UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Distributed Trajectory Generation for Multiple Autonomous Vehicles Using Bernstein Polynomial-based Methods

Bahareh Sabetghadam

Supervisor: Doctor Rita Maria Mendes de Almeida Correia da Cunha

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass with Distinction

2023

TÉCNICO LISBOA

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Distributed Trajectory Generation for Multiple Autonomous Vehicles Using Bernstein Polynomial-based Methods

Bahareh Sabetghadam

Supervisor: Doctor Rita Maria Mendes de Almeida Correia da Cunha

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass with Distinction

Jury

Chairman:	Doctor Paulo Jorge Coelho Ramalho Oliveira Instituto superior Técnico da Universidade de Lisboa
Members of the committee:	Doctor Isaac Kaminer Naval Postgraduate School, USA
	Doctor António Pedro Rodriguez Aguir Faculdade de Engenharia de Universidade do Porto
	Doctor António Manuel dos Santos Pascoal Instituto superior Técnico da Universidade de Lisboa
	Doctor João Manuel de Freitas Xavier Instituto superior Técnico da Universidade de Lisboa
	Doctor Rita Maria Mendes de Almeida Correia da Cunha Instituto superior Técnico da Universidade de Lisboa

2023

Abstract

This thesis presents an efficient and reliable trajectory generation framework that enables the computation of feasible, safe and collision-free trajectories for teams of autonomous vehicles while satisfying timing constraints imposed by real-time applications. The proposed framework aims to address a few key issues related to multi-vehicle trajectory generation:

(i) Satisfying safety and collision-avoidance constraints at all time instances during a mission; Majority of the existing trajectory generation methods rely on discretization in time or space and, thus, ensuring that collision avoidance constraints are satisfied in between discretization nodes is only possible by employing a fine enough discretization grid. This, however, usually translates into a significantly large number of constraints that can hinder the application of these methods to real-time multi-vehicle trajectory generation scenarios. To avoid the complications associated with time gridding, we parameterize trajectories with Bézier curves and propose the Bernstein relaxation and refinement method to guarantee that collision avoidance requirements are satisfied at any time instant, even for extended travel times. The proposed method exploits the unique properties of Bézier curves to replace the infinitely many constraints in the problem with a finite set of constraints. The proposed method also enables a flexible trade-off between conservatism of the resulting set of constraints and computational complexity.

(ii) Considering the rotational motion of a drone to avoid infeasibility problems with flights in tight spaces; To incorporate the rotational motion of a drone into the trajectory generation problem, in contrary to the commonly used sphere model, we approximate the drone body as an ellipsoid whose principal axes are aligned with the body frame axes. Although the symmetrical approximation of the vehicle's body (with a disc or a sphere in 2D and 3D planning) results in straightforward construction of the configuration space, it is very conservative and fails to validate trajectories that are feasible upon considering the vehicle's orientation. The ellipsoid model instead allows an explicit consideration of the drone's shape and orientation which is necessary for trajectory generation in unstructured environments with narrow gaps and small spaces between obstacles. In order to derive constraints for collision avoidance between ellipsoid-shaped bodies we use the separating hyperplane theorem of convex sets. The resulting set of constraints is seamlessly integrated into the proposed Bernstein-based trajectory generation method.

(iii) Decoupling inter-vehicle collision-avoidance constraints for a synchronous distributed scheme with low computation and communication demands; To alleviate the computational complexity of solving the problem centrally for a large group of vehicles, we present a scheme that enables local distributed trajectory generation by solving a small-scale optimization problem that only involves a vehicle's individual variables. To ensure that local decisions

satisfy the coupling collision avoidance constraints we adopt the Voronoi partitioning of space and enforce each vehicle to generate its trajectory inside (a subset of) its Voronoi cell towards the closest point (in the cell) to its goal position. A sequence of sub-problems are then solved in a receding-horizon manner, using only the relative position information exchanged between the neighboring agents, until the vehicles reach their goal positions. The set of local collision-avoidance constraints is derived taking into account the vehicle's orientation in order to avoid infeasiblity issues with generating trajectories in tight spaces for hundreds of drones. Also, the resulting set of constraints can be evaluated with the proposed Bernstein relaxation method to ensure that safe separation criteria between trajectories are met at any time instant of the planning horizon.

The thesis concludes with extensive simulation and experimental results, with up to 100 drone, to illustrate the efficacy of the proposed trajectory generation framework.

Keywords: Trajectory Generation, Multi-vehicle Applications, Bézier Curve, Ellipsoids, Separating Hyperplane, Voronoi Diagram, Real-time Re-planning

Resumo

Esta tese apresenta uma metodologia eficiente e confiável para geração de trajetórias exequíveis, seguras e livres de colisões para grupos de veículos autónomos e que satisfaz os requisitos temporais de aplicações de tempo real. A metodologia proposta visa abordar as seguintes questões-chave:

(i) Satisfazer restrições que permitem evitar colisões em todas as instâncias de tempo de uma missão; A maioria dos métodos de geração de trajetórias existentes depende da discretização no tempo ou no espaço, o que faz com que só seja possível garantir que as restricões são satisfeitas entre os nós de discretização empregando uma grelha de discretização fina. Em alternativa, parametrizam-se as trajetórias como curvas de Bézier e recorre-se ao método de relaxamento e refinamento de Bernstein para garantir que as restrições são satisfeitas em todos os instantes de tempo.

(ii) Considerar o movimento rotacional de um drone para evitar problemas de exequibilidade para voos em espaços confinados; Para incorporar o movimento rotacional de um drone no problema, aproxima-se o corpo do drone por um elipsóide cujos eixos principais estão alinhados com os eixos da estrutura do corpo. O modelo elipsóide permite considerar explicitamente a orientação do drone e gerar trajetórias em ambientes com passagens estreitas.

(iii) Desacoplar restrições para evitar colisões entre veículos e obter um esquema distribuído com baixas exigências de comunicação; Para aliviar a complexidade computacional de resolver o problema de forma centralizada, apresenta-se uma estratégia que permite gerar trajetórias de forma local e distribuída através da resolução de um problema de otimização de pequena escala que envolve apenas as variáveis individuais de cada veículo. Para garantir que as decisões locais satisfazem as restrições anti-colisão, adota-se a partição do espaço de Voronoi e força-se cada veículo a gerar a sua trajetória dentro da sua célula de Voronoi.

A tese conclui com extensos resultados de simulação, envolvendo até 100 drones, que ilustram a eficácia da metodologia de geração de trajetórias proposta.

Palavras-chave: Geração de Trajetórias, Aplicações Multiveículo, Curvas de Bézier, Hiperplano de Separação, Diagrama de Voronoi

Acknowledgements

As I present my thesis, I am filled with gratitude for the diverse experiences that I have had during my PhD journey. These experiences have significantly contributed to my growth not only as a researcher but also as an individual. I am immensely grateful to the incredible people that have stood by my side throughout this journey. This work would not have been possible without their support and encouragement.

First and foremost, I want to express my appreciation for the invaluable guidance and constant support provided by my supervisor, Prof. Rita Cunha. Her keen insights and patient discussions have played a pivotal role in shaping the trajectory of my research work. Thank you, Rita, for you help.

I extend my sincere thanks to Prof. António Pascoal for his unwavering encouragement throughout the entire process. His generosity in sharing his valuable insights and willingness to help whenever needed eased my journey through the complexities of pursuing my PhD. Thank you, António, for your support.

I must thank Stephanie Pascoal and Filipa Almeida for their exceptional kindness upon my arrival in Portugal. Their warm welcome made a world of difference in my adaptation process. Thank you for your hospitality.

A special appreciation goes to my friends and colleagues, Ali, Azar, Duyen, Helena, Hung, Meysam, Leopoldo, Rômulo, Shuhbam, and Vasco. Our informal conversations, whether over lunch breaks or through screens during lockdowns, provided a lifeline during the most challenging times. I am thankful for all the memorable moments we have shared.

I would also like to acknowledge the financial support provided by Fundação para a Ciência e Tecnologia under projects PTDC/EEI-AUT/1732/2020 and PCIF/MPG/0156/2019, which enabled me to pursue my research goals.

Last but certainly not least, I want to convey my heartfelt gratitude to my family for their unwavering support and encouragement. Their words of motivation and understanding helped me persevere through challenges. To my parents: Thank you for everything. I dedicate this thesis to you.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
  - 1.1.1 Autonomous Drone Cinematography . . . 2
  - 1.1.2 AUV range-based positioning . . . 4
- 1.2 Related work . . . 5
  - 1.2.1 Artificial Potential Field Methods . . . 5
  - 1.2.2 Grid-based Search . . . 6
  - 1.2.3 Sampling-based Methods . . . 6
  - 1.2.4 Numerical Optimal Control . . . 7
  - 1.2.5 Polynomial-based Methods . . . 7
- 1.3 Contribution . . . 8
  - 1.3.1 Bézier parameterization and efficient evaluation of Inequality constraints . 8
  - 1.3.2 Trajectory generation for drones in confined spaces using an ellipsoid model of the body . . . 9
  - 1.3.3 Distributed trajectory generation framework . . . 11

2 Trajectory Generation using Computationally Efficient Optimal Control Methods **13**
- 2.1 Introduction . . . 14
- 2.2 Problem Description . . . 14
- 2.3 Numerical Methods for Optimal Control Problems . . . 15
  - 2.3.1 Single Shooting Method . . . 16
  - 2.3.2 Direct Multiple Shooting Method . . . 18
  - 2.3.3 Direct Collocation Method . . . 21
  - 2.3.4 Available tools for solving OCPs and NLPs . . . 22
- 2.4 Simulation Results . . . 25
  - 2.4.1 Go-to-Formation Maneuver of 7 AUVs . . . 26
  - 2.4.2 Minimum time maneuver with collision avoidance . . . 29
  - 2.4.3 Trajectory optimization for range-based AUV positioning . . . 29
  - 2.4.4 Autonomous Drone Cinematography . . . 31
    - 2.4.4.1 Quadrotor model . . . 31
    - 2.4.4.2 Gimbal angles . . . 34
    - 2.4.4.3 Problem formulation . . . 36
    - 2.4.4.4 Simulation results . . . 36
  - 2.4.5 Cooperative planning for multiple drones . . . 37
  - 2.4.6 Trajectory re-planning in the receding horizon manner . . . 40

3 Bézier Curve-based Trajectory Generation Method for Differentially Flat Systems **45**
- 3.1 Introduction . . . 46
  - 3.1.1 Differentially flat systems . . . 46

3.1.2 Polynomial parameterization of the flat output . . . 48
3.2 Bernstein polynomials and Bézier curves . . . 52
3.2.1 Bernstein Polynomial: definition and basic properties . . . 53
3.2.2 Bézier curves . . . 57
3.2.2.1 Definition and shape features . . . 57
3.2.2.2 Algorithms . . . 60
3.2.3 Evaluating Inequality constraints using B-spline and Bézier curves properties 63
3.2.3.1 Literature Review . . . 64
3.2.3.2 Evaluating inequality constraints in Bernstein form . . . 68
3.2.4 Quantitative bounds on the distance between a Bézier curve and its control polygon . . . 70
3.2.4.1 Bézier control polygon . . . 70
3.2.4.2 Bounding functions . . . 72
3.2.4.3 Bound alternatives . . . 76
3.2.4.4 Sharpness of the bounds . . . 76
3.2.4.5 Bound improvement at the end points . . . 77
3.2.4.6 Polygonal Envelopes . . . 78
3.2.4.7 Convergence under subdivision . . . 78
3.2.4.8 Convergence under degree elevation . . . 81
3.3 Case study . . . 82
3.3.1 Go-to-Formation maneuver . . . 82
3.3.1.1 Simulation Results . . . 85
3.3.2 Collision-avoidance constraints for an ellipsoid model of the drone body . 87
3.3.2.1 Quadrotor model . . . 87
3.3.2.2 Collision Avoidance Constraint . . . 91
3.3.2.3 Simulation results . . . 94

4 Distributed Algorithm for Real-time Multi-drone Trajectory Re-planning 103
4.1 Literature Review . . . 104
4.2 problem formulation . . . 106
4.2.1 Decoupling the inter-vehicle collision avoidance constraint . . . 107
4.2.2 Finding the closest point to the goal position . . . 111
4.2.2.1 Overview . . . 112
4.2.2.2 Configuration Space Obstacle . . . 112
4.2.2.3 Support Mapping Function . . . 113
4.2.2.4 Simplices . . . 114
4.2.2.5 Convergence and termination . . . 115
4.2.2.6 Johnson's Distance Sub-algorithm . . . 117
4.2.3 Continuity conditions . . . 123
4.3 Simulation Results . . . 124

5 Conclusion and Future Work 129
5.1 Conclusion . . . 130
5.1.1 Summary . . . 130
5.2 Future Work . . . 131
5.2.1 Rational Bézier curves . . . 131
5.2.2 Computation delay compensation for real-time implementation . . . 132

