$\boldsymbol{u}(t) = \left[\boldsymbol{F}_T^B(t)^\mathsf{T}, \boldsymbol{\tau}_R(t)^\mathsf{T}\right]^\mathsf{T}. \tag{6.32}$$

For completeness, the UAV equations of motion from Chapter 2 are restated below

$$\ddot{\boldsymbol{r}}^L = -g\,[0,\,0,\,1]^\mathsf{T} + \frac{\boldsymbol{F}^L}{m}, \tag{6.33}$$

$$\dot{\boldsymbol{\omega}} = -\boldsymbol{I}^{-1}(\boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}) + \boldsymbol{I}^{-1}\boldsymbol{\tau}, \tag{6.34}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{\Gamma}(\boldsymbol{q})^\mathsf{T}\boldsymbol{\omega}, \tag{6.35}$$

where g is the gravity constant, m is the mass of the UAV, $\boldsymbol{I}$ is the inertia matrix of the UAV, and $\boldsymbol{\Gamma}$ is the quaternion dynamics matrix given as

$$\boldsymbol{\Gamma}(\boldsymbol{q}) = \begin{bmatrix} -q_x & q_w & q_z & -q_y \\ -q_y & -q_z & q_w & q_x \\ -q_z & q_y & -q_x & q_w \end{bmatrix}. \tag{6.36}$$

To complete the model, it is necessary to include the description of forces and torques acting on the UAV. The generalized forces $\boldsymbol{F}^L$ are given as

$$\boldsymbol{F}^L = \boldsymbol{q} \otimes \left(\boldsymbol{F}_T^B + \boldsymbol{F}_A^B\right) \otimes \boldsymbol{q}^* = \boldsymbol{R}_{\phi\theta\psi}(\boldsymbol{q})^\mathsf{T} \left(\boldsymbol{F}_T^B + \boldsymbol{F}_A^B\right), \tag{6.37}$$

where $\boldsymbol{q}^*$ is the conjugate of $\boldsymbol{q}$, $\otimes$ is the operator for quaternion product, $\boldsymbol{R}_{\phi\theta\psi}$ is the body-to-local frame rotation matrix, and $\boldsymbol{F}_A^B$ are aerodynamic forces given as

$$\boldsymbol{F}_A^B = -\left(F_{T,z}^B\right) \cdot \left(\boldsymbol{K}_D \dot{\boldsymbol{r}}^B\right)^\mathsf{T}, \tag{6.38}$$

where $\boldsymbol{K}_D$ is the lumped drag coefficient matrix and the collective thrust along the $\vec{z}^B$-axis. In the case of torque $\boldsymbol{\tau}$, the rotor inertia τ_{I_z} is assumed to be zero, therefore $\boldsymbol{\tau} = \boldsymbol{\tau}_R$.

The UAV parameters listed in Table 6.2 are based on the Crazyflie microUAV, described in [32], where a comparison of attributes from physical modeling to experimental data was performed to ensure a realistic model of the UAV's behavior. These parameters ensure a realistic and accurate representation of the Crazyflie microUAV in the considered model.

Table 6.2: Crazyflie UAV Model Parameters

Parameter	Symbol	Value
Gravitational acceleration	g	9.81305 m/s^2
UAV mass	m	0.032 kg
Arm length	l	0.0397 m
Moment of inertia, $\vec{x}^B$-axis	I_x	$6.410179 \cdot 10^{-6}$ kg·m^2
Moment of inertia, $\vec{y}^B$-axis	I_y	$6.410179 \cdot 10^{-6}$ kg·m^2
Moment of inertia, $\vec{z}^B$-axis	I_z	$9.860228 \cdot 10^{-6}$ kg·m^2
Propeller diameter	d_p	0.051 m
Aerodynamic coefficient	$\boldsymbol{K}_D$	$-1 \cdot 10^{-7} \cdot \begin{bmatrix} 10.2506 & 0.3177 & 0.4332 \\ 0.3177 & 10.2506 & 0.4332 \\ 7.7050 & 7.7050 & 7.5530 \end{bmatrix}$

6.3.1.2 Optimality Criterion

The general optimality criterion has been described in Equation (3.3). Further, the optimality criterion which is designed for the UAV trajectory optimization problem will be described. It evaluates both the state and the control deviation from their desired values over the trajectory. An additional term is included to minimize the quaternion distance. Since the orientation is represented by a quaternion, it is not appropriate to evaluate its deviation by a simple difference. Therefore, the distance between the current orientation $\boldsymbol{q}(t)$ and the final orientation $\boldsymbol{q}(t_f)$ is calculated as $|1 - \boldsymbol{q}(t)^T \cdot \boldsymbol{q}(t_f)|$. To evaluate the quaternions separately, a new symbol $\boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t)$ is introduced for the vector, which contains only position, velocity, and angular rate, given as

$$\boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t) = \left[\boldsymbol{r}^L(t)^\mathsf{T}, \dot{\boldsymbol{r}}^L(t)^\mathsf{T}, \boldsymbol{\omega}(t)^\mathsf{T}\right]^T. \tag{6.39}$$

The optimality criterion J in Equation (6.40) is defined as an integral over the flight time, from t_0 to t_f. This function measures the deviations between the UAV's current state $\boldsymbol{x}(t)$ and

control $\boldsymbol{u}(t)$ at any time t, and their respective desired final values $\boldsymbol{x}(t_f)$ and $\boldsymbol{u}(t_f)$.

$$\begin{aligned} J(\boldsymbol{x}(t), \boldsymbol{u}(t), t_0, t_f) = \int_{t_0}^{t_f} \Big[& \left(\boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t) - \boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t_f)\right)^T \boldsymbol{Q}_{\boldsymbol{x}\setminus\boldsymbol{q}} \left(\boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t) - \boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t_f)\right) \\ & + \boldsymbol{Q}_{\boldsymbol{q}} |1 - \boldsymbol{q}(t)^T \cdot \boldsymbol{q}(t_f)| \\ & + \left(\boldsymbol{u}(t) - \boldsymbol{u}(t_f)\right)^T \boldsymbol{R} \left(\boldsymbol{u}(t) - \boldsymbol{u}(t_f)\right) \Big] dt \end{aligned} \tag{6.40}$$

To achieve stable and energy-efficient flight, the deviation of the state without orientation $\boldsymbol{x}_{\boldsymbol{x}\setminus\boldsymbol{q}}(t)$, the quaternion $\boldsymbol{q}$, and the control $\boldsymbol{u}(t)$ are weighted by $\boldsymbol{Q}_{\boldsymbol{x}\setminus\boldsymbol{q}}$, $\boldsymbol{Q}_{\boldsymbol{q}}$, and $\boldsymbol{R}$, respectively. This weighting ensures favorable UAV behavior. The weights were set as

$$\begin{aligned} \boldsymbol{Q}_{\boldsymbol{x}\setminus\boldsymbol{q}} &= \text{diag}\left(1, 1, 1, 1, 1, 1, 0.328, 0.328, 0.328\right), \ \boldsymbol{Q}_{\boldsymbol{q}} = 100, \\ \boldsymbol{R} &= \text{diag}\left(1.223 \cdot 10^1, 2.820 \cdot 10^4, 2.820 \cdot 10^4, 3.017 \cdot 10^5\right). \end{aligned} \tag{6.41}$$

6.3.1.3 Static Obstacles

In the UAV trajectory planning model, obstacle avoidance is handled through constraints that adapt based on the UAV's position and the location of obstacles. These obstacles are modeled either as 3D spheres or 2D circles. The constraints ensure that the UAV maintains a safe distance from these obstacles, taking into account its dimensions and an additional safety margin.

The constraints for 3D spherical obstacles are expressed as

$$\left(x^L(t) - x^L_{obs}\right)^2 + \left(y^L(t) - y^L_{obs}\right)^2 + \left(z^L(t) - z^L_{obs}\right)^2 \geq \left(r_{obs} + r_{safe}\right)^2, \tag{6.42}$$

where $x^L(t), y^L(t), z^L(t)$ are the coordinates of the UAV, $x^L_{obs}, y^L_{obs}, z^L_{obs}$ are the coordinates of the center of a spherical obstacle, r_{obs} is the radius of the obstacle, and r_{safe} is the safety radius around the UAV. The safety radius is calculated as $r_{safe} = \left(l + \frac{d_p}{2}\right) \cdot 1.1$, taking into account the arm length and half the propeller diameter with an additional safety margin of 10%. For 2D circular obstacles, the constraints are analogously given as

$$\left(x^L(t) - x^L_{obs}\right)^2 + \left(y^L(t) - y^L_{obs}\right)^2 \geq \left(r_{obs} + r_{safe}\right)^2. \tag{6.43}$$

In the pathfinding task, obstacles were considered on a 2D or 3D grid map. In OCP, however, they are considered in the form of columns or spheres. Therefore, the obstacle radius r_{obs} is calculated as $r_{obs} = \frac{\sqrt{3}}{2} \cdot r_{grid}$ for 3D, where r_{grid} is the side length of a single gridmap cell, and as $r_{obs} = \frac{\sqrt{2}}{2} \cdot r_{grid}$ for 2D. With such a radius, the entire obstacle is inscribed, and thus, by the definition of the problem, the optimal trajectory does not intersect with any obstacles.

6.3.1.4 State and Control Constraints, and Boundary Conditions

The UAV's safe operation is ensured by imposing box constraints on its state and control variables. These constraints define allowable ranges for position, velocity, orientation, angular rate, torque, and thrust. The constraints were designed based on the physical layout of the laboratory flight space and the specifications of the Crazyflie microUAV.

The state box constraints have the following specific values

$$[-2, -2, 0]^\mathsf{T} \leq \boldsymbol{r}^L(t) \leq [2, 2, 2]^\mathsf{T}, \ -2 \leq \dot{\boldsymbol{r}}^L(t) \leq 2, \tag{6.44}$$

$$0 \leq q_w(t) \leq 1, \ -1 \leq \boldsymbol{q}_v(t) \leq 1, \ -\infty \leq \boldsymbol{\omega}(t) \leq \infty, \tag{6.45}$$

and the control box constraints are as follows

$$0 \leq F_{T,z}^{B}(t) \leq 0.6, \tag{6.46}$$

$$[-5.955, -5.955, -1.82063]^{\mathsf{T}} \cdot 10^{-3} \leq \boldsymbol{\tau}_R(t) \leq [5.955, 5.955, 1.82063]^{\mathsf{T}} \cdot 10^{-3}. \tag{6.47}$$

Additionally, a constraint is enforced to maintain the unit quaternion norm, which is necessary for the representation of a valid orientation. The equality constraint for the quaternion norm is also given as

$$\|\boldsymbol{q}(t)\| = \sqrt{q_w(t)^2 + q_x(t)^2 + q_y(t)^2 + q_z(t)^2} = 1. \tag{6.48}$$

The initial and final conditions are set for a stabilized UAV state, with the specified position from the gridmap. Thus, in the state $\boldsymbol{x}(t_0)$ and $\boldsymbol{x}(t_f)$, the position corresponds to the start and goal, respectively. The quaternion is set to $\boldsymbol{q}(t_0) = \boldsymbol{q}(t_f) = [1, 0, 0, 0]^{\mathsf{T}}$ and the remaining state elements are set to zero. For the control vector, the thrust is fixed at $F_{T,z}^{B}(t_0) = F_{T,z}^{B}(t_f) = m \cdot g = 0.3140176$ to counter a gravitational force and collective torque is set to zero. The final time t_f is constrained between 0 and the maximum flight duration, which is set to 3 minutes for the Crazyflie. These conditions and constraints comply with physical limitations and the operating environment.

6.3.2 Solution Evaluation

The evaluation of the solution is performed using the discretization error, adherence to constraints, optimality criterion value, number of iterations, and the time taken to find the solution. These metrics objectively measure the accuracy and efficiency of the proposed method. The discretization error provides insight into the approximation quality of the numerical solution, while the adherence to constraints ensures that the solution satisfies all relevant conditions. The value of the optimality criterion serves as a quantitative measure of the performance of the solution, and the time required to find the solution indicates the computational efficiency of the method. Together, these measures provide a comprehensive and detailed evaluation of the proposed solution.

The evaluation of the proposed solution includes several metrics that assess both accuracy and efficiency. The discretization error, which is primarily used as a stopping metric for the trajectory planning algorithm, is set with a threshold of $\epsilon_{a_{\max}} = 10^{-2}$. This error provides insight into the quality of the numerical approximation and is measured in terms of the maximum absolute error defined in Equation (6.14).

Adherence to constraints is equally important and ensures that the solution satisfies all relevant operational and physical conditions. This adherence is evaluated similarly to the discretization error using the composite Simpson's rule with $N_{simp} = 10$. The evaluation also measures the extent to which the state and control box constraints are violated and collisions with obstacles, including the safety radius around the obstacle.

The value of the optimality criterion, described in Equation (6.40), is another key metric that provides a quantitative measure of the performance of the solution. The number of iterations, limited to 10, reflects the convergence efficiency of the method. The limit was chosen based on the computational efficiency of the method with the chosen maximum absolute error threshold. The computation time, another critical factor, is measured from the start of finding the trajectory search through each iteration to the end of the computation. To limit the evaluation process,

the maximum computation time is set to 5 hours, ensuring that solutions that do not converge within this time frame are not considered viable.

6.3.2.1 Effect of Transcription on Trajectory

A key result of approximating through collocation is that equality and inequality constraints are only enforced at collocation nodes, rather than continuously. Therefore, the state and control trajectories solved from the NLP between nodes are projections that could potentially violate path boundaries and obstacles. This limitation allows for collision encounters in practice, even when adhering to constraints at collocation points. For instance, the optimized trajectory may pass through an obstacle between the collocation points. The state values at the collocation points remain valid, even if the interlaced trajectory passes through an obstacle.

To reduce this effect, the total number of points can be increased - by densifying the grid, the constraints will be evaluated at a greater number of points. Additionally, division of the trajectory into segments at the point where an obstacle is present. This is because the Gauss-Lobatto collocation points are most densely distributed at the edges of the segments. It is possible to test for constraint compliance in a similar way to the discretization error in the loop. The grid can then be adaptively modified until the violations are below the desired error threshold. Additionally, using a trajectory tracking algorithm with collision avoidance capability could prevent encounters.

6.3.3 Setup and Implementation Details

The trajectory planning algorithm, which utilizes both PSM and PSEM, was developed in Python using the open-source optimization modeling language Pyomo [20]. The custom implementation of PSM and PSEM allows for flexible adaptation of the methods, including variations in the solution process, such as starting with multiple segments, each with a distinct number of collocation points.

To solve the optimization problem defined within Pyomo, the IPOPT solver [92] is employed. IPOPT is an open-source solver that is particularly effective in solving large-scale nonlinear programming (NLP) problems using the interior-point method. Pyomo processes the problem description and interfaces with IPOPT, allowing for the use of an efficient C++ implementation standard for solving large-scale NLPs.

The dynamic model of the UAV, as described in Chapter 2, is incorporated using the SymPy library, which is designed for symbolic computations. The equations are translated into Python functions, which are crucial for both the problem description in Pyomo and the evaluation of the discretization error.

Incorporating the initial guess is a simple process. The state and control variables in Pyomo are set to these initial values before being passed to IPOPT for optimization. To ensure consistency across various initial guesses, the trajectory planning algorithm is executed on the Czech National Grid Infrastructure MetaCentrum[3]. To maintain comparable computational times, a minimum CPU computational power, based on the SPECfp2017 norm[4], is specified at 8.0. All computations are performed on AMD Epyc processors based on MetaCentrum reports and physical machine specifications. 16 CPUs and 50 GB of RAM are allocated to each job.

[3]MetaCentrum – `https://metavo.metacentrum.cz/en/about/index.html`
[4]SPECfp2017 – `https://www.spec.org/cpu2017/Docs/`

In order to ensure the consistent execution of the trajectory planning algorithm across different machines, a custom Apptainer[5] (formerly Singularity) image with Miniconda is created. The Miniconda environment simplifies the acquisition and deployment of IPOPT.

6.3.4 Results and Discussion

This section presents the results for different scenarios, emphasizing the impact of the initial guess on various criteria. It includes multiple outcomes for different scenarios, followed by a detailed solution of one scenario examined in depth. Environments with 3D obstacles will not be presented as LT* found either a direct path that would not provide enough information compared to a simple initialization or an interesting path that would not be feasible in terms of dynamic constraints and safety constraints on the distance to the obstacle. The multi-segment initial guess was only applicable to PSEM, as PSM is limited to approximating within a single segment. Therefore, information from this type of initial guess could not be utilized. Additionally, it features graphs of a specific trajectory, providing visual insights into the results.

It is important to note that in some situations, the trajectory may encounter an obstacle. This happens because PSM and PSEM are both collocation methods, which means that constraints are only evaluated at the collocation points. This issue was already discussed in Section 6.3.2.1.

The search for trajectories was limited to 10 iterations or 5 hours. However, due to the presence of non-convex obstacles and the nonlinear dynamics of the UAV with a large state space, some trajectories could not be found. In some cases, the solver only found a local minimum that did not satisfy the constraints or the discretization error, resulting in an unsatisfactory solution.

The following sections contain tables with information on the optimization process. The *Init. Level* from Sections 6.2.3.1 and 6.2.3.2 is included, which details the starting point of the optimization. The *Constr.* column indicates whether boundary conditions and specific range limits for the state and control variables were incorporated in the initial guess. *Method* denotes the segmentation approach: single (PSM) or multi-segment (PSEM) methods. The *Iter.* column indicates the number of iterations required by the adaptive algorithm to converge, which is a measure of computational effort and solution refinement.

The *optimality criterion* is numerically approximated by the Cleshaw-Curtis quadrature. *Absolute Error* captures the maximum absolute error (defined in Section 6.1.3) across the entire trajectory. *Sum Viol.* quantifies the total constraint violation, which is computed using Simpson's rule and reflects adherence to state and control limits as well as avoidance of obstacles. *Obstacle Viol.* represents the trajectory's infringement into obstacles, calculated by measuring the overlap with obstacle centers in meters. Last, *Total Time* denotes the computation time in seconds, reflecting the trajectory planning efficiency.

In addition, significant values are color-coded in the table. The best values are marked in green, while the worst values are marked in orange (according to the given criteria, the minimum values are the best). The 90th percentile is marked in light red and the 10th percentile is marked in blue.

6.3.4.1 One Obstacle

In the initial scenario, a single obstacle consisting of four adjacent cells was considered as depicted in Figure 6.1a where the LT* path is illustrated. The path for this case consisted of three

[5]Apptainer – `https://apptainer.org/`

Table 6.3: Evaluation of UAV trajectories found by PSM and PSEM for 1 obstacle with single-segment initialization.

Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
position	yes	PSM	6	4.42e+01	4.16e-03	5.47e-03	3.69e-03	124.17s
velocity	yes	PSEM	6	4.27e+01	8.01e-03	2.47e-02	1.79e-02	696.54s
velocity	yes	PSM	8	4.07e+01	6.63e-03	4.27e-03	2.25e-03	156.33s
orientation	yes	PSEM	3	4.10e+01	1.56e-03	3.20e-11	0.00e+00	296.98s
orientation	yes	PSM	7	4.12e+01	1.00e-02	6.36e-03	2.99e-03	115.08s
angular rate	yes	PSEM	6	4.27e+01	8.01e-03	2.47e-02	1.79e-02	666.52s
angular rate	yes	PSM	8	4.07e+01	6.63e-03	4.27e-03	2.25e-03	205.54s
ang. rate ctrl	yes	PSEM	6	3.97e+01	7.03e-03	2.14e-02	1.45e-02	691.34s
ang. rate ctrl	yes	PSM	8	4.13e+01	7.58e-03	4.54e-03	2.19e-03	155.84s

Table 6.4: Evaluation of UAV trajectories found by PSM and PSEM for 1 obstacle with multi-segment initialization.

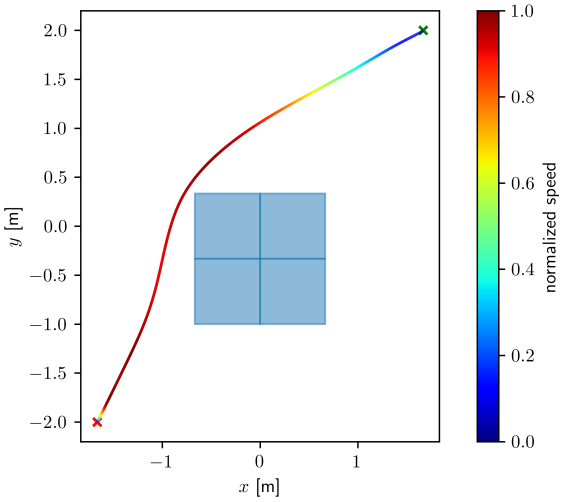
Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
position	yes	PSEM	1	3.78e+01	5.15e-03	1.40e-01	1.22e-01	25.17s
velocity	yes	PSEM	4	4.07e+01	7.29e-03	2.10e-02	1.43e-02	504.42s
angular rate	yes	PSEM	4	4.07e+01	7.29e-03	2.10e-02	1.43e-02	522.78s
ang. rate ctrl	yes	PSEM	3	4.41e+01	7.37e-03	2.44e-02	2.05e-02	351.95s

points including the start and the finish. As depicted in Figure 6.2, the trajectories followed the path. Figure 6.2c shows the passage through the obstacle. The target position is slightly offset from the desired value due to the higher tolerance for absolute error.

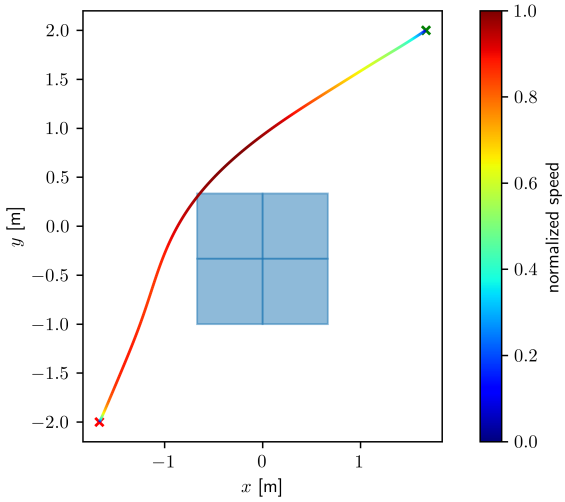
Interestingly, trajectories were not found in the scenario with the simplest initialization that directly links the initial and final states. Tables 6.3 and 6.4 show more detailed data on the planned trajectories for single and multi-segment initialization, respectively. In the case of single segment initial guess (see Table 6.3), the best results were achieved for PSEM with orientation level guess (see Figure 6.2a). The solution was found in 3 iterations, achieving the lowest absolute error without any constraint violations. Collisions were also avoided. For every trajectory, a solution was only found for the initial guess with forced constraints. Additionally, the position level was successful only in the case of PSM.

For multi-segment initial guesses (refer to Table 6.4), the most noteworthy outcomes were obtained for PSEM using a position-level guess, which included both inherently forced boundary and box constraints. The trajectories were found in a single iteration, achieving the lowest absolute error and computation time. However, there were some considerable collisions and slight violations of box constraints. The solution took more than 5 seconds longer without forcing constraints in the initial guess. No solution was found for the initial guess without forced constraints. Taking collisions into consideration, better trajectories were found when using initial guesses for velocity, angular rate, and angular rate with control. No solution was found for the initial orientation guess.

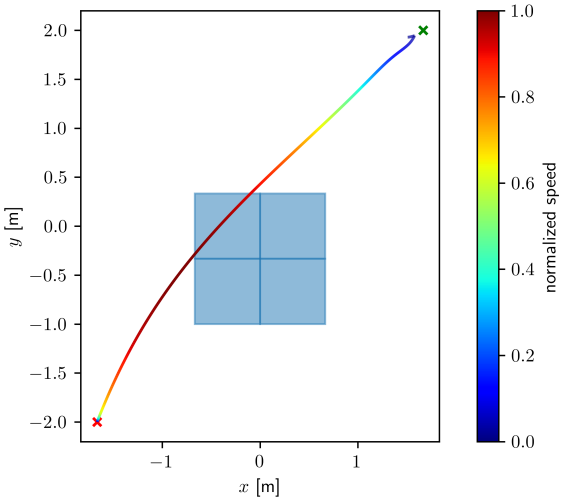
The optimality criterion had similar values in all cases. PSM required 6-8 iterations to find



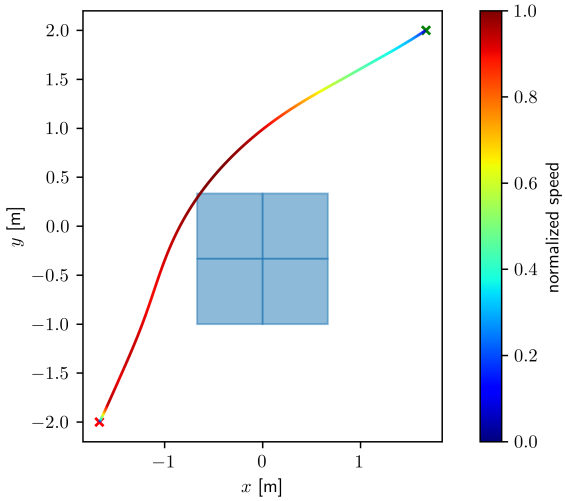
(a) PSEM with single-segment orientation initialization and forced constraints



(b) PSEM with single-segment angular rate initialization and forced constraints



(c) PSEM with multi-segment position initialization and forced constraints



(d) PSEM with multi-segment angular rate with control initialization and forced constraints

Figure 6.2: PSM and PSEM trajectory planning for one obstacle

Table 6.5: Evaluation of UAV trajectories found by PSM and PSEM for 2 obstacles with single-segment initialization.

Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
position	yes	PSEM	3	3.10e+01	7.78e-03	2.41e-02	2.11e-02	167.66s
position	yes	PSM	8	3.17e+01	9.16e-03	5.62e-03	3.30e-03	241.98s
velocity	yes	PSM	6	2.89e+01	9.62e-03	9.03e-03	6.51e-03	105.95s
velocity	yes	PSEM	7	3.00e+01	4.37e-03	1.34e-02	1.17e-02	730.64s
orientation	yes	PSM	8	2.88e+01	3.43e-03	5.74e-03	3.62e-03	163.24s
angular rate	yes	PSM	6	2.89e+01	9.62e-03	9.03e-03	6.51e-03	123.88s
angular rate	yes	PSEM	7	3.00e+01	4.37e-03	1.34e-02	1.17e-02	476.32s
ang. rate ctrl	yes	PSM	5	2.93e+01	8.66e-03	7.91e-03	6.20e-03	97.57s
ang. rate ctrl	yes	PSEM	8	3.03e+01	7.87e-03	4.71e-03	3.37e-03	1005.08s

a solution, while PSEM required 1-4 iterations. The computational time for each solution varied greatly. In most cases, PSM was significantly faster despite the larger number of iterations. This is because it approximated OCP at fewer collocation points. When comparing PSEM in the single and multisegment initial guess cases, planning was faster in the multisegment case.

6.3.4.2 Two Obstacles

In another scenario, the case where a UAV has to fly around two combined obstacles was investigated. Again, no trajectories were found without constraint enforcement. In most cases, the trajectory (see Figure 6.3) followed the path found by LT* shown in Figure 6.1b. Only in Figure 6.3b for the case of PSEM with the single-segment angular rate with control initialization and forced constraints, the path was different, but the results are similar. Interestingly, this trajectory was found in the shortest time. In the simple initialization mode, again the trajectory was not found.

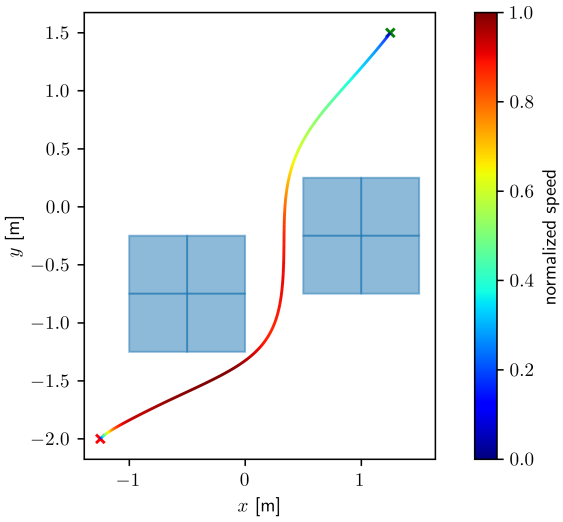
From Table 6.5 for the single-segment initial guess, it can be observed that the position initialization level needed the least number of iterations, but again the largest collisions and collision violations were recorded. It is difficult to select the best trajectory according to the resulting values. Interesting results were obtained, for example, for the trajectory found by the PSM for the orientation initial guess (see Figure 6.3a) with the best-achieved values for optimality criterion and absolute error.

Table 6.5 shows that the position guess was found in one iteration and with the fastest computation time. Constraint adherence in this case was of the order of magnitude comparable.

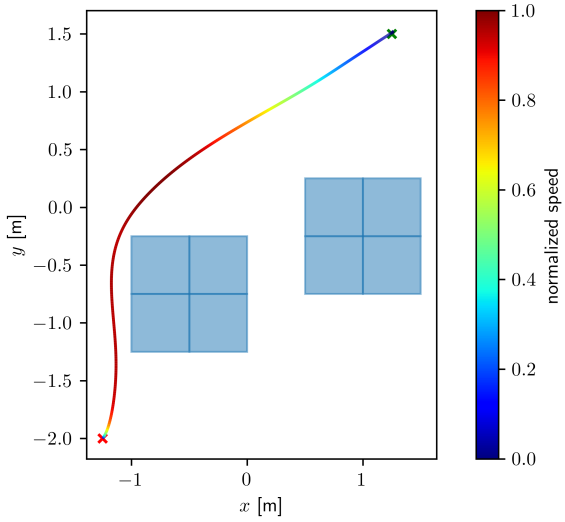
The optimality criterion had similar values throughout. It is noticeable that when using multi-segment initialization, the values were slightly lower. However, in the case of PSEM with multi-segment initialization, computation time significantly increased in three out of five cases compared to single-segment initialization.

6.3.4.3 Random Columns

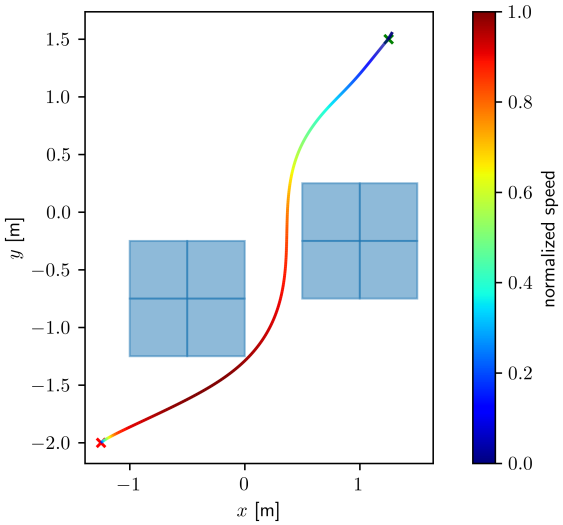
The scenario's environment was randomly generated throughout the flight space with a uniform distribution. It contains 30 obstacles that are incorporated into the OCP as columns. Figure 6.1c



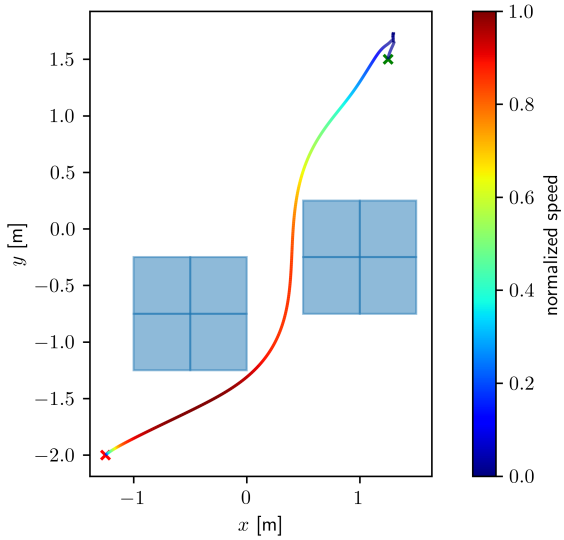
(a) PSM with single-segment orientation initialization and forced constraints



(b) PSEM with single-segment angular rate with control initialization and forced constraints



(c) PSEM with multi-segment orientation initialization and forced constraints

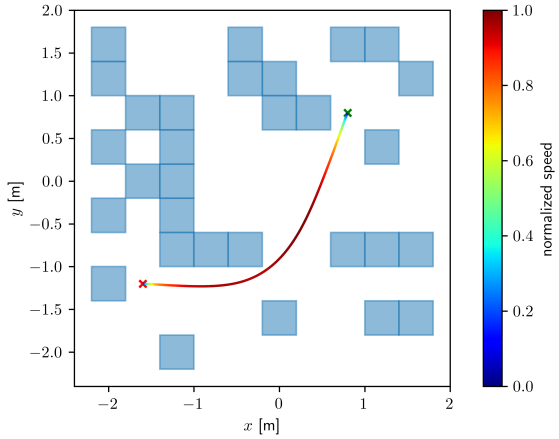


(d) PSEM with multi-segment velocity initialization and forced constraints

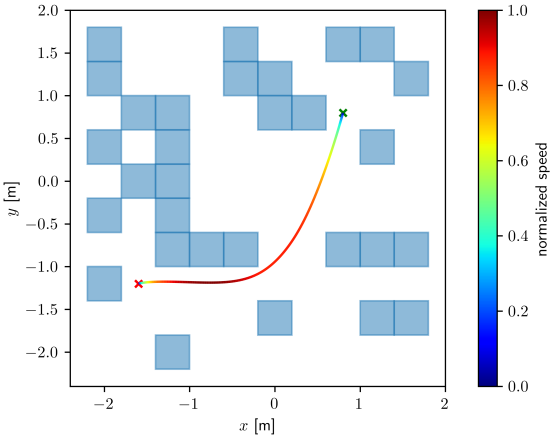
Figure 6.3: PSM and PSEM trajectory planning for two obstacles

Table 6.6: Evaluation of UAV trajectories found by PSM and PSEM for 2 obstacles with multi-segment initialization.

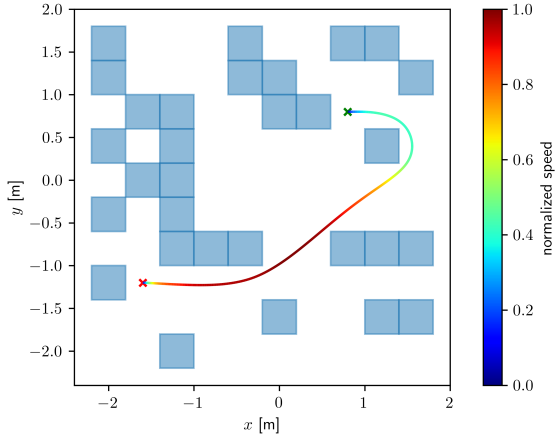
Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
position	yes	PSEM	1	2.72e+01	4.20e-03	2.00e-02	1.99e-02	80.94s
velocity	yes	PSEM	5	2.97e+01	5.09e-03	2.91e-02	2.91e-02	2308.56s
orientation	yes	PSEM	7	2.71e+01	3.75e-03	1.71e-02	1.61e-02	5547.63s
angular rate	yes	PSEM	5	2.97e+01	5.09e-03	2.91e-02	2.91e-02	2003.10s
ang. rate ctrl	yes	PSEM	3	2.82e+01	3.81e-03	1.46e-02	1.37e-02	972.14s



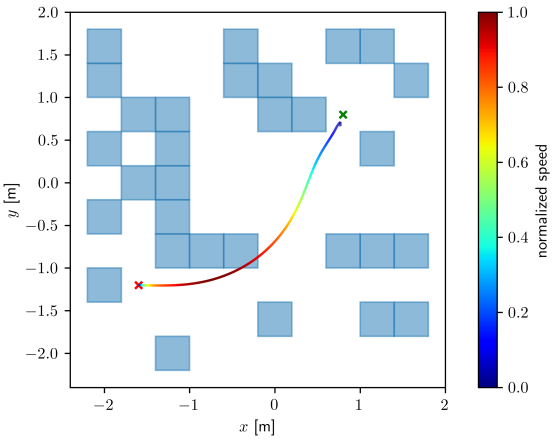
(a) PSEM with single-segment simple initialization



(b) PSEM with single-segment angular rate initialization with control and forced constraints



(c) PSEM with multi-segment simple initialization



(d) PSEM with multi-segment position initialization

Figure 6.4: PSM and PSEM trajectory planning for the environment with randomly generated columns

shows the LT* path through the environment, while Figure 6.4 displays example trajectories. Almost every trajectory follows the path, except for Figure 6.4c. In this scenario, a trajectory was found for a simple initial guess in both multi and single-segment initialization. For initial guesses with velocity and beyond, only trajectories with forced constraints were found.

Table 6.7 shows values for single-segment initialization. The trajectory with the PSEM simple initial guess (see Figure 6.4a) was found in only two iterations. In terms of constraint adherence, PSEM with position initialization recorded the best result, and PSEM with angular rate and control, shown in Figure 6.4b, recorded the best maximum absolute error, also it was found only in three iterations with a low value for optimality criterion.

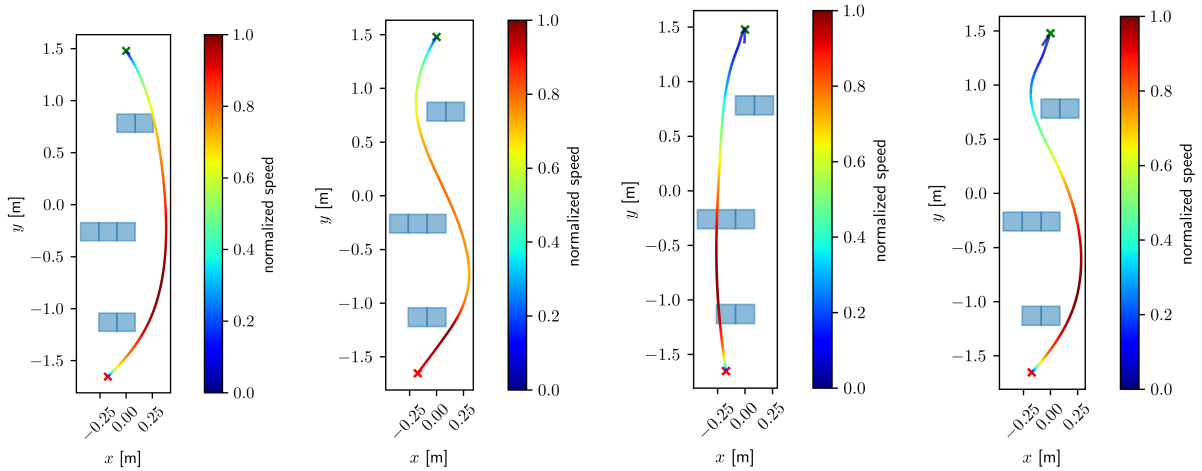
According to Table 6.8 with the multi-segment initialization, the trajectory with a single initialization experienced the lowest constraint violation, but at the cost of the highest number of iterations and computation time. In contrast, the trajectory with position initialization was found in one iteration with the best value of the optimality criterion and the lowest maximum absolute error.

