

Trajectory Planning for Flights in Multiagent and Dynamic Environments

by

Jesus Tordesillas Torres

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

© Massachusetts Institute of Technology 2022. All rights reserved.

Author
Department of Aeronautics and Astronautics
July 6, 2022

Certified by
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT
Thesis Supervisor

Certified by
Sertac Karaman
Associate Professor of Aeronautics and Astronautics, MIT

Certified by
Marco Pavone
Associate Professor of Aeronautics and Astronautics, Stanford
University

Accepted by
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Trajectory Planning for Flights in Multiagent and Dynamic Environments

by

Jesus Tordesillas Torres

Submitted to the Department of Aeronautics and Astronautics
on July 6, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

While efficient and fast trajectory planners in static worlds have been extensively proposed for UAVs (unmanned aerial vehicles), a 3D real-time planner for environments with static obstacles, dynamic obstacles, and other planning agents still remains an open problem. The dynamic nature of these environments demands high replanning rates, making this problem especially hard on computationally limited platforms. Existing state-of-the-art planners reduce the computational complexity at the expense of more conservative results by relying on three main simplifications or assumptions: First, the collision avoidance constraints are imposed using the Bernstein and B-Spline polynomial bases, which do not tightly enclose a given interval of a polynomial trajectory. Second, multiagent planners usually make centralized and/or synchronized computation assumptions, which lead to poor scalability with the number of agents or can degrade the overall performance. Finally, position and yaw are decoupled when optimizing perception-aware trajectories, which produces highly conservative results.

This thesis addresses the aforementioned limitations with the following contributions: First, it presents the MINVO basis, a polynomial basis that generates the simplex with minimum volume enclosing a polynomial curve, therefore reducing the conservativeness in the obstacle avoidance constraints. Leveraging the MINVO basis, this thesis then proposes a tractable way to avoid dynamic obstacles by imposing linear separability constraints between the polyhedral enclosures of the intervals of the trajectories. This is then extended to multiagent scenarios, and a decentralized and asynchronous obstacle avoidance algorithm among many replanning agents is presented. Real-time perception-aware planning is achieved by implicitly imposing the underactuated dynamics of the UAV through the Hopf fibration while jointly optimizing the full pose. Finally, a reduction of two orders of magnitude in the computation time is obtained by learning a policy that imitates the optimization-based planner. These proposed contributions are extensively evaluated in simulation, showing up to 32 agents planning in real time, and in real-world experiments, showcasing flights up to 5.8 m/s in unknown dynamic environments with only onboard computation.

Thesis Supervisor: Jonathan P. How

Title: R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT

Acknowledgments

First of all I would like to thank my advisor Professor Jonathan P. How. Thank you so much for the countless hours you have spent with me, guiding me throughout the research process, and prioritizing our weekly meetings over all the other work that you have to do. I cannot thank you enough for your continuous support and encouragement, especially in the moments when the research was not progressing as we expected.

Thank you to Professor Sertac Karaman and Professor Marco Pavone for serving on my thesis committee and providing me with great feedback in the committee meetings despite your extremely busy schedules. Your encouragement has helped me a lot and kept me motivated these past two years. Many thanks to my thesis readers Dr. Michael Everett and Professor Markus Ryll for their great feedback. Michael, thank you for your help, advice, and support over the past five years. Markus, I first met you when we carpooled together to ICRA'19 in Montreal. Your great lectures at VNAV and all of your publications made me more passionate about UAVs.

Thank you also to everyone else in the Aerospace Controls Lab. Thank you Kris, Yulun, Jeremy, Kaveh, Kasra, Björn, Samir, Shayegan, Nick, Kota,... A special thank you goes to Dong-Ki Kim: thank you for being one of my best friends here in Boston, for your support, for all the hours we have spent together at night working in the lab and encouraging each other, and for the squash games we have played together. Thanks also to Andrea Tagliabue: I still remember the day we met at JPL in Pasadena. At that time, I could not have imagined how much you were going to help me and how close of friends we were going to become. Thanks also to Parker Lusk for everything you have taught me, and for all the fun we have had flying drones outdoors.

Thanks also to many other very close friends I have here in Boston: John, Joe, James, Bruno, Jackson, Hugo, John Paul,... Thank you for all the discussions we have had and for how much we have helped each other. Thanks to Ángel, José, Diego, Marja, Fernando, Angie, Carlos, Marga, Sol, Andrea, Peter, Victoria, Taylor,

Kennedy, Pau, Nico, Rocío,... for our Thursday gatherings and all the fun stuff we have done almost every weekend. Thanks also to Peter, James, Olivia, Andrew, Joe, Sarah, Alex, Emma,... for so much time we have spent together this past year.

I would also like to give thanks to the professors at Escuela Técnica Superior de Ingenieros Industriales in Madrid that taught me so many things I have used throughout the PhD. Special thanks to Javier García de Jalón, Eduardo Caro, and Antonio Barrientos. Thanks also to my friends Antonio Caro, Pablo Vizcaíno, Joseda Madrigal, Juanjo Madrigal, Pablo Palafox, Javier Fernández, Rodrigo Aranguren, and Luis Paarup for all their help and encouragement throughout the years of my undergraduate education.

Last but not least, a special thanks to my family, especially to my parents Alberto and Maxi and to my siblings Alberto, Pablo, and Teresa. Thanks for the countless sacrifices you have made for me all these years and for supporting me at all times, especially when things got tough. Thanks also to my grandparents Marcial and Milagros, and Miguel and María. Thank you for all the time we have spent together in the fields picking grapes, olives, figs, and plums, and for showing me the beauty of the small hidden efforts that go unnoticed to everyone around us.

This work was supported by the La Caixa Fellowship, Boeing Research & Technology (MRA# 2017-656), the Air Force Office of Scientific Research MURI (FA9550-19-1-0386), and ONR (BRC award N000141712072).

Contents

Chapter 1 Introduction	17
1.1 Overview	17
1.2 Problem Statement	18
1.3 Literature Review	19
1.3.1 Polynomial Bases and Simplex Enclosures	19
1.3.2 Dynamic Obstacle Avoidance and Multiagent Environments	21
1.3.3 Perception-Aware Planning for UAVs	24
1.3.4 Imitation Learning for Trajectory Planning	28
1.4 Thesis Contributions and Structure	31
1.4.1 Non-conservative Simplex Enclosures	32
1.4.2 Planning in Multiagent and Dynamic Environments	32
1.4.3 Optimization-Based Perception-Aware Planning	33
1.4.4 Learning-Based Perception-Aware Planning	34
1.5 Related Publications	35
Chapter 2 MINVO Basis: Finding Simplexes with Minimum Volume Enclosing Polynomial Curves	37
2.1 Overview	37
2.2 Preliminaries	39
2.2.1 Notation and Definitions	39
2.2.2 Volume of the Convex Hull of a Polynomial Curve	40
2.3 Problems Definition	41
2.3.1 Given $P \in \mathcal{P}^n$, find $S \in \mathcal{S}^n$	41
2.3.2 Given $S \in \mathcal{S}^n$, find $P \in \mathcal{P}^n$	41

2.4	Equivalent Formulation	42
2.4.1	Constraints of Problems 1 and 2	42
2.4.2	Objective Function of Problem 1	44
2.4.3	Objective Function of Problem 2	45
2.4.4	Equivalent Formulation for Problems 1 and 2	45
2.5	Results	48
2.5.1	Results for $n = 1, 2, \dots, 7$	48
2.5.2	Results for $n > 7$	56
2.6	MINVO Basis Applied to Other Curves	58
2.6.1	Polynomial Curves of Degree n , Dimension k , and Embedded in a Subspace \mathcal{M} of Dimension m	58
2.6.2	Rational Curves	62
2.7	Comparison with SLEFEs	62
2.8	Final Remarks	66
2.8.1	Conversion between MINVO, Bernstein, and B-Spline	66
2.8.2	Tighter Volumes for Problem 1 via Subdivision	66
2.8.3	When Should each Basis Be Used?	67
Appendices		
2.A	Volume of the Convex Hull of a Polynomial Curve	68
2.B	Invertibility of the Matrix \mathbf{A}	69
2.C	Karush–Kuhn–Tucker Conditions (for odd n)	69
2.D	Plot of a Polynomial $x(t) \in \mathbb{R}[t]$ over $t \in [-1, 1]$	71
2.E	Application of the MINVO Basis to Surfaces	72
2.F	Definitions of the Union, Convex Hull, and Width of an Enclosure (for 2D Curves)	73
2.G	Comparison with SLEFEs and Bézier: Width	74
2.H	Comparison with SLEFEs and Bézier: Area and Number of Vertices	76
2.I	Comparison between MINVO and SLEFE in terms of Runtime and Simplicity in the Implementation	80

Chapter 3	MADER: Trajectory Planner in Multiagent and Dynamic Environments	83
3.1	Overview	83
3.2	Definitions	84
3.3	Assumptions	86
3.4	Polyhedral Representations	87
3.4.1	Polyhedral Representation of the Trajectory of the Agent s	88
3.4.2	Polyhedral Representation of the Trajectory of Other Agents	89
3.4.3	Polyhedral Representation of the Trajectory of the Obstacles	91
3.5	Optimization and Initial Guess	91
3.5.1	Collision-free Constraints	91
3.5.2	Other Constraints	91
3.5.3	Control Effort	93
3.5.4	Optimization Problem	94
3.5.5	Initial Guess	95
3.5.6	Degree of the Splines	98
3.6	Deconfliction	98
3.7	Results	102
3.7.1	Single-Agent Simulations	102
3.7.2	Multiagent Simulations Without Obstacles	104
3.7.3	Multiagent Simulations with Static and Dynamic Obstacles	108
Chapter 4	PANTHER: Perception-Aware Trajectory Planner in Dynamic Environments	111
4.1	Overview	111
4.2	PANTHER	113
4.2.1	Tracking and Prediction	113
4.2.2	Selection of the Obstacle in the PA Term	115
4.2.3	Planes and Initial Guesses	116
4.2.3.1	Separability Planes and Initial Guess for Position	116

4.2.3.2	Initial Guess for ψ	116
4.2.4	Optimization	118
4.2.4.1	Coupling Rotation and Acceleration with the Hopf Fibration	118
4.2.4.2	Cost Function	120
4.2.4.3	Collision Avoidance and Dynamic Limits Constraints	121
4.2.4.4	Optimization Problem	122
4.3	Results and Discussion	123
4.3.1	Simulation Experiments	123
4.3.1.1	Single Obstacle	123
4.3.1.2	Several Obstacles	126
4.3.1.3	Computational Analysis of the Replanning Step as a Function of the Number of Obstacles	128
4.3.2	Real-World Experiments	129
Chapter 5 Deep-PANTHER: Learning-Based Perception-Aware Trajectory Planner in Dynamic Environments		133
5.1	Overview	133
5.2	Deep-PANTHER	134
5.2.1	Expert and Student	134
5.2.2	Observation and Action	136
5.2.3	Loss: Capturing Multimodality	137
5.2.4	Generation of ψ Given the Position Trajectory	138
5.2.5	Testing	140
5.3	Results and Discussion	141
5.3.1	Static Obstacle	141
5.3.1.1	Cost vs Replanning time	142
5.3.1.2	Multimodality	144
5.3.2	Replanning with Dynamic Obstacles	145
5.3.3	Generalization to Other Obstacle Trajectories	146

5.4	Limitations	147
Chapter 6 Conclusions		149
6.1	Summary of Contributions	149
6.2	Future Work	150

List of Figures

1-1	Contributions of MADER	23
1-2	UAV using PANTHER to plan perception-aware trajectories in a dynamic unknown environment	24
1-3	Trajectories generated by the expert for four different goals	29
1-4	Comparison between the assignment matrix \mathbf{A} obtained by the WTAr, RWTAr, WTAc, RWTAc, and LSA approaches	30
1-5	Contributions of this thesis	31
2-1	Geometric interpretation of the barycentric coordinates	43
2-2	Relationship between Problems 1, 2, 3, and 4	48
2-3	Comparison between the MINVO, Bernstein, B-Spline, and Lagrange bases	50
2-4	Geometric interpretation of \mathbf{A}_{MV}	52
2-5	Solution obtained by the MINVO basis for $n = 3$	53
2-6	Comparison between the MINVO, Bernstein, and B-Spline bases for $n = 2$	54
2-7	Comparison between the MINVO, Bernstein, and B-Spline bases for $n = 3$	54
2-8	MINVO results for $n = 3$	55
2-9	Distribution of the roots of the MINVO basis functions for different n	56
2-10	MINVO basis functions obtained for $n = 10, 16, 24, 30$ using the model proposed	56
2-11	The MINVO basis applied for different curves with $k = 3$ and different values of m and n	59

2-12	Comparison of the convex hull of the MINVO and Bézier control points for $k = m = 2$ and different n	60
2-13	Comparison of the convex hull of the MINVO and Bézier control points for $k = m = 3$ and different n	60
2-14	Comparison of the convex hull of the MINVO and Bézier control points with respect to $\text{conv}(P)$	61
2-15	Tight enclosures of some rational curves	63
2-16	Comparison of the number of <i>raw</i> points	64
2-17	Tighter polyhedral outer approximations obtained by splitting a curve	67
2-18	Comparison of the convex hull of the MINVO and Bézier control points for $k = m \in \{1, 2\}$ and different n	72
2-19	MINVO basis applied to surfaces	73
2-20	2D trim curve of a CAD model	74
2-21	Models taken from the ABC dataset	75
2-22	Comparison between the width of the enclosures	76
2-23	Comparison between the area and number of vertices of the enclosures	77
2-24	Examples of the enclosures found for curves with $n = 2, \dots, 5$	78
2-25	Examples of the enclosures found for curves with $n = 6, 7$	79
3-1	Comparison of the volumes, areas, and lengths obtained	90
3-2	Polyhedral representations used by MADER	90
3-3	Example of a trajectory avoiding a dynamic obstacle	92
3-4	Collision-free constraints	93
3-5	Trajectories found by the Octopus Search Algorithm	96
3-6	Deconfliction between agents	99
3-7	Time allocated for each of the periods	100
3-8	Corridor environment used in the simulations	103
3-9	Boxplots and normalized histograms of the velocity profile	103
3-10	Time to reach the goal and number of times the UAV had to stop	104
3-11	Time notation for the algorithms that have replanning	106

3-12	Results for the circle environment	108
3-13	Results for the sphere environment	109
4-1	Modules of PANTHER, polyhedral representations, coordinate frames, and Hopf fibration	114
4-2	Simulation results	124
4-3	Computational analysis of different parts of the replanning step of PANTHER	128
4-4	Composite images of the experiments and relative distances	130
4-5	Snapshots of the onboard camera in the experiments, and computation times	131
5-1	Multimodal training in Deep-PANTHER	138
5-2	Optimal ψ trajectory given the position trajectory	139
5-3	Generation of $\mathbf{p}(t)$ and $\psi(t)$ from the observation	140
5-4	Testing scenario with a static obstacle and 64 different goals	142
5-5	Comparison of the cost and replanning time	143
5-6	Comparison between the MSE loss of π_{LSA} , $\pi_{\text{RWTAr-}\epsilon}$, and $\pi_{\text{RWTAc-}\epsilon}$	143
5-7	Comparison of the collision-free trajectories produced by LSA, RWTAr- ϵ , and RWTAc- ϵ	143
5-8	Snapshots of the trajectories produced by Deep-PANTHER in a dynamic environment	145
5-9	Trajectories used to train and test the student	146

List of Tables

1.1	Classification of the related work	25
1.2	Contributions of this thesis	32
2.1	Notation used in this chapter	38
2.2	Results for the MINVO basis	49
2.3	Roots of each $\lambda_i(t)$ of the MINVO basis	51
2.4	Comparison of the results obtained from the optimization and the model	57
2.5	Cases of polynomial curves of degree n , dimension k , and embedded in a subspace \mathcal{M} of dimension m	59
2.6	Comparison between the widths of the enclosures	75
2.7	Comparison of the computation times	81
3.1	Notation used in this chapter	85
3.2	Polyhedral representations of interval j	88
3.3	Comparison of the number of stops and time to reach the goal	105
3.4	Comparison between MADER and other state-of-the-art algorithms	107
3.5	Results for MADER in the circle and sphere environments	110
4.1	Notation used in this chapter	112
4.2	Commonly-used definitions for the differential flatness map	118
5.1	Notation used in this chapter	135
5.2	Generalization to other obstacle trajectories	147

Chapter 1

Introduction

1.1 Overview

While the last decade has seen an increase in the number of successful trajectory planners for unmanned aerial vehicles (UAVs) deployed in real-world scenarios, their applicability is often limited by several factors. First, many of these trajectory planners impose obstacle avoidance constraints by leveraging the Bézier and B-Spline polynomial bases, which can generate over-conservative results when outer approximating the volume occupied by a polynomial trajectory. Moreover, they typically assume that the environment is static, even when many UAV applications (delivery, aerial videography, emergency response, etc.) are highly dynamic due to the presence of cars, people, and/or other UAVs. In multiagent scenarios, many state-of-the-art planners rely on the assumptions of synchronization (all the agents plan at the same time or sequentially) and/or centralization (the optimization problem is solved jointly for the trajectories of all the agents). However, these assumptions can easily lead to conservative behaviors and scalability issues, especially when the number of agents is high. Finally, and when flying in unknown dynamic environments, another common assumption is the omnidirectional coverage of the sensor(s) of the UAV, even though most of the UAVs are equipped with sensors with a limited FOV. This thesis aims to address these assumptions and limitations, which is critical to fully exploit the potential of the UAVs and expand the range of their possible applications.

1.2 Problem Statement

Overall, this thesis investigates the problem of *trajectory planning for an underactuated multirotor-UAV in an unknown environment with dynamic obstacles and other agents*. Out of the many possible topics to study within this broad problem, this thesis focuses specifically on the following subproblems:

- Discretization of the trajectory to impose collision avoidance constraints can generate unsafe results, and a fine discretization of the trajectory leads to a very high computational burden. The convex hull property of the Bernstein or B-Spline bases has been extensively exploited to address this issue, but these bases provide undesirably conservative results. A polynomial basis that provides **tight simplex enclosures** of each interval of the trajectory is therefore crucial to achieve less conservative results, and still an open problem.
- Moreover, many of the state-of-the-art trajectory planners are either not applicable to dynamic environments, or suffer from very high computation times due to the extra *time* dimension. Naive solutions, like classifying as occupied all the space a dynamic obstacle will use in the planning horizon, can be extremely conservative and easily lead to unfeasible solutions. And point-wise constraints between the trajectories of the agent and the obstacles easily make the problem intractable. This leaves unanswered the question of how to impose **dynamic collision-avoidance constraints** while ensuring onboard real-time computational tractability.
- When other agents are present, the common assumptions of centralization (all the trajectories of the agents are jointly optimized) and/or synchronization (all the agents plan sequentially or at the same time) can generate conservative results or lead to scalability issues when the number of agents is high. However, ensuring safety in **decentralized and asynchronous** planners remains challenging, since the UAV has access only to past committed trajectories of the other agents, and the current committed trajectories keep changing during

the optimization time.

- When a UAV is flying in an unknown dynamic environment, it needs to keep the obstacles in the FOV of the camera to be able to predict their future trajectories and therefore to successfully avoid them. Hence, the use of standard non-perception-aware planners in this setting severely degrades the performance, as the flown trajectories are obtained without any consideration of the visibility of the obstacles. When this **perception awareness** (PA) is taken into account, the extensively used translation-yaw decoupling in the planning problem can generate highly conservative results, due to the fact that translation is optimized independently from yaw. However, and because of the underactuated nature of the UAV, the joint optimization of these two can lead to very high computation times. A formulation that benefits from the joint optimization of translation and yaw, and that guarantees real-time computational tractability is therefore desirable.
- The inclusion of a PA term in the optimization and the underactuated dynamics of the UAVs make translation and the *full* rotation coupled together. To guarantee real-time computation, and at the expense of more conservative results, it is common to simplify this problem by fixing some variables in the optimization (such as the time allocation or the planes that separate the trajectories) or by ignoring the multimodality of the problem. Therefore, the open question is whether it is possible to achieve **very fast computation times** without making these simplifications.

1.3 Literature Review

1.3.1 Polynomial Bases and Simplex Enclosures

Herron [61] attempted to find (for $n = 2$ and $n = 3$) the smallest n -simplex enclosing an n^{th} -degree polynomial curve. The approach of [61] imposed a specific structure on the polynomials of the basis and then solved the associated nonconvex optimization

problem over the roots of those polynomials. For this specific structure of the polynomials, a global minimizer was found for $n = 2$, and a local minimizer was found for $n = 3$. However, global optimality over all possible polynomials was not proven, and only the cases $n = 2$ and $n = 3$ were studied. Similarly, in the results of [81], Kuti et al. use the algorithm proposed in [86] to obtain a minimal 2-simplex that encloses a specific 2nd-degree polynomial curve. However, this approach requires running the algorithm for *each* different curve, no global optimality is shown, and only one case with $n = 2$ was analyzed. There are also works that have derived bounds on the distance between the control polygon and the curve [70,103,122], while others propose the SLEFE (subdividable linear efficient function enclosure) to enclose spline curves via subdivision [98,100,114,129]. However, the SLEFE depends *pseudo-linearly* on the coefficients of the polynomial curve (i.e., linearly except for a min/max operation) [129], which is disadvantageous when the curve is a decision variable in a time-critical optimization problem. This work focuses instead on enclosures that depend *linearly* on the coefficients of the curve.

Other works have focused on the properties of the smallest n -simplex that encloses a given generic convex body. For example, [52,69] derived some bounds for the volume of this simplex, while Klee [75] showed that any locally optimal simplex (with respect to its volume) that encloses a convex body must be a centroidal simplex. In other words, the centroid of its facets must belong to the convex body. Applied to a curve P , this means that the centroid of its facets must belong to $\text{conv}(P)$. Although this is a necessary condition, it is *not* sufficient for local optimality.

When the convex body is a polyhedron (or equivalently the convex hull of a finite set of points), [170] classifies the possible minimal circumscribing simplexes, and this classification is later used by [185] to derive a $O(k^4)$ algorithm that computes the smallest simplex enclosing a finite set of k points. This problem is also crucial for the hyperspectral unmixing in remote sensing, to be able to find the proportions or abundances of each macroscopic material (endmember) contained in a given image pixel [60,169], and many different iterative algorithms have been proposed towards this end [65,137,171]. All these works focus on obtaining the enclosing simplex for

a generic discrete set of points. Our work focuses instead on polynomial curves and, by leveraging their structure, we can avoid the need to iterate and/or discretize the curve.

The convex hull of curves has also been studied in the literature. For instance, [36, 134, 146] studied the boundaries of these convex hulls, while [29] focused on the patches of the convex hull of trajectories of polynomial dynamical systems. For a moment curve $[t t^2 \dots t^n]^T$ (where t is in some interval $[a, b]$), [113] found that the number of points needed to represent every point in the convex hull of this curve is $\frac{n+1}{2}$, giving therefore a tighter bound than the $n+1$ points found using the Carathéodory’s Theorem [22, 151]. This particular curve and the volume of its convex hull were also analyzed by [71] in the context of moment spaces and orthogonal polynomials. Although many useful properties of the convex hull of a curve are shown in all these previous works, none of them addresses the problem of finding the polynomial curve with largest convex hull enclosed in a given simplex.

1.3.2 Dynamic Obstacle Avoidance and Multiagent Environments

Many different UAV trajectory planners for static worlds have been proposed in the literature [21, 26, 46, 47, 55, 92, 93, 124, 164, 167, 182–184]. However, trajectory planning when there are static obstacles, dynamic obstacles, *and* other planning agents present at the same time still remains challenging.

To be able to guarantee safety, the trajectory of the planning agent and the ones of other obstacles/agents need to be encoded in the optimization (see Fig. 1-1). A common **representation** of this trajectory in the optimization is via points discretized along the trajectory [53, 87, 116, 130, 154, 186]. However, this does not usually guarantee safety between two consecutive discretization points and alleviating that problem by using a fine discretization of the trajectory can lead to a very high computational burden. To reduce this computational burden, polyhedral outer representations of each interval of the trajectory are extensively used in the literature,

with the added benefit of ensuring safety at all times (i.e., not just at the discretization points). A common way to obtain this polyhedral outer representation is via the convex hull of the control points of the Bernstein basis (basis used by Bézier curves) or the B-Spline basis [131, 157, 167, 182]. However, these bases do not yield very tight (i.e., with minimum volume) enclosures of the curve, leading to conservative results.

MADER (the trajectory planner presented in Chapter 3), addresses this conservativeness at its source and leverages the MINVO basis (Ref. [162] and Chapter 2 of this thesis) to obtain control points that generate the n -simplex (a tetrahedron for $n = 3$) with the minimum volume that completely contains each interval of the curve. Numerical global optimality (in terms of minimum volume) of this tetrahedron obtained by the MINVO basis is guaranteed both in position and velocity space.

When other agents are present, the **deconfliction** problem between the trajectories also needs to be solved. Most of the state-of-the-art approaches either rely on centralized algorithms [10, 80, 178] and/or on imposing an ad hoc priority such that an agent only avoids other agents with higher priority [28, 104, 117, 126, 136]. Some decentralized solutions have also been proposed [28, 88, 97], but they require synchronization between the replans of different agents. The challenge then is how to create a decentralized and asynchronous planner that solves the deconfliction problem and guarantees safety and feasibility for all the agents.

MADER solves this deconfliction in a decentralized and asynchronous way by including the trajectories other agents have committed to as constraints in the optimization. After the optimization, a collision check-recheck scheme ensures that the trajectory found is still feasible with respect to the trajectories other agents have committed to while the optimization was happening.

To impose **collision-free constraints** in the presence of static obstacles, a common approach is to *first* find convex decompositions of free space and *then* force (in the optimization problem) the outer polyhedral representation of each interval to be inside these convex decompositions [35, 89, 167]. However, this approach can be conservative, especially in cluttered environments in which the

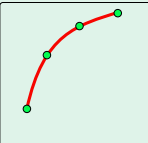
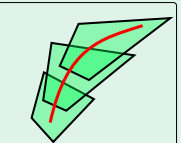

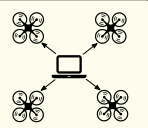

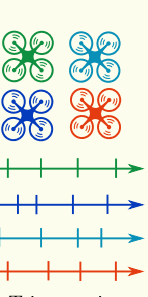
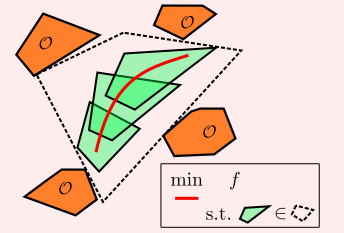
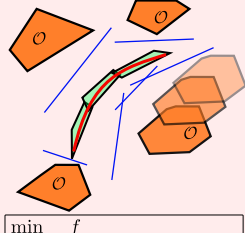
	Other Approaches		MADER
Interval represent.	 <p>Discretization Not safe Comput. time</p>	 <p>Bézier/B-Spline Safe Conservative</p>	 <p>MINVO basis Safe Less conservative</p>
Multi-Agent deconfliction	 <p>Centralized Strong assumption</p>	 <p>Sequential dec./cen. Ad-hoc priority</p>	 <p>Decentralized Asynchronous</p>
Collision-free constraints	 <p>Convex Decompositions Conservative Only for static obstacles</p>	 <p>Static + Dynamic Inclusion of the planes</p>	

Figure 1-1: Contributions of MADER.

convex decomposition algorithm may not find a tight representation of the free space. In the presence of dynamic obstacles, these convex decompositions become harder, and likely intractable, due to the extra time dimension.

To be able to impose collision-free constraints with respect to dynamic obstacles/agents, MADER imposes the separation between the polyhedral representations of each trajectory via planes. Moreover, MADER overcomes the conservativeness of a convex decomposition (imposed ad hoc before the optimization) by including a parameterization of these separating planes as decision variables in the optimization problem. The solver can thus choose the optimal location of these planes to determine collision avoidance. Including this plane parameterization reduces

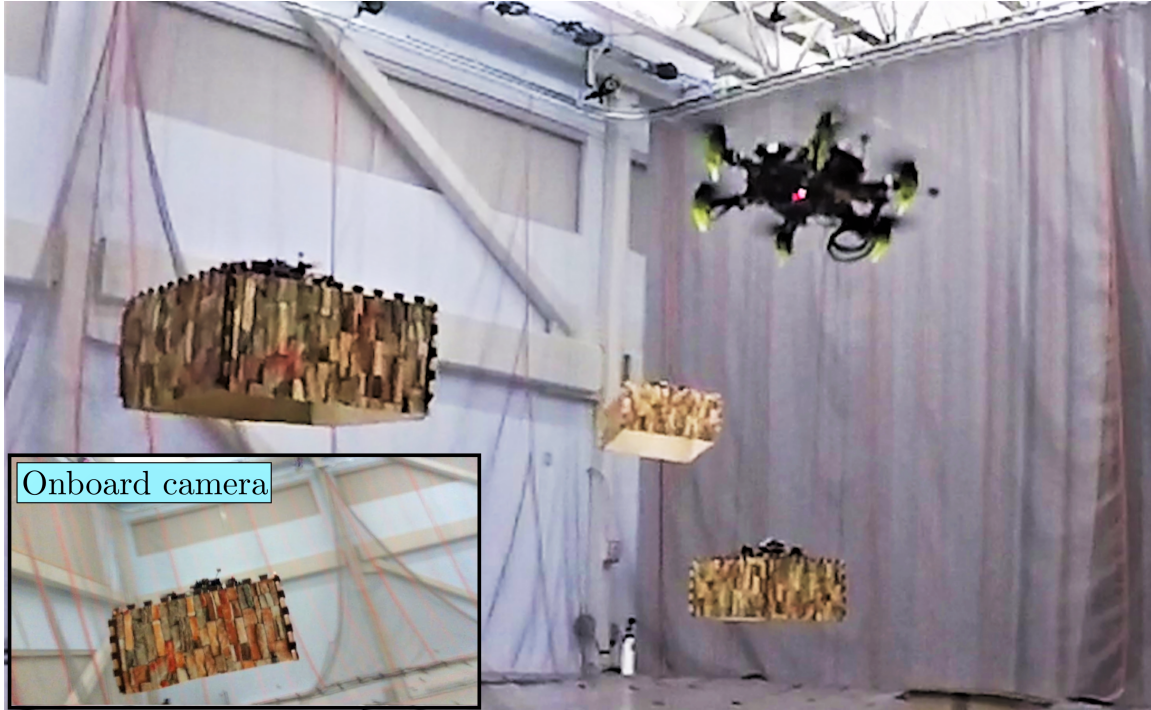


Figure 1-2: UAV using PANTHER (Chapter 4) to plan perception-aware trajectories in a dynamic unknown environment, with relative velocities of up to 6.3 m/s. All the computation runs onboard, and the UAV does not have any prior knowledge of the trajectories or specific shape/size of the dynamic obstacles.

conservativeness, but it comes at the expense of creating a nonconvex problem, for which a good initial guess is imperative. For this initial guess, Chapter 3 of this thesis presents a search-based algorithm that handles dynamic environments and obtains both the control points of the trajectory and the planes that separate it from other obstacles/agents.

1.3.3 Perception-Aware Planning for UAVs

When a UAV equipped with a limited FOV sensor is flying in an unknown environment (e.g., Fig 1-2), it is crucial to plan both the position and orientation of the UAV to maximize the detection and the tracking accuracy of the unknown obstacles while at the same time doing obstacle avoidance. This perception-aware (PA) component is especially important when flying in dynamic environments, because a consistent detection of the moving obstacles is necessary to obtain a good estimate

Table 1.1: Classification of the related work, together with a (nonexhaustive) list of references.

Formulation		Ref.
(A.1) Not PA		[26, 43, 54, 87, 143, 167, 173]
(A.2) PA with additional hardware		[18, 25, 38, 121]
(A.3) PA planning	Decoupling	[119, 149, 183]
	Joint opt.	[42, 127, 175], PANTHER

Goal	Ref.
(B.1) Reduce state estimation uncertainty	[5, 11, 32, 42, 50, 83, 119, 128, 132, 142, 148, 149, 175, 181]
(B.2) Record/chase a target	[27, 67, 68, 85, 127, 128, 159]
(B.3) Avoidance of dynamic obstacles	PANTHER

of their locations and prediction of their future trajectories.

Perception awareness for UAVs has been studied thoroughly in the literature, and, as shown in Table 1.1, the related work could be classified according to the formulation used and the goal itself. From the point of view of the **formulation** used, there are approaches that are **not PA** (A.1), which typically plan the translation and then have either a constant yaw or a yaw such that the FOV of the camera points in the direction of travel (e.g., see [26, 43, 54, 87, 143, 167, 173]). For instance, [43] used potential fields to avoid dynamic obstacles, but without taking into account perception awareness, which can degrade the detection and prediction of the trajectories of the obstacles.

Other approaches are **PA by including additional hardware** (A.2): For example, by gimbal-mounting the camera, some of its degrees of freedom can be controlled independently of the rotation of the UAV [18, 25, 121]. Another option is to mount omni-directional sensors [38]. However, these approaches usually require additional hardware and mechanical complexity, which is typically undesirable on small UAVs.

PA planning (A.3) has received increased attention over the last few years due to its inherent ability to leverage the trajectory planned to maximize the PA objective. The related works could be subclassified according to whether or not the translation and yaw of the UAV are jointly optimized. On one hand, there are approaches that decouple translation and yaw by optimizing them separately [119, 149, 183]. For instance, in [183], a yaw trajectory is obtained for a fixed translational path to gain information about unknown static obstacles. For features or landmarks whose locations are known a priori, [149] optimizes the time parametrization on a fixed

spatial and yaw path to maximize their visibility. In [119], translation is optimized first, and then yaw is optimized to guarantee the co-visibility of the features. While this decoupling of translation and yaw has computational advantages, it can lead to conservative results, since the translational trajectory (and consequently two degrees of freedom of the rotation as well) is fixed in the yaw optimization. Other works assume a downward-facing camera, and hence only translation (not yaw) is planned to keep a specific target in the FOV of the camera [159].

Another approach taken is to jointly optimize translation and yaw, which enables the planner to fully exploit *both* the position trajectory and the yaw angle [42,127,175]. This joint optimization leads to less conservative results than the approaches that decouple translation and yaw, but it typically comes at the expense of much higher computation times, especially when done in combination with dynamic obstacle avoidance constraints. For example, [175] proposed an on-manifold trajectory optimization approach that couples together translation with the full rotation, but the computation times required (up to 30 s) are not real time. Ref. [42] successfully presented a real-time MPC formulation that keeps the centroid of the VIO features in the center of the image while minimizing its projected velocity. However, this formulation does not include collision avoidance of static (or dynamic) obstacles, which greatly simplifies the complexity of the optimization problem. In [127], translation and yaw are optimized jointly, but only static obstacle avoidance is performed. The technical gap then is how to jointly optimize the full pose of the UAV, satisfy its underactuated dynamics, and guarantee safety in dynamic environments while maintaining real-time computational tractability.

The underactuated dynamics of the UAV (caused by the total thrust of the UAV being fixed in the body frame) makes this joint optimization especially hard, since a given spatio-temporal path fixes two degrees of freedom of the rotation, leaving only one extra degree of freedom in the rotation.¹ A typical way to impose this constraint is via the dynamic equations of the UAV. However, this comes at the expense of having differential equations as constraints in the optimization.

¹Usually referred to as *yaw*, *heading*, or simply ψ .

An alternative is to leverage the differential flatness of the UAVs [115] and make use of the map $(\mathbf{a} \in \mathbb{R}^3 \setminus [0 \ 0 \ -g]^T, \psi \in S^1) \rightarrow \mathbf{R}_b^w \in \text{SO}(3)$ that maps ψ and the acceleration \mathbf{a} to the rotation of the body. However, and due to the *hedgehog theorem* in S^2 [12, 20], there is no single continuous function that defines this map for all possible accelerations \mathbf{a} . For the most common definitions of this map, the singularity appears for each ψ at two antipodal points in the unit sphere of possible normalized relative accelerations, which means that there is at least one singularity with a great-circle distance $\leq 90^\circ$ with respect to the hovering condition. This closeness between the hovering condition and the singularity can limit the set of possible accelerations in aggressive flights, since an optimal solution that passes through or close to this singularity can provoke numerical instabilities and/or lead to artificial large changes in orientation. Recently, the Hopf map was leveraged in [174] to place the singularity in the inverted (“upside-down”) configuration, which is independent of ψ and has the farthest possible angle away from the hovering condition. Although flying highly aggressive trajectories is not the main goal of this work, we decide to use the Hopf map (as opposed to the commonly-used maps presented in [41, 115]) since it automatically maximizes the distance to the singularity by simply changing the definition of the map. In [174], however, the Hopf fibration was only used in the controller to track predefined trajectories. It was also leveraged in [176] to find the set of charts for a previously-optimized position trajectory, which are then used for the controller and to obtain the ψ trajectory. In Chapter 4 of this thesis, we propose instead to embed the Hopf fibration in the joint (translation *and* yaw) coupled planning optimization as a way to directly obtain trajectories in SE(3) that, by construction, satisfy the underactuated dynamics of the UAV.

From the point of view of the **goal** of the perception awareness, most of the related works focus on **reducing the state estimation uncertainty** (B.1), usually by keeping specific features/landmarks in the FOV, and/or choosing high-textured areas [5, 11, 32, 42, 50, 83, 119, 128, 132, 142, 148, 149, 175, 181]. These features are typically static in the world frame. Some of these approaches also leverage the Observability Gramian [50, 132, 142], especially when trying to ease the estimation

of an unknown parameter of the dynamical system.

Further relevant work addresses the problem of having a UAV **record or chase a target** (B.2) [27, 67, 68, 85, 127, 128, 159]. For example, [159] focused on tracking a moving target with a downward-facing camera, while [127] proposed a way to follow a moving target while avoiding other static obstacles in the environment. Most of these works focus therefore on chasing a static or dynamic target, not on avoiding it.

Our work differs from these two previous approaches because it proposes the use of PA planning to enhance the **avoidance of dynamic obstacles** (B.3). Compared to (B.1) or (B.2), PA planning to avoid unknown dynamic obstacles comes with many additional challenges, such as the coupling of *both* the ego-motion and the motion of the obstacle in the visibility cost and blur of the image, the inclusion of dynamic obstacle avoidance constraints in the optimization, the need to predict the future trajectories of the obstacles, and the consideration of the uncertainty of these predicted trajectories, just to name a few.

1.3.4 Imitation Learning for Trajectory Planning

Trajectory planning for UAVs in unknown dynamic environments requires very fast replanning times, which are usually achieved by simplifying the optimization problem by fixing some variables (such as the time allocation or the planes that separate the UAV from the obstacles) beforehand or by ignoring the multimodality of the problem (Ref. [163] and Chapter 4). While these simplifications help reduce the computation time, that is often achieved at the expense of more conservative planned trajectories. This leaves open the question of whether or not it is possible to obtain *faster* computation times while achieving *less conservative* trajectories.

Towards this end, Imitation Learning (IL) has recently gained interest due to its ability to train a computationally-cheap neural network (the student) to approximate the solution of a computationally-expensive algorithm (the expert). IL has been successfully used to compress MPC policies [73, 125, 135, 155] and/or to learn path planning policies [30, 96, 140]. Compared to other IL-based trajectory planning works, which typically either assume static worlds or do not take into account perception

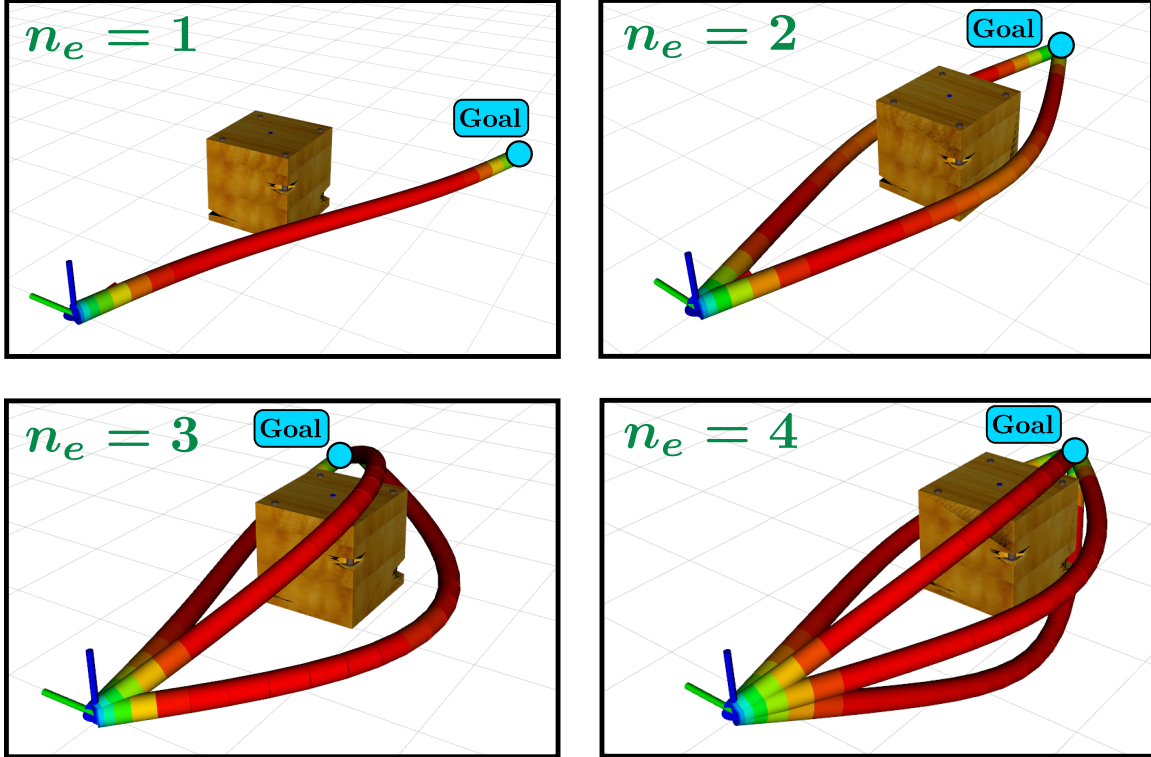


Figure 1-3: Trajectories generated by the expert for four different goals \bullet . For each of the cases, we plot all the distinct locally optimal solutions found by running a total of 10 optimizations with different initial guesses. The number of these solutions is denoted as n_e . Note how n_e changes depending on the specific goal. The colormap represents the velocity (red denotes a higher velocity).

awareness, Chapter 5 of this thesis proposes to use IL to obtain *perception-aware* trajectories that perform *obstacle avoidance in dynamic* environments.

When performing obstacle avoidance, capturing the multimodality of the trajectory planning problem is crucial to reduce the conservativeness. Indeed, for a given scenario, there may be $n_e \geq 1$ locally-optimal expert trajectories that avoid the obstacle(s) (e.g., see Fig. 1-3), where n_e may change between different scenarios. The use of a unimodal student that produces a single trajectory either introduces an artificial bias towards a specific direction of the space, or averages together the different expert trajectories, which can be catastrophic in obstacle avoidance scenarios. The challenge is then how to design and train a neural network capable of generating a multimodal trajectory prediction.

One possible approach is to use Mixture Density Networks to learn the parameters

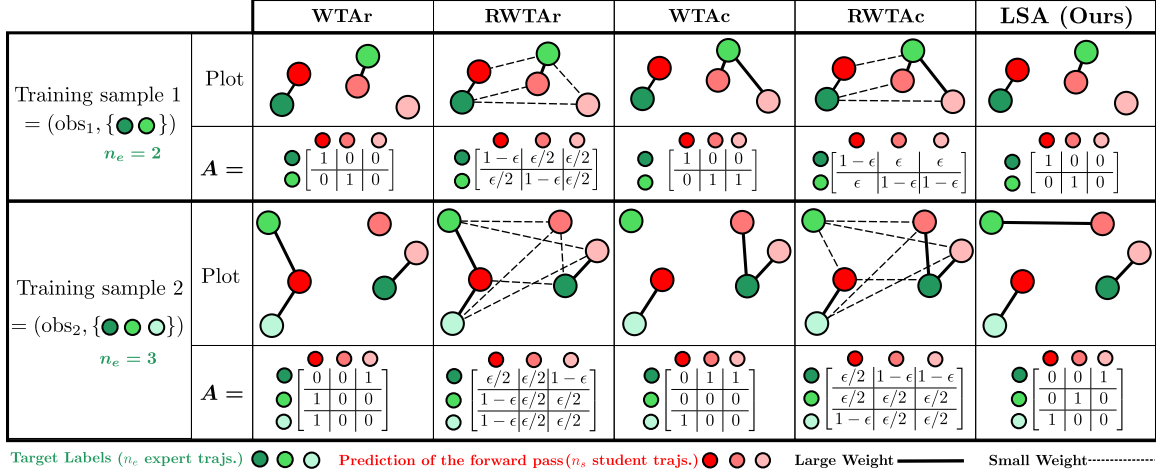


Figure 1-4: Comparison between the assignment matrix \mathbf{A} obtained by the WTAr, RWTAr, WTAc, RWTAc, and LSA approaches. This matrix \mathbf{A} is then the one used to weigh each (target, prediction) pair in the loss. In the figure, $\epsilon \geq 0$, $n_s = 3$ and obs_i denotes the observation of the training sample i . In WTAr and RWTAr, each row of \mathbf{A} sums up to 1, while in WTAc and RWTAc, each column of \mathbf{A} sums up to 1. We propose instead to obtain \mathbf{A} as the solution of the linear sum assignment (LSA) problem, which minimizes the total assignment cost, and guarantees that all the target labels have one distinct prediction assigned to them (i.e., all the rows sum up to 1, n_e columns sum up to 1, and $(n_s - n_e)$ columns sum up to 0). More visualizations of the WTAr and RWTAr assignments are available at [105, Fig. 3] and [45, Fig. 2].

of a Gaussian mixture model [16]. Mixture Density networks are however known to suffer from numerical instability and mode collapse [105, 141]. Another option is to capture the multimodality through the use of the Winner-Takes-All (WTAr or WTAc) losses [45, 58, 105] (see Fig. 1-4). These approaches use an binary assignment matrix \mathbf{A} that weighs the contribution of each (target, prediction) pair in the loss. In WTAr [45, 105], each target label is assigned to the closest prediction, while in WTAc, each prediction is assigned to the closest target label. Other works propose instead the use of the relaxed losses RWTAr [105, 141] and RWTAc [96], where the constraint of \mathbf{A} being a binary matrix is relaxed (see Fig. 1-4). These relaxed costs typically address the mode collapse problem (which happens when all the predictions of the network after training are close to the same target label), but due to the nonzero weights between all the predictions and all the target labels, the predictions of these relaxed costs may reach an equilibrium position that does not represent any of the

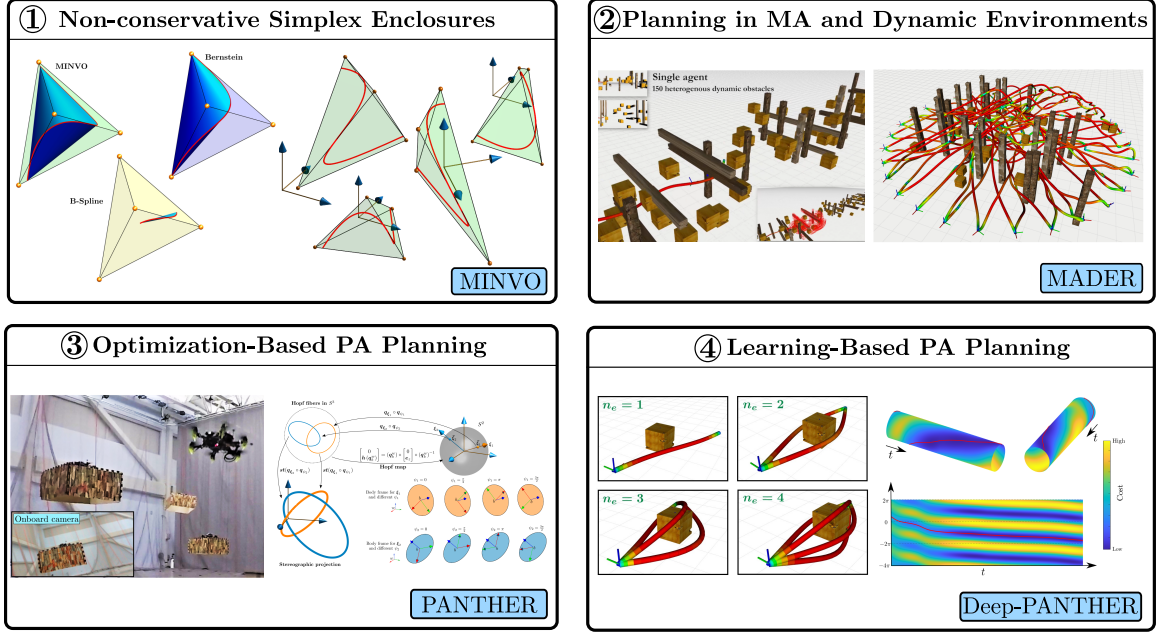


Figure 1-5: Graphical representation of the contributions of this thesis.

target labels [123].

In contrast to these approaches, and inspired by the multi-object detection and tracking algorithms [14, 23, 180], in Chapter 5 of this thesis we propose to use (in the loss) the optimal assignment matrix \mathbf{A} found by solving the linear sum assignment (LSA) problem, which minimizes the total assignment cost and guarantees that *all* the target labels are assigned to a *distinct* prediction (see Fig. 1-4). This ensures that a target label is not assigned to multiple predictions (reducing therefore the mode collapse problem) and that each prediction is not assigned to multiple target labels (being therefore less prone to equilibrium issues).

1.4 Thesis Contributions and Structure

This thesis aims to address the aforementioned gaps with the four contributions shown in Fig. 1-5 and Table 1.2. These contributions are summarized in the following subsections.

Table 1.2: Contributions of this thesis.

Contribution	Chapter	Publication
① Non-conservative Simplex Enclosures	2	[162]
② Planning in Multiagent and Dynamic Environments	3	[160]
③ Optimization-Based Perception-Aware Planning	4	[163]
④ Learning-Based Perception-Aware Planning	5	[161]

1.4.1 Non-conservative Simplex Enclosures

The first contribution of this thesis is summarized as follows:

- Formulation of the optimization problem whose minimizer is the polynomial basis that generates the smallest n -simplex that encloses **any** given n^{th} -degree polynomial curve. We show that this basis also obtains the n^{th} -degree polynomial curve with largest convex hull enclosed in **any** given n -simplex. Another formulation that imposes a specific structure on the polynomials of the basis is also presented.
- We derive high-quality feasible solutions for any $n \in \mathbb{N}$, obtaining simplexes that, for $n = 3$, are 2.36 and 254.9 times smaller than the ones obtained using the Bernstein and B-Spline bases respectively. For $n = 7$, these values increase to 902.7 and $2.997 \cdot 10^{21}$, respectively.
- Numerical global optimality (with respect to the volume) is proven for $n = 1, 2, 3$ using Sum-Of-Squares (SOS) programming, branch and bound, and moment relaxations. Numerical local optimality is proven for $n = 4$, and feasibility is guaranteed for $n \geq 5$.
- Extension to polynomial curves embedded in subspaces of higher dimensions, and to some rational curves.

1.4.2 Planning in Multiagent and Dynamic Environments

The second contribution of this thesis is summarized as follows (see also Fig. 1-1):

- Decentralized and asynchronous planning framework that solves the deconfliction between the agents by imposing as constraints the trajectories other agents have committed to, and then doing a collision check-recheck scheme to guarantee safety with respect to trajectories other agents have committed to during the optimization time.
- Collision-free constraints are imposed by using a novel polynomial basis in trajectory planning: the MINVO basis. In position space, the MINVO basis yields a volume 2.36 and 254.9 times smaller than the extensively-used Bernstein and B-Spline bases, respectively.
- Formulation of the collision-free constraints with respect to other dynamic obstacles/agents by including the planes that separate the outer polyhedral representations of each interval of every pair of trajectories as decision variables.
- Extensive simulations and comparisons with state-of-the-art baselines in cluttered environments. The results show up to a 33.9% reduction in the flight time, a 88.8% reduction in the number of stops (compared to Bernstein/B-Spline bases), shorter flight distances than centralized approaches, and shorter total times on average than synchronous decentralized approaches.

1.4.3 Optimization-Based Perception-Aware Planning

The third contribution of this thesis is summarized as follows:

- Real-time PA planning formulation that jointly optimizes the translation and the full rotation to maximize the visibility of unknown dynamic obstacles, while simultaneously avoiding them. Compared to non-PA approaches and PA decoupled approaches, our proposed coupled solution leads to a presence of the obstacle in the FOV 7.9 and 1.5 times more frequent, respectively. The success rates achieved are on average 2.98 times larger than other state-of-the-art approaches when flying in multi-obstacle dynamic environments.

- We show how the Hopf fibration can be embedded in the planning optimization to jointly optimize translation and yaw while implicitly imposing the underactuated dynamics that couples acceleration and orientation. This avoids the need to explicitly impose the dynamics of the UAV as differential constraints, while automatically guaranteeing the largest possible great-circle distance between the hovering condition and the differential flatness singularity. Dynamic obstacle avoidance constraints are imposed by leveraging the MINVO basis to reduce conservativeness.
- Extensive set of hardware experiments in unknown dynamic environments, with everything (navigation, perception, planning, and control) executed onboard the UAV, and without any prior knowledge of the trajectories or specific shape/size of the obstacles. The UAV achieves velocities of up to 5.8 m/s and relative velocities (with respect to the obstacles) of up to 6.3 m/s. The replanning times achieved onboard are ≈ 53 ms.