A Structure exploiting NLP solver 133

B FORCES Pro **137**
B.1 FORCES Pro High-Level Interface 137
B.1.1 Expressing the optimization problem in python 137

C B-spline Curves **141**
C.1 B-spline basis functions: definition and properties 141
C.2 B-spline curves: definition and properties 143
C.2.1 Convergence under knot insertion 145

List of Figures

1.1 System architecture on board a drone. The Shot Executer module provides desired shot types, whereas the Cinematography Planner generates optimal trajectories and gimbal commands [Sab+19]. . . . 3

1.2 Schematic of mutual visibility constraint. Other flying drones must be stayed out of the camera's field of view. . . . 4

1.3 The trajectory in blue maximizes the information content of the set of range measurements to the beacon (fixed at the origin) while the trajectory in green minimizes the mechanical energy usage. . . . 5

1.4 Existing motion planning approaches usually model the vehicle as a sphere or prism (**left**). Therefore, the configuration space (C-space) can be obtained merely by inflating the obstacles with the radius of the sphere. As a result, the vehicle can be treated as a single point in C-space and the collision checking is simplified (**right**). . . . 10

1.5 The conventional method of modeling the drone body as a sphere is not suitable for motion planning in tight spaces, as it fails to validate trajectories that might be feasible upon considering the drone's orientation. The less conservative ellipsoid model allows consideration for the 3D orientation. . . . 11

1.6 The 2D Voronoi diagram for 5 drones. The final position for the (red shaded) drone is shown with ■, and the closet point in its Voronoi cell to the goal position is shown with ●. The vehicle's trajectory is generated towards ● such that it is entirely within the vehicle's Voronoi cell and avoids the obstacle (black circle) inside the cell. . . . 12

2.1 Numerical approaches to continuous time OCPs [DG11] . . . 17

2.2 The state and input trajectories when solving an OCP with $n_x = 1$ and $n_u = 1$ via single shooting discretization. . . . 18

2.3 The sparsity pattern of the Hessian of the Lagrange function (**left**) and the sparsity pattern of the Jacobian of the inequality constraints (**right**) in the NLP resulting from single shooting discretization. . . . 18

2.4 Illustration of the direct multiple shooting method at an early iteration before the constraints are satisfied [DG11]. The continuity condition $\mathbf{x}_k(t_{k+1}, \mathbf{s}_k, \mathbf{q}_k) - \mathbf{s}_{k+1} = 0$ is enforced at $\mathbf{s}_0, \ldots, \mathbf{s}_{N-1}$. The state trajectory becomes continuous once the solution of the NLP is achieved. . . . 20

2.5 The sparsity pattern of the Hessian of the Lagrange function (**left**) and the sparsity pattern of the Jacobian of the equality constraints (**right**) in the NLP resulting from multiple shooting disretization. The variables are ordered as $\mathbf{z} = (\mathbf{s}_0, \mathbf{q}_0, \mathbf{s}_1, \mathbf{q}_1, \ldots, \mathbf{s}_{N-1}, \mathbf{q}_{N-1}, \mathbf{s}_N)^T$. . . . 20

2.6 Illustration of the direct collocation method with $d = 3$ at the time interval $[t_k, t_{k+1}]$ [DG11]. . . . 23

2.7 The sparsity pattern of the sparsity pattern of the Jacobian of the equality constraints in the NLP resulting from direct collocation with $d = 3$. The variables are ordered as $\mathbf{z} = (\mathbf{v}_{0,0}^T, \mathbf{q}_0^T, \mathbf{v}_{0,1}^T, \mathbf{v}_{0,2}^T, \mathbf{v}_{0,3}^T, \mathbf{v}_{1,0}^T, \mathbf{q}_1^T, \mathbf{v}_{1,1}^T, \dots)^T$ 23
2.8 Spatially de-conflicted trajectories for 7 vehicles moving from their initial positions (x) to their final positions (x) with a constant speed of $0.5\frac{m}{s}$ and $\psi_0 = \psi_f = 0$. 27
2.9 The course angle, course rate and angular acceleration of the 7 vehicles are within the given bounds in Table 2.1. 27
2.10 Spatially de-conflicted trajectories for 7 vehicles from their initial positions (x) to their final positions (x), with the boundary conditions, $v_0 = 0.1$m/s, $v_f = 0.5$m/s and $\psi_0 = \psi_f = 0$. 28
2.11 The speed and course angle of the 7 vehicles are within the given bounds in Table 2.1. 28
2.12 Temporally de-conflicted trajectories for two vehicles moving from their initial positions (x) to their final positions (x) in a cluttered environment. 29
2.13 The course angle and speed profile for the two vehicles in Figure 2.12 29
2.14 (a) and (b) show the AUV's trajectory obtained by minimizing the overall energy consumption and maximizing the $\log|FIM|$, respectively; (c) and (d) are the corresponding course rate and speed profiles for the trajectory in (a) and (b), respectively [SCP18]. 32
2.15 Two AUVs and a single beacon. The temporally separated trajectories generated in the absence and presence of $\log|FIM|$ in the objective fucntion are shown in (a) and (b), respectively. The input profiles for the trajectories in (a) and (b) are respectively shown in (c) and (d). 33
2.16 Illustration of reference frames. The origin of the camera and the vehicle reference frames coincide. 35
2.17 Generated trajectories for the single drone-single target example with different relative weights in the objective function (2.50). The trajectories guide the vehicle from its initial position (x) to the final position (x) while the gimbal is pointing towards the target (fixed at the origin). 37
2.18 The top view (**left**) and the gimbal pitch angle (**right**) of the generated trajectories in Fig. 2.17. 38
2.19 The linear velocity and acceleration of the generated trajectories in Fig. 2.17. As w_2 increases the resulting trajectory reduces the gimbal pitch angle rate at the cost of increased energy consumption. 38
2.20 Schematic of mutual visibility constraint. Other flying drones must be stayed out of the camera's field of view. 39
2.21 The generated trajectory for two drones (black and blue) at different time instances. The trajectories are generated such that both cameras have unobstructed views of the target moving on the red line. 39
2.22 The computed β angles for the two cameras show that the mutual visibility constraints are satisfied over the entire time interval. 40
2.23 The generated trajectory for two drones (black and blue) at different time instances. The trajectory for the first drone is generated such that both cameras have unobstructed views of the target moving on the red line. 41
2.24 The top view of the trajectories shown in Fig. 2.23. 42
2.25 The computed β angles show that both cameras have unobstructed views of the moving target. 42

2.26 (a) Replanning the trajectory in the presence of uncertainty in the target position estimates. The drone trajectory is re-planned at each time step taking into account the most recent measurements of the target and the other drone positions. (b) The initial and final trajectories of the drone. The final trajectory is obtained by applying the first part of the re-planned trajectory at each time step. 43

3.1 Converting the original constraints (left) into equivalent ones in the space of flat output (right). 48
3.2 Possible paths joining the initial and final points. 50
3.3 Three polytopes embedded within a non-convex set [VM98]. 51
3.4 Bernstein basis functions of degree 0, 1, 2, 3, 4, and 5. 54
3.5 A cubic Bézier curve with control polygon 58
3.6 A quartic Bézier curve (left) and its first derivative (right). The derivative at the endpoints depend only on the first two and last two control points. 58
3.7 A cubic Bézier curve contained within the convex hull defined by its 4 control points. The control points are shown in red circles and control polyline in dashed line segments. 59
3.8 A Bézier curve oscillates no more than the piece-wise linear interpolant to its control points. 59
3.9 A cubic Bézier curve is evaluated $\tau = 0.4$ (left). The pyramid scheme of the de Casteljau's algorithm (right) for 3 iterations. The right and left sides of the pyramid show the control points for the new control polygons. 60
3.10 A cubic Bézier curve is divided into two Bézier curves of the same degree, $n = 3$, at $\tau = 0.3, 0.5, 0.7$, using the de Casteljau's algorithm. 61
3.11 A cubic Bézier curve is expressed in terms of higher degree basis functions, $n = 4, 6, 15$, using the degree elevation algorithm. 62
3.12 A Bézier curve of degree 4 (red) and a Bézier curve of degree 5 (green). The solid lines are the control polylines and the shaded areas are the convex hulls of the curves. According to (3.82), the upper bound of the minimum distance between the two curves is $\|r_0 - s_4\| = 1$, and the lower bound, i.e., the distance between the convex hulls, is 0. 65
3.13 Comparing the minimum spatial (**left**) and temporal (**right**) distance between a Bézier curve of degree 4 (red) and a Bézier curve of degree 5 (green). The parameter value is indicated in the color bar to show the temporal evolution of the curves. The results are computed using Algorithm 1 with $\epsilon = 10^{-8}$ in < 4 ms. 66
3.14 A B-spline curve of degree 3 (bottom left) and the corresponding B-splines defined over the knot vector $T = [0, 0, 0, 0, 1, 1, 2, 2, 2, 4, 5, 5, 5, 5]$ (**top left**). Figures on the right show the effect of inserting a single knot at $t = 3$ on the B-splines and the control polyline. The three new control points, generated with Boehm's algorithm and shown with red dots, replace the original control points r_6 and r_7. 68
3.15 A cubic Bézier curve (**top left**) is subdivided into two Bézier curves of the same degree (**top right**) using the de Casteljau's algorithm. Successive refinement of the original control polygon after 2 (**bottom left**) and 3 (**bottom right**) subdivisions. The composite control polygon generated by repeated subdivisions converges to the Bézier curve. 70
3.16 $\alpha_i(\tau)$ for $n = 5$ and $i = 0, \ldots, 5$. For any partcular value of τ, the sum of α_i is 0, i.e., $\sum_{i=0}^{n} \alpha_i(\tau) = 0$. 71
3.17 $\beta_i(\tau)$ for $n = 5$ and $i = 1, 2, 3, 4$. $\beta_i(\tau)$ is monotonically increasing on $[0, \tau_i]$ and decreasing on $[\tau_i, 1]$, where $\tau_i = \frac{i}{n}$. The dashed line shows their piece-wise sum, i.e., $\sum_i \beta_{ki}(\tau)$. 72
3.18 The first anti-differences of α_{ki}, $\gamma_{ki}(\tau)$, for $n = 5$ and $i = 0, \ldots, 4$. The dashed line shows the piece-wise sum of the absolute of γ_i, i.e., $\sum_i |\gamma_{ki}|$. 73

3.19 One dimensional Bézier curve (solid black line), its control polyline (solid blue line), and the envelope (dashed line) constructed with the bound implied by (from left to right) (1) N_∞, (2) N_2, (3) N_1. The control point sequence for the Bézier curve is (from top to bottom) (1) $[0, 1, 1, 0]$, (2) $[0, 1, 3, 6, 10, 14]$, (3) $[0, 1, -1, 0]$, (4) $[0, 1, 2, 3, 2, 1]$. 79

3.20 The go-to-formation maneuver for 5 vehicles. Trajectories are generated with the Béier curve-based method proposed in this chapter. 88

3.21 Collision-free trajectories for 5 drones generated with Bézier curves of degree 8. . 88

3.22 To deal with the uncertainties in the obstacles' positions, trajectories for two drones are re-planned at different time instances. The trajectories are generated using Bézier curves of degree 8. Imposing continuity constraints at the joining point of consecutive segments does guarantee smoothness of the overall trajectory. 89

3.23 The quadrotor reference frames. 90

3.24 The quadrotor body can be represented as a sphere with radius $r_{\mathbf{D}}$ (**right**), or an ellipsoid aligned with the axes of the body frame (**left**). Approximating the drone body with an ellipsoid allows considering the quadrotor's rotational motion. 92

3.25 2D sketch of an ellipsoid E_1 and a hyperplane H in the original space (**left**) and the corresponding sketch in the transformed space (**right**) in which E_1 is transformed into a unit ball at the origin. 93

3.26 Comparing the proposed method in the thesis to the method in [Fal+17] for generating a trajectory that guides a drone through a gap inclined at 45°. 95

3.27 The orthogonal plane Π to an inclined gap [Fal+17] (**left**). The drone's trajectory must pass through the center of the gap, $\mathbf{p}_G$, while lying in the plane Π. A view of the traverse trajectory in the direction of the normal vector to the plane ${}^{\Pi}\mathbf{e}_3$ (**right**). 96

3.28 The generated trajectory for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a gap with the frame size of 120mm × 550mm, centered at $[3.4, 1.6, -2]$ and inclined at 30°. 97

3.29 The generated trajectory for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a gap with the frame size of 120mm × 550mm, centered at $[3.3, 1.7, -2]$ and inclined at 45°. 98

3.30 The generated trajectory for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a gap with the frame size of 120mm × 550mm, centered at $[3.5, 2, -2.5]$ and inclined at 90°. 99

3.31 Comparing the trajectory generated with the proposed method in the thesis (solid line) to the one generated using (3.208) (dashed line) for flying a drone through a 180mm × 550mm gap centered at $[3, 1.4, -2]$ inclined at 60°. 100