Table 6.7: Evaluation of UAV trajectories found by PSM and PSEM for random columns with single-segment initialization.

Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
none	no	PSEM	2	2.10e+01	5.38e-03	2.83e-03	2.01e-03	150.63s
none	no	PSM	5	1.86e+01	9.62e-03	3.88e-03	2.18e-03	140.36s
position	yes	PSEM	4	1.77e+01	4.38e-03	5.98e-04	3.38e-04	151.39s
position	yes	PSM	5	1.90e+01	8.22e-03	4.17e-03	2.20e-03	154.93s
velocity	yes	PSEM	4	1.72e+01	1.97e-03	1.04e-02	5.71e-03	394.50s
velocity	yes	PSM	5	1.76e+01	7.63e-03	2.94e-03	2.94e-03	101.50s
orientation	yes	PSEM	3	1.60e+01	1.56e-03	6.10e-03	3.49e-03	503.45s
orientation	yes	PSM	4	1.75e+01	3.94e-03	4.11e-03	4.10e-03	76.55s
angular rate	yes	PSEM	4	1.72e+01	1.97e-03	1.04e-02	5.71e-03	325.69s
angular rate	yes	PSM	5	1.76e+01	7.63e-03	2.94e-03	2.94e-03	128.18s
ang. rate ctrl	yes	PSM	3	2.39e+01	8.28e-03	3.73e-03	3.65e-03	46.71s
ang. rate ctrl	yes	PSEM	3	1.70e+01	1.17e-03	2.83e-03	1.04e-03	299.46s

Table 6.8: Evaluation of UAV trajectories found by PSM and PSEM for random columns with multi-segment initialization.

Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
none	no	PSEM	4	2.26e+01	7.44e-03	2.10e-04	1.29e-04	1576.18s
position	yes	PSEM	1	1.49e+01	2.96e-03	3.90e-02	3.89e-02	37.94s
velocity	yes	PSEM	2	2.45e+01	9.37e-03	1.49e-02	9.26e-03	345.94s
orientation	yes	PSEM	3	1.62e+01	6.39e-03	8.81e-04	7.96e-04	311.26s
angular rate	yes	PSEM	2	2.45e+01	9.37e-03	1.49e-02	9.26e-03	338.72s
ang. rate ctrl	yes	PSEM	3	1.76e+01	4.66e-03	1.53e-02	1.01e-02	346.07s



(a) PSEM with single-segment simple initialization

(b) PSEM with single-segment angular rate initialization

(c) PSEM with multi-segment simple initialization

(d) PSEM with multi-segment position initialization

Figure 6.5: PSM and PSEM trajectory planning for the environment with randomly generated columns

The optimality criterion values differed significantly more than in the previous two scenarios. It is unclear whether single or multi-segment initialization would result in significantly better values. Additionally, computation times varied.

6.3.4.4 Walls

This section presents the results of an experiment where three wall-like obstacles were intentionally placed between the start and the goal. The trajectories followed the path shown in Figure 6.1d, as seen in Figure 6.5. However, in cases of simple initialization, the trajectories deviated and went around the obstacles from one side (Figure 6.5a) or even went through them Figure 6.5c. Trajectories were found for all cases in this experiment.

Table 6.9 shows that only two iterations were needed to find many trajectories for single-segment initialization. The trajectory with PSEM velocity initial guess without forced constraints achieved good results. The same values were achieved by the trajectory found by PSEM with the angular rate initial guess without forced constraints (see Figure 6.5b).

According to Table 6.10 with the multi-segment initialization, the trajectory with a single-segment initial guess experienced the largest collisions, as shown in Figure 6.5c. However, the other measures are very good. The position initial guess trajectory (see Figure 6.5d) also reached interesting values but without such collisions. The other trajectories required more than one iteration and also consumed more computational time. The initial guesses for velocity and angular rate, with enforced constraints, reached interesting and equal values. However, only the velocity trajectory was planned faster.

In the comparison of computation times, single-segment initialization was more advantageous in this scenario. The other criteria took different values regardless of the number of segments in the initialization.

Table 6.9: Evaluation of UAV trajectories found by PSM and PSEM for walls with single-segment initialization.

Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
none	no	PSEM	2	1.68e+01	3.29e-03	1.51e-02	1.51e-02	81.96s
none	no	PSM	6	1.53e+01	6.60e-03	1.22e-02	1.22e-02	98.84s
position	yes	PSEM	5	1.61e+01	4.73e-03	4.56e-02	4.21e-02	351.79s
position	yes	PSM	6	1.54e+01	5.87e-03	1.29e-02	1.29e-02	107.82s
velocity	no	PSM	2	2.42e+01	6.29e-03	1.68e-02	1.63e-02	21.26s
velocity	no	PSEM	2	1.99e+01	9.67e-03	3.58e-03	2.04e-03	123.58s
velocity	yes	PSM	3	1.80e+01	2.86e-03	1.72e-02	1.47e-02	28.37s
velocity	yes	PSEM	4	2.23e+01	2.57e-03	4.19e-03	4.04e-03	808.55s
orientation	no	PSM	2	3.42e+01	8.34e-03	1.38e-02	7.78e-03	24.00s
orientation	yes	PSM	3	1.79e+01	9.91e-03	1.64e-02	1.33e-02	33.05s
orientation	no	PSEM	3	1.44e+01	7.85e-03	6.87e-03	6.05e-03	248.25s
orientation	yes	PSEM	5	1.50e+01	1.42e-03	1.08e-02	8.49e-03	407.64s
angular rate	no	PSM	2	2.42e+01	6.29e-03	1.68e-02	1.63e-02	30.20s
angular rate	no	PSEM	2	1.99e+01	9.67e-03	3.58e-03	2.04e-03	88.79s
angular rate	yes	PSM	3	1.80e+01	2.86e-03	1.72e-02	1.47e-02	26.63s
angular rate	yes	PSEM	4	2.23e+01	2.57e-03	4.19e-03	4.04e-03	670.05s
ang. rate ctrl	no	PSEM	2	2.72e+01	8.20e-03	1.68e-02	1.34e-02	21.45s
ang. rate ctrl	no	PSM	2	2.72e+01	8.20e-03	1.68e-02	1.34e-02	22.56s
ang. rate ctrl	yes	PSEM	3	1.95e+01	8.48e-03	3.59e-02	3.56e-02	252.05s
ang. rate ctrl	yes	PSM	5	1.68e+01	5.30e-03	7.58e-03	6.04e-03	94.77s

Table 6.10: Evaluation of UAV trajectories found by PSM and PSEM for walls with multi-segment initialization.

Init. Level	Constr.	Method	Iter.	Optimality Criterion	Absolute Error	Sum Viol.	Obstacle Viol.	Total Time
none	no	PSEM	1	1.32e+01	1.17e-03	4.31e-02	4.15e-02	77.25s
position	yes	PSEM	1	1.48e+01	6.68e-03	9.51e-03	9.51e-03	187.31s
velocity	yes	PSEM	3	2.15e+01	3.02e-03	2.65e-03	2.65e-03	321.88s
velocity	no	PSEM	3	1.38e+01	4.15e-03	1.22e-02	1.19e-02	1663.79s
orientation	yes	PSEM	3	1.71e+01	7.61e-03	8.60e-03	6.43e-03	592.40s
orientation	no	PSEM	3	1.27e+01	9.41e-03	1.71e-02	1.54e-02	657.86s
angular rate	yes	PSEM	3	2.15e+01	3.02e-03	2.65e-03	2.65e-03	525.20s
angular rate	no	PSEM	3	1.38e+01	4.15e-03	1.22e-02	1.19e-02	1133.86s
ang. rate ctrl	yes	PSEM	2	5.62e+01	7.95e-03	5.22e-03	4.52e-03	420.24s
ang. rate ctrl	no	PSEM	2	1.70e+01	7.69e-03	4.03e-03	3.25e-03	474.95s

6.3.4.5 Example of Resulting Trajectory

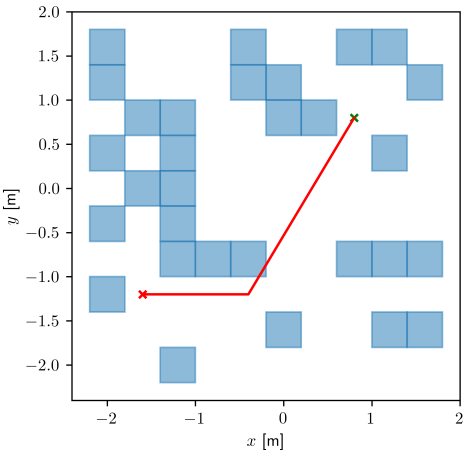
This section presents plots for a trajectory planned using PSEM based on a position single-segment initial guess with forced constraints in a scenario with randomly generated columns. The trajectories for all state and control components are shown, except for the scalar quaternion component, which is omitted for plot readability. Subsequently, the adaptive trajectory planning algorithm presents errors that have been evaluated objectively. The trajectory plots include the collocation points as well as the interleaved polynomial curves. The plots displaying the approximation error are presented as a step function, as the error is calculated as the integral between two collocation points.

Figure 6.6 shows the environment in the $\vec{x}^L$ and $\vec{y}^L$ plane. For completeness, Figures 6.6a and 6.6b are shown again with the LT* path interlaced and with the interlaced trajectory colored according to speed, respectively. Additionally, Figure 6.6c shows the environment on the $\vec{x}^L$ and $\vec{y}^L$ axes. Unlike the previously presented plots, here the collocation points are shown interleaved with polynomials according to the collocation grid. In addition, the figure is supplemented by a representation of obstacles as considered in the OCP. The square indicates the center of the obstacle and the circle indicates the safe space that the trajectory should not enter due to the size of the UAV, the obstacle, and the safe margin.

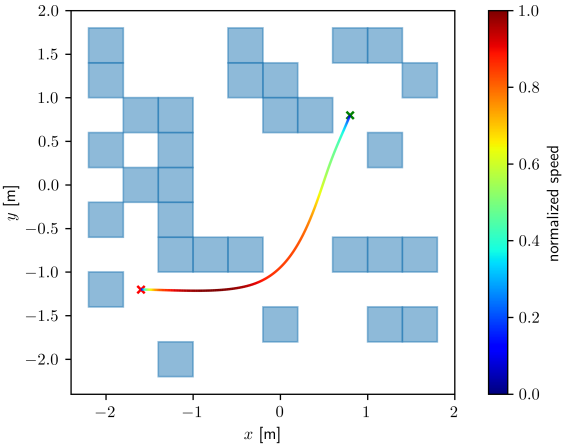
Figure 6.7 shows the different components of the state trajectory as a function of time. The largest density of collocation points is observed at the beginning of the trajectory, where abrupt state changes occur. In particular, the changes in angular velocity (see Figure 6.7d) are due to the UAV's principle of motion.

Figure 6.8 shows the control trajectory over time. For both collective thrust and torque, sharp changes are observed at the beginning of the trajectory, necessary for the UAV to gain sufficient speed toward the target. In Figure 6.8b, the discontinuity of the torque components can be observed just before 0.5 s. In this region, one segment ends and the next segment begins, the interconnection condition of the segment edges says that the values should be equal. There is no condition on the derivative of the control. While the low values of the torque components may not significantly affect the maximum absolute error, imposing a stricter tolerance for absolute error could have yielded a smoother trajectory.

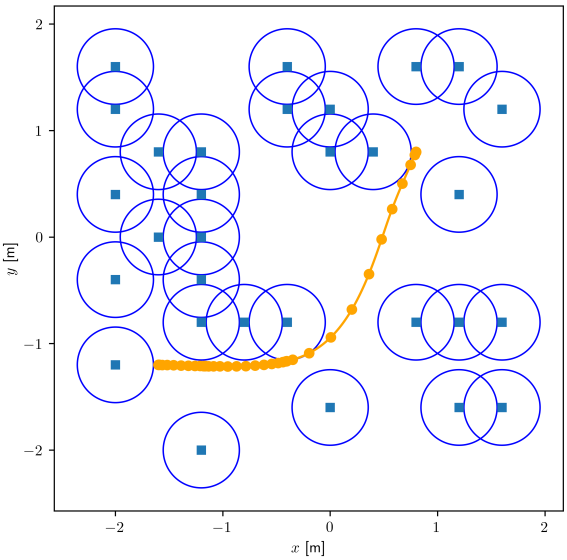
Figure 6.9 shows the errors that are used to evaluate whether and how to adjust the collocation grid before the next iteration. Specifically, Figure 6.9a shows the maximum absolute error, which decreases the edges of each segment in PSEM. This error is used to assess the overall trajectory's accuracy and identify segments that require increased accuracy. Figure 6.9b shows the maximum relative error. If the algorithm determines that the given segment is not accurate enough, it will decide whether to increase the degree of the polynomial or to divide the segment based on this error. Figure 6.9c shows the deflection used to determine the specific points at which the segment will be divided.



(a) LT* path

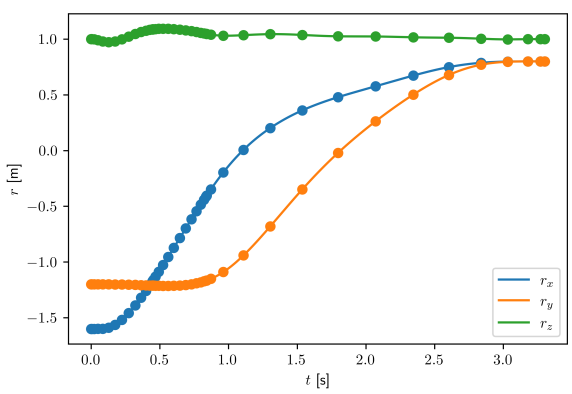


(b) Trajectory

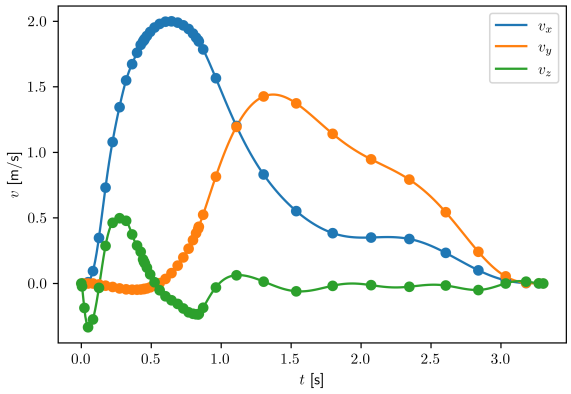


(c) Position with collocation points

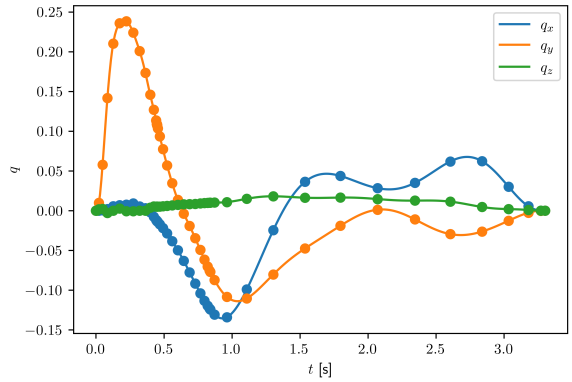
Figure 6.6: Path and trajectory in the environment



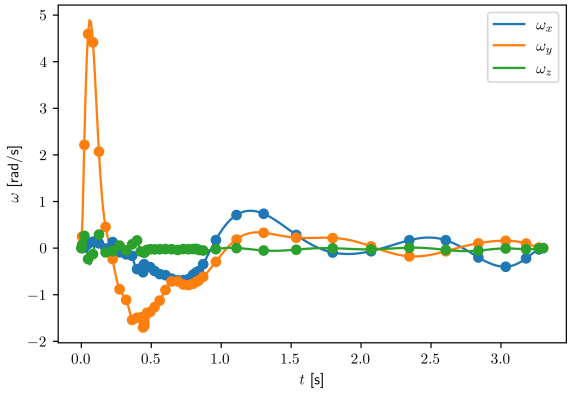
(a) Position



(b) Velocity

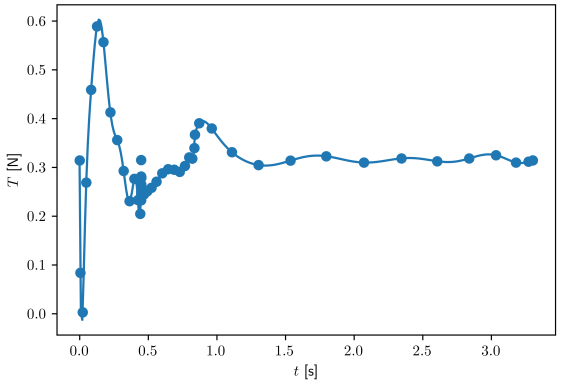


(c) Orientation

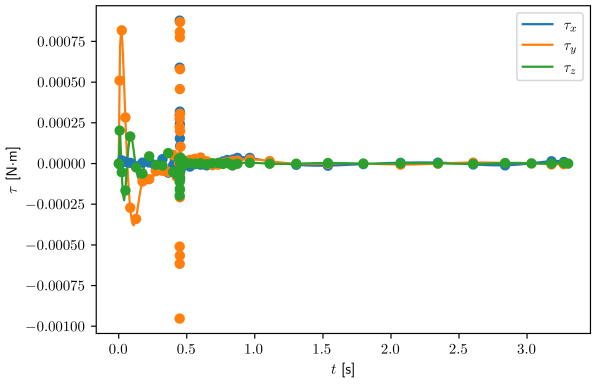


(d) Angular rate

Figure 6.7: Example of resulting state trajectory for randomly generated environment

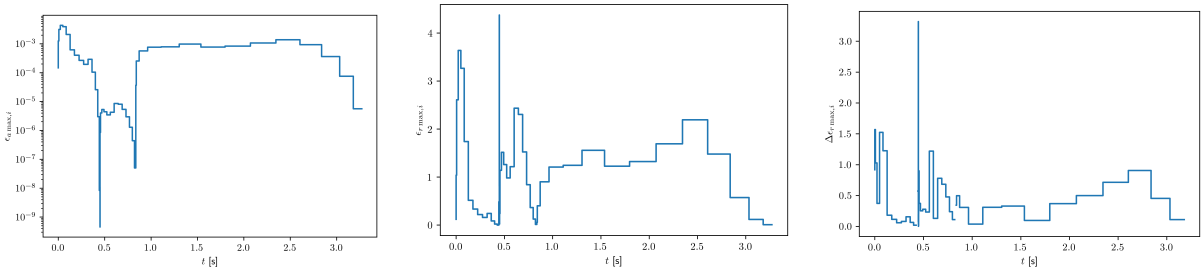


(a) Collective thrust



(b) Torque

Figure 6.8: Example of resulting control trajectory for randomly generated environment



(a) Maximum absolute error (b) Maximum relative error (c) Deflection of relative error

Figure 6.9: Example of errors of trajectory for randomly generated environment

6.4 Conclusion

This chapter describes the Pseudospectral Method (PSM) and the Pseudospectral Element Method (PSEM). It demonstrates how the path found using LT* can be used to make an initial guess of the trajectory. A series of experiments was conducted to test which schemes and levels of initial guess are advantageous in different scenarios.

The results show that PSEM required fewer iterations than PSM, regardless of the initial guess. However, in terms of computational time, PSM was often faster because it uses fewer collocation points and therefore has fewer variables to solve. Nevertheless, in most cases, PSEM produced the best results according to the monitored criteria.

It shall be stressed that for a simple initial guess, where only the Optimal Control Problem (OCP) boundary points are interpolated in time, no trajectories were found for the one and two obstacle scenarios. In the other scenarios, the trajectory was found but often collided with obstacles.

The multi-segment position initial guess with PSEM tends to be the fastest found and yields the best values, indicating that it is advantageous to stick to the path from LT*. However, a larger number of collisions with static obstacles and constraints is the problem. It is difficult to obtain consistent results for other levels of guesses, possibly due to inaccurate guesses of other components of the trajectory. The initial guess may require further fine-tuning. In this form of initial guess, it is recommended to utilize the position level while incorporating the measure of collisions and the state constraints in the stop conditions of the adaptive planning algorithm. This guarantees a collision-free and feasible trajectory.

The previously shown trajectories can serve as a reference for a basic feedback controller to control the UAV. While following the optimal trajectory, such a controller can correct possible deviations in case of UAV malfunction or inaccuracies in the trajectory. Such a trajectory tracking controller will be proposed in Chapter 7. Optionally, a more advanced controller can also provide collision avoidance to prevent collisions resulting from poor constraint evaluation during trajectory planning.

In the future, the work could be extended by conducting a detailed analysis of an adequate number of initial guess collocation points and exploring various ways of segmentation. Additionally, it would also be possible to select only a certain subset of the gridmap based on the path and set the conditions in the OCP solely on this subset. The implementation could certainly be optimized to improve solution time, convergence, and numerical stability.

7 Interpolating Control Based Trajectory Tracking for UAV

Section 4.2 presented Interpolating Control (IC) as an effective methodology for controlling systems given some constraints. The deployment of IC in UAV trajectory tracking, which is one of the main objectives of this work, is not possible in the form presented in the literature. It only allows controlling the system to the origin of the state space. Thus, it is not possible to steer the system to a selected setpoint or to track a reference trajectory. The goal of this chapter is to design an IC-based trajectory tracking method, that can be implemented for the UAV. This goal will be pursued through:

- Designing the IC for steering to an arbitrary setpoint in Section 7.1.
- Generalizing the designed IC for tracking an arbitrary trajectory in Section 7.1.
- Comparing the performance of proposed methods with Model Predictive Control (MPC) in terms of both computation time and control quality in Section 7.2.1.
- Designing a trajectory tracking controller for UAVs and testing its performance in a laboratory setting in Section 7.2.2.

7.1 Interpolating Control with Trajectory Tracking Capability

The IC is only able to stabilize the system to the origin, i.e., to zero; therefore, its extension will be proposed in this section with the ability to steer the system along a given trajectory.

In order to be able to control the system to a setpoint other than the origin, all control laws between which interpolation is performed must also be capable of controlling to an arbitrary setpoint or along a given reference trajectory.

At the beginning of Chapter 4, Linear Quadratic Regulator (LQR) was introduced as a solution to a special case of the unconstrained OCP. Moreover, its version capable of steering to an arbitrary setpoint and also along a given trajectory was described. However, for Vertex Control (VC, introduced also in Section 4.2.2), which is often used in the standard IC as a low gain controller, no algorithm that can control the system at least to an arbitrary setpoint has been described in the literature. An algorithm with this capability will be designed below.

If the vertices of VC control law located at the origin are shifted by a vector given by a setpoint $\boldsymbol{x}_r$, the controller adjusted in this way will steer the system to the setpoint $\boldsymbol{x}_r$. The set $\mathcal{C}^N$ and the admissible control on its vertices remain the same. The control at the coordinates

of the setpoint will be set as $\boldsymbol{u}(\boldsymbol{x}_r) = 0$. Changes in a partition of VC are shown in Figure 7.1. For comparison, Figure 7.1a shows the VC partition for control to the origin and Figures 7.1b–7.1f show the partitions for control to several setpoints. It is possible to directly observe how each region changes and stretches towards the setpoint.

In Section 4.2.2, the VC control law was described for each simplex $\mathcal{C}^{N^{(j)}}$ as

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{K}^{(j)}\boldsymbol{x},\ \boldsymbol{x} \in \mathcal{C}^{N^{(j)}}, \tag{7.1}$$

where $\boldsymbol{K}^{(j)}$ was obtained using

$$\boldsymbol{K}^{(j)} = \boldsymbol{U}^{(j)}\left(\boldsymbol{V}^{(j)}\right)^{-1} \tag{7.2}$$

with the matrix of vertices $\boldsymbol{V}^{(j)} = \left[\boldsymbol{v}_1^{(j)}, \boldsymbol{v}_2^{(j)}, \ldots, \boldsymbol{v}_n^{(j)}\right]$ generating $\mathcal{C}^{N^{(j)}}$ and the matrix of admissible control at the corresponding vertices $\boldsymbol{U}^{(j)} = \left[\boldsymbol{u}_1^{(j)}, \boldsymbol{u}_2^{(j)}, \ldots, \boldsymbol{u}_n^{(j)}\right]$.

Since the origin of the state space was part of each simplex $\mathcal{C}^{N^{(j)}}$, it was replaced in $\boldsymbol{V}^{(j)}$ by the coordinates of the setpoint $\boldsymbol{x}_r$ resulting in the matrix $\boldsymbol{V}_r^{(j)} = \left[\boldsymbol{v}_{1,r}^{(j)}, \boldsymbol{v}_{2,r}^{(j)}, \ldots, \boldsymbol{v}_{n,r}^{(j)}\right]$ adjusted as

$$\begin{aligned} \boldsymbol{v}_{i,r}^{(j)} &= \boldsymbol{v}_i^{(j)} - \boldsymbol{x}_r, \quad \boldsymbol{v}_i^{(j)} \in \boldsymbol{x}^0, i = 1, 2, \ldots, n, \\ \boldsymbol{v}_{i,r}^{(j)} &= \boldsymbol{v}_i^{(j)}, \qquad\quad \boldsymbol{v}_i^{(j)} \notin \boldsymbol{x}^0, i = 1, 2, \ldots, n, \end{aligned}$$

where $\boldsymbol{x}^0$ is the origin of state space.

The calculation of the parameter $\boldsymbol{K}^{(j)}$ needs to be modified so that $\boldsymbol{u}(\boldsymbol{x}_r) = 0$, therefore, it is replaced by a setpoint dependent function

$$\boldsymbol{K}^{(j)}(\boldsymbol{x}_r) = \boldsymbol{U}^{(j)}\left(\boldsymbol{V}_r^{(j)}\right)^{-1} \tag{7.3}$$

and the resulting control law is given as

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{x}_r) = \boldsymbol{K}^{(j)}(\boldsymbol{x}_r)\,\boldsymbol{x},\ \boldsymbol{x} \in \mathcal{C}^{N^{(j)}},\ \boldsymbol{x}_r \in \mathcal{C}^N. \tag{7.4}$$

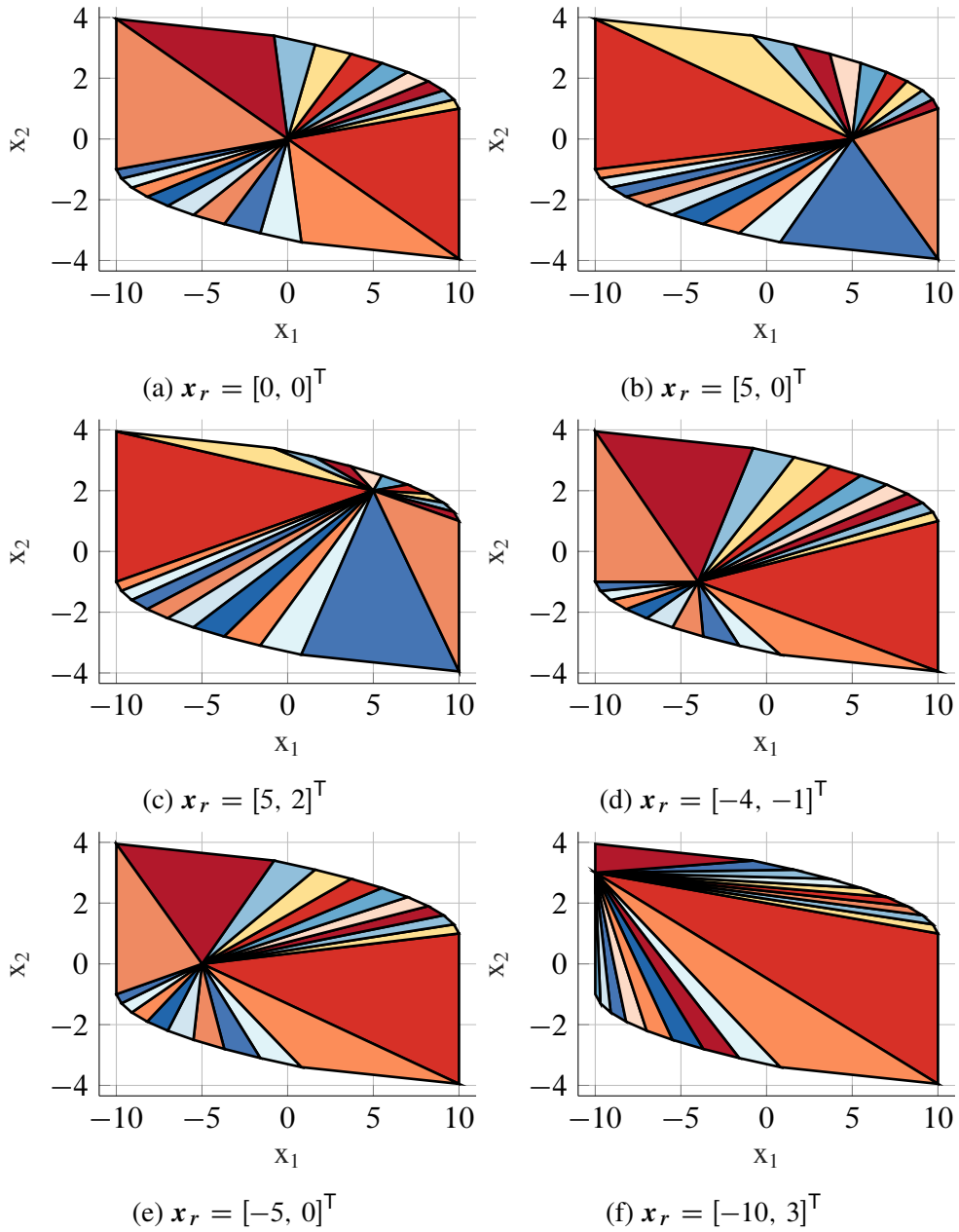
The resulting VC control laws are shown in Figure 7.2, where the original control law is presented in Figure 7.2a and the setpoint control laws are shown in Figures 7.2b–7.2f. This modified VC is fully capable of driving the system to a setpoint $\boldsymbol{x}_r$ within $\mathcal{C}^N$.

Unfortunately, unlike the LQR, no possibility was found to adapt the VC easily to reflect the whole reference trajectory within the constraints. For this reason, VC will only be used in cases of system stabilization and control to the setpoint. In case of trajectory tracking, another LQR will be used in this work in the role of a low-gain controller $\boldsymbol{u}^l$, with weights chosen to cover as much state-space $\mathcal{X}$ as possible.

7.1.1 Implicit Interpolating Control

Now that both LQR and VC are capable of trajectory tracking or setpoint control, the extension of the IC itself will be described. First, to reflect the setpoint in the IC, the original control law from Equation (4.44) must be adjusted. The control law emerged from the closed-loop system state decomposition

$$\boldsymbol{x} = c\boldsymbol{x}^l + (1 - c)\,\boldsymbol{x}^h, \tag{7.5}$$

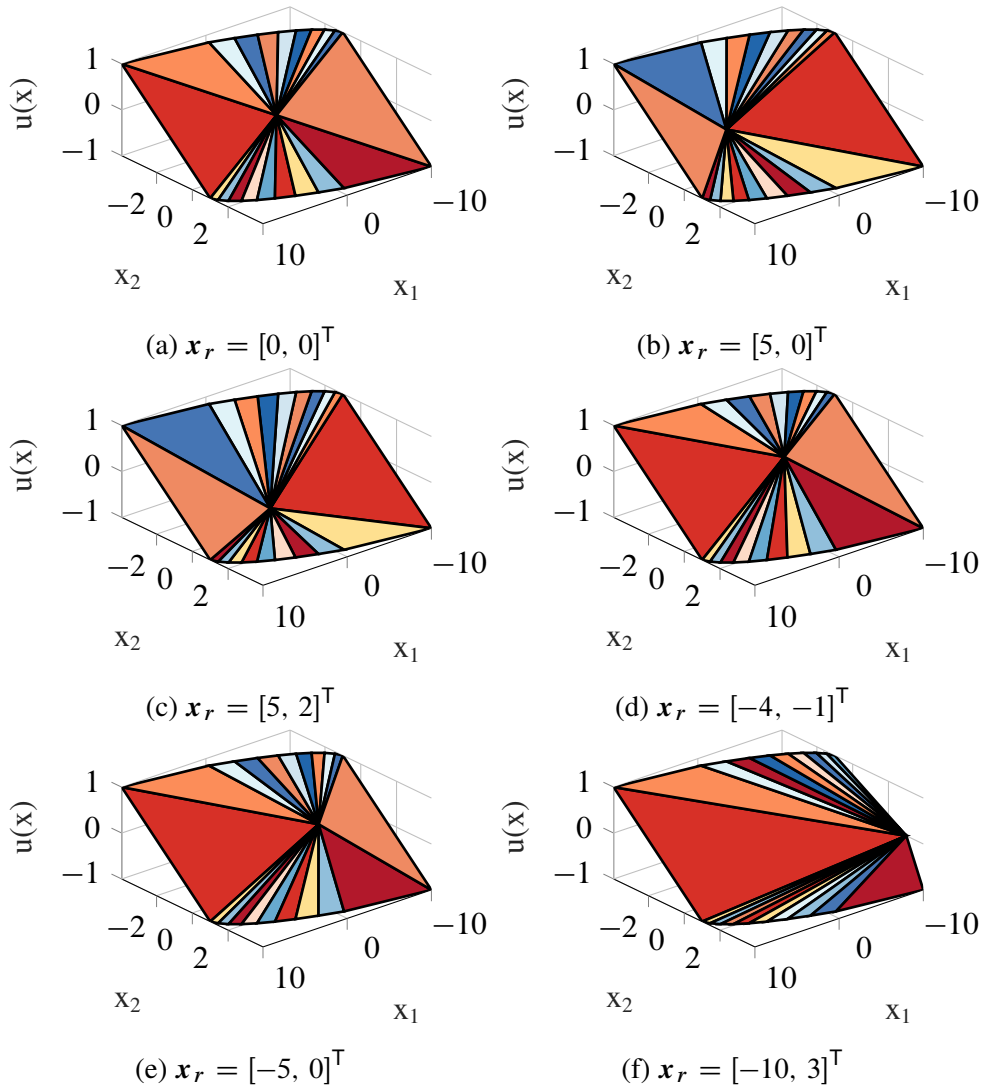


(a) $\boldsymbol{x}_r = [0,\ 0]^\mathsf{T}$ (b) $\boldsymbol{x}_r = [5,\ 0]^\mathsf{T}$

(c) $\boldsymbol{x}_r = [5,\ 2]^\mathsf{T}$ (d) $\boldsymbol{x}_r = [-4,\ -1]^\mathsf{T}$

(e) $\boldsymbol{x}_r = [-5,\ 0]^\mathsf{T}$ (f) $\boldsymbol{x}_r = [-10,\ 3]^\mathsf{T}$

Figure 7.1: Partitions of Vertex Control state space for several setpoints



(a) $\boldsymbol{x}_r = [0,\ 0]^\mathsf{T}$ (b) $\boldsymbol{x}_r = [5,\ 0]^\mathsf{T}$

(c) $\boldsymbol{x}_r = [5,\ 2]^\mathsf{T}$ (d) $\boldsymbol{x}_r = [-4,\ -1]^\mathsf{T}$

(e) $\boldsymbol{x}_r = [-5,\ 0]^\mathsf{T}$ (f) $\boldsymbol{x}_r = [-10,\ 3]^\mathsf{T}$

Figure 7.2: Explicit control law of Vertex Control state space for several setpoints

where c, $\boldsymbol{x}$, $\boldsymbol{x}^l$, and $\boldsymbol{x}^h$ are the interpolating coefficient, interpolated state vector, and state vectors for a low and high-gain controller, respectively. The original control law was in the form

$$\boldsymbol{u}(\boldsymbol{x}) = c\boldsymbol{u}^l\left(\boldsymbol{x}^l\right) + (1-c)\,\boldsymbol{u}^h\left(\boldsymbol{x}^h\right) \tag{7.6}$$

with low-gain controller $\boldsymbol{u}^l$ and high-gain controller $\boldsymbol{u}^h$. In Section 4.2.3, it was stated that the high-gain controller $\boldsymbol{u}^h$ is designed to achieve faster system response at the cost of a smaller region of state space in which the controller satisfies the constraints. In contrast, the low-gain controller $\boldsymbol{u}^l$ is applicable in a larger region of the state space.

Given that both controllers between which the interpolation is performed allow setpoint control, the resulting control law for setpoint control has the form

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{x}_r) = c\boldsymbol{u}^l\left(\boldsymbol{x}^l, c\boldsymbol{x}_r\right) + (1-c)\,\boldsymbol{u}^h\left(\boldsymbol{x}^h, (1-c)\,\boldsymbol{x}_r\right), \tag{7.7}$$

where $\boldsymbol{u}^l$ and $\boldsymbol{u}^h$ are the low and high-gain controllers for control to the constant setpoint, respectively. In Equation (7.7), the setpoint is scaled for $\boldsymbol{u}^l$ and $\boldsymbol{u}^h$ according to the interpolating coefficient in order to generate the corresponding control action.

Now, the remaining task is to obtain the optimal interpolating coefficient c^*. The original non-linear program without using auxiliary variables was of the form

$$\begin{aligned}
J(\boldsymbol{x}^l, c) &= c, && (7.8)\\
\text{s.t.}\quad \boldsymbol{F}^l\boldsymbol{x}^l &\leq \boldsymbol{g}^l, && (7.9)\\
\boldsymbol{F}^h\boldsymbol{x}^h &\leq \boldsymbol{g}^h, && (7.10)\\
c\boldsymbol{x}^l + (1-c)\,\boldsymbol{x}^h &= \boldsymbol{x}, && (7.11)\\
0 \leq c &\leq 1, && (7.12)
\end{aligned}$$

where $\boldsymbol{x}$ represents the current state for which the control is sought. As the value of the state $\boldsymbol{x}$ changes, the interpolation coefficient c may not be optimal for the new state, and the constraints could be violated, therefore the NLP needs to be solved again.

The IC can be adjusted to setpoint control similarly to the VC. For the VC, the central vertex was shifted from the origin to the setpoint $\boldsymbol{x}_r$ coordinates. For the IC, the positively invariant set of the high-gain controller $\boldsymbol{\Omega}^h$ is located at the origin. By shifting the center of the set from the origin coordinates to the setpoint coordinates, an LP is obtained that provides an interpolation coefficient c that is optimal for control to the setpoint $\boldsymbol{x}_r$.