1.4.4 Learning-Based Perception-Aware Planning

The fourth contribution of this thesis is summarized as follows:

- Novel multimodal learning-based trajectory planning framework able to generate collision-free trajectories that avoid a dynamic obstacle while maximizing its presence in the FOV.
- Computation times two orders of magnitude faster than a multimodal optimization-based planner, while achieving a similar total cost.
- Multimodal loss that achieves a mean squared error (MSE) of the predicted trajectories with respect to the expert trajectories up to 18 times smaller than the (Relaxed) Winner-Takes-All approaches.
- The proposed approach also presents a very good generalization to environments where the obstacle is following a different trajectory than the one used in training.

1.5 Related Publications

Title	Ref.	Venue	Code/Video
Real-time planning with multi-fidelity models for agile flights in unknown environments	[165]	ICRA'19	Code , Video
FASTER: Fast and Safe Trajectory Planner for Flights in Unknown Environments ¹	[167]	IROS'19	Code , Video
FASTER: Fast and Safe Trajectory Planner for Navigation in Unknown Environments	[166]	T-RO	Code , Video
Trajectory planner for agile flights in unknown environments	[168]	Master's thesis	Code , Video
MINVO Basis: Finding Simplexes with Minimum Volume Enclosing Polynomial Curves	[162]	CAD	Code , Video
MADER: Trajectory Planner in Multiagent and Dynamic Environments ²	[160]	T-RO	Code , Video
PANTHER: Perception-Aware Trajectory Planner in Dynamic Environments	[163]	IEEE Access	Code , Video
Deep-PANTHER: Learning-Based Perception-Aware Trajectory Planner in Dynamic Environments	[161]	RA-L (in review)	Video
Autonomous Off-road Navigation over Extreme Terrains with Perceptually-challenging Conditions ³	[158]	ISER'20	Video
LION: Lidar-Inertial Observability-Aware Navigator for Vision-Denied Environments ⁴	[156]	ISER'20	Video
NeBula: Quest for Robotic Autonomy in Challenging Environments; TEAM CoSTAR at the DARPA Subterranean Challenge	[6]	JFR	Videos
Onboard Detection and Localization of Drones Using Depth Maps	[24]	IEEE Access	Video

¹ Finalist to the best paper award on Search and Rescue Robotics.

² Invited for presentation at ICRA'22.

³ 2nd place in the DARPA Subterranean Challenge (Tunnel competition).

⁴ 1st place in the DARPA Subterranean Challenge (Urban competition).




Chapter 2

MINVO Basis: Finding Simplexes with Minimum Volume Enclosing Polynomial Curves

2.1 Overview

This chapter studies the polynomial basis that generates the smallest n -simplex enclosing a given n^{th} -degree polynomial curve in \mathbb{R}^n . Although the Bernstein and B-Spline polynomial bases provide feasible solutions to this problem, the simplexes obtained by these bases are not the smallest possible, which leads to overly conservative results in many CAD (computer-aided design) applications. We first prove that the polynomial basis that solves this problem (MINVO basis) also solves for the n^{th} -degree polynomial curve with largest convex hull enclosed in a given n -simplex. Then, we present a formulation that is *independent* of the n -simplex or n^{th} -degree polynomial curve given. By using Sum-Of-Squares (SOS) programming, branch and bound, and moment relaxations, we obtain high-quality feasible solutions for any $n \in \mathbb{N}$, and prove (numerical) global optimality for $n = 1, 2, 3$ and (numerical) local optimality for $n = 4$. The results obtained for $n = 3$ show that, for *any* given 3rd-degree polynomial curve in \mathbb{R}^3 , the MINVO basis is able to obtain an enclosing

Table 2.1: Notation used in this chapter.

Symbol	Meaning
$a, \mathbf{a}, \mathbf{A}$	Scalar, column vector, matrix
$ \mathbf{A} , \text{tr}(\mathbf{A})$	Determinant of \mathbf{A} , trace of \mathbf{A}
\mathbf{t}	$[t^r \ t^{r-1} \ \dots \ 1]^T$ (r given by the context)
$\hat{\mathbf{t}}$	$[1 \ \dots \ t^{r-1} \ t^r]^T$ (r given by the context)
$\mathbb{R}[t]$	Set of univariate polynomials in t with coefficients in \mathbb{R}
$\mathbf{p}(t)$	Column vector whose coordinates are polynomials in $\mathbb{R}[t]$
P	Polynomial curve $P := \{\mathbf{p}(t) \mid t \in [-1, 1]\}$
\mathbf{P}	Coefficient matrix of P . $\mathbf{p}(t) = \mathbf{P}\mathbf{t}$
\mathcal{P}^n	Set of all possible n^{th} -degree polynomial curves
$\text{conv}(P)$	Convex hull of P
n	Maximum degree of the entries of $\mathbf{p}(t)$
k	Number of rows of $\mathbf{p}(t)$
\mathcal{M}	Subspace with the smallest dimension that contains P . $\mathcal{M} \subseteq \mathbb{R}^k$
m	Dimension of \mathcal{M}
S	Simplex
S^n	Set of all possible n -simplexes
\mathbf{V}	Matrix whose columns are the vertices of a simplex. This definition will be generalized in Section 2.6.1
$\mathbf{0}, \mathbf{1}$	Column vectors of zeros and ones
$\mathbf{a} \geq \mathbf{b}$	Element-wise inequality
s	Number of intervals the polynomial curve is subdivided into ($s = 1$ means no subdivision)
SL_h	SLEFE of a polynomial curve using h breakpoints in each interval of subdivision
\propto	Proportional to
$\partial \cdot$	Frontier of a set
$\lfloor \cdot \rfloor$	Floor function
$\text{abs}(\cdot)$	Absolute value
$\cdot_{a \times b}$	Size of a matrix (a rows \times b columns)
\mathbf{e}	$[0 \ 0 \ \dots \ 0 \ 1]^T$ (size given by the context)
\mathbf{I}_n	Identity matrix of size $n \times n$
$\mathbf{M}_{:,c:d}$	Matrix formed by columns $c, c+1, \dots, d$ of \mathbf{M}
\mathbb{S}_+^a	Positive semidefinite cone (set of all symmetric positive semidefinite $a \times a$ matrices)
$\text{vol}(\cdot)$	Volume (Lebesgue measure)
π	Hyperplane
$\text{dist}(\mathbf{a}, \pi)$	Distance between the point \mathbf{a} and the hyperplane π
$\text{odd}(a, b)$	1 if both a and b are odd, 0 otherwise
NLO, NGO	Numerical Local Optimality, Numerical Global Optimality
MV, Be, BS	MINVO, Bernstein, and B-Spline
  	Color notation for the MV, Be, and BS bases respectively

simplex whose volume is 2.36 and 254.9 times smaller than the ones obtained by the Bernstein and B-Spline bases, respectively. When $n = 7$, these ratios increase to 902.7 and $2.997 \cdot 10^{21}$, respectively.

2.2 Preliminaries

2.2.1 Notation and Definitions

The notation used throughout this chapter is summarized in Table 2.1. Unless otherwise noted, all the indexes start at 0. For instance, $\mathbf{M}_{0,3}$ is the fourth element of the first row of the matrix \mathbf{M} . Let us also introduce the two following common definitions and their respective notations:

Polynomial curve P of degree n and dimension k :

$$P := \{\mathbf{p}(t) \mid t \in [a, b], a \in \mathbb{R}, b \in \mathbb{R}, b > a\}$$

where $\mathbf{p}(t) := [p_0(t) \cdots p_{k-1}(t)]^T := \mathbf{P}t \in \mathbb{R}^k$, $p_i(t)$ is a polynomial in $\mathbb{R}[t]$, and $n \in \mathbb{N}$ is the highest degree of all the $p_i(t)$. The $k \times (n+1)$ matrix \mathbf{P} is the coefficient matrix. Without loss of generality we will use the interval $t \in [-1, 1]$ (i.e., $a = -1$ and $b = 1$), and assume that the parametrization of $\mathbf{p}(t)$ has minimal degree (i.e., no other polynomial parametrization with lower degree produces the same spatial curve P). The subspace containing P and that has the smallest dimension will be denoted as $\mathcal{M} \subseteq \mathbb{R}^k$, and its dimension will be m . We will work with the case $n = m = k$, and refer to such curves simply as **n^{th} -degree polynomial curves**. Note that we will use the term *polynomial curve* to refer to a curve with only one segment, and not to a curve with several polynomial segments. The set of all possible n^{th} -degree polynomial curves will be denoted as \mathcal{P}^n . Section 2.6 will then extend the results to curves with arbitrary n , m and k .

n -simplex: Convex hull of $n + 1$ affinely independent points $\mathbf{v}_0, \dots, \mathbf{v}_n \in \mathbb{R}^n$. These points are the **vertices** of the simplex, and will be stacked in the **matrix of vertices** $\mathbf{V} := [\mathbf{v}_0 \dots \mathbf{v}_n]$. The letter S will denote a particular simplex, while \mathcal{S}^n will denote the set of all possible n -simplexes. A simplex with $\mathbf{V} = [\mathbf{0} \ \mathbf{I}_n]$ will be called the **standard n -simplex**.

We will use the basis matrix of a segment of a non-clamped uniform B-Spline for comparison [34, 133, 145], and refer to this basis simply as the **B-Spline basis**.

Moreover, throughout this chapter we will use the term *numerical local optimality* (NLO) to classify a solution for which the first-order optimality measure and the maximum constraint violation are smaller than a predefined small tolerance [4]. Similarly, we will use the term *numerical global optimality* (NGO) to classify a feasible solution for which the difference between its objective value and a lower bound on the global minimum (typically obtained via a relaxation of the problem) is less than a specific small tolerance. All the tolerances and parameters of the numerical solvers used are available in the [code](#).

Finally, and for purposes of clarity, we will use the term *MINVO basis* to refer to both the global minimizers of the problem (proved for $n = 1, 2, 3$) and the proposed locally-optimal/feasible solutions (obtained for $n \geq 4$).

2.2.2 Volume of the Convex Hull of a Polynomial Curve

At several points throughout the chapter, we will make use of the following theorem, that we prove in Appendix 2.A:

Theorem 1: The volume of the convex hull of $P \in \mathcal{P}^n$ ($t \in [-1, 1]$), is given by:

$$\text{vol}(\text{conv}(P)) = \frac{\text{abs}(|\mathbf{P}_{:,0:n-1}|)}{n!} 2^{\frac{n(n+1)}{2}} \prod_{0 \leq i < j \leq n} \binom{j-i}{j+i}$$

Proof: See Appendix 2.A. □

Note that, as the curve P satisfies $n = m = k$ (see Section 2.2.1), the volume of its convex hull is nonzero and therefore $|\mathbf{P}_{:,0:n-1}| \neq 0$.

2.3 Problems Definition

The goal of this chapter is to find the smallest simplex $S \in \mathcal{S}^n$ enclosing a given polynomial curve $P \in \mathcal{P}^n$, and to find the polynomial curve $P \in \mathcal{P}^n$ with largest convex hull enclosed in a given simplex $S \in \mathcal{S}^n$.

2.3.1 Given $P \in \mathcal{P}^n$, find $S \in \mathcal{S}^n$

Problem 1: Given the polynomial curve $P \in \mathcal{P}^n$, find the simplex $S \in \mathcal{S}^n$ with minimum volume that contains P . In other words:

$$\begin{aligned} \min_{S \in \mathcal{S}^n} \quad & \text{vol}(S) \\ \text{s.t.} \quad & P \subset S \end{aligned}$$

For $n = 2$, Problem 1 tries to find the triangle with the smallest area that contains a planar 2nd-degree polynomial curve. For $n = 3$, it tries to find the tetrahedron with the smallest volume that contains a 3rd-degree polynomial curve in 3D. Similar geometric interpretations apply for higher n .

Letting f_1 denote the objective function of this problem, we have that $f_1 := \text{vol}(S) \propto \text{abs}\left(\left| \begin{bmatrix} \mathbf{V}^T \\ \mathbf{1} \end{bmatrix} \right|\right)$. Note that, as the volume of the convex hull of P is nonzero (see Section 2.2.2), then it is guaranteed that $\left| \begin{bmatrix} \mathbf{V}^T \\ \mathbf{1} \end{bmatrix} \right| \neq 0$.

2.3.2 Given $S \in \mathcal{S}^n$, find $P \in \mathcal{P}^n$

Problem 2: Given a simplex $S \in \mathcal{S}^n$, find the polynomial curve $P \in \mathcal{P}^n$ contained in S , whose convex hull has maximum volume:

$$\begin{aligned} \min_{P \in \mathcal{P}^n} \quad & -\text{vol}(\text{conv}(P)) \\ \text{s.t.} \quad & P \subset S \end{aligned}$$

By the definition of a simplex (see Section 2.2.1), its vertices are affinely

independent and therefore the matrix of vertices of the given simplex S satisfies $\left| \left[\mathbf{V}^T \mathbf{1} \right] \right| \neq 0$.

Letting f_2 denote the objective function of this problem, we have that $f_2 := -\text{vol}(\text{conv}(P)) \propto -\text{abs}(|\mathbf{P}_{:,0:n-1}|)$. Now note that the optimal solution for this problem is guaranteed to satisfy $|\mathbf{P}_{:,0:n-1}| \neq 0$, which can be easily proven by noting that we are maximizing the absolute value of $|\mathbf{P}_{:,0:n-1}|$, and that there exists at least one feasible solution (for example the Bézier curve whose control points are the vertices of S) with $|\mathbf{P}_{:,0:n-1}| \neq 0$.

2.4 Equivalent Formulation

Let us now study the constraints and the objective functions of Problems 1 and 2.

2.4.1 Constraints of Problems 1 and 2

Both problems share the same constraint $P \subset S$ (i.e., $\mathbf{p}(t) \in S \ \forall t \in [-1, 1]$), which is equivalent to $\mathbf{p}(t)$ being a convex combination of the vertices \mathbf{v}_i of the simplex for $t \in [-1, 1]$:

$$P \subset S \equiv \begin{cases} \mathbf{p}(t) = \sum_{i=0}^n \lambda_i(t) \mathbf{v}_i \\ \sum_{i=0}^n \lambda_i(t) = 1 \quad \forall t \\ \lambda_i(t) \geq 0 \quad \forall i = 0, \dots, n \quad \forall t \in [-1, 1] \end{cases} \quad (2.1)$$

The variables $\lambda_i(t)$ are usually called barycentric coordinates [13,40,62], and their geometric interpretation is as follows. Let us first define $\boldsymbol{\pi}_i$ as the hyperplane that passes through the points $\{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_n\} \setminus \{\mathbf{v}_i\}$, and \mathbf{n}_i as its normal vector that points towards the interior of the simplex. Choosing now $q \in \{0, \dots, n\} \setminus \{i\}$, and using the fact that $\sum_{j=0}^n \lambda_j(t) = 1$, we have that $\mathbf{p}(t) = \mathbf{v}_q + \sum_{j=0}^n \lambda_j(t) (\mathbf{v}_j - \mathbf{v}_q)$. Therefore:

$$\text{dist}(\mathbf{p}(t), \boldsymbol{\pi}_i) := (\mathbf{p}(t) - \mathbf{v}_q)^T \mathbf{n}_i = \sum_{j=0}^n \lambda_j(t) (\mathbf{v}_j - \mathbf{v}_q)^T \mathbf{n}_i =$$

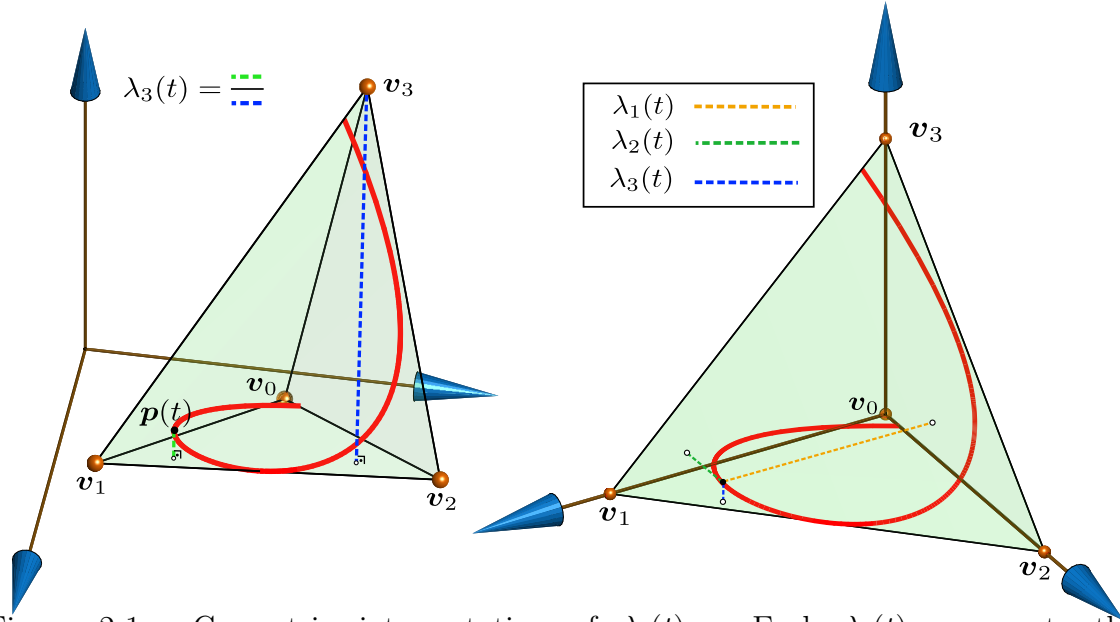


Figure 2-1: Geometric interpretation of $\lambda_i(t)$. Each $\lambda_i(t)$ represents the distance between the curve $\mathbf{p}(t)$ and the hyperplane formed by the vertices $\{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_n\} \setminus \{\mathbf{v}_i\}$, divided by the distance from the vertex \mathbf{v}_i to that hyperplane (left). For the standard 3-simplex in 3D (i.e., $\mathbf{V} = [\mathbf{0} \ \mathbf{I}_3]$), the curve in red has $\mathbf{p}(t) = [\lambda_1(t) \ \lambda_2(t) \ \lambda_3(t)]^T$ (right).

$$= \lambda_i(t) (\mathbf{v}_i - \mathbf{v}_q)^T \mathbf{n}_i = \lambda_i(t) \text{dist}(\mathbf{v}_i, \boldsymbol{\pi}_i) ,$$

which implies that

$$\lambda_i(t) = \frac{\text{dist}(\mathbf{p}(t), \boldsymbol{\pi}_i)}{\text{dist}(\mathbf{v}_i, \boldsymbol{\pi}_i)} . \quad (2.2)$$

Hence, $\lambda_i(t)$ represents the ratio between the distance from the point $\mathbf{p}(t)$ of the curve to the hyperplane $\boldsymbol{\pi}_i$ and the distance from \mathbf{v}_i to that hyperplane $\boldsymbol{\pi}_i$ (see Fig. 2-1 for the case $n = 3$).¹ From Eq. 2.2 it is clear that each $\lambda_i(t)$ is an n^{th} -degree polynomial, that we will write as $\lambda_i(t) := \boldsymbol{\lambda}_i^T \mathbf{t}$, where $\boldsymbol{\lambda}_i$ is its vector of coefficients. Matching now the coefficients of $\mathbf{p}(t)$ with the ones of $\sum_{i=0}^n \lambda_i(t) \mathbf{v}_i$, the first constraint

¹Note that multiplying numerator and denominator of Eq. 2.2 by the area of the facet that lies on the plane $\boldsymbol{\pi}_i$, each $\lambda_i(t)$ can also be defined as a ratio of volumes, as in [40].

of Eq. 2.1 can be rewritten as $\mathbf{P} = \mathbf{V}\mathbf{A}$, where

$$\mathbf{A} := \begin{bmatrix} \boldsymbol{\lambda}_0^T \\ \boldsymbol{\lambda}_1^T \\ \vdots \\ \boldsymbol{\lambda}_n^T \end{bmatrix} = \left[\boldsymbol{\lambda}_0 \ \boldsymbol{\lambda}_1 \ \cdots \ \boldsymbol{\lambda}_n \right]^T .$$

Note that \mathbf{A} is a $(n+1) \times (n+1)$ matrix whose i^{th} row contains the coefficients of the polynomial $\lambda_i(t)$ in decreasing order. The second and third constraints of Eq. 2.1 can be rewritten as $\mathbf{A}^T \mathbf{1} = \mathbf{e}$ and $\mathbf{A}\mathbf{t} \geq 0 \ \forall t \in [-1, 1]$ respectively. We conclude therefore that

$$P \subset S \equiv \begin{cases} \mathbf{P} = \mathbf{V}\mathbf{A} \\ \mathbf{A}^T \mathbf{1} = \mathbf{e} \\ \mathbf{A}\mathbf{t} \geq 0 \ \forall t \in [-1, 1] \end{cases} . \quad (2.3)$$

2.4.2 Objective Function of Problem 1

Using the constraints in Eq. 2.3, and noting that the matrix \mathbf{A} is invertible (as proven in Appendix 2.B), we can write

$$f_1 \propto \text{abs} \left(\left| \left[\mathbf{V}^T \mathbf{1} \right] \right| \right) = \text{abs} \left(\left| \mathbf{A}^{-T} \left[\mathbf{P}^T \mathbf{e} \right] \right| \right) \propto \text{abs} \left(\left| \mathbf{A}^{-1} \right| \right) = \frac{1}{\text{abs}(|\mathbf{A}|)},$$

where we have used the fact that everything inside $\left[\mathbf{P}^T \mathbf{e} \right]$ is given (i.e., not a decision variable of the optimization problem), and the fact that $|\mathbf{A}| = |\mathbf{A}^T|$. We can therefore minimize $-\text{abs}(|\mathbf{A}|)$. Note that now the objective function f_1 is **independent** of the given curve P .

2.4.3 Objective Function of Problem 2

Similar to the previous subsection, and noting that \mathbf{V} is given in Problem 2, we have that

$$f_2 \propto -\text{abs}(|\mathbf{P}_{:,0:n-1}|) = -\text{abs} \left(\left\| \begin{bmatrix} \mathbf{P} \\ \mathbf{e}^T \end{bmatrix} \right\| \right) = -\text{abs} \left(\left\| \begin{bmatrix} \mathbf{V} \\ \mathbf{1}^T \end{bmatrix} \mathbf{A} \right\| \right) \propto -\text{abs}(|\mathbf{A}|) ,$$

and therefore the objective function f_2 is now **independent** of the given simplex S .

2.4.4 Equivalent Formulation for Problems 1 and 2

Note that now the dependence on the given polynomial curve (for Problem 1) or on the given simplex (for Problem 2) appears only in the constraint $\mathbf{P} = \mathbf{V}\mathbf{A}$. As \mathbf{A} is invertible (see Appendix 2.B), we can safely remove this constraint from the optimization, leaving \mathbf{A} as the only decision variable, and then use $\mathbf{P} = \mathbf{V}\mathbf{A}$ to obtain \mathbf{V} (for Problem 1) or \mathbf{P} (for Problem 2). We end up therefore with the following optimization problem:²

Problem 3:

$$\begin{aligned} \min_{\mathbf{A} \in \mathbb{R}^{(n+1) \times (n+1)}} \quad & -\text{abs}(|\mathbf{A}|) \\ \text{s.t.} \quad & \mathbf{A}^T \mathbf{1} = \mathbf{e} \\ & \mathbf{A} \mathbf{t} \geq \mathbf{0} \quad \forall t \in [-1, 1] \end{aligned}$$

Remark: As detailed above, Problem 3 does not depend on the specific given n^{th} -degree polynomial curve (for Problem 1) or on the specific given n -simplex (for Problem 2). Hence, its optimal solution \mathbf{A}^* for a specific n can be applied to obtain the optimal solution of Problem 1 for **any** given polynomial curve $P \in \mathcal{P}^n$ (by using $\mathbf{V}^* = \mathbf{P}(\mathbf{A}^*)^{-1}$) and to obtain the optimal solution of Problem 2 for **any** given simplex $S \in \mathcal{S}^n$ (by using $\mathbf{P}^* = \mathbf{V}\mathbf{A}^*$).

²Note that in the objective function of Problem 3 the $\text{abs}(\cdot)$ is not necessary, since any permutation of the rows of \mathbf{A} will change the sign of $|\mathbf{A}|$. We keep it simply for consistency purposes, since later in the solutions we will show a specific order of the rows of \mathbf{A} for which (for some n) $|\mathbf{A}| < 0$, but that allows us to highlight the similarities and differences between this matrix and the ones the Bernstein and B-Spline bases use.

As the objective function of Problem 3 is the determinant of the nonsymmetric matrix \mathbf{A} , it is clearly a nonconvex problem. We can rewrite the constraint $\mathbf{A}\mathbf{t} \geq \mathbf{0} \quad \forall t \in [-1, 1]$ of Problem 3 using Sum-Of-Squares programming [17]:

- If n is odd, $\lambda_i(t) \geq 0 \quad \forall t \in [-1, 1]$ if and only if

$$\begin{cases} \lambda_i(t) = \hat{\mathbf{t}}^T ((t+1)\mathbf{W}_i + (1-t)\mathbf{V}_i) \hat{\mathbf{t}} \\ \mathbf{W}_i \in \mathbb{S}_+^{\frac{n+1}{2}}, \mathbf{V}_i \in \mathbb{S}_+^{\frac{n+1}{2}} \end{cases}$$

- If n is even, $\lambda_i(t) \geq 0 \quad \forall t \in [-1, 1]$ if and only if

$$\begin{cases} \lambda_i(t) = \hat{\mathbf{t}}^T \mathbf{W}_i \hat{\mathbf{t}} + \hat{\mathbf{t}}^T (t+1)(1-t)\mathbf{V}_i \hat{\mathbf{t}} \\ \mathbf{W}_i \in \mathbb{S}_+^{\frac{n}{2}+1}, \mathbf{V}_i \in \mathbb{S}_+^{\frac{n}{2}} \end{cases}$$

Note that the *if and only if* condition applies because $\lambda_i(t)$ is a univariate polynomial [17]. The decision variables would be the positive semidefinite matrices \mathbf{W}_i and \mathbf{V}_i , $i = 0, \dots, n$. Another option is to use the Markov–Lukács Theorem ([153, Theorem 1.21.1],[77, Theorem 2.2],[138]):

- If n is odd, $\lambda_i(t) \geq 0 \quad \forall t \in [-1, 1]$ if and only if

$$\lambda_i(t) = (t+1)g_i^2(t) + (1-t)h_i^2(t).$$

- If n is even, $\lambda_i(t) \geq 0 \quad \forall t \in [-1, 1]$ if and only if

$$\lambda_i(t) = g_i^2(t) + (t+1)(1-t)h_i^2(t).$$

where $g_i(t)$ and $h_i(t)$ are polynomials of degrees $\deg(g_i(t)) \leq \lfloor n/2 \rfloor$ and $\deg(h_i(t)) \leq \lfloor (n-1)/2 \rfloor$. The decision variables would be the coefficients of the polynomials $g_i(t)$ and $h_i(t)$, $i = 0, \dots, n$. In Appendix 2.C we derive the Karush–Kuhn–Tucker (KKT) conditions of Problem 3 using this theorem.

Regardless of the choice of the representation of the constraint $\mathbf{A}\mathbf{t} \geq \mathbf{0} \quad \forall t \in [-1, 1]$ (SOS or the Markov–Lukács Theorem), no generality has been lost so far.

However, these formulations easily become intractable for large n due to the high number of decision variables. We can reduce the number of decision variables of Problem 3 by imposing a structure in $\lambda_i(t)$. As Problem 1 is trying to minimize the volume of the simplex, we can impose that the facets of the n -simplex be tangent to several internal points $\mathbf{p}(t)$ of the curve (with $t \in (-1, 1)$), and in contact with the first and last points of the curve ($\mathbf{p}(-1)$ and $\mathbf{p}(1)$) [75,185]. Using the geometric interpretation of the $\lambda_i(t)$ given in Section 2.4.1, this means that each $\lambda_i(t)$ should have either real double roots in $t \in (-1, 1)$ (where the curve is tangent to a facet), and/or roots at $t \in \{-1, 1\}$. These conditions, together with an additional symmetry condition between different $\lambda_i(t)$, translate into the formulation shown in Problem 4, in which the decisions variables are the roots of $\lambda_i(t)$ and the coefficients b_i .

Problem 4:

$$\min_{\mathbf{A} \in \mathbb{R}^{(n+1) \times (n+1)}} -\text{abs}(|\mathbf{A}|) \quad \text{subject to:}$$

If n is odd:

$$\begin{aligned} \lambda_i(t) &= -b_i(t-1) \prod_{j=1}^{\frac{n-1}{2}} (t-t_{ij})^2 & i = 0, 2, \dots, n-1 \\ \lambda_i(t) &= \lambda_{n-i}(-t) & i = 1, 3, \dots, n \\ b_i &\geq 0 & i = 0, 2, \dots, n-1 \\ \mathbf{A}^T \mathbf{1} &= \mathbf{e} \end{aligned}$$

If n is even:

$$\begin{aligned} \lambda_i(t) &= -b_i(t+1)(t-1) \prod_{j=1}^{\frac{n-2}{2}} (t-t_{ij})^2 & i \text{ odd integer} \in [0, n/2-1] \\ \lambda_i(t) &= b_i \prod_{j=1}^{\frac{n}{2}} (t-t_{ij})^2 & i \text{ even integer} \in [0, n/2-1] \\ \lambda_i(t) &= \lambda_{n-i}(-t) & i = n/2+1, \dots, n \\ b_i &\geq 0 & i = 0, 1, \dots, n/2 \\ \mathbf{A}^T \mathbf{1} &= \mathbf{e} \end{aligned}$$

$$\begin{aligned} \lambda_i(t) &= -b_i(t+1)(t-1) \prod_{j=1}^{\frac{n-2}{4}} (t-t_{ij})^2 (t+t_{ij})^2 & i = n/2, i \text{ odd} \\ \lambda_i(t) &= b_i \prod_{j=1}^{\frac{n}{4}} (t-t_{ij})^2 (t+t_{ij})^2 & i = n/2, i \text{ even} \end{aligned}$$

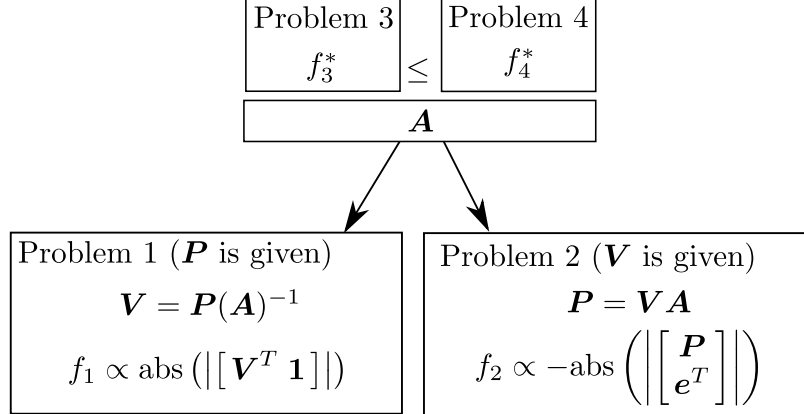


Figure 2-2: Relationship between Problems 1, 2, 3, and 4: Problem 3 and 4 have the same objective function, but the feasible set of Problem 4 is contained in the feasible set of Problem 3, and therefore $f_3^* \leq f_4^*$. Both Problem 3 and 4 generate a solution \mathbf{A} , which can be applied to **any** polynomial curve $P \in \mathcal{P}^n$ to find the simplex $S \in \mathcal{S}^n$ in Problem 1, or to **any** simplex $S \in \mathcal{S}^n$ to find the polynomial curve $P \in \mathcal{P}^n$ in Problem 2.

Letting f_i denote the objective function of Problem i , the relationship between Problems 1, 2, 3 and 4 is given in Fig. 2-2. First note that the constraints and structure imposed on $\lambda_i(t)$ in Problem 4 guarantee that they are nonnegative for $t \in [-1, 1]$ and that they sum up to 1. Hence the feasible set of Problem 4 is contained in the feasible set of Problem 3, and therefore, $f_3^* \leq f_4^*$ holds. The matrix \mathbf{A} found in Problem 3 or 4 can be used to find the vertices of the simplex in Problem 1 (by simply using $\mathbf{V} = \mathbf{P}(\mathbf{A})^{-1}$, where \mathbf{P} is the coefficient matrix of the polynomial curve given), or to find the coefficient matrix of the polynomial curve in Problem 2 (by using $\mathbf{P} = \mathbf{V}\mathbf{A}$, where \mathbf{V} contains the vertices of the given simplex).

2.5 Results

2.5.1 Results for $n = 1, 2, \dots, 7$

Using the nonconvex solvers *fmincon* [4] and *SNOPT* [56, 57] (with the *YALMIP* interface [90, 91]), we were able to find NLO solutions for Problem 4 for $n = 1, 2, \dots, 7$, and the same NLO solutions were found in Problem 3 for $n = 1, 2, 3, 4$. Problem 3 and 4 become intractable for $n \geq 5$ and $n \geq 8$ respectively. The optimal matrices \mathbf{A}

Table 2.2: Results for the MINVO basis. \mathbf{A}_{MV} , \mathbf{A}_{Be} and \mathbf{A}_{BS} denote the coefficient matrix of the MINVO, Bernstein, and B-Spline bases respectively ($t \in [-1, 1]$). The greater the absolute value of the determinant, the smaller the associated simplex (for Problem 1) and the larger the convex hull of the curve (for Problem 2). The matrices \mathbf{A}_{MV} found are **independent** of the given polynomial curve (in Problem 1), or of the given simplex (in Problem 2). NGO and NLO denote numerical Global/Local Optimality.

n	\mathbf{A}_{MV}	$\text{abs}(\mathbf{A}_{MV})$	$\frac{\text{abs}(\mathbf{A}_{MV})}{\text{abs}(\mathbf{A}_{Be})}$	$\frac{\text{abs}(\mathbf{A}_{MV})}{\text{abs}(\mathbf{A}_{BS})}$	Problem 3	Problem 4
1	$\begin{bmatrix} \frac{1}{2} & -1 & 1 \\ & 1 & 1 \end{bmatrix}$	0.5	1.0	1.0	NGO	NGO
2	$\begin{bmatrix} \frac{1}{8} & 3 & -2\sqrt{3} & 1 \\ & -6 & 0 & 6 \\ & 3 & 2\sqrt{3} & 1 \end{bmatrix}$	0.3248	1.299	5.196	NGO	NGO
3	$\begin{bmatrix} -0.4302 & 0.4568 & -0.02698 & 0.0004103 \\ 0.8349 & -0.4568 & -0.7921 & 0.4996 \\ -0.8349 & -0.4568 & 0.7921 & 0.4996 \\ 0.4302 & 0.4568 & 0.02698 & 0.0004103 \end{bmatrix}$	0.3319	2.360	254.9	NGO	NGO
4	$\begin{bmatrix} 0.5255 & -0.5758 & -0.09435 & 0.1381 & 0.03023 \\ -1.108 & 0.8108 & 0.9602 & -0.8108 & 0.1483 \\ 1.166 & 0 & -1.732 & 0 & 0.643 \\ -1.108 & -0.8108 & 0.9602 & 0.8108 & 0.1483 \\ 0.5255 & 0.5758 & -0.09435 & -0.1381 & 0.03023 \end{bmatrix}$	0.5678	6.057	$1.675 \cdot 10^5$	NLO (at least)	NLO (at least)
5	$\begin{bmatrix} -0.7392 & 0.7769 & 0.3302 & -0.3773 & -0.0365 & 0.04589 \\ 1.503 & -1.319 & -1.366 & 1.333 & -0.121 & 0.002895 \\ -1.75 & 0.5424 & 2.777 & -0.9557 & -1.064 & 0.4512 \\ 1.75 & 0.5424 & -2.777 & -0.9557 & 1.064 & 0.4512 \\ -1.503 & -1.319 & 1.366 & 1.333 & 0.121 & 0.002895 \\ 0.7392 & 0.7769 & -0.3302 & -0.3773 & 0.0365 & 0.04589 \end{bmatrix}$	1.6987	22.27	$1.924 \cdot 10^9$	Feasible (at least)	NLO (at least)
6	$\begin{bmatrix} 1.06 & -1.134 & -0.7357 & 0.8348 & 0.1053 & -0.1368 & 0.01836 \\ -2.227 & 2.055 & 2.281 & -2.299 & -0.08426 & 0.2433 & 0.0312 \\ 2.59 & -1.408 & -4.27 & 2.468 & 1.58 & -1.081 & 0.152 \\ -2.844 & 0 & 5.45 & 0 & -3.203 & 0 & 0.5969 \\ 2.59 & 1.408 & -4.27 & -2.468 & 1.58 & 1.081 & 0.152 \\ -2.227 & -2.055 & 2.281 & 2.299 & -0.08426 & -0.2433 & 0.0312 \\ 1.06 & 1.134 & -0.7357 & -0.8348 & 0.1053 & 0.1368 & 0.01836 \end{bmatrix}$	9.1027	117.8	$4.750 \cdot 10^{14}$	Feasible (at least)	NLO (at least)
7	$\begin{bmatrix} -1.637 & 1.707 & 1.563 & -1.682 & -0.3586 & 0.4143 & -0.006851 & 2.854 \cdot 10^{-5} \\ 3.343 & -3.285 & -3.947 & 4.173 & 0.6343 & -0.9385 & -0.02111 & 0.05961 \\ -4.053 & 2.722 & 6.935 & -4.96 & -2.706 & 2.269 & -0.2129 & 0.00535 \\ 4.478 & -1.144 & -9.462 & 2.469 & 6.311 & -1.745 & -1.312 & 0.435 \\ -4.478 & -1.144 & 9.462 & 2.469 & -6.311 & -1.745 & 1.312 & 0.435 \\ 4.053 & 2.722 & -6.935 & -4.96 & 2.706 & 2.269 & 0.2129 & 0.00535 \\ -3.343 & -3.285 & 3.947 & 4.173 & -0.6343 & -0.9385 & 0.02111 & 0.05961 \\ 1.637 & 1.707 & -1.563 & -1.682 & 0.3586 & 0.4143 & 0.006851 & 2.854 \cdot 10^{-5} \end{bmatrix}$	89.0191	902.7	$2.997 \cdot 10^{21}$	Feasible (at least)	NLO (at least)

found are shown in Table 2.2, and are denoted as \mathbf{A}_{MV} .³ Their determinants $|\mathbf{A}_{MV}|$ are also compared with the one of the Bernstein and B-Spline matrices (denoted as \mathbf{A}_{Be} and \mathbf{A}_{BS} respectively). The corresponding plots of the MINVO basis functions are shown in Fig. 2-3, together with the Bernstein, B-Spline and Lagrange bases for comparison. All of these bases satisfy $\sum_{i=0}^n \lambda_i(t) = 1$, and the MINVO, Bernstein, and B-Spline bases also satisfy $\lambda_i(t) \geq 0 \quad \forall t \in [-1, 1]$. The roots of each of the MINVO basis functions $\lambda_i(t)$ for $n = 1, \dots, 7$ are shown in Table 2.3 and plotted in Fig. 2-9.

One natural question to ask is whether the basis found constitutes a global minimizer for either Problem 3 or Problem 4. To answer this, first note that

³Note that any permutation in the rows of \mathbf{A}_{MV} will not change the objective value, since only the sign of the determinant is affected. Despite this multiplicity of solutions, we will refer to any matrix shown in Table 2.2 as, e.g., *the* optimal solution \mathbf{A}_{MV} , *the* feasible solution \mathbf{A}_{MV} , etc.

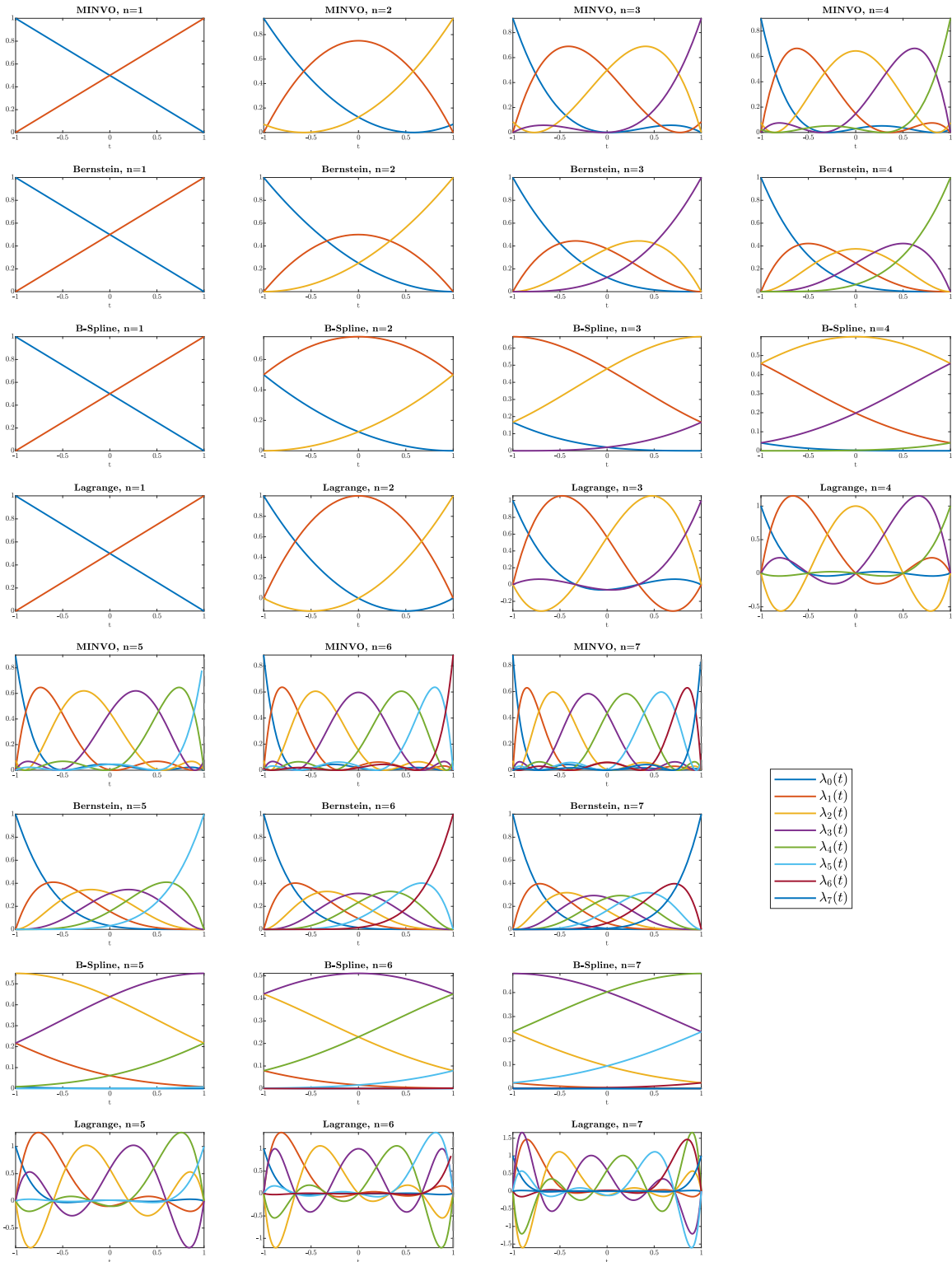


Figure 2-3: Comparison between the MINVO, Bernstein, B-Spline, and Lagrange bases for $n = 1, 2, \dots, 7$. All these bases satisfy $\sum_{i=0}^n \lambda_i(t) = 1$, and the MINVO, Bernstein, and B-Spline bases also satisfy $\lambda_i(t) \geq 0 \quad \forall t \in [-1, 1]$.

Table 2.3: Roots of each $\lambda_i(t)$ of the MINVO basis. $\mathbf{r}(\lambda_i(t))$ is the column vector that contains the roots of $\lambda_i(t)$. All the roots lying in $(-1, 1)$ are double roots, while the ones in $\{-1, 1\}$ are single roots. Each $\lambda_i(t)$ has n real roots in total. These roots are plotted in Fig. 2-9.

n	Roots of $\lambda_i(t)$, $t \in [-1, 1]$	
1	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \end{bmatrix}$	$= \begin{bmatrix} 1.0 \\ -1.0 \end{bmatrix}$
2	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \\ \mathbf{r}(\lambda_2)^T \end{bmatrix}$	$= \begin{bmatrix} \frac{1}{\sqrt{3}} & & \\ -1.0 & 1.0 & \\ -\frac{1}{\sqrt{3}} & & \end{bmatrix}$
3	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \\ \mathbf{r}(\lambda_2)^T \\ \mathbf{r}(\lambda_3)^T \end{bmatrix}$	$\approx \begin{bmatrix} 0.03088 & 1.0 & & \\ -1.0 & 0.7735 & & \\ -0.7735 & 1.0 & & \\ -1.0 & -0.03088 & & \end{bmatrix}$
4	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \\ \mathbf{r}(\lambda_2)^T \\ \mathbf{r}(\lambda_3)^T \\ \mathbf{r}(\lambda_4)^T \end{bmatrix}$	$\approx \begin{bmatrix} -0.2872 & 0.835 & & & \\ -1.0 & 0.3657 & 1.0 & & \\ -0.8618 & 0.8618 & & & \\ -1.0 & -0.3657 & 1.0 & & \\ -0.835 & 0.2872 & & & \end{bmatrix}$
5	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \\ \mathbf{r}(\lambda_2)^T \\ \mathbf{r}(\lambda_3)^T \\ \mathbf{r}(\lambda_4)^T \\ \mathbf{r}(\lambda_5)^T \end{bmatrix}$	$\approx \begin{bmatrix} -0.4866 & 0.5121 & 1.0 & & & \\ -1.0 & 0.04934 & 0.8895 & & & \\ -0.9057 & 0.5606 & 1.0 & & & \\ -1.0 & -0.5606 & 0.9057 & & & \\ -0.8895 & -0.04934 & 1.0 & & & \\ -1.0 & -0.5121 & 0.4866 & & & \end{bmatrix}$
6	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \\ \mathbf{r}(\lambda_2)^T \\ \mathbf{r}(\lambda_3)^T \\ \mathbf{r}(\lambda_4)^T \\ \mathbf{r}(\lambda_5)^T \\ \mathbf{r}(\lambda_6)^T \end{bmatrix}$	$\approx \begin{bmatrix} -0.6135 & 0.2348 & 0.9137 & & & & \\ -1.0 & -0.1835 & 0.6449 & 1.0 & & & \\ -0.9317 & 0.2822 & 0.9214 & & & & \\ -1.0 & -0.6768 & 0.6768 & 1.0 & & & \\ -0.9214 & -0.2822 & 0.9317 & & & & \\ -1.0 & -0.6449 & 0.1835 & 1.0 & & & \\ -0.9137 & -0.2348 & 0.6135 & & & & \end{bmatrix}$
7	$\begin{bmatrix} \mathbf{r}(\lambda_0)^T \\ \mathbf{r}(\lambda_1)^T \\ \mathbf{r}(\lambda_2)^T \\ \mathbf{r}(\lambda_3)^T \\ \mathbf{r}(\lambda_4)^T \\ \mathbf{r}(\lambda_5)^T \\ \mathbf{r}(\lambda_6)^T \\ \mathbf{r}(\lambda_7)^T \end{bmatrix}$	$\approx \begin{bmatrix} -0.7 & 0.008364 & 0.7132 & 1.0 & & & & \\ -1.0 & -0.3509 & 0.4068 & 0.9355 & & & & \\ -0.9481 & 0.05239 & 0.7315 & 1.0 & & & & \\ -1.0 & -0.753 & 0.4399 & 0.9408 & & & & \\ -0.9408 & -0.4399 & 0.753 & 1.0 & & & & \\ -1.0 & -0.7315 & -0.05239 & 0.9481 & & & & \\ -0.9355 & -0.4068 & 0.3509 & 1.0 & & & & \\ -1.0 & -0.7132 & -0.008364 & 0.7 & & & & \end{bmatrix}$

both Problem 3 and Problem 4 are polynomial optimization problems. Therefore, we can make use of Lasserre's moment method [82], and increase the order of the moment relaxation to find tighter lower bounds of the original nonconvex polynomial optimization problem. Using this technique, we were able to obtain, for $n = 1, 2, 3$ and for Problem 4, the same objective value as the NLO solutions found before, proving therefore numerical global optimality for these cases. For Problem 3, the moment relaxation technique becomes intractable due to the high number of variables. Hence, to prove numerical global optimality in Problem 3 we instead use the branch-and-

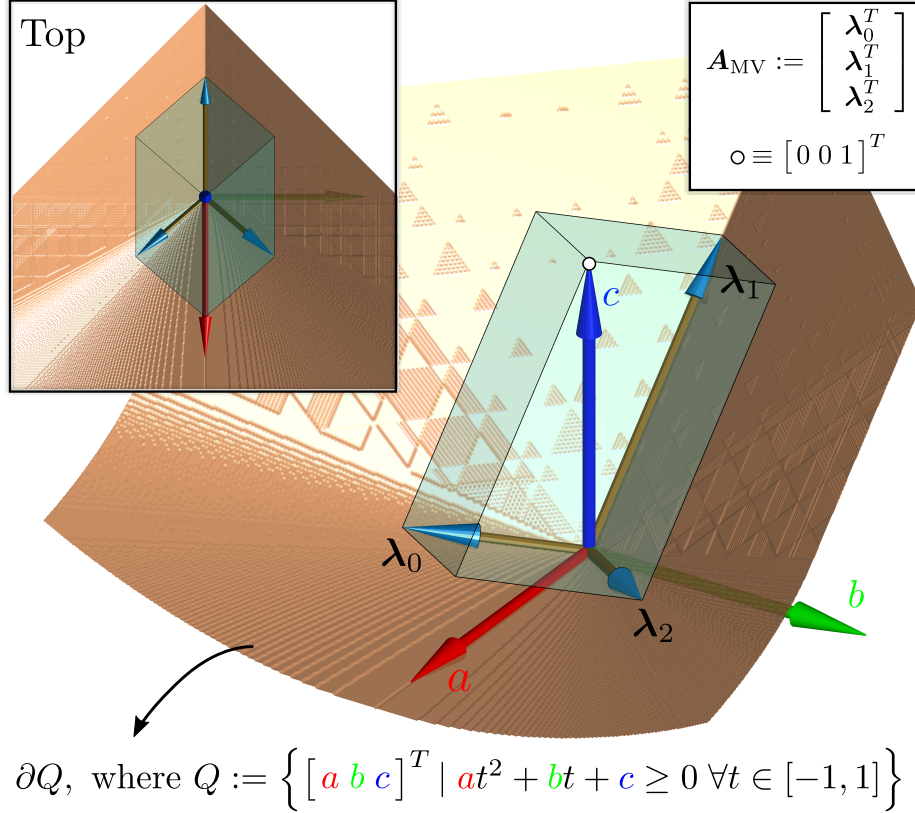


Figure 2-4: For $n = 2$, the MINVO basis has $\mathbf{A}_{\text{MV}} = [\lambda_0 \ \lambda_1 \ \lambda_2]^T$, where λ_0, λ_1 , and λ_2 are vectors that span the parallelepiped \square , and whose sum is $[001]^T$. Here, ∂Q (\square) is the frontier of the cone Q formed by the coefficients of the polynomials that are nonnegative for all $t \in [-1, 1]$. Note how the globally-optimal vectors λ_0, λ_1 , and λ_2 belong to ∂Q .

bound algorithm, which proves global optimality by reducing to zero the gap between the upper bounds found by a nonconvex solver and the lower bounds found using convex relaxations [102]. This technique proved to be tractable for cases $n = 1, 2, 3$ in Problem 3, and zero optimality gap was obtained.

All these results lead us to the following conclusions, which are also summarized in Table 2.2:

- The matrices \mathbf{A}_{MV} found for $n = 1, 2, 3$ are (numerical) global optima of both Problem 3 and Problem 4.
- The matrix \mathbf{A}_{MV} found for $n = 4$ is at least a (numerical) local optimum of both Problem 3 and Problem 4.