3.32 The experimental results for a single drone flying through a gap inclined at 60°. . 101

3.33 Comparing collision-free trajectories, for two drones switching positions, generated with an ellipsoid model of the drone body (solid lines) and a sphere model (dashed lines). 102

3.34 Generated trajectories for 4 drones flying through a gap and switching positions in a given time. The initial (**left**) and final (**right**) positions of the drones are shown in the XY-plane. Using a sphere model of the drone body yields an infeasible problem. 102

3.35 Collision-free trajectories for steering two drones through a narrow gap in a wall towards their desired position on the other side. The initial and final positions are shown with squares and circles respectively. 102

4.1 (**a**) The Voronoi diagram for six drones in 2D space. The Voronoi boundary edges are shown with solid black lines, and the buffered Voronoi cells are shaded in dark blue. (**b**) The Voronoi diagram for 10 drones in a collision-free configuration in 3D space. The Voronoi boundary $\partial\mathcal{V}$ is shaded in light blue, and the buffered Voronoi cells $\bar{\mathcal{V}}$ for two neighboring drones in the center are shown in dark blue. 110
4.2 The Minkowski difference of two convex polygons A and B. Since A and B intersect, C contains the origin. 112
4.3 Finding the minimum distance between two convex bodies A and B is equivalent to finding the minimum distance of their Minkowski difference, C, to the origin. . 113
4.4 The support point of the Minkowski difference can be determined by subtracting the support points of the two shapes. 113
4.5 Simplices in $\mathbb{R}^2$. 114
4.6 Each feature of a simplex is linked to a Voronoi region. 115
4.7 Finding the closest point to the origin on a polygon using the GJK algorithm. The set of vertices $Y = V_k \cup \{\mathbf{w}_k\}$ and $V_{k+1} \subset Y_k$ are shown at each iteration. The iterative search descends such that the generated simplex at each iteration offers a better approximation of the $\nu(C)$ that the previous one. 118
4.8 Examples of the closest feature of a polyhedron to the origin are shown above. The closest feature can be a vertex, an edge or a face with $|V| = 1, 2,$ or 3, respectively. If $|V| = 4$ the origin is contained in the interior of P. Once the closest feature specified with V is obtained, the closest point, i.e., $\nu(P)$, can be determined as shown above. 119
4.9 The bottom-up approach to searching all $2^{m+1} - 1$ non-empty subsets of Y for a 2-simplex used in Johnson's distance subalgorithm (**left**). The method presented in Alg. 6 to 8 conducts a search only through 2^m subsets whose Voronoi region can possibly contain the origin (**right**). 120
4.10 A m-simplex is linked to $2^{m+1}-1$ Voronoi regions associated to its vertices, edges, faces, and volume. The list of 2^m Voronoi regions that can possibly contain the origin is given in this table. It should be noted that $\mathbf{v}_1$ is the latest vertex added to V. 124
4.11 Comparing collision-free trajectories generated with the centralized solution (**left**) and the proposed decentralized approach (**right**) for five drones flying from their initial positions to given final positions. While the central solution yields a shorter flight time, its computation time is significantly longer than average time required to solve the sub-problems in the distributed method. 126
4.12 (**a**) Initial (left) and final (right) position configurations for 18 drones. Each drone is assigned a unique color and a number next to it. (**b**) Collision−free trajectories for 18 drones switching their positions in a 3 m × 5 m × 2 m space. The total flight time for the drones to reach their final positions is 5.1 s using the proposed method, which is shorter than the 6.3 s flight time obtained with the BVC. . . . 127
4.13 Collision−free trajectories for 100 drones flying from their initial positions (dots) to randomly specified final positions (squares) at different replanning steps. . . . 128

B.1 Supported problem in FORCES Pro high-level interface [DJ19] 137

C.1 B-spline basis functions of degree 0, $T = \{0, 0.25, 0.5, 0.75, 1\}$. 141
C.2 B-spline basis functions of degree 1, $T = \{0, 0.25, 0.5, 0.75, 1\}$. 142
C.3 B-spline basis functions of degree 2, $T = \{0, 0.25, 0.5, 0.75, 1\}$ 142
C.4 B-spline basis functions of degree 1, $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$. 143
C.5 B-spline basis functions of degree 2, $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$. 143

C.6 A cubic B-spline curve and its control polygon (left), the corresponding B-spline basis functions(right) with $T = \{0, 0, 0, 0, 1, 1, 2, 2, 2, 4, 5, 5, 5, 5\}$. The control polygon is obtained by connecting the points with coordinates $(\bar{t}_i, c_i), i = 0, \ldots, n$, where $\bar{t}_i = \frac{t_{i+1}+\cdots+t_{i+k}}{k}$. 144
C.7 A cubic B-spline curve and its control polygon (left), the corresponding B-spline basis functions(right) with unclamped $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. The curve is not clamped at the end points, and is only defined over the interval $t \in [t_k, t_{m-k}] = [3, 5]$ 144

List of Tables

2.1 Upper/lower bounds on states and inputs 26
2.2 Upper/lower bounds used in the simulations. 37

3.1 The smallest possible constant that bounds the distance between a Bézier curve and its control polygon, for 1, 2, and ∞-norm, and $n = 2, \ldots, 8$. 78
3.2 Upper and lower bounds on the states and inputs of the vehicle's model 87
3.3 Recorded computation times for generating trajectories using different approaches to evaluating inequality constraints. 87

4.1 Comparing the number of successful trials and the average flight time achieved with the BVC and the proposed method in the paper. 128
4.2 Recorded computation times for finding the closest point in a Voronoi cell to the goal position and solving the optimization problem in simulation examples with 18, 34, and 100 drones. 128

C.1 B-spline basis functions of degree 0, $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$. 142

Chapter 1

Introduction

1.1 Motivation

Motion planning, also referred to as path planning, is an essential part of real-world applications of autonomous vehicles. Motion planning can be defined as the computational problem of finding a valid trajectory (or path) that guides the vehicle from an initial state to a (given) final state. In most applications, the main concern with motion planning, rather than just finding a feasible trajectory between the initial and final states, is the optimality of the trajectory with respect to a certain objective function that measures the accomplishment of mission-specific goals. Therefore, in *optimal motion planning*, the goal is to find the trajectory, out of all admissible ones, that minimizes or maximizes an objective function. A straightforward approach to optimal motion planning is to formulate it as a standard form optimization problem that can be solved using existing methods and algorithms. In informal settings, such an optimization problem can be written as

$$\begin{aligned} \underset{\mathbf{P}(.)}{\text{minimize}}\ & J(.) \\ \text{s.t. } & \text{boundary conditions,} \\ & \text{dynamic constraints,} \\ & \text{collision avoidance constraints,} \end{aligned}$$

where $P(.)$ is the trajectory and $J(.)$ is the cost function defined based on mission goals and objectives. The set of constraints define the feasible region for the optimization problem. The boundary conditions contain the initial and final values of the vehicle's state. The constraints imposed by the vehicle dynamics must also be considered to ensure that the generated trajectory is dynamically feasible, and finally the collision avoidance constraint need to be included to secure safe distance between the vehicle and obstacles in the environment. The above optimization problem usually involves other constraints to account for mission-specific objectives and requirements.

The above problem generally casts as a non-convex optimization problem, and thus, finding the optimal trajectory, $P^*(.)$, can be computationally challenging. On the other hand, with rapid advances in related technologies, autonomous vehicles continue to take part in more complex missions, and even engage in teams of collaborating vehicles to take on increasingly demanding tasks. This requires the above optimization problem to incorporate extra constraints to guarantee that inter-vehicle collisions are avoided. The increased number of constraints and optimization variables in multi-vehicle missions would further aggravate the computational issues of finding optimal trajectories. In the following, we describe the trajectory generation problem in two particular applications–aerial cinematography and AUV range-based positioning–where dealing with conflicting objectives and challenging constraints is inevitable for generating trajectories that fulfill mission goals and requirements. Existing motion planning algorithms and tools lack the versatility and efficacy required to accommodate the specific needs of such missions. This calls for the development of optimal trajectory generation frameworks with significantly enhanced computational efficiency that allow for seamless integration of complex constraints and objectives into the problem.

1.1.1 Autonomous Drone Cinematography

For years now, drones have been used to capture aerial shots in professional photography, in real estate, and even in high-end cinematography and filmmaking. Besides significant cost reductions, drones have the technical capability to produce high-quality dynamic aerial shots a lot simpler and faster than the traditional crane/jib or helicopter. With lower time spent for preparation, drones can fly up to 400ft high in the air for a high-altitude footage or move just a few inches

from the ground for a low-angle tracking shot; tasks that have never been possible before with a crane. Yet, the majority of drone videography have to date been carried out by two trained operators; a pilot to fly the drone and a cameraman to handle camera angles and movements. Therefore, conceiving and exploring new ideas for enabling drones to accomplish these tasks autonomously or with minimal human intervention will be at the core of future developments.

One such idea has been investigated in the MULTIDRONE project [20], where an intelligent platform involving multiple drones is developed for outdoor sports event video production. In such applications, whether it be filming paddling competitions on flatwater rivers and lakes, or making footage of bikers and runners in mountain trails and city streets, the team of drones should work harmoniously to take videos that meet the director's needs and expectations in terms of visual quality. The system architecture on board each of the drones used in MULTIDRONE trials and experimental media productions is shown in Fig. 1.1. Given the target's position and a specific shot type, the planner must generate trajectories that not only comply with video aesthetic quality objectives but also ensure flight safety and feasibility. Thus, the trajectory planner plays a critical role in generating visually pleasing videos.

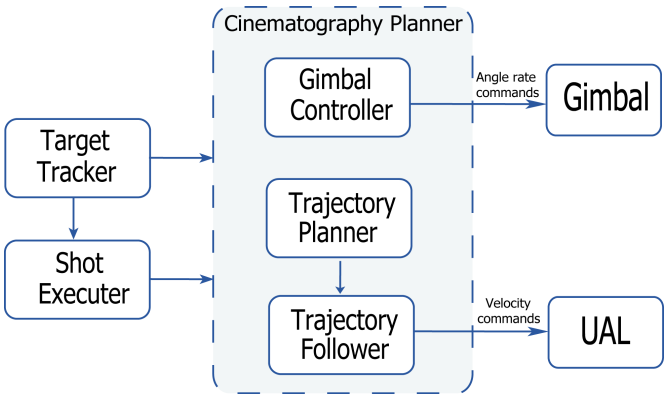


Figure 1.1: System architecture on board a drone. The Shot Executer module provides desired shot types, whereas the Cinematography Planner generates optimal trajectories and gimbal commands [Sab+19].

In a representative trial, admissible trajectories ought to satisfy several constraints imposed by drone dynamics, gimbal movement limitations, surrounding environment, and video framing and composition;

- The constraints enforced by dynamic capabilities of drones must be taken into account to ensure that the generated trajectories are dynamically feasible and respect the bounds on the flight attitude, speed and acceleration.

- The position and orientation of the drone can obviously alter the camera position and angle, and therefore, in order to secure smooth camera movements, the gimbal mechanical limits to rotate around each axis should also be explicitly considered while generating the drones' trajectories.

- Constraints on the camera's field of view (FOV) need to be considered to guarantee that unwanted objects, including landing gears and other drones, stay out of the FOV for an unobstructed shot of the target of interest.

- Additional measures must be considered for the generated trajectories to be safe. This includes keeping a minimum distance to obstacles or in between the drones, and steering away from no-fly zones on the map to avoid potential collisions and hazards.

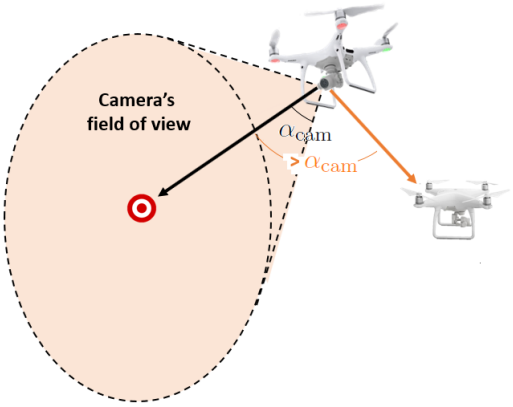


Figure 1.2: Schematic of mutual visibility constraint. Other flying drones must be stayed out of the camera's field of view.

In order to generate videos that both experts and novices can enjoy, trajectories must also optimize an objective function derived from aesthetic perception of aerial videos. The objective function may include different terms to maximize the smoothness of camera movement over the entire trajectory and/or minimize the positional and angular jerk. Trajectories that reach such objectives and satisfy the above mentioned requirements can only be obtained by solving an optimization problem with properly defined cost function and constraints.