The invariant set $\boldsymbol{\Omega}^h$ is represented in inequality (7.10) in the original NLP. Therefore, the IC for the setpoint $\boldsymbol{x}_r$ is obtained by modifying this inequality. The resulting NLP has the form

$$\begin{aligned}
J(\boldsymbol{x}^l, c) &= c, && (7.13)\\
\text{s.t.}\quad \boldsymbol{F}^l\boldsymbol{x}^l &\leq \boldsymbol{g}^l, && (7.14)\\
\boldsymbol{F}^h\left(\boldsymbol{x}^h - \boldsymbol{x}_r\right) &\leq \boldsymbol{g}^h, && (7.15)\\
c\boldsymbol{x}^l + (1-c)\,\boldsymbol{x}^h &= \boldsymbol{x}, && (7.16)\\
0 \leq c &\leq 1, && (7.17)
\end{aligned}$$

where $\boldsymbol{x}_r = \boldsymbol{x}_{r,k}$, $\boldsymbol{x} = \boldsymbol{x}_k$. Again, if the state $\boldsymbol{x}$ value changes or the setpoint $\boldsymbol{x}_r$ value changes, the NLP must be solved again because c would not be optimal for the new values and the constraints could be violated. Analogous to the original IC, using an auxiliary variable $\boldsymbol{r}^l = c \cdot \boldsymbol{x}^l$ and substituting equation (7.16) into inequality (7.15) the problem becomes linear

$$J(\boldsymbol{r}^l, c) = c, \tag{7.18}$$

$$\text{s.t.} \quad \boldsymbol{F}^l \boldsymbol{r}^l \leq c\boldsymbol{g}^l, \tag{7.19}$$

$$\boldsymbol{F}^h \left(\boldsymbol{x} - \boldsymbol{r}^l\right) \leq (1-c)\left(\boldsymbol{g}^h + \boldsymbol{F}^h \boldsymbol{x}_r\right), \tag{7.20}$$

$$0 \leq c \leq 1, \tag{7.21}$$

where c^* is found by minimizing the criterion $J(\boldsymbol{x}^l, c)$ or $J(\boldsymbol{r}^l, c)$. In Equation (4.52), it can be seen that $\boldsymbol{\Omega}^h$ is shifted by the coordinates of $\boldsymbol{x}_r$.

The extended version of Interpolating Control (eIC) with an additional set $\boldsymbol{\Omega}^m$, presented in Section 4.2.4, can also be adjusted for the setpoint control. The set was described as

$$\boldsymbol{\Omega}^m = \{\boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}^m \boldsymbol{x} \leq \boldsymbol{g}^m\}, \tag{7.22}$$

$$\boldsymbol{\Omega}^h \subset \boldsymbol{\Omega}^m \subset \boldsymbol{\Omega}^l. \tag{7.23}$$

The control law depends on where in the state space the current state of the system is. If $\boldsymbol{x} \in \boldsymbol{\Omega}^l \setminus \boldsymbol{\Omega}^m$, the interpolation is done between $\boldsymbol{u}^l$ and $\boldsymbol{u}^m$ and the IC control law is in form

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{x}_r) = c\boldsymbol{u}^l\left(\boldsymbol{x}^l, c\boldsymbol{x}_r\right) + (1-c)\,\boldsymbol{u}^m\left(\boldsymbol{x}^m, (1-c)\,\boldsymbol{x}_r\right),\ \boldsymbol{x} \in \boldsymbol{\Omega}^l \setminus \boldsymbol{\Omega}^m. \tag{7.24}$$

If the $\boldsymbol{x} \in \boldsymbol{\Omega}^m$, the interpolation is performed between $\boldsymbol{u}^m$ and $\boldsymbol{u}^h$, which results in control law

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{x}_r) = c\boldsymbol{u}^m\left(\boldsymbol{x}^m, c\boldsymbol{x}_r\right) + (1-c)\,\boldsymbol{u}^h\left(\boldsymbol{x}^h, (1-c)\,\boldsymbol{x}_r\right),\ \boldsymbol{x} \in \boldsymbol{\Omega}^m. \tag{7.25}$$

The additional invariant set $\boldsymbol{\Omega}^m$ can be shifted to the state-space coordinates of $\boldsymbol{x}_r$ together with the invariant set $\boldsymbol{\Omega}^h$. Using the LQR as a medium-gain controller $\boldsymbol{u}^m$ is preferable to VC because there is no need to recalculate its parameters while directly reflecting the $\boldsymbol{x}_r$ setpoint.

Further, the IC for trajectory tracking will be explored. As mentioned, no possibility was found to adapt the VC easily to reflect the whole reference trajectory within the constraints. It is possible to interpolate between VC for setpoint control and trajectory-tracking LQR; however, to obtain the best control quality, another LQR will be used as a low-gain controller $\boldsymbol{u}^l$. The disadvantage is that LQR may not include the same positively invariant set volume while following the constraints as VC. This holds even with careful tuning of its weight matrices, i.e. $\boldsymbol{\Omega}^l \subseteq \mathcal{C}^N$ in case that $\mathcal{C}^N$ is maximal N-step controlled positively invariant set (if $\mathcal{C}^N = \mathcal{C}^{N+1}$).

In order to reflect the entire reference trajectory not only in terms of the individual control laws, i.e. $\boldsymbol{u}^h$, $\boldsymbol{u}^l$ and possibly $\boldsymbol{u}^m$, but also in terms of the interpolating coefficient c, it would be necessary to search for an interpolating coefficient c for each reference point of the trajectory $\boldsymbol{x}_{r,k}^{k+N}$ and then to design a procedure that would fuse the knowledge of all the coefficients to obtain one particular time-varying interpolating coefficient c. When the state $\boldsymbol{x}_k$ of the controlled system changes, the knowledge of the individual coefficients c would no longer be relevant and the whole problem would have to be solved again. The problem would be solved N times, where N represents the length of the reference trajectory. For these reasons, the entire trajectory

will only be reflected in the individual control laws between which the interpolation is performed. The problem of finding the interpolating coefficient will always reflect only the first current point of the reference trajectory, i.e. $\boldsymbol{x}_r = \boldsymbol{x}_{r,k}$. As a result, the problem can be described by the same NLP (7.13)–(7.17) and LP (7.18)–(7.21) as for control to a setpoint.

For the successful tracking respecting the constraints, the whole reference trajectory must be located in the invariant set $\boldsymbol{\Omega}^l$ including its boundary, $\boldsymbol{x}_{r,k}^{k+N} \in \boldsymbol{\Omega}^l$. The control law differs from setpoint control in that the reference is not the setpoint but the entire trajectory

$$\boldsymbol{u}(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}^{k+N}) = c_k \boldsymbol{u}^l\left(\boldsymbol{x}_k^l, c_k \boldsymbol{x}_{r,k}^{k+N}\right) + (1 - c_k)\, \boldsymbol{u}^h\left(\boldsymbol{x}_k^h, (1 - c_k)\, \boldsymbol{x}_{r,k}^{k+N}\right), \tag{7.26}$$

where the notation is extended to indicate the time instant k for clarity.

Modifying the eIC as was done for the setpoint control is problematic because the reference trajectory would have to be adjusted to keep constraint $\boldsymbol{x}_{r,k}^{k+N} \in \boldsymbol{\Omega}^m$ for the case $\boldsymbol{x}_k \in \boldsymbol{\Omega}^m$. This adjustment is non-trivial and it results in a significant increase of computational time. Therefore, this kind of eIC will not be employed for trajectory tracking within this thesis.

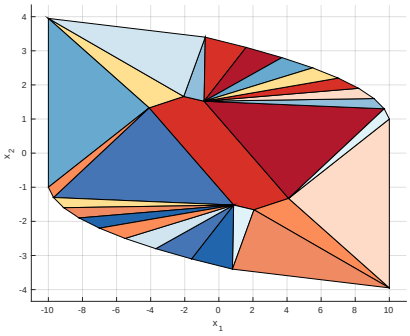
7.1.2 Explicit Interpolating Control

Similar to the MPC, as mentioned in Section 4.2.5, a solution in explicit form can be obtained for the IC. Further, obtaining an explicit solution for setpoint control and trajectory tracking will now be discussed.

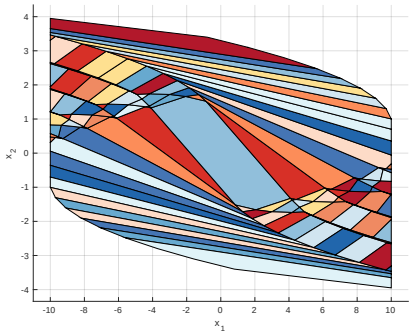
In the control to the setpoint case, as described in (7.18)–(7.21), there are two parametric variables in the multi-parametric solution. The first is the current state vector $\boldsymbol{x}$ and the second is the setpoint $\boldsymbol{x}_r$. The problem arises with the employment of $\boldsymbol{x}_r$ due to the multiplication $(1 - c)\boldsymbol{x}_r$ in inequality (4.52), the multi-parametric program becomes nonlinear. The program could be transformed into the multi-parametric LP using an auxiliary variable $\boldsymbol{x}^r = (1 - c)\boldsymbol{x}_r$, however, there is a problem with the transformation of the solution back to the coordinates of $\boldsymbol{x}_r$. If $c = 1$, the information about $\boldsymbol{x}_r$ cannot be obtained.

Instead of searching for the multi-parametric NLP solution, a procedure similar to the VC for a constant setpoint can be used, which involves modifying the standard explicit IC. As mentioned in Section 7.1.1, steering the system to a setpoint can be conducted in the IC by moving the high-gain controller set $\boldsymbol{\Omega}^h$ to the setpoint $\boldsymbol{x}_r$ coordinates. An explicit solution can be obtained in the case of the fixed setpoint. Figures 7.3 and 7.4 show several partitions for the IC and the MPC. It can be observed that the area shifts in the x_1-axis to the setpoint coordinate, however, the MPC does not shift in the x_2-axis. This effect is due to the absence of an equilibrium point of the system in the example other than $x_2 = 0$. This phenomenon could be detected, for example, by the LQR $\boldsymbol{u}^h$ and its parameter $\bar{\boldsymbol{L}}$, where for the second state the compensation scaling for the setpoint is zero. However, the figures show that the standard explicit IC solution for steering to the origin (see Figure 7.3a) can be utilized and customized for steering to a given setpoint $\boldsymbol{x}_r$. The explicit solution for eIC can be modified similarly to the IC by not shifting only the set $\boldsymbol{\Omega}^h$, but both $\boldsymbol{\Omega}^h$ and $\boldsymbol{\Omega}^m$ at the same time.

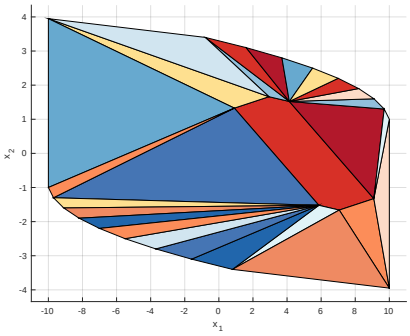
If $\boldsymbol{\Omega}^h$ or $\boldsymbol{\Omega}^m$ share a part of boundaries with the invariant set $\boldsymbol{\Omega}^l$, a simple transformation cannot be executed because new simplices are formed that were not present in the original explicit solution. This problem can be solved by transforming the partition in the positive direction, i.e. $\boldsymbol{x}_r \geq 0$, in all coordinates at once and solving the LP for the missing simplices. Subsequently, the same procedure is repeated for the negative direction, i.e. $\boldsymbol{x}_r \leq 0$. Thanks to this simple procedure, all simplices are discovered and filled in that would otherwise remain hidden.



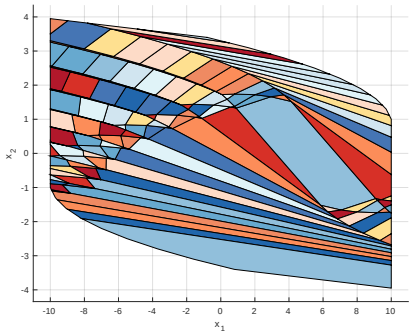
(a) IC, $\boldsymbol{x_r} = [0,\ 0]^\mathsf{T}$



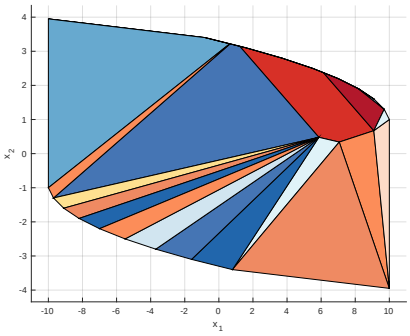
(b) MPC, $\boldsymbol{x_r} = [0,\ 0]^\mathsf{T}$



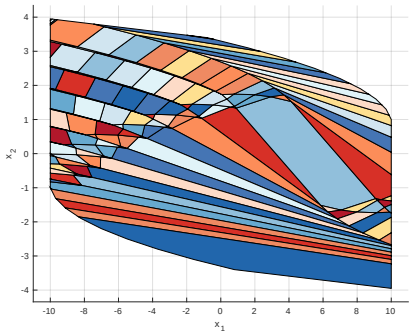
(c) IC, $\boldsymbol{x_r} = [5,\ 0]^\mathsf{T}$



(d) MPC, $\boldsymbol{x_r} = [5,\ 0]^\mathsf{T}$

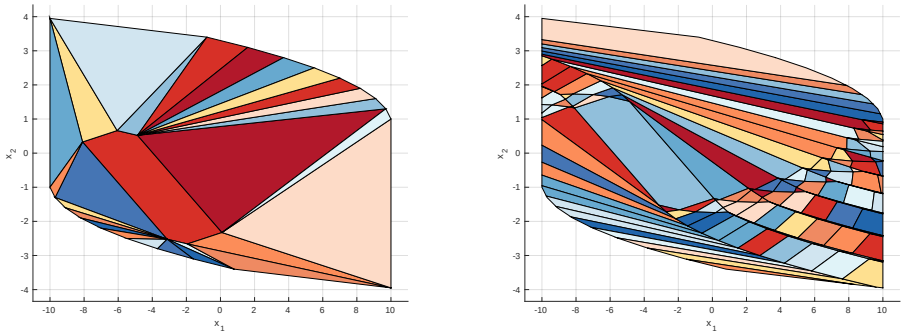


(e) IC, $\boldsymbol{x_r} = [5,\ 2]^\mathsf{T}$



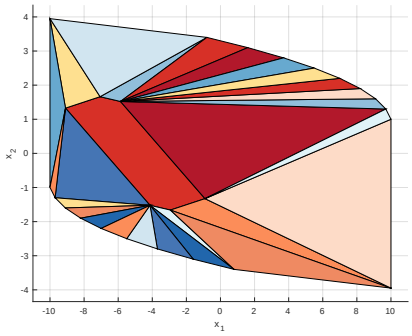
(f) MPC, $\boldsymbol{x_r} = [5,\ 2]^\mathsf{T}$

Figure 7.3: Partitions of Interpolating Control and Model Predictive Control state space for several setpoints Part I

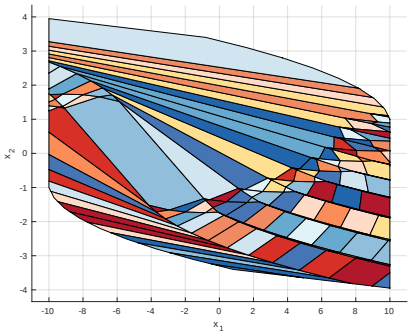


(a) IC, $\boldsymbol{x_r} = [-4,\ -1]^\mathsf{T}$

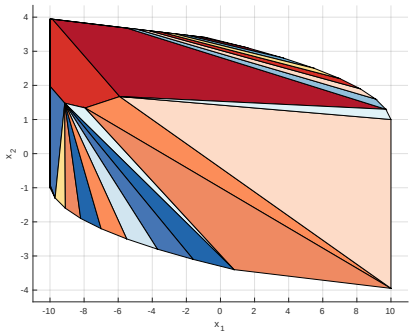
(b) MPC, $\boldsymbol{x_r} = [-4,\ -1]^\mathsf{T}$



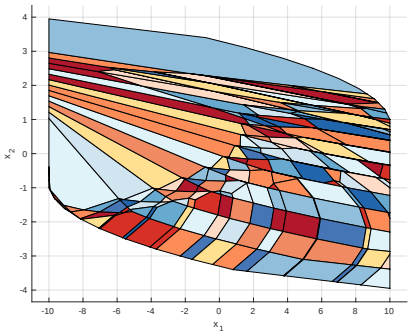
(c) IC, $\boldsymbol{x_r} = [-5,\ 0]^\mathsf{T}$



(d) MPC, $\boldsymbol{x_r} = [-5,\ 0]^\mathsf{T}$



(e) IC, $\boldsymbol{x_r} = [-10,\ 3]^\mathsf{T}$



(f) MPC, $\boldsymbol{x_r} = [-10,\ 3]^\mathsf{T}$

Figure 7.4: Partitions of Interpolating Control and Model Predictive Control state space for several setpoints Part II

Finally, the explicit control law is obtained by constructing a piecewise affine function

$$\boldsymbol{u}_{k,i}(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}) = \boldsymbol{M}_i \boldsymbol{x}_k + \boldsymbol{n}_i, \tag{7.27}$$

where i is the i-th simplex, k is current time instant and $\boldsymbol{M}_i \in \Re^{m\times n}$ and $\boldsymbol{n}_i \in \Re^m$ are defined as

$$\begin{bmatrix}\boldsymbol{M}_i & \boldsymbol{n}_i\end{bmatrix} = \begin{bmatrix}u_1^{(i)} & u_2^{(i)} & \dots & u_{n+1}^{(i)}\end{bmatrix}\begin{bmatrix}V_1^{(i)} & V_2^{(i)} & \dots & V_{n+1}^{(i)} \\ 1 & 1 & \dots & 1\end{bmatrix}, \tag{7.28}$$

where $V_j^{(i)}$ is the $j-th$ vertex of the i-th simplex. The control value is given by the vertex. Similar to VC, the values at the vertices of a multi-parametric solution do not change, only the control function must be adjusted because of the vertices that have shifted. If $V_j^{(i)} \in \boldsymbol{\Omega}^h$, which is the set shifted by setpoint $\boldsymbol{x}_{r,k}$, the control value is given by $\boldsymbol{u}^h(\boldsymbol{x}_k, \boldsymbol{x}_{r,k})$, if it is on the boundary of $\boldsymbol{\Omega}^l$ (i.e. $\boldsymbol{F}^l\boldsymbol{x}_k = \boldsymbol{g}^l$), it is given by $\boldsymbol{u}^l(\boldsymbol{x}_k, \boldsymbol{x}_{r,k})$. Moreover, the partition is only shifted when the setpoint $\boldsymbol{x}_{r,k}$ changes, thus the recalculation does not have to be done each time. The evaluation can be further accelerated by finding only the simplex that is relevant to the current state $\boldsymbol{x}_k$ and setpoint $\boldsymbol{x}_{r,k}$ and constructing only the local control law $\boldsymbol{u}_{k,i}$ for that simplex.

Since in the IC based trajectory tracking the same NLP/LP is solved as in the setpoint control, the modifications of the explicit solution are also applicable to trajectory tracking. When tracking trajectories, the individual controllers $\boldsymbol{u}^l$ and $\boldsymbol{u}^h$ reflect the entire trajectory $\boldsymbol{x}_{r,k}^{k+N}$. Therefore, the explicit solution cannot be used to obtain the control value directly. Instead of constructing the explicit control function, an explicit function is constructed for the interpolation coefficient c, which depends on the current state $\boldsymbol{x}_k$ and the current reference trajectory point $\boldsymbol{x}_{r,k} = \boldsymbol{x}_{r,k}^k$, as

$$c_{k,i}(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}) = \boldsymbol{G}_i \boldsymbol{x}_k + \boldsymbol{h}_i, \tag{7.29}$$

where i is i-th simplex, k is current time instant and $\boldsymbol{G}_i \in \Re^{m\times n}$ and $\boldsymbol{h}_i \in \Re^m$ are defined as

$$\begin{bmatrix}\boldsymbol{G}_i & \boldsymbol{h}_i\end{bmatrix} = \begin{bmatrix}c_1^{(i)} & c_2^{(i)} & \dots & c_{n+1}^{(i)}\end{bmatrix}\begin{bmatrix}V_1^{(i)} & V_2^{(i)} & \dots & V_{n+1}^{(i)} \\ 1 & 1 & \dots & 1\end{bmatrix}. \tag{7.30}$$

If $V_j^{(i)} \in \boldsymbol{\Omega}^h$, $c = 0$, otherwise $c = 1$. Subsequently, the individual control laws can be calculated and interpolated into the IC control law as

$$\boldsymbol{u}(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}^{k+N}) = c_k \boldsymbol{u}^l\left(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}^{k+N}\right) + (1 - c_k)\, \boldsymbol{u}^h\left(\boldsymbol{x}_k, \boldsymbol{x}_{r,k}^{k+N}\right). \tag{7.31}$$

However, if the value $\boldsymbol{x}_{r,k}$ in the reference trajectory changes over time, the explicit interpolating coefficient function needs to be adjusted. Therefore, it may be more convenient to solve the problem implicitly, as described in Section 7.1.1.

7.1.3 Conclusion of Interpolating Control Modifications

Section 7.1 discussed IC-based trajectory tracking. Firstly, the setpoint control for Vertex Control, which is often used as a low-gain controller in IC, was presented. Then, in Section 7.1.1 the IC modification that allows steering the system to an arbitrary setpoint was designed. This modification is necessary to enable the UAV to change its position according to the position setpoint.

Quadrotor UAVs are agile aerial robots. However, when they fly rapid trajectories, it can be invaluably beneficial to consider not only the current setpoint but also the future reference trajectory due to the UAV's inertia. Therefore, Section 7.1.1 discussed a method that allows IC-based tracking for an arbitrary trajectory. This is enabled by using two LQRs with trajectory tracking capability. The reference was weighted for each controller based on an interpolating coefficient that was calculated for the current point in the reference trajectory.

Finally, Section 7.1.2 presented the explicit solution to IC for setpoint control. The control law was geometrically modified to follow the changes in the setpoint. This can serve as an interesting alternative for platforms where it is not possible or beneficial to solve the linear program of IC online.

7.2 Numerical Illustrations

In this section, a comparison of IC and MPC, will be made. Since, the original problem (4.1)-(4.4) was described as an Optimal Control Problem (OCP), for which both MPC and IC are suboptimal solutions, the comparison of both control methods will be performed with regard to OCP criterion (4.1). Computational demands, which are essential for onboard implementation in the UAV control, will serve as an additional indicator.

To develop intuition, numerical experiments will be first performed to control a simple 2nd order LTI system. This simplified dynamic system will allow for clear insights and straightforward visualization of the control performance. The experiments will evaluate three control tasks: stabilization, steering to an arbitrary point, and trajectory tracking. Emphasis on the second-order case will illustrate the fundamental properties and differences between MPC and IC before extending to the higher-order UAV dynamics.

Subsequently, the ability of the IC to control the UAV along the reference trajectory will then be verified and compared with the MPC. A comparison will be also presented in Section 7.2.2, where the IC will be compared with the MPC, which has a limited prediction horizon and thus comparable computational demands to the IC. The analysis aims to assess the trade-offs between control approaches for trajectory tracking while satisfying the computational constraints of the onboard UAV implementation.

The controllers will be tested in several scenarios with different setpoints, initial conditions, and reference trajectories. The evaluation will be performed according to the quadratic criterion (4.1), the Integral Square Error (ISE) defined as

$$ISE = \frac{1}{N}\sum_{k=0}^{N} e_{1,k}^2, \tag{7.32}$$

with respect to the discrete LTI system, where $e_{1,k} = (r_{1,k} - x_{1,k})$ is the control error considering the first element of state $x_{1,k}$ only, and the energy required to control the system described as

$$E = \frac{1}{N}\sum_{k=0}^{N} \boldsymbol{u}_k^2. \tag{7.33}$$

In some cases, the evaluation will be performed for various initial conditions. The average of the values and their variance is used in order to make the comparison clearer. For example,

for ISE, the average value between realizations is denoted by

$$\overline{ISE} = \frac{1}{M}\sum_{i=0}^{M} ISE_i, \tag{7.34}$$

where ISE_i is the ISE value for i-th of M realizations, and the variance of the values as

$$\sigma^2(ISE) = \frac{1}{M}\sum_{i=0}^{M}\left(ISE_i - \overline{ISE}\right)^2. \tag{7.35}$$

The figures of state, control, and interpolating coefficient were generated for all scenarios. The interpolating coefficient is convenient to observe because it gives the usage ratio for the optimal LQR problem (when $c = 0$, the controller is optimal).

Explicit solutions were acquired in MATLAB using Multi-Parametric Toolbox 3 (MPT3) [41], which is a MATLAB toolbox for multi-parametric optimization, computational geometry, and MPC. A standard implementation of the explicit MPC in MPT3 has been used. Computational geometric features and multi-parametric solver from MPT3 were employed for the design of explicit IC. In MATLAB, the implicit solutions were acquired using YALMIP [61], which is a toolbox for modeling and solving optimization problems such as semidefinite, quadratic, or linear programming.

The optimization package CVXPY [28] was used to obtain the implicit solution to both MPC and IC in Python programming language, which interfaced with LP and QP solvers. The Polytope[1] package was used to work with invariant sets.

7.2.1 Stabilization, Setpoint Control, and Trajectory Tracking for 2nd Order LTI System

First, the system used for IC and MPC performance testing will be presented. It is a deterministic discrete-time LTI system with box-constrained state and control vectors described as follows

$$\boldsymbol{x}_{k+1} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\boldsymbol{x}_k + \begin{bmatrix}1\\ 0.3\end{bmatrix}\boldsymbol{u}_k, \tag{7.36}$$

$$\begin{bmatrix}-10\\ -5\end{bmatrix} \leq \boldsymbol{x}_k \leq \begin{bmatrix}10\\ 5\end{bmatrix}, \tag{7.37}$$

$$-1 \leq \boldsymbol{u}_k \leq 1. \tag{7.38}$$

The quadratic criterion is defined with constant weights $\boldsymbol{Q} = \boldsymbol{I}_2$, $\boldsymbol{R} = 1$. Both, MPC and IC, were designed for the N-step positively invariant controlled set $\mathcal{C}^{14}$.

7.2.1.1 Problem Description and Controller Design

In stabilizing control, the eIC was designed with an additional VC law for the S-step controlled positively invariant set $\mathcal{C}^S$, where $S = 5$. The number of steps S in $\mathcal{C}^S$ was determined by its gradual increase and subsequent evaluation of control quality in the stabilization from the vertices of $\mathcal{C}^N$ to the origin.

[1] Polytope – `https://pypi.org/project/polytope/`

Table 7.1: Evaluation of the criterion for the MPC, IC, and eIC in the case of stabilization from the vertices of $\mathcal{C}^N$ to the origin

	$\bar{J}$	%	$\sigma^2(J)$
MPC	$7.81 \cdot 10^2$	-	$2.51 \cdot 10^4$
IC	$8.71 \cdot 10^2$	+11.52%	$8.71 \cdot 10^4$
eIC	$7.81 \cdot 10^2$	+0.00%	$2.51 \cdot 10^4$

Table 7.2: Evaluation of the ISE and energy consumption for the MPC, IC, and eIC in case of stabilization from the vertices of $\mathcal{C}^N$ to the origin

	$\overline{ISE}$	%	$\sigma^2(ISE)$	$\bar{E}$	%	$\sigma^2(E)$
MPC	24.5	-	5.32	0.490	-	$8.41 \cdot 10^{-2}$
IC	27.5	+12.24%	5.44	0.453	−7.55%	$8.95 \cdot 10^{-2}$
eIC	24.5	+0.00%	5.32	0.490	+0.00%	$8.41 \cdot 10^{-2}$

For the control to the constant setpoint the eIC includes the additional set $\boldsymbol{\Omega}^m$ with LQR. Criterial weights for designing $\boldsymbol{u}^m$ control law were derived based on criterion used for designing $\boldsymbol{u}^h$, where weight $\boldsymbol{Q}^m$ was set as $\boldsymbol{Q}^m = \boldsymbol{Q}^h = \boldsymbol{I}_2$. The weight $\boldsymbol{R}^m$ was initially set as $\boldsymbol{R}^m = \boldsymbol{R}^h = 1$; to allow $\boldsymbol{\Omega}^m$ to cover a larger region than $\boldsymbol{\Omega}^h$, $\boldsymbol{R}^m$ was gradually increased. Several eICs with different values of $\boldsymbol{R}^m$ were designed and tested for control from the vertices of set $\boldsymbol{\Omega}^l$ to the origin. Subsequently, according to the value of the criterion function, the controller with the best result was selected, whose parameter was set as $\boldsymbol{R}^m = 72$.

In the case of trajectory tracking, MPC and IC were designed for N-steps, where $N = 150$ is the simulation length. To design the IC and the incorporated LQRs, the weighting matrices were chosen as follows

$$\boldsymbol{Q}^h = \boldsymbol{Q}, \boldsymbol{R}^h = \boldsymbol{R}, \boldsymbol{Q}^l = \begin{bmatrix} 10^{-2} & 0 \\ 0 & 5^{-2} \end{bmatrix}, \boldsymbol{R}^l = 10 \cdot \boldsymbol{R}, \tag{7.39}$$

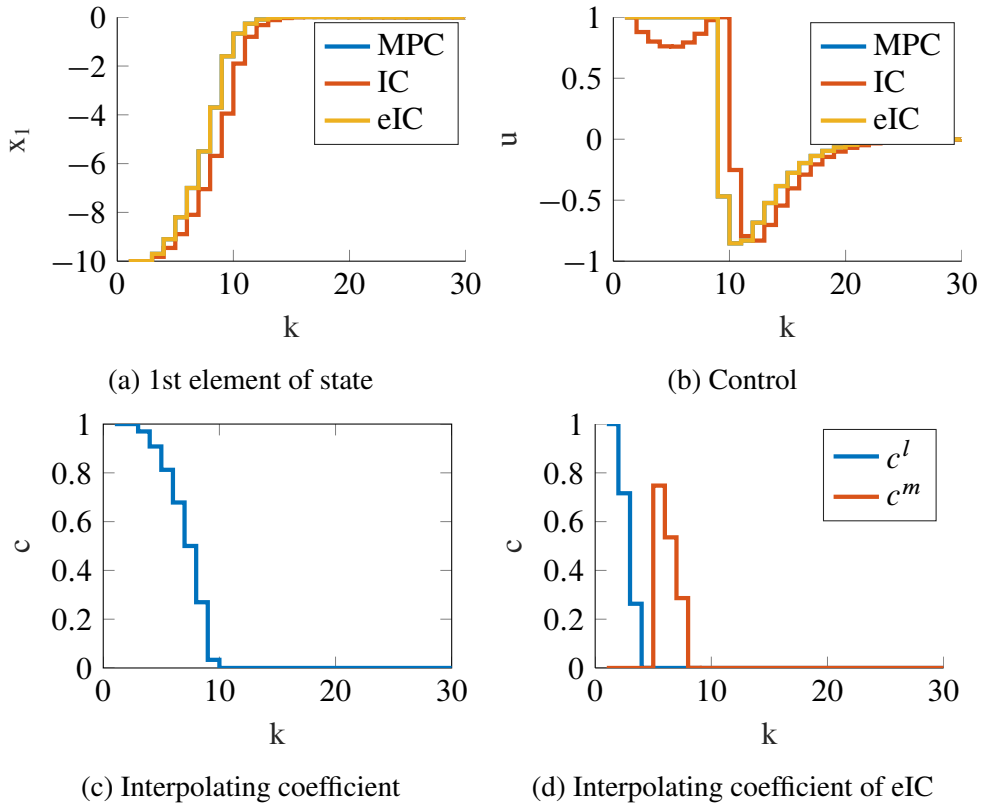
where the LQR for the invariant set $\boldsymbol{\Omega}^l$ was designed to cover as large volume as possible with the increase of weight for control action penalty $\boldsymbol{R}^l$ and decrease of weight for state penalty $\boldsymbol{Q}^l$ according to state constraints (7.37).

The evaluation was performed using MATLAB in discrete-time simulations using a standard desktop PC with Intel Core i7-4790.

7.2.1.2 Stabilization with 2nd Order LTI System

In the stabilization case, the system was controlled from all vertices of $\mathcal{C}^{14}$ to the origin (see Figure 7.5). On average, the IC delivered an 11.52% increase in criterion value (see Table 7.1), however, with reduced energy effort by 7.55% (see Table 7.2). The performance of eIC was identical to the MPC in both the criterion and the ISE and energy since it was designed to efficiently cover the state space.

The IC stabilized more slowly than MPC initially due to less aggressive control Figure 7.5. The figures of the interpolating coefficient IC (in Figure 7.5c) and eIC (in Figure 7.5d) show both controllers behaved as $\boldsymbol{u}^h$ given by optimal LQR after 10 steps, even earlier for the eIC.



(a) 1st element of state

(b) Control

(c) Interpolating coefficient

(d) Interpolating coefficient of eIC

Figure 7.5: Stabilization of system

For the case of stabilizing control, the IC showed slightly worse performance at much lower complexity, mainly because it did not use the full range of control action at first. The eIC even achieved identical results to the MPC.

7.2.1.3 Setpoint Control with 2nd Order LTI System

The ability of the controllers to steer the system to a given setpoint was tested on the reference signals generated by the sine wave

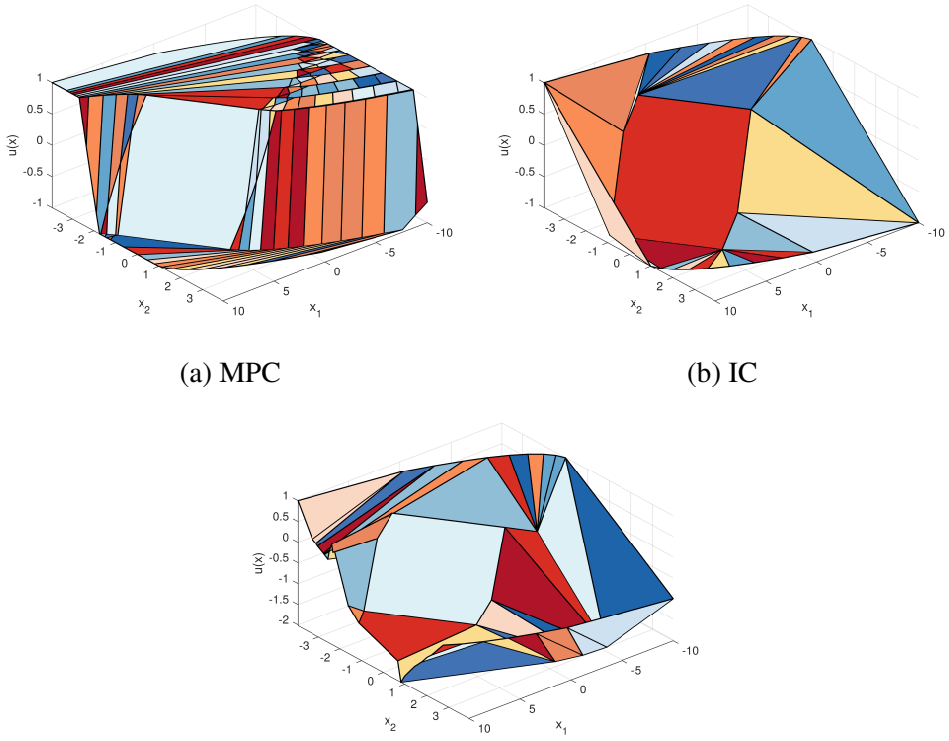
$$\boldsymbol{x}_{r,k} = \left[10 \cdot \sin\left(\frac{9\pi k}{N}\right), 0\right]^\mathsf{T}, \tag{7.40}$$

where $N = 150$ was given by the length of the simulation, and by step-function

$$\boldsymbol{x}_{r,k} = \begin{cases} [-10, 0]^\mathsf{T}, & k < \frac{N}{2}, \\ [10, 0]^\mathsf{T}, & k \geq \frac{N}{2}. \end{cases} \tag{7.41}$$

The MPC uses a full 4D explicit solution (2D state and 2D setpoint), while the interpolating controllers apply 2D explicit laws adjusted for the setpoint (see Figure 7.6). The MPC control law is much more complex than the interpolating methods, with 11,372 regions to just 27 regions for the IC and 45 for the eIC. However, the interpolating controllers must adjust their solution when the setpoint $\boldsymbol{x}_r$ changes.

When following the sine wave, the interpolating controllers reduced the cost, the ISE, and energy compared to the MPC (see Table 7.3). As can be seen in Figure 7.7, this is due



(a) MPC

(b) IC

(c) eIC

Figure 7.6: Explicit solution to MPC, IC, and eIC for setpoint $\boldsymbol{x}_r = [5,\ 0]^\mathsf{T}$

Table 7.3: Evaluation of the criterion, ISE, and energy consumption for the MPC, IC, and eIC in case of following the sine wave signal in setpoint control case

	J	%	ISE	%	E	%
MPC	$1.16 \cdot 10^5$	-	39.0	-	$7.68 \cdot 10^{-1}$	-
IC	$1.08 \cdot 10^5$	-6.90%	32.1	-17.69%	$6.29 \cdot 10^{-1}$	-18.10%
eIC	$1.07 \cdot 10^5$	-7.76%	31.2	-20.00%	$6.20 \cdot 10^{-1}$	-19.27%

Table 7.4: Evaluation of the criterion, ISE, and energy consumption for the MPC, IC, and eIC in case of following the step signal from origin in setpoint control case

	J	%	ISE	%	E	%
MPC	$1.66 \cdot 10^5$	-	14.2	-	$1.48 \cdot 10^{-1}$	-
IC	$1.74 \cdot 10^5$	$+4.82\%$	19.4	$+36.62\%$	$1.05 \cdot 10^{-1}$	-29.05%
eIC	$1.75 \cdot 10^5$	$+5.42\%$	20.3	$+42.96\%$	$9.9 \cdot 10^{-2}$	-33.04%

to the overly aggressive control of the MPC, caused by the setpoint $\boldsymbol{x}_{r,k}$ varying at every time step k with the resulting overshoot.