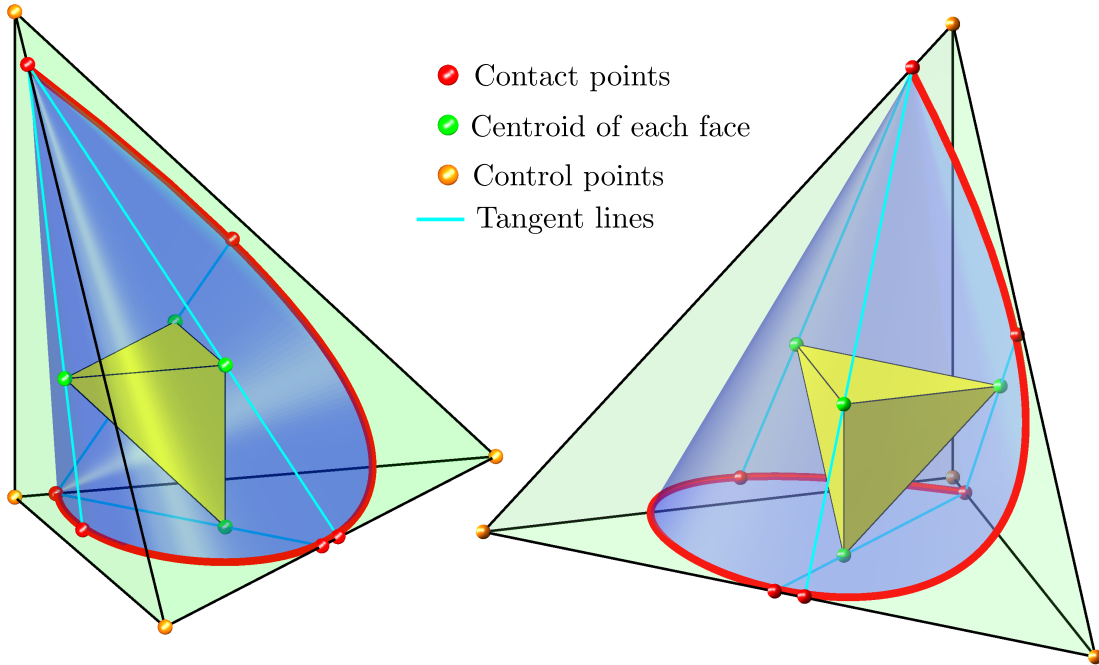
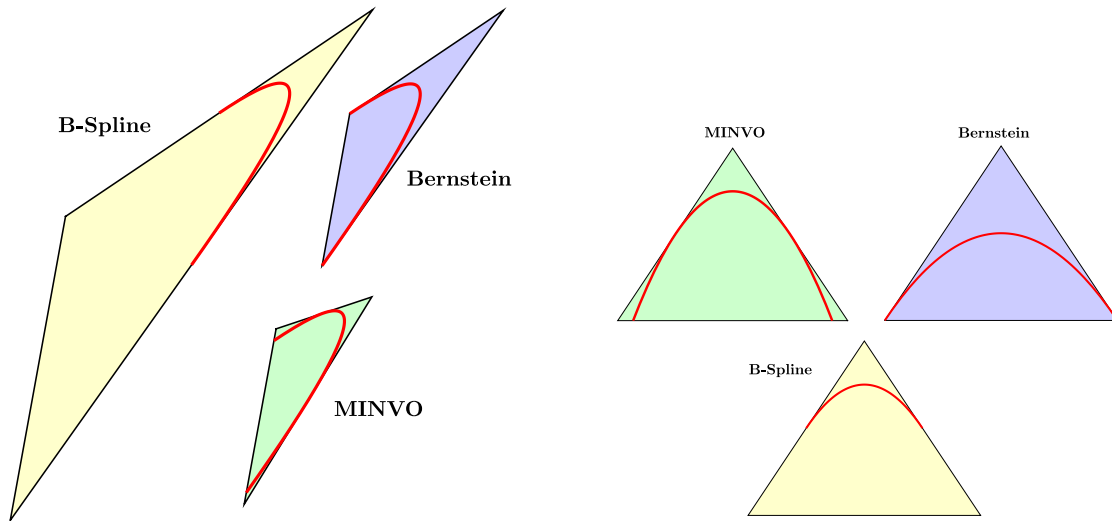


Figure 2-5: Solution obtained by the MINVO basis for $n = 3$. Note how the centroids of each of the facets (\bullet , vertices of the yellow tetrahedron) belong to $\text{conv}(P)$, which is a necessary condition for an extremal simplex [75]. Moreover, note that $\text{conv}(P)$ is tangent to the simplex along the blue lines. The red points \bullet denote the contact points between the curve and the simplex, which happen at the roots of the MINVO basis functions.

- The matrices \mathbf{A}_{MV} found for $n = 5, 6, 7$ are at least (numerical) local optima for Problem 4, and are at least feasible solutions for Problem 3.

The geometric interpretation of Problem 3 (for $n = 2$) is shown in Fig. 2-4. The rows of \mathbf{A} are vectors that lie in the cone of the polynomials that are nonnegative in $t \in [-1, 1]$ (and whose frontier is shown in orange in the figure). As Problem 3 is maximizing the volume of the parallelepiped spanned by these vectors, the optimal minimizer is obtained in the frontier of the cone, while guaranteeing that the sum of these vectors is $[0 \ 0 \ 1]^T$.

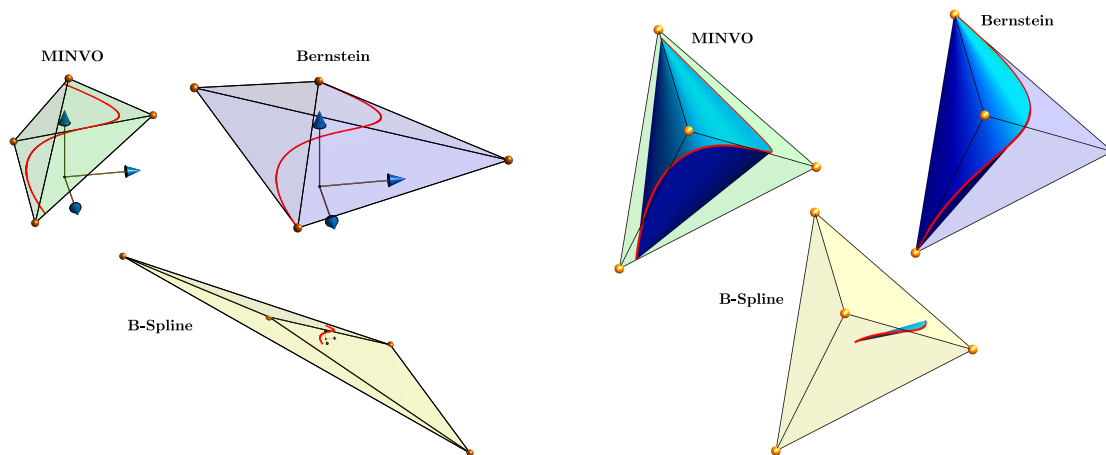
In Fig. 2-5 we check that the centroids of each of the facets of the simplex belongs to $\text{conv}(P)$, which is a necessary condition for that simplex to be minimal [75]. Note also that $\text{conv}(P)$ is tangent to the simplex along four lines (in blue in the figure), and that the contact points of the curve with the simplex happen at the roots of the



(a) For any given curve $P \in \mathcal{P}^2$, the MINVO basis finds an enclosing 2-simplex that is 1.3 and 5.2 times smaller than the one found by the Bernstein and B-Spline bases respectively.

(b) For any given 2-simplex, the MINVO basis finds a curve $P \in \mathcal{P}^2$ inscribed in the simplex, and whose convex hull is 1.3 and 5.2 times larger than the one found by the Bernstein and B-Spline bases respectively.

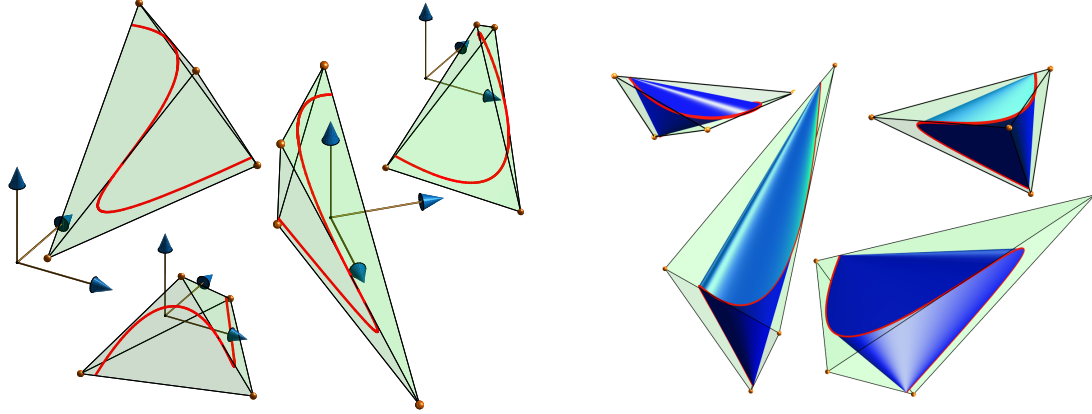
Figure 2-6: Comparison between the MINVO, Bernstein, and B-Spline bases for $n = 2$. The MINVO basis obtains numerically globally optimal results for $n = 1, 2, 3$.



(a) For **any** given 3rd-degree polynomial curve, the MINVO basis finds an enclosing 3-simplex that is 2.36 and 254.9 times smaller than the one found by the Bernstein and B-Spline bases respectively.

(b) For **any** given 3-simplex, the MINVO basis finds a 3rd-degree polynomial curve inscribed in the simplex, and whose convex hull is 2.36 and 254.9 times larger than the one found by the Bernstein and B-Spline bases respectively.

Figure 2-7: Comparison between the MINVO, Bernstein, and B-Spline bases for $n = 3$. The MINVO basis obtains numerically globally optimal results for $n = 1, 2, 3$.



(a) Simplexes found by the MINVO basis for four different given 3rd-degree polynomial curves (Problem 1).

(b) Polynomial curves (and their convex hulls in blue) obtained using the MINVO basis for four different given 3-simplexes (Problem 2).

Figure 2-8: MINVO results for $n = 3$, where numerical global optimality is guaranteed.

MINVO basis functions.

When the polynomial curve is given (i.e., Problem 1), the ratio between the volume of the simplex S_α obtained by a basis α and the volume of the simplex S_β obtained by a basis β ($\alpha, \beta \in \{\text{MV}, \text{Be}, \text{BS}\}$) is given by

$$\frac{\text{vol}(S_\alpha)}{\text{vol}(S_\beta)} = \frac{\text{abs}(|\mathbf{A}_\beta|)}{\text{abs}(|\mathbf{A}_\alpha|)}.$$

Similarly, when the simplex is given (i.e., Problem 2), the ratio between the volume of the convex hull of the polynomial curve P_α found by a basis α and the volume of the convex hull of the polynomial curve P_β found by a basis β ($\alpha, \beta \in \{\text{MV}, \text{Be}, \text{BS}\}$) is given by

$$\frac{\text{vol}(\text{conv}(P_\alpha))}{\text{vol}(\text{conv}(P_\beta))} = \frac{\text{abs}(|\mathbf{A}_\alpha|)}{\text{abs}(|\mathbf{A}_\beta|)}.$$

These ratios are shown in Table 2.2, and they mean the following for Problem 1 (Problem 2 respectively):

- For $n = 3$, the MINVO basis finds a simplex that has a volume (a polynomial curve whose convex hull has a volume) ≈ 2.36 and ≈ 254.9 times smaller (larger) than the one the Bernstein and B-Spline bases find respectively.
- For $n = 7$, the MINVO basis finds a simplex that has a volume (a polynomial

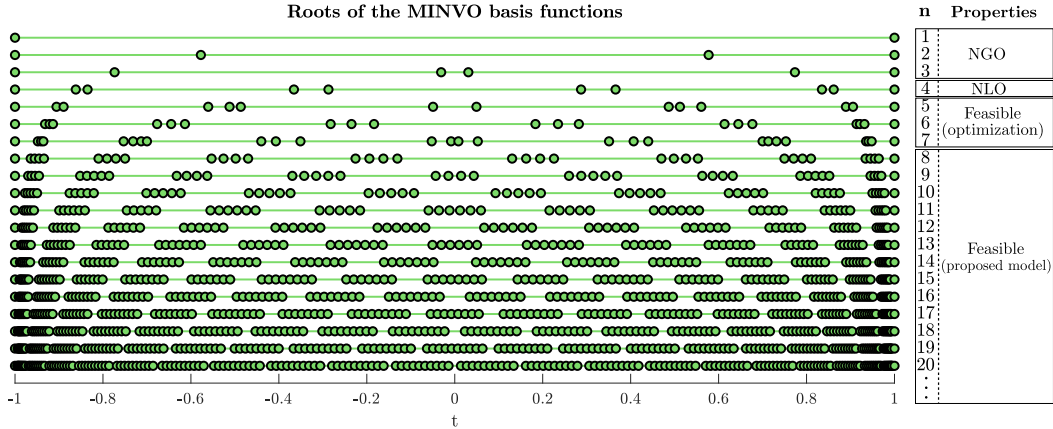


Figure 2-9: Distribution of the roots of the MINVO basis functions for different n . The results for $n \leq 7$ were obtained by solving the optimization problems (Section 2.5.1, see also Table 2.3), while the results for $n \geq 8$ were obtained using the model proposed in Section 2.5.2.

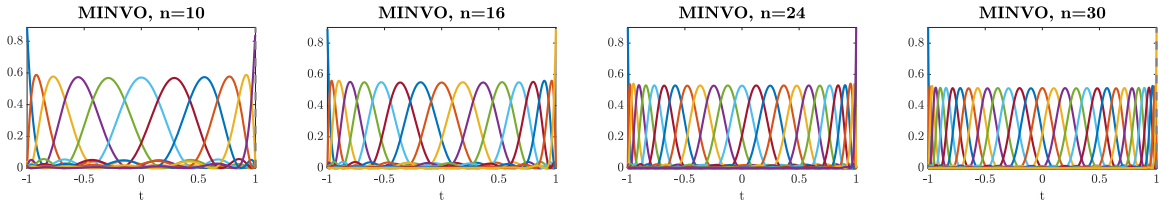


Figure 2-10: MINVO basis functions obtained for $n = 10, 16, 24, 30$ using the model proposed in Section 2.5.2.

curve whose convex hull has a volume) ≈ 902.7 and $\approx 2.997 \cdot 10^{21}$ times smaller (larger) than the one the Bernstein and B-Spline bases find respectively.

An analogous reasoning applies to the volume ratios of other n . These comparisons are shown in Fig. 2-6 (for $n = 2$), and in Fig. 2-7 (for $n = 3$). More examples of the MINVO bases applied to different polynomial curves and simplexes are shown in Fig. 2-8.

2.5.2 Results for $n > 7$

In Section 2.5.1, we obtained the results for $n = 1, \dots, 7$ by using the optimization problems. However, solving these problems becomes intractable when $n > 7$. To address this problem, we present a model that finds high-quality feasible solutions by extrapolating for $n > 7$ the pattern found for the roots of the MINVO basis (see the cases $n = 1, \dots, 7$ in Fig. 2-9). Specifically, and noting that the double roots for a

Table 2.4: Comparison of the results obtained from the optimization (available only for $n = 1, \dots, 7$) with the results obtained using the model proposed in Section 2.5.2 (available for all $n \in \mathbb{N}$). The results obtained using the proposed model are also compared with the Bernstein and B-Spline bases.

		Degree n										
		1	2	3	4	5	6	7	8	10	14	18
Opt.	$\text{abs}(\mathbf{A}_{\text{MV}})$	0.5	0.325	0.332	0.568	1.7	9.1	89.0	-	-	-	-
Proposed model	$\text{abs}(\mathbf{A}_{\text{MV}})$	0.5	0.325	0.332	0.567	1.69	9.08	88.2	1590.0	3.3e6	2.7e16	5.7e30
	$\log_{10} \text{abs} \left(\frac{ \mathbf{A}_{\text{MV}} }{ \mathbf{A}_{\text{Be}} } \right)$	0	0.114	0.373	0.782	1.35	2.07	2.95	4.0	6.57	13.6	23.2
	$\log_{10} \text{abs} \left(\frac{ \mathbf{A}_{\text{MV}} }{ \mathbf{A}_{\text{BS}} } \right)$	0	0.716	2.41	5.22	9.28	14.7	21.5	29.7	50.9	113.0	203.0

degree n tend to be distributed in n clusters, we found that the MINVO double roots in the interval $(-1, 1)$ for the degree n can be approximated by

$$\sin \left(\frac{c_0 \left(k - \frac{s_{j,n}-1}{2} \right) + c_1 \left(j - \frac{n-1}{2} \right)}{n + c_2} \right), \quad (2.4)$$

where $s_{j,n} := \lfloor \frac{n+\text{odd}(j,n)}{2} \rfloor$ models the number of roots per cluster $j \in \{0, \dots, (n-1)\}$, and $k \in \{0, \dots, (s_{j,n} - 1)\}$ is the index of the root inside a specific cluster.⁴

Here, $c_0 \approx 0.2735$, $c_1 \approx 3.0385$, and $c_2 \approx 0.4779$ were found by optimizing the associated nonlinear least-squares problem. This proposed model, with only three parameters, is able to obtain a least-square residual of $5.02 \cdot 10^{-3}$ with respect to the MINVO roots lying in $(-1, 1)$ found for $n = 2, \dots, 7$. The distribution of roots generated by this proposed model ($n \geq 8$) is shown in Fig. 2-9. Each root can then be assigned to a polynomial i of the basis by simply following the same assignment pattern found for $n = 1, \dots, 7$. Then, and by solving a linear system, the polynomials can be scaled to enforce $\sum_{i=0}^n \lambda_i(t) = 1 \forall t$ (or equivalently, $\mathbf{A}^T \mathbf{1} = \mathbf{e}$). Note that this proposed model, although not guaranteed to be optimal, is guaranteed to be feasible by construction. Some examples of the MINVO basis functions are shown in Fig. 2-10.

⁴The intuition behind the design of Eq. 2.4 is as follows: The $\sin \left(\frac{\cdot}{n+} \right)$ forces every root to be in $[-1, 1]$, and makes the centers of the clusters more densely distributed near the extremes $t \in \{-1, 1\}$, and less around $t = 0$. The numerator inside the $\sin(\cdot)$ is a weighted sum of the index of the root inside the cluster (centered around $\frac{s_{j,n}-1}{2}$), and the index of the cluster (centered around $\frac{n-1}{2}$). Finally, note that, by construction, this formula enforces symmetry with respect to $t = 0$.

The comparison of $|\mathbf{A}_{MV}|$ between the proposed model and the optimization results of Section 2.5.1 is shown in Table 2.4. For $n = 1, \dots, 7$, the relative error between the objective value obtained using the optimization and the one obtained using the proposed model is always $< 9.2 \cdot 10^{-3}$. The proposed model also produces much smaller simplexes than the Bernstein and B-Spline bases.

2.6 MINVO Basis Applied to Other Curves

2.6.1 Polynomial Curves of Degree n , Dimension k , and Embedded in a Subspace \mathcal{M} of Dimension m

So far we have studied the case of $n = m = k$ (i.e., a polynomial curve of degree n and dimension $k = n$ and for which n is also the dimension of \mathcal{M} , see Section 2.2.1). The most general case would be any k, n and m , as shown in Table 2.5. In all these cases, and using the $(n + 1) \times (n + 1)$ matrix \mathbf{A}_{MV} , we can still apply the equation $\mathbf{V}_{k \times (n+1)} = \mathbf{P}_{k \times (n+1)} \mathbf{A}_{MV}^{-1}$ to obtain all the $n + 1$ MINVO control points in \mathbb{R}^k of the given curve (columns of the matrix \mathbf{V}). The convex hull of the control points is a polyhedron that is guaranteed to contain the curve because the curve is a convex combination of the control points. Note also that, when $n = m$, all the cases below the diagonal of Table 2.5 have the same optimality properties (NGO/NLO/Feasible) as the diagonal element that has the same n .

For $k = 3$, Fig. 2-11 shows a cubic curve embedded in a two-dimensional subspace ($m = 2, n = 3$), a segment embedded in a one-dimensional subspace ($m = n = 1$) and a quadratic curve embedded in a two-dimensional subspace ($m = n = 2$).

For $k = m = 2$, the comparison between the area of the convex hull of the MINVO control points (Area_{MV}) and the area of the convex hull of the Bézier control points (Area_{Be}) is shown in Fig. 2-12. Note that this ratio is constant for any polynomial curve for the case $n = 2$, but depends on the given curve for the cases $n > 2$. To generate the boxplots of Fig. 2-12, we used a total of 10^4 polynomial curves passing through $n + 1$ points randomly sampled from the square $[-1, 1]^2$. Although

Table 2.5: All the possible cases of polynomial curves of degree n , dimension k , and embedded in a subspace \mathcal{M} of dimension m . Note that $m \leq \min(k, n)$ always holds.

		Degree n				
		1	2	3	4	≥ 5
Dimension k	1	NGO	F	F	F	F
	2	NGO	NGO	F	F	F
	3	NGO	NGO	NGO	F	F
	4	NGO	NGO	NGO	NLO	F
	≥ 5	NGO	NGO	NGO	NLO	F

In all the cases (and for any m): there are $n + 1$ control points, and $\text{conv}(\text{control points})$ is a polyhedron $\subset \mathcal{M} \subseteq \mathbb{R}^k$ that encloses the curve and that has at most $n + 1$ vertices.
In and below the diagonal: When $n = m$, the polyhedron is an n -simplex embedded in \mathbb{R}^k that is at least numerically globally optimal (NGO), numerically locally optimal (NLO), or feasible (F).

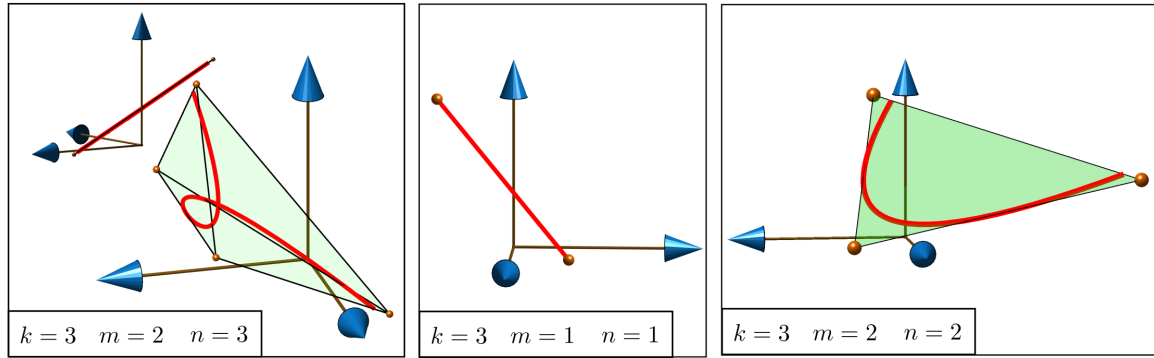


Figure 2-11: The MINVO basis applied for different curves with $k = 3$ and different values of m and n : A cubic curve embedded in a two-dimensional subspace (left), a segment embedded in a one-dimensional subspace (middle) and a quadratic curve embedded in a two-dimensional subspace (right).

it is not guaranteed that $\text{Area}_{\text{MV}} < \text{Area}_{\text{Be}}$ for any polynomial with $n > 2$, the Monte Carlo analysis performed using these random polynomial curves shows that $\text{Area}_{\text{MV}} < \text{Area}_{\text{Be}}$ holds for the great majority of them, with improvements up to ≈ 200 times for the case $n = 7$.

Similarly, the results for $k = m = 3$ are shown in Fig. 2-13, where we used a total of 10^4 polynomial curves passing through $n + 1$ points randomly sampled from the cube $[-1, 1]^3$. Again, it is not guaranteed that $\text{Vol}_{\text{MV}} < \text{Vol}_{\text{Be}}$ for any polynomial with $n > 3$, but the Monte Carlo results obtained show that this is true in most of these random polynomials. For the case $n = 7$, the MINVO basis obtains convex hulls up to ≈ 550 times smaller than the Bézier basis.

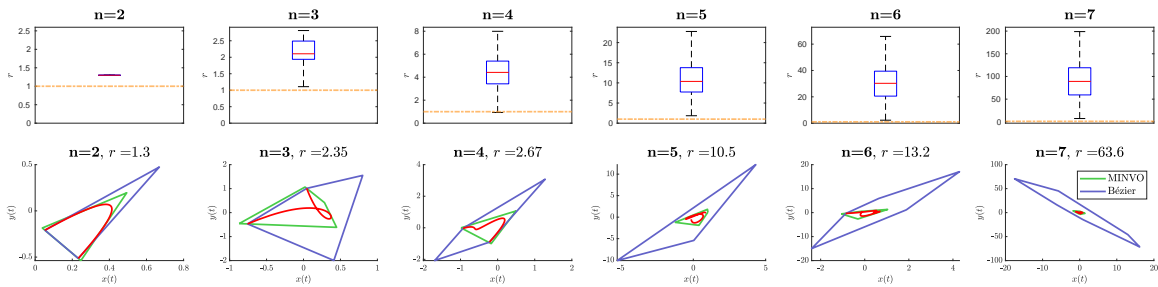


Figure 2-12: Comparison of the convex hull of the MINVO and Bézier control points for $k = m = 2$ and different n . Here r denotes the ratio of the areas $\frac{\text{Area}_{\text{Be}}}{\text{Area}_{\text{MV}}}$. The boxplots (top) have been obtained from 10^4 polynomials passing through $n+1$ random points in the square $[-1, 1]^2$. The yellow dashed line highlights the value $r = \frac{\text{Area}_{\text{Be}}}{\text{Area}_{\text{MV}}} = 1$. Some of these random curves and the associated convex hulls are shown at the bottom.

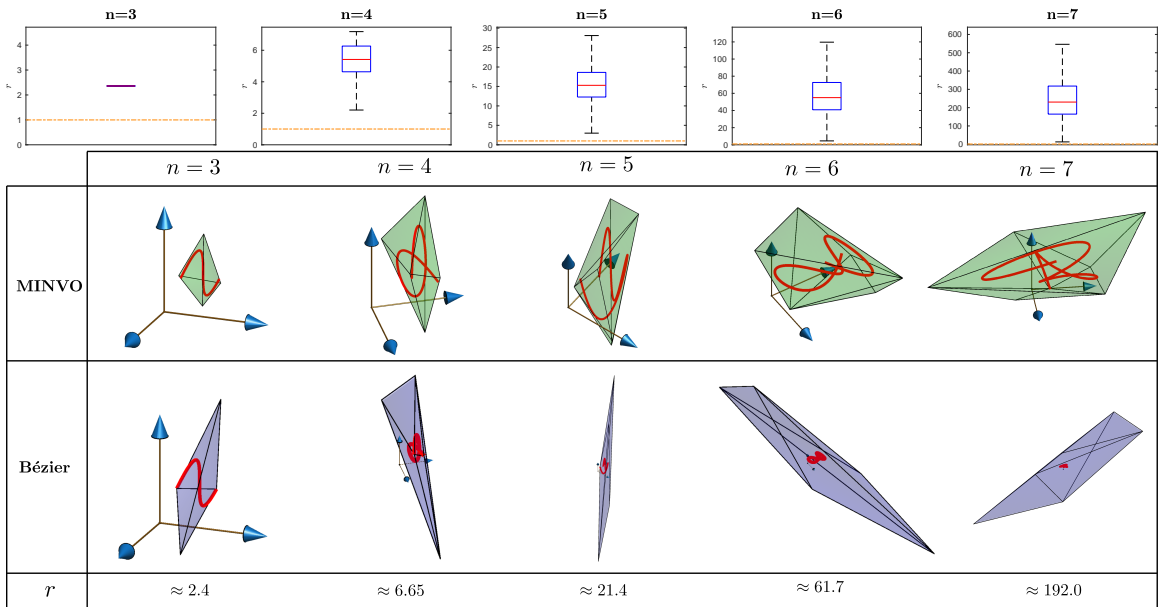


Figure 2-13: Comparison of the convex hull of the MINVO and Bézier control points for $k = m = 3$ and different n . Here r denotes the ratio of the volumes $\frac{\text{Vol}_{\text{Be}}}{\text{Vol}_{\text{MV}}}$. The boxplots (top) have been obtained from 10^4 polynomial curves passing through $n+1$ random points in the cube $[-1, 1]^3$ were used. The yellow dashed line highlights the value $r = \frac{\text{Vol}_{\text{Be}}}{\text{Vol}_{\text{MV}}} = 1$. Some of these random curves and the associated convex hulls are shown at the bottom.

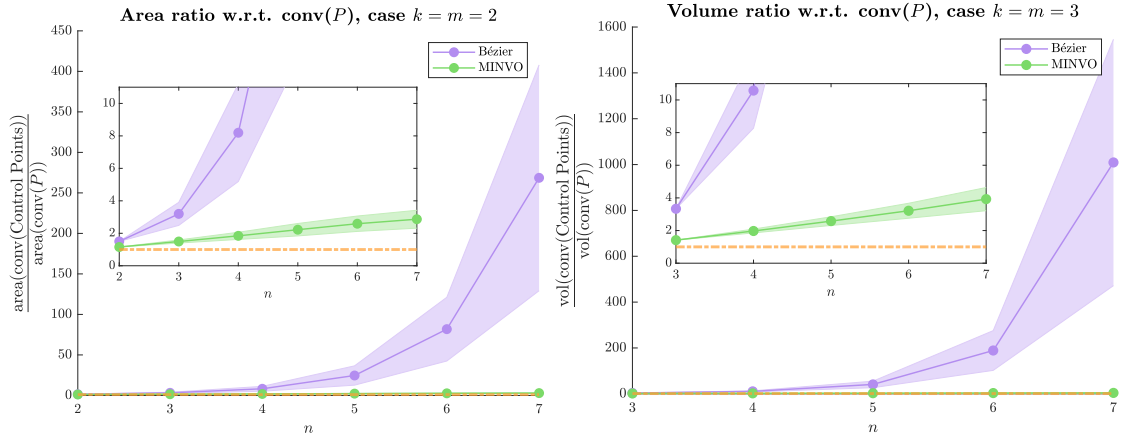


Figure 2-14: Comparison of the convex hull of the MINVO and Bézier control points with respect to $\text{conv}(P)$ for the cases $k = m = 2$ (left) and $k = m = 3$ (right). For each n , a total of 10^3 polynomial curves passing through $n + 1$ random points in the cube $[-1, 1]^k$ were used. The shaded area is the 1σ interval, where σ is the standard deviation. The yellow dashed line marks a ratio of 1. Note how the growth of the ratio for the MINVO basis is approximately linear with n , while for the Bézier basis it is approximately exponential.

Qualitatively, and for the comparisons shown above, the MINVO enclosures are much smaller than the Bézier enclosures when used in “tangled” curves. In these curves, the Bézier control points tend to be spread out and far from the curve, leading therefore to large and conservative Bézier enclosures.

Finally, we compare in Fig. 2-14 how these polyhedral convex hulls, obtained by either the MINVO or Bézier control points, approximate $\text{conv}(P)$, which is the convex hull of the curve P . Similar to the previous cases, here we used a total of 10^3 polynomial curves passing through $n + 1$ points randomly sampled from the cube $[-1, 1]^k$. The error in the MINVO outer polyhedral approximation is approximately linear as n increases, but it is exponential for the Bézier basis. For instance, when $n = 7$ and $k = 3$, the Bézier control points generate a polyhedral outer approximation that is ≈ 1010 times larger than the volume of $\text{conv}(P)$, while the polyhedral outer approximation obtained by the MINVO control points is only ≈ 3.9 times larger.

2.6.2 Rational Curves

Via projections, the MINVO basis is also able to obtain the smallest simplex that encloses some rational curves, which are curves whose coordinates are the quotients of two polynomials. For instance, given the n -simplex obtained by the MINVO basis for a given n^{th} -degree polynomial curve P , we can project every point $\mathbf{p}(t)$ of the curve via a perspective projection, using a vertex as the viewpoint, and the opposite facet as the projection plane. Note that this perspective projection of the polynomial curve will be a rational curve. If the n -simplex is the smallest one enclosing the polynomial curve, then each facet is also the smallest $(n - 1)$ -simplex that encloses the projection. This can be easily proven by contradiction, since if the facet were not a minimal $(n - 1)$ -simplex for the projected curve, then the n -simplex would not be minimal for the original 3D curve (see [144] for instance). Let us define $[\mathbf{v}_0 \dots \mathbf{v}_n] := [\mathbf{0} \ \mathbf{I}_n]$, and let π_i denote the plane that passes through the vertices $\{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_n\} \setminus \{\mathbf{v}_i\}$. Then, for a standard n -simplex, the perspective projection of $\mathbf{p}(t) := [\boldsymbol{\lambda}_1 \dots \boldsymbol{\lambda}_n]^T \mathbf{t}$ onto the plane π_i , using \mathbf{v}_i as the viewpoint, is given by:

$$\begin{cases} \frac{1}{1-\lambda_0^T \mathbf{t}} [\boldsymbol{\lambda}_1 \dots \boldsymbol{\lambda}_n]^T \mathbf{t} & \text{Projection onto } \pi_0 \\ \frac{1}{1-\lambda_i^T \mathbf{t}} [\boldsymbol{\lambda}_1 \dots \boldsymbol{\lambda}_{i-1} \ 0 \ \boldsymbol{\lambda}_{i+1} \dots \boldsymbol{\lambda}_n]^T \mathbf{t} & \text{Projection onto } \pi_i, i > 0 \end{cases}$$

This projection can also be applied successively from \mathbb{R}^i to \mathbb{R}^j ($i > j \geq 1$). Fig. 2-15 shows the case $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ (for all the four possible projections), and some projections of the cases $\mathbb{R}^6 \rightarrow \mathbb{R}^2$, $\mathbb{R}^{10} \rightarrow \mathbb{R}^2$, $\mathbb{R}^5 \rightarrow \mathbb{R}^3$, and $\mathbb{R}^{12} \rightarrow \mathbb{R}^3$.

2.7 Comparison with SLEFEs

In this section, we compare the enclosures for n^{th} -degree 2D polynomial curves obtained using these three techniques with and without subdivision:

- **MINVO**: The curve is divided into s subintervals, and then the MINVO enclosure for each of the subintervals is computed.

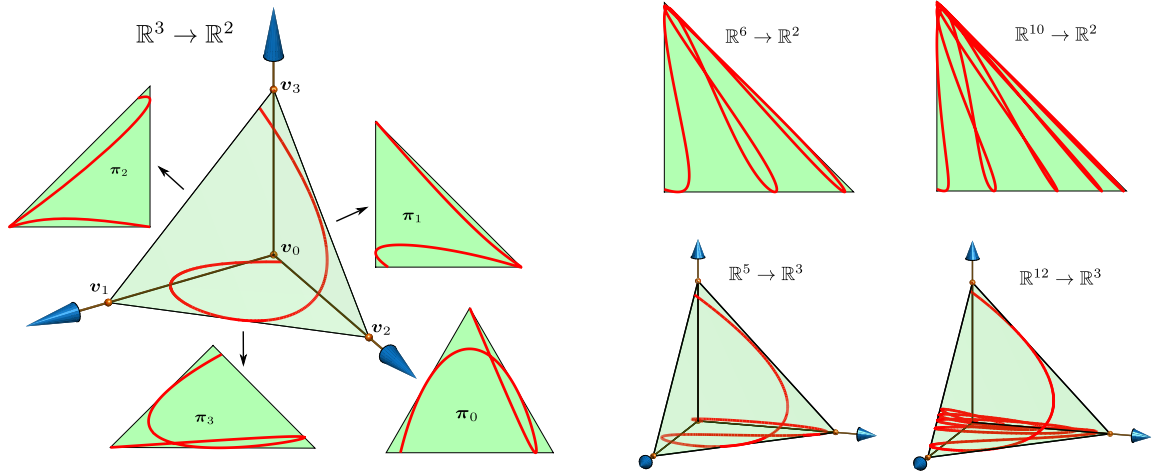


Figure 2-15: The MINVO basis also obtains simplexes that tightly enclose some rational curves (curves whose coordinates are the quotient of two polynomials). On the left, the standard simplex is the smallest 3-simplex containing the 3D curve in red. This means that each facet i (contained in the plane π_i) is also the smallest 2-simplex enclosing the projection of the curve onto that facet using the opposite vertex v_i as viewpoint. On the right, different successive projections to \mathbb{R}^2 and \mathbb{R}^3 are shown.

- **Bézier**: The curve is divided into s subintervals, and then the Bézier enclosure for each of the subintervals is computed.
- **SLEFE**: The curve is divided into s subintervals, and the SLEFE (subdividable linear efficient function enclosure [98, 99, 129]) is obtained using h breakpoints⁵ per subinterval (i.e., $h - 1$ linear segments per subinterval). A SLEFE obtained with h breakpoints per subinterval will be denoted as SL_h .

Note that $s = 1$ corresponds to the case where no subdivision is performed. When $s > 1$, the subintervals of the curve are obtained by evenly splitting the time interval. Moreover, SL_2 (i.e., $h = 2$) corresponds to a SLEFE with only one linear segment (i.e., two breakpoints) per subinterval.

We compare the width, the union, and the convex hull (defined in Appendix 2.F) for the different enclosures. The comparison of the width of the enclosures produced is

⁵As an example, if the time subinterval is $[0.6, 1]$, then three uniformly-distributed breakpoints would be $\{0.6, 0.8, 1\}$, and the SLEFE for that subinterval will consist of a convex enclosure for the part of the curve in $t \in [0.6, 0.8]$, and another convex enclosure for the part of the curve in $t \in [0.8, 1]$.

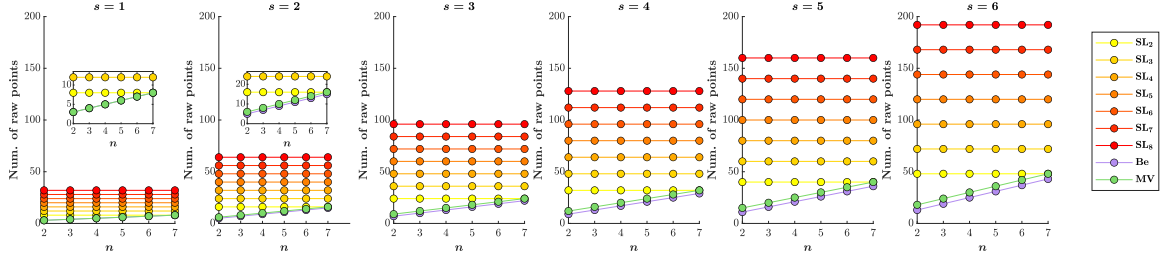


Figure 2-16: Comparison of the number of *raw* points produced by the MINVO (MV) enclosure, Bézier (Be) enclosure, and SLEFE (SL) for n^{th} -degree 2D polynomial curves. Here, s is the number of subintervals the curve is divided into and SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h - 1$ linear segments per subinterval). The MINVO enclosures have fewer raw points than all SL_h ($h \in [2, 8]$) for $n \in [2, 6]$. Compared to Bézier, the MINVO enclosure has $s - 1$ additional raw points, but achieves areas that are up to 30 times smaller (see Fig. 2-23).

available in Appendix 2.G. The comparison of the area and number of vertices of the union and the convex hull is available in Appendix 2.H. In Appendix 2.I, we compare MINVO and SLEFE in terms of runtime and simplicity of their implementation.

Several conclusions can be drawn regarding the comparison between MINVO, Bézier, and SLEFE:

- Compared to SL_2 , MINVO achieves a smaller area for most of the n - s combinations tested, and sometimes using only half of the vertices needed by SL_2 . Compared to SL_h ($h \in [3, 8]$), MINVO typically achieves a smaller area for the cases where either s is small or the degree n is high, and it usually does so using fewer number of vertices than SL_h . On the other hand, SL_h tends to achieve a smaller area when either s is large or the degree n is small, but it usually requires more vertices than MINVO. Hence, MINVO is advantageous with respect to SL_h in applications where having a small number of vertices is crucial. For example, a smaller number of vertices can substantially reduce the total computation time in algorithms that, after finding the enclosure, need to iterate through all of the vertices found to impose a constraint or perform a specific operation/check for each of them. If the number of vertices is not important for the specific application, then SL_h ($h \geq 4$) should be chosen, since

it typically achieves a smaller area of the union and area of the convex hull.

- Compared to Bézier, MINVO also achieves a smaller union and hull area for the cases where either s is small or the degree n is high, and, when $n \in [3, 7]$, it does so using only up to 1.3 times the number of vertices of Bézier.
- In terms of the width of the enclosure (Appendix 2.G), SLEFE performs better than MINVO. The use of SLEFE is hence desirable in the cases where the width of the enclosure is more important for the specific application.
- For any of the techniques, operations like the union, the convex hull, or the outer boundaries computation for SLEFE ([129]) are typically either not possible or computationally expensive in the applications where the curve itself is a decision variable of an optimization problem (as in, e.g., [48, 147, 160, 163]). Instead, we can use the *raw points* of the enclosure, which in 2D can be defined as the *unique* control points of each subinterval (for MINVO and Bézier), and as the *unique* vertices of each of the rectangles generated per breakpoint of each subinterval (for SL_h). The number of these *raw points* for a general 2D curve is $ns + s$ for MINVO, $ns + 1$ for Bézier, and $4hs$ for SL_h . The comparison of these raw points is shown in Fig. 2-16. The MINVO enclosures have fewer raw points than all SL_h ($h \in [2, 8]$) for $n \in [2, 6]$. Compared to Bézier, the MINVO enclosure has $s - 1$ additional raw points, but achieves areas that are up to 30 times smaller (see Fig. 2-23).
- As noted in [129], SLEFE depends *pseudo-linearly* on the coefficients of the polynomial curve (i.e., linearly except for a min/max operation). In contrast, the MINVO or Bézier enclosures depend *linearly* on the coefficients of the polynomial curve. This makes the MINVO and Bézier enclosures more suitable for time-critical optimization problems in which the coefficients of the curve are decision variables.

2.8 Final Remarks

2.8.1 Conversion between MINVO, Bernstein, and B-Spline

Given a curve $P \in \mathcal{P}^n$, the control points (i.e., the vertices of the n -simplex that encloses that curve) using a basis α can be obtained from the control points of a different basis β ($\alpha, \beta \in \{\text{MV}, \text{Be}, \text{BS}\}$) using the formula

$$\mathbf{V}_\alpha = \mathbf{P}\mathbf{A}_\alpha^{-1} = \mathbf{V}_\beta\mathbf{A}_\beta\mathbf{A}_\alpha^{-1}. \quad (2.5)$$

For instance, to obtain the Bernstein control points from the MINVO control points we can use $\mathbf{V}_{\text{Be}} = \mathbf{V}_{\text{MV}}\mathbf{A}_{\text{MV}}\mathbf{A}_{\text{Be}}^{-1}$. The matrices \mathbf{A}_{MV} are the ones shown in Table 2.2, while the matrices \mathbf{A}_{Be} and \mathbf{A}_{BS} are available in [133]. Note that all the matrices need to be expressed in the same interval ($t \in [-1, 1]$ in this work), and that the inverses of these matrices can be easily precomputed offline.

2.8.2 Tighter Volumes for Problem 1 via Subdivision

As shown in Section 2.7, and at the expense of adding more vertices, tighter polyhedral solutions for Problem 1 can be obtained by dividing the polynomial curve into several subintervals and then computing the convex hull of the MINVO enclosure of each subinterval. To subdivide the curve, one can do it first in Bézier form (leveraging therefore the properties of De Casteljau's algorithm), and then compute the MINVO control points as linear functions of the Bézier control points of that subinterval using Eq. 2.5. Alternatively, one can also tabulate (offline) the inverse of the matrices \mathbf{A}_{MV} , expressed in the subinterval desired, and then simply compute the MINVO control points of that subinterval as $\mathbf{V}_{\text{MV}} = \mathbf{P}\mathbf{A}_{\text{MV}}^{-1}$.

Several examples for $n = 3$ are shown in Fig. 2-17, where the original curve $P \in \mathcal{P}^3$ is split into 5 subintervals (i.e., $s = 5$), and the resulting convex hull is a polyhedron with 20 vertices that is 1.19 times smaller than the smallest 3-simplex that encloses the whole curve (i.e., the simplex found by applying the MINVO basis to the whole curve).

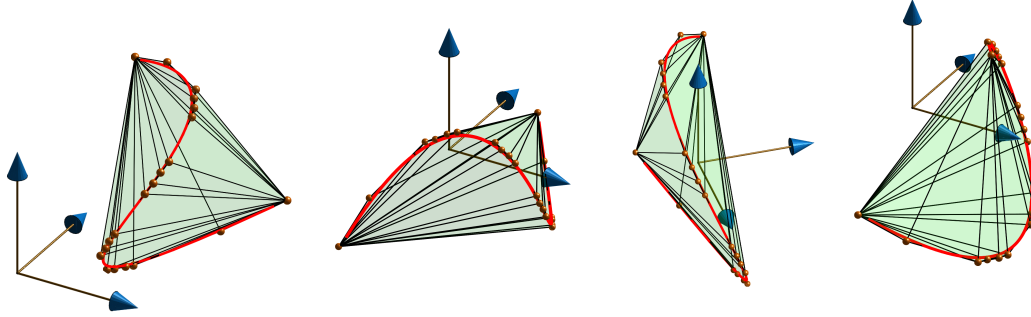


Figure 2-17: Tighter polyhedral outer approximations for a curve $P \in \mathcal{P}^n$ can be obtained by splitting the curve into several subintervals, calculating the MINVO n -simplexes that enclose each one of these subintervals and then computing the convex hull of all these simplexes. In all these cases shown, the original curve $P \in \mathcal{P}^3$ is splitted into 5 subintervals (i.e., $s = 5$), and the resulting convex hull is a polyhedron with 20 vertices that is 1.19 times smaller than the smallest 3-simplex that encloses the whole curve (i.e., the simplex found by applying the MINVO basis to the whole curve).

Depending on the specific application, one might also be interested in obtaining a sequence of overlapping polyhedra whose union (a nonconvex set in general) completely encloses the curve. This can be obtained by simply computing the MINVO enclosure for every subinterval of the curve.

2.8.3 When Should each Basis Be Used?

The Bernstein (Bézier) and B-Spline bases have many useful properties that are not shared by the MINVO basis. For example, a polynomial curve passes through the first and last Bézier control points, the derivative of a Bézier curve can be easily computed from the difference of the Bézier control points, and the B-Spline basis has built-in smoothness in curves with several segments. Hence, it may be desirable to use the Bézier or B-Spline control points to design and model the curve, and then convert the control points of every interval to the MINVO control points (using the simple linear transformation given in Section 2.8.1) to perform collision/intersection checks, or to impose collision-avoidance constraints in an optimization problem [61, 160, 163]. This approach benefits from the properties of the Bernstein/B-Spline bases, while also exploiting the enclosures obtained by the MINVO basis.

Appendix 2.A Volume of the Convex Hull of a Polynomial Curve

The volume of the convex hull of any n^{th} -degree polynomial curve can be easily obtained using the result from [71, Theorem 15.2]. In this work,⁶ it is shown that the volume of the convex hull of a curve R with $\mathbf{r}(t) := \left[\frac{t+1}{2} \left(\frac{t+1}{2} \right)^2 \cdots \left(\frac{t+1}{2} \right)^n \right]^T$ is given by

$$\text{vol}(\text{conv}(R)) = \prod_{j=1}^n B(j, j) = \prod_{j=1}^n \left(\frac{((j-1)!)^2}{(2j-1)!} \right) = \frac{1}{n!} \prod_{j=1}^n \left(\frac{j!(j-1)!}{(2j-1)!} \right) = \frac{1}{n!} \prod_{0 \leq i < j \leq n} \binom{j-i}{j+i},$$

where $B(x, y)$ denotes the beta function. Let us now define $\tilde{\mathbf{t}}$ as $\tilde{\mathbf{t}} := \left[t^n \ t^{n-1} \ \dots \ t \right]^T$, \boxtimes as any number in \mathbb{R} and write $\mathbf{r}(t)$ as:

$$\mathbf{r}(t) = \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 & \frac{1}{2} & \boxtimes \\ 0 & 0 & \cdots & \frac{1}{2^2} & \boxtimes & \boxtimes \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \frac{1}{2^{n-1}} & \cdots & \boxtimes & \boxtimes & \boxtimes \\ \frac{1}{2^n} & \boxtimes & \cdots & \boxtimes & \boxtimes & \boxtimes \end{bmatrix}}_{:=\mathbf{R}} \mathbf{t} = \mathbf{R}_{:,0:n-1} \tilde{\mathbf{t}} + \mathbf{R}_{:,n}$$

Now, defining $\mathbf{Q} := \mathbf{P}_{:,0:n-1} \mathbf{R}_{:,0:n-1}^{-1}$, note that $(\mathbf{p}(t) - \mathbf{P}_{:,n}) = \mathbf{P}_{:,0:n-1} \tilde{\mathbf{t}} = \mathbf{Q} \mathbf{R}_{:,0:n-1} \tilde{\mathbf{t}} = \mathbf{Q}(\mathbf{r}(t) - \mathbf{R}_{:,n}) = \mathbf{Q} \mathbf{r}(t) - \mathbf{Q} \mathbf{R}_{:,n}$. As the translation part does not affect the volume, we can write

$$\text{vol}(\text{conv}(P)) = \text{vol}(\text{conv}(\{\mathbf{p}(t) - \mathbf{P}_{:,n} \mid t \in [-1, 1]\})) = \text{vol}(\text{conv}(\{\mathbf{Q} \mathbf{r}(t) \mid t \in [-1, 1]\})) = \dots$$

$$\dots = \text{vol}(\mathbf{Q} \text{conv}(R)) = \text{abs} \left(\frac{|\mathbf{P}_{:,0:n-1}|}{|\mathbf{R}_{:,0:n-1}|} \right) \text{vol}(\text{conv}(R)),$$

where we have used the notation $\mathbf{Q} \text{conv}(R)$ to denote the set $\{\mathbf{Q} \mathbf{x} \mid \mathbf{x} \in \text{conv}(R)\}$.