1.1.2 AUV range-based positioning

Thanks to technological advances needed for reliable deployment, control and recovery, AUVs are now capable of exploring and sampling previously impenetrable environments such as deep ocean and under glaciers. With the development of deep-diving long-endurance AUVs, the near future will witness a significant increase in the number of AUVs' long-term, long-range, and deep-water missions. One of the main challenges for AUVs to be deployed in long-range surveys is the limited amount of energy available on-board the vehicle. This poses severe restrictions upon AUV's maneuverability. In order to increase the mission duration and fulfill the mission goals as great as possible, e.g. exploring larger areas of the seafloor, with a given energy (power) storage, it is necessary to generate energy efficient trajectories by minimizing a cost function involving the overall power consumption. The actual power consumption is dependent on different factors and should not be oversimplified by merely computing the mechanical energy usage. In addition to the power consumed for propulsion, other contributing factors such as computers and sensors carried onboard the vehicle must be considered as well. Furthermore, the power required for emergency recovery between recharge periods should be taken into consideration to ensure that, in case of unforeseen situations like adverse weather conditions, the AUV can be safely recovered.

The GPS failure to provide location under water is another major challenge that should be dealt with in AUV missions. In deep-sea and under-ice explorations, where occasional surfacing for position update is not possible, positioning need to be performed without GPS support. The AUV positioning system that relies on range-based and geophysical (terrain, geomagnetic, and gravimetric) methods serves as an alternative. The range-based positioning system fuse the range measurements, between the vehicle and one or more external references in the form of beacons at known positions, to estimate the AUV position. Therefore, the performance of this system is highly dependent on the AUV's trajectory and the information provided by the measurements. In order to obtain the set of measurements that yield the most relevant information for estimating the position, trajectories must be generated such that a proper quantifier for the information content of measurements is maximized.

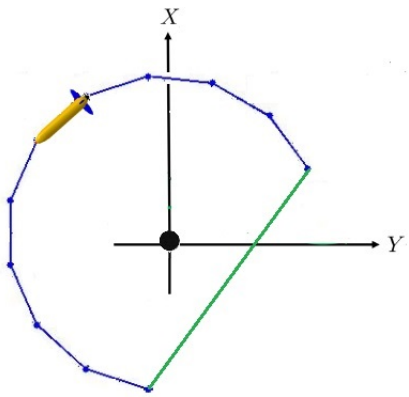


Figure 1.3: The trajectory in blue maximizes the information content of the set of range measurements to the beacon (fixed at the origin) while the trajectory in green minimizes the mechanical energy usage.

Fig. 1.3 compares the trajectory that maximizes the determinant of the Fisher information matrix (FIM), measuring the information content provided by range measurements, with one that minimizes the overall mechanical energy usage. It is clear that in order to find a trajectory that is energy efficient and also sufficiently exciting for estimation, a compromise must be struck between the two competing objectives. Other limiting factors for conducting long-range deep-water missions, such as under-water communication and harsh weather condition, induce more conflicting objectives. Therefore, a multi-objective optimization problem must be solved to generate trajectories that offer the best trade-off among all competing mission objectives.

1.2 Related work

In the following, we give a brief review of the most notable motion planning algorithms used in autonomous vehicle applications [GKM10]. A more in-depth look into related work is given in Chapter 2 and Chapter 3.

1.2.1 Artificial Potential Field Methods

The basic premise behind potential field methods is simple; they assign real-valued and differentiable potential functions to the configuration space and consider the vehicle as a (positively-charged) particle reacting to forces due to the potential field [Kha86]. The final point has the lowest potential and therefore attracts the vehicle while obstacles emit positive charges and repel the vehicle. Potential field methods are known for their low computational complexity but a serious issue in these methods is that they are incomplete; the particle can get stuck in a local minimum in the potential field that does not correspond to the final point in the configuration space, and thus they fail to find a path. The extended artificial potential field method, proposed in [MM08], uses dynamic internal agent states to avoid the local minimum problem. Another way to address the completeness problem is to develop navigation function [RK88] or a probabilistic navigation function [HSS19] that does not have minimum points except at the target point. Another major drawback of potential field methods is that non-holonomic constraints and vehicle kinematics are difficult to accommodate. The authors in [KNN12] apply the potential function method to multiple non-holonomic vehicles that are described with a simple tricycle model. In [SKI05] potential function method is used to the high-speed navigation of UGVs such that dynamic constraints, including vehicle side slip and rollover, are satisfied. In general, potential functions provide an efficient path planning method suitable for online applications, but complicated and underactuated dynamics remain difficult to manage.

1.2.2 Grid-based Search

Grid-based approaches cover the configuration space with a grid. Each point on the grid denotes a configuration (or state) of the vehicle ([Don+93], [DX95], [FLS05]). The vehicle at a grid point can move to adjacent grid points as long as the line between them is completely contained within the free space. Grid-based methods rely on a collision detection module to verify that edges are in the free space. These approaches discretize the set of actions, and assign a (non-negative) number or cost to each edge to quantitatively express characteristics of the corresponding path, e.g. the length of a path. These numbers are later used by search algorithms, like Dijikstra or A* [DP85], to find the shortest path from the start to the goal. The performance of the grid-based search is highly dependent on the grid resolution. The search is faster over coarser grids, but it might fail to find paths in $\mathcal{C}_{free}$ for certain pairs of start and goal configurations. Also, the optimality of the solution is only valid if the resolution of the grid is sufficiently high. However, finer search can be very slow especially for high-dimension configuration spaces, since the number of points on a grid grows exponentially with the dimension. Therefore, grid-based search approaches for high-dimensional planning problems result in an adverse computational burden. This motivates incorporating random sampling methods and probabilistic methods as will be discussed shortly. Yet, the major drawback to grid-based search methods remains to be their inability to take into account the vehicle dynamic model, including nonholonomic and actuator constraints.

1.2.3 Sampling-based Methods

The main idea behind sampling-based motion planning is to avoid the obstacle space, $\mathcal{C}_{obs}$, using some collision detection algorithm, while conducting a search throughout the configuration space (or the state space) with a (random) sampling scheme. The most commonly used methods in this category are the Probabilistic Roadmap Method (PRM) ([Kav+96]) and the Rapidly-exploring Random Tree (RRT) ([KL00]). Single-query methods such as RRT are generally faster than multiple-query methods like PRM. However, single-query methods must repeat the search for each pair of initial and final configurations, while multiple-query methods are able to use the generated roadmap (graph) multiple times as long as the environment does not change.

All sampling-based methods perform a few common steps, such as generating new nodes and adding collision-free edges, to grow a tree or roadmap rooted at the initial configuration. The term *node* used in technical literature denotes the configuration of the vehicle in the configuration space. New nodes are generated in a random, quasirandom, or deterministic manner. The characteristics of the generated nodes are dependent on the sampling method. A new node is usually chained to the nearest node to form an edge. Newly added nodes and edges are usually tested immediately for compatibility with constraints and invalid nodes are discarded. This is enabled by a collision detection module, considered as a black box by the algorithm, that checks for inclusion in the free space, $\mathcal{C}_{free}$ ([LaV06]). In order to speed up the computation time, the so-called lazy methods avoid collision checks until a complete path, from the initial configuration to the final configuration, is formed ([BK00], [Hau15]).

Compared to grid-based methods, sampling-based methods can efficiently search complex high-dimensional configuration spaces by randomly building a space-filling tree or a graph, avoiding the complexities of building $\mathcal{C}_{obs}$ representations. One major problem with sampling-based methods is that the resulting path is usually not feasible due to lack of smoothness. This requires these methods to be paired with post-processing techniques for generating smooth feasible paths as proposed in [FDF05] and [KP06]. Also, RRT and PRM in their original versions, cannot provide an optimal trajectory across the entire configuration space, and thus, they are best suited for finding feasible solutions. Optimal extensions to standard RRT and PRM algorithms, PRM* and RRT* ([KF11]), have difficulty coping with complicated or underactuated dynamics. Algorithms proposed in ([Per+12]) and ([GT10]) for growing a standard RRT or

RRT* in domains with constrained and underactuated dynamics use linear quadratic regulation (LQR)-based heuristics via linearizing the system dynamics about newly sampled points.

1.2.4 Numerical Optimal Control

From a control perspective, the optimal motion planning problem is an instance of an optimal control problem, and thus, optimal control techniques can be used to find the optimal trajectory. In general, these methods solve for the (control) trajectory that minimizes a cost function and satisfies constraints including those imposed by vehicle dynamics. These problems are often too complex to be solved analytically and therefore one must resort to numerical approaches. High sensitivity to initial guess (especially in non-convex problems), long computational times, and poor convergence to the optimal solution were three main concerns with earlier use of numerical optimal control methods for trajectory generation [Bet98]. Therefore, optimal control methods were frequently used in an off-line manner or in conjunction with other motion planning methods, such as RRT, for a good initial guess. All the three mentioned issues have benefited from the advances in numerical optimization algorithms and computational power. With an efficient and robust solver, an optimal control problem mainly becomes one of a problem formulation. Recent work has demonstrated the applicability of numerical optimal control methods to trajectory generation in complex applications including real-time planning for automated drone cinematography [Näg+17], for autonomous flight through narrow windows [GM16], and autonomous racing [Váz+20].

Receding horizon control (RHC) or model predictive control (MPC) fall into the same category in which a finite-horizon problem is solved numerically over an ever-receding horizon. In order to converge to the global optimal solution, using an appropriate cost-to-go function that captures the discarded portion of the trajectory is essential. The author in [Jad01] uses receding horizon control to solve trajectory planning problems for general nonlinear systems in an obstacle-free environments, and provides necessary conditions for the stability of the RH scheme. The results in[Kuw07], [Liu+17a], and [GGJ12] illustrate the efficacy of MPC for real-time trajectory generation applications. Solving the optimization problem repeatedly for a reduced time horizon can significantly reduce the computational effort, while allowing for model uncertainties to be directly taken into account. However, except for trivial cases, optimality and completeness are usually difficult to prove, and proper design of terminal cost function is necessary to ensure that the on-line planner is complete.

In general, optimal control methods are complete and their computational complexity is not aggravated by high-dimensional state spaces the same way as grid-based search methods are, since more generic numerical algorithms discretize the problem with respect to time. Yet, the major benefit of optimal control methods, over all the other motion planning algorithms mentioned above, is their ability to explicitly account for constraints of different type. In Chapter 2 we compare pros and cons of different numerical optimal control approaches, and review state-of-the-art algorithms and tools for solving optimal control problems.

1.2.5 Polynomial-based Methods

Polynomial-based motion planning methods are exclusively used for a class of dynamic systems called differentially flat systems. The special structure of systems with flatness property allows eliminating the state equations of the system, i.e., differential equations, by transforming the original motion planning problem into a prroblem in the space of a set of independent variables [VM98]. Polynomial-based methods parameterize each of these variables, known as differentially flat outputs, with a polynomial function [MMR02]. This reduces the problem of finding functions in an infinite dimensional space into an approximate one of finding a finite set of parameters, i.e. polynomial coefficients. Some earlier work has hypothesized that the motion planning problem for differentially flat systems can be solved in real-time [MRS95], [AF98], yet these studies

were all limited to planning in the absence of actuator and path constraints. In the presence of inequality constraints, polynomial parameterization of flat outputs converts the motion planning problem into a semi-infinite optimization problem involving a finite number of variables and an infinite number of constraints.

Different methods have been proposed to convert the resulting semi-infinite optimization problem into one that is computationally tractable. [VM98] is the first paper that addresses inequality constraints in the problem, and proposes different techniques for constructing a finite set of constraints that guarantees satisfaction of the original inequality constraints. More efficient methods for handling inequality constraints have been proposed using special types of polynomial basis functions. Parameterizing trajectories as B-spline ([MVP16], [VP17c]) and Bézier curves ([CCE08], [Cho+15], [Cic+17]), in particular, have gained popularity in motion planning as they can add an intuitive and geometric interpretation to the design while providing computational benefits to the problem. Overall, polynomial-based methods provide a reliable and efficient motion planning scheme for differentially flat systems, that is capable of generating trajectories consistent with vehicle dynamics while satisfying timing constraints of real-time application.

1.3 Contribution

1.3.1 Bézier parameterization and efficient evaluation of Inequality constraints

In this thesis, we leverage the unique properties of Bernstein polynomials and propose an efficient trajectory generation method for differentially flat systems. As mentioned before, polynomial-based methods reduce the dimensionality of the underlying optimization problem by expressing the flat outputs as polynomial functions that are fully described with a limited number of coefficients. In the presence of inequality constraints polynomial-based methods result in semi-infinite optimization problems. Such problems involve constraints that bound functions of a finite number of variables, i.e., the coefficients, over an entire time interval. Different approaches have been proposed to tackle the resulting semi-infinite optimization problem by replacing the constraints with an (approximate) finite set of constraints.

The work reported in [VM98] deals with linear inequality constraints, and proposes three methods for replacing them with a finite set of constraints on the polynomial coefficients. The resulting constraints can guarantee the satisfaction of the original inequality constraints at any instant of time in a given interval. The paper also suggests polytopic approximation of nonlinear inequalities so that they can be replaced by linear inequalities. The convex polytopic approximation can be done once (off-line) assuming that the constraints do not change in the course of a mission, however, inner approximation of nonlinear inequalities with linear inequalities might lead to overly conservative set of constraints.