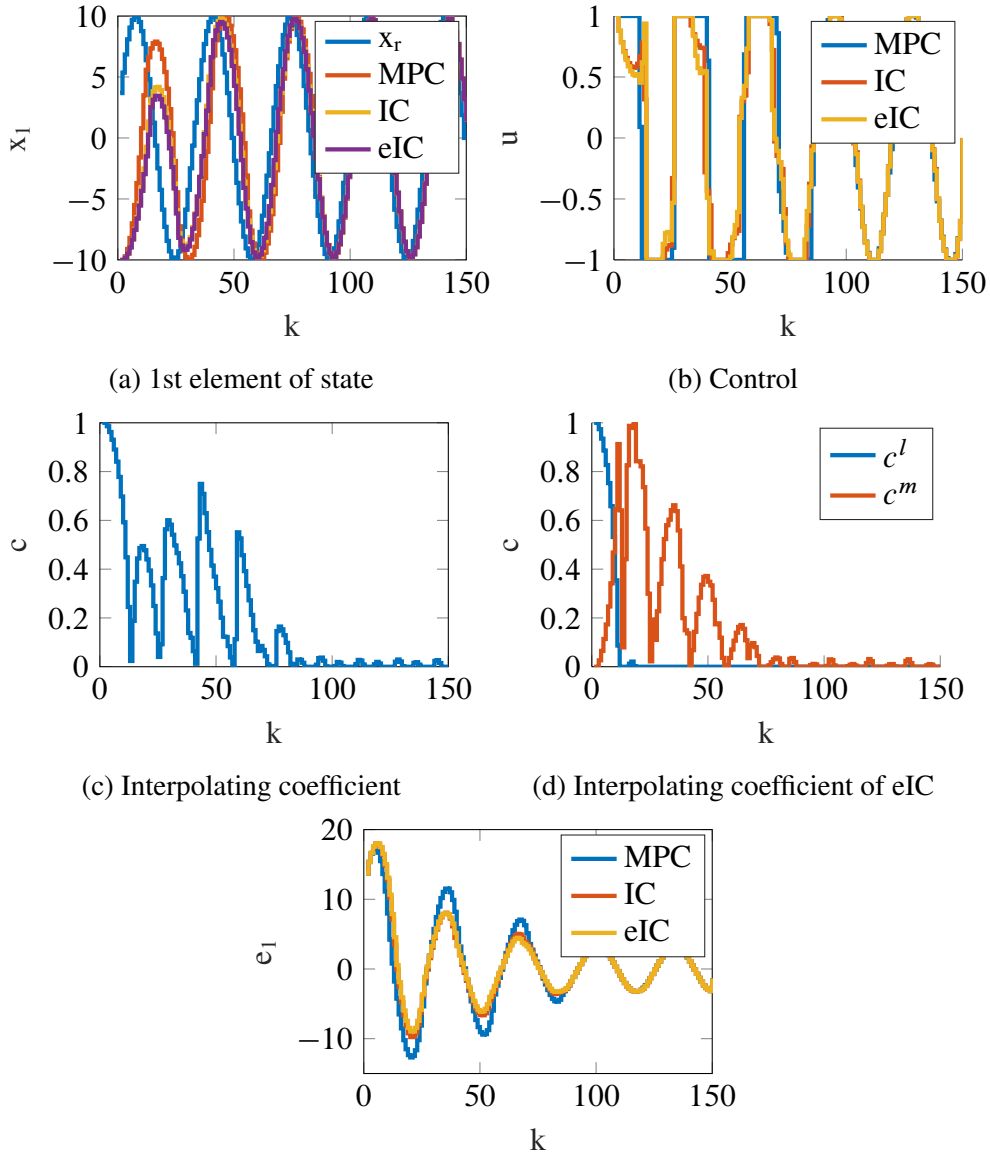
In the case of the step signal, which was followed from the origin of the state space, all controllers showed the same behavior (see Figure 7.8), however, from the half of the simulation where the step occurred, the MPC achieved a faster response as it used the full range of control action. According to the value of the interpolating coefficient, the interpolating controllers use the low-gain controller significantly during the step, which may explain the reduced control action. This increased ISE by 30-40% but decreased energy by 30% (see Table 7.4).

Results were similar for the random initial conditions that were sampled with a uniform distribution over the set $\mathcal{C}^N$ (see Tables 7.5 and 7.6). According to Tables 7.7 and 7.8, worse results were obtained when controlling from the vertices of the set $\mathcal{C}^N$. For the interpolating controllers, there was an increase in the criterion of 6 to 7%, likely due to poorer response to large errors (see Table 7.5).

The computational time evaluation of the setpoint control algorithms was not conducted. Preliminary testing showed that the transformation of the IC partition takes time comparable to the entire computation of the MPC control action. When the computation time of the IC control action was included, the time was even longer than that of MPC. Since the eIC has more regions than the IC, the transformation took almost twice as long. In order to make the methods comparable in terms of computational effort, the partition transformation algorithm would need optimization. For this reason, an implicit problem will be solved for the trajectory tracking task.

Table 7.5: Evaluation of the criterion for the MPC, IC, and eIC in case of following the step signal from random points of state space in setpoint control case

	$\bar{J}$	%	$\sigma^2(J)$
MPC	$1.66 \cdot 10^4$	-	$1.27 \cdot 10^4$
IC	$1.74 \cdot 10^4$	$+4.82\%$	$1.26 \cdot 10^4$
eIC	$1.75 \cdot 10^4$	$+5.42\%$	$1.30 \cdot 10^4$



(a) 1st element of state

(b) Control

(c) Interpolating coefficient

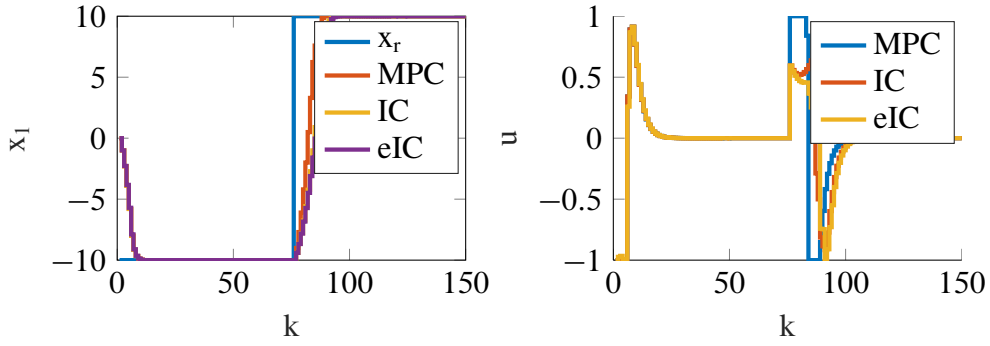
(d) Interpolating coefficient of eIC

(e) Control error

Figure 7.7: Following the sine wave signal in setpoint control case

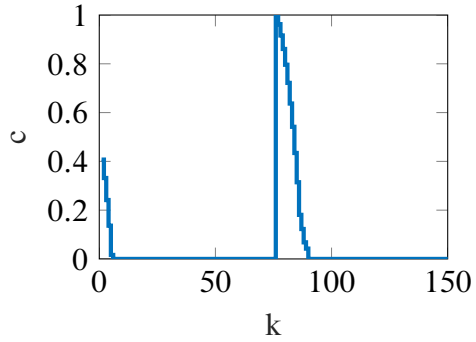
Table 7.6: Evaluation of the ISE and energy consumption for the MPC, IC, and eIC in case of following the step signal from random points of state space in the setpoint control case

	$\overline{ISE}$	%	$\sigma^2(ISE)$	$\bar{E}$	%	$\sigma^2(E)$
MPC	14.4	-	0.731	0.148	-	$6.77 \cdot 10^{-3}$
IC	19.6	+36.11%	0.732	0.104	−29, 73%	$6.88 \cdot 10^{-3}$
eIC	20.4	+41.67%	0.757	0.0979	−33.85%	$6.87 \cdot 10^{-3}$

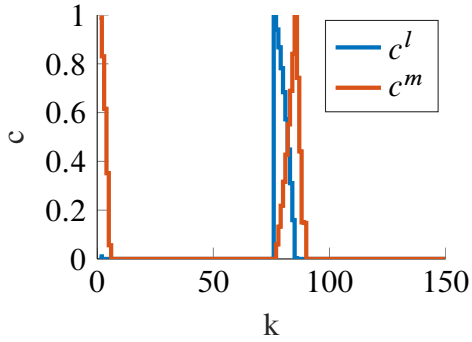


(a) 1st element of state

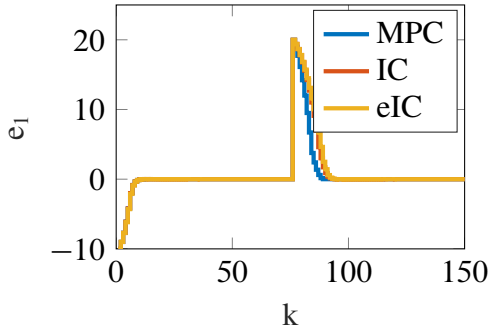
(b) Control



(c) Interpolating coefficient



(d) Interpolating coefficient of eIC



(e) Control error

Figure 7.8: Following the step signal in setpoint control case

Table 7.7: Evaluation of the criterion for the MPC, IC, and eIC in case of following the step signal from vertices of set $\mathcal{C}^N$ in the setpoint control case

	$\bar{J}$	%	$\sigma^2(J)$
MPC	$1.84 \cdot 10^4$	-	$4.60 \cdot 10^6$
IC	$1.96 \cdot 10^4$	+6.52%	$6.35 \cdot 10^6$
eIC	$1.97 \cdot 10^4$	+7.07%	$6.69 \cdot 10^6$

Table 7.8: Evaluation of the ISE and energy consumption for the MPC, IC, and eIC in case of following the step signal from vertices of set $\mathcal{C}^N$ in the setpoint control case

	$\overline{ISE}$	%	$\sigma^2(ISE)$	$\bar{E}$	%	$\sigma^2(E)$
MPC	0.521	-	4.10	$3.76 \cdot 10^{-3}$	-	$2.74 \cdot 10^{-2}$
IC	0.677	+29.94%	5.25	$2.46 \cdot 10^{-3}$	−34.57%	$1.80 \cdot 10^{-2}$
eIC	0.702	+34.74%	5.43	$2.34 \cdot 10^{-3}$	−37.77%	$1.71 \cdot 10^{-2}$

Table 7.9: Evaluation of the criterion, ISE, and energy consumption for the MPC, IC, and eIC in the case of tracking the sinewave reference trajectory

	J	%	ISE	%	E	%
MPC	$3.13 \cdot 10^2$	-	$3.20 \cdot 10^{-1}$	-	$5.36 \cdot 10^{-1}$	-
IC	$3.13 \cdot 10^2$	+1%	$3.23 \cdot 10^{-1}$	+0.94%	$5.42 \cdot 10^{-1}$	+1.12%

7.2.1.4 Trajectory Tracking with 2nd Order LTI System

The IC based trajectory tracking was experimentally tested with three different reference signals; first, a sine wave signal described in Equation (7.40), second a step function

$$\boldsymbol{x}_{r,k} = \begin{cases} [-5, 0]^\mathsf{T}, & k < \frac{N}{2}, \\ [5, 0]^\mathsf{T}, & k \geq \frac{N}{2} \end{cases} \tag{7.42}$$

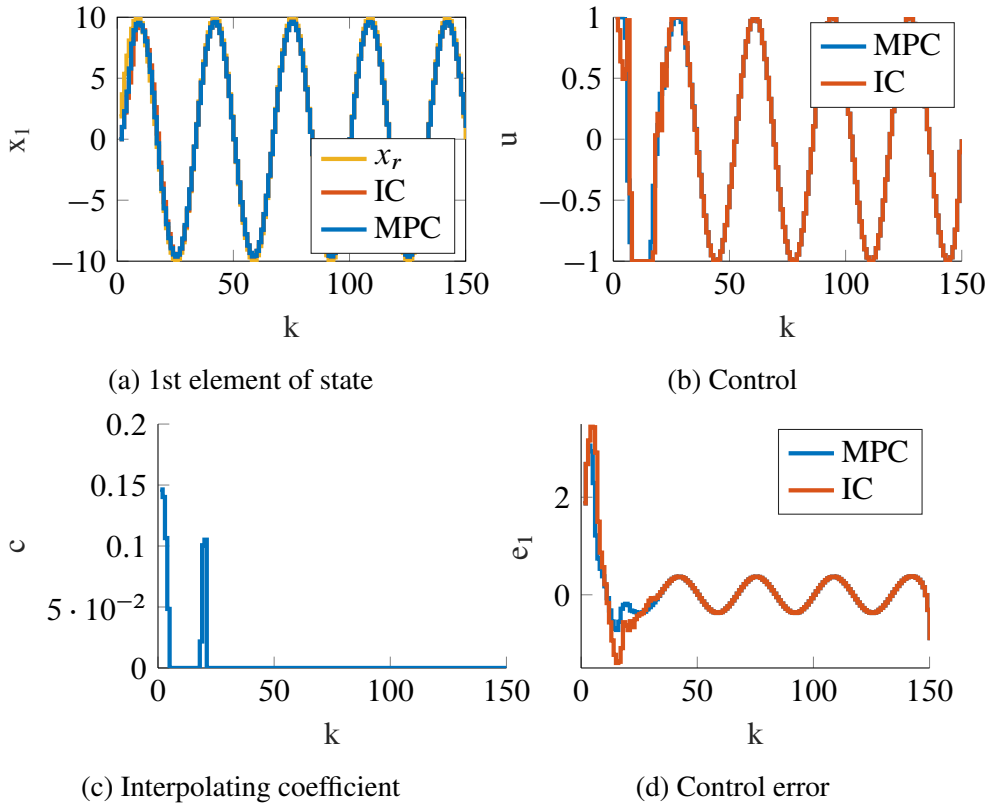
and finally a triangular reference signal

$$\boldsymbol{x}_{r,k} = \left[10 - 20 \left| \frac{2k}{N} - 1 \right|, 0 \right]^\mathsf{T}. \tag{7.43}$$

Although only the reference for the first state is non-zero, it is still a trajectory since it is time-parameterized. Since the effect of different initial conditions was tested in the case of the control to the setpoint, trajectory tracking was tested with zero initial conditions (i.e. $\boldsymbol{x}_0 = [0, 0]^\mathsf{T}$). As already mentioned, due to the difficulty with implementation, the eIC was not used in the case of trajectory tracking. Thus, the MPC is only compared to the IC.

Figure 7.9 shows the results for the scenario with the sinewave reference signal. In Figure 7.9b, it can be observed that aggressive control occurred for both controllers at the beginning of the simulation, in particular, the MPC used the full range of action control. Even though there was a sharp decrease in the interpolating coefficient (see Figure 7.9c) and it reached low values even in the maxima, the IC did not achieve as a smooth reduction in control error as the MPC. However, after a few steps, both controllers began to follow the reference trajectory almost identically, which can be observed both in terms of the state in Figure 7.9a and the control error in Figure 7.10d. Table 7.9 also shows that IC is only 1% worse than MPC in terms of optimality criterion, ISE, and energy consumption.

The scenario with a step reference trajectory is shown in Figure 7.10. Again, there is a rapid decrease in the interpolating coefficient in Figure 7.10c. Nevertheless, it can be observed in Figure 7.10a that the system has a slightly slower response in the case of IC control. Also, around the time instant $k = 75$ where the step occurs, IC does not use control actions as effectively



(a) 1st element of state

(b) Control

(c) Interpolating coefficient

(d) Control error

Figure 7.9: Tracking a sine wave reference signal

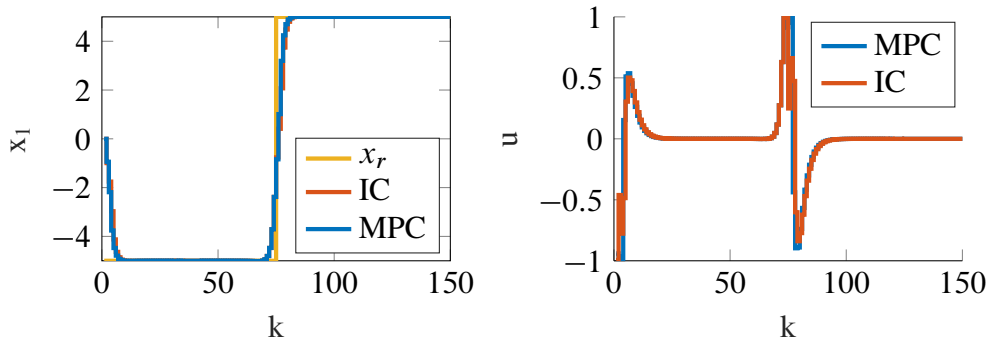
Table 7.10: Evaluation of the criterion, ISE, and energy consumption for the MPC, IC, and eIC in the case of tracking the step reference trajectory

	J	%	ISE	%	E	%
MPC	$1.18 \cdot 10^2$	-	$6.24 \cdot 10^{-1}$	-	$7.83 \cdot 10^{-2}$	-
IC	$1.37 \cdot 10^2$	$+16\%$	$7.60 \cdot 10^{-1}$	$+21.79\%$	$7.40 \cdot 10^{-2}$	-5.49%

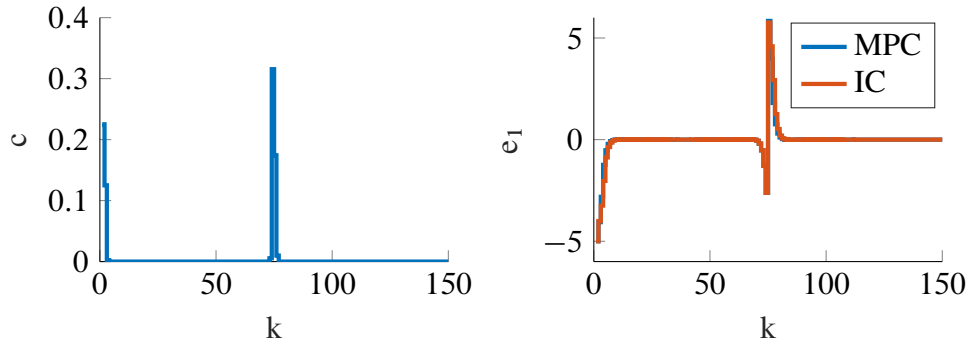
as MPC (see Figure 7.10b). Table 7.10 shows that the minor differences between the IC and MPC control were much more evident in both the optimality criterion, where the IC scored 16% worse and in the ISE, where the error had an increase of even 21.79%. However, due to the fluctuations in the control around $k = 75$, there was an energy saving of 5.49%.

In the scenario with triangular reference trajectory, shown in Figure 7.11, the behavior differed as in the sinusoidal trajectory tracking only at the beginning of the simulation, and thereafter the system showed the same behavior for both MPC and IC. Figure 7.11c shows a very clear drop in interpolating coefficient c, which is zero after a few steps. Both the state response (see Figure 7.11a) and the control error (see Figure 7.11d) are adjusted more slowly by the IC and in Figure 7.11b there are significant fluctuations at the start in the IC's control action. Table 7.11 shows that IC performed only about 8% worse than MPC for both the optimality criterion and ISE, and almost 7% energy savings were achieved compared to the MPC.

Based on Figures 7.9, 7.10, and 7.11 of the sine wave, step, and triangular trajectory tracking, respectively, it can be seen that the MPC and the IC behave similarly. The MPC achieves better tracking only in the presence of large trajectory changes. Despite the rapid changes, the system

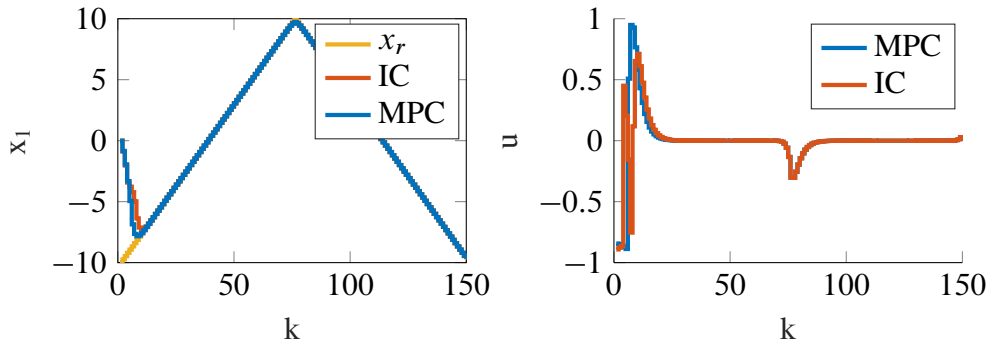


(a) 1st element of state (b) Control

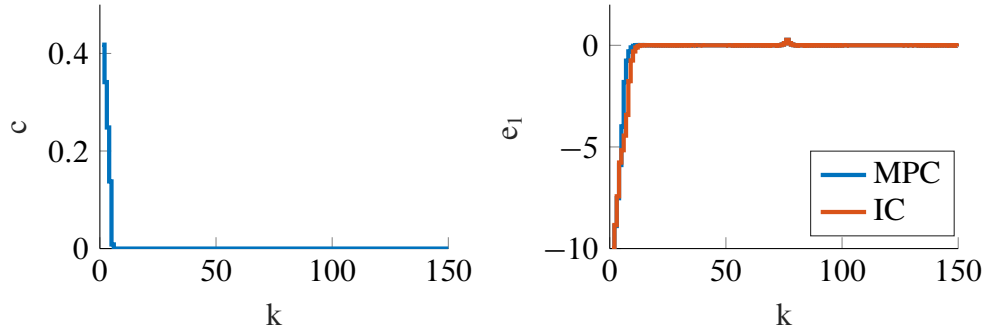


(c) Interpolating coefficient (d) Control error

Figure 7.10: Tracking a step reference signal



(a) 1st element of state (b) Control



(c) Interpolating coefficient (d) Control error

Figure 7.11: Tracking a triangular reference signal

Table 7.11: Evaluation of the criterion, ISE, and energy consumption for the MPC, IC, and eIC in the case of tracking the triangular reference trajectory

	J	%	ISE	%	E	%
MPC	$2.91 \cdot 10^2$	-	1.79	-	$5.29 \cdot 10^{-2}$	-
IC	$3.13 \cdot 10^2$	$+8\%$	1.93	$+7.82\%$	$4.93 \cdot 10^{-2}$	-6.81%

is controlled more smoothly compared to the case of setpoint control. The excellent performance of the IC is also indicated by the low value of the interpolating coefficient in all three scenarios depicted in figures 7.9c, 7.10c, and 7.11c.

In general, interpolating controllers perform well in scenarios where there are small or smooth changes in the reference signal generating setpoint $\boldsymbol{x}_{r,k}$. In the scenario where a smooth sine wave reference signal was followed, the IC achieved the same results as the MPC, while in setpoint control, where the controller was not provided with information about the entire reference trajectory but only its point, the IC and eIC achieved even better results than the MPC.

These results are particularly impressive in the context of MPC employed in the trajectory tracking scenario, which knew the entire reference trajectory 150 steps ahead. In most applications, the prediction horizon is greatly reduced to minimize the computational complexity of the MPC, however, this reduction may negatively affect robustness and control quality. In contrast, parameters of the LQR, which is used in the IC for tracking, can be calculated in the case of an LTI system in advance for long trajectories without a noticeable increase in computational complexity in the control loop itself.

In the trajectory tracking scenario, in addition to the optimality criterion, ISE, and energy consumption, the computational time to acquire each control action was also monitored. To achieve higher-quality results, several solvers with the same parameters were used to ensure the accuracy of the solution. Both QP and LP were solved using the well-known solvers SEDUMI [90], GUROBI [37], and CPLEX [25]. The computational time for all trajectories was similar, so the times measured while following the sine wave trajectory are presented.

The computational time demands are compared in Table 7.12. The calculation of invariant sets, LQRs, and initialization in YALMIP are not included in the results because they can be computed in advance outside the control loop. In Table 7.12, the total time of computing in [s], the maximum computing time of control action in [ms], and the percentage reduction compared with MPC are denoted. The results imply, that the most suitable solver for the investigated example is GUROBI, which performed better with solving QP for MPC and even LP for IC. Regardless of the type of solver, the IC has achieved significant computational time savings.

In case $\boldsymbol{x}_k \in \boldsymbol{\Omega}^h$, it is not necessary to solve the LP for IC because it is known that $c = 0$. The LQR $\boldsymbol{u}^h$ for $\boldsymbol{\Omega}^h$ can be used directly. This setup was also investigated and it resulted in another dramatic decrease in computational time ($\sim 10 - 20\%$) compared to the standard setup, however, the longest period $t_{\max}$ remained the same.

7.2.2 Trajectory Tracking for UAV

Now, the ability of IC to steer a system with such fast dynamics as a UAV will be tested. First, the stabilization capability of the UAV using IC will be investigated. Further, the modified IC will be employed in a scenario where the UAV will be controlled along a reference trajectory.

Table 7.12: The time demands for MPC and IC for the tracking of sine wave reference trajectory

SEDUMI	t [s]	%	$t_{\max}$ [ms]	%
MPC	854	-	102	-
IC	110	−87%	17	−83%
GUROBI	t [s]	%	$t_{\max}$ [ms]	%
MPC	126	-	17	-
IC	31	−75%	5	−71%
CPLEX	t [s]	%	$t_{\max}$ [ms]	%
MPC	218	-	30	-
IC	84	−62%	15	−50%

7.2.2.1 Planar UAV Model

For easier analysis and better insight, a planar UAV model (see Figure 7.12) will be employed rather than the full model described in Chapter 2. The planar model exhibits similar behavior but is reduced both in the number of state variables and in the complexity of the equations of motion because it describes motion in only two axes and it is sufficient to describe rotational motion around one axis only. The model will be described using a similar notation as in Chapter 2. The dynamics along $\vec{y}^L$ and $\vec{z}^L$-axis with attitude ϕ as rotation around axis $\vec{x}^L$ is considered.

The state vector is composed as

$$\boldsymbol{x} = [y^L, z^L, \phi, v_y^L, v_z^L, \omega], \tag{7.44}$$

where y^L and z^L is a position in $\vec{y}^L$ and $\vec{z}^L$-axis in [m], respectively, v_y^L and v_z^L are velocities along the same axes in $\left[\mathrm{m}\cdot\mathrm{s}^{-1}\right]$, ϕ is the angle in [rad] and ω the angular rate around $\vec{x}^L$-axis in $\left[\mathrm{rad}\cdot\mathrm{s}^{-1}\right]$.

The nonlinear planar UAV dynamics is described as

$$\ddot{y}^L(t) = -\frac{F_T^B(t)}{m}\sin(\phi(t)), \tag{7.45}$$

$$\ddot{z}^L(t) = -g + \frac{F_T^B}{m}\cos(\phi(t)), \tag{7.46}$$

$$\ddot{\phi}(t) = \frac{\tau_{R_x}(t)}{I_x}, \tag{7.47}$$

where F_T^B is the force of collective thrust in body coordinate frame in [N], m is the mass of UAV in [kg], τ_{R_x} is the collective torque in $[\mathrm{N}\cdot\mathrm{m}]$ produced by rotors around $\vec{x}^L$-axis and I_x is the moment of inertia around $\vec{x}^L$ in $\left[\mathrm{kg}\cdot\mathrm{m}^2\right]$.

The dynamics can be linearized according to the equilibrium in the hover state given by

$$\phi = 0,\ F_T^B = m\cdot g,\ \tau_{R_x} = 0. \tag{7.48}$$

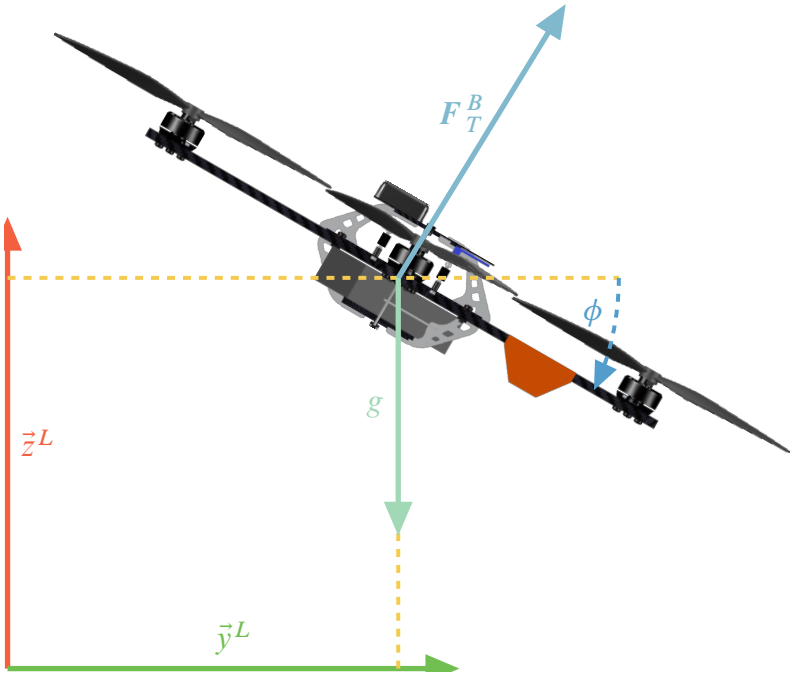


Figure 7.12: Planar UAV in the local frame

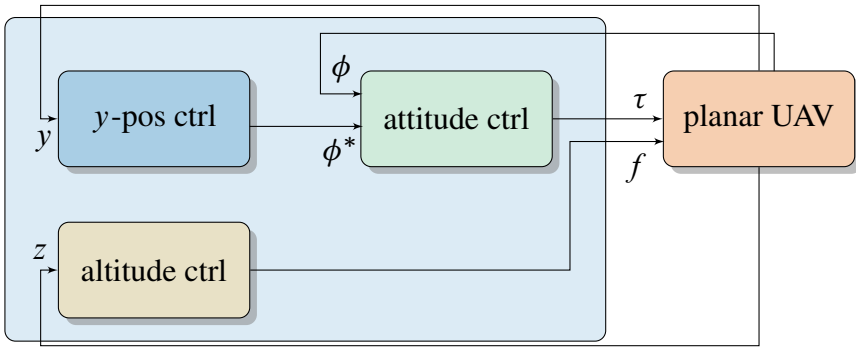


Figure 7.13: Schematics of planar UAV control system

After the linearization, the system description changes to the form

$$\Delta \ddot{y}^L(t) = -g\Delta\phi(t), \tag{7.49}$$

$$\Delta \ddot{z}^L(t) = -g + \frac{\Delta F_T^B}{m}, \tag{7.50}$$

$$\Delta \ddot{\phi}(t) = \frac{\Delta \tau_{R_x}(t)}{I_x}, \tag{7.51}$$

where variables of a linear system are denoted by Δ.

As the IC cannot be directly applied to non-linear systems, the proposed controller can control the translation in $\vec{y}^L$ and $\vec{z}^L$ using desired acceleration, if the attitude control is handled by the UAV's autopilot. The desired acceleration is recalculated according to the equations (7.49)–(7.51) to attitude control reference as

$$\phi_r(t) = -\frac{\ddot{y}^L_{r,k}(t)}{g}, \tag{7.52}$$

$$\boldsymbol{F}_T^B(t) = m\left(\ddot{z}^L_r(t) + g\right), \tag{7.53}$$

where ϕ_r and $\boldsymbol{F}_T^B$ are desired angle and collective thrust, respectively. Moreover, as a side effect, the controller is independent of the UAV parameters m and I_x, as they are only compensated in the calculations for the attitude and thrust control setpoint. The scheme of the controller is depicted in Figure 7.13. The angle constraint is indirectly satisfied within the $\vec{y}^L$-axis position control reference.

The parameters of the model were chosen to match a real UAV, specifically the Crazyflie 2.0 from Bitcraze (see Figure 7.14). Similarly, the state and control constraints were set according to either experimentally measured or manufacturer-supplied values. Regarding the parameters of the model, the mass of the drone was set as $m = 0.03\,\text{kg}$, the moment of inertia as $I_x = 2.3951 \cdot 10^{-5}\,\text{kg} \cdot \text{m}^2$. Constraints were chosen as

$$\begin{aligned} -2 \leq & y^L \leq 2, & -5 \leq \dot{y}^L \leq 5, \\ -1.25 \leq & z^L \leq 1.25, & -5 \leq \dot{z}^L \leq 5. \end{aligned}$$

The position constraints are set to comply with the parameters of the testing laboratory (see Figure 7.15) during real-world tests, and the velocity constraints are set to allow controller testing even during faster maneuvers. The condition for the position in the z-axis is negative to enable

Figure 7.14: MicroUAV Crazyflie 2.0 by Bitcraze equipped with Lighthouse deck for indoor localization

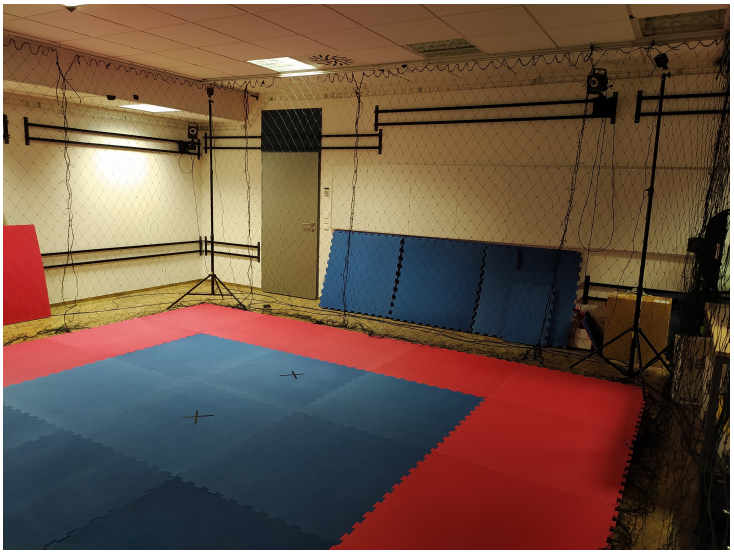


Figure 7.15: Flight arena equipped with HTC Vive V2 base station and VICON Motion Capture system

stabilization control. For the controller, the $\vec{z}^L$-axis position measurement is transformed so that the system zero is at a height of 1.25 m.

The weights of the quadratic criterion, which were also utilized in the design of the controllers, were chosen adequately according to the constraints and behavior of the UAV

$$\boldsymbol{Q}_y^h = \begin{bmatrix} \frac{1}{0.5\cdot(2.5)^2} & 0 \\ 0 & \frac{1}{0.5\cdot(5)^2} \end{bmatrix} = \begin{bmatrix} 0.16 & 0 \\ 0 & 0.04 \end{bmatrix}, \quad \boldsymbol{R}_y^h = \frac{1}{2}\cdot 0.15 \cdot \frac{4}{0.03}\cdot \sin 30^\circ = 5,$$
$$\boldsymbol{Q}_z^h = \begin{bmatrix} \frac{1}{0.1\cdot(1.25)^2} & 0 \\ 0 & \frac{1}{0.5\cdot(5)^2} \end{bmatrix} = \begin{bmatrix} 0.64 & 0 \\ 0 & 0.04 \end{bmatrix}, \quad \boldsymbol{R}_z^h = 5^{-2},$$

where $\boldsymbol{Q}_y^h$, $\boldsymbol{R}_y^h$ and $\boldsymbol{Q}_z^h$, $\boldsymbol{R}_z^h$ are weights for the position control utilized in MPC, IC, and eIC in $\vec{y}^L$ and $\vec{z}^L$-axis, respectively.

7.2.2.2 Stabilizing Interpolating Control for UAV

For the stabilization case, explicit versions of the MPC and the interpolating controllers were designed. The N-step controlled positively invariant set $\mathcal{C}^N$ was constructed for both position controllers; in the case of $\vec{y}^L$-axis $N = 100$ and in the case of $\vec{z}^L$-axis $N = 100$. These values were used as MPC prediction and control horizon settings. The resulting control functions are shown in Figures 7.16 and 7.17. The eIC was constructed with additional additional S-step controlled positively invariant set $\mathcal{C}^S$ with a different VC law, where $S = 75$ and $S = 69$ for $\vec{y}^L$ and $\vec{z}^L$-axis position control, respectively.

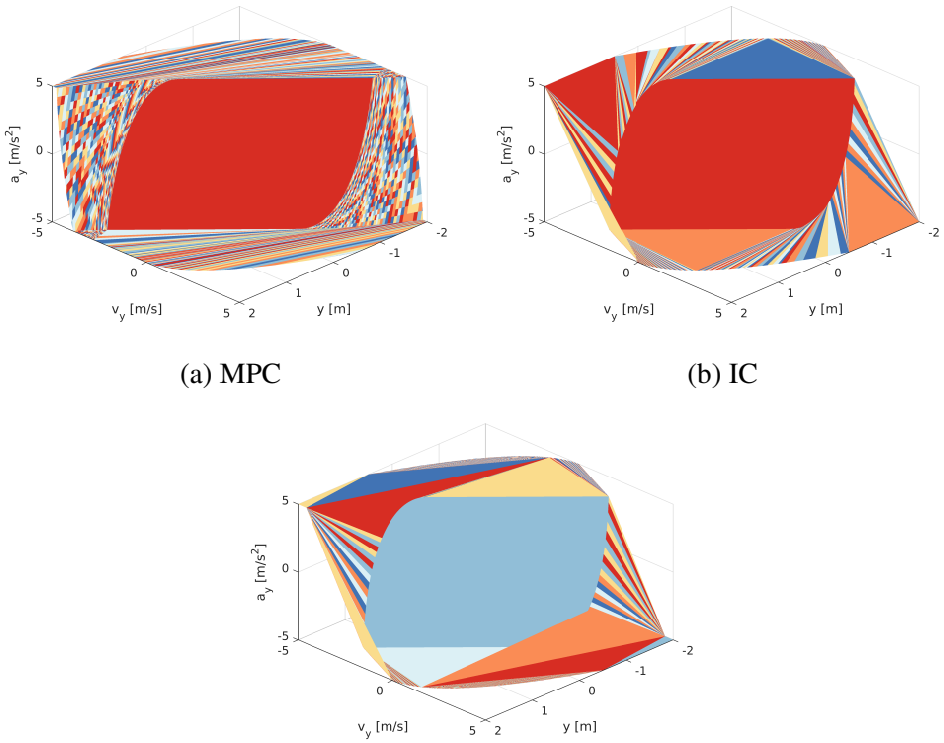
The simulations were performed together with the computational demands analysis in MATLAB on a standard desktop PC with Intel Core i9-9900 and 64GB DDR4 RAM. The nonlinear dynamics of the planar quadrotor UAV was simulated in Simulink. The initial conditions were generated with zero speed, angle, and angular rate with the coordinates y^L and z^L at the vertices of $\mathcal{C}^N$. All the compared controllers were implemented using MPT3 [41] and transformed into MATLAB functions.

A block of discrete PID controller with experimentally obtained parameters $K_p = 0.3$, $K_d = 0.003$, and $K_i = 0.0001$ was used to simulate the attitude control, which would otherwise be done by the autopilot. The PID controller was executed with a period $T_{s_att} = 0.001$s.

The complexity of the controllers, the time required to construct the explicit control functions, and the time required to compute the control actions can be found in Table 7.13. It can be seen, that the MPC has a more complex control law by an order of magnitude and took much longer to obtain than the interpolating controllers. Interestingly, the explicit law for IC has only a few regions less than eIC but took almost twice as long to obtain. Difficulties in constructing the controller function can be caused, for example, by the shape of the sets on which the IC is defined. The table also includes the time taken to compute the control action for 10^4 random samples of the state space from the set $\mathcal{C}^N$, where the interpolating controllers were much faster.

The comparison of the control quality was based on the values of the criterion of optimality (see Equation (4.1)), Integral Square Error (ISE, see Equation (7.32)), and energy consumption (see Equation (7.33)). Their average and variance are presented in Tables 7.14 and 7.15. The analysis of results implies that the interpolating controllers behave comparably to MPC.

The example with planar UAV stabilization problem showed that the IC is certainly a good alternative to the MPC, where it delivers similar quality results with much lower complexity and computational demands.