As the determinant of $\mathbf{R}_{:,0:n-1}$ is $|\mathbf{R}_{:,0:n-1}| = \prod_{i=1}^n \frac{1}{2^i} = 2^{-\frac{n(n+1)}{2}}$, we can conclude

⁶Note that [71] uses the convention $t \in [0, 1]$ (instead of $t \in [-1, 1]$), and therefore it uses the curve $[t \ t^2 \ \dots \ t^n]^T$. Note also that the convex hull of a moment curve is equal to a cyclic polytope [7, 187] with infinitely many points evenly distributed along the curve.

that:

$$\boxed{\text{vol}(\text{conv}(P)) = \frac{\text{abs}(|\mathbf{P}_{:,0:n-1}|)}{n!} 2^{\frac{n(n+1)}{2}} \prod_{0 \leq i < j \leq n} \left(\frac{j-i}{j+i} \right)} \quad \square$$

Appendix 2.B Invertibility of the Matrix \mathbf{A}

From Eq. 2.3, we have that the $(n+1) \times (n+1)$ matrix \mathbf{A} satisfies

$$\begin{bmatrix} \mathbf{P} \\ \mathbf{e}^T \end{bmatrix} = \begin{bmatrix} \mathbf{V} \\ \mathbf{1}^T \end{bmatrix} \mathbf{A}.$$

As $\text{abs} \left(\begin{bmatrix} \mathbf{P} \\ \mathbf{e}^T \end{bmatrix} \right) = \text{abs}(|\mathbf{P}_{:,0:n-1}|) \neq 0$, and $\left| \begin{bmatrix} \mathbf{V} \\ \mathbf{1}^T \end{bmatrix} \right| = |\mathbf{V}^T \mathbf{1}| \neq 0$ (see Section 2.3),

we have that $\text{rank} \left(\begin{bmatrix} \mathbf{P} \\ \mathbf{e}^T \end{bmatrix} \right) = \text{rank} \left(\begin{bmatrix} \mathbf{V} \\ \mathbf{1}^T \end{bmatrix} \right) = n+1$. Using now the fact that $\text{rank}(\mathbf{BC}) \leq \min(\text{rank}(\mathbf{B}), \text{rank}(\mathbf{C}))$, we conclude that $\text{rank}(\mathbf{A}) = n+1$ (i.e., \mathbf{A} has full rank), and therefore \mathbf{A} is invertible. \square

Appendix 2.C Karush–Kuhn–Tucker Conditions (for odd n)

In this Appendix we derive the KKT conditions for this problem:

$$\boxed{\begin{array}{l} \min_{\mathbf{A} \in \mathbb{R}^{(n+1) \times (n+1)}} \quad -\ln(|\mathbf{A}^T \mathbf{A}|) \\ \text{s.t.} \quad \mathbf{A}^T \mathbf{1} = \mathbf{e} \\ \mathbf{A} \mathbf{t} \geq \mathbf{0} \quad \forall t \in [-1, 1] \end{array}}$$

which is equivalent to Problem 3. For the sake of brevity, we present here the case n odd (the case n even can be easily obtained with small modifications). In the following, $\mathbf{V} * \mathbf{W}$ will be the matrix resulting from the row-wise discrete convolution (i.e., $(\mathbf{V} * \mathbf{W})_{i,:} = \mathbf{V}_{i,:} * \mathbf{W}_{i,:}$), and $\text{Top}(\mathbf{a}, \mathbf{b})$ will denote the Toeplitz matrix whose

first column is \mathbf{a} and whose first row is \mathbf{b}^T . Let us also define:

$$\mathbf{R}_G := \text{Top} \left(\begin{bmatrix} 1 \\ 1 \\ \mathbf{0}_{n-1} \end{bmatrix}, \begin{bmatrix} 1 \\ \mathbf{0}_{n-1} \end{bmatrix} \right) = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0}_n^T \end{bmatrix} + \begin{bmatrix} \mathbf{0}_n^T \\ \mathbf{I}_n \end{bmatrix} \quad \mathbf{R}_H := \text{Top} \left(\begin{bmatrix} -1 \\ 1 \\ \mathbf{0}_{n-1} \end{bmatrix}, \begin{bmatrix} -1 \\ \mathbf{0}_{n-1} \end{bmatrix} \right) = \begin{bmatrix} -\mathbf{I}_n \\ \mathbf{0}_n^T \end{bmatrix} + \begin{bmatrix} \mathbf{0}_n^T \\ \mathbf{I}_n \end{bmatrix} \quad \mathbf{L}_q := \begin{bmatrix} \mathbf{0}^T & \mathbf{0} \\ \mathbf{I}_{q-1} & \mathbf{0} \end{bmatrix}_{q \times q},$$

and the matrices $\mathbf{G} \in \mathbb{R}^{(n+1) \times \frac{n+1}{2}}$, $\mathbf{H} \in \mathbb{R}^{(n+1) \times \frac{n+1}{2}}$, and $\boldsymbol{\lambda} \in \mathbb{R}^{(n+1)}$. We know that

$$(\mathbf{A}\mathbf{t})_i \geq 0 \quad \forall t \in [-1, 1] \stackrel{\text{(a)}}{\iff} (\mathbf{A}\mathbf{t})_i = (t+1)g_i^2(t) + (1-t)h_i^2(t) \stackrel{\text{(b)}}{\iff} \dots$$

$$\dots \stackrel{\text{(b)}}{\iff} \mathbf{A}_{i,:} = (\mathbf{G}_{i,:} * \mathbf{G}_{i,:}) \mathbf{R}_G^T + (\mathbf{H}_{i,:} * \mathbf{H}_{i,:}) \mathbf{R}_H^T \iff \mathbf{A} = \left[\mathbf{G} * \mathbf{G} \mid \mathbf{H} * \mathbf{H} \right] \begin{bmatrix} \mathbf{R}_G^T \\ \mathbf{R}_H^T \end{bmatrix}, \quad (2.6)$$

where $g_i(t)$ and $h_i(t)$ are polynomials of degree $\frac{n-1}{2}$. Note that (a) is given by the Markov–Lukács Theorem (see Section 2.4.4). In (b) we have simply used the discrete convolution to multiply $g_i(t)$ by itself, and the Toeplitz matrix \mathbf{R}_G to multiply the result by $(t+1)$ [133]. An analogous reasoning applies for the term with h_i .⁷ Using now \mathbf{G} and \mathbf{H} as the decision variables of the primal problem (where \mathbf{A} is given by Eq. 2.6), the Lagrangian is

$$\mathcal{L} = -\ln(|\mathbf{A}^T \mathbf{A}|) + \boldsymbol{\lambda}^T (\mathbf{A}^T \mathbf{1} - \mathbf{e}).$$

Differentiating the Lagrangian yields to

$$\frac{\partial \mathcal{L}}{\partial \mathbf{G}_{ij}} = \text{tr} \left(- \underbrace{\frac{\partial \ln(|\mathbf{A}^T \mathbf{A}|)}{\partial \mathbf{A}}}_{=2\mathbf{A}^+ = 2\mathbf{A}^{-1}} \mathbf{Q}_{\mathbf{G}_{ij}} \right) + \text{tr} (\boldsymbol{\lambda} \mathbf{1}^T \mathbf{Q}_{\mathbf{G}_{ij}}) = \text{tr} \left((-2\mathbf{A}^{-1} + \boldsymbol{\lambda} \mathbf{1}^T) \mathbf{Q}_{\mathbf{G}_{ij}} \right),$$

⁷Alternatively, we can also write:

$$(\mathbf{A}_{i,:})^T = \mathbf{R}_G \text{Top} \left(\begin{bmatrix} \mathbf{G}_{i,:} \\ \mathbf{0}_{\frac{n-1}{2}} \end{bmatrix}, \begin{bmatrix} \mathbf{G}_{i,0} \\ \mathbf{0}_{\frac{n-1}{2}} \end{bmatrix} \right) \mathbf{G}_{i,:} + \mathbf{R}_H \text{Top} \left(\begin{bmatrix} \mathbf{H}_{i,:} \\ \mathbf{0}_{\frac{n-1}{2}} \end{bmatrix}, \begin{bmatrix} \mathbf{H}_{i,0} \\ \mathbf{0}_{\frac{n-1}{2}} \end{bmatrix} \right) \mathbf{H}_{i,:}$$

where

$$\mathbf{Q}_{\mathbf{G}_{ij}} := \frac{\partial \mathbf{A}}{\partial \mathbf{G}_{ij}} = 2 \left(\mathbf{L}_{(n+1)} \right)^{i-1} \left[\begin{array}{c} \left[\mathbf{G}_{i,:} \ \mathbf{0}^T \right] \left(\mathbf{L}_n^T \right)^{j-1} \mathbf{R}_G^T \\ \mathbf{0}_{n \times (n+1)} \end{array} \right]_{(n+1) \times (n+1)}. \quad (2.7)$$

Same expression applies for $\mathbf{Q}_{\mathbf{H}_{ij}} := \frac{\partial \mathbf{A}}{\partial \mathbf{H}_{ij}}$, but using $\mathbf{H}_{i,:}$ and \mathbf{R}_H^T instead. The KKT equations can therefore be written as follows:

KKT equations: Solve for $\mathbf{G}, \mathbf{H}, \lambda$:

$$\begin{cases} \operatorname{tr} \left(\left(-2\mathbf{A}^{-1} + \lambda \mathbf{1}^T \right) \mathbf{Q}_{\mathbf{G}_{ij}} \right) = 0 & \forall i \in \{0, \dots, n\}, \forall j \in \left\{ 0, \dots, \frac{n-1}{2} \right\} \\ \operatorname{tr} \left(\left(-2\mathbf{A}^{-1} + \lambda \mathbf{1}^T \right) \mathbf{Q}_{\mathbf{H}_{ij}} \right) = 0 & \forall i \in \{0, \dots, n\}, \forall j \in \left\{ 0, \dots, \frac{n-1}{2} \right\} \\ \mathbf{A}^T \mathbf{1} = \mathbf{e} \end{cases}$$

where \mathbf{A} is given by Eq. 2.6. and $\mathbf{Q}_{\mathbf{G}_{ij}}$ by Eq. 2.7. $\mathbf{Q}_{\mathbf{H}_{ij}}$ is also given by Eq. 2.7, but using $\mathbf{H}_{i,:}$ and \mathbf{R}_H^T instead.

Appendix 2.D Plot of a Polynomial $x(t) \in \mathbb{R}[t]$ over $t \in [-1, 1]$

This chapter has focused on polynomial curves as defined in Section 2.2.1. One particular case of such curves corresponds to the plot of a polynomial $x(t) \in \mathbb{R}[t]$ over $t \in [-1, 1]$. Indeed, that plot simply corresponds to a curve that has $\mathbf{p}(t) = [t \ x(t)]^T$. Some examples of such curves and their associated convex hulls are shown in Fig. 2-18. Here, we generate a polynomial $x(t) \in \mathbb{R}[t]$ passing through $n + 1$ random points in $[-1, 1]$, and then plot the control points for the case $k = 1$ (i.e., $p(t) = x(t)$) and the convex hull of the control points for the case $k = 2$ (i.e., $\mathbf{p}(t) = [t \ x(t)]^T$). Note that for the cases with $k = 1$, a smaller convex hull can be obtained by simply reparametrizing the curve to a first-degree curve: $p(t) = x_{\min} + \frac{t+1}{2}(x_{\max} - x_{\min})$ (where $x_{\min} := \min_{t \in [-1, 1]} x(t)$ and $x_{\max} := \max_{t \in [-1, 1]} x(t)$), and then using the matrices \mathbf{A}_{MV} and \mathbf{A}_{Be} corresponding to $n = 1$. This would give x_{\min} and x_{\max} as the control points of the curve.

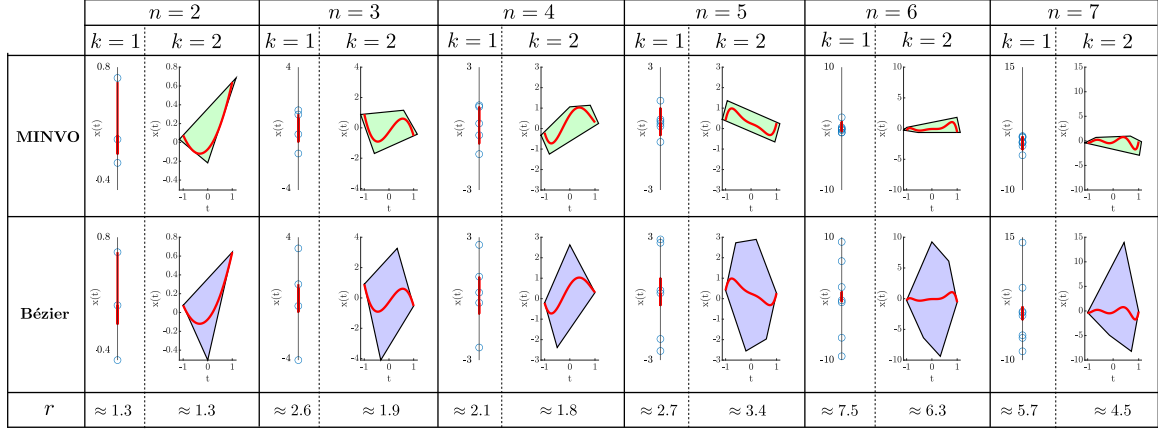


Figure 2-18: Comparison of the convex hull of the MINVO and Bézier control points for $k = m \in \{1, 2\}$ and different n . Letting $x(t)$ denote a polynomial of degree n , the column with $k = 1$ shows the plot of the curve $p(t) = x(t)$, and the blue circles denote the control points. The column with $k = 2$ shows the plot of the curve $\mathbf{p}(t) = [t x(t)]^T$ (which corresponds to the graph of the polynomial $x(t)$ over $t \in [-1, 1]$) and the convex hull of the control points. In the last row, r denotes the ratio of the longitudes $\frac{\text{Long}_{\text{Be}}}{\text{Long}_{\text{MV}}}$ (for $k = 1$) or the areas $\frac{\text{Area}_{\text{Be}}}{\text{Area}_{\text{MV}}}$ (for $k = 2$) of the convex hulls of the control points. The polynomials $x(t)$ pass through $n + 1$ random points in the interval $[-1, 1]$.

Appendix 2.E Application of the MINVO Basis to Surfaces

Similar to any other polynomial basis that is nonnegative and is a partition of unity (i.e., the polynomials in the basis sum up to one), the MINVO basis can be applied to generate polynomial surfaces of degree (q, r) contained in the convex hull of its $(q + 1) \cdot (r + 1)$ control points. Let us define $\mathbf{u}_q := [u^q u^{q-1} \dots 1]^T$ and let $\mathbf{A}_{\text{MV},q}$ denote the matrix \mathbf{A}_{MV} for $n = q$. Analogous definitions apply for v , \mathbf{v}_r , and $\mathbf{A}_{\text{MV},r}$. Moreover, let $\mathbf{V}_{\text{MV},j} \in \mathbb{R}^{(q+1) \times (r+1)}$ contain the coordinate $j \in \{x, y, z\}$ of the control points. The parametric equation of the surface will then be given by:

$$\mathbf{s}(u, v) = \begin{bmatrix} (\mathbf{A}_{\text{MV},q} \mathbf{u}_q)^T \mathbf{V}_{\text{MV},x} \\ (\mathbf{A}_{\text{MV},q} \mathbf{u}_q)^T \mathbf{V}_{\text{MV},y} \\ (\mathbf{A}_{\text{MV},q} \mathbf{u}_q)^T \mathbf{V}_{\text{MV},z} \end{bmatrix} \mathbf{A}_{\text{MV},r} \mathbf{v}_r$$

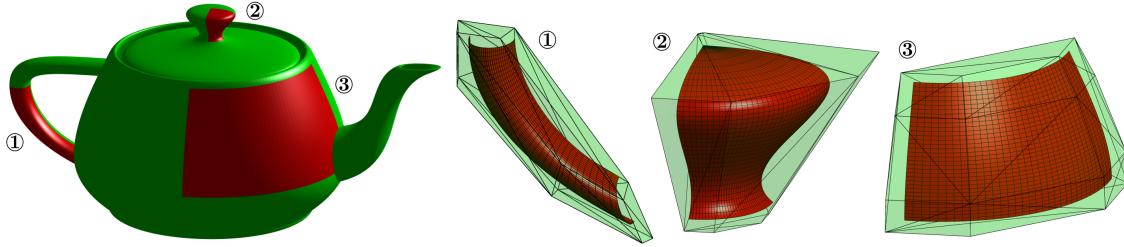


Figure 2-19: MINVO basis applied to generate outer polyhedra that enclose three bicubic patches of the teapot.

where $\mathbf{s}(u, v) \in \mathbb{R}^3$ and $u, v \in [-1, 1]$. An example of the MINVO basis applied to generate different bicubic surfaces (i.e., $q = r = 3$) is shown in Fig. 2-19. Note however that, when applied to surfaces, the MINVO basis is not guaranteed to generate always a smaller enclosing polyhedron than, for example, the Bézier basis.

Appendix 2.F Definitions of the Union, Convex Hull, and Width of an Enclosure (for 2D Curves)

Let us define $I := \{0, 1, \dots, s - 1\}$ and $J := \{0, 1, \dots, h - 2\}$, where s denotes the number of subdivisions used, and h is the number of breakpoints per subinterval used in SLEFE. Moreover, let \mathcal{E} denote an enclosure, $\text{conv}(\cdot)$ the convex hull of a set of vertices, $\text{vert}(\cdot)$ the vertices of an enclosure, and $\text{lssbb}(\cdot)$ the length of the smallest side of the smallest arbitrarily-oriented bounding box of an enclosure. The union, convex hull, and width are then defined as follows:

	Union	Convex hull	Width
MV/Be	$\bigcup_{i \in I} \mathcal{E}_i$	$\text{conv} \left(\bigcup_{i \in I} \text{vert}(\mathcal{E}_i) \right)$	$\max \left(\bigcup_{i \in I} \text{lssbb}(\mathcal{E}_i) \right)$
SL	$\bigcup_{i \in I, j \in J} \mathcal{E}_{ij}$	$\text{conv} \left(\bigcup_{i \in I, j \in J} \text{vert}(\mathcal{E}_{ij}) \right)$	$\max \left(\bigcup_{i \in I, j \in J} \text{lssbb}(\mathcal{E}_{ij}) \right)$

Here, \mathcal{E}_i (for MV and Be) is the enclosure of the subinterval i , while \mathcal{E}_{ij} (for SL) is the enclosure of the segment j of the subinterval i . Note that the area of the union is

the sum of the individual areas minus the overlapping area, and it can be computed numerically using [106]. The area of the convex hull is the area of the smallest convex set that contains $\bigcup_{i \in I} \text{vert}(\mathcal{E}_i)$ (for MV/Be) or $\bigcup_{i \in I, j \in J} \text{vert}(\mathcal{E}_{ij})$ (for SL), and it can be computed numerically using [107].

Appendix 2.G Comparison with SLEFEs and Bézier: Width

In this section, we compare the MINVO enclosure, Bézier enclosure, and SLEFE in terms of their widths. Given that the width is an important metric for some CAD applications, for this comparison we use data from different real CAD models.

We first use one of the 2D trim curves of the model 10-23022015-110975 from <https://traceparts.com>. This curve, shown in Fig. 2-20, is a B-Spline of degree 8, which can be split into 4 Bézier curves. For each of these curves, the comparison between the widths of the enclosures is shown in Table 2.6.



Figure 2-20: 2D trim curve of the model 10-23022015-110975 from <https://traceparts.com>. This curve can be split into four Bézier curves, shown in different colors.

We then perform a similar analysis using six models taken from the ABC dataset [76], which is an extensive dataset of CAD models. These models are shown in Fig. 2-21. We obtain the MINVO enclosure, Bézier enclosure, and SLEFE of the 2D (approximate) preimages of ten 3D curves of these models. These 2D curves have degrees ranging from 3 to 9. The results are shown in Fig. 2-22.

All these previous results (Table 2.6 and Fig. 2-22) allow us to conclude that SLEFE performs much better than MINVO in terms of the width of the enclosure. Hence, in applications where it is crucial to have a small width of the enclosure, SLEFE should be preferred over MINVO.

Table 2.6: Comparison between the width of the MINVO enclosure, Bézier enclosure, and SLEFE for the curve shown in Fig. 2-20. SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h - 1$ linear segments per subinterval). We use $s = 1$ for all the cases of this table.

	Curve 1	Curve 2	Curve 3	Curve 4
$\frac{\text{Width}_{Be}}{\text{Width}_{MV}}$	1.05	0.94	0.92	0.96
$\frac{\text{Width}_{SL_2}}{\text{Width}_{MV}}$	5.95	1.43	1.20	3.92
$\frac{\text{Width}_{SL_3}}{\text{Width}_{MV}}$	2.20	0.67	1.24	1.62
$\frac{\text{Width}_{SL_4}}{\text{Width}_{MV}}$	1.12	0.30	0.49	0.79
$\frac{\text{Width}_{SL_5}}{\text{Width}_{MV}}$	0.58	0.19	0.30	0.47
$\frac{\text{Width}_{SL_6}}{\text{Width}_{MV}}$	0.37	0.13	0.20	0.30
$\frac{\text{Width}_{SL_7}}{\text{Width}_{MV}}$	0.22	0.087	0.14	0.20
$\frac{\text{Width}_{SL_8}}{\text{Width}_{MV}}$	0.16	0.067	0.11	0.15
$\frac{\text{Width}_{SL_9}}{\text{Width}_{MV}}$	0.13	0.054	0.085	0.12
$\frac{\text{Width}_{SL_{10}}}{\text{Width}_{MV}}$	0.097	0.042	0.067	0.090

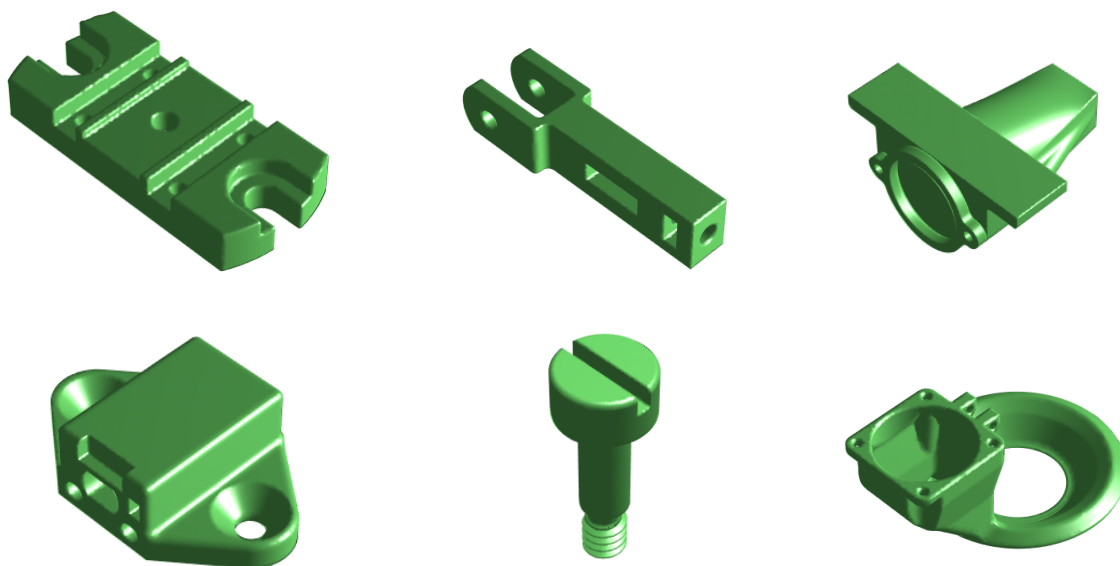


Figure 2-21: Models taken from the ABC dataset [76].

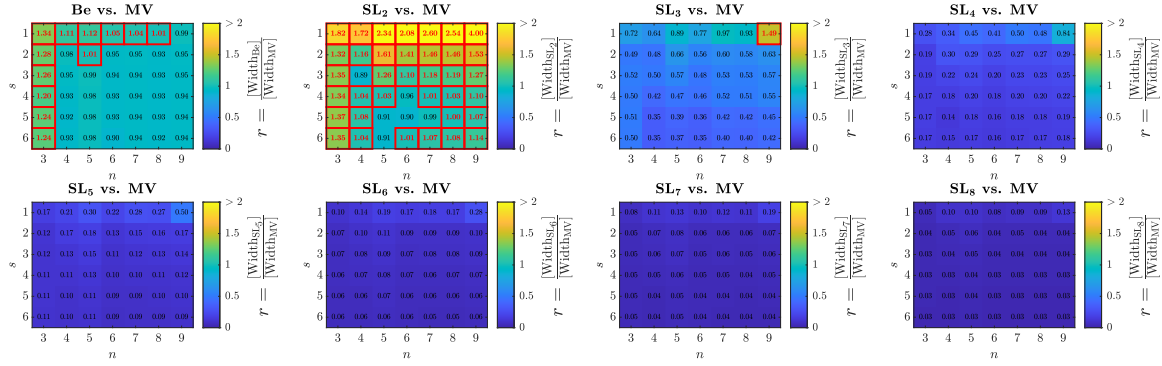


Figure 2-22: Comparison between the width of the MINVO (MV) enclosure, Bézier (Be) enclosure, and SLEFE (SL) for n^{th} -degree 2D polynomial curves. Here, s is the number of subintervals the curve is divided into, SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h - 1$ linear segments per subinterval), and $[\cdot]$ denotes the mean operator. For every n - s combination, a total of 10 polynomial curves obtained from the models shown in Fig. 2-21 were used. The red squares denote the n - s combination for which MINVO achieves a smaller width.

Appendix 2.H Comparison with SLEFEs and Bézier: Area and Number of Vertices

The n^{th} -degree 2D polynomial curves used pass through $n + 1$ points $\{\mathbf{x}_0, \dots, \mathbf{x}_n\}$ that satisfy the dynamical system $\mathbf{x}_{k+1} = \mathbf{x}_k + \text{rand}() - 0.15 \cdot \mathbf{1}$, where $\text{rand}()$ is a random vector in $[0, 1]^2$ and $\mathbf{x}_0 = \mathbf{0}$. Note that these curves are artificially generated, and do not correspond to real CAD data. The results are shown in Fig. 2-23, and some examples are available in Figs. 2-24 and 2-25.

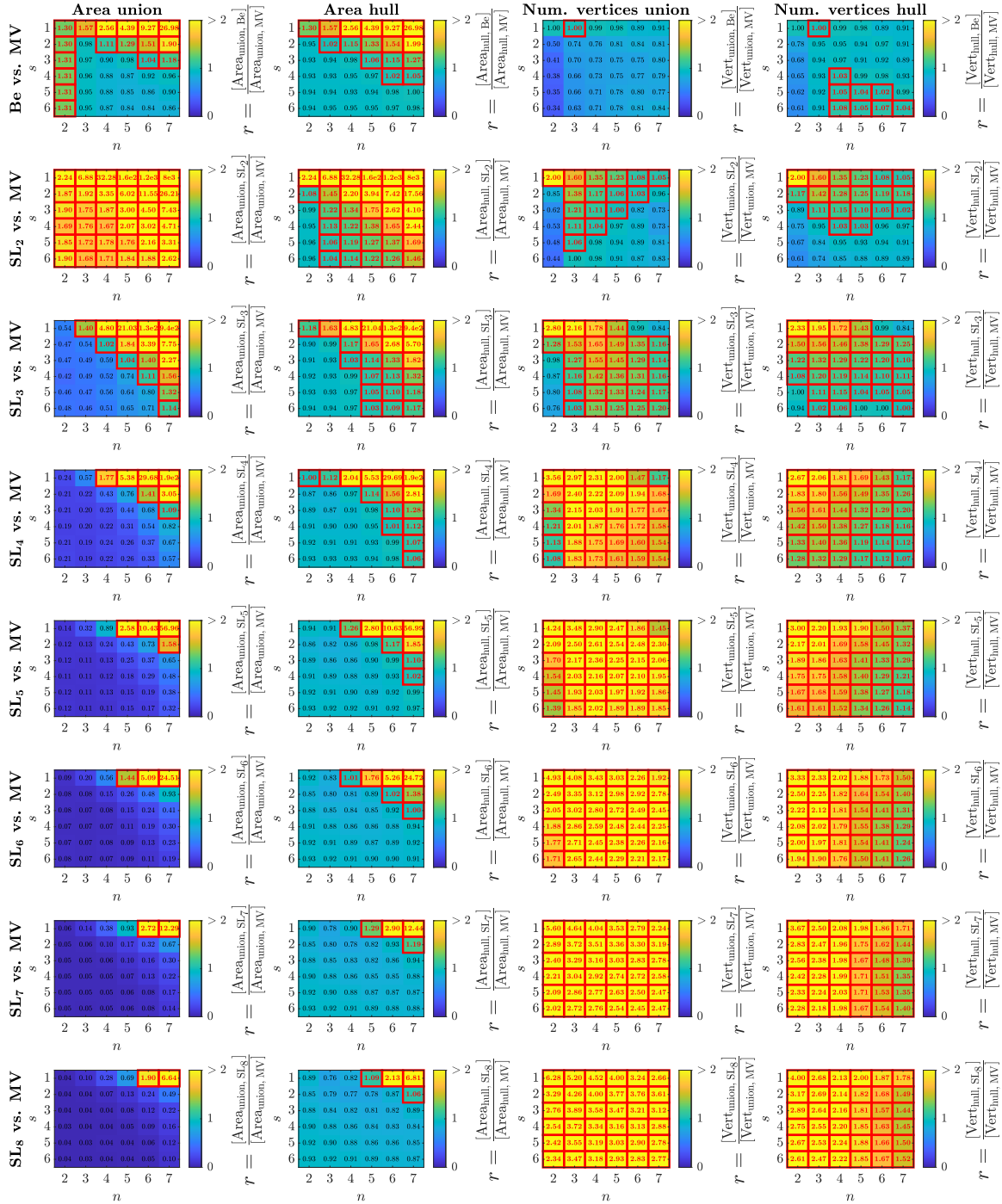


Figure 2-23: Comparison between the MINVO (MV) enclosure, Bézier (Be) enclosure, and SLEFE (SL) for n^{th} -degree 2D polynomial curves. Here, s is the number of subintervals the curve is divided into, SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h-1$ linear segments per subinterval), and $[\cdot]$ denotes the mean operator. For every n - s combination, a total of 100 polynomial curves obtained as described in Appendix 2.H were used. The red squares denote the n - s combination for which MINVO achieves a smaller area (first two columns) or fewer number of vertices (last two columns).

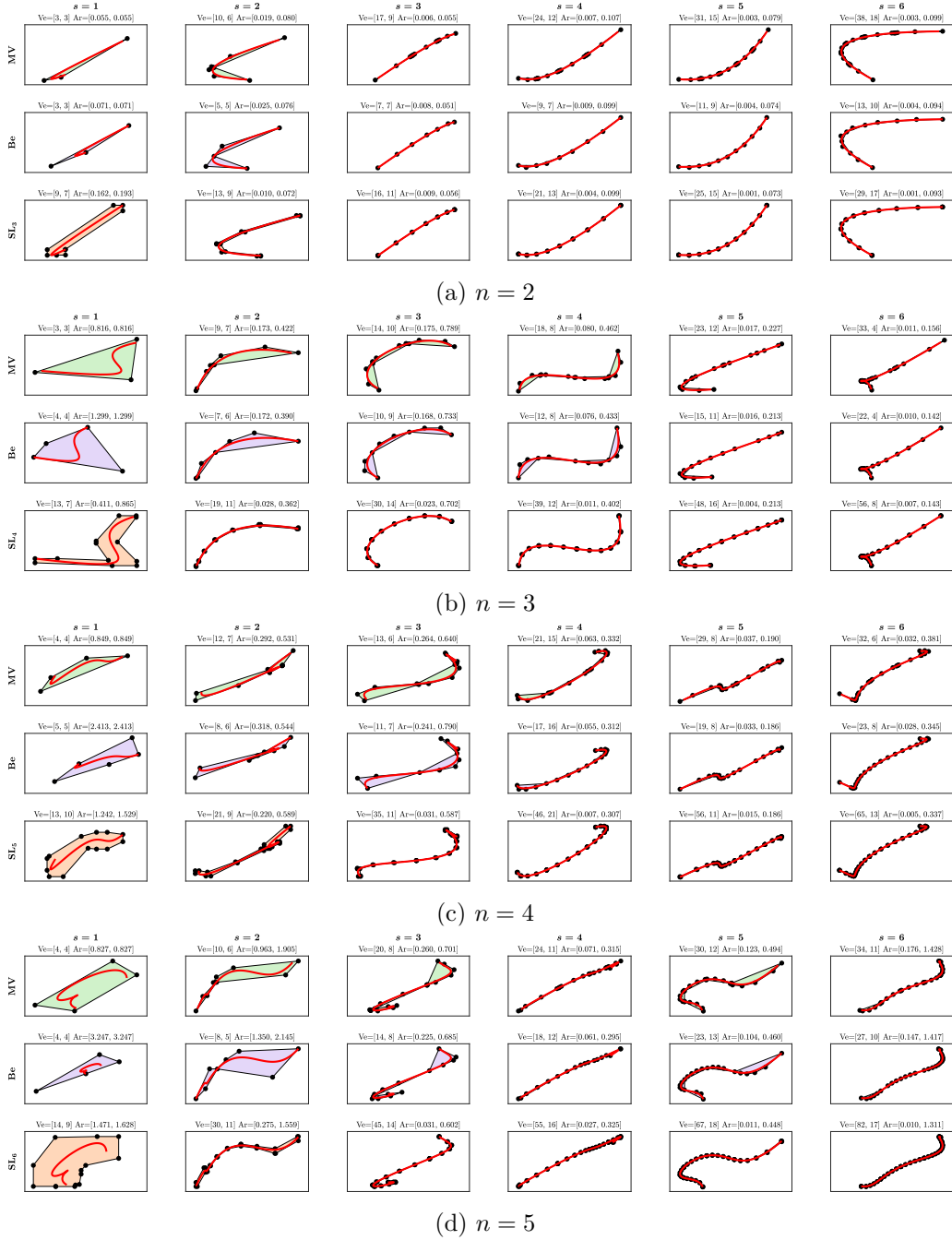


Figure 2-24: Comparison between the MINVO (MV) enclosure, the Bézier (Be) enclosure, and the SLEFE (SL) for n^{th} -degree 2D polynomial curves obtained as described in Appendix 2.H. SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h - 1$ linear segments per subinterval), and s is the number of subdivisions used. In all these plots, $h = n + 1$ is used. The notation used is $Ve = [a, b]$ (where a is the number of vertices of the union and b is the number of vertices of the convex hull) and $Ar = [c, d]$ (where c is the area of the union of the enclosures and d is the area of the convex hull of the enclosures). The black points are the vertices of the union. Note that this figure shows only some cases that have $h = n + 1$, but the results in Fig. 2-23 include all the cases with different values of h , n , and s .

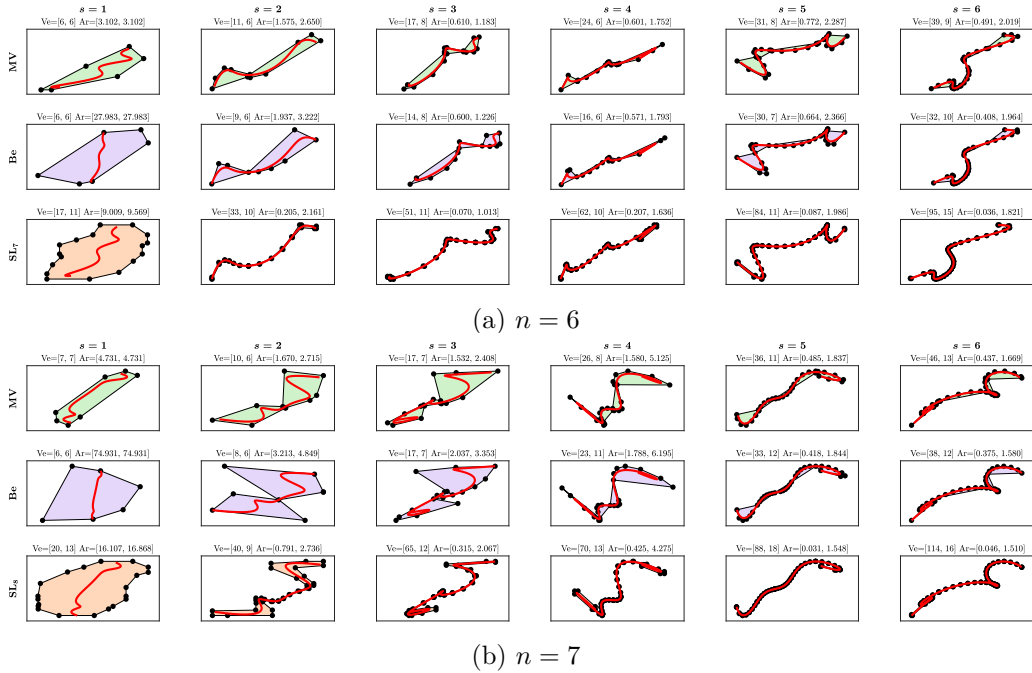


Figure 2-25: Comparison between the MINVO (MV) enclosure, the Bézier (Be) enclosure, and the SLEFE (SL) for n^{th} -degree 2D polynomial curves obtained as described in Appendix 2.H. SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h - 1$ linear segments per subinterval), and s is the number of subdivisions used. In all these plots, $h = n + 1$ is used. The notation used is $Ve = [a, b]$ (where a is the number of vertices of the union and b is the number of vertices of the convex hull) and $Ar = [c, d]$ (where c is the area of the union of the enclosures and d is the area of the convex hull of the enclosures). The black points are the vertices of the union. Note that this figure shows only some cases that have $h = n + 1$, but the results in Fig. 2-23 include all the cases with different values of h , n , and s .

Appendix 2.I Comparison between MINVO and SLEFE in terms of Runtime and Simplicity in the Implementation

In Section 2.7 the MINVO enclosure and the SLEFE are compared in terms of enclosing area and number of vertices. The **enclosing area** is important in terms of conservativeness of the enclosure with respect to the curve, while the **number of vertices** can have a direct impact on the computation time (for example, in settings where the curve is a decision variable in an optimization problem and there is one constraint per vertex), and on the memory needed to store them.

There are also other applications where the **runtime to obtain the enclosure** is important. In this section, we compare the runtimes needed to obtain the SLEFE and the MINVO enclosure from a curve given in its Bézier form (i.e., \mathbf{V}_{Be} is given):

- The MINVO enclosure is obtained by simply doing $\mathbf{V}_{\text{MV}} = \mathbf{V}_{\text{Be}} \mathbf{A}_{\text{Be}} \mathbf{A}_{\text{MV}}^{-1}$ (Eq. 2.5), where the term $\mathbf{A}_{\text{Be}} \mathbf{A}_{\text{MV}}^{-1}$ is tabulated offline using the matrices available in Table 2.2 and [133] for each degree n . If the curve were given instead in its monomial form (i.e., \mathbf{P} is given), then the computation of the MINVO enclosure would also be a simple matrix multiplication: $\mathbf{V}_{\text{MV}} = \mathbf{P} \mathbf{A}_{\text{MV}}^{-1}$, where $\mathbf{A}_{\text{MV}}^{-1}$ is tabulated offline.
- The SLEFE enclosure is obtained as detailed in [120, Section 3.3], using the tabulated values given by the SubLiME package [179]. We optimize the speed of the SLEFE implementation leveraging vector and matrix operations. Note however that the SLEFE computation requires $\max(\cdot)$ and $\min(\cdot)$ operators, and hence it cannot be obtained as a single matrix multiplication.

The timing results are shown in Table 2.7. On average, the MINVO enclosure can be obtained 20.4 times faster than the SLEFE enclosure. These timing results were obtained using Matlab[®] R2021b on an AlienWare Aurora r8 desktop running Ubuntu 18.04 and equipped with an Intel[®] Core[™] i9-9900K CPU, 3.60GHz×16 and 62.6 GiB.

Table 2.7: Computation times required to find the MINVO (MV) enclosure and the SLEFE (SL) for n^{th} -degree polynomial curves. SL_h denotes the SLEFE computed using h breakpoints per subinterval (i.e., $h - 1$ linear segments per subinterval), $ct(\cdot)$ denotes the computation time, and $[\cdot]$ denotes the mean operator. For each cell in this table, a total of 30 random polynomials passing through random points in $[-1, 1]$ were used. All these cases have $s = 1$.

		Degree n					
		2	3	4	5	6	7
Ratios of comp. times	$\frac{[ct(SL_2)]}{[ct(MV)]}$	24.8	24.0	20.8	18.3	15.6	13.3
	$\frac{[ct(SL_3)]}{[ct(MV)]}$	24.4	22.8	20.9	18.3	15.6	14.4
	$\frac{[ct(SL_4)]}{[ct(MV)]}$	27.3	24.9	21.0	18.9	16.1	15.1
	$\frac{[ct(SL_5)]}{[ct(MV)]}$	26.4	23.3	20.8	18.6	16.9	15.6
	$\frac{[ct(SL_6)]}{[ct(MV)]}$	26.3	25.0	21.8	19.2	16.8	15.1
	$\frac{[ct(SL_7)]}{[ct(MV)]}$	26.2	24.2	22.3	19.0	17.5	15.6
	$\frac{[ct(SL_8)]}{[ct(MV)]}$	27.8	25.9	22.8	19.8	17.6	15.6

Finally, another aspect that one may consider is the **simplicity** of the implementation. As detailed above, only a single matrix multiplication is required to obtain the MINVO enclosure, which translates into a simple one line of code in most of the modern programming languages. The SLEFE computation also has a simple implementation, in this case involving sums, multiplications, $\max(\cdot)$, and $\min(\cdot)$ operators. Code examples of how to use MINVO and SLEFE are available at <https://github.com/mit-acl/minvo> (for both MINVO and SLEFE) and [179] (for SLEFE).

Chapter 3

MADER: Trajectory Planner in Multiagent and Dynamic Environments

3.1 Overview

This chapter presents MADER, a 3D decentralized and asynchronous trajectory planner for UAVs that generates collision-free trajectories in environments with static obstacles, dynamic obstacles, and other planning agents. Real-time collision avoidance with other dynamic obstacles or agents is done by performing outer polyhedral representations of every interval of the trajectories and then including the plane that separates each pair of polyhedra as a decision variable in the optimization problem. MADER uses our recently developed MINVO basis ([162]) to obtain outer polyhedral representations with volumes 2.36 and 254.9 times, respectively, smaller than the Bernstein or B-Spline bases used extensively in the planning literature. Our decentralized and asynchronous algorithm guarantees safety with respect to other agents by including their committed trajectories as constraints in the optimization and then executing a collision check-recheck scheme. Finally, extensive simulations in challenging cluttered environments show up to a 33.9% reduction in the flight time,

and a 88.8% reduction in the number of stops compared to the Bernstein and B-Spline bases, shorter flight distances than centralized approaches, and shorter total times on average than synchronous decentralized approaches.

3.2 Definitions

This chapter will use the notation shown in Table 3.1, together with the following two definitions:

- **Agent:** Element of the environment with the ability to exchange information and take decisions accordingly (i.e., an agent can change its trajectory given the information received from the environment).
- **Obstacle:** Element of the environment that moves on its own without consideration of the trajectories of other elements in the environment. An obstacle can be static or dynamic.


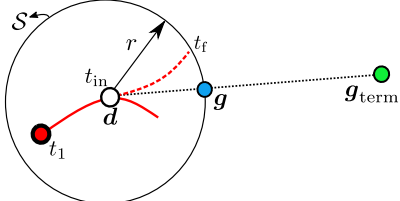
Note that here we are calling dynamic obstacles what some works in the literature call *noncooperative agents*.

This work will also use clamped uniform B-Splines, which are B-Splines defined by $n + 1$ control points $\{\mathbf{q}_0, \dots, \mathbf{q}_n\}$ and $m + 1$ knots $\{t_0, t_1, \dots, t_m\}$ that satisfy:

$$\underbrace{t_0 = \dots = t_p}_{p+1 \text{ knots}} < \underbrace{t_{p+1} < \dots < t_{m-p-1}}_{\text{Internal Knots}} < \underbrace{t_{m-p} = \dots = t_m}_{p+1 \text{ knots}}$$

and where the internal knots are equally spaced by Δt (i.e., $\Delta t := t_{k+1} - t_k \quad \forall k = \{p, \dots, m - p - 1\}$). The relationship $m = n + p + 1$ holds, and there are in total $m - 2p = n - p + 1$ intervals. Each interval $j \in J$ is defined in $t \in [t_{p+j}, t_{p+j+1}]$. In this work we will use $p = 3$ (i.e., cubic B-Splines). Hence, each interval will be a polynomial of degree 3, and it is guaranteed to lie within the convex hull of its 4 control points $\{\mathbf{q}_j, \mathbf{q}_{j+1}, \mathbf{q}_{j+2}, \mathbf{q}_{j+3}\}$. Moreover, clamped B-Splines are guaranteed to pass through the first and last control points (\mathbf{q}_0 and \mathbf{q}_n). The velocity and acceleration of

Table 3.1: Notation used in this chapter

Symbol	Meaning
$\mathbf{p}, \mathbf{v}, \mathbf{a}, \mathbf{j}$	Position, Velocity, Acceleration and Jerk, $\in \mathbb{R}^3$.
\mathbf{x}	State vector: $\mathbf{x} := [\mathbf{p}^T \mathbf{v}^T \mathbf{a}^T]^T \in \mathbb{R}^9$
m	$m + 1$ is the number of knots of the B-Spline.
n	$n + 1$ is the number of control points of the B-Spline.
p	Degree of the polynomial of each interval of the B-Spline. In this work we will use $p = 3$.
J	Set that contains the indexes of all the intervals of a B-Spline $J := \{0, 1, \dots, m - 2p - 1\}$.
ξ	$\xi :=$ Number of agents + Number of obstacles
s	Index of the planning agent.
I	Set that contains the indexes of all the obstacles/agents, except the agent s . $I := \{0, 1, \dots, \xi - 1\} \setminus s$.
L	$L = \{0, 1, \dots, n\}$.
l	Index of the control point. $l \in L$ for position, $l \in L \setminus \{n\}$ for velocity and $l \in L \setminus \{n - 1, n\}$ for acceleration.
i	Index of the obstacle/agent, $i \in I$.
j	Index of the interval, $j \in J$.
ρ	Radius of the sphere that models the agents.
B_i, B_s	B_i is the 3D axis-aligned bounding box (AABB) of the shape of the agent/obstacle i . For simplicity, we assume that the obstacles do not rotate. Hence, B_i does not change for a given obstacle/agent i . The AABB of the planning agent is denoted as B_s .
$\boldsymbol{\eta}_s$	Each entry of $\boldsymbol{\eta}_s$ is the length of each side of the AABB of the planning agent (agent whose index is s). I.e., $\boldsymbol{\eta}_s := 2 [\rho \rho \rho]^T \in \mathbb{R}^3$
\mathcal{C}_{ij}	Set of vertexes of the polyhedron that completely encloses the trajectory of the obstacle/agent i during the initial and final times of the interval j of the agent s .
\mathbf{c}	Vertex of a polyhedron, $\in \mathbb{R}^3$.
$\mathbf{q}, \mathbf{v}, \mathbf{a}$	Position, velocity, and acceleration control points, $\in \mathbb{R}^3$.
b	Notation for the basis used: MINVO ($b = \text{MV}$), Bernstein ($b = \text{Be}$), or B-Spline ($b = \text{BS}$).
\mathcal{Q}_j^b	Set that contains the 4 position control points of the interval j of the trajectory of the agent s using the basis b . $\mathcal{Q}_{j-1}^{\text{MV}} \cap \mathcal{Q}_j^{\text{MV}} = \emptyset$ in general. If $b = \text{Be}$, the last control point of interval $j - 1$ is also the first control point of interval j . If $b = \text{BS}$, the last 3 control points of interval $j - 1$ are also the first 3 control points of interval j . Analogous definition for the set \mathcal{V}_j^b , which contains the three velocity control points.
\mathbf{Q}_j^b	Matrix whose columns contain the 4 position control points of the interval j of the trajectory of the agent s using the basis b . Analogous definition for the matrix \mathbf{V}_j^b , whose columns are the three velocity control points.
$f_j^{\text{BS} \rightarrow \text{MV}}(\cdot)$	Linear function (see Eq. 3.1) such that $\mathcal{Q}_j^{\text{MV}} = f_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{Q}_j^{\text{BS}})$
$h_j^{\text{BS} \rightarrow \text{MV}}(\cdot)$	Linear function (see Eq. 3.1) such that $\mathcal{V}_j^{\text{MV}} = h_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{V}_j^{\text{BS}})$
$\boldsymbol{\pi}_{ij}(\mathbf{n}_{ij}, d_{ij})$	Plane $\mathbf{n}_{ij}^T \mathbf{x} + d_{ij} = 0$ that separates \mathcal{C}_{ij} from \mathcal{Q}_j^b .
$\mathbf{1}$, $\text{abs}(\mathbf{a})$, $\mathbf{a} \leq \mathbf{b}$, \oplus , $\text{conv}(\cdot)$	Column vector of ones, element-wise absolute value, element-wise inequality, Minkowski sum, and convex hull.
	Unless otherwise noted, this colormap in the trajectories will represent the norm of the velocity (blue 0 m/s and red v_{\max}).
Snapshot at $t = t_1$ (current time):	
	
\mathbf{g}_{term} (●) is the terminal goal, and ● is the current position of the UAV. — is the trajectory the UAV is currently executing. - - - is the trajectory the UAV is currently optimizing, starts at $t = t_{\text{in}}$ and finishes at $t = t_{\text{f}}$. \mathbf{d} (○) is a point in —, used as the initial position of - - -. S is a sphere of radius r around \mathbf{d} . - - - will be contained in S . \mathbf{g} (●) is the projection of \mathbf{g}_{term} onto the sphere S .	
$\mathbf{p}_i(t)$	Predicted trajectory of an obstacle i , and committed trajectory of agent i .
\mathcal{T}_j, γ_j	\mathcal{T}_j is a uniform discretization of $[t_{p+j}, t_{p+j+1}]$ (timespan of interval j of the trajectory of agent s) with step size γ_j and such that $t_{p+j}, t_{p+j+1} \in \mathcal{T}_j$.
$\mathbf{p}_i(\mathcal{T}_j)$	$\{\mathbf{p}_i(t) \mid t \in \mathcal{T}_j\}$

a B-Spline are B-Splines of degrees $p - 1$ and $p - 2$ respectively, whose control points are given by ([182]):

$$\begin{aligned} \mathbf{v}_l &= \frac{p(\mathbf{q}_{l+1} - \mathbf{q}_l)}{t_{l+p+1} - t_{l+1}} \quad \forall l \in L \setminus \{n\} \\ \mathbf{a}_l &= \frac{(p-1)(\mathbf{v}_{l+1} - \mathbf{v}_l)}{t_{l+p+1} - t_{l+2}} \quad \forall l \in L \setminus \{n-1, n\} \end{aligned}$$

Finally, and as shown in Table 3.1, to obtain \mathbf{g} we project the terminal goal \mathbf{g}_{term} to a sphere \mathcal{S} centered on \mathbf{d} . This is done only for simplicity, and other possible way would be to choose \mathbf{g} as the intersection between \mathcal{S} and a piecewise linear path that goes from \mathbf{d} to \mathbf{g}_{term} , and that avoids the static obstacles (and potentially the dynamic obstacles or agents as well). If a voxel grid of the environment is available, this piecewise linear path could be obtained by running a search-based algorithm, as done in [167].

3.3 Assumptions

This chapter relies on the following four assumptions:

- Let $\mathbf{p}_i^{\text{real}}(t)$ denote the real future trajectory of an obstacle i , and $\mathbf{p}_i(t)$ the one obtained by a given tracking and prediction algorithm. The smallest dimensions of the axis-aligned box D_{ij} for which

$$\mathbf{p}_i^{\text{real}}(t) \in \text{conv}(D_{ij} \oplus \mathbf{p}_i(\mathcal{T}_j)) \quad \forall t \in [t_{p+j}, t_{p+j+1}]$$

is satisfied will be denoted as $2(\boldsymbol{\alpha}_{ij} + \boldsymbol{\beta}_{ij}) \in \mathbb{R}^3$. Here, \mathcal{T}_j is a uniform discretization of $[t_{p+j}, t_{p+j+1}]$ with step size γ_j (see Table 3.1), $\boldsymbol{\alpha}_{ij}$ represents the error associated with the prediction and $\boldsymbol{\beta}_{ij}$ the one associated with the discretization of the trajectory of the obstacle. The values $\boldsymbol{\alpha}_{ij}, \boldsymbol{\beta}_{ij}$ and γ_j are assumed known. This assumption is needed to be able to obtain an outer polyhedral approximation of the Minkowski sum of a bounding box and any continuous trajectory of an obstacle (Section 3.4.3).

- Similar to other works in the literature (see [88] for instance), we assume that an agent can communicate without delay with other agents. Specifically, we assume that the planning agent has access to the committed trajectory $\mathbf{p}_i(t)$ of agent i when this condition holds:

$$\exists t \in [t_{\text{in}}, t_{\text{f}}] \text{ s.t. } (\mathbf{p}_i(t) \oplus B_i) \cap (\mathcal{S} \oplus B_s) \neq \emptyset$$

This condition ensures that the agent s knows the trajectories of the agents whose committed trajectories, inflated with their AABBs, pass through the sphere \mathcal{S} (inflated with B_s) during the interval $[t_{\text{in}}, t_{\text{f}}]$. Note also that all the agents have the same reference time, but trigger the planning iterations asynchronously.

- Two agents do not commit to a new trajectory at the very same time. Note that, as time is continuous, the probability of this assumption not being true is essentially zero. Letting t_4 denote the time when a UAV commits to a trajectory, the reason behind this assumption is to guarantee that it is safe for a UAV to commit to a trajectory at $t = t_4$ having checked all the committed trajectories of other agents at $t < t_4$ (this will be explained in detail in Section 3.6).
- Finally, we assume for simplicity that the obstacles do not rotate (and hence B_i is constant for an obstacle i). However, this is not a fundamental assumption in MADER: To take into account the rotation of the objects, one could still use MADER, but use for the inflation (Section 3.4) the largest AABB that contains all the rotations of the obstacle during a specific interval j .

3.4 Polyhedral Representations

To avoid the computational burden of imposing infinitely-many constraints to separate two trajectories, we need to compute a tight polyhedral outer representation of every interval of the optimized trajectory (trajectory that agent s is trying to obtain), the

Table 3.2: Polyhedral representations of interval j from the point of view of agent s . Here, $\mathcal{R}_{ij}^{\text{MV}}$ denotes the set of MINVO control points of every interval of the trajectory of agent i that falls in $[t_{\text{in}} + j\Delta t, t_{\text{in}} + (j + 1)\Delta t]$ (timespan of the interval j of the trajectory of agent s).

	Trajectory	Inflation	Polyhedral Repr.
Agent s	B-Spline	No Inflation	$\text{conv}(\mathcal{Q}_j^{\text{MV}})$
Other agents $i \in I$	B-Spline	$B'_i = B_i$ inflated with $\boldsymbol{\eta}_s$	$\text{conv}(B'_i \oplus \mathcal{R}_{ij}^{\text{MV}})$
Obstacles $i \in I$	Any	$B'_i = B_i$ inflated with $\boldsymbol{\eta}_s + 2(\boldsymbol{\beta}_{ij} + \boldsymbol{\alpha}_{ij})$	$\text{conv}(B'_i \oplus \mathbf{p}_i(\mathcal{T}_j))$

trajectory of the other agents and the trajectory of other obstacles (see also Table 3.2).

3.4.1 Polyhedral Representation of the Trajectory of the Agent s

When using B-Splines, one common way to obtain an outer polyhedral representation for each interval is to use the polyhedron defined by the control points of each interval. As the functions in the B-Spline basis are positive and form a partition of unity, this polyhedron is guaranteed to completely contain the interval. However, this approximation is far from being tight, leading therefore to great conservativeness both in the position and in the velocity space. To mitigate this, [157] used the Bernstein basis for the constraints in the velocity space. Although this basis generates a polyhedron smaller than the B-Spline basis, it is still conservative, as this basis does not minimize the volume of this polyhedron. We instead use both in position *and* velocity space our recently derived MINVO basis [162] that, by construction, is a polynomial basis that attempts to obtain the simplex with minimum volume that encloses a given polynomial curve. As shown in Fig. 3-1, this basis achieves a volume that is 2.36 and 254.9 times smaller (in the position space) and 1.29 and 5.19 times smaller (in the velocity space) than the Bernstein and B-Spline bases respectively. For each interval j , the vertexes of the MINVO control points (\mathbf{Q}_j^{MV} and \mathbf{V}_j^{MV} for position and velocity respectively) and the B-Spline control points (\mathbf{Q}_j^{BS} , \mathbf{V}_j^{BS}) are

related as follows:

$$\begin{aligned} \mathbf{Q}_j^{\text{MV}} &= \mathbf{Q}_j^{\text{BS}} \mathbf{A}_{\text{pos}}^{\text{BS}}(j) \left(\mathbf{A}_{\text{pos}}^{\text{MV}} \right)^{-1} \\ \mathbf{V}_j^{\text{MV}} &= \mathbf{V}_j^{\text{BS}} \mathbf{A}_{\text{vel}}^{\text{BS}}(j) \left(\mathbf{A}_{\text{vel}}^{\text{MV}} \right)^{-1} \end{aligned} \quad (3.1)$$

where the matrices \mathbf{A} are known, and are available in our recent work [162] (for the MINVO basis) and in [133] (for the Bernstein and B-Spline bases). For the B-Spline bases, and because we are using clamped uniform splines, the matrices $\mathbf{A}_{\text{pos}}^{\text{BS}}(j)$ and $\mathbf{A}_{\text{vel}}^{\text{BS}}(j)$ depend on the interval j . Eq. 3.1, together with the fact that \mathbf{V}_j^{BS} is a linear combination of \mathbf{Q}_j^{BS} , allow us to write

$$\begin{aligned} \mathcal{Q}_j^{\text{MV}} &= f_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{Q}_j^{\text{BS}}) \\ \mathcal{V}_j^{\text{MV}} &= h_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{Q}_j^{\text{BS}}) \end{aligned} \quad (3.2)$$

where $f_j^{\text{BS} \rightarrow \text{MV}}(\cdot)$ and $h_j^{\text{BS} \rightarrow \text{MV}}(\cdot)$ are known linear functions.