Time gridding has been widely used in polynomial-based motion planning for converting the semi-infinite problem to a standard optimization problem [MK11], [RBR16], [CLS16]. In this method, inequality constraints are only evaluated on a finite set of time samples. The choice of these sample points is an open question. One common way is to choose evenly spaced points within the time interval. However, the satisfaction of constraints at some sampled points does not generally guarantee the satisfaction of constraints on the entire time interval. Using finer discretization can remedy this issue, but, a large number of grid points increases the number of constraints and the total computation time needed to solve the problem.

The convex hull property of B-spline and Bézier curves has been extensively used in the literature to circumvent the limitations of time gridding. Since a B-spline or Bézier curve is entirely contained within the convex hull of its control points, the collision-free requirements can be satisfied at all time instances (and not just at sampled points) by imposing separation between the convex hulls. To avoid collisions with static obstacles, [Gao+18], [Zho+19], [TLH19], [Pre+17], and [Tan+19] construct a convex decomposition of the free space and force the convex hull representing a trajectory to be inside it. [TH21] addresses dynamic obstacles and moving

agents by employing planes that separate the simplices enclosing the trajectories. Also, to reduce the conservatism, it uses MINVO basis [TH20] to obtain minimum volume simplices enclosing a polynomial curve.

The minimum distance between two Bézier curves can be computed to any desired accuracy using the efficient and robust algorithms in [Che+09], and [Cha+11]. In order to deal with inter-vehicle collision avoidance constraints, the work reported in [Cic+16] takes advantage of the algorithm in [Cha+11] and finds the control points of trajectories such that their minimum distance is greater than the safe distance that must be maintained between two vehicles. This method can eliminate the inherent conservatism of the above mentioned approaches that separate convex hulls. However, the major drawback is the substantial increase in computational cost due to the resulting non-smooth functions. [MVP17] exploits the basic properties of B-spline basis functions to convert a general inequality constraint on a B-spline curve to a finite set of constraints on its coefficients. The so-called B-spline relaxation can bring about a significant reduction of the computational effort in solving the optimization problem provided that all constraints are expressed in the form of B-splines. However, the resulting set of constraints might be conservative due to the gap between a curve and its B-spline coefficients. To reduce the conservatism, [MVP17] suggests representing the curves in a higher dimensional basis which also invoke an increase in the number of constraints. Therefore it's necessary to make a trade-off between the conservatism and the computational efficiency.

In Chapter 3, we use Bézier curves to parameterize trajectories, and we show that temporal and spatial collision avoidance constraints can be expressed as Bernstein-Bézier curves and surfaces accordingly. We then propose an efficient method for evaluating inequality constraints in Bernstein form. We employ a similar approach to [MVP17] and replace each inequality constraint on a Bernstein polynomial by a finite set of constraints on its control points. We also present an efficient method for finding closer control points to the polynomial curve which can be used to reduce the conservatism in the resulting set of constraints. We show that using Bernstein basis, instead of B-spline basis, allows refining the control polygon locally between any two points along the curve. We also present a criterion for deciding whether the refined control polygon is close enough to the actual curve.

1.3.2 Trajectory generation for drones in confined spaces using an ellipsoid model of the body

In this thesis we develop a trajectory generation method that would enable drones to fly through unstructured environments with narrow gaps and small spaces between obstacles. Finding safe and feasible trajectories for steering multiple drones towards some desired positions in tight spaces or guiding them through gaps smaller in width than their diameters is impossible without taking the drones' orientations into account. Existing motion planning approaches usually ignore the drone's orientation and real shape, and model the drone body as a sphere (or a circle in two-dimensional space), which allows constructing the collision-free space by merely inflating the obstacles with the radius of the sphere (see Fig. 1.4). As a result, the drone can be treated as a single point in the space. This approach simplifies the collision checking against obstacle, however, it is very conservative and yields infeasible problems when dealing with confined spaces (see Fig. 1.5).

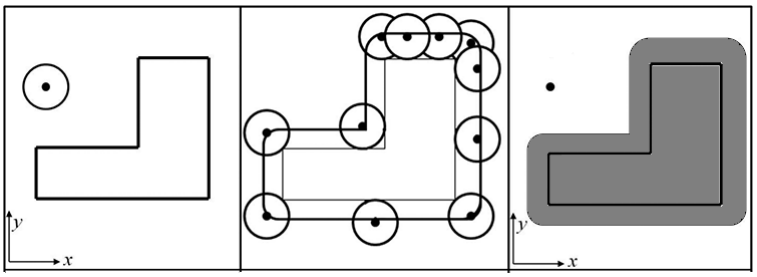


Figure 1.4: Existing motion planning approaches usually model the vehicle as a sphere or prism (**left**). Therefore, the configuration space (C-space) can be obtained merely by inflating the obstacles with the radius of the sphere. As a result, the vehicle can be treated as a single point in C-space and the collision checking is simplified (**right**).

One of the most common ways put forward to fly a drone through an inclined narrow gap while avoiding collisions with the gap frame is to directly constrain its attitude angles to specific values at the gap's position [HK14], [Loi+16]. In other words, the drone's orientation is aligned with that of the gap while traversing it. The proper attitude angles are determined using prior knowledge of the gap's pose and position, or estimates of them from detection algorithms with onboard sensing. A similar approach is used in [Fal+17] to fly a quadrotor through inclined gaps of arbitrary orientations with two-piece trajectories. The first segment of a trajectory enables state estimation via gap detection, while the second steers the quadrotor on the plane that is orthogonal to the gap and passes through its center. Forcing the quadrotor to fly on this plane minimizes the risk of collision with the gap frame, however this might be too conservative and lead to suboptimal trajectories with respect to a given objective function. Moreover, this approach is completely impractical for applications involving multiple drones.

Another way of considering the drone orientation while inspecting for collisions against obstacles is to approximate the drone body as an ellipsoid whose principal axes are aligned with the drone's body frame axes. The ellipsoid model can eliminate the inherent conservatism of the sphere model and allow for consideration of trajectories whose feasibility rely on the consideration of the body's attitude. Yet, as opposed to the symmetrical sphere model, there are no simple geometric constraints for identifying the collision-free space.

The ellipsoid model has been employed in a number of papers, however, in most of them the collision avoidance constraint is roughly evaluated by checking collisions between the ellipsoid and a sampled set of points from the obstacle. An example is the work reported in [Liu+18] that employs an ellipsoid model of the drone body to compute the flight attitude along sequences of motion primitives ([Liu+17b]). A primitive is then considered to be collision-free if the intersection of a subset of the point cloud, representing the obstacles, with the ellipsoid at some sampled states in time along the primitive is empty.

In this thesis, we use the separating hyperplane theorem of convex sets to derive constraints for collision avoidance between an ellipsoid-shaped body and an ellipsoid, sphere or, polygon shaped type of obstacle. More specifically, a moving hyperplane is designed such that the ellipsoid, i.e. the drone, is on one side of the separating plane and the obstacle on the other side. The resulting set of constraints can be seamlessly integrated into the Bézier curve-based method proposed in Chapter 3, to ensure that feasible collision-free trajectories for flying drones in confined spaces can be obtained under timing constraints of real-time applications. The proposed approach in this thesis will guarantee collision avoidance between two ellipsoids at every time instant, as against the approach in [Liu+18] whose performance is heavily dependent on the sampling points distribution.

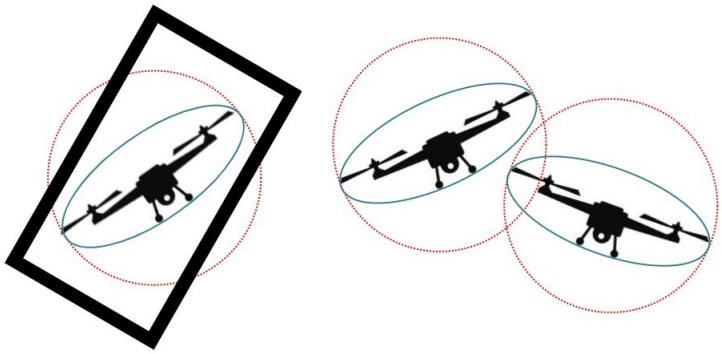


Figure 1.5: The conventional method of modeling the drone body as a sphere is not suitable for motion planning in tight spaces, as it fails to validate trajectories that might be feasible upon considering the drone's orientation. The less conservative ellipsoid model allows consideration for the 3D orientation.

1.3.3 Distributed trajectory generation framework

In this thesis, we present a distributed algorithm to generate collision-free trajectories for a group of quadrotors flying through a common workspace. Generating collision-free trajectories for multiple vehicles would require incorporating inter-vehicle collision avoidance constraints in the optimization problem. Thus, for a large group of vehicles, the optimization problem would involve a large number of constraints and decision variables, and the computational cost of solving it centrally can be prohibitively high. To reduce the computational complexity, a multitude of distributed schemes have been proposed for decomposing the optimization problem into smaller sub-problems that can be solved locally by each vehicle. The major challenge is to ensure that local decisions do also satisfy the coupling collision avoidance constraints. This is mainly addressed by exchanging information among the vehicles on their current states, future input sequences, etc. Depending on the communication strategy, the sub-problems might be solved sequentially or concurrently, with possibly several iterations of optimization and communication to achieve the required performance.

In sequential methods the vehicles solve their sub-problems successively [Kuw+07], [CHL10], [Ted+10]. In order to avoid inter-vehicle collisions, the generated trajectories of the preceding vehicles is used as constraints in the succeeding vehicles' sub-problems. These methods implicitly assign a priority level to the vehicles' objectives based on the sequence order. This issue has been addressed by cooperative distributed trajectory generation frameworks [OG15], [KH11], where vehicles solve their sub-problems in sequence while making modifications to neighboring vehicles' future plans. These methods require substantial communication between vehicles since each vehicle needs a full representation of its neighbors' decisions. Also, solving the vehicles' optimization sub-problems sequentially does not allow for high rates of re-planning trajectories.

In synchronous methods, the vehicles solve their optimization sub-problems simultaneously. The inclusion of inter-vehicle collision avoidance in these methods is not straightforward as vehicles are unaware of the future plans of their neighboring vehicles while solving their individual sub-problems. Different approaches have been proposed to ensure that locally generated trajectories satisfy the coupling inter-vehicle collision avoidance constraints [VLM08], [WD14], [Dai+17], [Zho+17], [VP17a]. These approaches require different types of information, such as vehicles' positions, velocities, or future input sequences, to be exchanged between vehicles. Therefore, while synchronous methods allow solving the optimization sub-problems in parallel, they are not all suitable for applications with limited communication.

In chapter 4, we develop a synchronous distributed trajectory generation framework with low computation and communication demands. We use the Voronoi diagram of the group of vehicles to decompose the space into non-overlapping regions. At each re-planning step, the

vehicles update their Voronoi cells according to the position information received from their neighbors. Each vehicle then generates its trajectory such that it is entirely within its Voronoi cell. Since the goal position may lie outside the vehicle's Voronoi cell, the vehicle finds the closest point of its Voronoi cell to the goal position which is used as the terminal condition in the optimization sub-problem. The vehicles re-plan their trajectories, in a receding horizon manner, until they reach the goal positions. In order to avoid long computational delays between updating the Voronoi cell and re-planning the trajectory, we also present an efficient method for finding the closest point of the Voronoi cell to the vehicle's goal position.

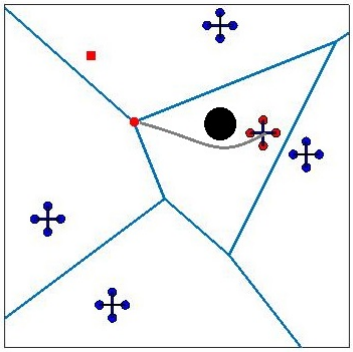


Figure 1.6: The 2D Voronoi diagram for 5 drones. The final position for the (red shaded) drone is shown with ■, and the closet point in its Voronoi cell to the goal position is shown with ●. The vehicle's trajectory is generated towards ● such that it is entirely within the vehicle's Voronoi cell and avoids the obstacle (black circle) inside the cell.

Chapter 2

Trajectory Generation using Computationally Efficient Optimal Control Methods

2.1 Introduction

With the advances in computing power and optimization algorithms, optimal control methods have become more popular in motion planning for autonomous vehicles. Using optimal control methods allows for objective functions and constraints of different types to be directly encoded in the problem. This together with their inherent multi-variable nature has made them a promising approach for solving multiple-vehicle trajectory generation problems. [LRG07], [Häu+12], [LDM15] are just a few examples of using optimal control in real-time (or near real-time) motion planning for single-vehicle or multi-vehicle applications.