(a) MPC

(b) IC

(c) eIC

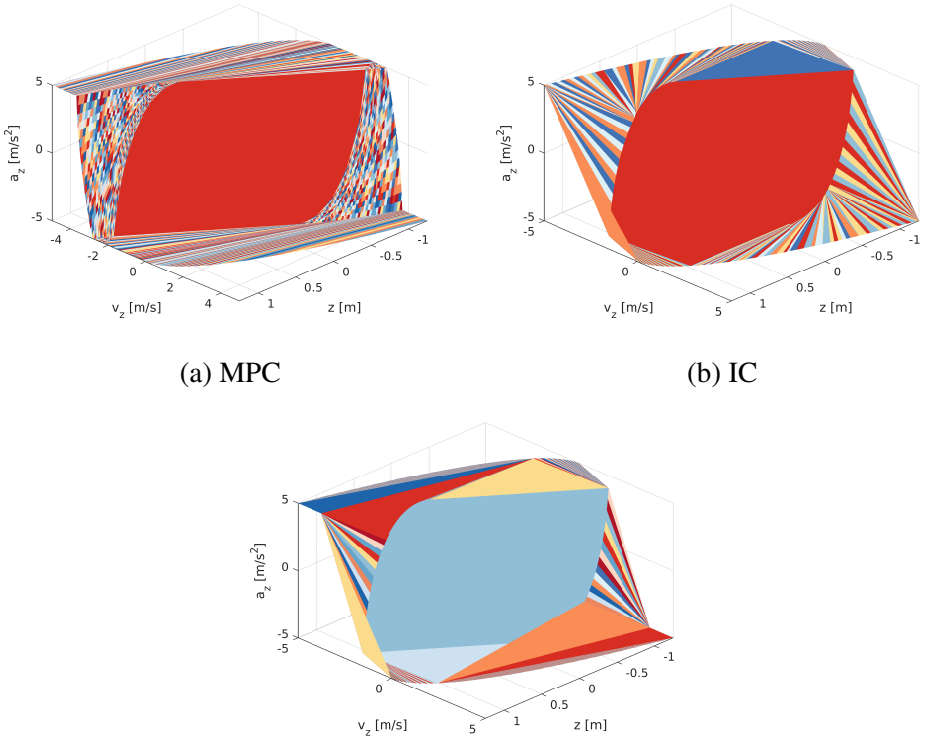
Figure 7.16: Explicit solution to MPC, IC, and eIC for $\vec{y}^L$-axis position control

Table 7.13: The complexity of the solution and the time demands for IC, eIC, and MPC

	complexity		time demands	
	regions	comp. time [min]	time [sec]	%
MPC	7240	46.87	25.37	-
IC	552	32.83	2.08	−91.80%
eIC	560	16.93	2.22	−91.25%

Table 7.14: Evaluation of the criterion for IC, eIC, and MPC, where $\bar{J}$ is the average of the criterion values of all realizations, $\sigma^2(J)$ is the variance of the criterion values and % is the percentage of criterion value change compared to the MPC

	$\bar{J}$	$\sigma^2(J)$	%
MPC	7.9625	$5.74 \cdot 10^3$	-
IC	7.9639	$5.75 \cdot 10^3$	+0.02%
eIC	7.9639	$5.75 \cdot 10^3$	+0.02%



(a) MPC

(b) IC

(c) eIC

Figure 7.17: Explicit solution to MPC, IC, and eIC for $\vec{z}^L$-axis position control

Table 7.15: Evaluation of the ISE and energy consumption for the MPC, IC, and eIC in case of following the step signal from vertices of set $\mathcal{C}^N$

	$\overline{ISE}$	%	$\sigma^2(ISE)$	$\bar{E}$	%	$\sigma^2(E)$
MPC	$9.47 \cdot 10^{-1}$	-	$6.55 \cdot 10^{-1}$	43.9	-	$2.07 \cdot 10^{-2}$
IC	$9.46 \cdot 10^{-1}$	-0.11%	$6.55 \cdot 10^{-1}$	44.3	$+0.91\%$	$2.08 \cdot 10^{-2}$
eIC	$9.46 \cdot 10^{-1}$	-0.11%	$6.55 \cdot 10^{-1}$	44.3	$+0.91\%$	$2.08 \cdot 10^{-2}$

7.2.2.3 Parameters of Trajectory Tracking for UAV

In the following sections, the trajectory tracking for the UAV will be tested. The tests are performed both in a planar model simulation and a 3D simulation environment [75] based on PyBullet (see Figure 7.18), as well as in the laboratory using Crazyflie UAV wirelessly controlled with cflib library[2].

Simulations were performed on the same standard desktop PC with Intel Core i9-9900 and 64GB DDR4 RAM with Ubuntu 22 and Python 3.8. The LPs and QPs were modeled using Python-embedded modeling language for convex optimization problems CVXPY and GUROBI [37] was utilized as an LP/QP solver because of the performance in Section 7.2.1.4. In the laboratory, the same software setup was used, but calculations were performed on a laptop HP Spectre x360 Convertible 13 with Intel Core i7-8550 and 16GB DDR3 RAM.

A PID controller with the same parameters as in the stabilization case was used to simulate the behavior of the attitude controller. The controller was implemented directly in Python and it controlled the mathematical model of the UAV with a period of $T_{s_att} = 0.001$s.

For the trajectory tracking case, the IC was designed with LQR controller $\boldsymbol{u}^l$ for trajectory tracking using the following weights

$$\boldsymbol{Q}_y^l = \begin{bmatrix} 2^{-2} & 0 \\ 0 & 5^{-2} \end{bmatrix} = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.04 \end{bmatrix}, \qquad \boldsymbol{R}_y^l = 10 \cdot 0.5 \cdot 0.15 \cdot \frac{4}{0.03} \cdot \sin 30° = 50,$$
$$\boldsymbol{Q}_z^l = \begin{bmatrix} 1.25^{-2} & 0 \\ 0 & 5^{-2} \end{bmatrix} = \begin{bmatrix} 0.64 & 0 \\ 0 & 0.04 \end{bmatrix}, \quad \boldsymbol{R}_z^l = 10 \cdot \left(5^{-2}\right) = 0.4.$$

The predictive horizon was chosen to be 8s as in [6], which with the period of the discrete model $T_s = 0.01$s makes $N = 800$. For such a long horizon, the MPC solution would not be achievable in real-time.

Therefore, the input blocking technique [21, 5], which is a form of move blocking, was used to reduce the complexity of the MPC. This technique fixes the values of the control variables in $\boldsymbol{u}_k^{k+N}$ for several time steps, resulting in a reduction of the number of degrees of freedom. To further reduce the complexity of the problem, the prediction for $T_s = 0.01$s is performed only in the first step and afterward, the model with $T_s = 0.2$s is considered, and as the horizon progresses, the control action is considered constant in intervals given by the scheme

$$\boldsymbol{U}_{block} = [1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 10], \tag{7.54}$$

resulting in a reduction of the control vector variables by 95%. This variant of the predictive controller with move blocking will be referred to hereafter as MPCMB.

Section 7.1.1 discussed the impossibility of reflecting the reference trajectory in eIC without violating the constraints. However, preliminary testing has shown that the IC may not perform well when tracking a high-frequency reference signal. Therefore, a version of the eIC was designed that uses an interpolation of both trajectory tracking and setpoint controllers. The LQR $\boldsymbol{u}^m$ of this eIC considers only the current setpoint $\boldsymbol{x}_{r,k}$. This ensures the problem's constraints are respected without compromising the trajectory tracking functionality, even if the full trajectory

[2]CFLib – `https://github.com/bitcraze/crazyflie-lib-python`

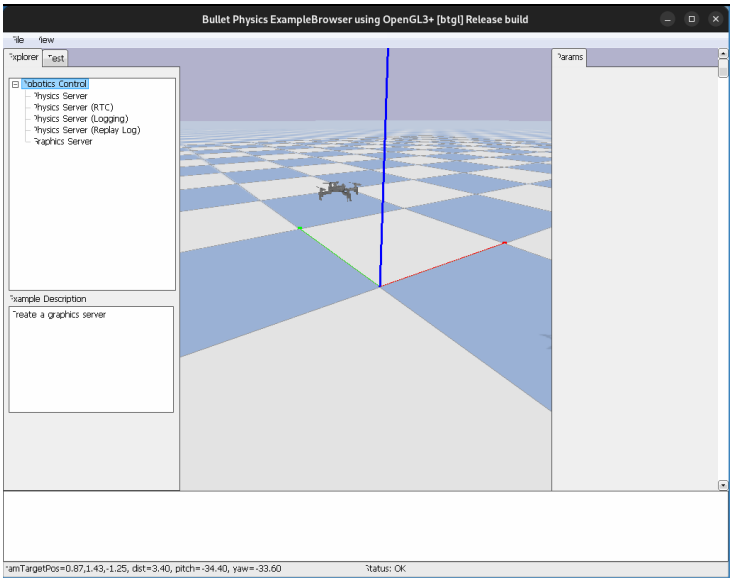


Figure 7.18: GUI of PyBullet-based Gym for single and multi-agent reinforcement learning with nano-quadcopters

is not scaled. The controller $\boldsymbol{u}^m$ was designed using weights

$$\boldsymbol{Q}_y^m = \begin{bmatrix} 2^{-2} & 0 \\ 0 & 5^{-2} \end{bmatrix} = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.04 \end{bmatrix}, \qquad \boldsymbol{R}_y^m = 0.5 \cdot 0.15 \cdot \frac{4}{0.03} \cdot \sin 30^\circ = 5,$$
$$\boldsymbol{Q}_z^m = \begin{bmatrix} 1.25^{-2} & 0 \\ 0 & 5^{-2} \end{bmatrix} = \begin{bmatrix} 0.64 & 0 \\ 0 & 0.04 \end{bmatrix}, \quad \boldsymbol{R}_z^m = \left(5^{-2}\right) = 0.04.$$

7.2.2.4 Trajectory Tracking with 2D Nonlinear Model

The planar model was implemented in Python based on the nonlinear dynamics described in equations in (7.45)–(7.47). Two types of reference trajectories were tracked: Ellipse and Lemniscate of Gerono (figure-eight shaped curve). Reference signals were generated according to equations

$$y_{r,k}^L = 1.8 \cos\left(\omega_s \cdot k \cdot T_s\right), \tag{7.55}$$
$$z_{r,k}^L = \frac{1.5}{2} \sin\left(\omega_s \cdot 2 \cdot k \cdot T_s\right), \tag{7.56}$$

for lemniscate and

$$y_{r,k}^L = 1.8 \cos\left(\omega_s \cdot k \cdot T_s\right), \tag{7.57}$$
$$z_{r,k}^L = 1.0 \sin\left(\omega_s \cdot k \cdot T_s\right), \tag{7.58}$$

for the elliptical reference. Other reference states were equal to zero vectors, however, since the reference signal is time-parameterized and hence it is the reference trajectory.

The responses to both reference signals were tested in two variants with different ω_s. First, tests were conducted for $\omega_s = 0.4$ and then for $\omega_s = 0.6$. In case of the higher value of ω_s,

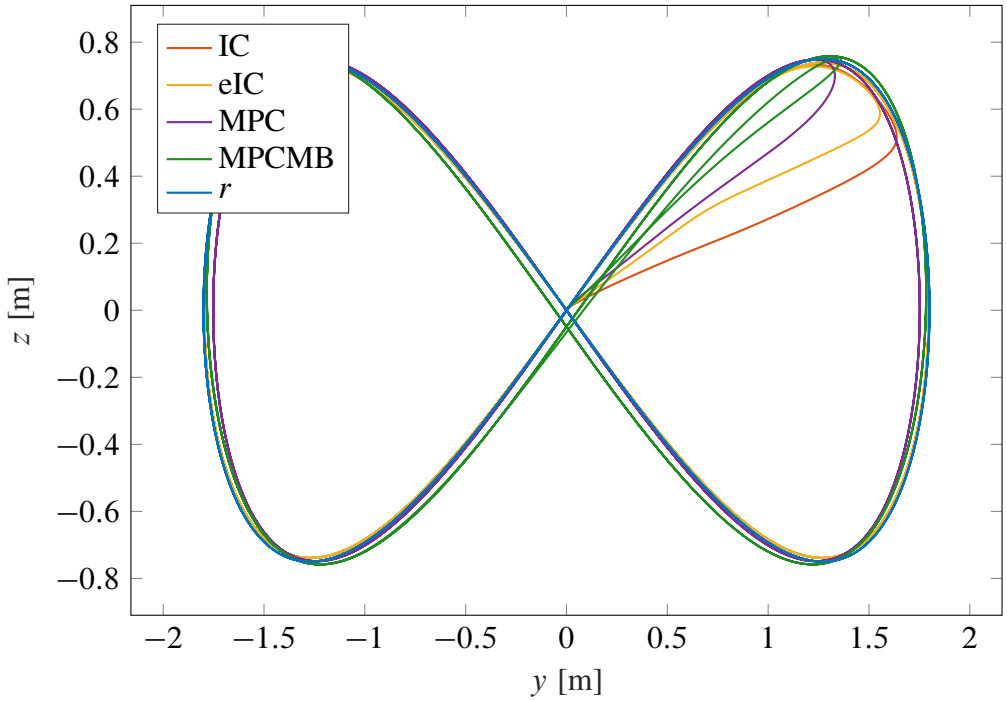


Figure 7.19: Path from tracking the lemniscate trajectory with the planar nonlinear model

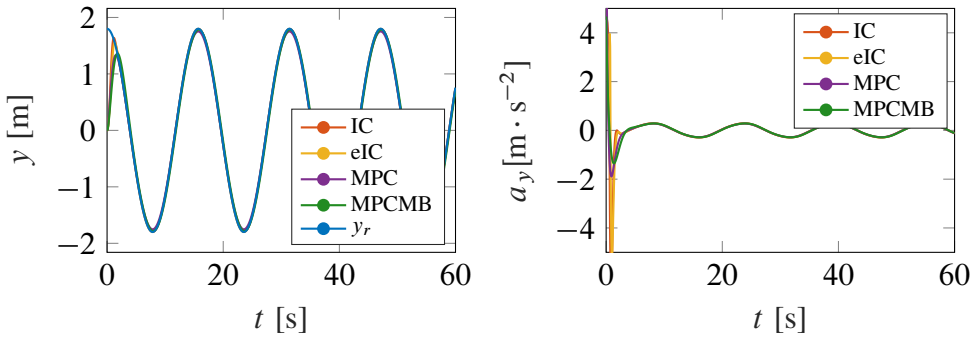
Table 7.16: Evaluation of the criterion, ISE, and energy consumption for the MPC, MPCMB IC, and eIC for the tracking of lemniscate reference trajectory with the planar UAV model

	J	%	ISE	%	E	%
MPC	7.07	-	1.76	-	21.20	-
MPCMB	7.74	+9.48	2.19	+24.43	16.80	-20.75
eIC	8.23	+16.41	1.73	-1.70	27.30	+28.77
IC	8.68	+22.77	1.50	-14.77	33.80	+59.43

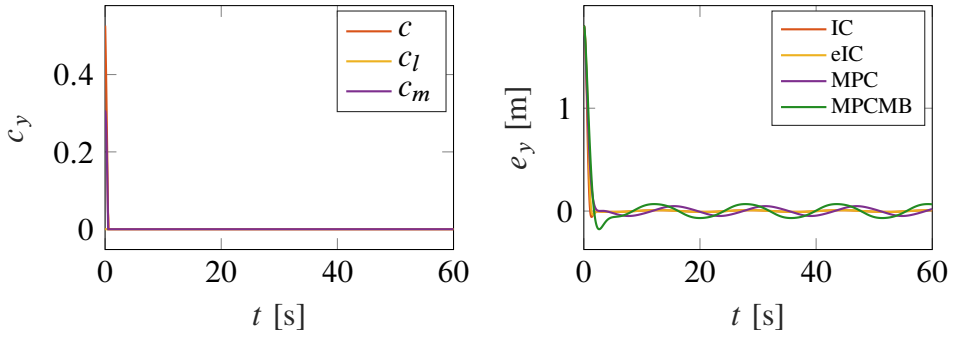
the reference signals will be referred to as tracking a high-frequency lemniscate/elliptical trajectory. The UAV's initial conditions were set to the origin of the state-space, nevertheless, the reference signals at $t\,[\mathrm{s}] = 0$ were nonzero, thus the UAV had to converge towards the reference at the start of the simulation.

Tracking Lemniscate Trajectory with $\omega_s = 0.4$ At the start of the simulation with the lemniscate reference trajectory, it can be seen in Figure 7.19 that the interpolating and predictive controllers have quite different paths. The predictive controllers follow the reference trajectory at a more distant point while the interpolating controllers try to align with the reference as soon as possible.

However, according to Figures 7.20 and 7.21 for each axis, it can be observed that the interpolating coefficients were zero except at the very beginning of the simulation in the $\vec{y}^L$-axis, thus the interpolating controllers achieved optimal behavior for most of the time. From Table 7.16, it can be seen that the interpolation-based controllers are worse in optimality criterion because they consumed more energy. However, they followed the trajectory more closely in terms of ISE.

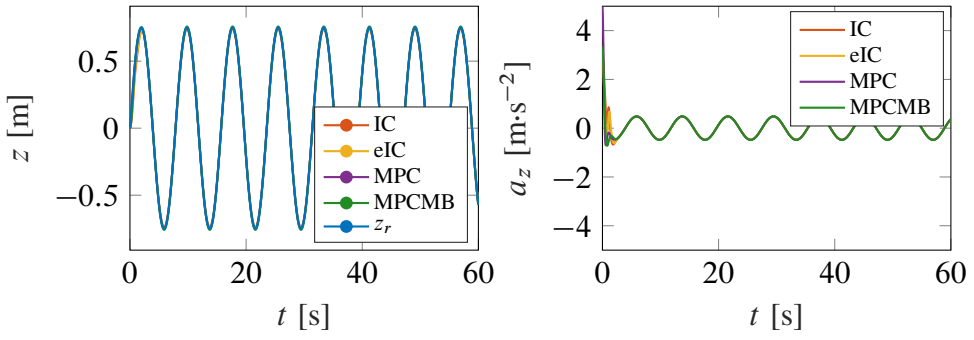


(a) Position (b) Control

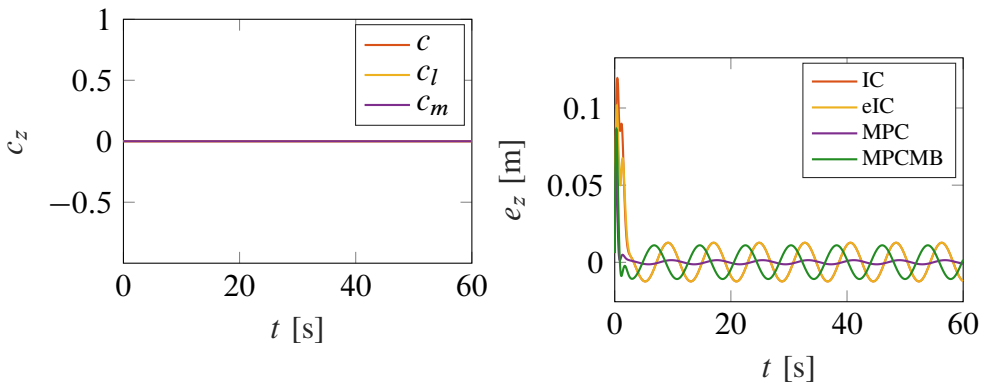


(c) Interpolating coefficient (d) Control Error

Figure 7.20: Tracking lemniscate trajectory with the planar nonlinear model in $\vec{y}^L$-axis



(a) Position (b) Control



(c) Interpolating coefficient (d) Control error

Figure 7.21: Tracking lemniscate trajectory with the planar nonlinear model in $\vec{z}^L$-axis

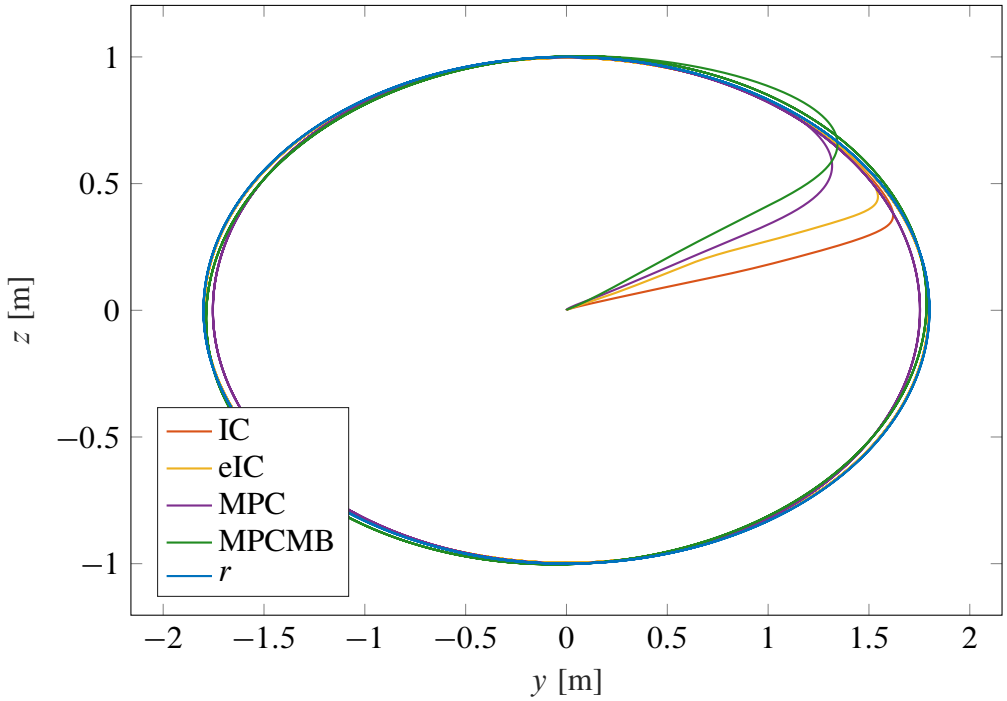
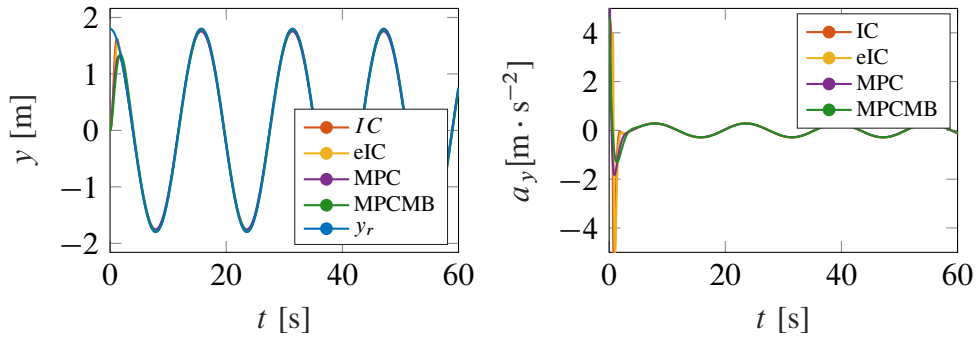


Figure 7.22: Path from tracking the spiral trajectory with the planar nonlinear model

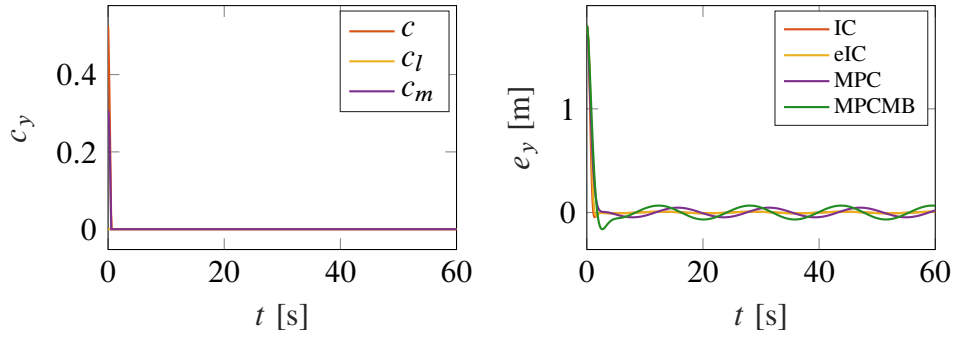
Table 7.17: Evaluation of the criterion, ISE, and energy consumption for the MPC, MPCMB IC, and eIC for the tracking of elliptical reference trajectory with the planar UAV model

	J	%	ISE	%	E	%
MPC	5.77	-	1.80	-	13.50	-
MPCMB	6.16	+6.76	2.21	+22.78	9.43	-30.15
eIC	6.58	+14.04	1.75	-2.78	20.40	+51.11
IC	6.97	+20.80	1.52	-15.56	26.90	+99.26

Tracking Elliptical Trajectory with $\omega_s = 0.4$ In the case of the elliptical reference trajectory, very similar results were obtained as in the previous case with the lemniscate trajectories. The predictive controllers again tracked the reference more slowly from the origin (see Figure 7.22), and according to Figures 7.23 and 7.24, the interpolation coefficient was zero except for the origin on the $\vec{y}$-axis for IC. According to Table 7.17, the interpolating controllers again closely followed the trajectory, but at the cost of a large increase in energy consumption, and therefore obtained worse values of the optimality criterion than the predictive controllers.

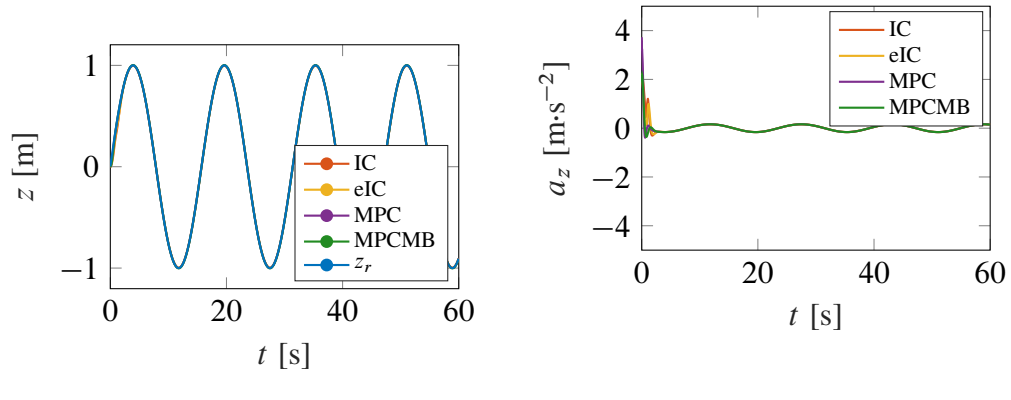


(a) Position (b) Control

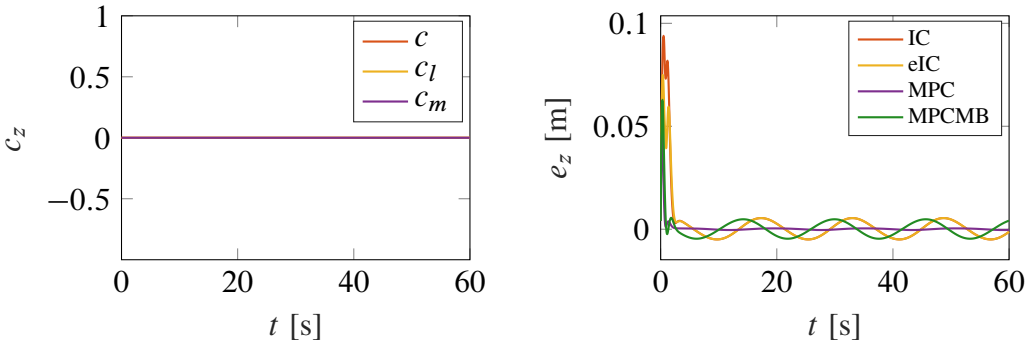


(c) Interpolating coefficient (d) Control Error

Figure 7.23: Tracking elliptical trajectory with the planar nonlinear model in $\vec{y}^{\,L}$-axis



(a) Position (b) Control



(c) Interpolating coefficient (d) Control error

Figure 7.24: Tracking elliptical trajectory with the planar nonlinear model in $\vec{z}^{\,L}$-axis

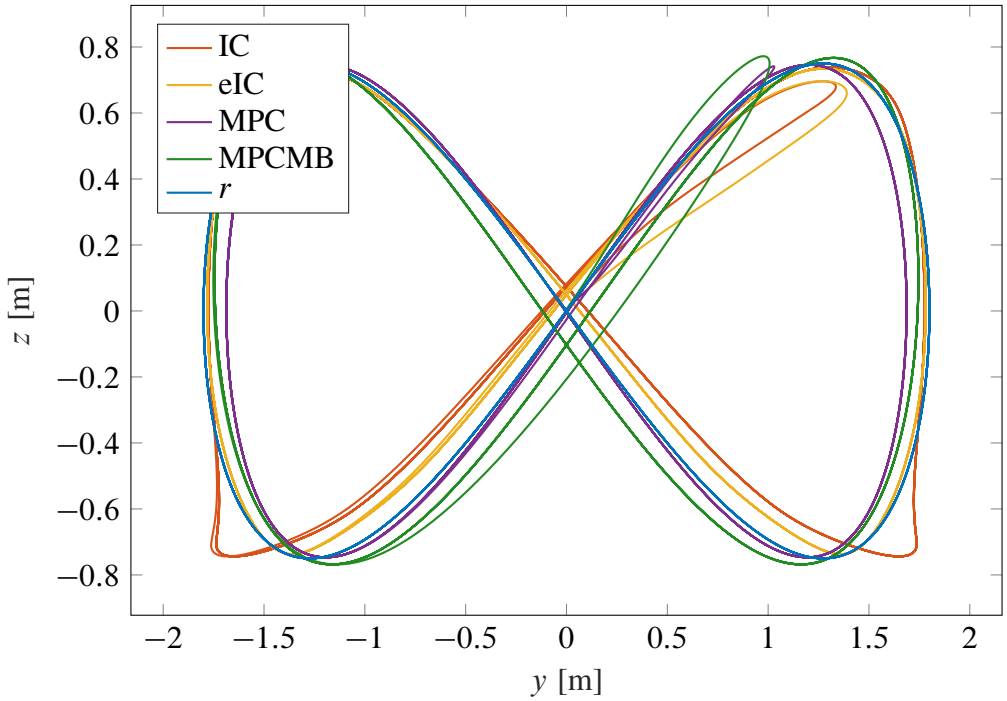


Figure 7.25: Path from tracking the high-frequency lemniscate trajectory with the planar nonlinear model

Table 7.18: Evaluation of the criterion, ISE, and energy consumption for the MPC, MPCMB IC, and eIC for the tracking of high-frequency lemniscate reference trajectory with the planar UAV model

	J	%	ISE	%	E	%
MPC	13.60	-	2.02	-	61.20	-
MPCMB	15.70	+15.44	2.64	+30.69	58.30	-4.74
eIC	16.40	+20.59	1.73	-14.36	79.30	+29.58
IC	23.50	+72.79	3.86	+91.09	98.00	+60.13

Tracking Lemniscate and Elliptical Trajectory with $\omega_s = 0.6$ Very different results were obtained for the high-frequency trajectories. According to Figures 7.25 and 7.26, the interpolating controllers again tried to follow the reference faster. It can be seen that the IC deviates significantly at two points. It could be caused by the way the controllers interact with each other because when the $\vec{y}^L$ controller increases the ϕ angle to generate higher acceleration, it also results in a deflection of the collective thrust that the $\vec{z}^L$ controller requires.

Figures 7.27 to 7.30 show large fluctuations of the IC's interpolating coefficient throughout the simulation, except for the z-axis when tracking the elliptical trajectory.

Tables 7.18 and 7.19 show that the MPC achieved the best results according to the optimality criterion. However, the eIC again followed the trajectory most closely. In contrast, the IC achieved the worst results by all criteria.

Computational Demands A comparison of the computational demands of the tested controllers is presented below. The computational demands of the controllers were evaluated

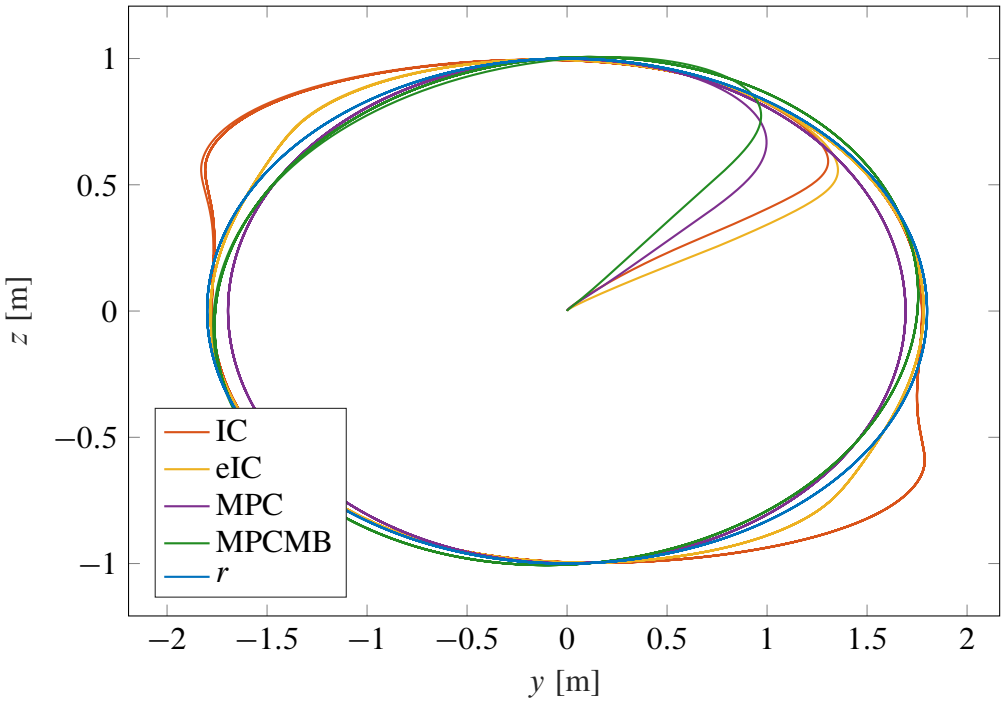
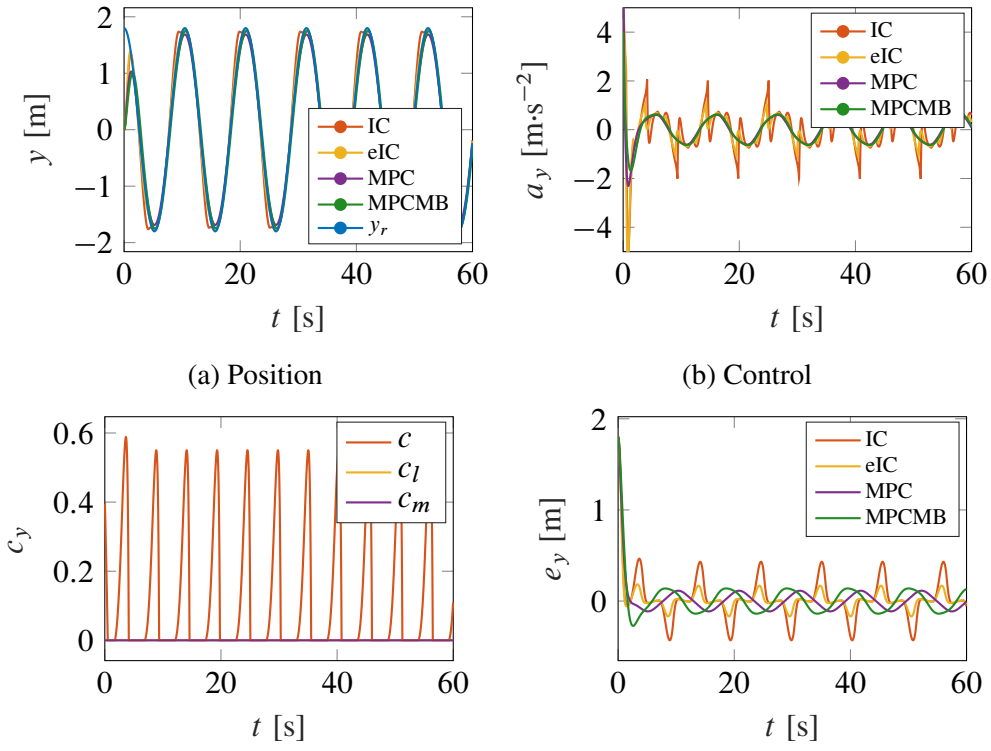


Figure 7.26: Path from tracking the high-frequency spiral trajectory with the planar nonlinear model