3.4.2 Polyhedral Representation of the Trajectory of Other Agents

We first increase the sides of B_i by $\boldsymbol{\eta}_s$ to obtain the inflated box B'_i . Now, note that the trajectory of the agent $i \neq s$ is also a B-Spline, but its initial and final times can be different from t_{in} and t_{f} (initial and final times of the trajectory that agent s is optimizing). Therefore, to obtain the polyhedral representation of the trajectory of the agent i in the intervals $[t_{\text{in}}, t_{\text{in}} + \Delta t]$, $[t_{\text{in}} + \Delta t, t_{\text{in}} + 2\Delta t]$, \dots , $[t_{\text{f}} - \Delta t, t_{\text{f}}]$ we first compute the MINVO control points of every interval of the trajectory of agent i that falls in one of these intervals. The convex hull of the boxes B'_i placed in every one of these control points will be the polyhedral representation of the interval j of the trajectory of the agent i . We denote the vertexes of this outer polyhedral representation as \mathcal{C}_{ij} .

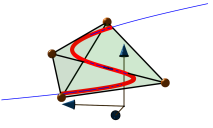
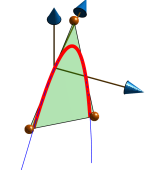
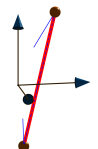
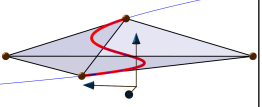
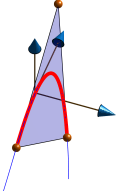
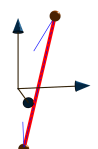

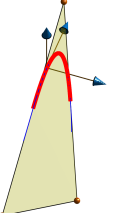
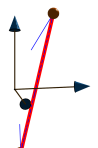
	Position	Velocity	Acceleration
MINVO	V_{MV} 	A_{MV} 	L_{MV} 
Bernstein	$V_{Be} = 2.36 V_{MV}$ 	$A_{Be} = 1.29 A_{MV}$ 	$L_{Be} = L_{MV}$ 
B-Spline	$V_{BS} = 254.88 V_{MV}$ 	$A_{BS} = 5.19 A_{MV}$ 	$L_{BS} = L_{MV}$ 

Figure 3-1: Comparison of the volumes, areas, and lengths obtained by the MINVO basis (ours), Bernstein basis (used by the Bézier curves) and B-Spline basis for an interval (—) of a given uniform B-Spline (—). In the acceleration space, the three bases generate the same control points.

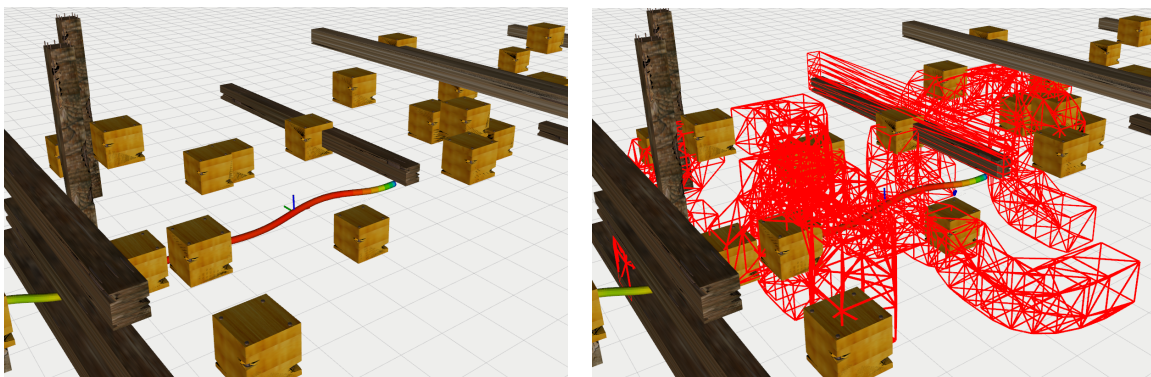


Figure 3-2: To impose collision-free constraints, MADER uses polyhedral representations of each interval of the trajectories of other agents/obstacles. On the left, a given scenario with dynamic obstacles and on the right the polyhedral representations obtained (in red).

3.4.3 Polyhedral Representation of the Trajectory of the Obstacles

For each interval j we first increase the sides of B_i by $\eta_s + 2(\beta_{ij} + \alpha_{ij})$, and denote this inflated box B'_i . Here β_{ij} and α_{ij} are the values defined in Section 3.3. We then place B'_i in $\mathbf{p}_i(\mathcal{T}_j)$, where $\mathbf{p}_i(\mathcal{T}_j)$ denotes the set of positions of the obstacle i at the times \mathcal{T}_j (see Table 3.1) and compute the convex hull of all the vertexes of these boxes. Given the first assumption of Section 3.3, this guarantees that the convex hull obtained is an outer approximation of all the 3D space occupied by the obstacle i (inflated by the size of the agent s) during the interval j . The static obstacles are treated in the same way, with $\mathbf{p}_i(t) = \text{constant}$. An example of these polyhedral representations is shown in Fig. 3-2.

3.5 Optimization and Initial Guess

3.5.1 Collision-free Constraints

Once the polyhedral approximations of the trajectories of the other obstacles/agents have been obtained, we enforce the collision-free constraints between these polyhedra and the ones of the optimized trajectory as follows: we introduce the planes π_{ij} (characterized by \mathbf{n}_{ij} and d_{ij}) that separate them as decision variables in the optimization problem and force this way the separation between the vertexes in \mathcal{C}_{ij} and the MINVO control points $\mathcal{Q}_j^{\text{MV}}$ (see Figs. 3-3 and 3-4):

$$\begin{aligned} \mathbf{n}_{ij}^T \mathbf{c} + d_{ij} &> 0 \quad \forall \mathbf{c} \in \mathcal{C}_{ij}, \forall i \in I, j \in J \\ \mathbf{n}_{ij}^T \mathbf{q} + d_{ij} &< 0 \quad \forall \mathbf{q} \in \mathcal{Q}_j^{\text{MV}}, \forall j \in J \end{aligned} \tag{3.3}$$

3.5.2 Other Constraints

The initial condition (position, velocity and acceleration) is imposed by $\mathbf{x}(t_{\text{in}}) = \mathbf{x}_{\text{in}}$. Note that \mathbf{p}_{in} , \mathbf{v}_{in} and \mathbf{a}_{in} completely determine \mathbf{q}_0 , \mathbf{q}_1 and \mathbf{q}_2 , so these control points

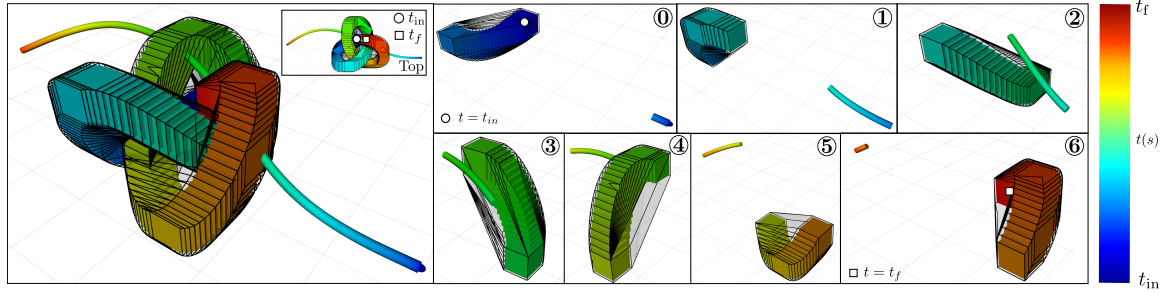


Figure 3-3: Example of a trajectory avoiding a dynamic obstacle. The obstacle has a box-like shape and is moving following a trefoil knot trajectory. The trajectory of the obstacle is divided into as many segments as the optimized trajectory has. An outer polyhedral representation (whose edges are shown as black lines) is computed for each of these segments, and each segment of the trajectory avoids these polyhedra.

are not included as decision variables.

For the final condition, we use a final stop condition imposing the constraints $\mathbf{v}(t_f) = \mathbf{v}_f = \mathbf{0}$ and $\mathbf{a}(t_f) = \mathbf{a}_f = \mathbf{0}$. These conditions require $\mathbf{q}_{n-2} = \mathbf{q}_{n-1} = \mathbf{q}_n$, so the control points \mathbf{q}_{n-1} and \mathbf{q}_n can also be excluded from the set of decision variables. The final position is included as a penalty cost $\|\mathbf{q}_{n-2} - \mathbf{g}\|_2^2$ in the objective function, weighted with a parameter $\omega \geq 0$. Here \mathbf{g} is the goal (projection of the \mathbf{g}_{term} onto a sphere \mathcal{S} of radius r around \mathbf{d} , see Table 3.1). Note that, as we are using clamped uniform B-Splines with a final stop condition, \mathbf{q}_{n-2} coincides with the last position of the B-Spline. The reason of adding this penalty cost for the final position, instead of including $\mathbf{q}_{n-2} = \mathbf{g}$ as a hard constraint, is that a hard constraint can easily lead to infeasibility if the heuristics used for the total time $(t_f - t_{\text{in}})$ underestimates the time needed to reach \mathbf{g} .

To force the trajectory generated to be inside the sphere \mathcal{S} , we impose the constraint

$$\|\mathbf{q} - \mathbf{d}\|_2^2 \leq r^2 \quad \forall \mathbf{q} \in \mathcal{Q}_j^{\text{MV}}, \forall j \in J \quad (3.4)$$

Moreover, we also add the constraints on the maximum velocity and acceleration:

$$\begin{aligned} \text{abs}(\mathbf{v}) &\leq \mathbf{v}_{\text{max}} \quad \forall \mathbf{v} \in \mathcal{V}_j^{\text{MV}}, \forall j \in J \\ \text{abs}(\mathbf{a}_l) &\leq \mathbf{a}_{\text{max}} \quad \forall l \in L \setminus \{n-1, n\} \end{aligned} \quad (3.5)$$

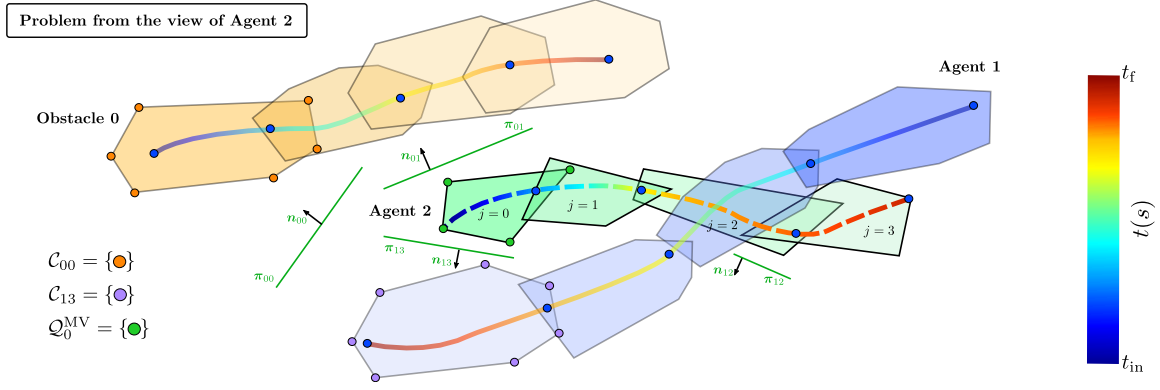


Figure 3-4: Collision-free constraints between agent 2 and both the obstacle 0 and agent 1. This figure is from the view of agent 2. t_{in} and t_{f} are the initial and final times of the trajectory being optimized (dashed lines), and they are completely independent of the initial and final optimization times of agent 1. $\mathcal{Q}_0^{\text{MV}}$ are the control points of the interval 0 of the optimized trajectory using the MINVO basis. \mathcal{C}_{13} are the vertexes of the convex hull of the vertexes of the control points of all the intervals of the trajectory of agent 1 that fall in $[t_{\text{f}} - \Delta t, t_{\text{f}}]$. Note that the trajectories and polyhedra are in 3D, but they are represented in 2D for visualization purposes.

where we are using the MINVO velocity control points for the velocity constraint. For the acceleration constraint, the B-Spline and MINVO control points are the same (see Fig. 3-1). Note that the velocity and acceleration are constrained independently on each one of the axes $\{x, y, z\}$.

3.5.3 Control Effort

The evaluation of a cubic clamped uniform B-Spline in an interval $j \in J$ can be done as follows [133]:

$$\mathbf{p}(t) = \mathbf{Q}_j^{\text{BS}} \underbrace{\mathbf{A}_{\text{pos}}^{\text{BS}}(j)}_{:=\mathbf{u}_j} \begin{bmatrix} u_j^3 \\ u_j^2 \\ u_j \\ 1 \end{bmatrix}$$

where $u_j := \frac{t-t_{p+j}}{t_{p+j+1}-t_{p+j}}$, $t \in [t_{p+j}, t_{p+j+1}]$ and $\mathbf{A}_{\text{pos}}^{\text{BS}}(j)$ is a known matrix that depends on each interval. Specifically, and with the knots chosen, we will have $\mathbf{A}_{\text{pos}}^{\text{BS}}(0) \neq \mathbf{A}_{\text{pos}}^{\text{BS}}(1) \neq \mathbf{A}_{\text{pos}}^{\text{BS}}(2) = \dots = \mathbf{A}_{\text{pos}}^{\text{BS}}(m-2p-3) \neq \mathbf{A}_{\text{pos}}^{\text{BS}}(m-2p-2) \neq \mathbf{A}_{\text{pos}}^{\text{BS}}(m-2p-1)$.

Now, note that

$$\frac{d^r \mathbf{p}(t)}{dt^r} = \frac{1}{\Delta t^r} \mathbf{Q}_j^{\text{BS}} \mathbf{A}_{\text{pos}}^{\text{BS}}(j) \frac{d^r \mathbf{u}_j}{du_j^r}$$

Therefore, as the jerk is constant in each interval (since $p = 3$), the control effort is:

$$\int_{t_{\text{in}}}^{t_{\text{f}}} \|\mathbf{j}(t)\|^2 dt \propto \sum_{j \in J} \left\| \mathbf{Q}_j^{\text{BS}} \mathbf{A}_{\text{pos}}^{\text{BS}}(j) \begin{bmatrix} 6 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2 \quad (3.6)$$

3.5.4 Optimization Problem

Given the constraints and the objective function explained above, the optimization problem solved is as follows:¹

$$\begin{aligned} & \min_{\mathbf{Q}_j^{\text{BS}}, \mathbf{n}_{ij}, d_{ij}} \sum_{j \in J} \left\| \mathbf{Q}_j^{\text{BS}} \mathbf{A}_{\text{pos}}^{\text{BS}}(j) \begin{bmatrix} 6 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2 + \omega \|\mathbf{q}_{n-2} - \mathbf{g}\|_2^2 \\ & \text{s.t.} \\ & \mathbf{x}(t_{\text{in}}) = \mathbf{x}_{\text{in}} \\ & \mathbf{v}(t_{\text{f}}) = \mathbf{v}_f = \mathbf{0} \\ & \mathbf{a}(t_{\text{f}}) = \mathbf{a}_f = \mathbf{0} \\ & \mathbf{n}_{ij}^T \mathbf{c} + d_{ij} > 0 \quad \forall \mathbf{c} \in \mathcal{C}_{ij}, \forall i, j \\ & \mathbf{n}_{ij}^T \mathbf{q} + d_{ij} < 0 \quad \forall \mathbf{q} \in \mathcal{Q}_j^{\text{MV}} := f_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{Q}_j^{\text{BS}}), \forall i, j \\ & \|\mathbf{q} - \mathbf{d}\|_2^2 \leq r^2 \quad \forall \mathbf{q} \in \mathcal{Q}_j^{\text{MV}} := f_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{Q}_j^{\text{BS}}), \forall j \\ & \text{abs}(\mathbf{v}) \leq \mathbf{v}_{\text{max}} \quad \forall \mathbf{v} \in \mathcal{V}_j^{\text{MV}} := h_j^{\text{BS} \rightarrow \text{MV}}(\mathcal{Q}_j^{\text{BS}}), \forall j \\ & \text{abs}(\mathbf{a}_l) \leq \mathbf{a}_{\text{max}} \quad \forall l \in L \setminus \{n-1, n\} \end{aligned}$$

This problem is clearly nonconvex since we are minimizing over the control points *and* the planes π_{ij} (characterized by \mathbf{n}_{ij} and d_{ij}). Note also that the decision variables are the B-Spline control points $\mathcal{Q}_j^{\text{BS}}$. In the constraints, the MINVO control

¹In the optimization problem, $\forall i$ and $\forall j$ denote, respectively, $\forall i \in I$ and $\forall j \in J$.

Algorithm 1: Octopus Search

```

1 Function GetInitialGuess():
2   Compute  $\mathbf{q}_0$ ,  $\mathbf{q}_1$  and  $\mathbf{q}_2$  from  $\mathbf{p}_{in}$ ,  $\mathbf{v}_{in}$  and  $\mathbf{a}_{in}$ 
3   Add  $\mathbf{q}_2$  to  $Q$ 
4   while  $Q$  is not empty do
5      $\mathbf{q}_l \leftarrow$  First element of  $Q$ 
6     Remove first element of  $Q$ 
7      $\mathcal{M} \leftarrow$  Uniformly sample  $\mathbf{v}_l$  satisfying  $v_{max}$  and  $a_{max}$ 
8     if any of the conditions 1-6 is true then
9       continue
10    if  $\|\mathbf{q}_l - \mathbf{g}\|_2 < \epsilon''$  and  $l = (n - 2)$  then
11       $\mathbf{q}_{n-1} \leftarrow \mathbf{q}_{n-2}$ 
12       $\mathbf{q}_n \leftarrow \mathbf{q}_{n-2}$ 
13      return  $\{\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_{n-2}, \mathbf{q}_{n-1}, \mathbf{q}_n\} \cup \pi_{ij}$ 
14    for every  $\mathbf{v}_l$  in  $\mathcal{M}$  do
15       $\mathbf{q}_{l+1} \leftarrow \mathbf{q}_l + \frac{t_{l+p+1} - t_{l+1}}{p} \mathbf{v}_l$ 
16      Store in  $\mathbf{q}_{l+1}$  a pointer to  $\mathbf{q}_l$ 
17      Add  $\mathbf{q}_{l+1}$  to  $Q$ 
18  return Closest Path found

```

points $\mathcal{Q}_j^{\text{MV}}$ and $\mathcal{V}_j^{\text{MV}}$ are simply linear transformations of the decision variables (see Eq. 3.2). We solve this problem using the augmented Lagrangian method [15, 31], and with the globally-convergent method-of-moving-asymptotes (MMA) [152] as the subsidiary optimization algorithm. The interface used for these algorithms is NLOpt [51]. The time allocated per trajectory is chosen before the optimization as $(t_f - t_{in}) = \frac{\|\mathbf{g} - \mathbf{d}\|_2}{v_{max}}$.

3.5.5 Initial Guess

To obtain an initial guess (which consists of both the control points $\{\mathbf{q}_0, \dots, \mathbf{q}_n\}^{\text{BS}}$ and the planes π_{ij}), we use the Octopus Search algorithm shown in Alg. 1. The Octopus Search takes inspiration from A* [59], but it is designed to work with B-Splines, handle dynamic obstacles/agents, and use the MINVO basis for the collision check. Each control point will be a node in the search. All the open nodes are kept in a priority queue Q , in which the elements are ordered in increasing order of $f = g + \epsilon h$, where g is the sum of the distances (between successive control points) from \mathbf{q}_0 to the current node (cost-to-come), h is the distance from the current node to the goal

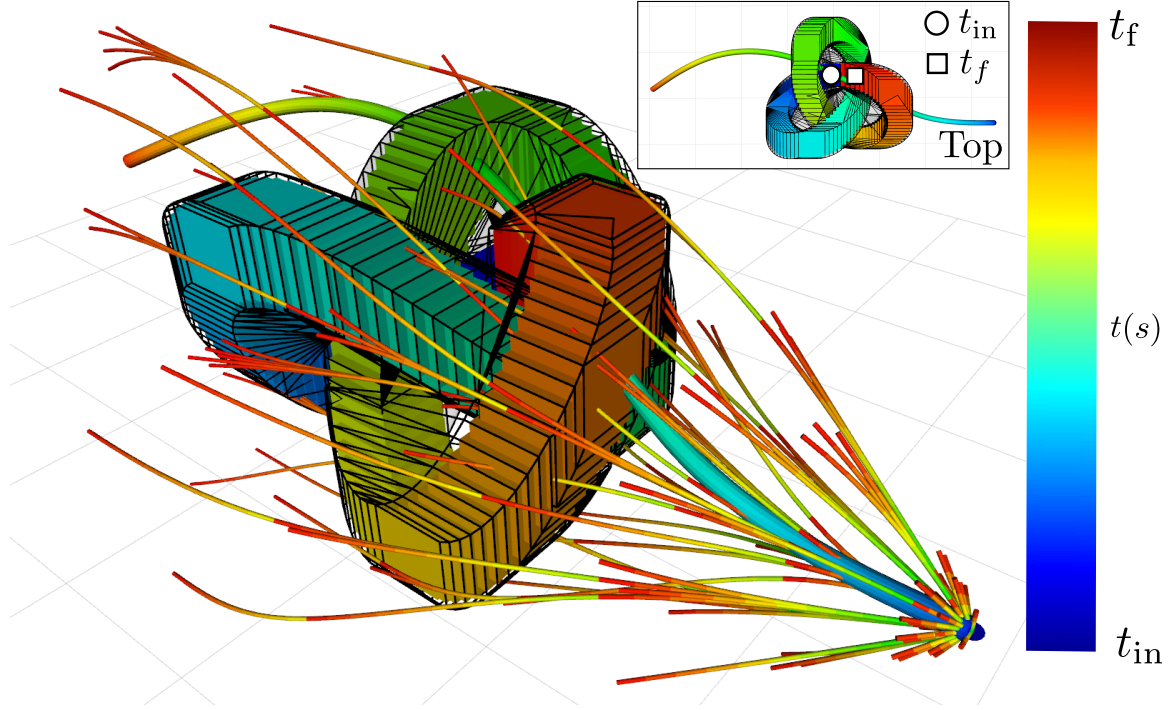


Figure 3-5: Example of the trajectories found by the Octopus Search in an environment with a dynamic obstacle following a trefoil knot trajectory. The best trajectory found is the thickest one in the figure.

(heuristics of the cost-to-go), and ϵ is the bias. Similar to A^* , this ordering of the priority queue makes nodes with lower f be explored first.

The way the algorithm works is as follows: First, we compute the control points $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2$, which are determined from $\mathbf{p}_{\text{in}}, \mathbf{v}_{\text{in}}$ and \mathbf{a}_{in} . After adding \mathbf{q}_2 to the queue Q (line 3), we run the following loop until there are no elements in Q : First we store in \mathbf{q}_l the first element of Q , and remove it from Q (lines 5-6). Then, we store in a set \mathcal{M} velocity samples for \mathbf{v}_l that satisfy both v_{max} and a_{max} .² After this, we discard the current \mathbf{q}_l if any of these conditions are true (l.s. denotes linearly separable):

1. $\mathcal{Q}_{l-3}^{\text{MV}}$ is not l.s. from $\mathcal{C}_{i,l-3}$ for some $i \in I$.
2. $l = (n - 2)$ and $\mathcal{Q}_{n-4}^{\text{MV}}$ is not l.s. from $\mathcal{C}_{i,n-4}$ for some $i \in I$.

²In these velocity samples, we use the B-Spline velocity control points, to avoid the dependence with past velocity control points that appears when using the MINVO or Bernstein bases. But note that this is only for the initial guess, in the optimization problem the MINVO velocity control points are used.

3. $l = (n - 2)$ and $\mathcal{Q}_{n-3}^{\text{MV}}$ is not l.s. from $\mathcal{C}_{i,n-3}$ for some $i \in I$.
4. $\|\mathbf{q}_l - \mathbf{d}\|_2 > r$.
5. $\|\mathbf{q}_l - \mathbf{q}_k\|_\infty \leq \epsilon'$ for some \mathbf{q}_k already added to Q .
6. Cardinality of \mathcal{M} is zero.

Condition 1 ensures that the convex hull of $\mathcal{Q}_{l-3}^{\text{MV}}$ does not collide with any interval $l - 3$ of other obstacle/agent $i \in I$. The linear separability is checked by solving the following feasibility linear problem for the interval $j = l - 3$ of every obstacle/agent $i \in I$:

$$\begin{aligned} \mathbf{n}_{ij}^T \mathbf{c} + d_{ij} &> 0 \quad \forall \mathbf{c} \in \mathcal{C}_{ij} \\ \mathbf{n}_{ij}^T \mathbf{q} + d_{ij} &< 0 \quad \forall \mathbf{q} \in \mathcal{Q}_j^{\text{MV}} \end{aligned} \tag{3.7}$$

where the decision variables are the planes $\boldsymbol{\pi}_{ij}$ (defined by \mathbf{n}_{ij} and d_{ij}). We solve this problem using GLPK [3]. Note that we also need to check the conditions 2 and 3 due to the fact that $\mathbf{q}_{n-2} = \mathbf{q}_{n-1} = \mathbf{q}_n$ and hence the choice of \mathbf{q}_{n-2} in the search forces the choice of \mathbf{q}_{n-1} and \mathbf{q}_n . In all these three previous conditions, the MINVO control points are used.

As in the optimization problem we are forcing the trajectory to be inside the sphere \mathcal{S} , we also discard \mathbf{q}_l if condition 4 is not satisfied. Additionally, to keep the search computationally tractable, we discard \mathbf{q}_l if it is very close to another \mathbf{q}_k already added to Q (condition 5): we create a voxel grid of voxel size $2\epsilon'$, and add a new control point to Q only if no other point has been added before within the same voxel. Finally, we also discard \mathbf{q}_l if there are not any feasible samples for \mathbf{v}_l (condition 6).

Then, we check if we have found all the control points and if \mathbf{q}_{n-2} is sufficiently close to the goal \mathbf{g} (distance less than ϵ''). If this is the case, the control points \mathbf{q}_{n-1} and \mathbf{q}_n (which are the same as \mathbf{q}_{n-2} due to the final stop condition) are added to the list of the corresponding control points, and are returned together with all the separating planes $\boldsymbol{\pi}_{ij} \forall i \in I, \forall j \in J$ (lines 10-13). If the goal has not been reached yet, we use the velocity samples \mathcal{M} to generate \mathbf{q}_{l+1} and add them to Q (lines 14-17).

If the algorithm is not able to find a trajectory that reaches the goal, the one found that is closest to the goal is returned (line 18).

Fig. 3-5 shows an example of the trajectories found by the Octopus Search algorithm in an environment with a dynamic obstacle following a trefoil knot trajectory.

3.5.6 Degree of the Splines

In this work, we focused on the case $p = 3$ (i.e., cubic splines). However, MADER could also be used with higher (or lower) order splines. For instance, one could use splines of fourth-degree polynomials (i.e., $p = 4$), minimize snap (instead of jerk), and then use the corresponding MINVO polyhedron that encloses each fourth-degree interval for the obstacle avoidance constraints [162]. The reason behind the choice of cubic splines, instead of higher/lower order splines, is that cubic splines are a good trade-off between dynamic feasibility of a UAV [115] and computational tractability.

3.6 Deconfliction

To guarantee that the agents plan trajectories asynchronously while not colliding with other agents that are also constantly replanning, we use a deconfliction scheme divided in these three periods (see Fig. 3-6):

- The **Optimization** period happens during $t \in (t_1, t_2]$. The optimization problem will include the polyhedral outer representations of the trajectories $p_i(t)$, $i \in I$ in the constraints. All the trajectories other agents commit to during the Optimization period are stored.
- The **Check** happens during $t \in (t_2, t_3]$. The goal of this period is to check whether the trajectory found in the optimization collides with the trajectories other agents have committed to during the optimization. This collision check is done by performing feasibility tests solving the Linear Program 3.7 $\forall j$ for every agent i that has committed to a trajectory while the optimization was being

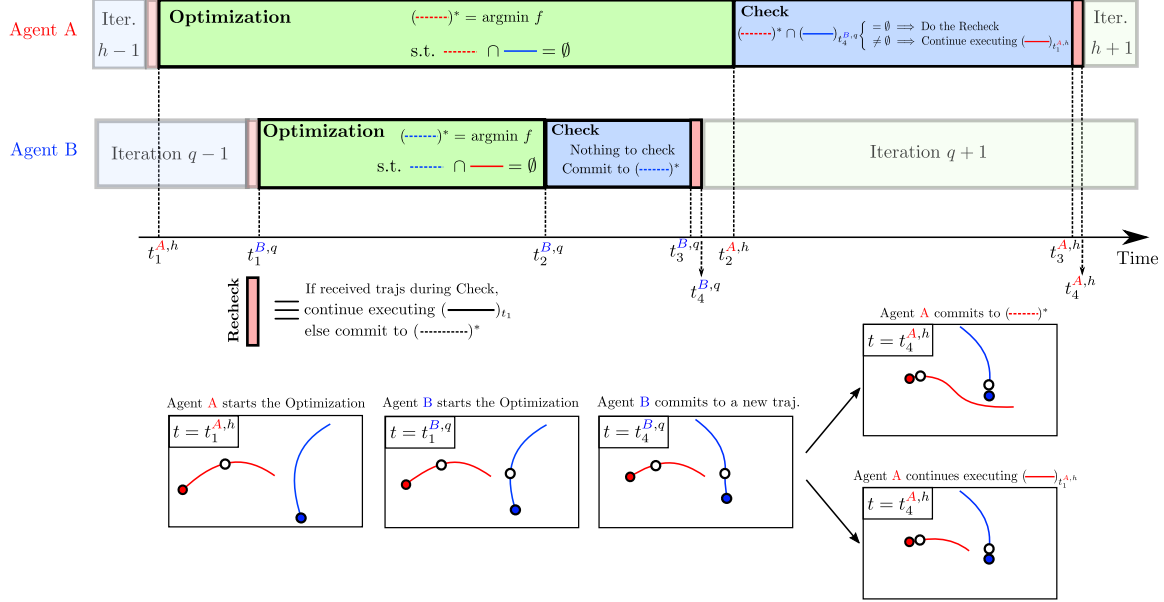


Figure 3-6: Deconfliction between agents. Each agent includes the trajectories other agents have committed to as constraints in the optimization. After the optimization, a collision check-recheck scheme is performed to ensure feasibility with respect to trajectories other agents have committed to while the optimization was happening. In this example, agent B starts the optimization after agent A, but commits to a trajectory before agent A. Hence, when agent A finishes the optimization it needs to check whether the trajectory found collides or not with the trajectory agent B committed to at $t = t_4^{B,q}$. If it collides, agent A will simply keep executing the trajectory available at $t = t_1^{A,h}$. If it does not collide, agent A will do the Recheck step to ensure no agent has committed to any trajectory during the Check period, and if this Recheck step is satisfied, agent A will commit to the trajectory found.

performed (and whose new trajectory was not included in the constraints at t_1). A boolean flag is set to true if any other agent commits to a new trajectory during this Check period.

- The **Recheck** period aims at checking whether agent A has received any trajectory during the Check period, by simply checking if the boolean flag is true or false. As this is a single Boolean comparison in the code, it allows us to assume that no trajectories have been published by other agents while this recheck is done, avoiding therefore an infinite loop of rechecks.

With h denoting the replanning iteration of an agent A, the time allocation for each of these three periods described above is explained in Fig. 3-7: to choose the

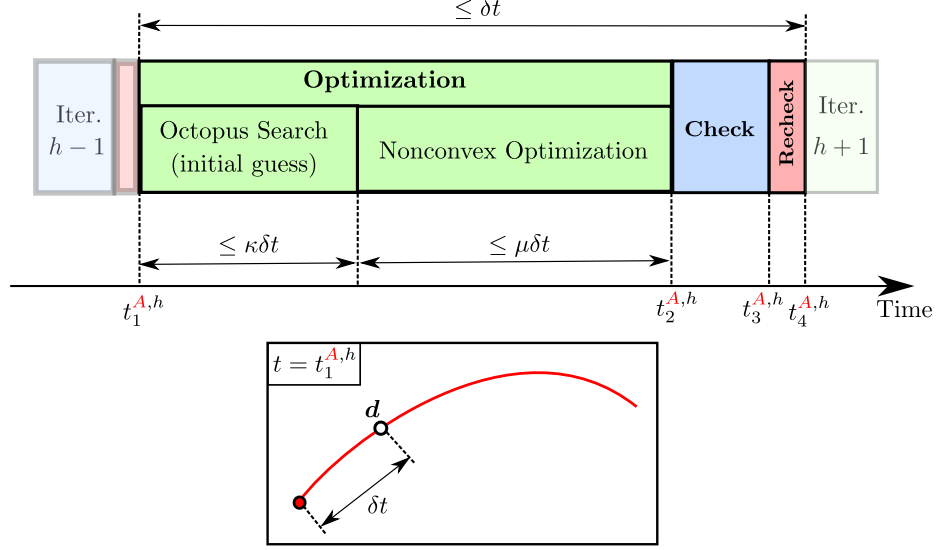


Figure 3-7: At $t = t_1^{A,h}$, agent **A** chooses the point \mathbf{d} (\circ) along the current trajectory that it is executing, with an offset δt from the current position \bullet . Then, it allocates $\kappa\delta t$ seconds to obtain an initial guess. The closest trajectory found to \mathbf{g} is used as the initial guess if the search has not finished by that time. Then, the nonconvex optimization runs for $\mu\delta t$ seconds, choosing the best feasible solution found if no local optimum has been found by then. κ and μ satisfy $\kappa > 0$, $\mu > 0$, $\kappa + \mu < 1$.

initial condition of the iteration h , Agent **A** first chooses a point \mathbf{d} along the trajectory found in the iteration $h - 1$, with an offset of δt seconds from the current position \bullet . Here, δt should be an estimate of how long iteration h will take. To obtain this estimate, and similar to our previous work [167], we use the time iteration $h - 1$ took multiplied by a factor $\alpha \geq 1$: $\delta t = \alpha (t_4^{A,h-1} - t_1^{A,h-1})$. Agent **A** then should finish the replanning iteration h in less than δt seconds. To do this, we allocate a maximum runtime of $\kappa\delta t$ seconds to obtain an initial guess, and a maximum runtime of $\mu\delta t$ seconds for the nonconvex optimization. Here $\kappa > 0$, $\mu > 0$ and $\kappa + \mu < 1$, to give time for the Check and Recheck. If the Octopus Search takes longer than $\kappa\delta t$, the trajectory found that is closest to the goal is used as the initial guess. Similarly, if the nonconvex optimization takes longer than $\mu\delta t$, the best feasible solution found is selected.

Fig. 3-6 shows an example scenario with only two agents **A** and **B**. Agent **A** starts its h -th replanning step at $t_1^{A,h}$, and finishes the optimization at $t_2^{A,h}$. Agent **B** starts its q -th replanning step at $t_1^{B,q}$. In the example shown, agent **B** starts the

optimization later than agent A ($t_1^{B,q} > t_1^{A,h}$), but solves the optimization earlier than agent A ($t_2^{B,q} < t_2^{A,h}$). As no other agent has obtained a trajectory while agent B was optimizing, agent B does not have to check anything, and commits directly to the trajectory found. However, when agent A finishes the optimization at $t_2^{A,h} > t_4^{B,q}$, it needs to check if the trajectory found collides with the one agent B has committed to. If they do not collide, agent A will perform the Recheck by ensuring that no trajectory has been published while the Check was being performed.

An agent will keep executing the trajectory found in the previous iteration if any of these four scenarios happens:

1. The trajectory obtained at the end of the optimization collides with any of the trajectories received during the Optimization.
2. The agent has received any trajectory from other agents during the Check period.
3. No feasible solution has been found in the Optimization.
4. The current iteration takes longer than δt seconds.

Under the third assumption explained in Section 3.3 (i.e., two agents do not commit to their trajectory at the very same time), this deconfliction scheme explained guarantees safety with respect to the other agents, which is proven as follows:

- If the planning agent commits to a new trajectory in the current replanning iteration, this new trajectory is guaranteed to be collision-free because it included all the trajectories of other agents as constraints, and it was checked for collisions with respect to the trajectories other agents have committed to during the planning agent’s optimization time.
- If the planning agent does not commit to a new trajectory in that iteration (because one of the scenarios 1–4 occur), it will keep executing the trajectory found in the previous iteration. This trajectory is still guaranteed to be collision-free because it was collision-free when it was obtained and other agents have

included it as a constraint in any new plans that have been made recently. If the agent reaches the end of this trajectory (which has a final stop condition), the agent will wait there until it obtains a new feasible solution. In the meanwhile, all the other agents are including its position as a constraint for their trajectories, guaranteeing therefore safety between the agents.

3.7 Results

We now test MADER in several single-agent and multiagent simulation environments. The computers used for the simulations are an *AlienWare Aurora r8* desktop (for the C++ simulations of 3.7.2), a *ROG Strix GL502VM* laptop (for the Matlab simulations of Section 3.7.2) and a *general-purpose-N1* Google Cloud instance (for the simulations of Sections 3.7.1 and 3.7.3).

3.7.1 Single-Agent Simulations

To highlight the benefits of the MINVO basis with respect to the Bernstein or B-Spline bases, we first run the algorithm proposed in a corridor-like environment ($73 \text{ m} \times 4 \text{ m} \times 3 \text{ m}$) depicted in Fig. 3-8 that contains 100 randomly deployed dynamic obstacles of sizes $0.8 \text{ m} \times 0.8 \text{ m} \times 0.8 \text{ m}$. All the obstacles follow a trajectory whose parametric equations are those of a trefoil knot [110]. The radius of the sphere \mathcal{S} used is $r = 4.0 \text{ m}$, and the velocity and acceleration constraints for the UAV are $\mathbf{v}_{max} = 5 \cdot \mathbf{1} \text{ m/s}$ and $\mathbf{a}_{max} = [20 \ 20 \ 9.6]^T \text{ m/s}^2$. The velocity profile for v_x is shown in Fig. 3-9. For the same given velocity constraint ($v_{max} = 5 \text{ m/s}$), the mean velocity v_x achieved by the MINVO basis is 4.15 m/s , higher than the ones achieved by the Bernstein and B-Spline basis (3.23 m/s and 2.79 m/s respectively).

We now compare the time it takes for the UAV to reach the goal using each basis in the same corridor environment but varying the total number of obstacles (from 50 obstacles to 250 obstacles). Moreover, and as a stopping condition is not a safe condition in a world with dynamic obstacles, we also report the number of times the UAV had to stop. The results are shown in Table 3.3 and Fig. 3-10. In terms

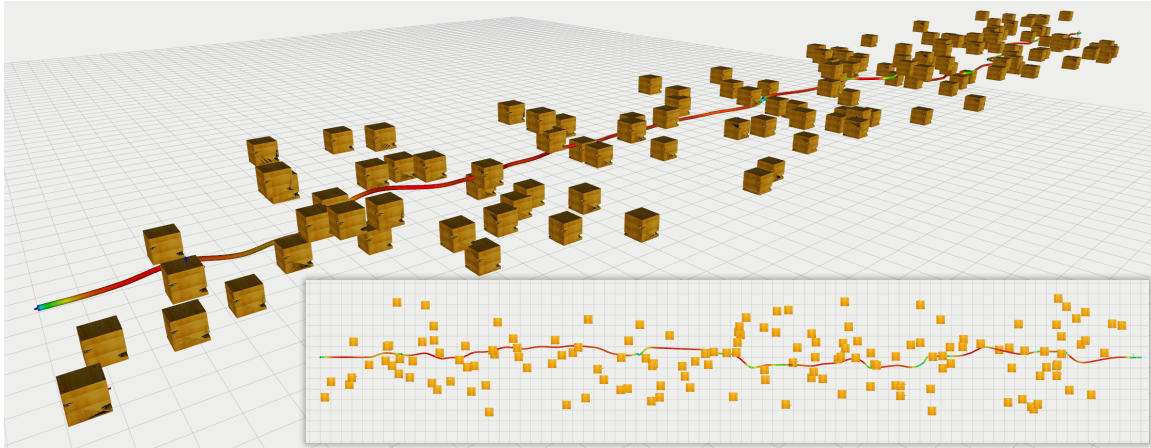


Figure 3-8: Corridor environment of size $73 \text{ m} \times 4 \text{ m} \times 3 \text{ m}$ used for the single-agent simulation. It contains 100 randomly deployed dynamic obstacles that follow a trefoil knot trajectory. The corridor is along the x direction.

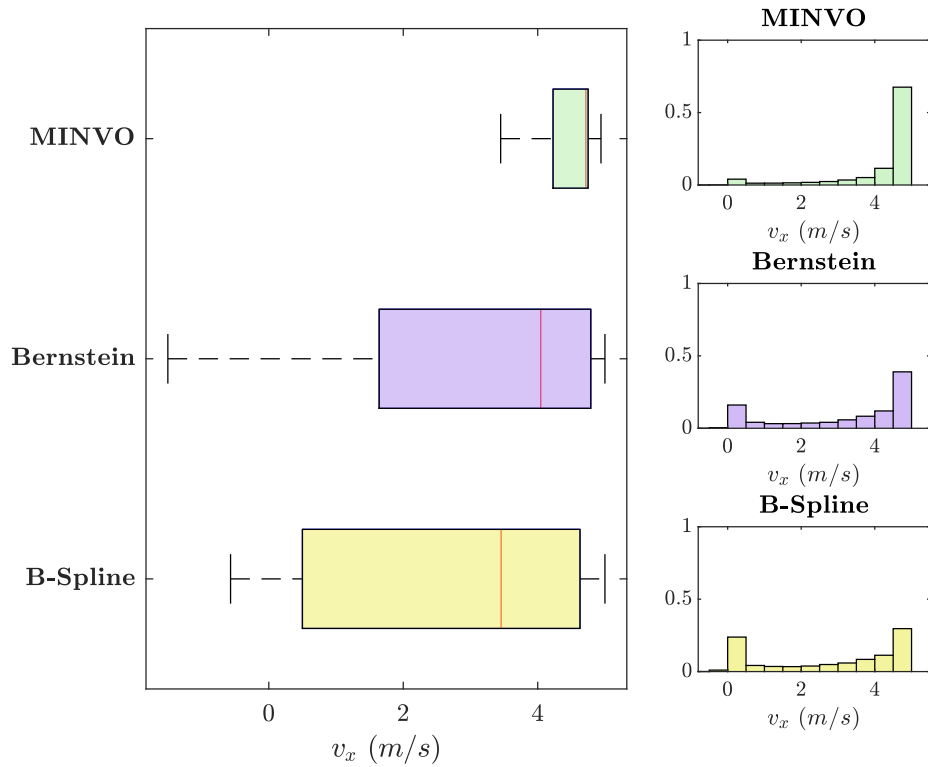


Figure 3-9: Boxplots and normalized histograms of the velocity profile of v_x in the corridor environment shown in Fig. 3-8. The velocity constraint used was $v_{max} = 5 \cdot \mathbf{1} \text{ m/s}$. The histograms are for all the velocities obtained across ten different simulations.

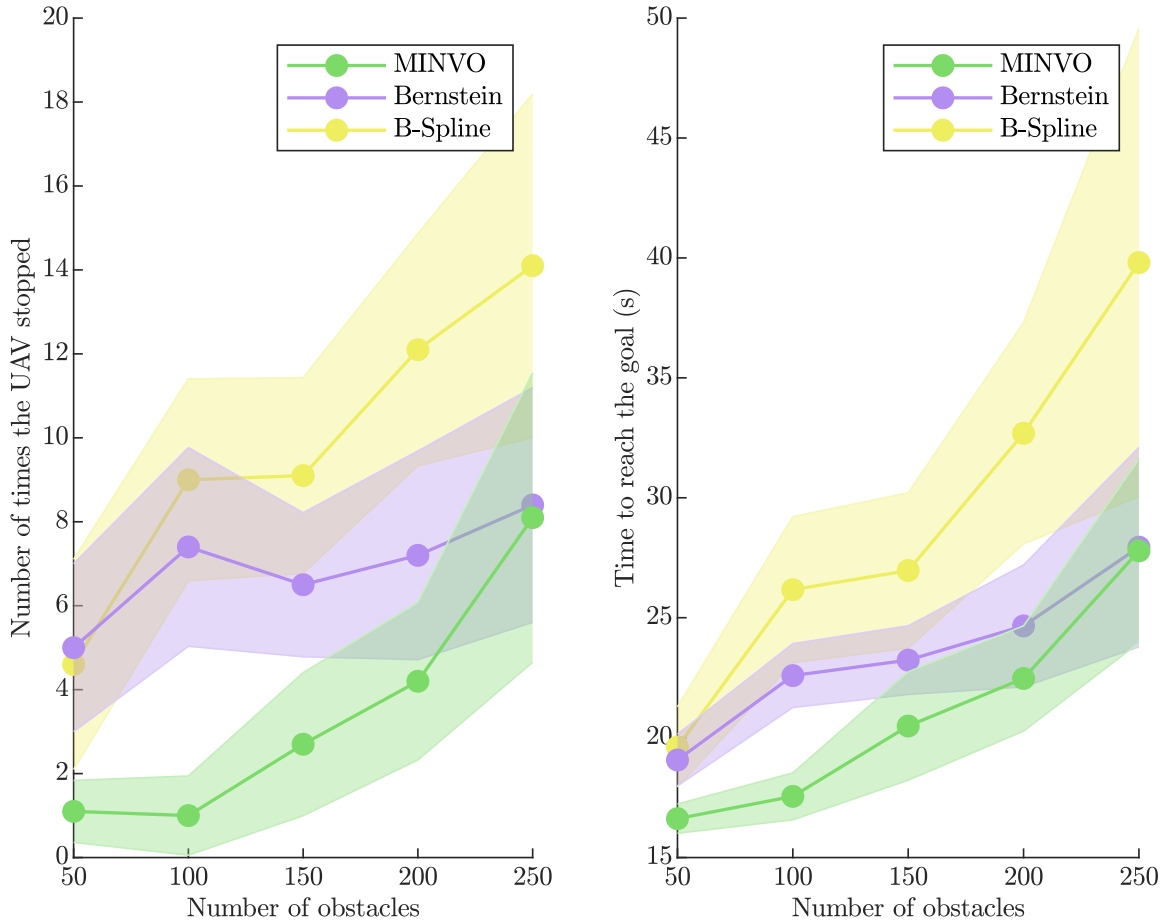


Figure 3-10: Time to reach the goal and number of times the UAV had to stop for different number of obstacles. 5 simulations were performed for each combination of basis (MINVO, Bernstein and B-Spline) and number of obstacles. The shaded area is the 1σ interval, where σ is the standard deviation.

of number of stops, the use of the MINVO basis achieves reductions of 86.4% and 88.8% with respect to the Bernstein and B-Spline bases respectively. In terms of the time to reach the goal, the MINVO basis achieves reductions of 22.3% and 33.9% compared to the Bernstein and B-Spline bases respectively. The reason behind all these improvements is the tighter outer polyhedral approximation of each interval of the trajectory achieved by the MINVO basis in the velocity and position spaces.

3.7.2 Multiagent Simulations Without Obstacles

We now compare MADER with the following different state-of-the-art algorithms:

Table 3.3: Comparison of the number of stops and time to reach the goal in a corridor-like environment using different bases and with different number of obstacles.

Basis	50 obstacles		100 obstacles		150 obstacles		200 obstacles		250 obstacles	
	Stops	Time (s)	Stops	Time (s)	Stops	Time (s)	Stops	Time (s)	Stops	Time (s)
B-Spline	4.6 ± 2.50	19.61 ± 1.67	9.0 ± 2.40	26.17 ± 3.04	9.1 ± 2.31	26.97 ± 3.24	12.1 ± 2.77	32.69 ± 4.61	14.10 ± 4.09	39.81 ± 9.76
Bernstein	5.0 ± 2.0	19.07 ± 1.08	7.4 ± 2.36	22.59 ± 1.33	6.5 ± 1.72	23.23 ± 1.43	7.2 ± 2.49	24.65 ± 2.54	8.4 ± 2.80	27.94 ± 4.16
MINVO (ours)	1.1 ± 0.74	16.62 ± 0.61	1 ± 0.94	17.55 ± 0.98	2.7 ± 1.70	20.48 ± 2.26	4.2 ± 1.87	22.47 ± 2.19	8.1 ± 3.45	27.78 ± 3.83

- Sequential convex programming (SCP,³ [10]).
- Relative Bernstein Polynomial approach (RBP,³ [126]).
- Distributed model predictive control (DMPC,⁴ [97]).
- Decoupled incremental sequential convex programming (dec_iSCP,⁴ [28]).
- Search-based motion planing⁵ [88], both in its sequential version (decS_Search) and in its non-sequential version (decNS_Search).

To classify these different algorithms, we use the following definitions:

- **Decentralized:** Each agent solves its own optimization problem.
- **Replanning:** The agents have the ability to plan several times as they fly (instead of planning only once before starting to fly). The algorithms with replanning are also classified according to whether they satisfy the real-time constraint in the replanning: algorithms that satisfy this constraint are able to replan in less than δt or at least have a trajectory they can keep executing in case no solution has found by then (see Fig. 3-7). Algorithms that do not satisfy this constraint allow replanning steps longer than δt (which is **not** feasible in the real world), and simulations are performed by simply having a simulation time that runs completely independent of the real time.
- **Asynchronous:** The planning is triggered independently by each agent without considering the planning status of other agents. Examples of synchronous algorithms include the ones that trigger the optimization of all the agents at

³https://github.com/qwerty35/swarm_simulator

⁴https://github.com/carlosluisg/multiagent_planning

⁵https://github.com/sikang/mpl_ros

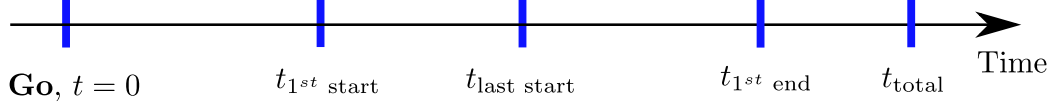


Figure 3-11: Time notation for the algorithms that have replanning.

the same time or that impose that one agent cannot plan until another agent has finished.

- **Discretization for inter-agent constraints:** The collision-free constraints between the agents are imposed only on a finite set of points of the trajectories. The discretization step will be denoted as h seconds.

In the test scenario, 8 agents are in a 8×8 m square, and they have to swap their positions. The velocity and acceleration constraints used are $\mathbf{v}_{max} = 1.7 \cdot \mathbf{1}$ m/s and $a_{max} = 6.2 \cdot \mathbf{1}$ m/s², with a drone radius of 15 cm. Moreover, we define the safety ratio as $\min_{i,i'} d_{min}^{i,i'} / (\rho_i + \rho_{i'})$ [126], where $d_{min}^{i,i'}$ is the minimum distance over all the pairs of agents i and i' , and $\rho_i, \rho_{i'}$ denote their respective radii. Safety is ensured if safety ratio > 1 . For the RBP and DMPC algorithms, the downwash coefficient c was set to $c = 1$ (so that the drone is modeled as a sphere as in all the other algorithms).

The results obtained, together with the classification of each algorithm, are shown in Table 3.4. For the algorithms that replan as they fly, we show the following times (see Fig. 3-11): $t_{1^{st}start}$ (earliest time a UAV starts flying), $t_{last start}$ (latest time a UAV starts flying), $t_{1^{st}end}$ (earliest time a UAV reaches the goal) and the total time t_{total} (time when all the UAVs have reached their goals). Note that the algorithm `decNS_Search` is synchronous ([link](#)), and it does not satisfy the real-time constraints in the replanning iterations ([link](#)). Several conclusions can be drawn from Table 3.4:

- Algorithms that use discretization to impose inter-agent constraints are in general not safe due to the fact that the constraints may not be satisfied between two consecutive discretization points. A smaller discretization step may solve this, but at the expense of very high computation times.
- Compared to the centralized solution that generates safe trajectories (RBP), MADER achieves a shorter overall flight distance. The total time of MADER

Table 3.4: Comparison between MADER (ours), SCP ([10]), RBP ([126]), DMPC ([97]), dec_iSCP ([28]), decS_Search and decNS_Search ([88]) . For SCP and MADER, the time and distance results are the mean of 5 runs, and the safety ratio is the minimum across all the runs. The test environment consists of 8 agents in a square that swap their positions without obstacles. For the algorithms that have replanning, the values in the columns of computation and execution times are $t_{1^{st}start} | t_{last\ start} | t_{1^{st}end}$ (see Fig. 3-11). The superscript * means the available implementation of the algorithm is in MATLAB (rest is in C++). Algorithms that have replanning but do not satisfy the real-time constraints in the replanning are denoted as Yes/No in the *Replan?* column of the table.