Optimal control theory addresses the problem of finding the control input of a dynamical system that optimizes a given performance measure. The different approaches to solving continuous-time optimal control problems (OCPs) have been categorized into indirect and direct methods in [DG11]. The indirect optimal control methods date back to the calculus of variations and the classical work by Euler and Lagrange. However, substantial progress was only made in 1950s when Pontryagin maximum principle was formulated for general OCPs with inequality path constraints. The maximum principle describes the necessary optimality conditions for OCPs in continuous time. Indirect methods employ these conditions to derive a boundary value problem in ordinary differential equations, which is then discretized, by collocation or shooting techniques, for computing a numerical solution. Therefore, these methods are characterized as "first optimize-then discretize" [DG11]. Indirect methods have been very popular in space applications since the Sputnik and Apollo space missions.

Direct methods, described as "first discretize-then optimize", convert the original infinite-dimensional OCP to a finite-dimensional optimization problem. This is achieved by transforming the continuous-time dynamic system into a discrete-time system usually via numerical methods. The resulting optimization problem is then solved with standard optimization methods. All direct methods are based on one form of finite-dimensional parameterization of the control trajectory, however, they are different in the way they treat the state trajectory. Nowadays, direct methods are the most frequently used techniques for solving OCPs in real world applications mainly due to their capability to easily handle different types of constraints.

In this Chapter, after a brief description of the different numerical approaches for solving continuous-time OCPs, we study three of the direct methods- single shooting, multiple shooting, and collocation. We also look into the structure of the optimization problems resulting from each of these methods, which is exploited in order to obtain an efficient solver for the problem. Next, we consider three different case studies- go-to-formation maneuver, range-based AUV positioning, and autonomous drone cinematography- and formulate their multi-vehicle trajectory generation problems as OCPs. We then employ direct multiple shooting along with a structure-exploiting solver to generate optimal trajectories in different scenarios. The work presented in this Chapter does not develop a new optimal control technique; rather, it makes use of advances in numerical optimization algorithms and tools to efficiently solve constrained optimal trajectory generation problems in complex missions involving multiple vehicles.

2.2 Problem Description

The optimal trajectory generation problem that we address here consists of generating collision-free trajectories from the initial positions of the vehicles to some goal positions while optimizing a performance index. The generated trajectories should also satisfy different constraints including those imposed by the vehicles' dynamics and environmental conditions. This problem naturally translates into an optimal control problem. For a single vehicle the OCP can be formulated as

$$\min_{\mathbf{x}(.),\mathbf{u}(.),t_f} \int_0^{t_f} L(\mathbf{x}(t),\mathbf{u}(t))dt \tag{2.1}$$

$$s.t. \quad \dot{\mathbf{x}}(t) = f(\mathbf{x}(t),\mathbf{u}(t)) \tag{2.1a}$$

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{2.1b}$$

$$\mathbf{x}(t_f) = \mathbf{x}_f \tag{2.1c}$$

$$c_{\text{col}}(\mathbf{x}(t), \mathbf{x}_{\text{obst},i}(t)) \leq 0 \quad i \in \{1,\dots,N_{\text{obst}}\} \tag{2.1d}$$

$$\underline{\mathbf{x}} \leq \mathbf{x}(t) \leq \bar{\mathbf{x}} \tag{2.1e}$$

$$\underline{\mathbf{u}} \leq \mathbf{u}(t) \leq \bar{\mathbf{u}} \tag{2.1f}$$

In the above problem, (2.1) is the cost function to be minimized. Some common choices are the vehicle's travel time and the overall energy consumption. The ODE (2.1a) describes the vehicle's model where $\mathbf{x}$ and $\mathbf{u}$ are the state and input vectors of the model, respectively. The initial and the desired final positions of the vehicle are forced with the boundary constraints (2.1b) and (2.1c). The inequality (2.1d) is the collision avoidance constraint and (2.1e) and (2.1f) are the upper/lower bounds on the state and input. This problem may include other equality and inequality constraints. All constraints must hold for all $t \in [0, t_f]$. (See Chapter 4 for a multiple vehicle problem formulation.)

In the following, we adopt the notation and terminology of [DG11] to explain numerical approaches to solving the above continuous-time optimal control problems.

2.3 Numerical Methods for Optimal Control Problems

Different methods exist for solving a general OCP. In [DG11] these methods are classified into three main classes. The first class of methods utilizes the Bellman's principle of optimality which is the basis of dynamic programming in discrete time. In continuous time, however, it leads to a nonlinear partial differential equation (PDE) in the state space, the so-called Hamilton-Jacobi-Bellman (HJB) equation [Kir04]. For a simplified OCP with no inequality constraints, given by

$$\min_{\mathbf{x}(.),\mathbf{u}(.)} \int_0^{t_f} L(\mathbf{x}(t),\mathbf{u}(t))dt + E(\mathbf{x}(t_f)) \tag{2.2}$$

$$s.t. \quad \mathbf{x}(0) - \mathbf{x}_0 = 0 \tag{2.2a}$$

$$\dot{\mathbf{x}}(t) - f(\mathbf{x}(t),\mathbf{u}(t)) = 0, \qquad t \in [0, t_f] \tag{2.2b}$$

the HJB equation is obtained as

$$-\frac{\partial V}{\partial t}(\mathbf{x},t) = \min_u L(\mathbf{x},u) + \nabla_{\mathbf{x}} V(\mathbf{x},t)^T f(\mathbf{x},u). \tag{2.3}$$

The HJB equation (2.3) describes the time evolution of the value function $V(\mathbf{x},t)$, expressed as

$$V(\mathbf{x},t) = \min_{u_{[t,t_f]}} \int_t^{t_f} L(\mathbf{x}(s),\mathbf{u}(s))ds + E(\mathbf{x}(t_f)). \tag{2.4}$$

The PDE (2.3) contains partial derivatives of the value function with respect to t and $\mathbf{x}$, and should be solved backward for $t \in [0, t_f]$ with the boundary condition

$$V(\mathbf{x}(t_f), t_f) = E(\mathbf{x}(t_f)). \tag{2.5}$$

There are some special cases, such as linear quadratic problems, in which HJB equation takes a simpler form [Kir04], however, in general solving the HJB equation analytically can be a

challenging task. Yet, it provides necessary conditions for optimality, and thus, gives insights into the problem which can be exploited to find an approximate solution using numerical techniques. Like dynamic programming, solving HJB equation (2.3) suffers from the curse-of-dimensionality, and numerical solution of the PDE in large state dimensions gets computationally expensive. In addition, the HJB equation admits a solution only for a sufficiently smooth value function, while in many situations the value function is not differentiable with respect to state variables [DG11].

The second class encompasses the calculus of variations and the Euler-Lagrange differential equations, and also the so-called Pontryagin Maximum Principle which describes the necessary optimality conditions for a continuous-time optimal control problem. Using these conditions and solving the resulting boundary-value problem (BVP) numerically is called the indirect approach to optimal control. The necessary optimality conditions for the OCP (2.2) can be derived as [Kir04]

$$\mathbf{x}^*(0) = \bar{\mathbf{x}}_0 \qquad \text{(initial value)} \tag{2.6}$$

$$\dot{\mathbf{x}}^*(t) = f(\mathbf{x}^*(t), \mathbf{u}^*(t)) \quad t \in [0, t_f] \qquad \text{(ODE model)} \tag{2.6a}$$

$$\dot{\boldsymbol{\lambda}}^*(t) = -\nabla_{\mathbf{x}} H(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}^*(t)) \quad t \in [0, t_f] \qquad \text{(adjoint equations)} \tag{2.6b}$$

$$\mathbf{u}^*(t) = \arg\min_{\mathbf{u}} h(\mathbf{x}^*(t), \boldsymbol{\lambda}^*(t), \mathbf{u}(t)) \quad t \in [0, t_f] \qquad \text{(minimum principle)} \tag{2.6c}$$

$$\boldsymbol{\lambda}^*(t_f) = \nabla E(\mathbf{x}^*(t_f)) \qquad \text{(adjoint final value)} \tag{2.6d}$$

where H is the Hamiltonian function defined as

$$H(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}) = L(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T f(\mathbf{x}, \mathbf{u}). \tag{2.7}$$

The above necessary optimality conditions form a two points boundary value problem. These conditions can either be used to check if a given trajectory can be a possible solution or be solved numerically to obtain candidate solutions to an optimal control problem. It can be observed from (2.6) that the number and type of the conditions in (2.6) match the number and type of the unknowns: $\mathbf{u}^*(t)$ can be determined by the minimum principle, while $\mathbf{x}^*(t)$ and $\boldsymbol{\lambda}^*(t)$ are obtained from the ODE and the adjoint equations [DG11].

The indirect approach is an exact and elegant way to characterize and compute the optimal solution, yet it suffers a few major drawbacks; (a) the controls must be eliminated from the problem by algebraic manipulations, which is not always straightforward or might even be impossible, (b) optimal controls might be a discontinuous function of $\mathbf{x}$ and $\boldsymbol{\lambda}$, such that the BVP is possibly involving a non-smooth differential equation, and (c) the differential equation might become very nonlinear and unstable and not suitable for a forward simulation. All these issues can be partially resolved [ZNS20], [BB20].

In the third class, the direct approaches to optimal control, the problem is first discretized by parameterizing the infinite dimensional control trajectory and/or state trajectory, and then the resulting finite dimensional NLP is solved using the available numerical optimization methods. The main difference of the various direct methods is the way they parameterize the state trajectory. In the following we briefly review different direct methods, namely direct single shooting, direct multiple shooting and direct collocation methods.

2.3.1 Single Shooting Method

Direct single shooting method, first presented in [HR71] and [SS78], parameterizes the control input and uses an embedded ODE solver to eliminate the continuous time dynamic model of the system from the OCP. To tackle the general OCP with path constraints, expressed as

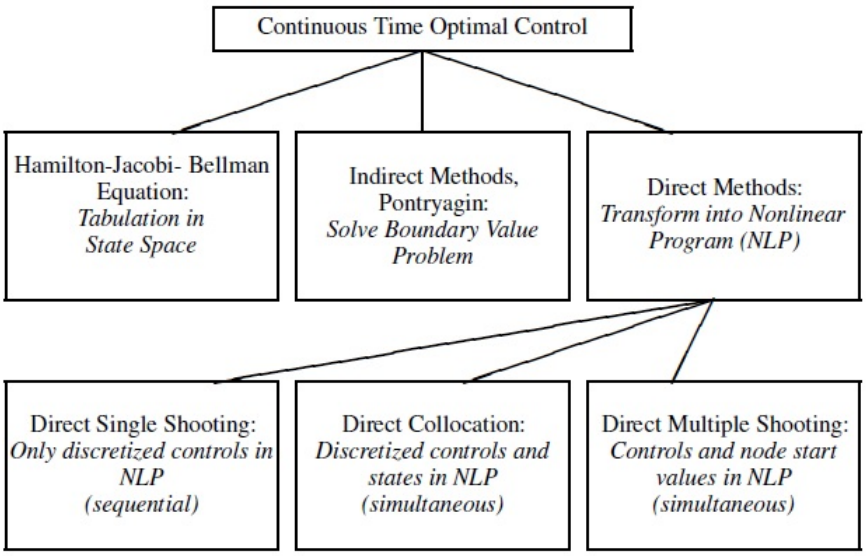


Figure 2.1: Numerical approaches to continuous time OCPs [DG11]

$$\min_{\mathbf{x}(.),\mathbf{u}(.)} \int_0^{t_f} L(\mathbf{x}(t),\mathbf{u}(t))dt + E(\mathbf{x}(t_f)) \tag{2.8}$$

$$s.t. \quad \mathbf{x}(0) = \mathbf{x}_0, \tag{2.8a}$$

$$\dot{\mathbf{x}}(t) - f(\mathbf{x}(t),\mathbf{u}(t)) = 0, \qquad t \in [0, t_f], \tag{2.8b}$$

$$h(\mathbf{x}(t),\mathbf{u}(t)) \leq 0, \qquad t \in [0, t_f], \tag{2.8c}$$

$$r(\mathbf{x}(t_f)) \leq 0, \tag{2.8d}$$

the control function $\mathbf{u}(t)$ is parameterized as a polynomial, a piecewise polynomial or, more commonly, a piecewise constant function, in which case $\mathbf{u}(t)$ is set as

$$\mathbf{u}(t) = \mathbf{q}_k, \qquad t \in [t_k, t_{k+1}], \quad k \in \mathbb{N}^+, \tag{2.9}$$

over a fixed time grid $0 = t_0 < t_1 < \cdots < t_N = t_f$ with N parameters $\mathbf{q}_k \in \mathbb{R}^{n_u}, k = 0, \ldots, N-1$. Using (2.9), the state trajectory $\mathbf{x}(t)$ over the interval $[0, t_f]$ is obtained by a forward integration of the dynamic system (2.8b), starting from $\mathbf{x}_0$. Thus, with single shooting method, the OCP is converted into the following NLP

$$\min_{\mathbf{q}} \int_0^{t_f} L(\mathbf{x}(t,\mathbf{q}),\mathbf{u}(t,\mathbf{q}))dt + E(\mathbf{x}(t_f,\mathbf{q})), \tag{2.10}$$

$$h(\mathbf{x}(t_k,\mathbf{q}),\mathbf{u}(t_k,\mathbf{q})) \leq 0, \qquad k = 0, \ldots, N, \tag{2.10a}$$

$$r(\mathbf{x}(t_f,\mathbf{q})) \leq 0, \tag{2.10b}$$

where $\mathbf{q} = [\mathbf{q}_0, \ldots, \mathbf{q}_{N-1}]^T \in \mathbb{R}^{Nn_u}$ is the optimization variable. In (2.10), the path constraints are discretized over the same time grid as used for the control input discretization. The above problem can usually be solved by a dense NLP solver. Fig. 2.2 shows an example of the state and input trajectories in single shooting method for a system with $n_x = 1$ and $n_u = 1$. Also, the sparsity pattern of the Hessian of the Lagrange function and the structure of the Jacobian of the inequality constraints are shown in Fig. 2.3.