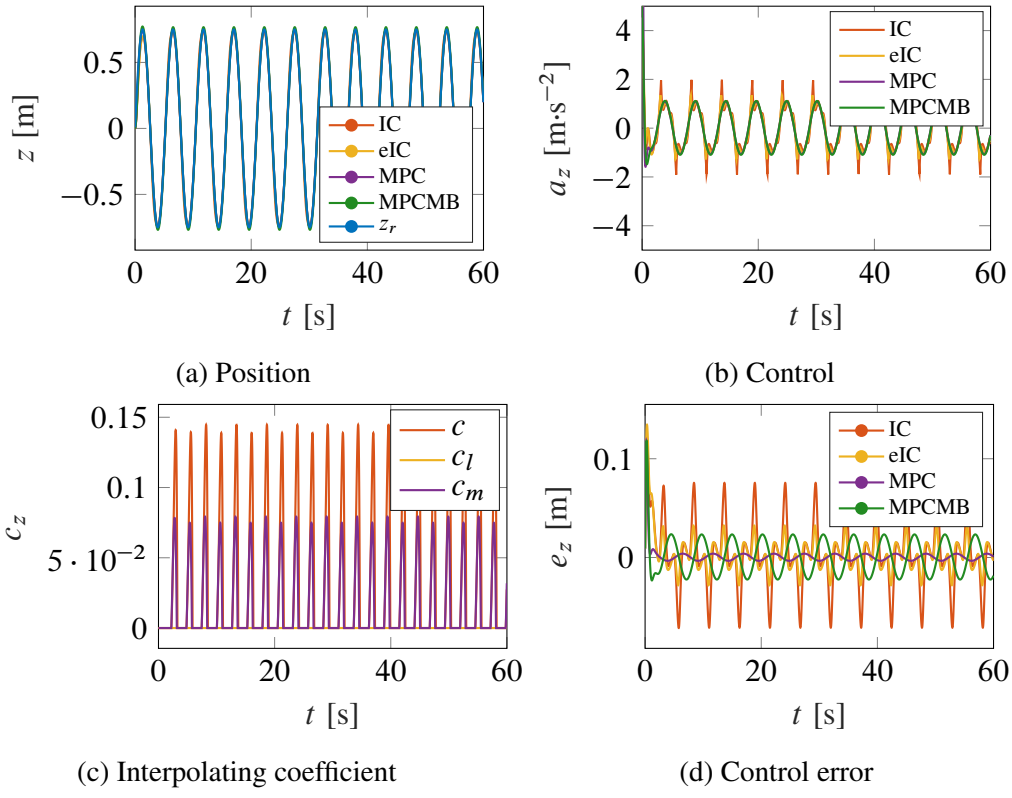
(a) Position

(b) Control

(c) Interpolating coefficient

(d) Control Error

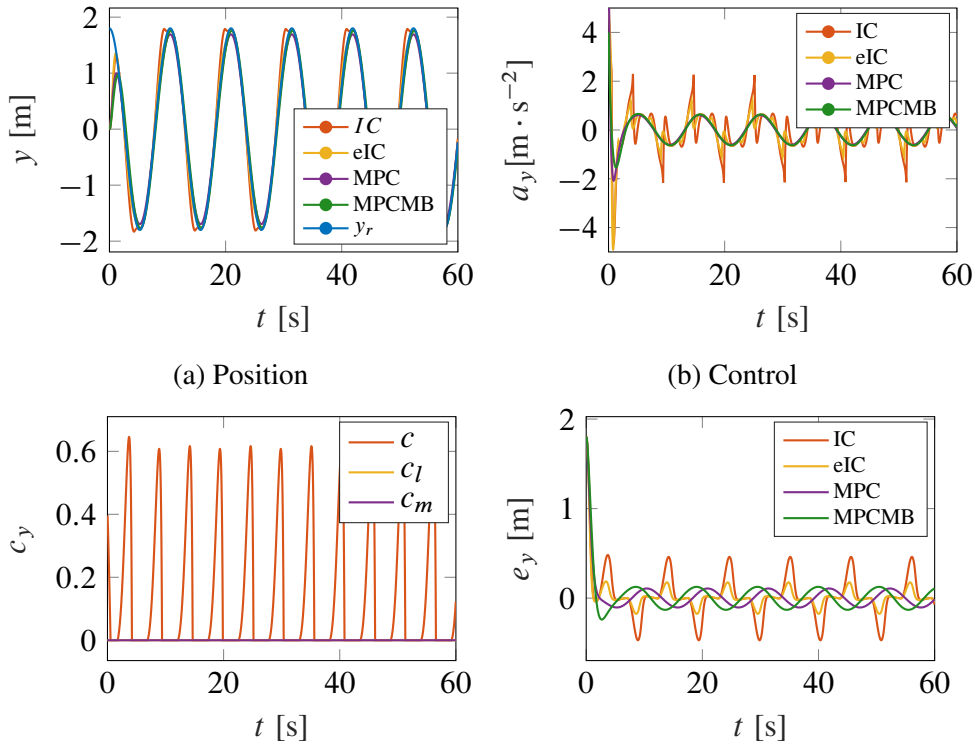
Figure 7.27: Tracking high-frequency lemniscate trajectory with the planar nonlinear model in $\vec{y}^L$-axis



(a) Position (b) Control

(c) Interpolating coefficient (d) Control error

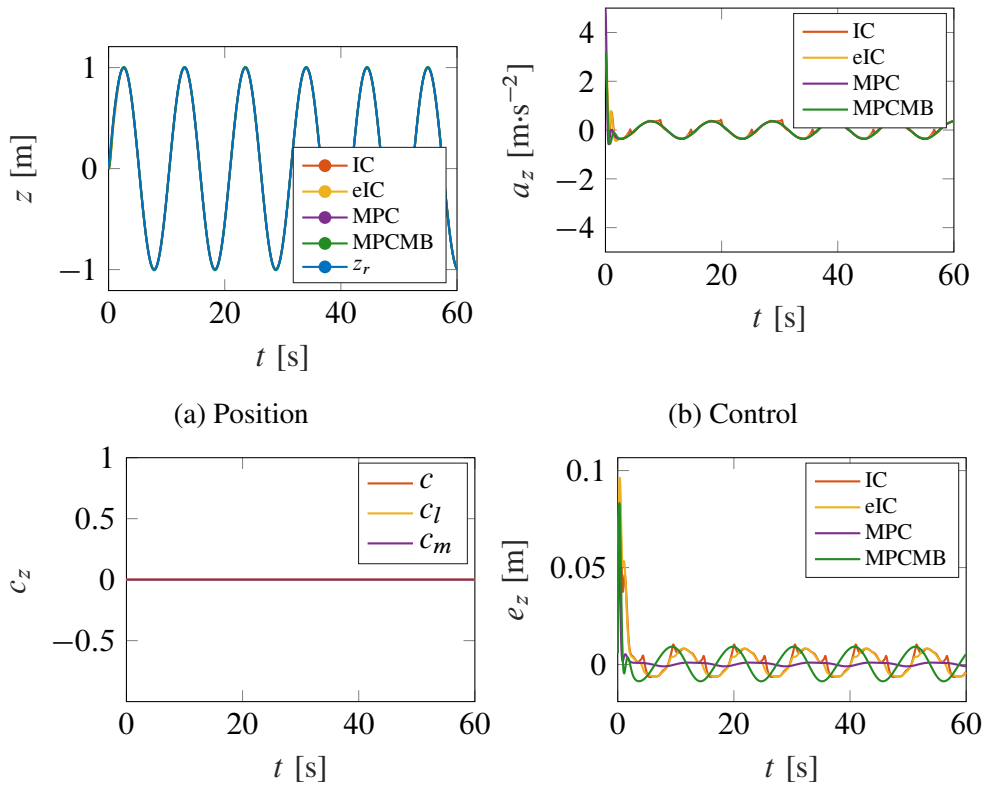
Figure 7.28: Tracking fast lemniscate trajectory with the planar nonlinear model in $\vec{z}^L$-axis



(a) Position (b) Control

(c) Interpolating coefficient (d) Control Error

Figure 7.29: Tracking high-frequency elliptical trajectory with the planar nonlinear model in $\vec{y}^L$-axis



(a) Position

(b) Control

(c) Interpolating coefficient

(d) Control error

Figure 7.30: Tracking high-frequency elliptical trajectory with the planar nonlinear model in $\vec{z}^L$-axis

Table 7.19: Evaluation of the criterion, ISE, and energy consumption for the MPC, MPCMB IC, and eIC for the tracking of high-frequency elliptical reference trajectory with the planar UAV model

	J	%	ISE	%	E	%
MPC	9.95	-	2.04	-	26.60	-
MPCMB	10.9	+9.55	2.53	+24.02	22.70	-14.66
eIC	11.9	+19.60	1.80	-11.76	42.70	+60.53
IC	16.1	+61.81	4.32	+111.76	56.70	+113.16

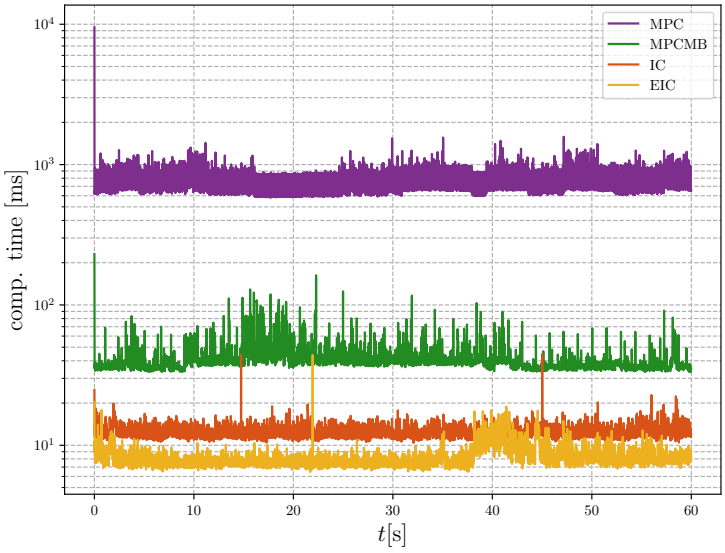


Figure 7.31: Computational time for tracking the fast lemniscate trajectory with a planar model of the UAV

Table 7.20: The time demands for IC, eIC, MPC and MPCMB for the tracking of high-frequency lemniscate reference trajectory with the planar UAV model

	t [s]	%	$t_{\max}$ [ms]	%
MPC	4180	-	9507	-
MPCMB	247	-94.09	230	-97.58
eIC	49	-98.83	44	-99.54
IC	75	-98.21	45	-99.53

for each scenario and all reference signals. However, since the results were similar, only the results for the high-frequency lemniscate trajectory are presented here.

For the planar nonlinear model scenarios, the total computation time for the control actions (denoted as t[s]) and the longest computation time (denoted as $t_{\max}$ [ms]) were recorded. The results are also supplemented with logarithmic plots showing the computation time over time.

From the logarithmic plot comparing MPC, MPCMB, IC, and eIC on a planar UAV model shown in Figure 7.31, it can be seen that calculating the MPC takes the longest time to compute the control action, followed by MPCMB. IC and eIC have similar computational demands, but eIC is faster most of the time, probably due to the shape of the $\boldsymbol{\Omega}^m$ set and LP, since interpolation was only performed between $\boldsymbol{\Omega}^h$ and $\boldsymbol{\Omega}^m$ in all scenarios.

Table 7.20 shows similar order of magnitude differences concerning the total computation time of all control actions and the longest computation time. The results clearly show that the interpolation control is a time-efficient alternative to predictive control.

7.2.2.5 Trajectory Tracking in Gym-Pybullet-Drones Environment

A PyBullet-based gym for single and multi-agent reinforcement learning with nano quadcopters [75] was utilized to test controllers under more realistic conditions. PyBullet is a popular physics engine that provides accurate and efficient simulation of rigid body dynamics. One of the advantages of the simulated environment is that it simulates the behavior of the Crazyflie UAVs, which will also be used to test the controllers in the lab. The MPC was not used in the laboratory experiment with the Crazyflie UAV or in the PyBullet simulation due to its high computational cost.

The performance of the IC, eIC, and MPCMB was tested by tracking two types of reference trajectories: the ellipse and the lemniscate. The reference trajectories used in the PyBullet simulation differed from those used in the planar model experiment. In the PyBullet simulation, the elliptical and lemniscate references were set in the $\vec{x}^L$ and $\vec{y}^L$ axes, while the $\vec{z}^L$-axis was set to a faster cosine signal to provide a more complex test of the $\vec{z}^L$-controller. Specifically, the z-axis reference signal was set to

$$z^L_{r,k} = -0.8\cos\left(3 \cdot \omega_s \cdot k \cdot T_s\right) + 1, \tag{7.59}$$

for both types of reference trajectories.

Since the PyBullet simulation is 3D, the controller designed to control the motion in the $\vec{y}^L$-axis was also used to control the motion in the $\vec{x}^L$-axis. The output of the $\vec{x}^L$ and $\vec{y}^L$ controllers had to be rotated around the $\vec{z}^L$-axis to reflect the yaw angle ψ of the UAV. Equation (7.52) is modified to reflect the rotation and generate $\phi_{r,k}$ and $\theta_{r,k}$ as

$$\phi_{r,k} = \frac{1}{g}\left(\ddot{x}^L_{r,k} \cdot \sin\left(\psi_k\right) - \ddot{y}^L_{r,k} \cdot \cos\left(\psi_k\right)\right), \tag{7.60}$$

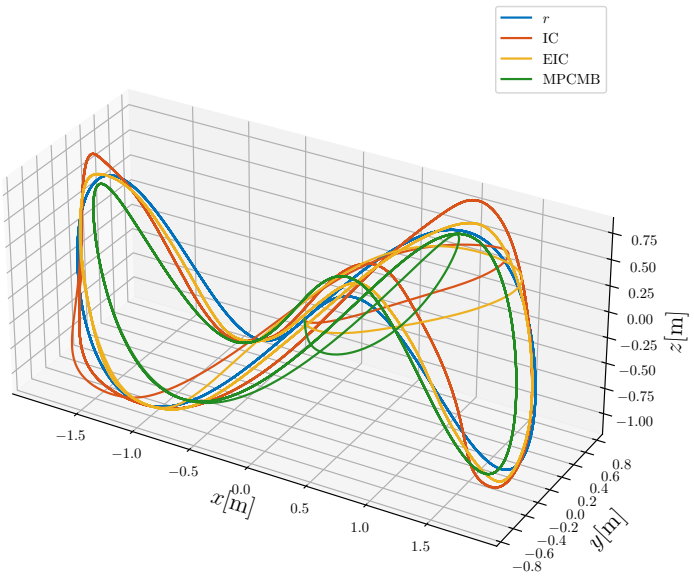
$$\theta_{r,k} = \frac{1}{g}\left(\ddot{x}^L_{r,k} \cdot \cos\left(\psi_k\right) + \ddot{y}^L_{r,k} \cdot \sin\left(\psi_k\right)\right), \tag{7.61}$$

where $\ddot{x}^L_{r,k}$, and $\ddot{y}^L_{r,k}$ are the outputs of the $\vec{x}^L$ and $\vec{y}^L$ controllers, respectively, and $\phi_{r,k}$ and $\theta_{r,k}$ are the reference Euler angles for the attitude controller. This is implemented in the PyBullet simulation. This transformation was necessary because the planar model used in the design of the controllers did not take into account the yaw angle of the UAV. By rotating the outputs of the $\vec{x}^L$ and $\vec{y}^L$ controllers, the yaw angle was accounted for, allowing the UAV to accurately track the reference trajectory in 3D space.

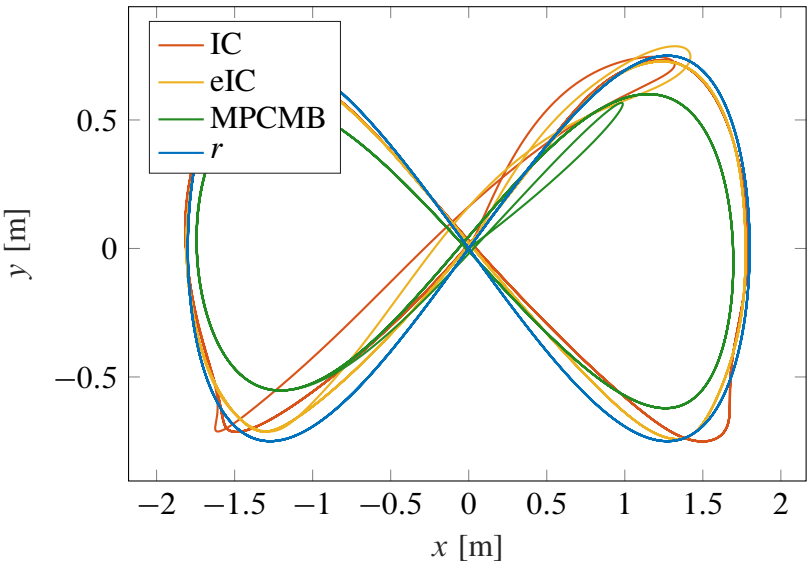
Since the results of the standard version are similar to those of the planar model and the fast ellipse reference provides very similar results to the fast lemniscate, only the case of the fast lemniscate will be presented in detail.

Figure 7.32 shows the path of the UAV in 3D and in the x^L-y^L plane. It can be seen that the IC has difficulty following the trajectory in similar areas as with the planar model, while the MPCMB significantly shortens the reference trajectory.

According to Figures 7.33 to 7.35 for the individual axes, it was most difficult to control the UAV in the $\vec{z}^L$-axis due to the high-frequency reference. For both interpolating controllers, there were spikes in the value of the interpolating coefficient up to 0.8 when tracking the reference position. This led to overshoots of more than 0.3 m in the z^L position. On the x-axis, the control performance of all controllers was comparable, with the interpolating controllers able to reduce the control error earlier, but at the cost of a less smooth control signal. On the y-axis, however, only the MPCMB had overshoots up to 0.4 m.

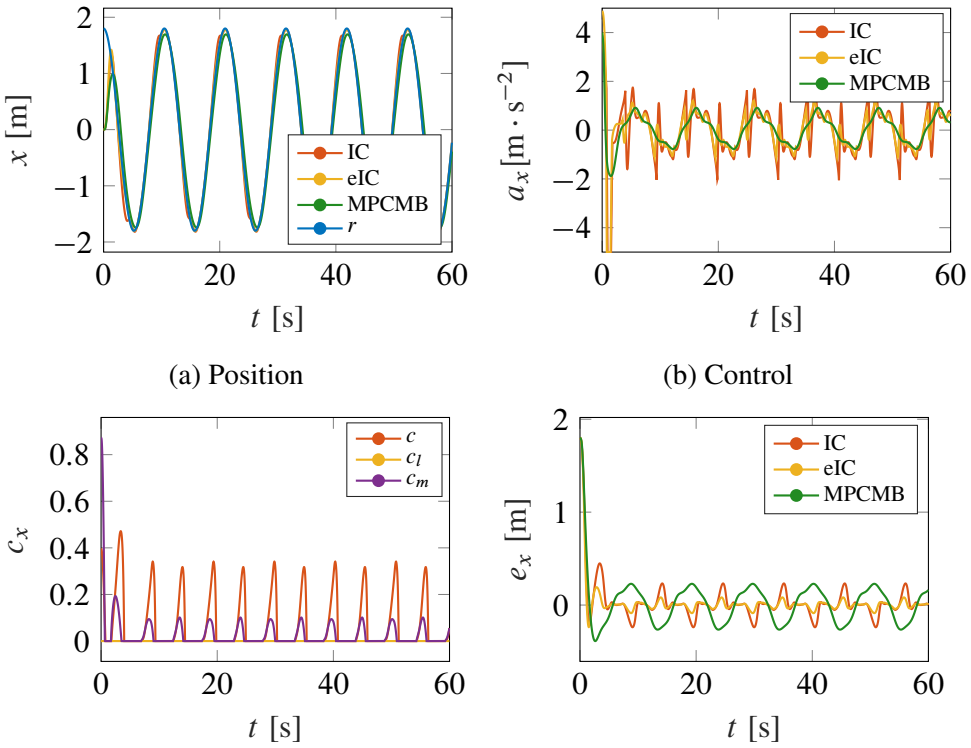


(a) 3D path



(b) x^L/y^L path

Figure 7.32: Path from tracking the high-frequency lemniscate trajectory in simulator



(a) Position

(b) Control

(c) Interpolating coefficient

(d) Control Error

Figure 7.33: Tracking fast lemniscate trajectory in the simulator in $\vec{x}^L$-axis

Table 7.21: Evaluation of the criterion, ISE, and energy consumption for the MPC, MPCMB IC, and eIC for the tracking of high-frequency lemniscate reference trajectory in simulator

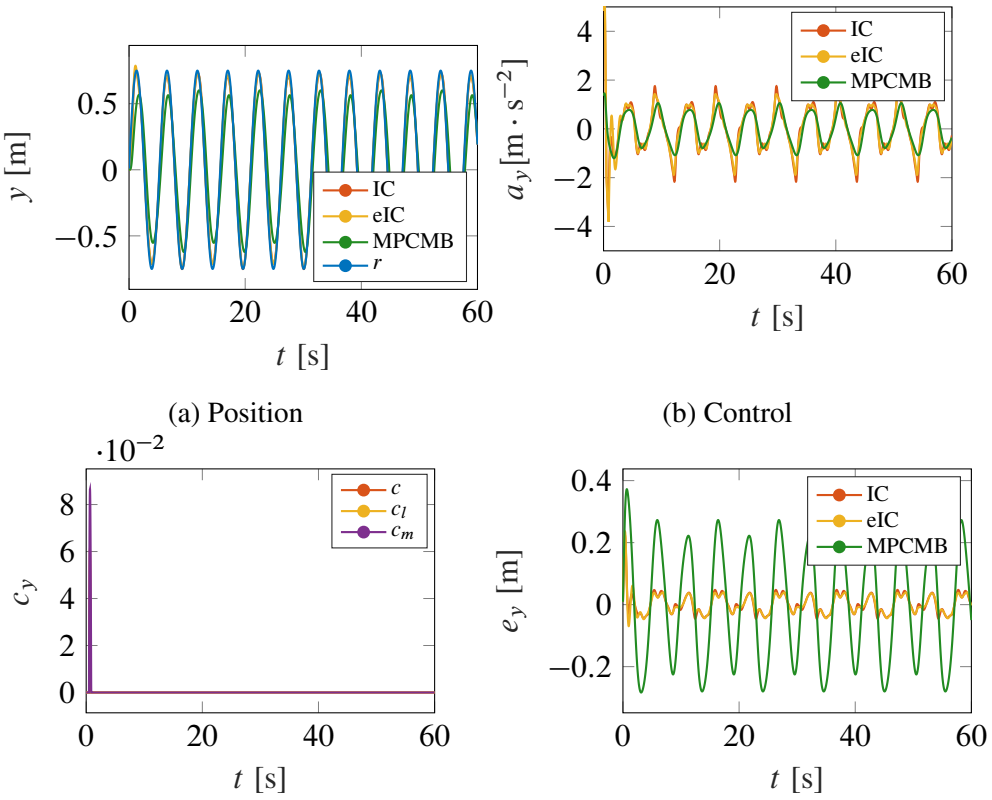
	J	%	ISE	%	E	%
MPCMB	52.8	-	6.40	-	2.79	-
eIC	73.8	+39.77	2.71	-57.66	3.77	+35.13
IC	192	+263.64	5.20	-18.75	4.68	+67.74

The MPCMB achieved the best optimality criterion value of the three controllers, as seen in Table 7.21. The IC was too aggressive, consuming excessive energy, and its optimality criterion value was 264% higher than the MPCMB. In contrast, the eIC performed 40% worse than the MPCMB in terms of the optimality criterion but had precise trajectory tracking. The eIC attained substantially lower ISE than the MPCMB while maintaining reasonable energy use.

A comparison of computational complexity was performed for tracking the high-frequency lemniscate trajectory, as the results were similar across references.

In the 3D PyBullet simulation, control calculations for three axes increased the computation time (Figure 7.36). The interpolation-based controllers exhibited periodic spikes in run time. As shown in Table 7.22, the MPCMB required the most time at 335 s total and 374 ms maximum per step. The eIC was significantly faster at 72 s total (82% less) and 68 ms max per step. The IC took 109 s total (86% less) and 51 ms max.

In summary, the simulation results demonstrate the efficiency advantages of the interpolation-based controllers over MPCMB. The eIC provided the best trade-off, tracking the aggressive trajectory at over 80% lower computational cost.



(a) Position

(b) Control

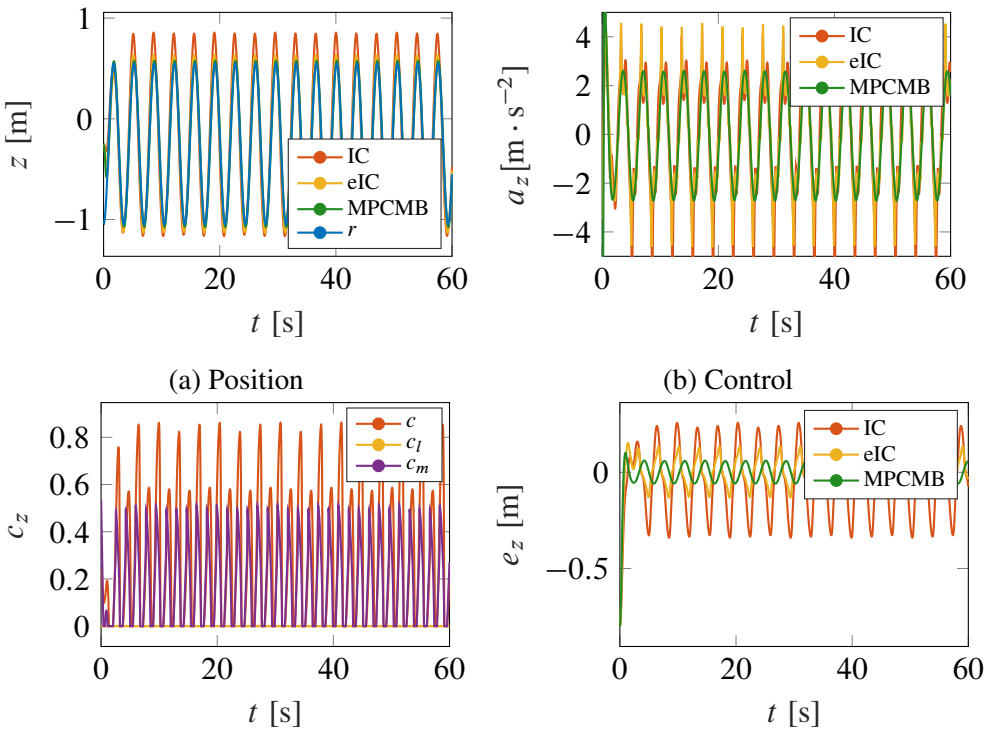
(c) Interpolating coefficient

(d) Control Error

Figure 7.34: Tracking fast lemniscate trajectory in the simulator in $\vec{y}^L$-axis

Table 7.22: The time demands for IC, eIC, MPC and MPCMB for the tracking of high-frequency lemniscate reference trajectory in simulator

	t [s]	%	$t_{\max}$ [ms]	%
MPCMB	335	-	374	-
eIC	72	-78.51	68	-81.82
IC	109	-67.46	51	-86.36



(a) Position (b) Control

(c) Interpolating coefficient (d) Control error

Figure 7.35: Tracking fast lemniscate trajectory in the simulator in $\vec{z}^L$-axis

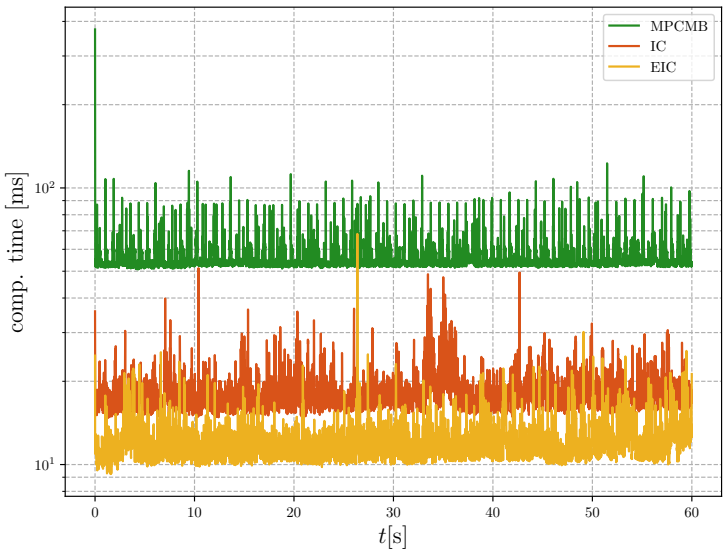


Figure 7.36: Computational time for tracking the fast lemniscate trajectory in simulator

7.2.2.6 Trajectory Tracking with Crazyflie UAV in the Laboratory

Laboratory tests were conducted using Crazyflie 2.0 UAVs equipped with the Lighthouse positioning deck[3], which can determine the full pose of the UAV with high accuracy using the HTC SteamVR Base Station 2.0. In the lab flight arena, four Base Stations were evenly distributed across the ceiling of the room. An advantage of the system is that Crazyflie determines its position based on the stations, eliminating the need to close the information loop as required by other systems such as motion capture systems.

The Crazyflie was controlled directly from a laptop, where the control actions were calculated. The control signals were sent wirelessly using a Crazyradio PA USB radio dongle. As mentioned before, the Python library cflib was used to communicate with the drone.

During the lab tests, there was a problem with the computation time for MPCMB. The computation time was too high and exceeded the expected time $T_s = 0.01$s several times, which resulted in MPCMB not being able to control the Crazyflie UAV in the $\vec{z}^L$-axis. As a result, the UAV kept oscillating and crashing. Two types of tests were done to solve this problem.

First, MPCMB was tested with interpolating controllers so that the controllers only generated roll and pitch angle references and sent direct requests for the position in the $\vec{z}^L$-axis using the `send_zdistance_setpoint` function.

Then, interpolating controllers were tested where direct requests for the collective thrust of the UAV were sent using the function `send_setpoint`. In this case, parallel computing was also tested using class Pool from the Python multiprocessing library[4]. Note that to compute the control input, estimates of the position and velocity in the local frame and yaw angle were sent asynchronously from the UAV to the laptop.

In the lab tests, it is important to note that the reference values were adjusted for safety reasons. Specifically, the amplitude in the $\vec{x}^L$-axis was reduced from 1.8 m to 1.5 m and the amplitude in the $\vec{z}^L$-axis was reduced from 1.0 m to 0.7 m, while ω_s was kept at 0.6 for both the $\vec{x}^L$ and $\vec{y}^L$ axes, but was reduced to 0.3 for the $\vec{z}^L$-axis. The resulting reference for lemniscate was given by

$$x^L_{r,k} = 1.5 \cos\left(0.6 \cdot k \cdot T_s\right), \tag{7.62}$$

$$y^L_{r,k} = \frac{1.5}{2} \sin\left(0.6 \cdot 2 \cdot k \cdot T_s\right), \tag{7.63}$$

$$z^L_{r,k} = -0.7 \cdot 0.8 \cos\left(3 \cdot 0.3 \cdot k \cdot T_s\right) + 0.7 \tag{7.64}$$

and for the elliptical reference by

$$x^L_{r,k} = 1.5 \cos\left(0.6 \cdot k \cdot T_s\right), \tag{7.65}$$

$$y^L_{r,k} = 1.0 \sin\left(0.6 \cdot k \cdot T_s\right), \tag{7.66}$$

$$z^L_{r,k} = -0.7 \cdot 0.8 \cos\left(3 \cdot 0.3 \cdot k \cdot T_s\right) + 0.7. \tag{7.67}$$

It is also worth noting that the lemniscate and ellipse trajectories produced similar results to the previous examples. Therefore, only the lemniscate trajectory will be presented.

Since the controllers only dealt with Crazyflie's motion in the $\vec{x}^L$ and $\vec{y}^L$ axes, the plots presented here only show the performance of the controllers in these axes. In the lab, the interpolation controllers rapidly pursued the lemniscate reference (Figure 7.37). In contrast,

[3]LH deck – `https://www.bitcraze.io/products/lighthouse-positioning-deck/`

[4]Pool class for multiprocessing – `https://docs.python.org/3/library/multiprocessing.html#multiprocessing.pool.Pool`

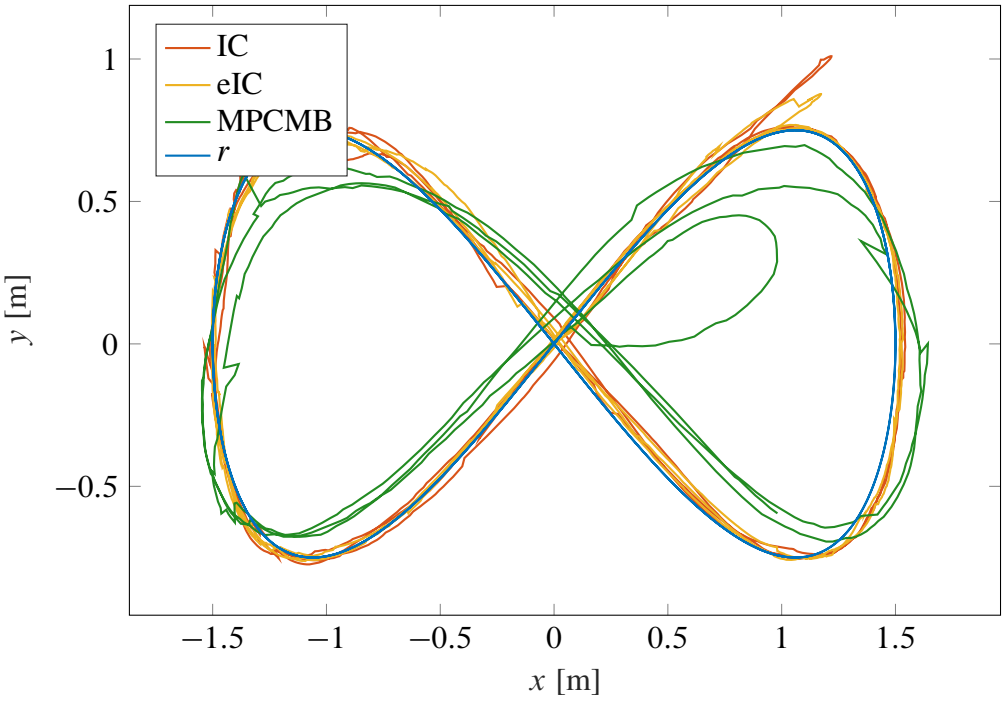


Figure 7.37: Path from tracking the lemniscate trajectory with Crazyflie UAV in laboratory

Table 7.23: Evaluation of the criterion, ISE, and energy consumption for the MPCMB IC, and eIC for the tracking of lemniscate reference trajectory in x-y plane in the laboratory experiment

	J	%	ISE	%	E	%
MPCMB	9.67	-	3.85	-	0.42	-
eIC	10.70	+10.65	1.42	-63.12	1.09	+161.39
IC	11.10	+14.79	1.39	-63.90	1.17	+180.58

the MPCMB followed the path slowly, resulting in a smaller path compared to the reference lemniscate.

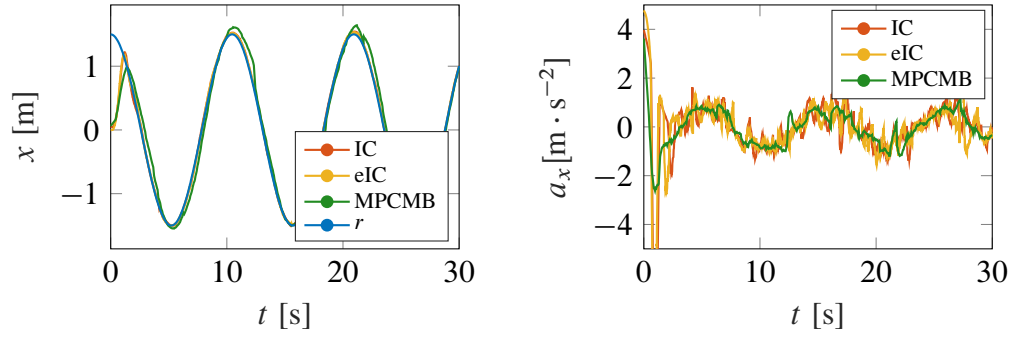
For both axes, the MPCMB oscillated in error up to 0.5 m (see Figures 7.38 and 7.39), but had smoother control without spikes. The interpolating coefficients in Figures 7.38c and 7.39c have high values at the beginning of the flight, with several smaller spikes during the measurement.

The MPCMB achieved the lowest cost, while the interpolating controllers had 10–15% higher values (Table 7.23). However, both interpolating controllers tracked the trajectory more accurately according to the ISE, but at significantly higher energy consumption.

In the lab tests, the number of computed control actions N_s and the average computation time $\bar{t}$ [ms] were measured. There was a more obvious difference between the interpolation controllers (Figure 7.40). Note that the previous tests were performed on a desktop PC, while the lab test was performed on a laptop.

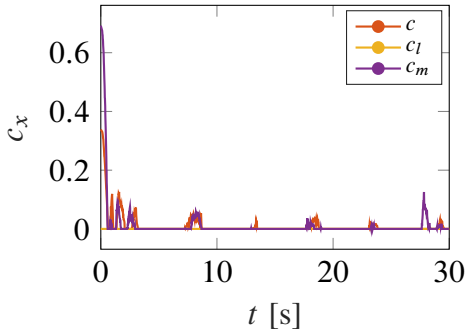
As seen in Table 7.24, the eIC computed 2658 samples in 10ms on average time (357% more than MPCMB). The IC computed 1407 samples in 21ms on average (142% more). Maximum computation times were 118ms for eIC (73% less than MPCMB) and 120ms for IC (72% less).

In summary, the MPCMB optimized the cost function better in the laboratory setting. However, the interpolating controllers provided more accurate trajectory tracking based on ISE.

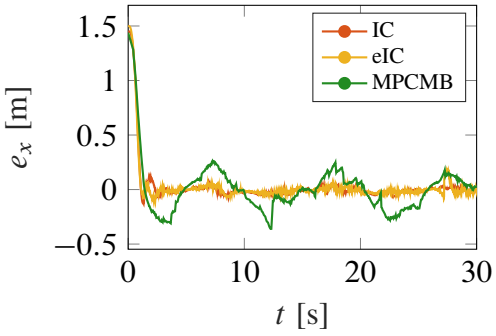


(a) Position

(b) Control

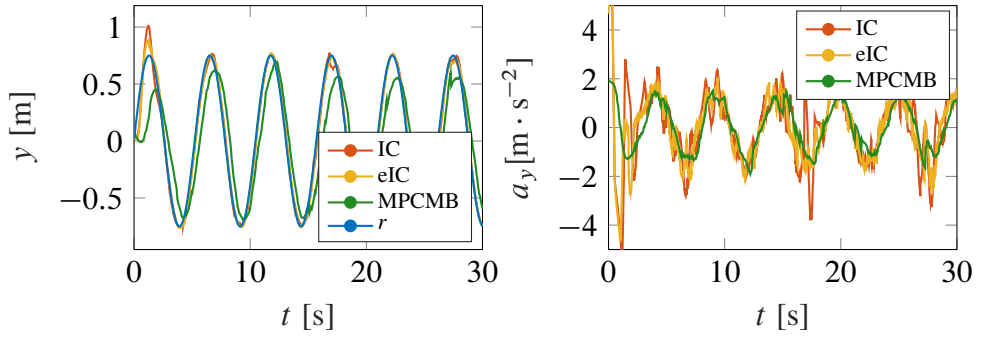


(c) Interpolating coefficient



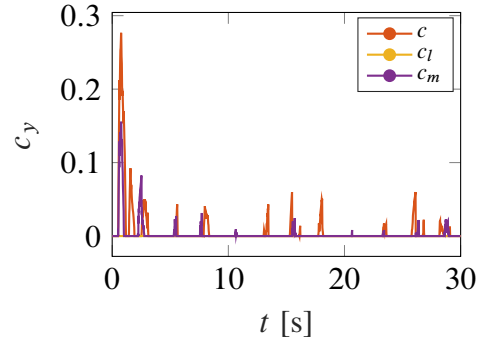
(d) Control error

Figure 7.38: Tracking the lemniscate trajectory with Crazyflie UAV in the laboratory in $\vec{x}^L$-axis

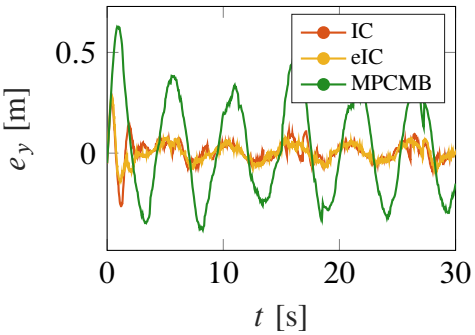


(a) Position

(b) Control



(c) Interpolating coefficient



(d) Control Error

Figure 7.39: Tracking the lemniscate trajectory with Crazyflie UAV in the laboratory in $\vec{y}^L$-axis

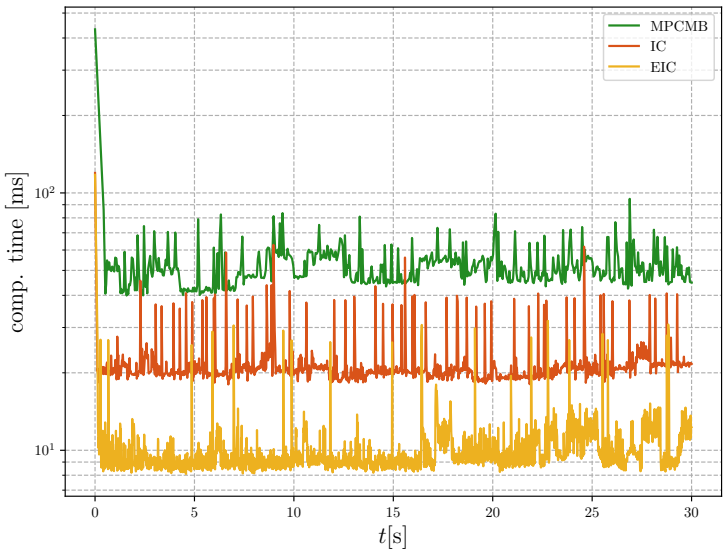


Figure 7.40: Computational time for tracking the fast lemniscate trajectory with Crazyflie UAV in laboratory

Table 7.24: The time demands for IC, eIC and MPCMB for the tracking of lemniscate reference trajectory in x-y plane in the laboratory experiment

	t [s]	%	N_s	%	$\bar{t}$ [ms]	%	$t_{\max}$ [ms]	%
MPCMB	30	-	582	-	51	-	432	-
eIC	26	-13.33	2658	+356.70	10	-80.39	118	-72.69
IC	30	+0.00	1407	+141.75	21	-58.82	120	-72.22

Computationally, the interpolating controller was much more efficient, especially the eIC. Due to the long computation time, MPCMB could not be used for altitude control using the total thrust of the UAV.