Method	Decentr.?	Replan?	Async.?	Without discretization?	Time (s)			Total Flight Distance (m)	Safety	
					Computation	Execution	Total		Safe?	Safety ratio
SCP, $h_{SCP} = 0.3$ s	No	No	No	No	1.150	7.215	8.366	77.144	No	0.142
SCP, $h_{SCP} = 0.25$ s					3.099	7.855	10.954	101.733	No	0.149
SCP, $h_{SCP} = 0.2$ s					8.741	7.855	16.596	90.917	No	0.335
SCP, $h_{SCP} = 0.17$ s					37.375	9.775	47.150	77.346	No	0.685
RBP, batch_size=1	No	No	No	Yes	0.228	15.623	15.851	90.830	Yes	1.055
RBP, batch_size=2					0.236	15.623	15.859	91.789	Yes	1.075
RBP, batch_size=4					0.277	14.203	14.480	92.133	Yes	1.023
RBP, no batches					0.461	14.203	14.664	93.721	Yes	1.057
DMPC*, $h_{DMPC} = 0.45$ s	Yes	No	No	No	4.952	23.450	28.402	79.650	No	0.683
DMPC*, $h_{DMPC} = 0.36$ s					4.350	20.710	25.060	97.220	No	0.914
DMPC*, $h_{DMPC} = 0.3$ s					4.627	18.430	23.057	78.580	Yes	1.015
DMPC*, $h_{DMPC} = 0.25$ s					3.796	15.980	19.776	79.590	No	0.824
dec_iSCP*, $h_{iSCP} = 0.4$ s	Yes	No	No	No	2.631	13.130	15.761	102.320	No	0.017
dec_iSCP*, $h_{iSCP} = 0.3$ s					4.276	15.470	19.746	97.770	No	0.550
dec_iSCP*, $h_{iSCP} = 0.2$ s					9.639	14.060	23.699	95.790	No	0.917
dec_iSCP*, $h_{iSCP} = 0.15$ s					14.138	14.060	28.198	102.030	No	0.906
decS_Search, $u = 2$ m/s ³	Yes	No	No	Yes	15.336	6.491	21.827	79.354	Yes	1.337
decS_Search, $u = 3$ m/s ³					7.772	6.993	14.764	80.419	Yes	1.768
decS_Search, $u = 4$ m/s ³					34.557	9.491	44.048	83.187	Yes	1.491
decS_Search, $u = 5$ m/s ³					3.104	8.491	11.595	80.804	Yes	1.474
decNS_Search, $u = 2$ m/s ³	Yes	Yes	No	Yes	0.021 0.233 33.711	34.288	80.116	Yes	1.416	
decNS_Search, $u = 3$ m/s ³		0.003 0.058 12.810			13.346	79.752	Yes	1.150		
decNS_Search, $u = 4$ m/s ³		0.030 0.600 53.824			53.899	79.752	Yes	1.502		
decNS_Search, $u = 5$ m/s ³		0.002 0.051 9.096			9.169	88.753	Yes	1.335		
MADER (ours)	Yes	Yes	Yes	Yes	0.550 2.468 7.975	10.262	82.487	Yes	1.577	

is also shorter than the one of RBP.

- Compared to decentralized algorithms (DMPC, dec_iSCP, decS_Search and decNS_Search), MADER is the one with the shortest total time, except for the case of decNS_Search with $u = 5$ m/s³. However, for this case the flight distance achieved by MADER is 6.3 m shorter. Moreover, MADER is asynchronous and satisfies the real-time constraints in the replanning, while decNS_Search does not.
- From all the algorithms shown in Table 3.4, MADER is the only algorithm that is decentralized, has replanning, satisfies the real-time constraints in the replanning and is asynchronous.

For MADER, and measured on the simulation environment used in this section, each UAV performs on average 12 successful replans before reaching the goal. On

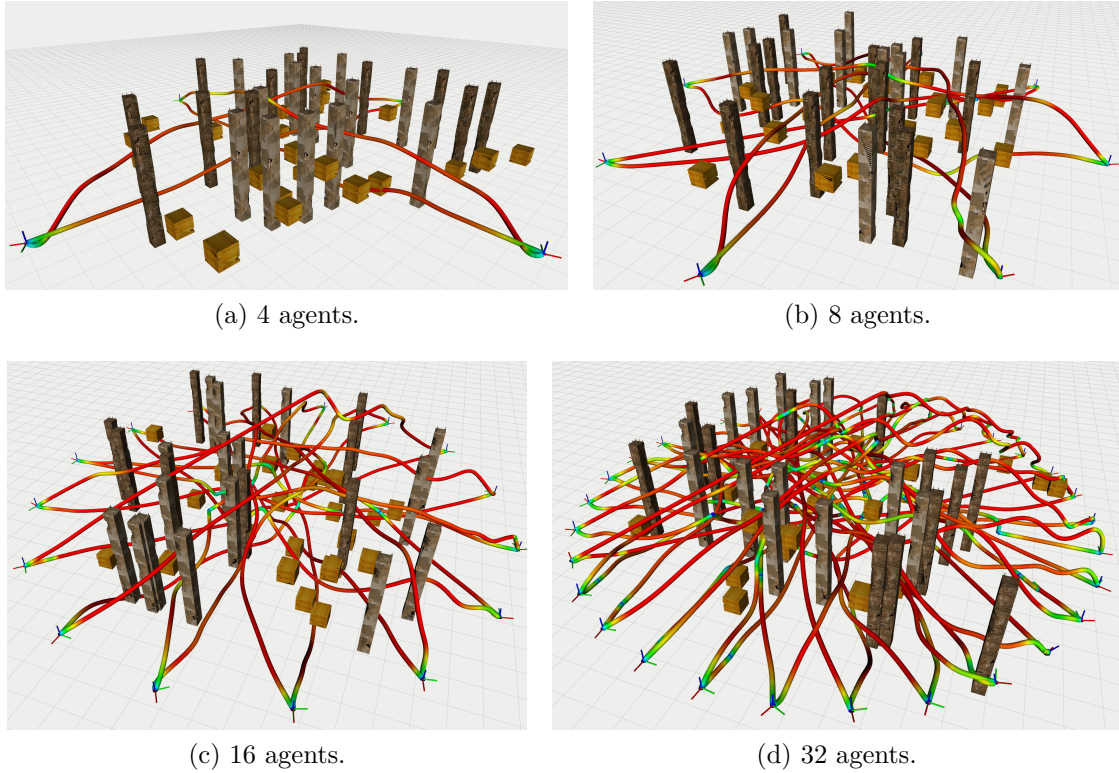


Figure 3-12: Results for the circle environment, that contains 25 static obstacles (pillars) and 25 dynamic obstacles (boxes).

average, the check step takes ≈ 2.87 ms, the recheck step takes ≈ 0.034 μ s, and the total replanning time is ≈ 199.6 ms. Approximately half of this replanning time is allocated to find the initial guess (i.e., $\kappa = 0.5$).

3.7.3 Multiagent Simulations with Static and Dynamic Obstacles

We now test MADER in multiagent environments that have also static and dynamic obstacles. For this set of experiments, we use $\alpha_j = \beta_j = 3 \cdot \mathbf{1}$ cm, $\gamma_j = 0.1$ s $\forall j$, $r = 4.5$ m (radius of the sphere \mathcal{S}), and a drone radius of 5 cm. We test MADER in the following two environments:

- **Circle environment:** the UAVs start in a circle formation and have to swap their positions while flying in a world with 25 static obstacles of size 0.4 m \times 8 m \times 0.4 m and 25 dynamic obstacles of size 0.6 m \times 0.6 m \times 0.6 m following

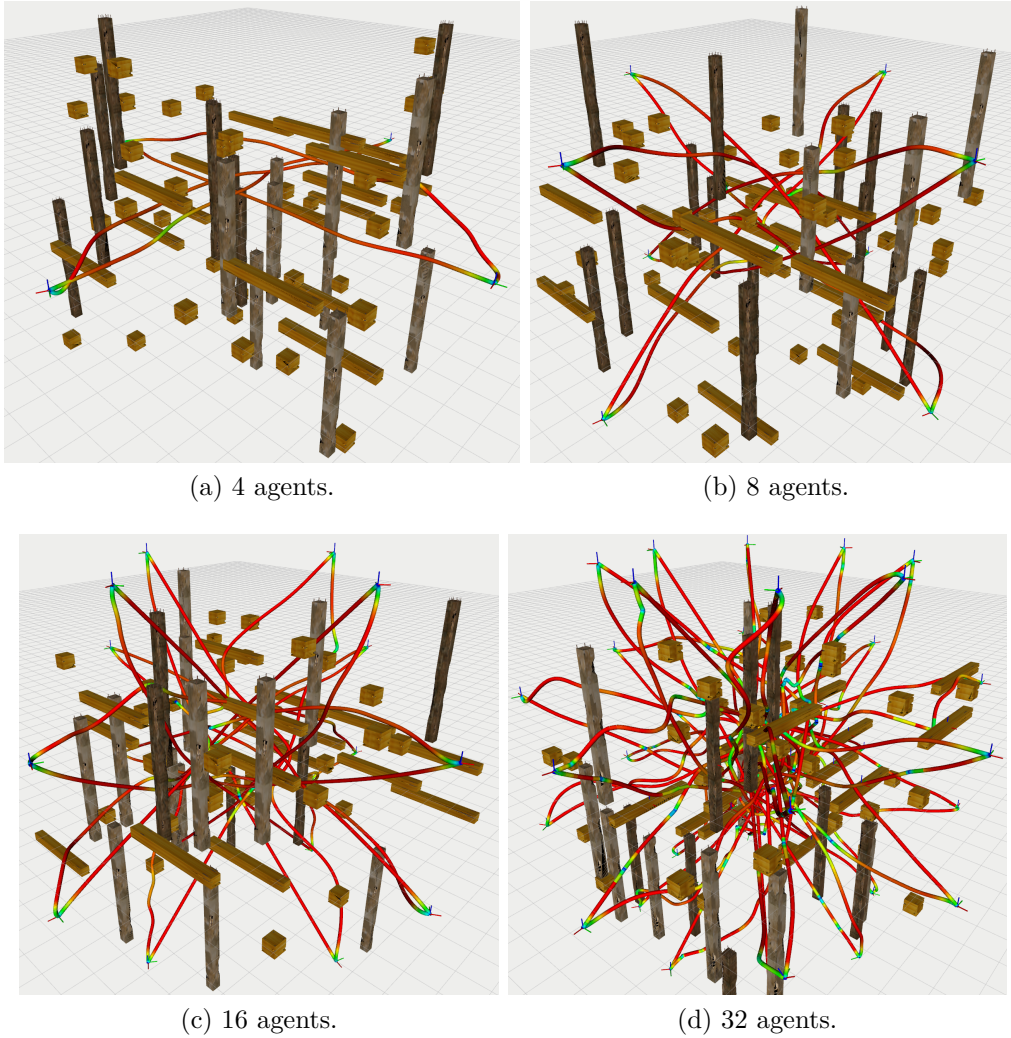


Figure 3-13: Results for the sphere environment, that contains 18 static obstacles (pillars) and 52 dynamic obstacles (boxes and horizontal poles).

a trefoil knot trajectory [110]. The radius of the circle the UAVs start from is 10 m.

- **Sphere environment:** the UAVs start in a sphere formation and have to swap their positions while flying in a world with 18 static obstacles of size $0.4 \text{ m} \times 8 \text{ m} \times 0.4 \text{ m}$, 17 dynamic obstacles of size $0.4 \text{ m} \times 4 \text{ m} \times 0.4 \text{ m}$ (moving in z) and 35 dynamic obstacles of size $0.6 \text{ m} \times 0.6 \text{ m} \times 0.6 \text{ m}$ following a trefoil knot trajectory. The radius of the sphere the UAVs start from is 10 m.

The results can be seen in Table 3.5 and in Figs. 3-12 and 3-13. All the safety

Table 3.5: Results for MADER in the circle and sphere environments.

Environment	Num of Agents	Time (s)				Flight Distance per agent (m)	Safety ratio between agents	Number of stops per agent
		$t_{1^{st}start}$	$t_{last\ start}$	$t_{1^{st}end}$	Total			
Circle	4	0.339	0.563	10.473	11.403	21.052	5.444	0.000
	8	0.404	0.559	7.544	11.108	21.025	1.959	0.000
	16	0.300	0.764	7.773	13.972	21.737	4.342	0.188
	32	0.532	1.195	9.092	18.820	22.105	1.834	1.500
Sphere	4	0.452	0.584	10.291	11.124	20.827	4.242	0.000
	8	0.425	0.618	9.204	12.561	21.684	1.903	0.125
	16	0.363	0.845	8.909	13.175	21.284	1.905	0.125
	32	0.357	1.725	9.170	18.275	22.284	1.155	1.000

ratios between the agents are > 1 , and the flight distances achieved (per agent) are approximately 21.5 m. With respect to the number of stops, none of the UAVs had to stop in the circle environment with 4 and 8 agents and in the sphere environment with 4 agents. For the circle environment with 16 and 32 agents, each UAV stops (on average) 0.188 and 1.5 times respectively. For the sphere environment with 8, 16, and 32 agents, each UAV stops (on average) 0.125, 0.125, and 1.0 times respectively. Note also that only the two agents that have been the closest are the ones that determine the actual value of the safety ratio, while the other agents do not contribute to this value. This means that, while the safety ratio is likely to decrease with the number of agents, a monotonic decrease of the safety ratio with respect to the number of agents is not strictly required.

Chapter 4

PANTHER: Perception-Aware Trajectory Planner in Dynamic Environments

4.1 Overview

This chapter presents PANTHER, a real-time perception-aware (PA) trajectory planner for multirotor-UAVs in dynamic environments. PANTHER plans trajectories that avoid dynamic obstacles while also keeping them in the sensor field of view (FOV) and minimizing the blur to aid in object tracking. The rotation and translation of the UAV are jointly optimized, which allows PANTHER to fully exploit the differential flatness of multirotors to maximize the PA objective. Real-time performance is achieved by implicitly imposing the underactuated dynamics of the UAV through the Hopf fibration. PANTHER is able to keep the obstacles inside the FOV 7.9 and 1.5 times more than non-PA approaches and PA approaches that decouple translation and yaw, respectively. The projected velocity (and hence the blur) is reduced by 18% and 34%, respectively. This leads to average success rates three times larger than state-of-the-art approaches in multi-obstacle avoidance scenarios. The MINVO basis is used to impose low-conservative collision avoidance constraints

Table 4.1: Notation used in this chapter.

Symbol	Meaning
$\text{abs}(\mathbf{a}), \mathbf{a} \leq \mathbf{b}$	Element-wise absolute value, element-wise inequality.
$\ \cdot\ $	Euclidean norm.
c_α, s_α	$\cos(\alpha), \sin(\alpha)$
\circ	Quaternion multiplication.
g	$g \approx 9.81 \text{ m/s}^2$
$\sigma(\cdot)$	Sigmoid function [111].
$\mathbf{e}_z, \mathbf{1}$	$\mathbf{e}_z := [0 \ 0 \ 1]^T, \mathbf{1} := [1 \ 1 \ 1]^T$
FOV, AABB	Field of view, axis-aligned bounding box.
$\text{SO}(n), \text{SE}(n)$	Special orthogonal group, special Euclidean group.
S^n	n -sphere.
$\text{wrap}_{-\pi}^{\pi}(\cdot)$	Wrapping of an angle in $[-\pi, \pi)$
$N(\cdot)$	Normal distribution.
$\text{norminv}(\cdot)$	Inverse of the standard normal cumulative distribution function [108].
$\text{diag}(\cdot)$	Diagonal matrix.
$\mathcal{S}_{p,m}^d$	Set of clamped uniform splines with dimension d , degree p , and $m + 1$ knots.
n ($n_{\mathbf{p}}$ and n_ψ)	$n := m - p - 1$ $n + 1$ is the number of control points of the spline.
$\mathbf{p}, \mathbf{v}, \mathbf{a}, \mathbf{j}$	Position, Velocity, Acceleration, and Jerk of the UAV, $\in \mathbb{R}^3$. All of them are of the body w.r.t. the world frame, and expressed in the world frame.
$\boldsymbol{\xi}$	Relative acceleration, expressed in the world frame: $\boldsymbol{\xi} := [\mathbf{a}_x \ \mathbf{a}_y \ \mathbf{a}_z + g]^T$. We will assume $\boldsymbol{\xi} \neq \mathbf{0}$.
$\psi, \dot{\psi}$	Angle (and its derivative) such that $\mathbf{q}_b^w = \mathbf{q}_\xi \circ [c_{\psi/2} \ 0 \ 0 \ s_{\psi/2}]^T$ (see Section 4.2.4.1).
\mathbf{x}	State vector: $\mathbf{x} := [\mathbf{p}^T \ \mathbf{v}^T \ \mathbf{a}^T \ \psi \ \dot{\psi}]^T \in \mathbb{R}^{11}$.
\mathbf{p}^a	Point expressed in the frame a . For the definitions of this table that include the sentence “expressed in the world frame”, the notation of the frame is omitted.
$\tilde{\mathbf{p}}, \bar{\mathbf{p}}$	$\tilde{\mathbf{p}} := [\mathbf{p}^T \ 1]^T, \bar{\mathbf{p}} := \frac{\mathbf{p}}{\ \mathbf{p}\ }$
$\mathbf{T}_b^a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0}^T & 1 \end{bmatrix}$	Transformation matrix: $\tilde{\mathbf{p}}^a = \mathbf{T}_b^a \tilde{\mathbf{p}}^b$. Analogous definition for the quaternion \mathbf{q}_b^a .
$\text{rot}(\mathbf{q})$	Rotation matrix associated with the quaternion \mathbf{q} .
J	Set of indexes of all the intervals $J = \{0, 1, \dots, m - 2p - 1\}$.
j	Index of the interval of the trajectory, $j \in J$.
I	Set of indexes of the tracked obstacles.
i	Index of the obstacle, $i \in I$.
i^*	Index of the obstacle used in the PA term of the cost function.
$(\mathbf{p}_i)^a(t)$	Mean of the predicted position of obstacle i , expressed in frame a
$(\mathbf{p}_i)^w(t), \boldsymbol{\sigma}_i(t)$	The predicted trajectory of the obstacle i , in the world frame, is $\sim N((\mathbf{p}_i)^w(t), (\text{diag}(\boldsymbol{\sigma}_i(t)))^2)$.
f	Focal length of the camera in meters.
θ	Opening angle of the cone that approximates the FOV.
$[q_w \ q_x \ q_y \ q_z]^T$	Components of a unit quaternion.
$\mathbf{1}_c$	1 if c is true, 0 otherwise.
$\text{inFOV}(\mathbf{T}_c^w, (\mathbf{p}_i)^w)$	$\mathbf{1}_{(\mathbf{p}_i)^w \in \text{FOV}} \approx \mathbf{1}_{((\mathbf{p}_i)^c)_z / \ (\mathbf{p}_i)^c\ \geq c_{\theta/2}} \approx \sigma(\gamma(-c_{\theta/2} + ((\mathbf{p}_i)^c)_z / \ (\mathbf{p}_i)^c\))$. γ is a positive parameter.
$L_{\mathbf{p}}, L_\psi$	$L_{\mathbf{p}} := \{0, 1, \dots, n_{\mathbf{p}}\}, L_\psi := \{0, 1, \dots, n_\psi\}$.
l	Index of the control point. $l \in L_{\mathbf{p}}$ for $\mathbf{p}(t)$, $l \in L_{\mathbf{p}} \setminus \{n_{\mathbf{p}}\}$ for $\mathbf{v}(t)$, $l \in L_{\mathbf{p}} \setminus \{n_{\mathbf{p}} - 1, n_{\mathbf{p}}\}$ for $\mathbf{a}(t)$, $l \in L_\psi$ for ψ and $l \in L_\psi \setminus \{n_\psi\}$ for $\dot{\psi}$
$\mathbf{q}_l, \mathbf{v}_l, \mathbf{a}_l, \psi_l, \dot{\psi}_l$	Position, velocity, and acceleration control points, ($\in \mathbb{R}^3$), ψ and $\dot{\psi}$ control points ($\in \mathbb{R}$).
$\mathcal{Q}_j^{\text{MV}}$	Set of position control points of the interval j using the MINVO basis. Analogous definition for the velocity control points $\mathcal{V}_j^{\text{MV}}$.
δ	$\in [0, 1]$, percentile of the standard normal distribution (see next row).
$\mathcal{C}_{ij}^{\text{MV}}$	Set of vertexes of the convex hull of the set obtained by inflating $(\mathcal{Q}_j^{\text{MV}})_{\text{obs } i}$ with $\text{norminv}(\delta) \cdot \boldsymbol{\sigma}_i(t_{\text{end } j})$, half of the sides of the AABB of the obstacle i and half of the sides of the AABB of the agent.
$\boldsymbol{\pi}_{ij}(\mathbf{n}_{ij}, d_{ij})$	Plane $\mathbf{n}_{ij}^T \mathbf{x} + d_{ij} = 0$ that separates $(\mathcal{Q}_j^{\text{MV}})_{\text{agent}}$ from $\mathcal{C}_{ij}^{\text{MV}}$.
$\mathbf{h}(\cdot), \mathbf{st}(\cdot)$	Hopf fibration, stereographic projection.
Snapshot at $t = t_1$ (current time):	
\mathbf{g}_{term} (●) is the terminal goal, and ● is the current position of the UAV. — is the trajectory the UAV is currently executing. ---- is the trajectory the UAV is currently optimizing, $t \in [t_{\text{in}}, t_f]$ \mathbf{d} (○) is a point in —, used as the initial position of ---- \mathcal{M} is a sphere of radius r around \mathbf{d} . \mathbf{g} (●) is the projection of \mathbf{g}_{term} (●) onto the sphere \mathcal{M} . \mathbf{d}, \mathbf{g} , and \mathbf{g}_{term} are expressed in the world frame.	

in position and velocity space. Finally, extensive hardware experiments in unknown dynamic environments with all the computation running onboard are presented, with velocities of up to 5.8 m/s, and with relative velocities (with respect to the obstacles) of up to 6.3 m/s. The only sensors used are an IMU, a forward-facing depth camera, and a downward-facing monocular camera.

This chapter uses the notation shown in Table 4.1.

4.2 PANTHER

PANTHER comprises four modules: Tracker and predictor, selector of the obstacle in the PA term, planes and initial guess generator, and optimization (see Fig. 4-1a). A summary of how all these modules work together is as follows: First the incoming point clouds of the onboard depth sensor are clustered and tracked using the Hungarian algorithm [78] to obtain the trajectory, as a probability distribution, of each of the obstacles (Section 4.2.1). The obstacle i^* that the UAV is most likely to collide with is then selected to be included in the PA term of the cost function (Section 4.2.2). Then, a kinodynamic search-based planner (Octopus Search Algorithm [160]) is run to find a initial guess of the translational trajectory $\mathbf{p}(t)$ that avoids the probabilistic trajectories of the obstacles found before (Section 4.2.3.1). This translational guess and the obstacle i^* selected are then used to run a graph search algorithm to find the $\psi(t)$ guess (Section 4.2.3.2). Finally, the $\mathbf{p}(t)$ and $\psi(t)$ guesses are used for the nonconvex optimization to obtain the optimized trajectory, that is sent to the controller of the UAV (Section 4.2.4). In this framework, the coupling between rotation and acceleration is imposed implicitly using the Hopf fibration. All these modules are described in detail in the following subsections.

4.2.1 Tracking and Prediction

We create a k-d tree representation of the point clouds coming from the onboard depth sensor, and perform Euclidean clustering to group the points that are more likely to belong to the same obstacle (see Fig. 4-1a). For each cluster found, we compute the

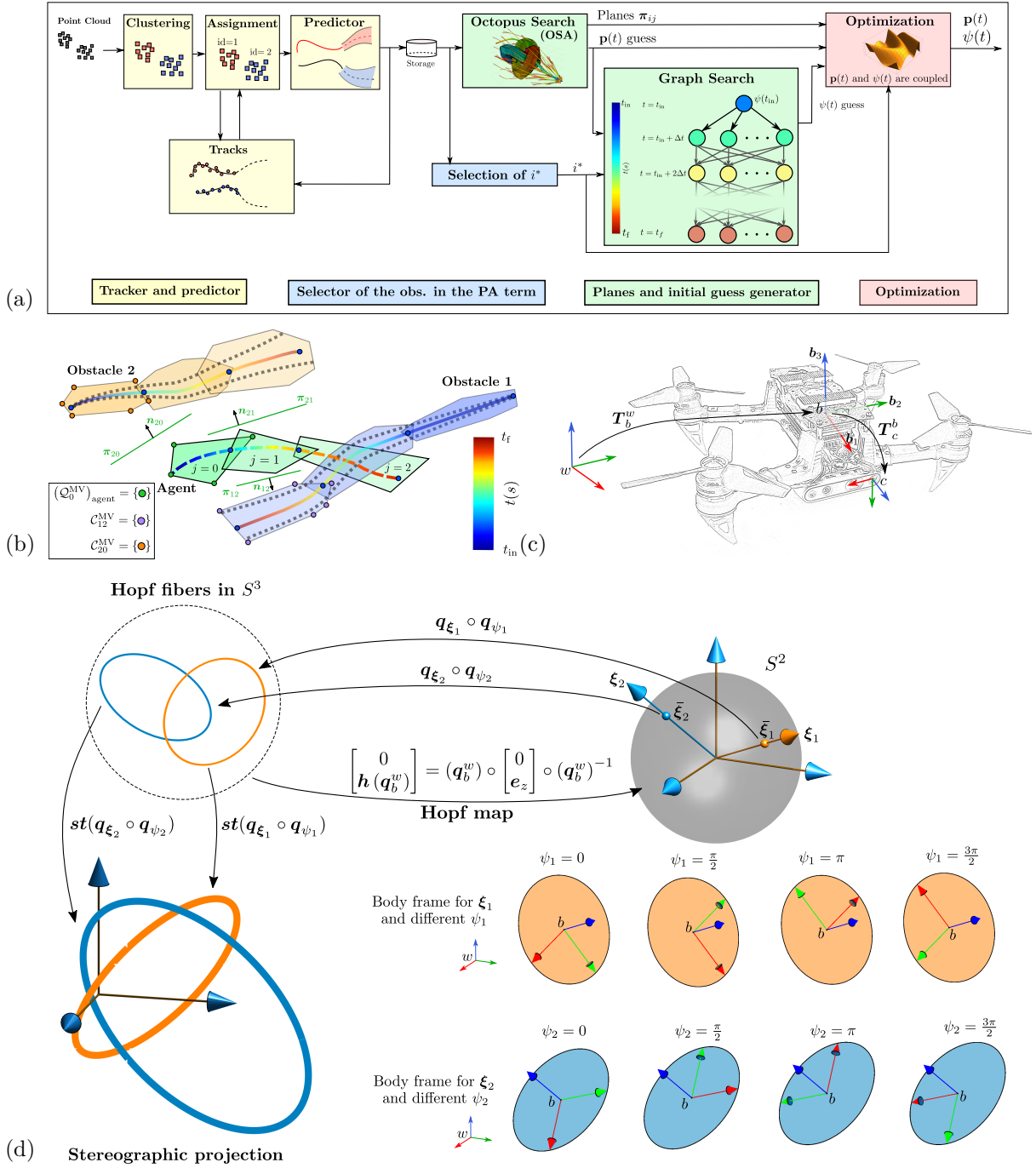


Figure 4-1: (a) Different modules of PANTHER. (b) Predicted trajectories of the obstacles and convex representation of each segment of the trajectory of the agent and the obstacles. (c) World, body, and camera frames. (d) Hopf fibration and its stereographic projection, partly inspired from [101]. Given a specific relative acceleration ξ (with $\xi \neq -e_z$), the quaternion $q_b^w = q_\xi \circ q_\psi$ is a fiber (specifically a circle) in S^3 parameterized by ψ . On the bottom right, the body frames for different values of ψ_i for each ξ_i are shown.

AABB (axis-aligned bounding box) centered on the centroid of that cluster.¹ Then, to assign each cluster to a specific track, we minimize the total assignment cost using the Hungarian algorithm [78], where the cost is the pairwise distance between the centroid of each cluster and the prediction of the tracks at the time the point cloud was produced. If this distance is above a specific threshold (usually $\approx 1\text{--}2$ m), we create a new track for it. If a cluster is not assigned to any track (which can happen if there are more clusters than tracks), then a new track is created for it. Finally, given a sliding window history of all the observations associated with a track, we fit a polynomial for each coordinate $\{x, y, z\}$. To capture the stochasticity of the prediction problem, the predicted position at time t is then approximated by a 3D Gaussian distribution (mean from the value of the fitted polynomial and a diagonal covariance matrix obtained from the prediction intervals [64, Section 5.7]).

4.2.2 Selection of the Obstacle in the PA Term

When there are several predicted trajectories, and to maintain computational tractability, the agent needs to choose which one of them to include in the PA term of the cost function. It does so by choosing the most likely obstacle to collide with in the future, using a simple heuristic of the probability of collision based on Boole’s inequality [66]:

$$i^* = \operatorname{argmax}_{i \in I} \sum_{u=0}^{U-1} P(\|(\mathbf{p}_i)^w(t_u) - \boldsymbol{\kappa}(u)\|_\infty \leq R)$$

where U is the number of samples taken, $t_u := t_{\text{in}} + \frac{u}{U}(t_f - t_{\text{in}})$ and $\boldsymbol{\kappa}(u) := \mathbf{d} + \frac{u}{U}(\mathbf{g}_{\text{term}} - \mathbf{d})$ is a point in a straight line from \mathbf{d} to \mathbf{g}_{term} . Note that although only one obstacle is included in the PA objective function, all the predictions of the tracked obstacles are included in the collision avoidance constraints.

Additionally, and to address the trade-off between gathering information about the obstacle, and gathering information about the direction of travel, the UAV will include

¹Regardless of whether or not the obstacle is convex, this produces an outer convex approximation of the visible part of the obstacle.

the obstacle i^* in the PA term if the angle between $(\mathbf{g}_{\text{term}} - \mathbf{d})$ and $((\mathbf{p}_{i^*})^w(t_{\text{in}}) - \mathbf{d})$ is smaller than a predefined angle α_0 (typically $\approx 90^\circ$). Otherwise the UAV will try to align the FOV of the camera with the direction of travel.

4.2.3 Planes and Initial Guesses

4.2.3.1 Separability Planes and Initial Guess for Position

We use the Octopus Search Algorithm (OSA) [160], which is a search-based algorithm that operates directly on the control points of the position spline. It ensures collision-free constraints between the agent and the dynamic obstacles by finding the planes that separate the inflated MINVO polyhedral representation of each interval j of the trajectory of the obstacle i (denoted as $\mathcal{C}_{ij}^{\text{MV}}$) and the MINVO polyhedral representation of that interval j of the trajectory of the agent, denoted as $(\mathcal{Q}_j^{\text{MV}})_{\text{agent}}$ (see Fig. 4-1b). The outputs of this algorithm are both the position control points and the planes $\boldsymbol{\pi}_{ij}$ (given by $\mathbf{n}_{ij}^T \mathbf{x} + d_{ij} = 0$) $\forall i, \forall j$. The position control points are then used as initial guess in the optimization, while the planes $\boldsymbol{\pi}_{ij}$ are held fixed in the optimization. The reader is referred to our previous work [160] for a more in-depth explanation of the OSA.

4.2.3.2 Initial Guess for ψ

To obtain the initial guess for ψ , we uniformly sample the position guess spline obtained through the OSA, and for each of these position samples, we uniformly sample several values of $\psi \in [-\pi, \pi)$. Each one of these \mathbf{p} - ψ samples will be a *node*, and all the nodes associated with the same position sample, but with different ψ , will constitute a *layer* (see Fig. 4-1a). Then, we create a graph connecting with directed edges all the nodes of one layer to the nodes of the next layer [183]. Each node has therefore a time, position, acceleration, and yaw associated with it, and all the nodes of the same layer have the same time, position, and acceleration. The cost of the edge

between two nodes n_1 and n_2 of the graph is then given by

$$c_\psi \left(\text{wrap}_{-\pi}^\pi(\psi_{n_2} - \psi_{n_1}) \right)^2 + c_{\Psi_{\max}} \cdot \mathbf{1}_{\left| \frac{\text{wrap}_{-\pi}^\pi(\psi_{n_2} - \psi_{n_1})}{t_{n_2} - t_{n_1}} \right| > \Psi_{\max}} + c_{\text{FOV}} (1 - \text{inFOV}((\mathbf{T}_{n_2}(t_{n_2}))_c^w, (\mathbf{p}_{i^*}(t_{n_2}))^w))$$




Here, c_ψ , $c_{\Psi_{\max}}$, and c_{FOV} are nonnegative weights, while ψ_{n_u} , t_{n_u} , and $(\mathbf{T}_{n_u}(t_{n_u}))_c^w$ are the angle ψ , the time, and the transformation matrix associated with node n_u . Note that the edge cost is guaranteed to be nonnegative at all times. The transformation matrix can be directly obtained from the position, acceleration, and yaw of the node. The first term in the cost penalizes the distance between two ψ angles, the second term penalizes edges that do not satisfy the limit Ψ_{\max} , and the last one rewards the visibility of the obstacle. The units of the weights above are such that the corresponding term is dimensionless (see Section 4.3). To choose these weight values, we first set $c_{\Psi_{\max}}$ to a large value to guarantee the Ψ_{\max} constraint. Then, c_ψ and c_{FOV} are selected as a trade-off between smoothness and inclusion of the obstacle i^* in the FOV of the UAV. The root node of the graph corresponds to the state \mathbf{d} (see last row of Table 4.1). We solve the search problem using Dijkstra’s algorithm [37], with early termination when the search reaches a node of the last layer. Letting Λ denote the indexes of the nodes of the path found, we shift the angles $\psi_{n_\lambda} \forall \lambda \in \Lambda$ (by adding or subtracting $2\pi r$, $r \in \mathbb{Z}$) such that the absolute difference between two consecutive angles is $\leq \pi$. Using $\hat{\psi}_{n_\lambda}$ to denote these shifted angles, a spline is fitted to these angles by solving the following constrained least square problem:

$$\begin{aligned} \min_{\psi(t) \in \mathcal{S}_{2,m}^1} \sum_{\lambda \in \Lambda} \left\| \psi(t_{n_\lambda}) - \hat{\psi}_{n_\lambda} \right\|_2^2 \\ \text{s.t. } \psi(t_{\text{in}}) = \psi_{\text{in}}, \quad \dot{\psi}(t_{\text{in}}) = \dot{\psi}_{\text{in}}, \quad \dot{\psi}(t_{\text{f}}) = 0 \end{aligned} \tag{4.1}$$

Note that, as this problem is a quadratic program with linear equality constraints, its solution can be easily found by simply solving the linear Karush-Kuhn-Tucker (KKT) conditions associated with it [72, 79].² The control points of this fitted spline are then used as the initial guess for $\psi(t)$ in the optimization.

²For a detailed explanation of the derivation of the resulting linear system of equations, see, e.g., [19, Example 5.1].

Table 4.2: Some commonly-used definitions for the differential flatness map ($\mathbf{a} \in \mathbb{R}^3 \setminus [0 \ 0 \ -g]^T, \psi \in S^1 \rightarrow \mathbf{R}_b^w \in \text{SO}(3)$). The colormap represents the great-circle distance to the closest singularity (yellow is closer), $(\cdot)_n$ denotes the normalization of a vector, and $\bar{\boldsymbol{\xi}} := ([\mathbf{a}_x \ \mathbf{a}_y \ \mathbf{a}_z + g]^T)_n$ is the normalized relative acceleration, expressed in the world frame. See also [8, 118, 149, 150, 176] for more possible definitions, which are usually rotations of the first two definitions of this table.

	Definition 1	Definition 2	Definition 3 (Hopf fibration)
Map	$\mathbf{b}_1 = \mathbf{b}_2 \times \mathbf{b}_3$ $\mathbf{b}_2 = (\mathbf{b}_3 \times [c_\psi \ s_\psi \ 0]^T)_n$ $\mathbf{b}_3 = \bar{\boldsymbol{\xi}}$ $\mathbf{R}_b^w = [\mathbf{b}_1 \ \mathbf{b}_2 \ \mathbf{b}_3]$	$\mathbf{b}_1 = ([-s_\psi \ c_\psi \ 0]^T \times \mathbf{b}_3)_n$ $\mathbf{b}_2 = \mathbf{b}_3 \times \mathbf{b}_1$ $\mathbf{b}_3 = \bar{\boldsymbol{\xi}}$ $\mathbf{R}_b^w = [\mathbf{b}_1 \ \mathbf{b}_2 \ \mathbf{b}_3]$	$\mathbf{q}_b^w = \underbrace{\frac{1}{\sqrt{2(1+\bar{\xi}_z)}}}_{:=\mathbf{q}_\xi} \begin{bmatrix} 1+\bar{\xi}_z \\ -\bar{\xi}_y \\ \bar{\xi}_x \\ 0 \end{bmatrix} \circ \underbrace{\begin{bmatrix} c_{\psi/2} \\ 0 \\ 0 \\ -s_{\psi/2} \end{bmatrix}}_{:=\mathbf{q}_\psi}$ $\mathbf{R}_b^w = \text{rot}(\mathbf{q}_b^w)$
Singularity	$\bar{\boldsymbol{\xi}} \parallel [c_\psi \ s_\psi \ 0]^T$. When $\psi = 0$: 	$\bar{\boldsymbol{\xi}} \parallel [-s_\psi \ c_\psi \ 0]^T$. When $\psi = 0$: 	$\bar{\boldsymbol{\xi}} = [0 \ 0 \ -1]^T$ 
Notes	<ul style="list-style-type: none"> Singularity=$f(\mathbf{a}, \psi)$ For a given ψ, singularity appears for two $\bar{\boldsymbol{\xi}}$ UAV is differentially flat, with flat outputs $\{\mathbf{p}, \psi\}$ [115] 	<ul style="list-style-type: none"> Singularity=$f(\mathbf{a}, \psi)$ For a given ψ, singularity appears for two $\bar{\boldsymbol{\xi}}$ UAV is differentially flat, with flat outputs $\{\mathbf{p}, \psi\}$ [41] 	<ul style="list-style-type: none"> Singularity=$f(\mathbf{a})$ Singularity appears for one $\bar{\boldsymbol{\xi}}$ UAV is differentially flat, with flat outputs $\{\mathbf{p}, \psi\}$ [174]

4.2.4 Optimization

4.2.4.1 Coupling Rotation and Acceleration with the Hopf Fibration

In a standard multirotor-UAV, the perpendicularity of the total thrust with respect to the plane spanned by \mathbf{b}_1 and \mathbf{b}_2 (see the coordinate frames shown in Fig. 4-1c) makes the UAV underactuated by imposing the following constraint [174]:

$$\text{rot}(\mathbf{q}_b^w) \mathbf{e}_z = \bar{\boldsymbol{\xi}} \quad (4.2)$$

where $\bar{\boldsymbol{\xi}}$ is the normalized relative acceleration expressed in the world frame (see Table 4.1). In a planning optimization problem where rotation and translation are jointly optimized, Eq. 4.2 needs to be satisfied at all times. A very common way to guarantee Eq. 4.2 is via direct imposition of the dynamic equations of the UAV as explicit constraints. However, these differential equations in the optimization problem typically lead to computationally-expensive problems, due to the fine sampling needed

in the discretization methods (shooting or collocation).

The direct imposition of the dynamic equations can be avoided by leveraging the differential flatness map $(\mathbf{a} \in \mathbb{R}^3 \setminus [0 \ 0 \ -g]^T, \psi \in S^1) \rightarrow \mathbf{R}_b^w \in \text{SO}(3)$, which takes the acceleration \mathbf{a} and ψ and maps them to the rotation of the body frame. Due to the *hedgehog theorem*³ in S^2 [12, 20], this map is guaranteed to have at least one singularity when tried to be defined with a single continuous function. Several possible definitions of this differential flatness map are shown in Table 4.2, all of which satisfy Eq. 4.2 by construction. In the first two definitions, one body axis is obtained as the cross product of $\mathbf{b}_3 \equiv \bar{\boldsymbol{\xi}}$ with a vector lying in the xy world plane, and the remaining body axis is such that the resulting body frame is right-handed. These two definitions present a singularity whenever the normalized relative acceleration $\bar{\boldsymbol{\xi}} \in S^2$ is parallel to a vector defined by ψ which lies in the xy world plane. This means that, for a given ψ , the singularity appears for two $\bar{\boldsymbol{\xi}}$ that have a great-circle distance of 90° with respect to the hovering condition. In aggressive flights, and due to numerical instabilities and artificial large changes of orientations near the singularity, this closeness between the hovering condition and the singularity can limit the set of possible accelerations for the planner.

The third definition of Table 4.2 leverages the Hopf fibration $\mathbf{h}(\cdot)$, which can be defined as a map $S^3 \rightarrow S^2$ [101, 174] that takes a unit quaternion \mathbf{q} and produces the resulting rotation of the vector $\mathbf{e}_z := [0 \ 0 \ 1]^T$ (see Fig. 4-1d):

$$\begin{bmatrix} 0 \\ \mathbf{h}(\mathbf{q}) \end{bmatrix} := \mathbf{q} \circ \begin{bmatrix} 0 \\ \mathbf{e}_z \end{bmatrix} \circ \mathbf{q}^{-1}$$

Making use now of the inverse image of the Hopf fibration, we have that \mathbf{q}_b^w will be a composition of two rotations:⁴ \mathbf{q}_ξ , that aligns \mathbf{b}_3 with $\boldsymbol{\xi}$, followed by \mathbf{q}_ψ , which is a rotation around $\boldsymbol{\xi}$ by an angle ψ . Given a specific $\boldsymbol{\xi}$ (with $\bar{\boldsymbol{\xi}} \neq -\mathbf{e}_z$), the quaternion $\mathbf{q}_b^w = \mathbf{q}_\xi \circ \mathbf{q}_\psi$ will then be a fiber (specifically a circle) in S^3 parametrized by ψ [101]. The main advantage of the Hopf fibration over the previous two definitions is that

³Also known as the *hairy ball theorem* in the literature.

⁴Note that \mathbf{q}_ψ , \mathbf{q}_ξ , and \mathbf{q}_b^w are guaranteed to be unit quaternions by construction.

the singularity only occurs when the UAV is inverted (i.e., when $\bar{\boldsymbol{\xi}} = -\mathbf{e}_z$), which is the orientation that has the largest possible great-circle distance from the hovering configuration, and hence much less likely to happen. Although the goal of this work is not to plan highly aggressive trajectories, and hence any of the singularities shown in Table 4.2 are unlikely to be reached, we use the Hopf map to automatically ensure the maximum distance to the singularity. Note also that, with the Hopf fibration, a second chart could be used to cover the inversion point $\bar{\boldsymbol{\xi}} = -\mathbf{e}_z$, but the use of multiple charts, while computationally cheap in the controller level [174] or in an intermediate check step in a decoupled \mathbf{p} - ψ optimization [176], would significantly increase the computation time when embedded in the \mathbf{p} - ψ joint planning optimization. This fact, together with the improbability of an upside-down configuration as being PA optimal, led us to the inclusion of only the first chart.

Our work differs from other works that have used the Hopf fibration for UAVs [174,176,177] as follows: Ref. [174] uses the Hopf fibration only in a controller to track predefined trajectories. In [176], the Hopf fibration is used in the planner to find the charts in a step after the \mathbf{p} optimization and before the ψ optimization, and [177] does not optimize ψ . We instead propose to embed the Hopf fibration map directly on the \mathbf{p} - ψ *joint* optimization, as a way to directly obtain trajectories in SE(3) that are dynamically feasible by construction, and with the crucial advantage of not needing to explicitly impose the dynamic equations as constraints in the optimization.

4.2.4.2 Cost Function

A PA term in the objective function should maximize the presence in the FOV of the predicted position of the obstacle $i^* \in I$. However, this alone is not enough to guarantee good PA trajectories, since a fast moving projected obstacle in the image plane may cause significant blur, which can lead to stereo matching failure and consequently tracking failure. To take into account both the presence in the FOV and the blur, we use $\frac{\text{inFOV}(\cdot)}{\epsilon_1 + \epsilon_2 \|\dot{\mathbf{s}}\|^2}$ as the running reward, where $\dot{\mathbf{s}}$ is the projected velocity in the image plane, and where ϵ_1 and ϵ_2 are nonnegative parameters such that $\epsilon_1 + \epsilon_2 > 0$. Note how this reward is high if the predicted position of the obstacle

is in the FOV with a small projected velocity, and it is approximately zero if the predicted position of the obstacle is not in the FOV, regardless of the value of the projected velocity. The position in the image plane of the projection of the obstacle can be obtained using the pinhole camera model as $\mathbf{s}(t) := \frac{f}{[(\tilde{\mathbf{p}}_{i^*}(t))^c]_z} [(\tilde{\mathbf{p}}_{i^*}(t))^c]_{x:y}$ (where each component of \mathbf{s} is expressed in meters, not in pixels), and

$$(\tilde{\mathbf{p}}_{i^*}(t))^c := \mathbf{T}_b^c \mathbf{T}_w^b(t) (\tilde{\mathbf{p}}_{i^*}(t))^w$$

$$\mathbf{T}_w^b(t) := \begin{bmatrix} \text{rot}(\mathbf{q}_\xi(t) \circ \mathbf{q}_\psi(t)) & \mathbf{p}(t) \\ \mathbf{0}^T & 1 \end{bmatrix}^{-1}$$

As detailed in Table 4.1, the discontinuity of the function $\text{inFOV}(\cdot)$ is addressed by approximating it with a sigmoid function.

In addition to the PA term explained above, we also add two terms in the cost function to maximize the smoothness in position (by minimizing jerk) and ψ (by minimizing $\ddot{\psi}$), and a terminal cost that penalizes the distance between $\mathbf{p}(t_f)$ and \mathbf{g} .

4.2.4.3 Collision Avoidance and Dynamic Limits Constraints

For the obstacle avoidance of dynamic obstacles, we first create a polyhedral outer representation of both the trajectory of the agent and of the obstacle (see Fig. 4-1b): For the agent, we make use of the MINVO basis [162] (a polynomial basis that finds the simplex with minimum volume enclosing a polynomial curve) to obtain the set of control points $(\mathcal{Q}_j^{\text{MV}})_{\text{agent}}$ whose convex hull encloses each segment j of the agent. Similarly, for each obstacle i , we first compute the MINVO control points of the segment j of the predicted mean $(\mathbf{p}_i)^w(t)$, and then we inflate it with $\text{norminv}(\delta) \cdot \boldsymbol{\sigma}_i(t_{\text{end } j})$, half of the sides the AABB (axis-aligned bounding box) of the obstacle i and half of the sides of the AABB of the agent. Here, $\delta \in [0, 1]$ is the percentile of the standard normal distribution, and hence it encodes the desired level of conservativeness in the inflation. The resulting polyhedron is denoted as $\mathcal{C}_{ij}^{\text{MV}}$.

To ensure safety between the agent and the obstacle i , we then impose linear separability constraints (via planes) between $(\mathcal{Q}_j^{\text{MV}})_{\text{agent}}$ and $\mathcal{C}_{ij}^{\text{MV}}$. The separating

planes are found during the initial guess search for the position spline (see Section 4.2.3.1), and are held fixed in the optimization. The MINVO basis is used in a similar way to impose low-conservative constraints in the velocity space. In the acceleration and jerk spaces, the MINVO control points are the same as the B-Spline control points. These constraints on \mathbf{v} , \mathbf{a} , \mathbf{j} , and $\dot{\psi}$ serve as a conservative approximation of the real actuator constraints of the motors of the UAV, while allowing us to reduce the complexity of the optimization problem.

4.2.4.4 Optimization Problem

Including the initial state and the final hovering condition, the optimization problem is:⁵

$$\begin{aligned}
& \min_{\mathbf{p}(t) \in \mathcal{S}_{3,m}^3, \psi(t) \in \mathcal{S}_{2,m}^1} \alpha_{\mathbf{j}} \int_{t_{\text{in}}}^{t_f} \|\mathbf{j}\|^2 dt + \alpha_{\psi} \int_{t_{\text{in}}}^{t_f} (\ddot{\psi})^2 dt \\
& - \alpha_{\text{FOV}} \int_{t_{\text{in}}}^{t_f} \frac{\text{inFOV}(\mathbf{T}_c^w, (\mathbf{p}_{i^*})^w)}{\epsilon_1 + \epsilon_2 \|\dot{\mathbf{s}}\|^2} dt + \alpha_{\mathbf{g}} \|\mathbf{p}(t_f) - \mathbf{g}\|^2 \\
& \text{s.t.} \\
& \mathbf{x}(t_{\text{in}}) = \mathbf{x}_{\text{in}}, \quad \mathbf{v}(t_f) = \mathbf{0}, \quad \mathbf{a}(t_f) = \mathbf{0}, \quad \dot{\psi}(t_f) = 0 \\
& \mathbf{n}_{ij}^T \mathbf{q} + d_{ij} < 0 \quad \forall \mathbf{q} \in (\mathcal{Q}_j^{\text{MV}})_{\text{agent}}, \quad \forall i \in I, \quad \forall j \in J \\
& \text{abs}(\mathbf{v}) \leq \mathbf{v}_{\text{max}} \quad \forall \mathbf{v} \in (\mathcal{V}_j^{\text{MV}})_{\text{agent}}, \quad \forall j \in J \\
& \text{abs}(\mathbf{a}_l) \leq \mathbf{a}_{\text{max}} \quad \forall l \in L_{\mathbf{p}} \setminus \{n_{\mathbf{p}} - 1, n_{\mathbf{p}}\} \\
& \text{abs}(\mathbf{j}_l) \leq \mathbf{j}_{\text{max}} \quad \forall l \in L_{\mathbf{p}} \setminus \{n_{\mathbf{p}} - 2, n_{\mathbf{p}} - 1, n_{\mathbf{p}}\} \\
& \text{abs}(\Psi_l) \leq \Psi_{\text{max}} \quad \forall l \in L_{\psi} \setminus \{n_{\psi}\}
\end{aligned}$$

Here, $\mathbf{x} := [\mathbf{p}^T \mathbf{v}^T \mathbf{a}^T \psi \dot{\psi}]^T$, $\{\alpha_{\mathbf{j}}, \alpha_{\psi}, \alpha_{\text{FOV}}, \alpha_{\mathbf{g}}\}$ are nonnegative weights, and the decision variables are the control points of the splines $\mathbf{p}(t)$ and $\psi(t)$. The degrees chosen for the splines $\mathbf{p}(t)$ and $\psi(t)$ are, respectively, 3 and 2, which are a good trade-off between computation time and dynamic feasibility for a UAV [115]. The units of the weights are such that the corresponding term is dimensionless (see Section 4.3). An empirical method to select these weight values is as follows: First set $\alpha_{\mathbf{g}}$ to a large

⁵Time dependence of the variables in the cost function has been omitted for simplicity.

value to ensure that the final location is near \mathbf{g} . Then, α_{FOV} , together with ϵ_1 and ϵ_2 , are tuned to obtain a good presence of the obstacle i^* in the FOV. Finally, α_j and α_ψ are progressively increased to improve the smoothness of $\mathbf{p}(t)$ and $\psi(t)$, without significantly deteriorating the FOV cost.

To solve this nonconvex optimization problem, we utilize the Interior Point Optimizer Ipopt [172]⁶ interfaced through CasADi [9] with MA27 and MA57 [63] as the linear solvers of Ipopt. All these optimization tools were installed and run onboard the UAV in the real-world experiments (Section 4.3.2). We approximate the PA term of the cost function using the composite Simpson’s rule for numerical integration [112].

4.3 Results and Discussion

4.3.1 Simulation Experiments

All the simulation experiments are run in an AlienWare Aurora r8 desktop running Ubuntu 20.04 and equipped with an Intel[®] Core[™] i9-9900K CPU, 3.60GHz×16 and 62.6 GiB. Moreover, and to focus the comparisons on the properties of the trajectories obtained by the planner, we assume, for all the algorithms benchmarked in simulation, that the UAV can perfectly track the trajectories obtained by the planner.

4.3.1.1 Single Obstacle

We first test PANTHER in an environment with a box-shaped obstacle of size $0.2 \times 0.2 \times 0.2 \text{ m}^3$ that follows a trefoil-knot [110] trajectory. During 60 s, the UAV is commanded to continuously fly between two different locations whose centroid is the area where the obstacle is moving. The camera has an image size of $120 \times 120 \text{ px}^2$, a limited FOV of $60^\circ \times 60^\circ$, and runs at a rate of 60 Hz. The weights used for this simulation are $c_{\Psi_{\text{max}}} = 10^6$, $c_{\text{FOV}} = 1$, $c_\psi = 0 \text{ rad}^{-2}$, $\alpha_j = 10^{-6} \text{ s}^5/\text{m}^2$, $\alpha_\psi = 0 \text{ s}^3/\text{rad}^2$,

⁶We classify an Ipopt solution as successful when Ipopt returns `Solve_Succeeded` (locally optimal solution) or `Solved_To_Acceptable_Level` (solution satisfying the acceptable tolerance level). For more details, see [1].