A major problem with the single shooting method is that for a large simulation time the function $\mathbf{x}(t,\mathbf{q})$ obtained from the simulation of the nonlinear dynamics can be highly nonlinear

with respect to the control input $\mathbf{q}$ [DG11]. Hence, the constraints and cost function resulting from discretization via single shooting are highly nonlinear functions of $\mathbf{q}$. Since most NLP solvers employ successive linearizaion of the problem to find a candidate solution, having highly nonlinear functions in the NLP may invalidate the linear approximations outside a very small neighborhood around the linearization point. This issue is addressed in direct multiple shooting method by using short integration intervals.

Before explaining the direct multiple shooting method, it should be noted that while t_f in (2.8) is fixed, free final time problems can be treated similarly by rescaling the time interval to $[0, 1]$ and introducing the new decision variable t_f to the problem with the augmented system given by

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ t_f \end{bmatrix}, \qquad \dot{\tilde{\mathbf{x}}} = \frac{d}{d\tau}\tilde{\mathbf{x}} = \tilde{f}(\tilde{\mathbf{x}}, \mathbf{u}), \tag{2.11}$$

where $\tau = \frac{t}{t_f} \in [0, 1]$, and

$$\tilde{f}(\tilde{\mathbf{x}}, \mathbf{u}) = \begin{bmatrix} t_f . f(\mathbf{x}, \mathbf{u}) \\ 0 \end{bmatrix}. \tag{2.12}$$

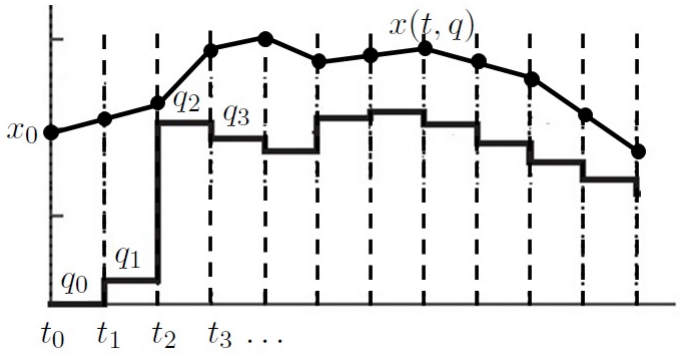


Figure 2.2: The state and input trajectories when solving an OCP with $n_x = 1$ and $n_u = 1$ via single shooting discretization.

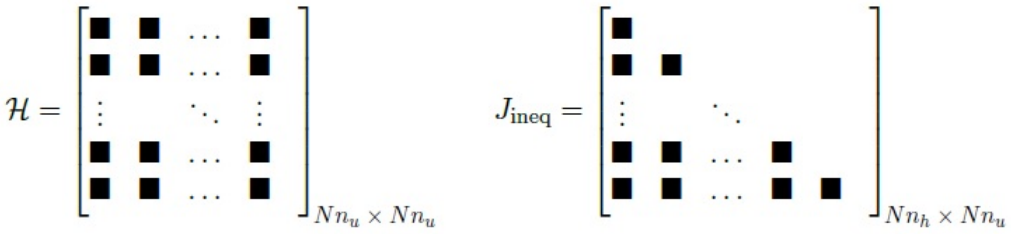


Figure 2.3: The sparsity pattern of the Hessian of the Lagrange function (**left**) and the sparsity pattern of the Jacobian of the inequality constraints (**right**) in the NLP resulting from single shooting discretization.

2.3.2 Direct Multiple Shooting Method

Direct multiple shooting method, developed originally by Bock and Plitt [Die+06], tackles the problem arising from long integration of dynamics in single shooting by limiting the integration over arbitrarily short time intervals. In order to convert the continuous OCP (2.8) into a NLP, direct multiple shooting discretizes the control input $\mathbf{u}(t)$ as well as the state $\mathbf{x}(t)$. Direct

multiple shooting first performs a finite-dimensional discretization of the continuous control input $\mathbf{u}(t)$, as in (2.9), and then solves the ODE (2.8.b), numerically, on each interval $[t_k, t_{k+1}]$ such that

$$\begin{aligned} &\dot{\mathbf{x}}_k(t, \mathbf{s}_k, \mathbf{q}_k) = f(\mathbf{x}_k(t, \mathbf{s}_k, \mathbf{q}_k), \mathbf{q}_k) \quad t \in [t_k, t_{k+1}], \\ &\mathbf{x}_k(t_k, \mathbf{s}_k, \mathbf{q}_k) = \mathbf{s}_k, \end{aligned} \tag{2.13}$$

where $\mathbf{s}_k$ is the state parameter at t_k, and $\mathbf{x}_k(t, \mathbf{s}_k, \mathbf{q}_k)$ is the state trajectory segment over the time interval $[t_k, t_{k+1}]$. Fig. 2.4 illustrates the discretization of the state and input via multiple shooting. The continuity of the state trajectory is ensured by the equality constraint

$$\mathbf{s}_{k+1} = \mathbf{x}_k(t_{k+1}, \mathbf{s}_k, \mathbf{q}_k), \quad k = 0, \dots, N. \tag{2.14}$$

Similarly to the single shooting method, the inequality path constraints are checked on the time grid used for the discretization of the control trajectory, i.e.,

$$h(\mathbf{s}_k, \mathbf{q}_k) \leq 0, \quad k = 0, \dots, N. \tag{2.15}$$

A finer sampling can also be used, provided that the embedded ODE solver returns some intermediate values within each time interval $[t_k, t_{k+1}]$. Using (2.14) and (2.15), the NLP resulting from discretization of the OCP (2.8) with multiple shooting method can be written as

$$\min_{\mathbf{s},\mathbf{q}} \sum_{k=0}^{N-1} l_k(\mathbf{s}_k, \mathbf{q}_k) + E(\mathbf{s}_N), \tag{2.16}$$

$$\mathbf{x}_0 - \mathbf{s}_0 = 0, \tag{2.16a}$$

$$\mathbf{x}_k(t_{k+1}, \mathbf{s}_k, \mathbf{q}_k) - \mathbf{s}_{k+1} = 0 \qquad k = 0, \dots, N-1, \tag{2.16b}$$

$$h(\mathbf{s}_k, \mathbf{q}_k) \leq 0 \qquad k = 0, \dots, N, \tag{2.16c}$$

$$r(\mathbf{s}_N) \leq 0 \tag{2.16d}$$

with each $l_k(\mathbf{s}_k, \mathbf{q}_k)$ being computed numerically on the time interval $[t_k, t_{k+1}]$ as

$$l_k(\mathbf{s}_k, \mathbf{q}_k) = \int_{t_k}^{t_{k+1}} L_i(\mathbf{x}_k(t, \mathbf{s}_k, \mathbf{q}_k), \mathbf{q}_k) dt. \tag{2.17}$$

The NLP (2.16) can be solved with any standard nonlinear programming algorithm. The key to efficiently solving this problem is to use the structured sparsity patterns of its Jacobian and the Hessian matrices. Figure 2.5 shows the sparsity pattern of Hessian and Jacobian matrices of the NLP (2.16). If the optimization variables are ordered as $\mathbf{z} = (\mathbf{s}_0^T, \mathbf{q}_0^T, \mathbf{s}_1^T, \mathbf{q}_1^T, \dots, \mathbf{s}_{N-1}^T, \mathbf{q}_{N-1}^T, \mathbf{s}_N^T)^T$ then he Jacobian of the equality constraints in NLP (2.16) is a block banded matrix obtained as

$$J_{\text{eq},k} = \begin{bmatrix} \frac{\partial \mathbf{x}_k(\mathbf{s}_k, \mathbf{q}_k)}{\partial \mathbf{s}_k} & \frac{\partial \mathbf{x}_k(\mathbf{s}_k, \mathbf{q}_k)}{\partial \mathbf{q}_k} & -I \end{bmatrix}, \quad k = 0, \dots, N-1. \tag{2.18}$$

Also, due to the separability of the Lagrange function, the Hessian of the NLP (2.16) is a block diagonal matrix computed as

$$\mathcal{H}_k = \begin{bmatrix} \frac{\partial^2 \mathcal{L}_k(\mathbf{s}_k, \mathbf{q}_k)}{\partial \mathbf{s}_k^2} & \frac{\partial^2 \mathcal{L}_k(\mathbf{s}_k, \mathbf{q}_k)}{\partial \mathbf{s}_k \partial \mathbf{q}_k} \\ \frac{\partial^2 \mathcal{L}_k(\mathbf{s}_k, \mathbf{q}_k)}{\partial \mathbf{q}_k \partial \mathbf{s}_k} & \frac{\partial^2 \mathcal{L}_k(\mathbf{s}_k, \mathbf{q}_k)}{\partial \mathbf{q}_k^2} \end{bmatrix} \quad k = 0, \dots, N-1 \tag{2.19}$$

where $\mathcal{L}(\mathbf{s}, \mathbf{q})$ is the Lagrange function. For a brief explanation of how this special structure can be exploited to reduce the computational complexity of solving the NLP (2.16) the reader is referred to Appendix A.

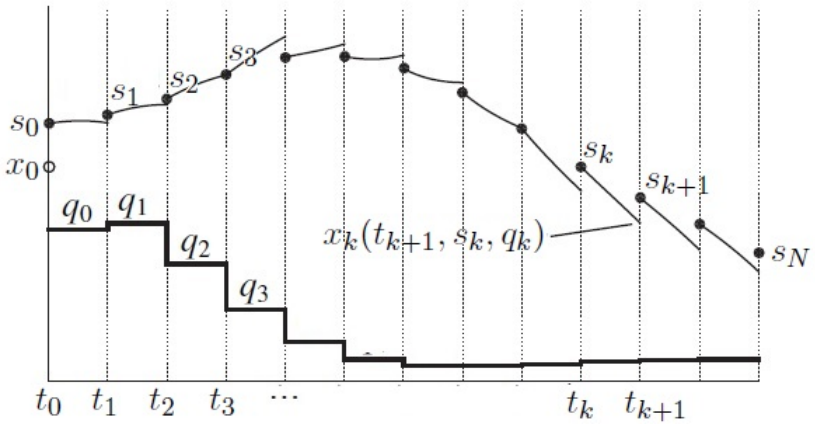


Figure 2.4: Illustration of the direct multiple shooting method at an early iteration before the constraints are satisfied [DG11]. The continuity condition $\mathbf{x}_k(t_{k+1}, \mathbf{s}_k, \mathbf{q}_k) - \mathbf{s}_{k+1} = 0$ is enforced at $\mathbf{s}_0, \ldots, \mathbf{s}_{N-1}$. The state trajectory becomes continuous once the solution of the NLP is achieved.

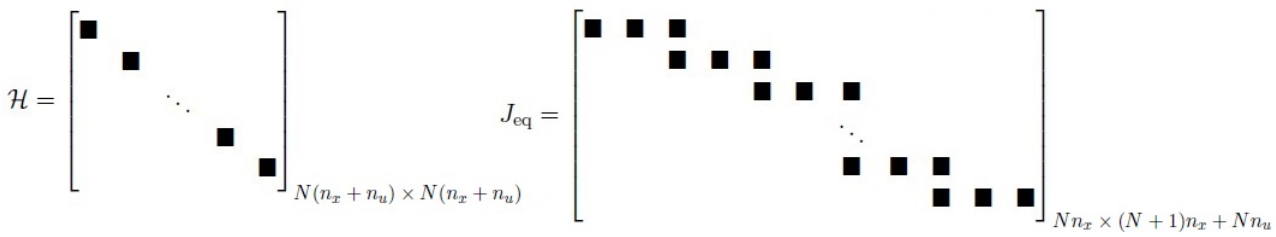


Figure 2.5: The sparsity pattern of the Hessian of the Lagrange function (**left**) and the sparsity pattern of the Jacobian of the equality constraints (**right**) in the NLP resulting from multiple shooting disretization. The variables are ordered as $\mathbf{z} = (\mathbf{s}_0, \mathbf{q}_0, \mathbf{s}_1, \mathbf{q}_1, \ldots, \mathbf{s}_{N-1}, \mathbf{q}_{N-1}, \mathbf{s}_N)^T$.