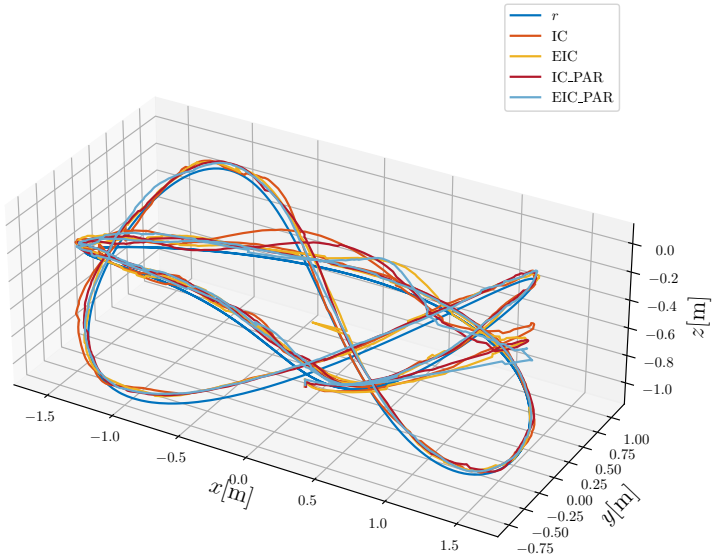
The laboratory testing of the interpolation-based controllers included the testing of the controllers for the $\vec{z}^L$-axis. The tests also investigated the effect of parallelizing the computation of control actions using a Pool from the multiprocessing library. The `send_setpoint` function was used to control the Crazyflie UAV.

The recorded UAV path is very similar for all controllers (Figure 7.41). There are minor differences between the standard and parallelized versions of the controllers. In particular, Figure 7.41b shows that both versions of the IC differ, but interestingly, in different areas.

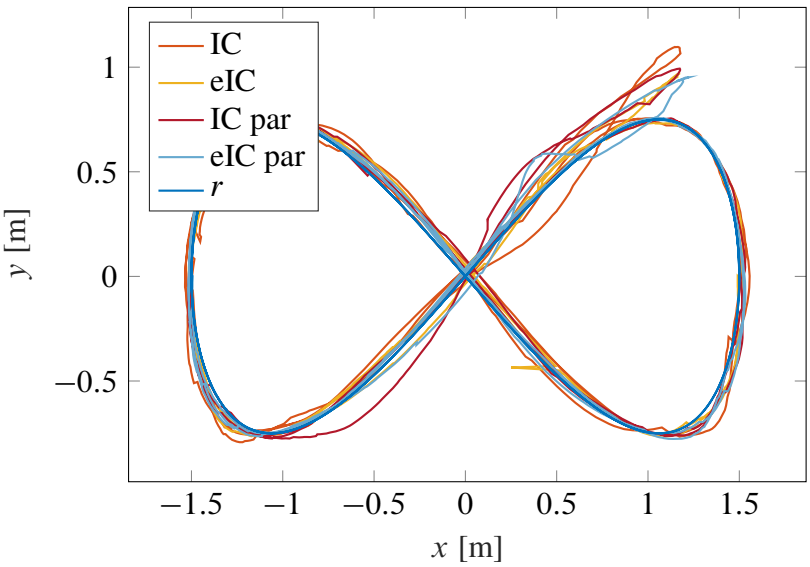
In addition, it can be observed that the position of the standard eIC changes abruptly at one point. Such a sudden change can be explained by a poor estimate of the UAV's position. However, all controllers successfully track the reference lemniscate trajectory.

Figures 7.42a, 7.43a and 7.44a for each axis show that although the differences between the controllers were small, the parallel eIC performed the smoothest control. In contrast, the non-parallel IC had the largest overshoots in all axes.

In the $\vec{z}^L$-axis, especially for the control error, a slight offset in position tracking can be seen, which may be due to the inaccuracy of the model, especially the parameters that affect the thrust-

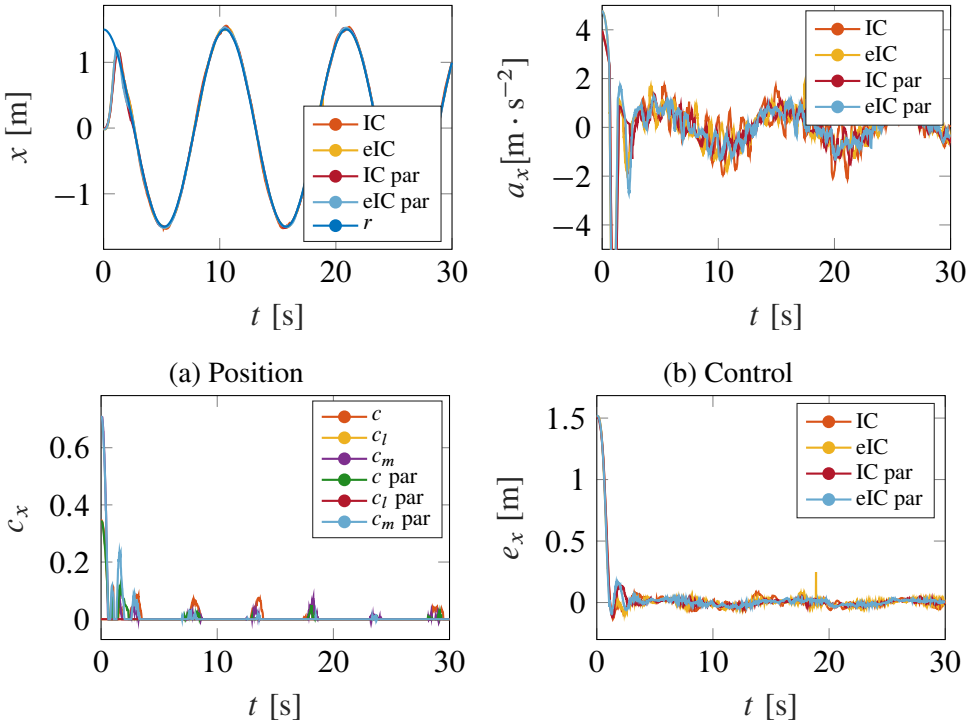


(a) 3D path



(b) x^L/y^L path

Figure 7.41: Path from tracking using the IC and eIC and their versions with parallel computing the fast figure8 trajectory in the laboratory experiment



(a) Position (b) Control

(c) Interpolating coefficient (d) Control error

Figure 7.42: Tracking the lemniscate trajectory with Crazyflie UAV in the laboratory in $\vec{x}^L$-axis fusing the IC and eIC and their versions with parallel computing

Table 7.25: Evaluation of the criterion, ISE, and energy consumption for the IC and eIC for the tracking of lemniscate reference trajectory in the laboratory experiment

	J	%	ISE	%	E	%
eIC par	20.2	-	1.56	-	1.62	-
IC par	21.1	+4.46	1.61	+3.21	1.65	+1.85
eIC	19.9	-1.49	1.57	+0.64	1.68	+3.70
IC	24.7	+22.28	1.73	+10.90	2.09	+29.01

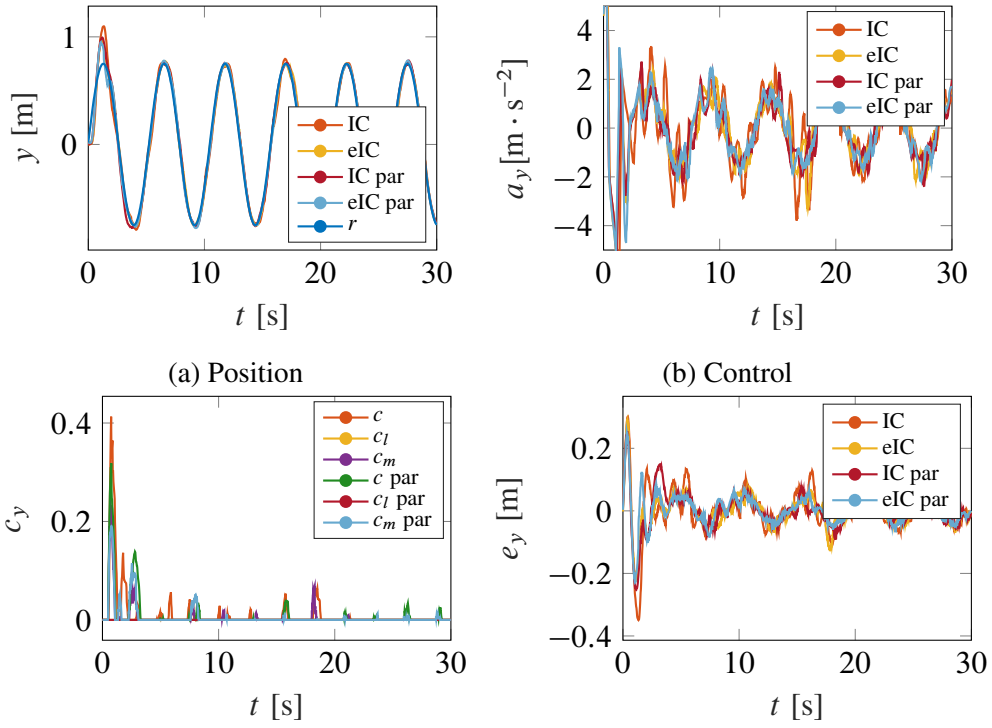
to-weight ratio of the UAV. Also, a small spike in the control error can be observed in eIC in the $\vec{x}^L$-axis, which is likely caused by a position estimation error.

The performance evaluation in Table 7.25 shows that both eICs achieved very similar results. However, the standard eIC achieved the best result in the optimality criterion, even though it consumed more energy and did not follow the trajectory according to ISE as precisely as the parallel eIC. This paradox is caused by the values of the weighting matrices and the small differences in the behavior of the two controllers.

Interestingly, the parallel IC was only 4.5% worse than the parallel eIC, which is even more intriguing since the standard IC is 22.3% worse.

The MPCMB could not stabilize the UAV and is excluded from the testing of computational demands. For comparison, the IC and eIC were tested in parallelized versions.

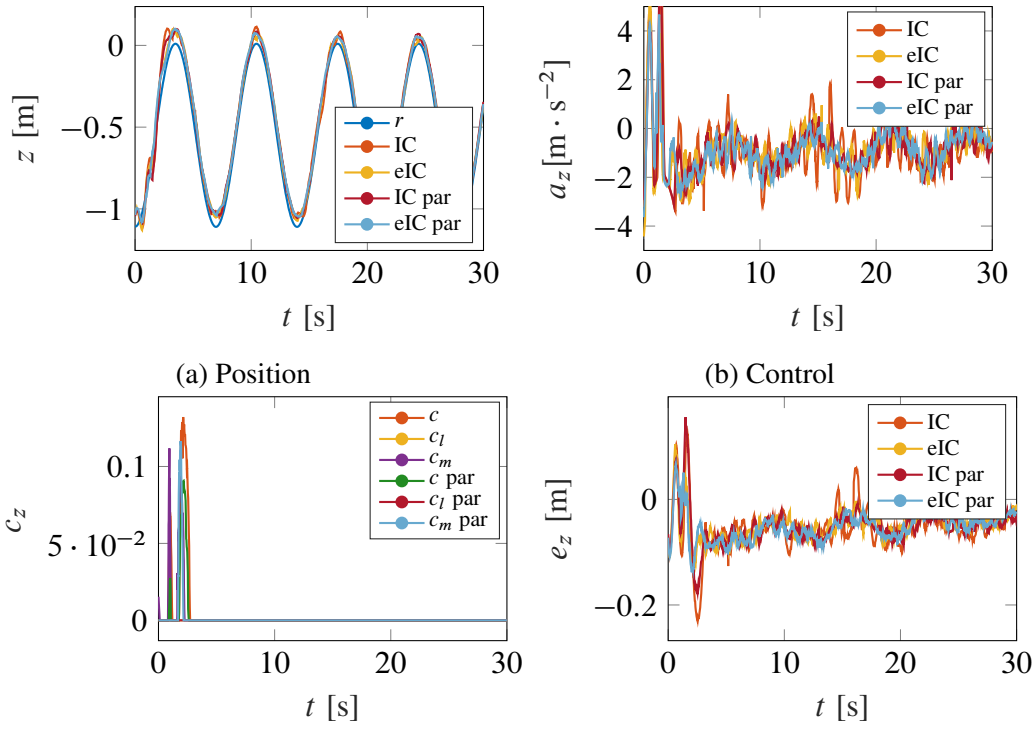
Figure 7.45 shows that the standard eIC was the fastest, but the parallel eIC was comparable. The standard eIC computed 2106 samples (+27.8% more than parallel) but achieved significantly



(a) Position (b) Control

(c) Interpolating coefficient (d) Control error

Figure 7.43: Tracking the lemniscate trajectory with Crazyflie UAV in the laboratory in $\vec{y}^L$-axis fusing the IC and eIC and their versions with parallel computing



(a) Position (b) Control

(c) Interpolating coefficient (d) Control error

Figure 7.44: Tracking the lemniscate trajectory with Crazyflie UAV in the laboratory in $\vec{z}^L$-axis fusing the IC and eIC and their versions with parallel computing

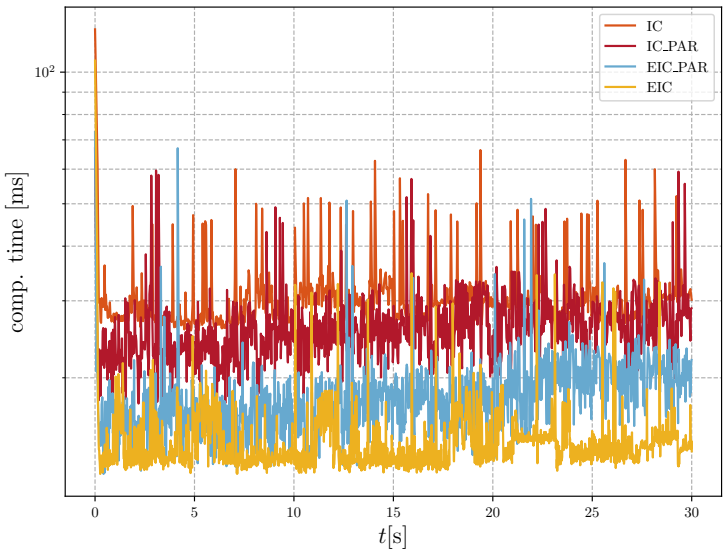


Figure 7.45: Computational time for tracking the high-frequency lemniscate trajectory with Crazyflie UAV in laboratory

Table 7.26: The time demands for the IC and eIC and their versions with parallel computing for the tracking of lemniscate reference trajectory in the laboratory experiment

	t [s]	%	N_s	%	$\bar{t}$ [ms]	%	$t_{\max}$ [ms]	%
eIC par	29	-	1648	-	18	-	73	-
IC par	30	+3.45	1153	-30.04	26	+44.44	67	-8.22
eIC	30	+3.45	2106	+27.79	14	-22.22	106	+45.21
IC	30	+3.45	993	-39.75	30	+66.67	125	+71.23

lower average times (see Table 7.26). Interestingly, parallelization decreased the times for IC but slightly increased them for eIC. However, the parallel versions had lower maximum times, typically on the first step (Figure 7.45).

The standard eIC was more efficient than its parallelized version as it computed more samples in less time. However, parallelization caused a decrease in the maximum time. In contrast, parallelization improved the computational performance of the IC by all measures.

7.3 Conclusions and Future Work

In this chapter, interpolation-based controllers have been designed that are capable of both setpoint control and trajectory tracking.

First, VC, which was previously used as a low-gain controller in the standard interpolation control, was modified for setpoint control. A simple modification of the VC was presented, where the control at the setpoint is set to zero and the rest of the control law is adjusted accordingly. However, no way was found to modify the VC for trajectory tracking. Therefore, the LQR was used as a low-gain controller in the interpolation-based controller for trajectory tracking.

Then, an implicit interpolating control method was proposed for setpoint control and trajectory control. The design for setpoint control exploited the similarity with VC. In the interpolation control, the LP is solved to find the optimal interpolation coefficient. In the LP, the inequality constraint containing the invariant set $\boldsymbol{\Omega}^h$ was modified by shifting it to the setpoint coordinates.

It was also found that the modified LP can be used for trajectory tracking if the invariant set is shifted to the coordinates of the current first point of the reference trajectory and the entire trajectory is reflected only within the controllers, between which interpolation is performed according to the interpolation coefficient.

In addition, the possibility of controlling to the setpoint was investigated for the explicit IC. It was found to geometrically shift the invariant set in the IC and modify the simplices of the explicit IC to follow the shifted invariant set. This modification can also be used for trajectory tracking similar to the implicit IC by reflecting only the first point of the trajectory in the LP.

The comparison was performed first for a simple 2nd order LTI system and then for a planar UAV model. The experiments with the 2nd order LTI system were performed in MATLAB. The controllers designed for the planar model were tested in a discrete simulation with the mathematical model, the simulation environment based on PyBullet, and in the laboratory on the Crazyflie UAV. The performance of the controllers was compared in terms of both control quality and computational complexity.

One measure of control quality was the value of the optimality criterion given by the OCP (Equation (4.1)). Another measure was the integral square error (Equation (7.32)), which indicates how accurately a reference is followed in the case of setpoint control or trajectory tracking. The final measure of control quality was energy consumption (Equation (7.33)), which shows how aggressive the individual controllers are.

In the case of the explicit control laws, their construction time, the number of regions, and the total computation time of the actions were compared in terms of computational complexity. In the case of implicit controllers, the total time to compute control actions, the maximum time to compute control actions, and in the case of the laboratory experiment, the number of computed control actions were measured.

Predictive controllers generally achieved better results, especially in terms of the optimality criterion. However, interpolation-based controllers with their computational efficiency can provide comparable performance and serve as a good alternative to predictive controllers, especially on platforms with limited computing power.

The experimental results showed that in particular, the eIC is a promising interpolation-based controller for trajectory tracking. It demonstrated faster computation times and higher accuracy in tracking the reference trajectory compared to IC. The parallel versions of eIC and IC also showed further reductions in computation time. These results suggest that interpolation-based controllers, particularly eIC and its parallel version, are viable options for trajectory tracking in real-time applications.

However, it is important to note that the performance of the interpolation-based controllers may be affected by the complexity and dynamics of the system. Further research is needed to investigate their performance in more complex and dynamic environments and to evaluate their scalability to larger systems. In addition, the implementation of the interpolation-based controllers may require careful tuning of the low-gain and additional middle controllers to achieve optimal performance.

There are several potential areas of research for the development of interpolation-based controllers for trajectory tracking. One promising direction is the use of IC with ellipsoidal invariant

sets [86], which do not require the use of LP solvers during the computation of the control action. This could be particularly useful for platforms where LP solvers are difficult to implement and could also reduce computational requirements. However, adapting the IC for tracking with elliptic sets has not yet been tested and would require further investigation.

In addition, the development of adaptive interpolation-based controllers that can adjust their interpolation parameters in real-time based on system dynamics and uncertainties could improve the robustness and adaptability of the controllers, especially in situations where system parameters are not well known.

Finally, it would be interesting to explore the possibility of implementing IC directly on the autopilot of the Crazyflie UAV. This would require careful consideration of the computational requirements and limitations of the platform but could potentially lead to more efficient and effective trajectory tracking.

In summary, several potential research areas for the development of interpolation-based controllers for trajectory tracking include the use of elliptic invariant sets, the design of LQ controllers for tracking, the analysis of additional controllers, the development of adaptive controllers, and the implementation of IC on specific platforms. These research areas could help further advance the field of trajectory tracking and improve the performance of interpolation-based controllers in various applications.

8 Conclusion

The goal of this thesis was to design a trajectory planning and tracking system for unmanned aerial vehicles (UAVs). The trajectory planning system is essential for any application where the UAV moves autonomously. A quadrotor helicopter was chosen as the type of UAV considered due to its widespread use. The initial chapters cover the UAV and the basics of trajectory tracking and planning. The UAV is presented along with a mathematical model of its behavior. Various methods for path and trajectory planning are then discussed, with emphasis on trajectory planning as an Optimal Control Problem (OCP). A Pseudospectral Method (PSM) is presented as a numerical method for solving this OCP. Furthermore, this thesis describes the trajectory tracking problem and introduces Model Predictive Control (MPC) as a standard method, as well as Interpolating Control (IC) as a promising alternative.

The primary goal of the thesis was divided into two subtasks. Based on the considerations that emerged in Chapter 3, **Goal 1** of the PSM proposal for UAV trajectory planning was stated in Chapter 5 as:

(a) *Acquiring a better initial guess using graph-based path planning methods and splitting the problem according to the initial path into several smaller interconnected segments.*

(b) Analysing a suitable version of the initial state and control trajectory guess based on UAV dynamics.

Among the presented methods, I chose LT* for the construction of the initial guess in Chapter 6 because, although the path is linked to the grid, it allows path planning even between directly visible nodes. The resulting path is thus constructed from only a few points, and thus more closely resembles reality.

The path is just a sequential list of points to be flown by the UAV. In order to use the path for the initial trajectory guess, I first had to calculate the time in which the UAV would pass the points. I calculated the time frame guess optimistically based on the maximum velocity of the UAV. Once I had the time frame, I could place the waypoints on the curve. From my experiments, I found that the cubic spline was the most stable for interlacing.

In addition to PSM, which approximates the OCP by only one polynomial for the entire trajectory, I implemented the Pseudospectral Elements Method (PSEM, also referred to as hp-PSM in the literature), which can approximate the OCP by several interleaved polynomials, each of which is called a segment.

Thus, for the initial PSEM guess, I took advantage of having multiple waypoints and sampled the interleaved curve as if there was always a trajectory segment between two waypoints. To test whether this solution was advantageous, I also used the single-segment initial guess constructed for the PSM. From the comparison for PSEM, I observed that when the multi-segment guess was used, the trajectory planning algorithm mostly found the trajectory in fewer iterations,

and the solution achieved better values of the optimality criterion and a smaller absolute error. However, the time to obtain a solution was consistently better only for the initial guess where the position was approximated.

From the curve describing the position as a function of time, I could derive the velocity in time according to the position derivative. Then I guessed the acceleration curve. For the orientation trajectory, I used the description of the relations for the SE(3) controller in Appendix C. The orientation was calculated based on the required force, which depends on the acceleration. From the trajectory of the quaternion describing the orientation, I derived the angular rate from the relations. Then I calculated the required control in the form of collective thrust and torque from the state trajectory elements determined above.

I tested the initial guesses at different levels by replacing some of the state and control components with values given by interpolation between boundary conditions. The values varied quite a bit between levels. However, when I used a simple guess without path information based only on boundary conditions, the trajectory was not found for the one and two obstacle scenarios, respectively. PSEM with multi-segment initial position guess gave stable and interesting results, but this solution often had collision trajectories. At the end of Chapter 6, I suggested that the trajectory finding condition for UAV trajectory planning could be improved by testing the collision rate and possibly violating other constraints.

Chapter 4 introduced the IC, and Chapter 5 revealed that the current IC cannot track trajectories. **Goal 2** was to design a trajectory tracking method based on IC in the following steps:

(a) *Adapting the current IC for the stabilization of the system to the origin to include trajectory tracking features.*

(b) Considering the future system development of the reference trajectory directly in the control law and supplement the IC with trajectory tracking features.

(c) Testing the developed controller in a laboratory setting, where its performance will be evaluated by tracking a trajectory with a UAV.

In Chapter 7, I described modifications of the IC to drive to a setpoint. This adjustment is made by moving the center of the high-gain controller set to the setpoint coordinates. Similarly, the region of the set can be adjusted in a linear program that is solved within the implicit IC to obtain the interpolation coefficient. I also described how to modify the explicit IC for a given setpoint. I described how the same modification can be used for trajectory tracking by performing the same operation on the current point of the reference trajectory instead of the setpoint. I also showed how to modify the extended IC (eIC), which uses interpolation between the three controllers for the setpoint control problem.

For IC, it is not enough to consider only the setpoint or the trajectory when obtaining the interpolation coefficient, but it is necessary that the control laws between which the interpolation is performed also provide this type of control. For this reason, I used the Linear Quadratic Regulator described in Chapter 4 for both setpoint control and trajectory tracking.

First, I compared the performance of IC for both setpoint control and then trajectory tracking using a simple example with a second-order linear time-invariant system. Next, I proposed IC-based trajectory tracking for UAVs. I also designed an eIC that allows trajectory tracking without the need to clip the reference trajectory, since the low-gain and medium-gain controllers only control to the current point of the reference trajectory as in the setpoint control, and

the entire trajectory is only reflected by the high-gain controller. The UAV Trajectory Tracking IC was tested in the control of a simple mathematical model, in a simulation environment based on PyBullet, and in the laboratory on the Crazyflie UAV.

Interpolation-based controllers were shown to provide comparable performance and to be a good alternative to predictive controllers, especially on platforms with limited computing power, thanks to their computational efficiency.

8.1 Challenges and Future Work

Both trajectory planning using PSM and tracking using IC offers the potential for improvement or extension of both methods.

For planning, a detailed analysis of a sufficient number of collocation points of the initial guess or different approaches to its segmentation could be performed. Also, as mentioned above, the trajectory condition could be modified to account for possible collisions outside the collocation points and possible constraint violations. Also, the convergence in planning could be improved by considering only the space and its constraints around the optimal path within the OCP.

IC-based trajectory tracking also provides several challenges. For example, standard IC can be modified for elliptic invariant sets and does not need to solve a linear program to obtain the interpolation coefficient, which could open up IC-based trajectory tracking to other platforms. Furthermore, it would be interesting to implement an IC directly for the autopilot of the Crazyflie microUAV or to design an IC to enable trajectory tracking of large UAV swarms.

Publications

Trajectory Planning

Z. Bouček, P. Neduchal and M. Flídr. DronePort: Smart Drone Battery Management System. In *Proc. of the 6th International Conference on Interactive Collaborative Robotics*, ICR 2021, St. Peterburg, Russia, September 27 – 30 2021.

Z. Bouček. Drone Trajectory Planning Considering Battery Capacity. In *Studentská vědecká konference: magisterské a doktorské studijní programy, sborník rozšířených abstraktů*, SVK21, Pilsen, Czech Republic, June 10 2021.

Z. Bouček, Optimální trajektorie kvadrotorové helikoptéry, In *Studentská vědecká konference: magisterské a doktorské studijní programy, sborník rozšířených abstraktů*, SVK18, Pilsen, Czech Republic, 2018.

Interpolating Control

Z. Bouček and M. Flídr, Interpolating Control Based Trajectory Tracking. In *Proc. of the 16th International Conference on Control, Automation, Robotics and Vision* ICARCV 2020, Shenzen, China, December 13 – 15 2020.

Z. Bouček and M. Flídr. Modification of Explicit Interpolating Controller for Control Problem with Constant Setpoint. In *Proc. of the 15th European Workshop on Advanced Control and Diagnosis*, ACD 2019, Bologna, Italy, November 21 – 22 2019.

Z. Bouček and M. Flídr. Explicit Interpolating Control of Unmaned Aerial Vehicle. In *Proc. of the 24th International Conference on Methods and Models in Automation and Robotics*, MMAR19, Międzyzdroje, Poland, August 26 – 29 2019.

Z. Bouček, Kvalita regulace interpolačního řízení, In *Studentská vědecká konference: magisterské a doktorské studijní programy, sborník rozšířených abstraktů*, SVK19, Pilsen, Czech Republic, 2019.

Others

O. Severa, Z. Bouček, P. Neduchal, L. Bláha, T. Myslivec, and M. Flídr. Droneport: From Concept To Simulation. In *System Engineering for Constrained Embedded Systems (DroneSE and RAPIDO)*, Association for Computing Machinery, New York, USA, 2022.

Z. Bouček, O. Straka, and M. Flídr. Attitude Estimation for Quadrotor UAV by Unscented Kalman Filter. In *Proc. of the 14th European Workshop on Advanced Control and Diagnosis*, ACD 2017, Bucharest, Romania, November 16 – 17 2017.

Z. Bouček and M. Flídr, Impact of multiple accelerometer IMU employment on the orientation estimate quality, In *Proc. of the 17th International Carpathian Control Conference*, ICCC16, Tatranská Lomnica, Slovakia, May 29 – June 1 2016.

Z. Bouček, Návrh řízení kvadrotorové helikoptéry, In *Studentská vědecká konference: magisterské a doktorské studijní programy, sborník rozšířených abstraktů*, SVK15, Pilsen, Czech Republic, 2015.

Bibliography

[1] B.D.O. Anderson and J.B. Moore. *Optimal Control: Linear Quadratic Methods*. Dover Books on Engineering. Dover Publications, 2007.

[2] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[3] M. Athans and P.L. Falb. *Optimal Control: An Introduction to the Theory and Its Applications*. Lincoln Laboratory publications. McGraw-Hill, 1966.

[4] Tomas Baca, Daniel Hert, Giuseppe Loianno, Martin Saska, and Vijay Kumar. Model Predictive Trajectory Tracking and Collision Avoidance for Reliable Outdoor Deployment of Unmanned Aerial Vehicles. *IEEE International Conference on Intelligent Robots and Systems*, pages 6753–6760, 2018.

[5] Tomas Baca, Giuseppe Loianno, and Martin Saska. Embedded model predictive control of unmanned micro aerial vehicles. In *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 992–997. IEEE, aug 2016.

[6] Tomas Baca, Petr Stepan, Vojtech Spurny, Daniel Hert, Robert Penicka, Martin Saska, Justin Thomas, Giuseppe Loianno, and Vijay Kumar. Autonomous landing on a moving vehicle with an unmanned aerial vehicle. *Journal of Field Robotics*, 36(5):874–891, 2019.

[7] Amin Basiri, Valerio Mariani, Giuseppe Silano, Muhammad Aatif, Luigi Iannelli, and Luigi Glielmo. A survey on the application of path-planning algorithms for multi-rotor UAVs in precision agriculture. *Journal of Navigation*, pages 1–20, 2022.

[8] John T Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming (Second edition)*. SIAM, 2010.

[9] Mahathi Bhargavapuri, Jay Patrikar, Soumya Ranjan Sahoo, and Mangal Kothari. A Low-Cost Tilt-Augmented Quadrotor Helicopter : Modeling and Control. *2018 International Conference on Unmanned Aircraft Systems, ICUAS 2018*, pages 186–194, 2018.

[10] Franco Blanchini and Stefano Miani. *Set-Theoretic Methods in Control*. Number 9783319179322 in Systems & Control: Foundations & Applications. Springer International Publishing, Cham, 2015.

[11] Gilbert A Bliss. *Lectures on the calculus of variations*. Chicago Univ. Press, Chicago, IL, 1946.

[12] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[13] S. Bouabdallah, a. Noth, and R. Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:1–6, 2004.

[14] Zdenek Boucek and Miroslav Flidr. Interpolating Control Based Trajectory Tracking *. In *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 701–706. IEEE, dec 2020.

[15] Zdeněk Bouček. Data and results from the thesis for UAV trajectory tracking, February 2024.

[16] Zdeněk Bouček. Singularity container and data from thesis work for UAV trajectory planning, February 2024.

[17] John P Boyd. *Chebyshev and Fourier Spectral Methods*. DOVER Publications, Inc., 2000.

[18] Thomas Braud and Nizar Ouarti. Constrained sigma points for attitude estimation. *2016 14th International Conference on Control, Automation, Robotics and Vision, ICARCV 2016*, 2016(November):13–15, 2017.

[19] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[20] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo — Optimization Modeling in Python*, volume 67 of *Springer Optimization and Its Applications*. Springer International Publishing, Cham, 2021.

[21] R. Cagienard, P. Grieder, E.C. Kerrigan, and M. Morari. Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6):563–570, jul 2007.

[22] Yong Bo Chen, Guan Chen Luo, Yue Song Mei, Jian Qiao Yu, and Xiao Long Su. UAV path planning using artificial potential field method updated by optimal control theory. *International Journal of Systems Science*, 47(6):1407–1420, 2016.

[23] Howie Choset. Robotic motion planning: Potential functions, 2010.

[24] Peter Corke. *Robotics, Vision and Control*, volume 73. Springer, Berlin, Heidelberg, 2011.

[25] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[26] C. R. Cutler and B. L. Ramaker. Dynamic matrix control- a computer control algorithm. *Joint Automatic Control Conference*, 17:72, 1980.

[27] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39(January):533–579, 2010.

[28] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[29] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, dec 1959.

[30] Carlos Bordons Eduardo F. Camacho. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 1999.

[31] Margarida Faria, Ivan Maza, and Antidio Viguria. Applying Frontier Cells Based Exploration and Lazy Theta* Path Planning over Single Grid-Based World Representation for Autonomous Inspection of Large 3D Structures with an UAS. *Journal of Intelligent & Robotic Systems*, 93(1-2):113–133, feb 2019.

[32] J. Förster. System identification of the crazyflie 2.0 nano quadrocopter. Bachelor's thesis, ETH Zurich, 2015.

[33] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, May 1989.

[34] C. Goerzen, Z. Kong, and B. Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance*, volume 57. Springer, 2010.

[35] Basile Graf. Quaternions and dynamics, 2008.

[36] Gene Grimm, Michael J. Messina, Sezai E. Tuna, and Andrew R. Teel. Nominally robust model predictive control with state constraints. *IEEE Transactions on Automatic Control*, 52(10):1856–1870, 2007.

[37] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[38] Per Olof Gutman and Michael Cwixel. Admissible Sets and Feedback Control for Discrete-Time Linear Dynamical Systems with Bounded Controls and States. *IEEE Transactions on Automatic Control*, 31(4):373–376, 1986.

[39] Tor Aksel N. Heirung, Joel A. Paulson, Jared O'Leary, and Ali Mesbah. Stochastic model predictive control – how does it work? *Computers and Chemical Engineering*, 114:158–170, 2018.

[40] D. M. Henderson. Shuttle Program. Euler angles, quaternions, and transformation matrices working relationships. Technical report, NASA Johnson Space Center, Houston, TX, United States, 1977.

[41] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013.

[42] David G. Hoag. Apollo Navigation, Guidence, and Control Systems: A Progress Report. Technical report, MIT Instrumentation Laboratory, 1969.

[43] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[44] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[45] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit-An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, may 2011.

[46] Haomiao Huang, Gabriel M. Hoffmann, Steven L. Waslander, and Claire J. Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In *2009 IEEE International Conference on Robotics and Automation*, pages 3277–3282. IEEE, may 2009.

[47] Wayne Johnson. *Helicopter Theory*. Dover Publications, New York, NY, USA, 1994.

[48] Mina Kamel, Javier Alonso-mora, Roland Siegwart, and Juan Nieto. Nonlinear Model Predictive Control for Multi-Micro Aerial Vehicle Robust Collision Avoidance. *CoRR*, abs/1703.0, 2017.

[49] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017.

[50] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *CoRR*, abs/1005.0416, 2010.

[51] George Em Karniadakis and Spencer Sherwin. Spectral/hp element methods for CFD. *Computers & Mathematics with Applications*, 38(11-12):284, 1999.

[52] Matthew P. Kelly. Transcription methods for trajectory optimization: a beginners tutorial, 2017.

[53] Eric C. Kerrigan and Jan M. Maciejowski. Feedback min-max model predictive control using a single linear program: robust stability and the explicit solution. *International Journal of Robust and Nonlinear Control*, 14(4):395–413, mar 2004.

[54] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505. Institute of Electrical and Electronics Engineers, 1985.

[55] Donald E Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, 2004.

[56] Gregor Klančar, Andrej Zdešar, Sašo Blažič, and Igor Škrjanc. *Wheeled Mobile Robotics*. Butterworth-Heinemann, 2017.

[57] Martin Klaučo and Michal Kvasnica. *MPC-Based Reference Governors*. Advances in Industrial Control. Springer International Publishing, Cham, 2019.

[58] Jack B. Kuipers. *Quaternions and Rotation Sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton University Press, New Jersey, 1999.

[59] Steven M. LaValle. Planning algorithms. *Planning Algorithms*, 9780521862:1–826, 2006.

[60] Hyeonbeom Lee and H. Jin Kim. Trajectory tracking control of multirotors from modelling to experiments: A survey. *International Journal of Control, Automation and Systems*, 15(1):281–292, 2017.

[61] J. Lofberg. Yalmip : a toolbox for modeling and optimization in matlab. In *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pages 284–289, 2004.

[62] Teppo Luukkonen. Modelling and Control of Quadcopter. *Journal of the American Society for Mass Spectrometry*, 22(7):1134–45, 2011.

[63] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. *IEEE Robotics & Automation Magazine*, 19(3):20–32, sep 2012.

[64] D. Limón Marruedo, T. Álamo, and E. F. Camacho. Input-to-state stable MPC for constrained discrete-time nonlinear systems with bounded additive uncertainties. *Proceedings of the IEEE Conference on Decision and Control*, 4(December):4619–4624, 2002.

[65] Jacob Mattingley and Stephen Boyd. CVXGEN: a code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, mar 2012.

[66] David Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[67] D.Q. Mayne, M.M. Seron, and S.V. Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224, feb 2005.

[68] Pedro Mercader, Daniel Rubin, Hoai Nam Nguyen, Alberto Bemporad, and Per Olof Gutman. Simple Interpolating Control*. *IFAC-PapersOnLine*, 51(25):42–47, 2018.

[69] Tiago Nascimento and Martin Saska. Embedded Fast Nonlinear Model Predictive Control for Micro Aerial Vehicles. *Journal of Intelligent & Robotic Systems*, 103(4):74, dec 2021.