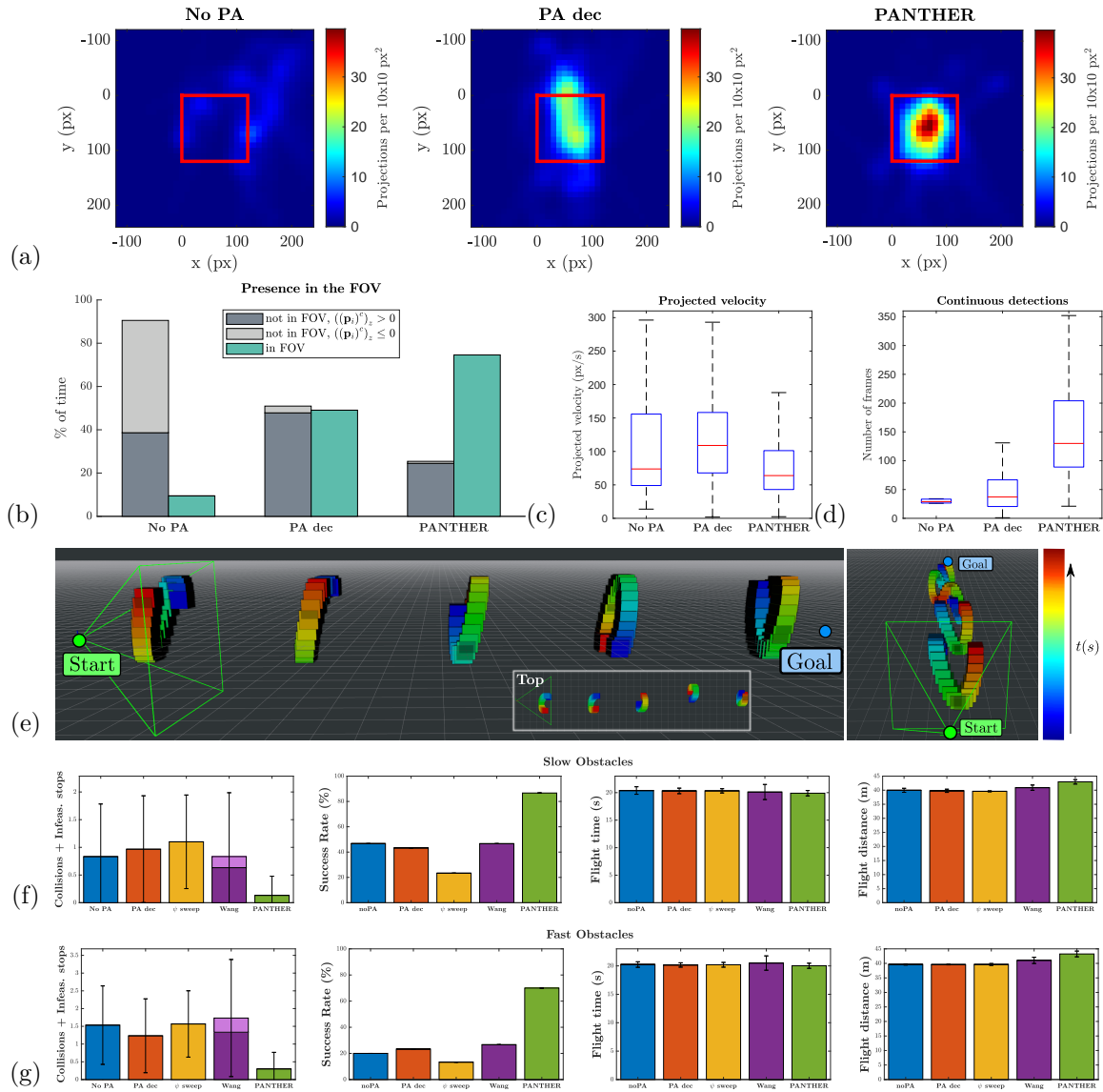


Figure 4-2: (a) Projections of the obstacle onto the image plane in the single-obstacle simulation experiments. The red square is the image plane, so any projection out of this region is not in the FOV of the camera. (b) Percentage of the time the obstacle was not in the FOV but in front of the camera (\blacksquare), not in the FOV and behind the camera (\square), and in the FOV (\blacksquare). (c) Velocity of the projection of the centroid of the obstacle onto the image plane. Higher projected velocities produce larger blur in the image. (d) Number of frames for each continuous detection. (e) Corridor simulation with five dynamic obstacles following random trefoil-knot trajectories [110]. The green pyramid represents the FOV of the camera. (f, g) Results for the corridor simulations with slow and fast obstacles, respectively. The algorithms considered are *no PA* (\blacksquare), *PA dec* (\square), ψ sweep (\blacksquare), Wang [173] (\blacksquare), and PANTHER (\blacksquare). In the left plot of both subfigures, \blacksquare represents the number of infeasible stops of algorithm [173]. The other algorithms have zero infeasible stops.

$\alpha_{\text{FOV}} = 20$, $\alpha_g = 70 \text{ m}^{-2}$, $\epsilon_1 = 0.3$, and $\epsilon_2 = 0.45 \text{ s}^2/\text{m}^2$. To focus this comparison on the capabilities of the planner, we let the agent perfectly know the trajectory of the obstacle in these simulations. We compare the following three approaches:

1. **No PA:** ψ is held constant and only the smoothness in position and terminal goal costs are optimized. Works that do not plan ψ include, e.g., [26, 43, 143].
2. **PA with position and ψ decoupled:** Translation \mathbf{p} is optimized first (as in the method *no PA*) and then it is held fixed while ψ is optimized with the PA term. We will refer to this algorithm as **PA dec**. This decoupling is done in, e.g., [119, 149, 183].
3. **PANTHER** (ours): Joint optimization of \mathbf{p} and ψ .

As will be explained in Section 4.2.4.2, two important metrics that characterize a good PA trajectory are the presence of the obstacle in the FOV and the norm of the projected velocity, which quantifies the blur. The percentage of time the obstacle was in the FOV of the camera is shown in Fig. 4-2b. PANTHER is able to keep the obstacle inside the FOV 7.9 and 1.5 times more than the algorithms *no PA* and *PA dec*, respectively. As *PA dec* decouples position and ψ in the optimization, the UAV lacks the ability to modify the spatial path (only ψ) to generate a better overall trajectory.

To qualitatively show the area of the projection, we apply a Gaussian filter to the histogram of the projection of the centroid of the obstacle onto each $10 \times 10 \text{ px}^2$ cell of the image plane. The results are shown in Fig. 4-2a, where we can see that PANTHER is able to keep the obstacle inside the FOV limits much better, and more frequently, than methods *no PA* and *PA dec*.

The velocity of the projection of the centroid of the obstacle onto the image plane is shown in Fig. 4-2c, which highlights that PANTHER is able to obtain a 18% and 34% decrease in the mean of the norm of the projected velocity with respect to *no PA* and *PA dec*, respectively, achieving, therefore, a much less blurred projection of the obstacle than those two methods.

Finally, and as a continuous detection of the dynamic obstacle is crucial to achieve a good tracking and prediction, we show in Fig. 4-2d the boxplot of the number of frames of each continuous detection for the different algorithms. A continuous detection is defined as a set of consecutive frames for which the obstacle stayed in the FOV of the camera. On average, PANTHER is able to achieve continuous detections of 155 frames, while the mean number of frames per continuous detection for methods *no PA* and *PA dec* are 39 and 46 frames, respectively.

4.3.1.2 Several Obstacles

We now test PANTHER in a simulation with several obstacles. The environment consists of a corridor of length of 39 m along the x direction with five dynamic obstacles that move following random trefoil-knot trajectories [110], see Fig. 4-2e. In all these simulations, the agent only has access to the size, current position, and velocity of the obstacles that are inside the FOV of the camera. The FOV of the camera is $70^\circ \times 70^\circ$, and has a sensing range of 5 m. The dynamic limits are $\mathbf{v}_{\max} = 2.6 \cdot \mathbf{1}$ m/s, $\mathbf{a}_{\max} = 15.5 \cdot \mathbf{1}$ m/s², $\mathbf{j}_{\max} = 50.0 \cdot \mathbf{1}$ m/s³, and $\Psi_{\max} = \pi$ rad/s. The weights used for PANTHER in these simulations are $c_{\Psi_{\max}} = 10^6$, $c_{\text{FOV}} = 1$, $c_\psi = 0$ rad⁻², $\alpha_j = 10^{-7}$ s⁵/m², $\alpha_\psi = 0$ s³/rad², $\alpha_{\text{FOV}} = 40$, $\alpha_g = 25$ m⁻², $\epsilon_1 = 0.3$, and $\epsilon_2 = 10^{-5}$ s²/m². The UAV is constrained to remain in $y \in [-4, 4]$ m and $z \in [-4, 4]$ m at all times.

For the benchmark, we use the algorithms explained before (*no PA*, *PA dec*, and PANTHER), and the two additional algorithms:

- Algorithm [173], proposed by Wang et al. This approach is not perception aware, but ψ tries to make the FOV of the camera point to the direction of travel. We will refer to this algorithm as **Wang**. Note also that this algorithm does not have constraints on \mathbf{j}_{\max} and that it has a different ψ convention (it uses definition 1 of Table 4.2).

- ψ **sweep**: ψ follows a sinusoidal trajectory that varies in $[-90^\circ, 90^\circ]$ as follows:

$$\psi(t) = \frac{\pi}{2} \sin\left(\frac{\Psi_{\max}}{\pi/2} t\right)$$

We test two scenarios with different maximum velocities of the obstacles. In the slow scenario, the obstacles move with velocities up to 2.12 m/s, while in the fast scenario, the obstacles move with velocities up to 4.07 m/s. In the results, we compare the number of collisions, infeasible stops, success rate, flight time, and flight distance. An infeasible stop happens when the drone passes instantly from a nonstop condition ($\mathbf{v} \neq \mathbf{0}$ or $\mathbf{a} \neq \mathbf{0}$) to a stop condition ($\mathbf{v} = \mathbf{0}$ and $\mathbf{a} = \mathbf{0}$). A run is considered successful if the UAV is able to reach the end of the corridor while not colliding with any of the obstacles. To make these simulations closer to real-world applications, where no prior information about the trajectories of the obstacles may be available, a simple constant velocity model is used in the predictor. The obstacles themselves are moving along trefoil-knot trajectories [110].

The results, for 30 different runs per algorithm, are shown in Figs. 4-2f and 4-2g for the slow and fast environments, respectively. In the slow scenario, PANTHER is able to succeed 87% of the runs, while the other algorithms have a success rate below 47%. None of the algorithms present infeasible stops except *Wang*, that has a mean of 0.2 infeasible stops per run (light purple in Fig. 4-2f). In the fast scenario, PANTHER succeeds 70% of the runs, while all the other algorithms have a success rate below 27%. In terms of flight times and flight distances, most of the algorithms achieve very similar results in both scenarios, with a total flight time of approximately 20 s, and an approximate total flight distance of 41 m. The total flight distance for PANTHER is approximately 3 m more than the rest of the algorithms. This is expected, because PANTHER has the ability to modify the spatial path to maximize the visibility of the obstacles. Even with this longer flight distance, the flight time of PANTHER is very similar (and sometimes even shorter) than the rest of the algorithms.

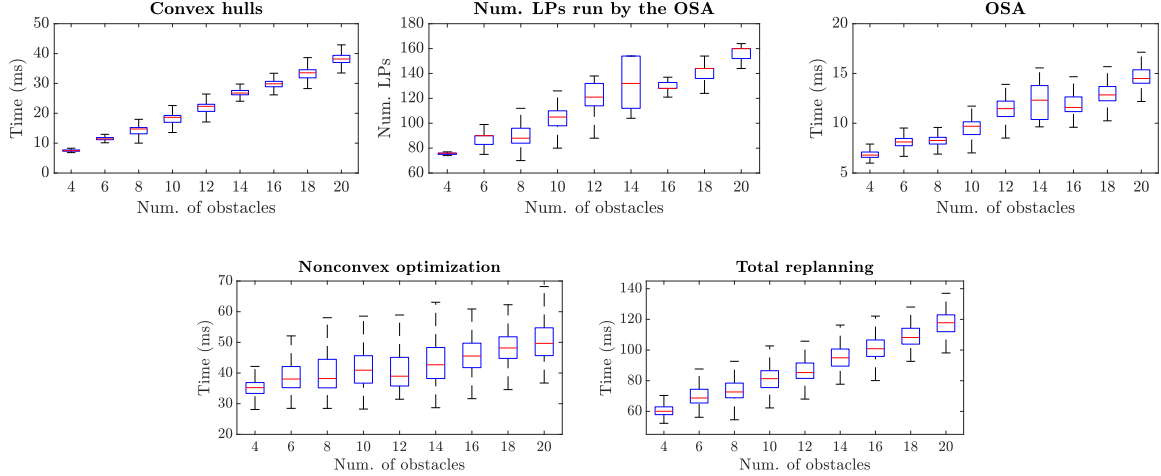


Figure 4-3: Computational analysis of different parts of the replanning step of PANTHER as a function of the number of obstacles. From left to right, and top to bottom: computation time of the generation of the convex hulls, number of linear programs (LPs) run by the OSA, computation time of the OSA, computation time of the nonconvex optimization, and total replanning time.

4.3.1.3 Computational Analysis of the Replanning Step as a Function of the Number of Obstacles

We now compare the computational cost of different parts of the replanning step of PANTHER. As the computational cost of each part highly depends on the specific position of the obstacles relative to the UAV, we perform a Monte Carlo analysis by randomly deploying obstacles (which follow trefoil-knot trajectories) in the spherical shell [109] limited by two spheres of radii 2 m and 5 m. The starting location $[0\ 0\ 1]^T$ m and $\mathbf{g}_{\text{term}} = [6\ 0\ 1]^T$ m are held fixed for every replanning iteration. The number of obstacles tested are $\{4, 6, \dots, 18, 20\}$, and, for each number of obstacles, we run 10 simulations of 5.0 s each. For these simulations, the UAV includes all the deployed obstacles in the planning problem (i.e., the set I contains the indexes of all the obstacles deployed), and we let the UAV know the trajectory of the obstacles perfectly. The weights used are the same as the ones used in Section 4.3.1.2. The results are shown in Fig. 4-3, where can see that the computation time required for the convex hull generation, the OSA, the optimization, and the total replanning time change approximately linearly with the number of obstacles. Similarly, the number of

linear programs run by the OSA also changes approximately linearly with the number of obstacles. The average solve time of one of these linear programs is 0.09 ms.

To obtain the ψ initial guess (Section 4.2.3.2), the average runtime of the Dijkstra’s algorithm on the ψ graph is 0.137 ms, and the average runtime to fit a spline to the ψ samples (Eq. 4.1) is 0.048 ms.

These results above show the computational analysis for the different parts of the replanning step of PANTHER (convex hull computation, generation of the $\mathbf{p}(t)$ and $\psi(t)$ initial guesses, and nonconvex optimization). For the computational cost of the tracker and predictor using real point clouds, see Section 4.3.2. The well-known results regarding the complexity analysis of the Hungarian algorithm are given in [39, 78].

4.3.2 Real-World Experiments

We run an extensive set of hardware experiments, where a UAV needs to go from a starting point to a goal location while avoiding unknown dynamic obstacles. The UAV used is equipped with a Qualcomm[®] Snapdragon Flight, an Intel[®] NUC i7DNK, and an Intel[®] RealSense Depth camera D435i. The tracker, planner, and the camera run on the Intel[®] NUC, while the control and state estimation run on the Qualcomm[®] Snapdragon Flight. Note that the main onboard computer (Intel[®] NUC) has similar computational power to the onboard hardware used in the recent literature (e.g., [26, 143, 173, 183]). For the controller, we run the approach presented in [95, 174] at 100 Hz to generate the desired orientation and angular rates from $\mathbf{p}(t)$ and $\psi(t)$. The commanded thrusts for the motors are then found from these attitude commands using a geometric controller [84], which is run at IMU rate (500 Hz). For state estimation, we use a visual inertial odometry (VIO) package [2] running at 30 Hz that leverages an extended Kalman filter to fuse the IMU measurements of the Snapdragon and the images of its downward-facing camera. To obtain a high-rate state estimate, we then integrate forward the IMU (which runs at 500 Hz) between consecutive VIO estimates.

The IMU of the RealSense camera is not used. All the computation of this UAV is running onboard, and it does not have any prior knowledge of the trajectories

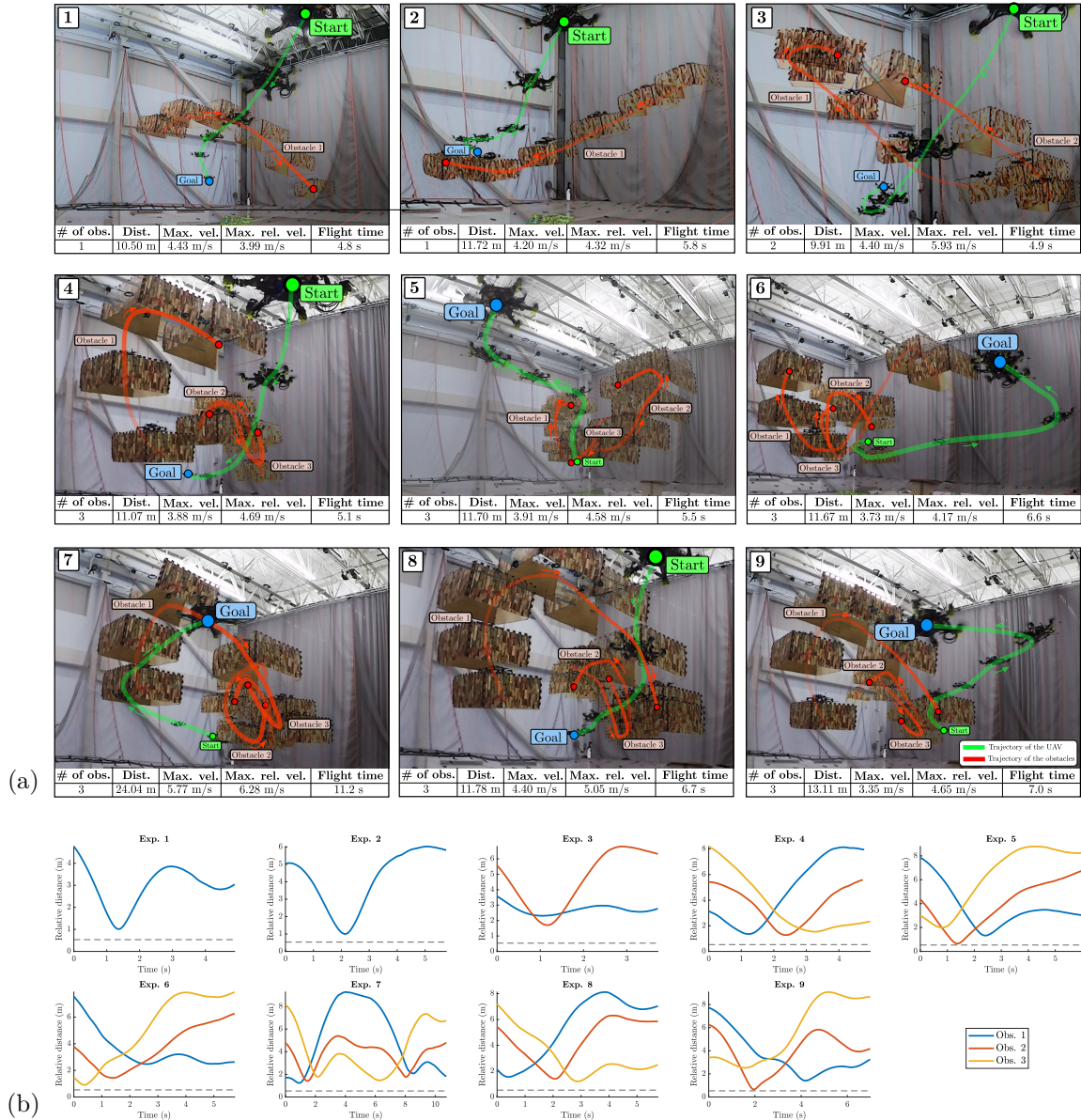
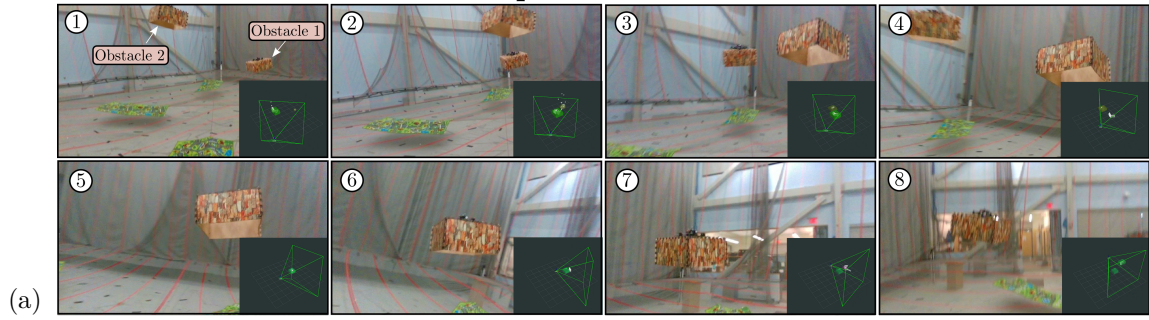
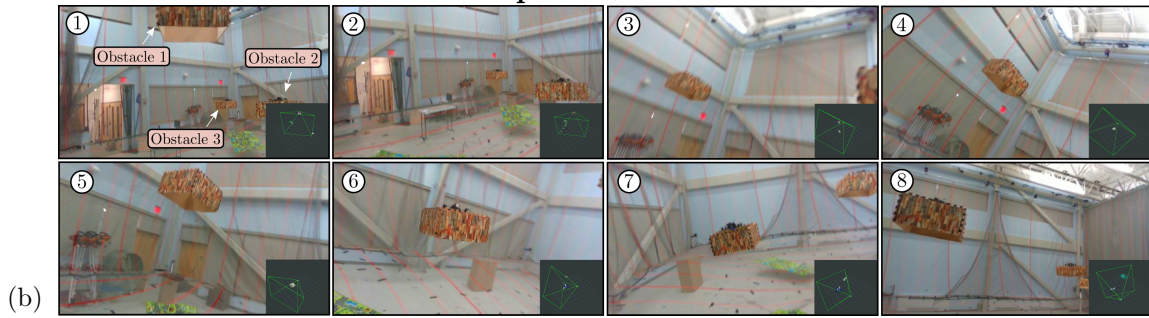


Figure 4-4: (a) Composite images of all the nine experiments. For visualization purposes, only the second half of Experiment 7 is shown. The table below every image shows the number of obstacles, flight distance, maximum velocity, maximum relative velocity (with respect to the obstacles), and flight time of each experiment. The number of obstacles is one, two, and three for the experiments 1-2, 3, and 4-9 respectively. (b) Relative distances between the agent and each one of the obstacles. Any relative distance above the dashed line guarantees safety.

Experiment 3



Experiment 6



Experiment 9

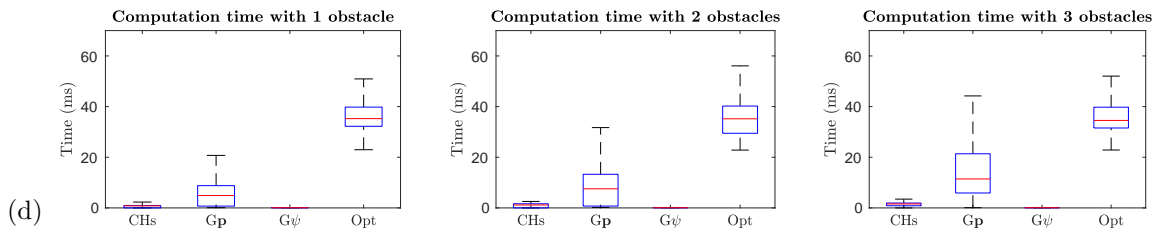
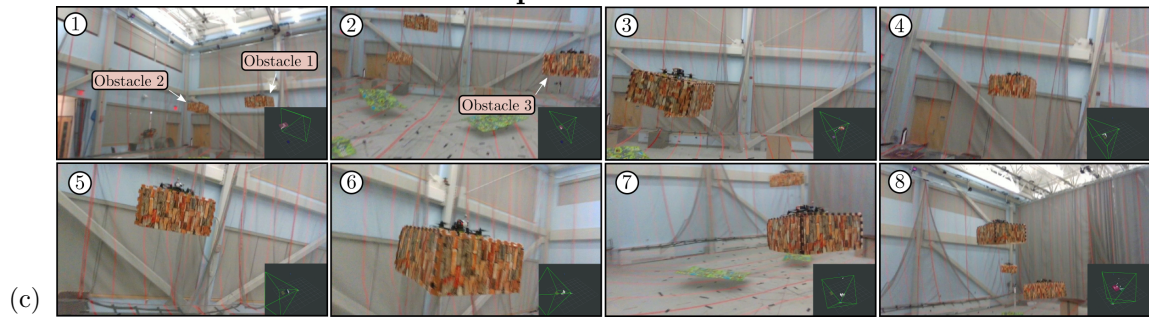


Figure 4-5: Snapshots of the onboard camera in experiments 3 (a), 6 (b), and 9 (c). (d) Computation times for each part of a replanning step, measured on the onboard Intel® NUC i7DNK. The tracker, predictor, and the depth camera were also running on this computer at the same time these times were measured. The notation used is: CHs (convex hull computation for the polyhedral outer representations), Gp (generation of the planes and the guess for the position $\mathbf{p}(t)$), G ψ (generation of the guess for $\psi(t)$) and Opt (Optimization time).

and specific shape/size of the obstacles. The weights used for these experiments are $c_{\Psi_{\max}} = 10^6$, $c_{\text{FOV}} = 1$, $c_{\psi} = 0 \text{ rad}^{-2}$, $\alpha_j = 0.05 \text{ s}^5/\text{m}^2$, $\alpha_{\psi} = 0.1 \text{ s}^3/\text{rad}^2$, $\alpha_{\text{FOV}} = 1$, $\alpha_g = 2 \cdot 10^4 \text{ m}^{-2}$, $\epsilon_1 = 0.1$, and $\epsilon_2 = 1 \text{ s}^2/\text{m}^2$.

To generate the dynamic obstacles, we use three other UAVs with a Qualcomm[®] SnapDragon Flight, and equip them with a box-shaped frame of $\approx 0.6 \times 0.6 \times 0.3 \text{ m}^3$. The obstacles are following trefoil-knot trajectories [110].

A total of 9 experiments were performed (see the [video](#)). The composite images of the trajectories flown by the agent and by the obstacles, together with the number of obstacles, distance flown, maximum velocity, maximum relative velocity with respect to the obstacles, and total flight time of each one of the experiments are shown in Fig. 4-4a. Experiments 1 and 2 were done with one obstacle, experiment 3 with two obstacles, and experiments 4-9 with three obstacles. The maximum velocity achieved by the agent, 5.77 m/s, happened in experiment 7. In that same experiment, the maximum relative velocity (6.28 m/s) with respect to the obstacles is also achieved. The relative distances between the UAV and the obstacles are shown in Fig. 4-4b. Any relative distance above the dashed horizontal line guarantees safety between the agent and the corresponding obstacle. For experiments 3, 6, and 9, different snapshots of the onboard camera are shown in Figs. 4-5a, 4-5b, and 4-5c, respectively. Note how the planned trajectories try to keep an obstacle in the FOV at all times to aid in obstacle tracking and prediction.

The computation times are shown in Fig. 4-5d. All these computation times were measured onboard, with the UAV flying, and with the depth camera node and the tracker running on the same computer (Intel[®] NUC i7DNK). The mean total replanning times are 48.70, 51.66, and 58.59 ms for the experiments with 1, 2, and 3 obstacles respectively. The point cloud of the camera is generated at 90 Hz, and the tracker (clustering, assignment, and prediction) is able to process each point cloud in $\approx 8.6 \text{ ms}$.

Chapter 5

Deep-PANTHER: Learning-Based Perception-Aware Trajectory Planner in Dynamic Environments

5.1 Overview

This chapter presents Deep-PANTHER, a learning-based perception-aware trajectory planner for UAVs in dynamic environments. Given the current state of the UAV, and the predicted trajectory and size of the obstacle, Deep-PANTHER generates multiple trajectories to avoid a dynamic obstacle while simultaneously maximizing its presence in the field of view (FOV) of the onboard camera. To obtain a computationally tractable real-time solution, imitation learning is leveraged to train a Deep-PANTHER policy using demonstrations provided by a multimodal optimization-based expert. Extensive simulations show replanning times that are two orders of magnitude faster than the optimization-based expert, while achieving a similar cost. By ensuring that each expert trajectory is assigned to one distinct student trajectory in the loss function, Deep-PANTHER can also capture the multimodality of the problem and achieve a mean squared error (MSE) loss with respect to the expert that is up to 18 times smaller than state-of-the-art (Relaxed) Winner-Takes-All

approaches. Deep-PANTHER is also shown to generalize well to obstacle trajectories that differ from the ones used in training.

This chapter uses the notation shown in Table 5.1.

5.2 Deep-PANTHER

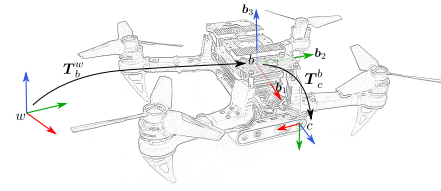
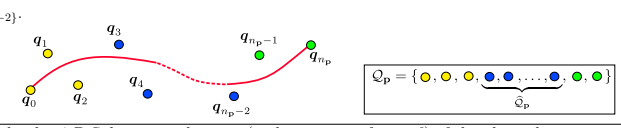
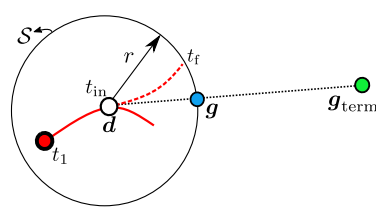
Deep-PANTHER is a multimodal trajectory planner able to generate a trajectory that avoids a dynamic obstacle, while trying to keep it in the FOV. To achieve very fast computation times, we leverage imitation learning, where Deep-PANTHER is the student (a neural network) that is trained to imitate the position trajectories generated by an optimization-based expert (Section 5.2.1). Both the student and the expert have an observation as input and an action as output (Section 5.2.2). The multimodality is captured through the design of the loss function (Section 5.2.3), and the trajectories for the extra degree of freedom of the rotation (ψ) can then be obtained from the position trajectories (Section 5.2.4). The final trajectory chosen for execution is obtained according to the cost and the constraint satisfaction (Section 5.2.5).

5.2.1 Expert and Student

Chapter 4 (Ref. [163]) presented PANTHER, an optimization-based perception-aware trajectory planner able to avoid dynamic obstacles while keeping them in the FOV. However, and as discussed in Section 1.3.4, real-time computation was achieved at the expense of conservative solutions. Hence, we design PANTHER* (the expert) by reducing the conservativeness of PANTHER as follows:

- The planes that separate the trajectory of the UAV from the obstacles (Fig. 4-1b) and the total time of the planned trajectory T are included as decision variables. To ensure that T does not go beyond the prediction horizon, the constraint $0 \leq T \leq T_{\text{pred}}$ is imposed for both the expert and the student. Here, T_{pred} is the total time of the future predicted trajectory of the obstacle, and it

Table 5.1: Notation used in this chapter.

Symbol	Meaning
$\mathcal{S}_{p,m}^d$	Set of clamped uniform splines with dimension d , degree p , and $m + 1$ knots.
\circ	Quaternion multiplication.
\odot	Element-wise product.
$\mathcal{U}(a,b)$	Uniform distribution in $[a,b]$.
$(\mathbf{a})_n, \hat{\mathbf{a}}$	Vector \mathbf{a} normalized: $(\mathbf{a})_n \equiv \hat{\mathbf{a}} := \frac{\mathbf{a}}{\ \mathbf{a}\ }$.
\mathbf{p}^a	Point expressed in the frame a . For the definitions of this table that include the sentence “expressed in the world frame”, the notation of the frame is omitted.
$\mathbf{T}_b^a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0}^T & 1 \end{bmatrix}$	Transformation matrix: $\begin{bmatrix} \mathbf{p}^a \\ 1 \end{bmatrix} = \mathbf{T}_b^a \begin{bmatrix} \mathbf{p}^b \\ 1 \end{bmatrix}$. Analogous definition for the quaternion \mathbf{q}_b^a .
$\mathbf{e}_x, \mathbf{e}_z, \mathbf{1}$	$\mathbf{e}_x := [1\ 0\ 0]^T$, $\mathbf{e}_z := [0\ 0\ 1]^T$, $\mathbf{1} := [1\ 1\ \dots\ 1]^T$.
FOV, MSE, LSA	Field of view, mean squared error, linear sum assignment.
\mathbf{p}	Position of the body frame expressed in the world frame. I.e., $\mathbf{p} := \mathbf{t}_b^w$.
\mathbf{a}	Acceleration of the body frame w.r.t. the world frame, and expressed in the world frame.
\mathbf{p}_{obst}	Mean of the predicted position of obstacle, expressed in the world frame.
g	$g \approx 9.81$ m/s ² .
ξ	Relative acceleration, expressed in the world frame: $\xi := [\mathbf{a}_x\ \mathbf{a}_y\ \mathbf{a}_z + g]^T$. We will assume $\xi \neq \mathbf{0}$.
$[q_w\ q_x\ q_y\ q_z]^T$	Components of a unit quaternion.
ψ	Angle such that $\mathbf{q}_b^w = \frac{1}{\sqrt{2(1+\xi_x)}} \begin{bmatrix} 1+\xi_x \\ -\xi_y \\ \xi_x \\ 0 \end{bmatrix} \circ \begin{bmatrix} c_{\psi/2} \\ 0 \\ 0 \\ s_{\psi/2} \end{bmatrix}$ ([163,174]).
World frame (w), body frame (b) and camera frame (c)	 <p>$\mathbf{R}_b^w := [\mathbf{b}_1\ \mathbf{b}_2\ \mathbf{b}_3]$. $\mathbf{b}_3 = (\xi)_n$ due to the perpendicularity of the total thrust with respect to the plane spanned by \mathbf{b}_1 and \mathbf{b}_2.</p>
Frame f	Coordinate frame such that $\mathbf{t}_f^b = \mathbf{0}$, $\mathbf{R}_f^b \mathbf{e}_z = [0\ 0\ 1]^T$, and that has the same ψ as the frame b .
$\mathbf{v}^f, \mathbf{a}^f$	Velocity and Acceleration of the body w.r.t. the world frame, and expressed in the frame f . $\in \mathbb{R}^3$.
θ	Opening angle of the cone that approximates the FOV.
n (n_p and n_ψ)	$n := m - p - 1$. $n + 1$ is the number of control points of the spline.
L_p, L_ψ	$L_p := \{0, 1, \dots, n_p\}$, $L_\psi := \{0, 1, \dots, n_\psi\}$.
l	Index of the control point. $l \in L_p$ for $\mathbf{p}(t)$, $l \in L_\psi$ for $\psi(t)$.
\mathbf{q}_l, ψ_l	Position B-Spline control point ($\in \mathbb{R}^3$), ψ B-Spline control point ($\in \mathbb{R}$).
$\mathcal{Q}_p, \hat{\mathcal{Q}}_p$	$\mathcal{Q}_p := \{(\mathbf{q}_l)^f\}_{l \in L_p}$. In other words, the position B-Spline control points of the planned trajectory for the UAV, expressed in frame f . $\hat{\mathcal{Q}}_p := \{(\mathbf{q}_l)^f\}_{l \in \{3, \dots, n_p-2\}}$. 
\mathcal{Q}_ψ	$\{\psi_l\}_{l \in L_\psi}$. In other words, the ψ B-Spline control points (with respect to frame f) of the planned trajectory for the UAV.
$\mathcal{Q}_{p,\text{obst}}$	B-Spline control points of a spline fit to the future predicted trajectory of obstacle. The future predicted trajectory of the obstacle can be obtained using a prediction module as in Section 4.2.1.
s_{obst}	Length of each side of the axis-aligned bounding box of the obstacle. $\in \mathbb{R}^3$.
T_{pred}	Prediction time for the future trajectory of the obstacle(s).
n_s	Number of trajectories produced by the student. It is a user-chosen parameter, and it is fixed (i.e., does not change between replanning steps).
$n_{\text{runs}}, n_{\text{sols}}, n_c$	The optimization problem of the expert is run n_{runs} times (with different initial guesses), producing $n_{\text{sols}} \leq n_{\text{runs}}$ distinct trajectories. The trajectories produced by the expert are then the best $n_c = \min(n_{\text{sols}}, n_s)$ trajectories obtained.
Snapshot at $t = t_1$ (current time):	 <p>\mathbf{g}_{term} (●) is the terminal goal, and ● is the current position of the UAV. --- is the trajectory the UAV is currently executing. -.-.- is the trajectory the UAV is currently planning, $t \in [t_{\text{in}}, t_t]$. \mathbf{d} (○) is a point in ---, used as the initial position of -.-.-. \mathcal{M} is a sphere of radius r around \mathbf{d}. \mathbf{g} (●) is the projection of \mathbf{g}_{term} (●) onto the sphere \mathcal{M}. T is the total time of the planned trajectory. I.e., $T := t_t - t_{\text{in}}$.</p>

is a user-chosen parameter.

- The future predicted trajectory of the obstacle is a spline whose control points are $\mathcal{Q}_{\mathbf{p},\text{obst}}$.
- The optimization problem is run n_{runs} times (with different initial guesses obtained by running the OSA, see Section 3.5.5), and $n_{\text{sols}} \leq n_{\text{runs}}$ distinct trajectories are obtained.

The student (Deep-PANTHER) consists of a fully connected feedforward neural network with two hidden layers, 64 neurons per layer, and with the ReLU activation function. The student produces a total of n_s trajectories, where n_s is a user-chosen parameter. Note that the trajectories produced by the expert are then the best $n_e = \min(n_{\text{sols}}, n_s)$ trajectories obtained in the optimization.

5.2.2 Observation and Action

We use the observation $(\mathbf{v}^f, \mathbf{a}^f, \mathbf{g}^f, \mathcal{Q}_{\mathbf{p},\text{obst}}, \mathbf{s}_{\text{obst}})$, where, as defined in Table 5.1, \mathbf{v}^f , \mathbf{a}^f , \mathbf{g}^f , and $\mathcal{Q}_{\mathbf{p},\text{obst}}$ are, respectively, the velocity of the UAV, the acceleration of the UAV, the projection of the terminal goal \mathbf{g}_{term} , and the control points of a spline fit to the future predicted trajectory of the obstacle. All of these quantities are expressed in the frame f . $\mathbf{s}_{\text{obst}} \in \mathbb{R}^3$ contains the length of each side of the axis-aligned bounding box of the obstacle. In this work, we use a spline in $\mathcal{S}_{3,13}^3$ for the predicted trajectory of the obstacle, leading to an observation size of 43.

The action is given by $(\mathcal{T}_k)_{k \in \{0, \dots, \beta-1\}}$, where $\beta = n_s$ for the student, and $\beta = n_e$ for the expert, and where

$$\mathcal{T}_k := \left(\left(\hat{\mathcal{Q}}_{\mathbf{p}} \right)_k, T_k \right) .$$

As defined in Table 5.1, $\hat{\mathcal{Q}}_{\mathbf{p}}$ contains all the B-Spline control points of the planned trajectory expressed in frame f except the first three and the last two, while T is the total time of the planned trajectory. Note that the first three and the last two control points need not to be included in the action because they are determined directly from the total time T and the initial and final conditions. We model the planned

trajectories (for both the expert and the student) as splines in $\mathcal{S}_{3,12}^3$, leading to an action size of 13β . The relationship between n_s and n_e is explained in Section 5.2.1 and Table 5.1.

The key advantage of using $\widehat{\mathcal{Q}}_{\mathbf{p}}$ instead of $\mathcal{Q}_{\mathbf{p}}$ is that every trajectory generated by the student will satisfy by construction the initial and final conditions for any given observation. It also helps reduce the action size. Moreover, the advantage of using the B-Spline position control points, instead of sampled future positions as in [96], is that every trajectory generated by the student is smooth by construction (\mathcal{C}^2 -continuous in our case), and it also avoids the need of a post-projection step into polynomial space.

5.2.3 Loss: Capturing Multimodality

As discussed in Section 1.3.4 and Fig. 1-3, the number of trajectories found by the expert changes depending on the specific observation. To train a neural network with a fixed-size output (n_s trajectories) to predict the varying-size output of the expert (n_e trajectories), we propose to use the approach shown in Fig. 5-1. The observation is passed through the neural network of the student to generate n_s trajectories, and through the expert to produce n_e trajectories. We then define $\mathbf{D}_{\mathbf{p}}$ as a matrix whose element (i, j) is the mean squared error (MSE) between the position control points of the i -th trajectory of the expert and the position control points of the j -th trajectory of the student. A similar definition applies to \mathbf{D}_T , but using the total time of the trajectory instead of the control points.

Letting \mathbf{A} denote the assignment matrix (whose (i, j) element is 1 if the i -th trajectory of the expert has been assigned to the j -th trajectory of the student, and 0 otherwise), we then find the optimal \mathbf{A} that minimizes the assignment cost $\mathbf{1}^T (\mathbf{A} \odot \mathbf{D}_{\mathbf{p}}) \mathbf{1}$, and that assigns a distinct student trajectory to every expert trajectory. Here, \odot denotes the element-wise product and $\mathbf{1}$ is a column vector of ones. This is an instance of the linear sum assignment (LSA) problem, and the optimal \mathbf{A} can be obtained leveraging the Jonker-Volgenant algorithm [33] (a variant of the Hungarian algorithm [78]). As we have that $n_e \leq n_s$, all the rows of \mathbf{A} sum up

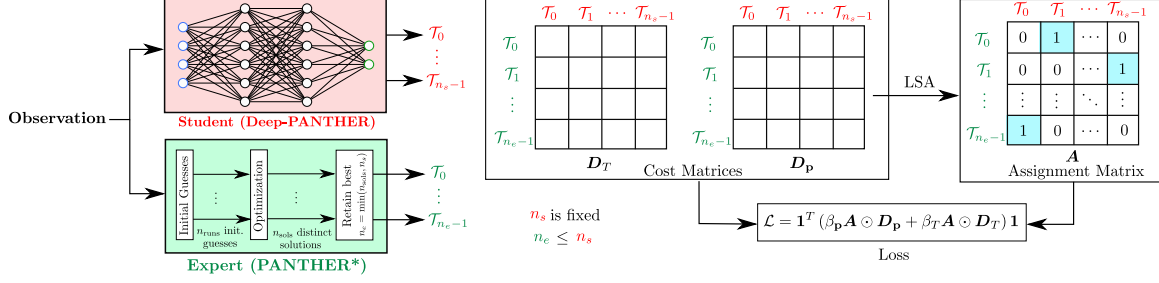


Figure 5-1: Multimodal training in Deep-PANTHER. The student outputs a fixed number of trajectories, denoted as n_s . The expert (PANTHER*) produces n_e trajectories, where $n_e \leq n_s$. Then, the cost matrix in position space (D_p) and in time space (D_T) are computed. Using D_p , the linear sum assignment (LSA) problem is solved to find the assignment matrix A , which is then used in the loss to penalize the expert-student assigned pairs.

to 1, n_e columns sum up to 1, and $(n_s - n_e)$ columns sum up to 0. To penalize only the MSE of the optimally-assigned student-expert pairs, the loss is then computed as

$$\mathcal{L} = \mathbf{1}^T (\beta_p \mathbf{A} \odot \mathbf{D}_p + \beta_T \mathbf{A} \odot \mathbf{D}_T) \mathbf{1} ,$$

where β_p and β_T are user-chosen weights.

Our approach ensures that all the expert trajectories have exactly one distinct student trajectory assigned to them, see Fig 1-4. Compared to WTAr, RWTAr, WTAc, and RWTAc, our LSA loss prevents the same student trajectory from being assigned to several expert trajectories (reducing therefore the equilibrium issues), guarantees that all the trajectories of the expert are captured in every training step, and also prevents the same expert trajectory from having several student trajectories assigned to it (being therefore less prone the mode collapse problems).

5.2.4 Generation of ψ Given the Position Trajectory

Each \mathcal{T}_k , together with the initial and final conditions contained in the observation, defines the position trajectory. Since $\mathbf{b}_3 := \mathbf{R}_b^w \mathbf{e}_z = (\boldsymbol{\xi})_n$ (see Fig. 5-2 and Table 5.1), this position trajectory determines part of the rotation, but leaves ψ free. We now

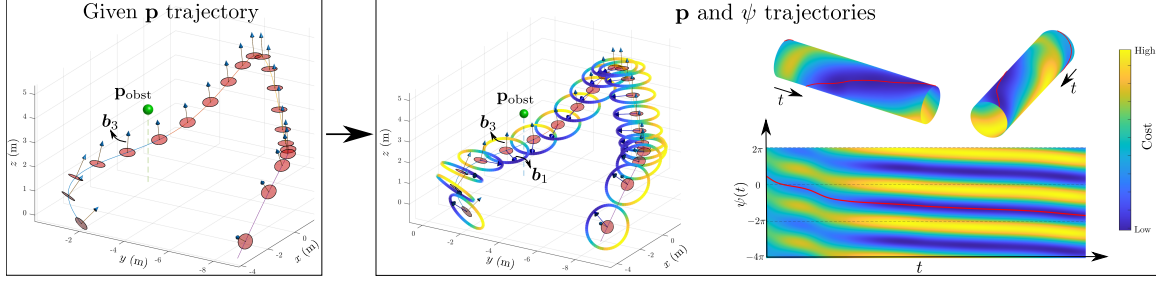


Figure 5-2: Optimal ψ trajectory (— in the right plots) given the position trajectory defined by $((\mathcal{Q}_{\mathbf{p}})_k, T_k)$. A spline is then fit to this ψ trajectory found. The sphere \bullet denotes the position of the obstacle \mathbf{p}_{obst} . For visualization purposes, we show here a static obstacle, but this method is also applicable when the obstacle is dynamic.

derive¹ a closed-form expression for $\psi(t)$ that maximizes the presence of the obstacle in the FOV given the position trajectory. Let $\mathbf{p} := \mathbf{t}_b^w$ be the position of the UAV, and let \mathbf{p}_{obst} denote the position of the obstacle (both expressed in the world frame). Let us also define $\mathbf{b}_1 := \mathbf{R}_b^w \mathbf{e}_x$. Using a cone with opening angle θ to model the FOV, the obstacle is in the FOV if and only if $\cos(\theta/2) \leq \mathbf{b}_1^T (\mathbf{p}_{\text{obst}} - \mathbf{p})_n$. We can therefore maximize the presence of the obstacle in the FOV by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{b}_1} \quad & -\mathbf{b}_1^T (\mathbf{p}_{\text{obst}} - \mathbf{p})_n \\ \text{s.t.} \quad & \mathbf{b}_1^T \boldsymbol{\xi} = 0 \\ & \|\mathbf{b}_1\|^2 = 1 \end{aligned}$$

where the two constraints guarantee that \mathbf{b}_1 is a unit vector perpendicular to $\boldsymbol{\xi}$. Computing the Lagrangian and solving the Karush-Kuhn-Tucker (KKT) conditions [72, 79] yields the optimal solution:²

$$\mathbf{b}_1 = \left((\mathbf{p}_{\text{obst}} - \mathbf{p}) - \frac{(\mathbf{p}_{\text{obst}} - \mathbf{p})^T \boldsymbol{\xi}}{\|\boldsymbol{\xi}\|^2} \boldsymbol{\xi} \right)_n \quad (5.1)$$

¹For simplicity, here we focus on the case where $\mathbf{R}_c^b = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$ and $\mathbf{t}_c^b = \mathbf{0}$. A similar derivation applies to more general cases. See also [44, Section II-D].

²Note that Eq. 5.1 presents a singularity when $(\mathbf{p}_{\text{obst}} - \mathbf{p})$ is parallel to $\boldsymbol{\xi}$. In that case, we can choose any \mathbf{b}_1 , since all of them are perpendicular to $(\mathbf{p}_{\text{obst}} - \mathbf{p})$ and therefore achieve the same (zero) cost in the objective function.

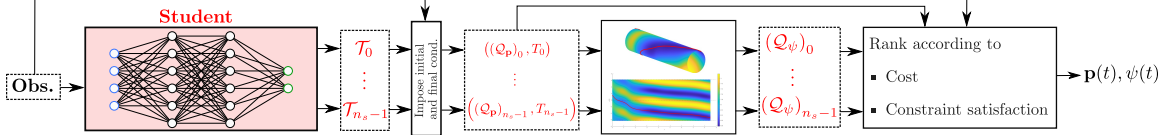


Figure 5-3: Generation of $\mathbf{p}(t)$ and $\psi(t)$ from the observation. The observation is fed into the neural network, which produces $(\mathcal{T}_k)_{k \in \{0, \dots, n_s-1\}}$. Then, for each \mathcal{T}_k , the initial and final conditions are imposed to generate the position trajectory, defined by all the position control points $(\mathcal{Q}_p)_k$ and the total time T_k . Using the results from Section 5.2.4, we then obtain $(\mathcal{Q}_\psi)_k$. The trajectory $((\mathcal{Q}_p)_k, (\mathcal{Q}_\psi)_k, T_k)$ that is collision-free and that has the smallest augmented cost is then chosen for execution.

Given that \mathbf{p}_{obst} , \mathbf{p} , and $\boldsymbol{\xi}$ are functions of time, Eq. 5.1 gives the evolution of \mathbf{b}_1 that maximizes the presence of the obstacle in the FOV (see Fig. 5-2). $\psi(t)$ can then be easily obtained from \mathbf{b}_1 and \mathbf{b}_3 , and a spline is fit to it to obtain the control points \mathcal{Q}_ψ .

Note that, in PANTHER*, position and rotation are coupled together in the optimization (as in Section 4.2.4.1). This coupling helps reduce the conservativeness that arises when they are optimized separately [119, 149, 183]. Deep-PANTHER (the student) learns to predict the position trajectory resulting from this *coupled* optimization problem, and then the closed-form solution is leveraged to obtain ψ from this position trajectory. In other words, Deep-PANTHER benefits from the coupling (since it is learning one of the outputs of the coupled optimization problem), while leveraging the closed-form solution for ψ .

5.2.5 Testing

In testing time the procedure is as follows (see Fig. 5-3): The observation is fed into the neural network, which produces $(\mathcal{T}_k)_{k \in \{0, \dots, n_s-1\}}$ (i.e., the intermediate position control points and the total times). Then, for each \mathcal{T}_k , the initial and final conditions are imposed to generate the position trajectory, defined by all the position control points $(\mathcal{Q}_p)_k$ and the total time T_k . The optimal ψ control points $(\mathcal{Q}_\psi)_k$ are then obtained as explained in Section 5.2.4. Then, and using the observation, each triple $((\mathcal{Q}_p)_k, (\mathcal{Q}_\psi)_k, T_k)$ is ranked according to the cost and the constraint satisfaction. The trajectory chosen for execution is then the one that is collision-free and achieves the smallest augmented cost, which is defined as $c_{\text{obj}} + \lambda c_{\text{dyn lim}}$, where c_{obj} is the cost

of the objective function (same one as the one used by PANTHER*), $c_{\text{dyn lim}}$ is a soft cost that penalizes the velocity, acceleration, and jerk violations, and $\lambda > 0$. If none of the trajectories generated by the student are collision-free, the UAV will continue executing the trajectory it had in the previous replanning step (which is collision-free) and will replan again.

5.3 Results and Discussion

To better compare the different aspects of the proposed framework, Section 5.3.1 first focuses on a stopped UAV that needs to plan a trajectory from the start location to the goal (without moving along that planned trajectory) while avoiding a static obstacle. Then, Section 5.3.2 studies the more general case where a UAV is flying and constantly replanning in a dynamic environment.

We use $n_s = 6$, $n_{\text{runs}} = 10$, $T_{\text{pred}} = 6$ s, and $\beta_{\mathbf{p}} = \beta_T = 1$.³ To train the neural network we use the Adam optimizer [74] and a learning rate of 10^{-3} . In all these simulations, and for all the algorithms tested, $\mathcal{Q}_{\mathbf{p}, \text{obst}}$ is obtained by simply fitting a spline to the ground-truth future positions of the obstacle. In real-world applications, this future predicted trajectory of the obstacle can be obtained from past observations as explained in Section 4.2.1.

5.3.1 Static Obstacle

In this section, the task is to plan once from the starting location to the goal (i.e., the UAV does not move along the planned trajectory and/or replan again). We collect 2K (observation, expert action) pairs,⁴ and use 75% of these pairs to train the student

³Note that $\beta_{\mathbf{p}}$ and β_T are adimensional because $\mathbf{D}_{\mathbf{p}}$ and \mathbf{D}_T are computed from normalized actions in $[-1, 1]$.

⁴In these collected expert demonstrations, the UAV starts stopped at $[0 \ 0 \ 1]^T$, $\mathbf{p}_{\text{obst}} = \mathbf{f}(r_{\text{obst}}, \theta_{\text{obst}}, 1)$, and $\mathbf{g}_{\text{term}} = \mathbf{f}(r_{\mathbf{g}_{\text{term}}}, \theta_{\mathbf{g}_{\text{term}}}, z_{\mathbf{g}_{\text{term}}})$, where:

$$\begin{cases} \mathbf{f}(r, \theta, z) := [r \cos(\theta) \ r \sin(\theta) \ z]^T \\ r_{\text{obst}} \sim \mathcal{U}(1.5 \text{ m}, 4.5 \text{ m}) \\ \theta_{\text{obst}} \sim \mathcal{U}(-\frac{\pi}{2} \text{ rad}, \frac{\pi}{2} \text{ rad}) \\ r_{\mathbf{g}_{\text{term}}} = r_{\text{obst}} + \mathcal{U}(1.0 \text{ m}, 6.0 \text{ m}) \\ \theta_{\mathbf{g}_{\text{term}}} = \theta_{\text{obst}} + \mathcal{U}(-0.17 \text{ rad}, 0.17 \text{ rad}) \\ z_{\mathbf{g}_{\text{term}}} \sim \mathcal{U}(-0.5 \text{ m}, 2.5 \text{ m}) \end{cases}$$

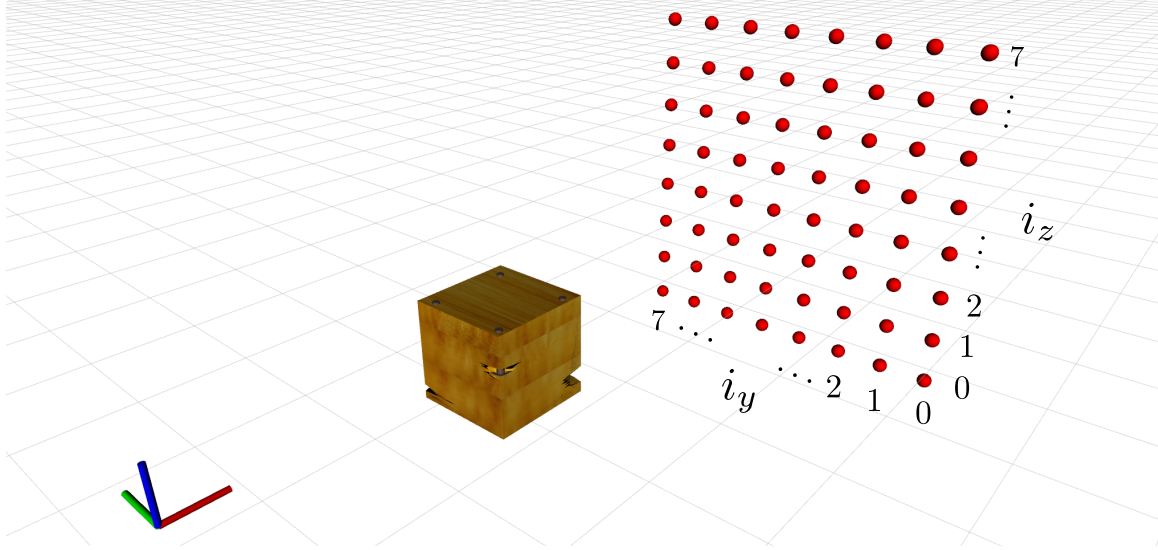


Figure 5-4: Testing scenario consisting of a static obstacle located at $\mathbf{p}_{\text{obst}} = [2.5 \ 0 \ 1]^T$ m and 64 different $\mathbf{g}_{\text{term}} = [7 \ a \ 1 + b]^T$ m (\bullet in the figure), where a and b are evenly spaced in $[-1.7, 1.7]$ m. The initial location (coordinate frame in the figure) is $[0 \ 0 \ 1]^T$ m.

offline (the rest of the pairs are used as the evaluation dataset in the MSE comparisons of Section 5.3.1.2). Section 5.3.1.1 first compares the cost vs replanning time, and then Section 5.3.1.2 analyzes how well the multimodality is captured.

5.3.1.1 Cost vs Replanning time

We compare the cost vs replanning time of these three different approaches: PANTHER (Ref. [163]), PANTHER* (the expert, see Section 5.2.1) and Deep-PANTHER (the student). The testing environment is shown in Fig. 5-4. For PANTHER* and Deep-PANTHER (which generate a multimodal output), we use in the comparisons the best (i.e., with smallest cost) collision-free trajectory found. The results are shown in Fig. 5-5, which highlights that Deep-PANTHER obtains a total cost similar to the one obtained by PANTHER*, but with a computation time that is two orders of magnitude smaller. Compared to PANTHER, Deep-PANTHER is able to obtain a lower cost, and with an improvement of one order of magnitude in computation time. For each of the 64 simulations performed, the trajectory obtained

Note that the obstacle is static but randomly located.

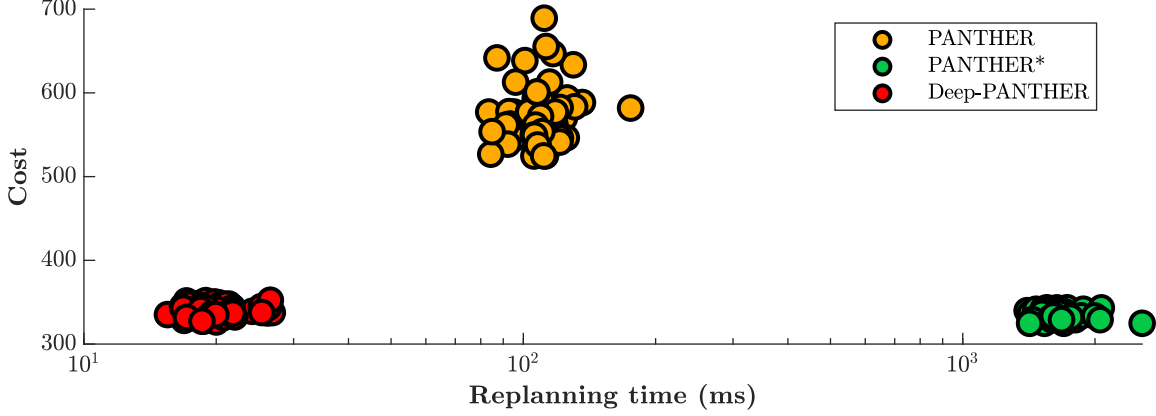


Figure 5-5: Comparison of the cost and replanning time. Deep-PANTHER is able to obtain a similar cost to the one obtained by PANTHER*, but with a computation time two order of magnitude smaller. Compared to PANTHER, Deep-PANTHER is able to get a smaller cost, and one order of magnitude faster. Note the logarithmic scale on the x axis.

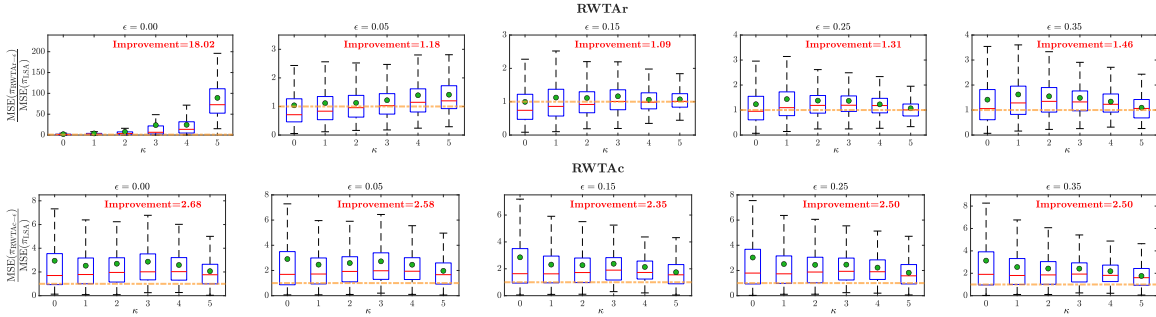


Figure 5-6: Comparison between the MSE loss of π_{LSA} , $\pi_{\text{RWTAr}-\epsilon}$, and $\pi_{\text{RWTAc}-\epsilon}$. In each of the boxplots, \bullet represents the mean. The dashed yellow line $----$ represents an MSE ratio of 1. Compared to RWTAr, our approach achieves an average MSE between 1.09 and 18.02 times smaller. Compared to RWTAc, our approach achieves an average MSE between 2.35 and 2.68 times smaller. In the plots, κ is the index of the ranking order based on the MSE loss with respect to the expert. If $n_e < n_s = 6$, that demonstration is not taken into account for the boxplots with $\kappa \geq n_e$.

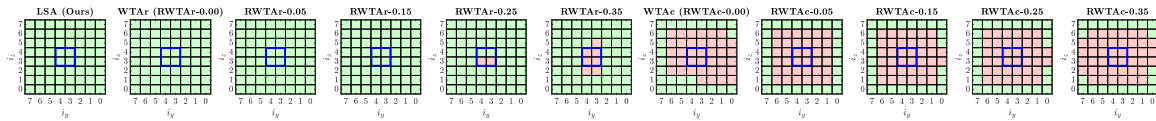


Figure 5-7: Comparison of the collision-free trajectories produced by LSA (our approach), RWTAr- ϵ , and RWTAc- ϵ . Each cell in each square represents a different \mathbf{g}_{term} (i_y and i_z are defined in Fig. 5-4), where \square means that at least one collision-free trajectory is obtained, while \blacksquare means that none of the trajectories are collision-free. The blue square \square is the projection of the obstacle onto the y - z plane.

by all the algorithms is collision-free.

5.3.1.2 Multimodality

Let π_{LSA} denote the policy trained using the approach presented in Section 5.2.3. As explained in Section 1.3.4, another possible approach is to use RWTAr [105, 141], where the assignment matrix \mathbf{A} has the value $(1 - \epsilon)$ in the minimum elements of each row of $\mathbf{D}_{\mathbf{p}}$, and $\frac{\epsilon}{n_s - 1}$ elsewhere. Similarly, RWTAc [96] uses an assignment matrix \mathbf{A} that has the value $(1 - \epsilon)$ in the minimum elements of each column of $\mathbf{D}_{\mathbf{p}}$, and $\frac{\epsilon}{n_e - 1}$ elsewhere.⁵ A policy trained using these approaches for a given $\epsilon \geq 0$ will be denoted as $\pi_{\text{RWTAr-}\epsilon}$ and $\pi_{\text{RWTAc-}\epsilon}$. Note that WTAr \equiv RWTAr and WTAc \equiv RWTAc when $\epsilon = 0$. We first train 11 policies (π_{LSA} , $\pi_{\text{RWTAr-}\epsilon}$, and $\pi_{\text{RWTAc-}\epsilon}$ for $\epsilon \in \{0, 0.05, 0.15, 0.25, 0.35\}$) using the same training set. For each of the policies, we then evaluate these metrics:

- MSE with respect to the trajectories of the expert.** For each of the trained policies, we obtain the optimal assignment between the trajectories of the expert and the student using the cost matrix $\mathbf{D}_{\mathbf{p}}$ [33]. The trajectories of the student are then ranked according to the position MSE loss with respect to their assigned expert trajectory, and the index of this ranking is denoted as r . For instance, the case $\kappa = 0$ corresponds to the trajectory of the student that best predicts an expert trajectory. The results are shown in Fig. 5-6, where values above 1 represent cases where LSA (our approach) performs better. Compared to RWTAr, our approach achieves an average MSE between 1.09 and 18.02 times smaller. Compared to RWTAc, our approach achieves an average MSE between 2.35 and 2.68 times smaller.
- Number of collision-free trajectories obtained.** Using the same testing scenario as in Section 5.3.1.1 (Fig. 5-4), Fig. 5-7 shows the number of collision-free trajectories produced by each algorithm. Our approach is able to produce

⁵Thus if $\mathbf{D}_{\mathbf{p}} = \begin{bmatrix} 3 & 1 & 5 & 8 \\ 4 & 3 & 2 & 7 \\ 6 & 5 & 6 & 3 \end{bmatrix}$, then RWTAr uses $\mathbf{A} = \begin{bmatrix} \epsilon/3 & 1 - \epsilon & \epsilon/3 & \epsilon/3 \\ \epsilon/3 & \epsilon/3 & 1 - \epsilon & \epsilon/3 \\ \epsilon/3 & \epsilon/3 & \epsilon/3 & 1 - \epsilon \end{bmatrix}$, while RWTAc uses $\mathbf{A} = \begin{bmatrix} 1 - \epsilon & 1 - \epsilon & \epsilon/2 & \epsilon/2 \\ \epsilon/2 & \epsilon/2 & 1 - \epsilon & \epsilon/2 \\ \epsilon/2 & \epsilon/2 & \epsilon/2 & 1 - \epsilon \end{bmatrix}$.

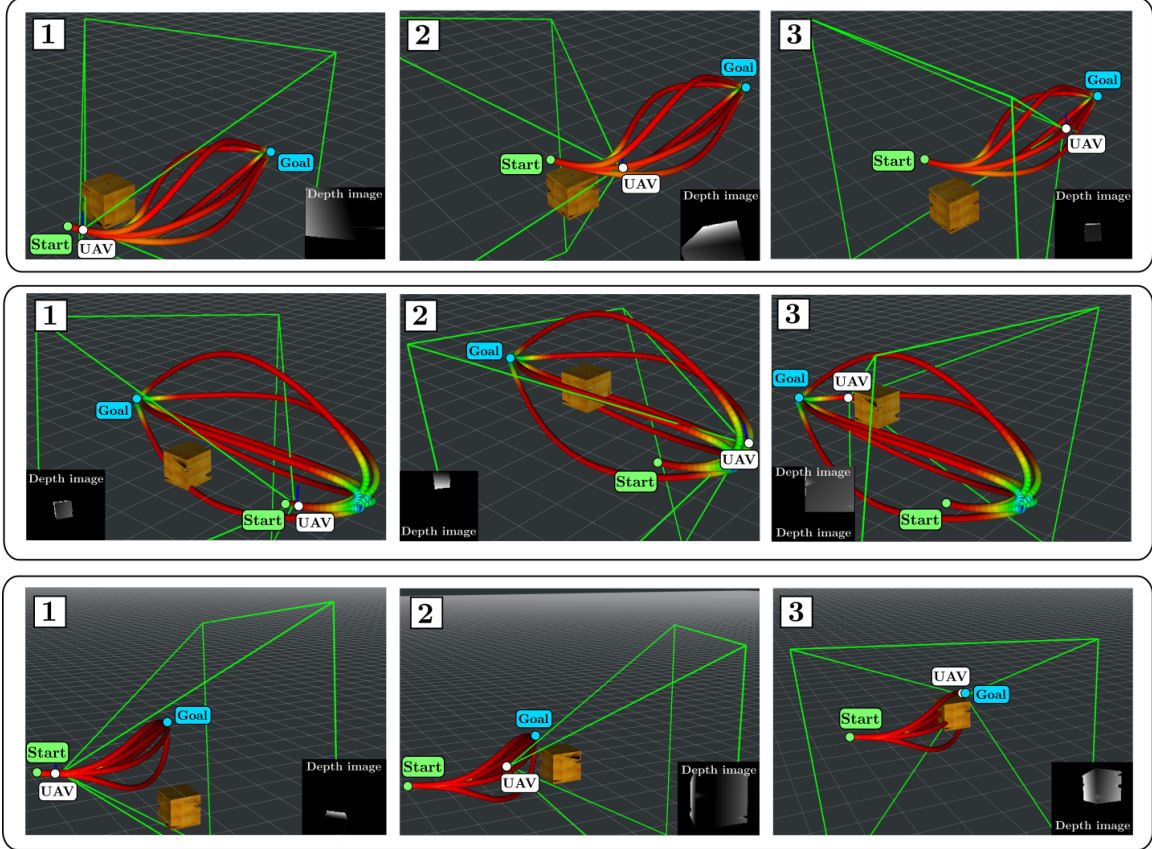


Figure 5-8: Snapshots of the trajectories produced by Deep-PANTHER in a dynamic environment to go from a starting location \bullet to a goal \bullet , together with the depth images of the onboard camera. The green pyramid represents the FOV of the camera, and the colormap represents the velocity (red denotes a higher velocity). Note that the UAV starts with a nonzero initial velocity, since it is replanning as it flies.

at least one collision-free trajectory for all the \mathbf{g}_{term} tested, while RWTAr- ϵ ($\epsilon \in \{0.25, 0.35\}$) and RWTAc- ϵ ($\epsilon \in \{0, 0.05, 0.15, 0.25, 0.35\}$) fail to generate a collision-free trajectory for some of the goals, especially for the ones that are directly behind the obstacle.

5.3.2 Replanning with Dynamic Obstacles

We train the student in an environment that consists of a dynamic obstacle flying a trefoil-knot trajectory [110]. The position, phase, and scale of this trefoil-knot trajectory, together with the terminal goal, are randomized. We use the Dataset Aggregation algorithm (Dagger) [139] to collect the data and train the student.

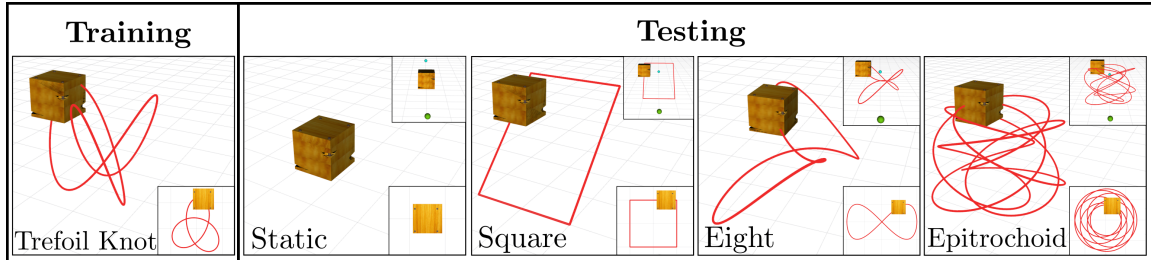


Figure 5-9: Trajectories used to train and test the student. The task for the UAV is to fly back and forth between the two goals ● and ●.

Dagger is an iterative dataset collection and policy training method that helps reduce covariate shift issues by querying actions of the expert while executing a partially trained policy. The total number of (observation, expert action) pairs collected is approximately 23K.

To test this trained policy, we deploy a dynamic obstacle following a trefoil-knot trajectory with a random phase, and manually select random \mathbf{g}_{term} . This makes the UAV replan from different initial positions, velocities, and accelerations, different states of the obstacle, and different goals. Some snapshots of the resulting collision-free trajectories generated by the student, together with the depth image of the onboard camera, are shown in Fig. 5-8. As explained in Section 5.2.5, the collision-free trajectory that has the smallest augmented cost is the one chosen for execution.

5.3.3 Generalization to Other Obstacle Trajectories

To evaluate how well the student in Section 5.3.2 (trained using trefoil-knot obstacle trajectories) generalizes, we test it with different obstacle trajectories: static, square, eight and epitrochoid (see Fig. 5-9). During 45 seconds, the UAV must fly back and forth between two goals separated 10 m, with the trajectory of the obstacle lying between these goals. The number of collision-free trajectories generated is shown in Table 5.2. Despite being trained with a different obstacle trajectory, the policy succeeded in generating at least one collision-free trajectory in all the approximately 740 replanning steps. In all the cases the UAV reached 8 goals during the total simulation time.

Table 5.2: Percentage of collision-free trajectories produced by the student for different obstacle trajectories. The student was trained using only trefoil-knot obstacle trajectories.

	Trefoil	Static	Square	Eight	Epitrochoid
$(n_s)_{\text{coll. free}} = 0$	0%	0%	0%	0%	0%
$(n_s)_{\text{coll. free}} \in \{1, 2, 3\}$	6%	0%	7%	2%	2%
$(n_s)_{\text{coll. free}} \in \{4, 5, 6\}$	94%	100%	93%	98%	98%

5.4 Limitations

Our approach also presents some limitations. On the one hand, the trajectories generated by the student guarantee by construction the smoothness, initial, and final constraints. However, they are not guaranteed to satisfy the dynamic limits or the safety constraints for any given observation. Although some approaches have been proposed to guarantee by construction a linear inequality constraint $\mathbf{Ax} \leq \mathbf{b}$ (e.g., the use of barycentric coordinates [49]), they require the use of the vertex enumeration algorithm, which can be computationally expensive when \mathbf{A} and \mathbf{b} depend on the observation. An additional challenge is how to do backpropagation through this vertex enumeration algorithm. Other methods such as the time scaling (used to satisfy the dynamic limits constraints [89, 96]) are not directly applicable in dynamic environments. Hence, a computationally-efficient method to enforce these constraints is desirable.

On the other hand, when a spline is fit to the ψ trajectory found by Eq. 5.1, the constraints on $\dot{\psi}$ are not taken into account. For very fast moving obstacles or fast planned position trajectories, this approach may lead to violations of constraints on the rate of ψ . At the expense of more computation time, another option is to solve a convex optimization problem that fits a spline taking into account these constraints.

Chapter 6

Conclusions

6.1 Summary of Contributions

Chapter 2 derived and presented the MINVO basis. The key feature of this basis is that it finds the smallest n -simplex that encloses a given n^{th} -degree polynomial curve (Problem 1), and also finds the n^{th} -degree polynomial curve with largest convex hull enclosed in a given n -simplex (Problem 2). For $n = 3$, the ratios of the improvement in the volume achieved by the MINVO basis with respect to the Bernstein and B-Spline bases are 2.36 and 254.9 respectively. When $n = 7$, these improvement ratios increase to 902.7 and $2.997 \cdot 10^{21}$ respectively. Numerical global optimality was proven for $n = 1, 2, 3$, numerical local optimality was proven for $n = 4$, and high-quality feasible solutions for all $n \geq 5$ were obtained. Finally, the MINVO basis was also applied to polynomial curves with different n , k , and m (achieving improvements ratios of up to ≈ 550), and to some rational curves.

Chapter 3 presented MADER, a decentralized and asynchronous planner that handles static obstacles, dynamic obstacles, and other agents. By using the MINVO basis, MADER obtains outer polyhedral representations of the trajectories that are much smaller than the volumes achieved using the Bernstein and B-Spline bases. To ensure nonconservative, collision-free constraints with respect to other obstacles and agents, MADER includes as decision variables the planes that separate each pair of outer polyhedral representations. Safety with respect to other agents is guaranteed

in a decentralized and asynchronous way by including their committed trajectories as constraints in the optimization and then executing a collision check-recheck scheme. Extensive simulations in dynamic multiagent environments have highlighted the improvements of MADER with respect to other state-of-the-art algorithms in terms of number of stops, computation/execution time, and flight distance.

Chapter 4 presented PANTHER, a perception-aware trajectory planner in dynamic environments. PANTHER is able to couple together the translation and the full rotation in the optimization, leading to perception-aware trajectories computed in real time that maximize the presence of the obstacles in the FOV while minimizing their projected velocity. Extensive hardware experiments in unknown dynamic environments, with all the computation running onboard, and with relative velocities of up to 6.3 m/s have shown its effectiveness.

Finally, Chapter 5 presented Deep-PANTHER, a learning-based perception-aware trajectory planner in dynamic environments. Deep-PANTHER is able to achieve a similar cost as the optimization-based expert, while having a computation time two orders of magnitude faster. The multimodality of the problem is captured by the design of a loss function that assigns a distinct student trajectory to each expert trajectory. This leads to MSE losses with respect to the expert up to 18 times smaller than the (Relaxed) Winner-Takes-All approaches. Deep-PANTHER also performs well in environments where the obstacle follows a different trajectory than the one used in training.

6.2 Future Work

The exciting results of Chapter 2 naturally lead to the following questions and conjectures, that we leave as future work:

- Is the global optimum of Problem 4 the same as the global optimum of Problem 3 (Section 2.4.4)? I.e., are we losing optimality by imposing the specific structure on $\lambda_i(t)$? On a similar note, is it possible to obtain for any n a bound on the distance between the objective value obtained by the model proposed in

Section 2.5.2, and the global minimum of Problem 3?

- Does there exist a recursive formula to obtain the solution of Problem 3 for a specific $n = q$ given the previous solutions for $n = 1, \dots, q - 1$? Would this recursive formula allow to obtain the *globally* optimal solutions for all $n \in \mathbb{N}$ of Problem 3?

Moreover, the way polynomials are scaled (to impose $\mathbf{A}^T \mathbf{1} = \mathbf{e}$) in Section 2.5.2 can suffer from numerical instabilities when the degree is very high ($n > 30$). This is expected, since the monomial basis used to compute \mathbf{A} is known to be numerically unstable [17]. A more numerically-stable scaling, potentially avoiding the use of the monomial basis, could therefore be beneficial for higher degrees.

Chapters 3 (MADER) and 4 (PANTHER) also point to some exciting directions for future work. The first one is the relaxation of some of the assumptions of Section 3.3, especially the assumption that there is no delay in the communication between the agents. Moreover, it would also be interesting to include the perception-aware component (PANTHER) into the multiagent planning framework (MADER). This would allow the agents equipped with limited FOV sensors to detect other agents and/or obstacles. Related to this, another potential future work is to study how a team of UAVs can collaborate to keep track of all the dynamic obstacles in the environment, sharing the local knowledge of the location of the obstacles while performing obstacle avoidance.

Related to the previous point, another direction of future work is how to optimally solve the trade-off between exploration and exploitation: when should the UAV include a specific (already tracked) obstacle in the PA term of the optimization, in order to predict its trajectory more accurately to be able to avoid it, and when should the UAV turn around to explore unknown space? Too much focus on exploitation may lead to collision with obstacles that were never detected, and too much focus on exploration may lead to a very poor trajectory prediction, and hence to a collision as well.

With respect to Chapter 5, future work includes the extension to multiple

dynamic obstacles, and the inclusion of the camera images directly in the observation. Moreover, we also plan to work on the limitations explained in Section 5.4 and study how to take into account the multimodality of the predicted future trajectory of the obstacle(s).

Finally, another interesting research direction of this thesis is how to incorporate disturbances in the planning problem, while still guaranteeing that the tracking error of the UAV remains bounded [94]. The incorporation of such disturbance information is especially important when flying outdoors under windy conditions, since a large deviation between the planned trajectory and the actual trajectory can provoke a collision with the obstacles.

Bibliography

- [1] Ipopt documentation. <https://coin-or.github.io/Ipopt/OUTPUT.html>. Accessed on 01/20/2022.
- [2] Qualcomm machine vision SDK. <https://developer.qualcomm.com/software/machine-vision-sdk>. Accessed on 02/15/2021.
- [3] GLPK: GNU Linear Programming Kit. <https://www.gnu.org/software/glpk/>, 2020.
- [4] MATLAB optimization toolbox, 2020. The MathWorks, Natick, MA, USA.
- [5] Markus W Achtelik, Simon Lynen, Stephan Weiss, Margarita Chli, and Roland Siegwart. Motion-and uncertainty-aware path planning for micro aerial vehicles. *Journal of Field Robotics*, 31(4):676–698, 2014.
- [6] Ali Agha, Kyohei Otsu, Benjamin Morrell, David D Fan, Rohan Thakker, Angel Santamaria-Navarro, Sung-Kyun Kim, Amanda Bouman, Xianmei Lei, Jeffrey Edlund, et al. Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge. *arXiv preprint arXiv:2103.11470*, 2021.
- [7] Fatema Ahmed Sadeq and Shatha Assaad Salman. *On the volume of a polytope in \mathbb{R}^n : Basics, Concepts, Methods*. Lambert Academic Publishing, 2012.
- [8] Ross E Allen and Marco Pavone. A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance. *Robotics and Autonomous Systems*, 115:174–193, 2019.
- [9] 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.
- [10] Federico Augugliaro, Angela P Schoellig, and Raffaello D’Andrea. Generation of collision-free trajectories for a quadcopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 1917–1922. IEEE, 2012.

- [11] Luca Bartolomei, Lucas Teixeira, and Margarita Chli. Perception-aware path planning for UAVs using semantic segmentation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5808–5815. IEEE, 2020.
- [12] Ivar Bendixson. Sur les courbes définies par des équations différentielles. *Acta Mathematica*, 24(1):1, 1901.
- [13] Marcel Berger. *Geometry I*. Springer Science & Business Media, 2009.
- [14] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016.
- [15] Ernesto G Birgin and José Mario Martínez. Improving ultimate convergence of an augmented lagrangian method. *Optimization Methods and Software*, 23(2):177–195, 2008.
- [16] Christopher M Bishop. Mixture density networks. 1994.
- [17] Grigoriy Blekherman, Pablo A Parrilo, and Rekha R Thomas. *Semidefinite optimization and convex algebraic geometry*. SIAM, 2012.
- [18] Rogerio Bonatti, Yanfu Zhang, Sanjiban Choudhury, Wenshan Wang, and Sebastian Scherer. Autonomous drone cinematographer: Using artistic principles to create smooth, safe, occlusion-free trajectories for aerial filming. In *International Symposium on Experimental Robotics*, pages 119–129. Springer, 2018.
- [19] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [20] Luitzen Egbertus Jan Brouwer. Über abbildung von mannigfaltigkeiten. *Mathematische annalen*, 71(1):97–115, 1911.
- [21] Nathan Bucki, Junseok Lee, and Mark W Mueller. Rectangular pyramid partitioning using integrated depth sensors (RAPPIDS): A fast planner for multicopter navigation. *IEEE Robotics and Automation Letters*, 5(3):4626–4633, 2020.
- [22] Constantin Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.
- [23] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

- [24] Adrian Carrio, Jesus Tordesillas, Sai Vemprala, Srikanth Saripalli, Pascual Campoy, and Jonathan P How. Onboard detection and localization of drones using depth maps. *IEEE Access*, 8:30480–30490, 2020.
- [25] Gang Chen, Wei Dong, Xinjun Sheng, Xiangyang Zhu, and Han Ding. An active sense and avoid system for flying robots in dynamic environments. *IEEE/ASME Transactions on Mechatronics*, 26(2):668–678, 2021.
- [26] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1476–1483. IEEE, 2016.
- [27] Jing Chen and Shaojie Shen. Using a quadrotor to track a moving target with arbitrary relative motion patterns. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5310–5317. IEEE, 2017.
- [28] Yufan Chen, Mark Cutler, and Jonathan P How. Decoupled multiagent path planning via incremental sequential convex programming. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5954–5961. IEEE, 2015.
- [29] Daniel Ciripoi, Nidhi Kaihnsa, Andreas Löhne, and Bernd Sturmfels. Computing convex hulls of trajectories. *arXiv preprint arXiv:1810.03547*, 2018.
- [30] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.
- [31] Andrew R Conn, Nicholas IM Gould, and Philippe Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.
- [32] Gabriele Costante, Christian Forster, Jeffrey Delmerico, Paolo Valigi, and Davide Scaramuzza. Perception-aware path planning. *arXiv preprint arXiv:1605.04151*, 2016.
- [33] David F Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.
- [34] Carl De Boor. *A practical guide to splines*, volume 27. Springer, 1978.
- [35] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for UAVs in cluttered environments. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 42–49. IEEE, 2015.
- [36] Douglas Derry. Convex hulls of simple space curves. *Canadian Journal of Mathematics*, 8:383–388, 1956.

- [37] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [38] Wenchao Ding, Wenliang Gao, Kaixuan Wang, and Shaojie Shen. An efficient B-spline-based kinodynamic replanning framework for quadrotors. *IEEE Transactions on Robotics*, 35(6):1287–1306, 2019.
- [39] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.
- [40] Christer Ericson. *Real-time collision detection*. CRC Press, 2004.
- [41] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robotics and Automation Letters*, 3(2):620–626, 2017.
- [42] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. PAMPC: Perception-aware model predictive control for quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.
- [43] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40), 2020.
- [44] Davide Falanga, Elias Mueggler, Matthias Faessler, and Davide Scaramuzza. Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 5774–5781. IEEE, 2017.
- [45] Michael Firman, Neill DF Campbell, Lourdes Agapito, and Gabriel J Brostow. Diversenet: When one right answer is not enough. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5598–5607, 2018.
- [46] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.
- [47] Peter R Florence, John Carter, Jake Ware, and Russ Tedrake. Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3D data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7631–7638. IEEE, 2018.
- [48] Simon Flöry. Fitting curves and surfaces to point clouds in the presence of obstacles. *Computer Aided Geometric Design*, 26(2):192–202, February 2009.

- [49] Thomas Frerix, Matthias Nießner, and Daniel Cremers. Homogeneous linear inequality constraints for neural network activations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 748–749, 2020.
- [50] Kristoffer M Frey, Ted J Steiner, and Jonathan P How. Towards online observability-aware trajectory optimization for landmark-based estimators. *arXiv preprint arXiv:1908.03790*, 2019.
- [51] Steven G. Johnson. The NLOpt nonlinear-optimization package. <http://github.com/stevengj/nlopt>, 2020.
- [52] Daniel Galicer, Mariano Merzbacher, and Damián Pinasco. The minimal volume of simplices containing a convex body. *The Journal of Geometric Analysis*, 29(1):717–732, 2019.
- [53] Fei Gao and Shaojie Shen. Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex qcqp. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6354–6361. IEEE, 2017.
- [54] Fei Gao, Luqi Wang, Boyu Zhou, Xin Zhou, Jie Pan, and Shaojie Shen. Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments. *IEEE Transactions on Robotics*, 36(5):1526–1545, 2020.
- [55] Fei Gao, William Wu, Wenliang Gao, and Shaojie Shen. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*, 36(4):710–733, 2019.
- [56] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 47:99–131, 2005.
- [57] Philip E. Gill, Walter Murray, Michael A. Saunders, and Elizabeth Wong. User’s guide for SNOPT 7.7: Software for large-scale nonlinear programming. Technical report, Department of Mathematics, University of California, San Diego, 2018.
- [58] Abner Guzman-Rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. *Advances in neural information processing systems*, 25, 2012.
- [59] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [60] Eligius MT Hendrix, Inmaculada García, Javier Plaza, and Antonio Plaza. On the minimum volume simplex enclosure problem for estimating a linear mixing model. *Journal of Global Optimization*, 56(3):957–970, 2013.

- [61] Gary Herron. Polynomial bases for quadratic and cubic polynomials which yield control points with small convex hulls. *Computer aided geometric design*, 6(1):1–9, 1989.
- [62] Kai Hormann and N Sukumar. *Generalized barycentric coordinates in computer graphics and computational mechanics*. CRC press, 2017.
- [63] HSL(2013). A collection of Fortran codes for large scale scientific computation. <http://www.hsl.rl.ac.uk>.
- [64] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018. <https://otexts.com/fpp2/>.
- [65] Marian-Daniel Iordache, José Bioucas-Dias, and Antonio Plaza. Unmixing sparse hyperspectral mixtures. In *2009 IEEE International Geoscience and Remote Sensing Symposium*, volume 4, pages IV–85. IEEE, 2009.
- [66] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte Carlo motion planning for robot trajectory optimization under uncertainty. In *Robotics Research*, pages 343–361. Springer, 2018.
- [67] Boseong Felipe Jeon and H Jin Kim. Online trajectory generation of a MAV for chasing a moving target in 3D dense environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1115–1121. IEEE, 2019.
- [68] Boseong Felipe Jeon, Dongsuk Shim, and H Jin Kim. Detection-aware trajectory generation for a drone cinematographer. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1450–1457. IEEE, 2020.
- [69] Atsushi Kanazawa. On the minimal volume of simplices enclosing a convex body. *Archiv der Mathematik*, 102(5):489–492, 2014.
- [70] Menelaos I Karavelas, Panagiotis D Kaklis, and Konstantinos V Kostas. Bounding the distance between 2d parametric Bézier curves and their control polygon. *Computing*, 72(1-2):117–128, 2004.
- [71] Samuel Karlin and Lloyd S Shapley. *Geometry of moment spaces*. Number 12. American Mathematical Soc., 1953.
- [72] William Karush. Minima of functions of several variables with inequalities as side constraints. *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*, 1939.
- [73] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Proceedings of Robotics: Science and Systems*, 2020.

- [74] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.
- [75] Victor Klee. Facet-centroids and volume minimization. *Studia Scientiarum Mathematicarum Hungarica*, 21:143–147, 1986.
- [76] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A big CAD model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [77] Mark Grigorevich Krein, D Louvish, et al. *The Markov moment problem and extremal problems*. American Mathematical Society, 1977.
- [78] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [79] HW Kuhn and AW Tucker. Nonlinear programming. *University of California Press*, 13:54, 1951.
- [80] Alex Kushleyev, Daniel Mellinger, Caitlin Powers, and Vijay Kumar. Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300, 2013.
- [81] József Kuti, Péter Galambos, and Péter Baranyi. Minimal volume simplex (MVS) approach for convex hull generation in TP model transformation. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 187–192. IEEE, 2014.
- [82] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.
- [83] Keuntaek Lee, Jason Gibson, and Evangelos A Theodorou. Aggressive perception-aware navigation using deep optical flow dynamics and PixelMPC. *IEEE Robotics and Automation Letters*, 5(2):1207–1214, 2020.
- [84] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010.
- [85] Guanrui Li, Alex Tunchez, and Giuseppe Loianno. PCMPC: Perception-constrained model predictive control for quadrotors with suspended loads using a single camera and IMU. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2012–2018. IEEE, 2021.
- [86] Jun Li and José M Bioucas-Dias. Minimum volume simplex analysis: A fast algorithm to unmix hyperspectral data. In *IGARSS 2008-2008 IEEE International Geoscience and Remote Sensing Symposium*, volume 3, pages III–250. IEEE, 2008.

- [87] Jiahao Lin, Hai Zhu, and Javier Alonso-Mora. Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2682–2688. IEEE, 2020.
- [88] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Towards search-based motion planning for micro aerial vehicles. *arXiv preprint arXiv:1810.03071*, 2018.
- [89] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.
- [90] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
- [91] Johan Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, 54(5):1007–1011, 2009.
- [92] Brett T Lopez and Jonathan P How. Aggressive 3-D collision avoidance for high-speed navigation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5759–5765. IEEE, 2017.
- [93] Brett T Lopez and Jonathan P How. Aggressive collision avoidance with limited field-of-view sensing. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 1358–1365. IEEE, 2017.
- [94] Brett T Lopez, Jean-Jacques E Slotine, and Jonathan P How. Dynamic tube MPC for nonlinear systems. In *2019 American Control Conference (ACC)*, pages 1655–1662. IEEE, 2019.
- [95] Brett Thomas Lopez. Low-latency trajectory planning for high-speed navigation in unknown environments. Master’s thesis, Massachusetts Institute of Technology, 2016.
- [96] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.
- [97] Carlos E Luis and Angela P Schoellig. Trajectory generation for multiagent point-to-point transitions via distributed model predictive control. *IEEE Robotics and Automation Letters*, 4(2):375–382, 2019.
- [98] David Lutterkort and Jörg Peters. Tight linear bounds on the distance between a spline and its B-spline control polygon. 1999.

- [99] David Lutterkort and Jörg Peters. Optimized refinable enclosures of multivariate polynomial pieces. *Computer Aided Geometric Design*, 18(9):851–863, 2001.
- [100] David Lutterkort and Jörg Peters. Tight linear envelopes for splines. *Numerische Mathematik*, 89(4):735–748, 2001.
- [101] David W Lyons. An elementary introduction to the Hopf fibration. *Mathematics magazine*, 76(2):87–98, 2003.
- [102] Johan Löfberg. Envelope approximations for global optimization. <https://yalp.github.io/tutorial/envelopesinbmbnb/>, August 2020.
- [103] Weiyin Ma and Renjiang Zhang. Efficient piecewise linear approximation of Bézier curves with improved sharp error bound. In *International Conference on Geometric Modeling and Processing*, pages 157–174. Springer, 2006.
- [104] Xiaobai Ma, Ziyuan Jiao, Zhenkai Wang, and Dimitra Panagou. Decentralized prioritized motion planning for multiple autonomous UAVs in 3D polygonal obstacle environments. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 292–300. IEEE, 2016.
- [105] Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7144–7153, 2019.
- [106] MathWorks. Area of polyshape. <https://www.mathworks.com/help/matlab/ref/polyshape.area.html>. (Accessed on 04/10/2022).
- [107] MathWorks. Convex hull. <https://www.mathworks.com/help/matlab/ref/convhull.html>. (Accessed on 04/10/2022).
- [108] MathWorks. Normal inverse cumulative distribution function. <https://www.mathworks.com/help/stats/norminv.html>. (Accessed on 02/22/2021).
- [109] Wolfram MathWorld. Spherical shell. <https://mathworld.wolfram.com/SphericalShell.html>. (Accessed on 01/24/2022).
- [110] Wolfram MathWorld. Trefoil knot. <https://mathworld.wolfram.com/TrefoilKnot.html>. (Accessed on 06/02/2019).
- [111] Wolfram MathWorld. Sigmoid function. <https://mathworld.wolfram.com/SigmoidFunction.html>, 2020. (Accessed on 02/20/2021).
- [112] Wolfram MathWorld. Simpson’s rule. <https://mathworld.wolfram.com/SimpsonsRule.html>, 2020. (Accessed on 02/17/2021).
- [113] Kostyantyn Mazur. Convex hull of (t, t^2, \dots, t^N) . *arXiv preprint arXiv:1706.02060*, 2017.

- [114] Colin McCann. Exploring properties of high-degree SLEFEs. 2004.
- [115] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.
- [116] Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE international conference on robotics and automation*, pages 477–483. IEEE, 2012.
- [117] Daniel Morgan, Giri P Subramanian, Soon-Jo Chung, and Fred Y Hadaegh. Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming. *The International Journal of Robotics Research*, 35(10):1261–1285, 2016.
- [118] Benjamin Morrell, Marc Rigter, Gene Merewether, Robert Reid, Rohan Thakker, Theodore Tzanetos, Vinay Rajur, and Gregory Chamitoff. Differential flatness transformations for aggressive quadrotor flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5204–5210. IEEE, 2018.
- [119] Varun Murali, Igor Spasojevic, Winter Guerra, and Sertac Karaman. Perception-aware trajectory generation for aggressive quadrotor flight using differential flatness. In *2019 American Control Conference (ACC)*, pages 3936–3943. IEEE, 2019.
- [120] Ashish Myles and Jörg Peters. Threading splines through 3D channels. *Computer-Aided Design*, 37(2):139–148, 2005.
- [121] Tobias Nägeli, Javier Alonso-Mora, Alexander Domahidi, Daniela Rus, and Otmar Hilliges. Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters*, 2(3):1696–1703, 2017.
- [122] David Nairn, Jörg Peters, and David Lutterkort. Sharp, quantitative bounds on the distance between a polynomial piece and its Bézier control polygon. *Computer Aided Geometric Design*, 16(7):613–631, 1999.
- [123] Sriram Narayanan, Ramin Moslemi, Francesco Pittaluga, Buyu Liu, and Manmohan Chandraker. Divide-and-conquer for lane-aware diverse trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15799–15808, 2021.
- [124] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online UAV replanning. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5332–5339. IEEE, 2016.

- [125] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A Theodorou, and Byron Boots. Imitation learning for agile autonomous driving. *The International Journal of Robotics Research*, 39(2-3):286–302, 2020.
- [126] Jungwon Park, Junha Kim, Inkyu Jang, and H Jin Kim. Efficient multi-agent trajectory planning with feasibility guarantee using relative Bernstein polynomial. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 434–440. IEEE, 2020.
- [127] Bryan Penin, Paolo Robuffo Giordano, and François Chaumette. Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions. *IEEE Robotics and Automation Letters*, 3(4):3725–3732, 2018.
- [128] Bryan Penin, Riccardo Spica, Paolo Robuffo Giordano, and François Chaumette. Vision-based minimum-time trajectory generation for a quadrotor UAV. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6199–6206. IEEE, 2017.
- [129] Jörg Peters and Xiaobin Wu. SLEVEs for planar spline curves. *Computer Aided Geometric Design*, 21(6):615–635, 2004.
- [130] Nikhil D Potdar, Guido CHE de Croon, and Javier Alonso-Mora. Online trajectory planning and control of a MAV payload system in dynamic environments. *Autonomous Robots*, pages 1–25, 2020.
- [131] James A Preiss, Karol Hausman, Gaurav S Sukhatme, and Stephan Weiss. Trajectory optimization for self-calibration and navigation. In *Robotics: Science and Systems*, 2017.
- [132] James A Preiss, Karol Hausman, Gaurav S Sukhatme, and Stephan Weiss. Simultaneous self-calibration and navigation using trajectory optimization. *The International Journal of Robotics Research*, 37(13-14):1573–1594, 2018.
- [133] Kaihuai Qin. General matrix representations for B-splines. *The Visual Computer*, 16(3-4):177–186, 2000.
- [134] Kristian Ranestad and Bernd Sturmfels. On the convex hull of a space curve. *Advances in Geometry*, 12(1):157–178, 2012.
- [135] Alexander Reske, Jan Carius, Yuntao Ma, Farbod Farshidian, and Marco Hutter. Imitation learning from MPC for quadrupedal multi-gait control. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5014–5020. IEEE, 2021.
- [136] D Reed Robinson, Robert T Mar, Katia Estabridis, and Gary Hewer. An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments. *IEEE Robotics and Automation Letters*, 3(2):1215–1222, 2018.

- [137] Derek M Rogge, Benoit Rivard, Jinkai Zhang, and Jilu Feng. Iterative spectral unmixing for optimizing per-pixel endmember sets. *IEEE Transactions on Geoscience and Remote Sensing*, 44(12):3725–3736, 2006.
- [138] Tae Roh and Lieven Vandenbergh. Discrete transforms, semidefinite programming, and sum-of-squares representations of nonnegative polynomials. *SIAM Journal on Optimization*, 16(4):939–964, 2006.
- [139] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- [140] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.
- [141] Christian Rupprecht, Iro Laina, Robert DiPietro, Maximilian Baust, Federico Tombari, Nassir Navab, and Gregory D Hager. Learning in an uncertain world: Representing ambiguity through multiple hypotheses. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3591–3600, 2017.
- [142] Paolo Salaris, Marco Cagnetti, Riccardo Spica, and Paolo Robuffo Giordano. Online optimal perception-aware trajectory generation. *IEEE Transactions on Robotics*, 35(6):1307–1322, 2019.
- [143] Nitin J Sanket, Chethan M Parameshwara, Chahat Deep Singh, Ashwin V Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. EVDodgeNet: Deep dynamic obstacle dodging with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10651–10657. IEEE, 2020.
- [144] M Sayrafiezadeh. An inductive proof for extremal simplexes. *Mathematics Magazine*, 65(4):252–255, 1992.
- [145] Isaac J Schoenberg. *Cardinal spline interpolation*. SIAM, 1973.
- [146] Vyacheslav D Sedykh. Structure of the convex hull of a space curve. *Journal of Soviet Mathematics*, 33(4):1140–1153, 1986.
- [147] Jevgenija Selinger and Lars Linsen. Efficient curvature-optimized G2-continuous path generation with guaranteed error bound for 3-axis machining. In *2011 15th International Conference on Information Visualisation*, pages 519–527. IEEE, 2011.

- [148] Igor Spasojevic, Varun Murali, and Sertac Karaman. Joint feature selection and time optimal path parametrization for high speed vision-aided navigation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5931–5938. IEEE, 2020.
- [149] Igor Spasojevic, Varun Murali, and Sertac Karaman. Perception-aware time optimal path parameterization for quadrotors. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3213–3219. IEEE, 2020.
- [150] Riccardo Spica, Paolo Robuffo Giordano, Markus Ryll, Heinrich H Bühlhoff, and Antonio Franchi. An open-source hardware/software architecture for quadrotor UAVs. *IFAC Proceedings Volumes*, 46(30):198–205, 2013.
- [151] Ernst Steinitz. Bedingt konvergente Reihen und konvexe Systeme. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1913(143):128–176, 1913.
- [152] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM journal on optimization*, 12(2):555–573, 2002.
- [153] Gábor Szegő. *Orthogonal polynomials*, volume 23. American Mathematical Society, 1939.
- [154] Michael Szmuk, Carlo Alberto Pascucci, and Behçet AÇikmeşe. Real-time quad-rotor path planning for mobile obstacle avoidance using convex optimization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.
- [155] Andrea Tagliabue, Dong-Ki Kim, Michael Everett, and Jonathan P How. Efficient guided policy search via imitation of robust tube MPC. *arXiv preprint arXiv:2109.09910*, 2021.
- [156] Andrea Tagliabue, Jesus Tordesillas, Xiaoyi Cai, Angel Angel Santamaria-Navarro, Jonathan P. How, Luca Carlone, and Ali-akbar Agha-Mohammadi. LION: lidar-inertial observability-aware navigator for vision-denied environments. *International Symposium on Experimental Robotics*, 2020.
- [157] Lvbang Tang, Hesheng Wang, Peng Li, and Yong Wang. Real-time trajectory generation for quadrotors using B-spline based non-uniform kinodynamic search. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1133–1138. IEEE, 2019.
- [158] Rohan Thakker, Nikhilesh Alatur, David D. Fan, Jesus Tordesillas, Michael Paton, Kyohei Otsu, and Ali-akbar Agha-mohammadi. Autonomous off-road navigation over extreme terrains with perceptually-challenging conditions. In *ISER*, 2020.

- [159] Justin Thomas, Jake Welde, Giuseppe Loianno, Kostas Daniilidis, and Vijay Kumar. Autonomous flight for detection, localization, and tracking of moving targets with a small quadrotor. *IEEE Robotics and Automation Letters*, 2(3):1762–1769, 2017.
- [160] Jesus Tordesillas and Jonathan P How. MADER: Trajectory planner in multiagent and dynamic environments. *IEEE Transactions on Robotics*, 2021.
- [161] Jesus Tordesillas and Jonathan P How. Deep-PANTHER: Learning-based perception-aware trajectory planner in dynamic environments. *IEEE Robotics and Automation Letters (in review)*, 2022.
- [162] Jesus Tordesillas and Jonathan P How. MINVO basis: Finding simplexes with minimum volume enclosing polynomial curves. *Computer-Aided Design*, 2022.
- [163] Jesus Tordesillas and Jonathan P How. PANTHER: Perception-aware trajectory planner in dynamic environments. *IEEE Access*, 10:22662–22677, 2022.
- [164] Jesus Tordesillas, Brett T Lopez, John Carter, John Ware, and Jonathan P How. Real-time planning with multi-fidelity models for agile flights in unknown environments. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.
- [165] Jesus Tordesillas, Brett T Lopez, John Carter, John Ware, and Jonathan P How. Real-time planning with multi-fidelity models for agile flights in unknown environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 725–731. IEEE, 2019.
- [166] Jesus Tordesillas, Brett T Lopez, Michael Everett, and Jonathan P How. FASTER: Fast and safe trajectory planner for navigation in unknown environments. *IEEE Transactions on Robotics*, 2021.
- [167] Jesus Tordesillas, Brett T Lopez, and Jonathan P How. FASTER: Fast and safe trajectory planner for flights in unknown environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.
- [168] Jesus Tordesillas Torres. Trajectory planner for agile flights in unknown environments. Master’s thesis, Massachusetts Institute of Technology, 2019.
- [169] Tatsumi Uezato, Mathieu Fauvel, and Nicolas Dobigeon. Hyperspectral unmixing with spectral variability using adaptive bundles and double sparsity. *IEEE Transactions on Geoscience and Remote Sensing*, 57(6):3980–3992, 2019.
- [170] Gert Vegter and Chee Yapy. Finding minimal circumscribing simplexes part 1: Classifying local minima. 1993.

- [171] Miguel Velez-Reyes, Angela Puetz, Michael P Hoke, Ronald B Lockwood, and Samuel Rosario. Iterative algorithms for unmixing of hyperspectral imagery. In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery IX*, volume 5093, pages 418–429. International Society for Optics and Photonics, 2003.
- [172] 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, 2006.
- [173] Yingjian Wang, Jialin Ji, Qianhao Wang, Chao Xu, and Fei Gao. Autonomous flights in dynamic environments with onboard vision. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.
- [174] Michael Watterson and Vijay Kumar. Control of quadrotors using the Hopf fibration on $SO(3)$. In *Robotics Research*, pages 199–215. Springer, 2020.
- [175] Michael Watterson, Sikang Liu, Ke Sun, Trey Smith, and Vijay Kumar. Trajectory optimization on manifolds with applications to quadrotor systems. *The International Journal of Robotics Research*, 39(2-3):303–320, 2020.
- [176] Michael Watterson, Ahmed Zahra, and Vijay Kumar. Geometric control and trajectory optimization for bidirectional thrust quadrotors. In *International Symposium on Experimental Robotics*, pages 165–176. Springer, 2018.
- [177] Jake Welde, James Paulos, and Vijay Kumar. Dynamically feasible task space planning for underactuated aerial manipulators. *IEEE Robotics and Automation Letters*, 6(2):3232–3239, 2021.
- [178] William Wu, Fei Gao, Luqi Wang, Boyu Zhou, and Shaojie Shen. *Temporal Scheduling and Optimization for Multi-MAV Planning*. PhD thesis, Hong Kong University of Science and Technology, 2019.
- [179] Xiaobin Wu and Jörg Peters. The SubLiME (subdividable linear maximum-norm enclosure) package. <http://www.cise.ufl.edu/research/Surflab/download/SubLiME.tar.gz>.
- [180] Yihong Xu, Aljosa Osep, Yutong Ban, Radu Horaud, Laura Leal-Taixé, and Xavier Alameda-Pineda. How to train your deep multi-object tracker. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6787–6796, 2020.
- [181] Zichao Zhang and Davide Scaramuzza. Perception-aware receding horizon navigation for MAVs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2534–2541. IEEE, 2018.

- [182] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.
- [183] Boyu Zhou, Jie Pan, Fei Gao, and Shaojie Shen. RAPTOR: Robust and perception-aware trajectory replanning for quadrotor fast flight. *IEEE Transactions on Robotics*, 2021.
- [184] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao. EGO-planner: An ESDF-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*, 2020.
- [185] Yunhong Zhou and Subhash Suri. Algorithms for a minimum volume enclosing simplex in three dimensions. *SIAM Journal on Computing*, 31(5):1339–1357, 2002.
- [186] Hai Zhu and Javier Alonso-Mora. Chance-constrained collision avoidance for MAVs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.
- [187] Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.