[70] Alex Nash, Sven Koenig, and Craig Tovey. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 147–154. AAAI Press, 2010.

[71] H. N. Nguyen, P. O. Gutman, S. Olaru, and M. Hovd. Explicit constraint control based on interpolation techniques for time-varying and uncertain linear discrete-time systems. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 44(PART 1):5741–5746, 2011.

[72] H.N. Nguyen. *Constrained Control of Uncertain, Time-Varying, Discrete-Time Systems: An Interpolation-Based Approach*. Lecture Notes in Control and Information Sciences. Springer International Publishing, 2014.

[73] Huan Nguyen, Mina Kamel, Kostas Alexis, and Roland Siegwart. Model Predictive Control for Micro Aerial Vehicles: A Survey. In *2021 European Control Conference (ECC)*, pages 1556–1563. IEEE, jun 2021.

[74] Sammy Omari, Minh Duc Hua, Guillaume Ducard, and Tarek Hamel. Nonlinear control of VTOL UAVs incorporating flapping dynamics. *IEEE International Conference on Intelligent Robots and Systems*, pages 2419–2425, 2013.

[75] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[76] Hardik Parwana, Jay S. Patrikar, and Mangal Kothari. A novel fully quaternion based nonlinear attitude and position controller. In *2018 AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, Inc., 2018.

[77] Jean C. Pereira, Valter J.S. Leite, and Guilherme V. Raffo. An ellipsoidal-polytopic based approach for aggressive navigation using nonlinear model predictive control. In *2021 International Conference on Unmanned Aircraft Systems, ICUAS 2021*, pages 827–835. IEEE, jun 2021.

[78] Bert Pluymers. *Robust model based predictive control-an invariant set approach*. PhD thesis, KATHOLIEKE UNIVERSITEIT LEUVEN, 2006.

[79] L.S. Pontryagin. *Mathematical Theory of Optimal Processes*. Classics of Soviet Mathematics. Taylor & Francis, 1987.

[80] Quan Quan. *Introduction to Multicopter Design and Control*. Springer Singapore, Singapore, 2017.

[81] J. Richalet, A. Rault, J. L. Testud, and J. Papon. Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14:413–428, 1978.

[82] J. Richalet, A. Rault, J.L. Testud, and J. Papon. Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14(5):413 – 428, 1978.

[83] Charles Richter, Adam Bry, Nicholas Roy, Bing An, Tongjun Zhang, Chao Yuan, and Kun Cui. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. *Springer Tracts in Advanced Robotics*, 114(5):649–666, 2016.

[84] I. Michael Ross and Mark Karpenko. A review of pseudospectral optimal control: From theory to flight. *Annual Reviews in Control*, 36(2):182–197, 2012.

[85] Bartomeu Rubí, Ramon Pérez, and Bernardo Morcego. A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors. *Journal of Intelligent & Robotic Systems*, sep 2019.

[86] Daniel Rubin, Pedro Mercader, Per-Olof Gutman, Hoai-Nam Nguyen, and Alberto Bemporad. Interpolation based predictive control by ellipsoidal invariant sets. *IFAC Journal of Systems and Control*, 12:100084, 2020.

[87] Sheila Scialanga and Konstantinos Ampountolas. Interpolating Control Toolbox (ICT). In *2019 18th European Control Conference (ECC)*, pages 2510–2515. IEEE, jun 2019.

[88] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access*, 7:48572–48634, 2019.

[89] Dhwanil Shukla and Narayanan Komerath. Multirotor Drone Aerodynamic Interaction Investigation. *Drones*, 2(4):43, 2018.

[90] Jos F Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999.

[91] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, jan 2000.

[92] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, mar 2006.

[93] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of Robot 3D Path Planning Algorithms. *Journal of Control Science and Engineering*, 2016, 2016.

[94] Larry C. Young. Orthogonal collocation revisited. *Computer Methods in Applied Mechanics and Engineering*, 345:1033–1076, 3 2019.

[95] F. Benjamin Zhan and Chaeles E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.

[96] Han Ye Zhang, Wei Ming Lin, and Ai Xia Chen. Path planning for the mobile robot: A review. *Symmetry*, 10(10), 2018.

[97] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 528–535, 2016.

[98] Xiaodong Zhang, Xiaoli Li, Kang Wang, and Yanjun Lu. A survey of modelling and identification of quadrotor robot. *Abstract and Applied Analysis*, 2014, 2014.

[99] Boyu Zhou, Jie Pan, Fei Gao, and Shaojie Shen. RAPTOR: Robust and Perception-Aware Trajectory Replanning for Quadrotor Fast Flight. *IEEE Transactions on Robotics*, 37(6):1992–2009, 2021.

A Quaternions

In this appendix, some properties of quaternions will be discussed. Quaternion [35, 40] is a hyper-complex number with four components

$$\boldsymbol{q} = [q_w, \boldsymbol{q}_v^{\mathsf{T}}]^{\mathsf{T}}, \, \boldsymbol{q}_v = [q_x, \, q_y, \, q_z]^{\mathsf{T}}, \tag{A.1}$$

where q_w is a scalar part of the quaternion and $\boldsymbol{q}_v$ is composed of the imaginary part. The essential operation with the quaternions is a quaternion product (A.2). In attitude representation, it is important for the rotations of vectors (A.11) and also for a description of rotational dynamics (A.16). The quaternion product is not commutative, and it is defined as

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} p_w q_w - \boldsymbol{p}_v^{\mathsf{T}} \cdot \boldsymbol{q}_v \\ q_w \boldsymbol{p}_v + p_w \boldsymbol{q}_v + \boldsymbol{p}_v \times \boldsymbol{q}_v \end{bmatrix}, \tag{A.2}$$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are general quaternions and $\times$ is a cross-product. Also, it is defined in the form of a matrix multiplication

$$\boldsymbol{p} \otimes \boldsymbol{q} = \boldsymbol{\Omega}_4(\boldsymbol{p})\boldsymbol{q} = \begin{bmatrix} p_w & -\boldsymbol{p}_v{}^{\mathsf{T}} \\ \boldsymbol{p}_v & p_w \cdot \boldsymbol{I}_{3\times 3} + \boldsymbol{\Omega}_3(\boldsymbol{p}_v) \end{bmatrix} \begin{bmatrix} q_w \\ \boldsymbol{q}_v \end{bmatrix}, \tag{A.3}$$

where $\boldsymbol{I}_{3\times 3}$ is an identity matrix and $\boldsymbol{\Omega}_3(\boldsymbol{p}_v)$ is skew-symmetric matrix

$$\boldsymbol{\Omega}_3(\boldsymbol{q}_v) = \begin{bmatrix} 0 & -q_z & q_y \\ q_z & 0 & -q_x \\ -q_y & q_x & 0 \end{bmatrix}. \tag{A.4}$$

Inverse operation to $\boldsymbol{\Omega}_3(\boldsymbol{q}_v)$ is denoted as $\mathrm{vex}(\boldsymbol{q}_v)$ and it is given that

$$\mathrm{vex}\left(\boldsymbol{\Omega}_3(\boldsymbol{q}_v)\right) = \boldsymbol{q}_v. \tag{A.5}$$

As was said, the quaternion product is not commutative, however, the matrix form for the opposite order of quaternions exists

$$\boldsymbol{p} \otimes \boldsymbol{q} = \overline{\boldsymbol{\Omega}}_4(\boldsymbol{q})\boldsymbol{p} = \begin{bmatrix} q_w & -\boldsymbol{q}_v{}^{\mathsf{T}} \\ \boldsymbol{q}_v & q_w \cdot \boldsymbol{I}_{3\times 3} - \boldsymbol{\Omega}_3(\boldsymbol{q}_v) \end{bmatrix} \begin{bmatrix} p_w \\ \boldsymbol{p}_v \end{bmatrix}. \tag{A.6}$$

Similar to the complex numbers, the quaternion conjugate is defined as

$$\boldsymbol{q}^* = [q_w, -\boldsymbol{q}_v^{\mathsf{T}}]^{\mathsf{T}}. \tag{A.7}$$

The norm of the quaternion is

$$\|\boldsymbol{q}\| = \sqrt{\boldsymbol{q} \times \boldsymbol{q}^*} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}. \tag{A.8}$$

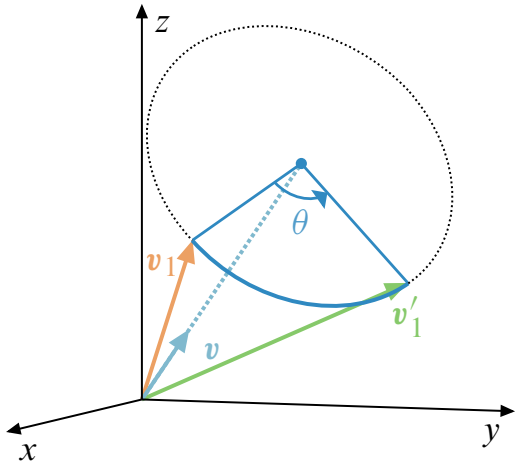


Figure A.1: Quaternion

Quaternion that represents the attitude has to observe the unit norm

$$\|\boldsymbol{q}\| = 1 \tag{A.9}$$

and it is called the unit quaternion. Inverse quaternion is

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|}. \tag{A.10}$$

Vector $\boldsymbol{r}^B$ is defined in frame B and it can be rotated by quaternion $\boldsymbol{q}$ to the frame L as

$$\boldsymbol{r}^L = \boldsymbol{q} \otimes \boldsymbol{r}^B \otimes \boldsymbol{q}^{-1}. \tag{A.11}$$

It has to be denoted that the rotation is due to a different dimension of rotated vector $\boldsymbol{r}^b$ and quaternion $\boldsymbol{q}$ performed as

$$\begin{bmatrix} 0 \\ \boldsymbol{r}^L \end{bmatrix} = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{r}^B \end{bmatrix} \otimes \boldsymbol{q}^{-1}. \tag{A.12}$$

Rotation from frame L to B is defined as

$$\boldsymbol{r}^B = \boldsymbol{q}^{-1} \otimes \boldsymbol{r}^L \otimes \boldsymbol{q}. \tag{A.13}$$

For a better understanding of the rotation by the quaternion, there is another relation

$$\boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \boldsymbol{v} \sin\left(\frac{\theta}{2}\right) \end{bmatrix}, \tag{A.14}$$

where $\boldsymbol{v}$ is a vector with the property $\|\boldsymbol{v}\| = 1$ that describes the axis of the rotation and θ is the angle of the rotation around $\boldsymbol{v}$. Figure A.1 depicts the rotation of vector $\boldsymbol{v}_1$ by the quaternion $\boldsymbol{q}$. In the mathematical notation, it is

$$\boldsymbol{v}_1' = \boldsymbol{q} \otimes \boldsymbol{v}_1 \otimes \boldsymbol{q}^{-1}. \tag{A.15}$$

The rotational dynamics can be described using quaternion and angular rate as

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \tag{A.16}$$

where $\boldsymbol{\omega} = \left[\omega_x, \omega_y, \omega_z\right]^\mathsf{T}$ is the angular rate in body frame B. Similarly to the quaternion product (A.3), a matrix notation for the angular rate exists as well

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{\Gamma}(\boldsymbol{q})^\mathsf{T}\boldsymbol{\omega}, \tag{A.17}$$

where matrix $\boldsymbol{\Gamma}(\boldsymbol{q})$ is composed as

$$\boldsymbol{\Gamma}(\boldsymbol{q}) = \left[-\boldsymbol{q}_v, \quad q_w \cdot \boldsymbol{I}_{3\times 3} - \boldsymbol{\Omega}_3(\boldsymbol{q}_v)\right]. \tag{A.18}$$

Matrix notation is beneficial for the relation in the opposite way, that is from the derivative of quaternion to the angular rate, which is according to [35]

$$\boldsymbol{\omega} = 2\boldsymbol{\Gamma}(\boldsymbol{q})\dot{\boldsymbol{q}} = -2\boldsymbol{\Gamma}(\dot{\boldsymbol{q}})\boldsymbol{q} \tag{A.19}$$

and there is also an important relation

$$2\boldsymbol{\Gamma}(\boldsymbol{q})\boldsymbol{\Gamma}(\dot{\boldsymbol{q}})^\mathsf{T} = -2\boldsymbol{\Gamma}(\dot{\boldsymbol{q}})\boldsymbol{\Gamma}(\boldsymbol{q})^\mathsf{T}, \ \boldsymbol{\Gamma}(\boldsymbol{q})\boldsymbol{\Gamma}(\boldsymbol{q})^\mathsf{T} = \boldsymbol{I}_{3\times 3}. \tag{A.20}$$

In [18], a relation for a discrete-time quaternion dynamics was denoted as

$$\boldsymbol{q}_{k+1} = \Theta(\boldsymbol{\omega}_k) \cdot \boldsymbol{q}_k, \tag{A.21}$$

where

$$\Theta(\boldsymbol{\omega}) = \begin{bmatrix} \cos(0.5\|\boldsymbol{\omega}\|T_s)\, I_3 - \Omega_3(\boldsymbol{\chi}) & \boldsymbol{\chi} \\ -\boldsymbol{\chi}^\mathsf{T} & \cos(0.5\|\boldsymbol{\omega}\|T_s) \end{bmatrix}, \tag{A.22}$$

$$\boldsymbol{\chi} = \sin(0.5\|\boldsymbol{\omega}\|T_s)\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}, \tag{A.23}$$

where T_s is the sampling period.

B Conversion of Attitude Representations

In this appendix, the relations for conversion between rotation matrix, quaternion, and Euler angles will be denoted. The conversion from the Euler angles to the quaternion is beneficial because of its physical properties. It is easier for example to define a set point for attitude in the Euler angles. The relation is based on the rotation matrices, which are composed for each angle separately. The Euler angles are ϕ, θ, ψ and they stand for the rotation around the axis $\vec{x}^L$, $\vec{y}^L$, $\vec{z}^L$, respectively. The corresponding rotation matrices are denoted as $\boldsymbol{R}_i$, where i is the angle, and they are composed as

$$\boldsymbol{R}_\psi = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R}_\theta = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \boldsymbol{R}_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}. \tag{B.1}$$

The rotation matrix for rotation in all three angles is composed by the multiplication of matrices (B.1). It is important to mention, that the resulting matrix is dependent on the sequence of matrices. For example matrix for the sequence $x^L - y^L - z^L$ it is defined as

$$\boldsymbol{R}_{\phi\theta\psi} = \boldsymbol{R}_\phi \boldsymbol{R}_\theta \boldsymbol{R}_\psi. \tag{B.2}$$

Quaternions can be also decomposed for the rotation in only one Euler angle.

$$\boldsymbol{q}_\phi = \begin{bmatrix} \cos\left(\frac{1}{2}\phi\right) \\ \sin\left(\frac{1}{2}\phi\right) \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{q}_\theta = \begin{bmatrix} \cos\left(\frac{1}{2}\theta\right) \\ 0 \\ \sin\left(\frac{1}{2}\theta\right) \\ 0 \end{bmatrix}, \boldsymbol{q}_\psi = \begin{bmatrix} \cos\left(\frac{1}{2}\psi\right) \\ 0 \\ 0 \\ \sin\left(\frac{1}{2}\psi\right) \end{bmatrix}. \tag{B.3}$$

The resulting quaternion is composed of the product of (B.3). Analogous to the rotation matrix, it is also dependent on the sequence of rotations. Quaternion for the rotation in the sequence $x^L - y^L - z^L$ is acquired with

$$\boldsymbol{q} = \boldsymbol{q}_\phi \otimes \boldsymbol{q}_\theta \otimes \boldsymbol{q}_\psi = \begin{bmatrix} \cos\left(\frac{1}{2}\phi\right)\cos\left(\frac{1}{2}\theta\right)\cos\left(\frac{1}{2}\psi\right) - \sin\left(\frac{1}{2}\phi\right)\sin\left(\frac{1}{2}\theta\right)\sin\left(\frac{1}{2}\psi\right) \\ \cos\left(\frac{1}{2}\psi\right)\cos\left(\frac{1}{2}\theta\right)\sin\left(\frac{1}{2}\phi\right) + \cos\left(\frac{1}{2}\phi\right)\sin\left(\frac{1}{2}\theta\right)\sin\left(\frac{1}{2}\psi\right) \\ \cos\left(\frac{1}{2}\psi\right)\cos\left(\frac{1}{2}\phi\right)\sin\left(\frac{1}{2}\theta\right) - \cos\left(\frac{1}{2}\theta\right)\sin\left(\frac{1}{2}\phi\right)\sin\left(\frac{1}{2}\psi\right) \\ \cos\left(\frac{1}{2}\phi\right)\cos\left(\frac{1}{2}\theta\right)\sin\left(\frac{1}{2}\psi\right) + \cos\left(\frac{1}{2}\psi\right)\sin\left(\frac{1}{2}\phi\right)\sin\left(\frac{1}{2}\theta\right) \end{bmatrix}. \tag{B.4}$$

On the other hand, a conversion from the quaternion to the Euler angles is beneficial for the presentation of results and comparison of attitudes. The following relations are valid

for the rotation in sequence $x^L - y^L - z^L$

$$\phi = \arctan\left(2q_w q_x - 2q_y q_z, q_w^2 - q_x^2 - q_y^2 + q_z^2\right), \tag{B.5}$$

$$\theta = \arctan\left(2q_w q_y + 2q_x q_z, \sqrt{1 - \left(2q_w q_y + 2q_x q_z\right)^2}\right), \tag{B.6}$$

$$\psi = \arctan\left(2q_w q_z - 2q_x q_y, q_w^2 + q_x^2 - q_y^2 - q_z^2\right). \tag{B.7}$$

The final relation, that will be presented, converts the quaternion into the rotation matrix

$$\boldsymbol{R}_{\phi\theta\psi}(\boldsymbol{q}) = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2q_x q_y - 2q_w q_z & 2q_w q_y + 2q_x q_z \\ 2q_w q_z + 2q_x q_y & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2q_y q_z - 2q_w q_x \\ 2q_x q_z - 2q_w q_y & 2q_w q_x + 2q_y q_z & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}. \tag{B.8}$$

For greater insight into the quaternion issues, see [58].

C Stabilizing Control of UAV

UAVs are inherently unstable systems and it is impossible to stabilize them without feedback. An exhaustive study of UAV's stability is carried out in [80, Chapter 10]. The UAV is unstable because the collective thrust counterbalances the gravitational force and as the attitude changes, the total thrust must change; otherwise, the effect of gravitational force would not be fully countered and the UAV would begin to change altitude. The UAV is most stable near the hover state because the collective thrust vector points directly against the gravitational force, therefore it is advantageous to control the UAV near the hover state.

When the UAV is near the stabilized hover state, the thrust along $\vec{z}^B$ results in the change of position in $\vec{z}^L$. Therefore it is the most agile during the movement along $\vec{z}^L$. To change the position in $\vec{x}^L$ and $\vec{y}^L$ the UAV must be tilted in the desired direction of motion. With the tilt, the UAV starts to accelerate. It is essential to maintain the tilt of the UAV in some reasonable neighborhood of the hover state because if the UAV is tilted too much it will start to descend. It is because of the propagation of the vector of thrust to $\vec{x}^L$ and $\vec{y}^L$; then the $\vec{z}^L$ component is not high enough to counter the gravitational force.

As previously indicated, it can be beneficial to split the stabilization control into several subsystems. The control of UAV is often split into the rotor speed, attitude, and position controls [63, 80]. An example of the stabilizing control system is pictured in Figure C.1.

In the UAV stabilization control problem there are many approaches [60, 85]. There are two main groups of controllers. The first group contains linear controllers [13], based on the standard control theory. The most common is PID or Proportional-Derivative (PD) controller [9]. They are used for their simplicity, deep insight into the performance, robustness, and many approaches to tune their parameters. Another controller that is employed in the UAV stabilization is the LQR which is a model-based optimal state-feedback controller. Its behavior is given by the weights in the quadratic criterion and the system dynamics.

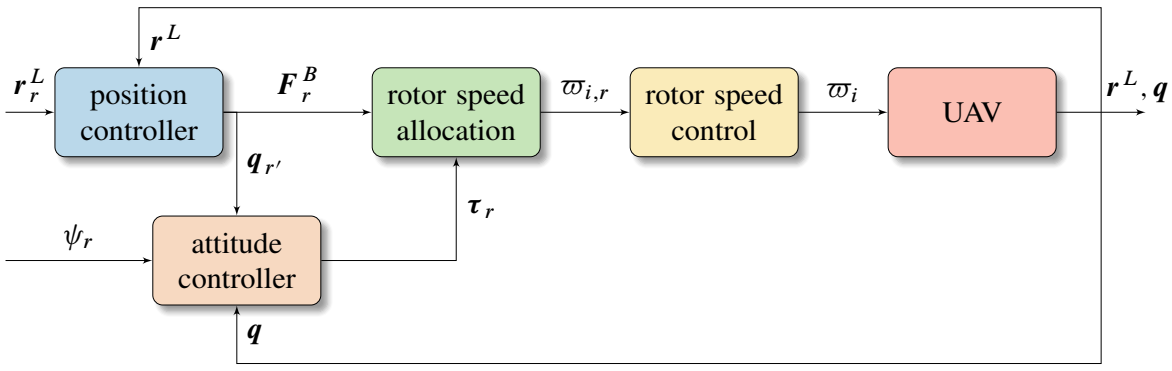


Figure C.1: Scheme of UAV Low Level Control

The second group consists of nonlinear controllers that can fully follow the three-dimensional Special Euclidean Group (SE(3)) trajectory [76, 63]. These controllers are designed to handle the inherent nonlinearities present in the UAV dynamics. In this context, the nonlinear controllers offer improved performance and adaptability compared to their linear counterparts, especially when dealing with complex trajectories and challenging operating conditions. Further, the nonlinear controller based on the quaternion representation will be presented. For this kind of controller, it is necessary to define the quaternion error

$$\boldsymbol{q}_e = \boldsymbol{q}_r^{-1} \otimes \boldsymbol{q}, \tag{C.1}$$

where $\boldsymbol{q}_r$ is the desired attitude quaternion and inverse quaternion is qiven by Equation (A.10).

The attitude controller is based on the second-order dynamics of attitude quaternion error $\boldsymbol{q}_e$

$$\ddot{\boldsymbol{q}}_e + 2k_{\boldsymbol{q}}k_{\boldsymbol{\omega}}\dot{\boldsymbol{q}}_e + k_{\boldsymbol{\omega}}\left(\boldsymbol{q}_e - \begin{bmatrix}1\\0\\0\\0\end{bmatrix}\right) = 0, \tag{C.2}$$

where $k_{\boldsymbol{q}}, k_{\boldsymbol{\omega}} \in \Re^{4\times4}$ are diagonal matrices with parameters of dynamics.

The derivative of attitude quaternion error is denoted as

$$\dot{\boldsymbol{q}}_e = \frac{1}{2}\boldsymbol{q}_e \otimes \left(\boldsymbol{\omega} - \boldsymbol{q}_e^{-1} \otimes \boldsymbol{\omega}_r \otimes \boldsymbol{q}_e\right) \tag{C.3}$$

and the angular acceleration in body frame can be described as

$$\dot{\boldsymbol{\omega}} = 2\boldsymbol{\Gamma}(\boldsymbol{q}_e)\ddot{\boldsymbol{q}}_e. \tag{C.4}$$

If the quaternion error dynamics (C.2) is employed using Equation (C.4) in Equation (2.6) for the angular acceleration dynamics, it results in the quaternion attitude controller

$$\boldsymbol{\tau}_r = \boldsymbol{I}\boldsymbol{\Gamma}(\boldsymbol{q}_e)\left(-2k_{\boldsymbol{q}}k_{\boldsymbol{\omega}}\dot{\boldsymbol{q}}_e - k_{\boldsymbol{\omega}}^2\left(\boldsymbol{q}_e - [1\,0\,0\,0]^\mathsf{T}\right)\right) + \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega}, \tag{C.5}$$

where $\boldsymbol{\tau}_r$ is a desired collective torque.

The corresponding relation for the position controller is

$$\boldsymbol{F}_r^L = m\left(\ddot{\boldsymbol{r}}_r^L + k_{\dot{\boldsymbol{r}}^L}\left(\dot{\boldsymbol{r}}_r^L - \dot{\boldsymbol{r}}^L\right) + k_{\boldsymbol{r}^L}\left(\boldsymbol{r}_r^L - \boldsymbol{r}^L\right) + g\vec{z}^L\right), \tag{C.6}$$

where $\ddot{\boldsymbol{r}}_r^L$, $\dot{\boldsymbol{r}}_r^L$ and $\boldsymbol{r}_r^L$ is desired acceleration, speed, and position in the local coordinate frame L, respectively; the control is in form of the desired translational force inflicting UAV's body in local coordinate frame $\boldsymbol{F}_r^L$ and $k_{\dot{\boldsymbol{r}}^L}, k_{\boldsymbol{r}^L} \in \Re^{3,3}$ are positively definite gain matrices.

Now, the connection of position and attitude controller, i.e. acquisition of $\boldsymbol{q}_r$, will be denoted. It is based on the connection between the desired force in frames L and B

$$\boldsymbol{F}_r^L = \boldsymbol{q}_{r'} \otimes \boldsymbol{F}_r^B \otimes \boldsymbol{q}_{r'}^{-1}. \tag{C.7}$$

The calculation of the first part of the desired attitude quaternion $\boldsymbol{q}_{r'}$ is obtained as follows

$$\boldsymbol{q}_{r'} = \frac{1}{\sqrt{2\left(1 + \dfrac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} \cdot \dfrac{\boldsymbol{F}_r^L}{\|\boldsymbol{F}_r^L\|}\right)}} \begin{bmatrix} 1 + \dfrac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} \cdot \dfrac{\boldsymbol{F}_r^L}{\|\boldsymbol{F}_r^L\|} \\ \dfrac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} \times \dfrac{\boldsymbol{F}_r^L}{\|\boldsymbol{F}_r^L\|} \end{bmatrix}. \tag{C.8}$$

If only the collective thrust is considered, the relation can be simplified with $\frac{\boldsymbol{F}_r^B}{\|\boldsymbol{F}_r^B\|} = [0,\ 0,\ 1]^\mathsf{T}$. In Figure C.1 it is depicted that the attitude controller input contains a part of the desired attitude quaternion $\boldsymbol{q}_{r'}$ and the heading ψ_r in [rad]. The complete desired attitude quaternion $\boldsymbol{q}_r$ can be composed using quaternion product and the relation between Euler angles and quaternions from Appendix B as

$$\boldsymbol{q}_r = \boldsymbol{q}_{r'} \otimes \begin{bmatrix} \cos\left(\frac{\psi_r}{2}\right) \\ 0 \\ 0 \\ \sin\left(\frac{\psi_r}{2}\right) \end{bmatrix}. \tag{C.9}$$

The quaternions $\boldsymbol{q}$ and $-\boldsymbol{q}$ represent the same attitude, which can cause a discontinuity in control since the rotation may not follow the shortest path. Therefore, the control algorithm should compare the sign of the desired quaternion with the previous one, for example, using $\boldsymbol{q}_{r_{t-1}}^\mathsf{T} \cdot \boldsymbol{q}_{r_t} \geq 0$ and, if necessary, correct the sign.

D Path Planning Algorithms

This appendix contains the pseudocodes of algorithms presented in Section 3.2. The first is the pseudocode of A* in Algorithm 2, where $\boldsymbol{s}$ is a current vertex, $\boldsymbol{s}'$ is neighbor of $\boldsymbol{s}$, $\boldsymbol{s}_{start}$ is the start vertex, and $\boldsymbol{s}_{goal}$ is the goal. Value g($\boldsymbol{s}$) is *cost-to-here* of vertex $\boldsymbol{s}$, h($\boldsymbol{s}$) is the cost-to-goal, c($\boldsymbol{s}, \boldsymbol{s}'$) is the distance of straight line from vertex $\boldsymbol{s}$ to $\boldsymbol{s}'$, parent($\boldsymbol{s}$) is parent of vertex $\boldsymbol{s}$.

In Algorithm 3 there is the utilized function for algorithm Theta*, which is based on A*, but it allows the movement directly in any angle between visible vertices. The changes in the Lazy Theta*, which employs the lazy evaluation of the line-of-sight check opposed to Theta* (where this check is performed in advance for every unexpanded neighbor), are denoted in Algorithm 4.

The pseudocode for the sampling-based algorithm RRT* is presented in Algorithm 5, where $\mathcal{T}$ is the tree structure for inserting of vertices, $\boldsymbol{s}_{init}$ is the vertex for the initialization of algorithm, $\boldsymbol{s}_{rand}$ is the sampled vertex. The function `steer` returns the adjusted sampled vertex according to given constraints, `obstacleFree` returns the true value if the coordinates of the vertex are free of obstacles, `chooseParent` returns parent of the given vertex, and `rewire` reconnects the tree to keep it dense and compact.

Algorithm 2 A* algorithm from [70]

```
1: function MAIN
2:     open = closed = ∅
3:     g(s_start) = 0
4:     parent(s_start) = s_start
5:     open.Insert(s_start, g(s_start) + h(s_start))
6:     while open≠ ∅ do
7:         s = open.Pop()
8:         [SetVertex(s)]                          ▷ Implemented only for Lazy Theta*
9:         if s = s_goal then
10:            return "path found"
11:        end if
12:        closed = closed ⋃{s}
13:        for ∀s′ ∈ nghbr_vis(s) do
14:            if s′ ∉open then
15:                g(s′) = ∞
16:                parent(s′) = NULL
17:            end if
18:            UpdateVertex(s, s′)
19:        end for
20:    end while
21:    return "no path found"
22: end function
23: function UPDATEVERTEX(s, s′)
24:    g_old = g(s′)
25:    ComputeCost(s, s′)
26:    if g(s′)<g_old then
27:        if s′ ∈ open then
28:            open.Remove(s′)
29:        end if
30:        open.Insert(s′,g(s′) + h(s′))
31:    end if
32: end function
33: function COMPUTECOST(s, s′)                     ▷ Path 1
34:    if g(s) + c(s, s′) < g(s′) then
35:        parent(s′) = s
36:        g(s′) = g(s) + c(s, s′)
37:    end if
38: end function
```

Algorithm 3 Theta* function for cost evaluation from [70]

```
1: function ComputeCost(s, s′)
2:     if LineOfSight(parent(s),s′) then                    ▷ Path 2
3:         if g(parent(s))+c(parent(s), s′)<g(s′) then
4:             parent(s′) = parent(s)
5:             g(s′) = g(parent(s)) + c(parent(s), s′)
6:         end if
7:     else                                                 ▷ Path 1
8:         if g(s)+c(s, s′) < g(s′) then
9:             parent(s′) = s
10:            g(s′) = g(s) + c(s, s′)
11:        end if
12:    end if
13: end function
```

Algorithm 4 Lazy Theta* function for cost evaluation and setting the vertex from [70]

```
1: function SetVertex(s)
2:     if ¬ LineOfSight(parent(s),s) then                   ▷ Path 1
3:         parent = argmin_{s′∈nghbr_vis(s)⋂closed} (g(s′) + c(s′, s))
4:         g(s) = min_{s′∈nghbr_vis(s)⋂closed} (g(s′) + c(s′, s))
5:     end if
6: end function
7: function ComputeCost(s, s′)                              ▷ Path 2
8:     if g(parent(s))+c(parent(s), s′)<g(s′) then
9:         parent(s′) = parent(s)
10:        g(s′) = g(parent(s)) + c(parent(s), s′)
11:    end if
12: end function
```

Algorithm 5 RRT* algorithm from [70]

function MAIN
  $\mathcal{T}$ = initializeTree()
  $\mathcal{T}$ = insertNode($\emptyset$, s_{init}, $\mathcal{T}$)
  for i = 1 **to** i = N **do**
    s_{rand} = sample(i)
    $s_{nearest}$ = nearest($\mathcal{T}$, s_{rand})
    (x_{new}, u_{new}, T_{new}) = steer($s_{nearest}$,s_{rand})
    if obstacleFree(x_{new}) **then**
      S_{near} = near($\mathcal{T}$, s_{new})
      s_{min} = chooseParent(S_{near}, $s_{nearest}$, s_{new}, x_{new})
      $\mathcal{T}$ = insertNode(s_{min}, s_{new}, $\mathcal{T}$)
      $\mathcal{T}$ = rewire($\mathcal{T}$, S_{near}, s_{min}, s_{new})
    end if
  end for
  return $\mathcal{T}$
end function
function CHOOSEPARENT(S_{near}, $s_{nearest}$, x_{new})
  $s_{min} = s_{nearest}$
  c_{min} = cost($s_{nearest}$) + c(s_{new})
  for $s_{near} \in S_{near}$ **do**
    (x', u', T') = steer(s_{near}, s_{new})
    if obstacleFree(x') & $x'(T') = s_{new}$ **then**
      c' = cost(s_{near}) + c(x')
      if c' <cost(s_{new}) & $c' < c_{min}$ **then**
        $s_{min} = s_{near}$
        $c_{min} = c'$
      end if
    end if
  end for
  return s_{min}
end function
function REWIRE($\mathcal{T}$, S_{near}, s_{min}, s_{new})
  for $s_{near} \in S_{near} \setminus \{s_{min}\}$ **do**
    (x', u', T') = steer(s_{new}, s_{near})
    if obstacleFree(x') & $x'(T') = s_{new}$ & cost(s_{near}) + c(x')< cost(s_{near}) **then**
      $\mathcal{T}$ = reconnect(s_{new}, s_{near}, $\mathcal{T}$)
    end if
  end for
  return $\mathcal{T}$
end function

E Procedures for Invariant Sets Calculation

In this appendix, two algorithms for the computation of the sets, that are employed in the Interpolating Control, will be described. Both algorithms are based on procedures in [72]. The Algorithm 6 computes the set $\boldsymbol{\Omega}^h$ that denotes the space where the Linear Quadratic Regulator can be used without violation of constraints. The Algorithm 7 computes the set $\mathcal{C}^N$, which defines the part of space from where the system can be steered in at least N steps to the origin.

Algorithm 6 Computation of robustly positive invariant set $\boldsymbol{\Omega}^h$

Input: Matrices $\boldsymbol{A}_{c1}, \boldsymbol{A}_{c2}, \ldots, \boldsymbol{A}_{cq}$, $\boldsymbol{D}$ and $\boldsymbol{K}$, sets $\mathcal{X}$, $\mathcal{W}$ and $\mathcal{U}$
Output: positive invariant set $\boldsymbol{\Omega}^h$.

$\boldsymbol{F}^h = \begin{bmatrix} \boldsymbol{F}^x \\ \boldsymbol{F}^u \boldsymbol{K} \end{bmatrix}$, $\boldsymbol{g}^h = \begin{bmatrix} \boldsymbol{g}^x \\ \boldsymbol{g}^u \end{bmatrix}$
$\mathcal{X}_0 = \left\{ \boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}^h \boldsymbol{x} \leq \boldsymbol{g}^h \right\}$
loop
    $$\mathcal{P} = \left\{ \boldsymbol{x} \in \mathcal{R}^n : \begin{bmatrix} \boldsymbol{F}^h \\ \boldsymbol{F}^h \boldsymbol{A}_{c1} \\ \boldsymbol{F}^h \boldsymbol{A}_{c2} \\ \vdots \\ \boldsymbol{F}^h \boldsymbol{A}_{cq} \end{bmatrix} \boldsymbol{x} \leq \begin{bmatrix} \boldsymbol{g}^h \\ \boldsymbol{g}^h - \max_{\boldsymbol{w} \in \mathcal{W}} \{ \boldsymbol{F}^i \boldsymbol{D} \boldsymbol{w} \} \\ \boldsymbol{g}^h - \max_{\boldsymbol{w} \in \mathcal{W}} \{ \boldsymbol{F}^i \boldsymbol{D} \boldsymbol{w} \} \\ \vdots \\ \boldsymbol{g}^h - \max_{\boldsymbol{w} \in \mathcal{W}} \{ \boldsymbol{F}^i \boldsymbol{D} \boldsymbol{w} \} \end{bmatrix} \right\}$$
    $\mathcal{P} =$MINIMALREPRESENTATION$(\mathcal{P})$ ▷ Erase redundant inequalities
    if $\mathcal{P} == \mathcal{X}^0$ **then**
        break
    end if
    $\mathcal{X}^0 = \mathcal{P}$
end loop
$\boldsymbol{\Omega}^h = \mathcal{X}^0$

Algorithm 7 Computation of robustly N-step positive invariant controlled set $\mathcal{C}^N$

Input: Matrices $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_q$, $\boldsymbol{B}_1, \boldsymbol{B}_2, \ldots, \boldsymbol{B}_q$, $\boldsymbol{D}$, sets $\mathfrak{X}$, $\mathfrak{U}$, $\mathfrak{W}$ and $\boldsymbol{\Omega}^h$, number of steps N

Output: positive invariant N-step controlled set $\mathcal{C}^N$

$i = 0$, $\mathcal{C}^0 = \boldsymbol{\Omega}^h$, $\mathcal{C}^0 = \left\{\boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}^h \boldsymbol{x} \leq \boldsymbol{g}^h\right\}$

loop

$$\mathcal{P}^i = \left\{ (\boldsymbol{x}, \boldsymbol{u}) \in \mathcal{R}^{n+m} : \boldsymbol{F}^i \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{B}_1 \\ \boldsymbol{A}_2 & \boldsymbol{B}_2 \\ \vdots & \vdots \\ \boldsymbol{A}_q & \boldsymbol{B}_q \\ \boldsymbol{F}^x & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{F}^u \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{g}^i - \max_{\boldsymbol{w} \in \mathfrak{W}} \{\boldsymbol{F}^i \boldsymbol{D} \boldsymbol{w}\} \\ \boldsymbol{g}^i - \max_{\boldsymbol{w} \in \mathfrak{W}} \{\boldsymbol{F}^i \boldsymbol{D} \boldsymbol{w}\} \\ \vdots \\ \boldsymbol{g}^i - \max_{\boldsymbol{w} \in \mathfrak{W}} \{\boldsymbol{F}^i \boldsymbol{D} \boldsymbol{w}\} \\ \boldsymbol{g}^x \\ \boldsymbol{g}^u \end{bmatrix} \right\}$$

$\mathcal{P}^{i(n)} = \left\{\boldsymbol{x} \in \mathcal{R}^n : \exists \boldsymbol{u} \in \mathfrak{U} \text{ such that } (\boldsymbol{x}, \boldsymbol{u}) \in \mathcal{P}^i\right\}$ ▷ Projection of $\mathcal{P}^i$ to $\mathcal{R}^n$

$\mathcal{C}^{i+1} = \mathcal{P}^{i(n)} \cap \mathfrak{X}$

$\mathcal{C}^{i+1} = \left\{\boldsymbol{x} \in \mathcal{R}^n : \boldsymbol{F}^{i+1} \boldsymbol{x} \leq \boldsymbol{g}^{i+1}\right\}$

if $\mathcal{C}^{i+1} == \mathcal{C}^i$ || i == N **then**

  $\mathcal{C}^N = \mathcal{C}^i$

  break

end if

$i = i + 1$

end loop