2.3.3 Direct Collocation Method

In the direct collocation method, the control and the state trajectories are both discretized on a fixed time grid, and the state trajectory between two consecutive grid points is approximated by a polynomial, e.g. a Lagrange polynomial. Using a set of collocation times $t_{k,i} \in [t_k, t_{k+1}], i = 0, \ldots, d$, $t_{k,0} = t_k$, the polynomial for the time interval $[t_k, t_{k+1}]$ is expressed as

$$p_k(t, \mathbf{v}_k) = \sum_{i=0}^{d} \mathbf{v}_{k,i} \ell_{k,i}(t) \tag{2.20}$$

where $\mathbf{v}_k = [\mathbf{v}_{k,0}, \ldots, \mathbf{v}_{k,d}] \in \mathbb{R}^{n_x \times (d+1)}$ is the matrix of coefficients and ℓ_k is the basis polynomial. The basis polynomials are typically chosen such that

$$\ell_{k,i}(t_{k,j}) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases} \tag{2.21}$$

and thus, the polynomial passes through the interpolation points $\mathbf{v}_{k,i}$, i.e.,

$$p_k(t_{k,i}, \mathbf{v}_k) = \mathbf{v}_{k,i}, \quad i = 0, \ldots, d. \tag{2.22}$$

Since the polynomial must coincide with the state at the beginning of the interval, $\mathbf{v}_{k,0}$ must satisfy

$$\mathbf{v}_{k,0} = p_k(t_{k,0}, \mathbf{v}_k) = p_k(t_k, \mathbf{v}_k) = \mathbf{s}_k. \tag{2.23}$$

where $\mathbf{s}_k$ is the discrete state at the grid point t_k. Also, the polynomial must satisfy the dynamics on the collocation points, i.e.

$$\dot{p}_k(t_{k,i}, \mathbf{v}_k) = f(p_k(t_{k,i}, \mathbf{v}_k), \mathbf{q}_k) = f(\mathbf{v}_{k,i}, \mathbf{q}_k) \tag{2.24}$$

In (2.24) the control input is assumed to be piece-wise constant, i.e., $\mathbf{u}(t) = \mathbf{q}_k$ for $[t_k, t_{k+1}]$. From (2.23) and (2.24), the integration of the system dynamics over the time interval $[t_k, t_{k+1}]$ is performed by solving the collocation equations, expressed as

$$c_k(\mathbf{v}_k, \mathbf{s}_k, \mathbf{q}_k) = \begin{bmatrix} \mathbf{v}_{k,0} - \mathbf{s}_k \\ \dot{p}_k(t_{k,1}, \mathbf{v}_k) - f(\mathbf{v}_{k,1}, \mathbf{q}_k) \\ \vdots \\ \dot{p}_k(t_{k,d}, \mathbf{v}_k) - f(\mathbf{v}_{k,d}, \mathbf{q}_k) \end{bmatrix} = 0. \tag{2.25}$$

Fig. 2.6 illustrates the interpolation polynomial and points over one time interval for $d = 3$. The continuity on the interval boundary is enforced by

$$p_k(t_{k+1}, \mathbf{v}_k) - \mathbf{s}_{k+1} = 0, \tag{2.26}$$

The path constraints (2.8d) can be imposed on the grid points t_k or on the fine grid provide by the collocation times $t_{k,i}$ as

$$h(\mathbf{v}_{k,i}, \mathbf{q}_k) \leq 0 \quad k = 0, \ldots, N \quad i = 0, \ldots, d \tag{2.27}$$

The state variables $\mathbf{s}_k$ can be eliminated form the decision variables using the linear equality constraint in (2.23). Thus, the NLP obtained from discretizing (2.8) with direct collocation method can be written as

$$\min_{\mathbf{v},\mathbf{q}} \sum_{k=0}^{N-1} l_k(\mathbf{v}_k, \mathbf{q}_k) + E(\mathbf{v}_{N,0}), \tag{2.28}$$

$$\mathbf{v}_{0,0} - \mathbf{x}_0 = 0, \tag{2.28a}$$

$$\dot{p}(t_{k,i}, \mathbf{v}_k) - f(\mathbf{v}_{k,i}, \mathbf{q}_k) = 0, \qquad k = 0, \dots, N-1, \quad i = 1, \dots, d, \tag{2.28b}$$

$$p_k(t_{k+1}, \mathbf{v}_k) - \mathbf{v}_{k+1,0} = 0, \qquad k = 0, \dots, N-1, \tag{2.28c}$$

$$h(\mathbf{v}_{k,0}, \mathbf{q}_k) \leq 0, \qquad k = 0, \dots, N, \tag{2.28d}$$

$$r(\mathbf{v}_{N,0}) \leq 0. \tag{2.28e}$$

Direct collocation method provides a natural way for improving the numerical accuracy of the integration by increasing d. However, in practice choosing $d > 4$ can be counterproductive for solving complex OCPs, mainly because increasing d can worsen the conditioning of linear algebra underlying the NLP (2.28) [DG11]. Also, direct collocation method allows refining the parameterization of the continuous control input $\mathbf{u}(t)$. The collocation equations (2.25) is based on the standard piece-wise constant input parameterization which enforces a unique discrete input value $\mathbf{q}_k$ over the entire interval $[t_k, t_{k+1}]$. However, more degrees of freedom in the discrete input can be added by using a different input for each collocation time $t_{k,i}, \dots, t_{k,d}$, in which case the collocation equations can be re-written as

$$c_k(\mathbf{v}_k, \mathbf{s}_k, \mathbf{q}_k) = \begin{bmatrix} \mathbf{v}_{k,0} - \mathbf{s}_k \\ \dot{p}_k(t_{k,1}, \mathbf{v}_k) - f(\mathbf{v}_{k,1}, \mathbf{q}_{k,1}) \\ \vdots \\ \dot{p}_k(t_{k,d}, \mathbf{v}_k) - f(\mathbf{v}_{k,d}, \mathbf{q}_{k,d}) \end{bmatrix} = 0. \tag{2.29}$$

Accordingly, the optimization variables of the NLP are modified as

$$\mathbf{z} = (\mathbf{v}_{0,0}, \mathbf{v}_{0,1}, \mathbf{q}_{0,1}, \dots \mathbf{v}_{0,d}, \mathbf{q}_{0,d}, \mathbf{v}_{1,0}, \mathbf{v}_{1,1}, \mathbf{q}_{1,1}, \dots, \mathbf{v}_{1,d}, \mathbf{q}_{1,d}, \dots) \tag{2.30}$$

Using either (2.25) or (2.29), direct collocation method yields a large-scale but sparse NLP. Fig. 2.7 shows the sparsity pattern in the equality constraints Jacobian matrix of the resulting NLP for $d = 3$. One of the most important variants of orthogonal collocation methods is the pseudo-spectral optimal control method [Gar+17]. Pseudo-spectral methods use only one collocation interval but a set of high order basis polynomials for approximation over the entire interval. The Legendre Pseudo-spectral method in [EKR95], as an example, uses Lagrange polynomials and Gauss-Lobatto quadrature rule to discretize the problem, and incorporates the path constraints in the NLP by enforcing them at the LGL nodes. It has been shown that the solution obtained by pseudo-spectral methods, contrary to most other direct methods, satisfies the necessary optimality conditions. However, the Jacobian and Hessian matrices in the resulting NLP are dense and usually more expensive to factorize than those in direct collocation [DG11].

2.3.4 Available tools for solving OCPs and NLPs

As explained in the previous section, direct optimal control approaches parameterize the control and/or state trajectories and approximate the original infinite-dimensional problem by a finite-dimensional NLP. Using a reliable and efficient NLP solver is therefore necessary for a successful implementation of these approaches. Here, we briefly review a number of tools available for solving OCPs and NLPs.

DIDO, powered by the pseudo-spectral optimal control theory [RF03], was first became known for its demonstration in the globally optimal maneuver of the International Space Station in 2006 [Bed+07], and has been since used to solve optimal control problems in aerospace,

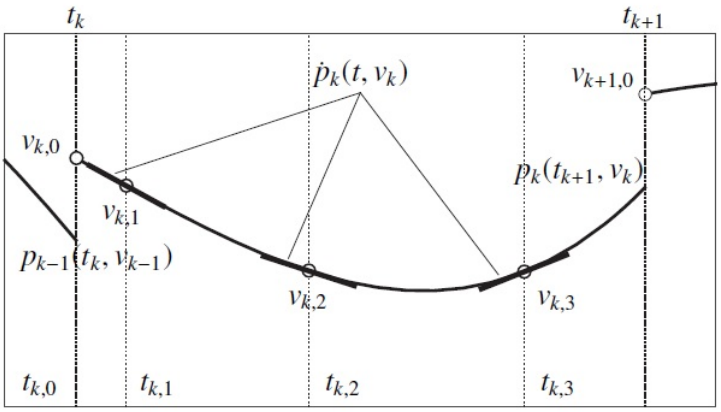


Figure 2.6: Illustration of the direct collocation method with $d = 3$ at the time interval $[t_k, t_{k+1}]$ [DG11].

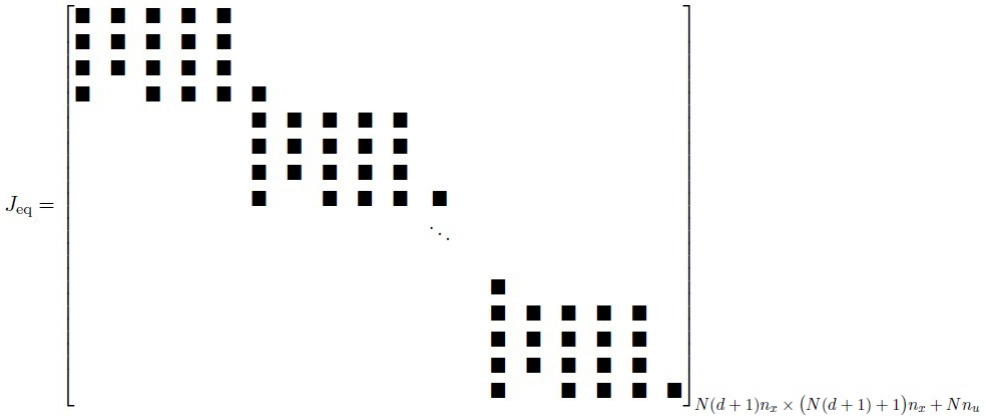


Figure 2.7: The sparsity pattern of the sparsity pattern of the Jacobian of the equality constraints in the NLP resulting from direct collocation with $d = 3$. The variables are ordered as $\mathbf{z} = (\mathbf{v}_{0,0}^T, \mathbf{q}_0^T, \mathbf{v}_{0,1}^T, \mathbf{v}_{0,2}^T, \mathbf{v}_{0,3}^T, \mathbf{v}_{1,0}^T, \mathbf{q}_1^T, \mathbf{v}_{1,1}^T, \dots)^T$

robotics, biotech and more [Ros20]. Since its emergence, DIDO has evolved from its simple use of NLP solvers such as SNOPT [GMS05] to a multifaceted MATLAB package that outputs, other than the optimal state and control, the Hamiltonian, costates, and path covectors. These information can help verify numerical results when used alongside classical tools such as Pontryagin's principle. One advantage of DIDO over other solvers is that it does not require an initial guess of the solution.

GPOPS-II [PR14] is a software based on a class of variable-order Gaussian quadrature collocation methods for solving general OCPs. It supports the NLP solver SNOPT and IPOPT [WB06] to solve the resulting large but sparse NLP. The enhancements to GPOPS-II, including varying polynomial degree and adaptive placement of collocation points, has made it capable of solving a wider range of OCPs as compared to its previous version GPOPS [Rao+10]. ICLOCS2, the new version of ICLOCS, [NFK18] is an optimal control software that is equipped with both direct multiple shooting and direct collocation to transcribe OCPs into NLPs. ICLOCS supports a number of optimization software packages, such as IPOPT, MATLAB fmincon, and WORHP [BW12], to solve the resulting (large-scale) NLPs. While previous versions of ICLOCS often led to long computation times, ICLOCS2 is designed to enable flexible trade-offs between accuracy level and computational effort.

ACADO is a software environment and algorithm collection for automatic control and dynamic optimization [HFD11]. It provides a framework for using RTI scheme [DBS05], which is based on the direct multiple shooting method, to solve nonlinear optimal control problems. The RTI scheme yields a large but sparse quadratic programming (QP) which can be condensed and solved using dense linear algebra QP solver qpOASES [Fer+14] (ACADO's default QP solver) or can be alternatively solved with structure exploiting QP solvers. A comparison of the computational complexity of these two approaches is given in [Vuk+13]. The RTI-based solver in ACADO is designed to provide approximate solutions within short computation times. Also, ACADO has automoatic code generation functionalities which allows exporting optimized and efficient C-code for solving nonlinear OCPs. These features have made ACADO suitable for real-time optimal control implementations.

FORCESPro [Zan+17] is a software package that is specifically designed for the purpose of fast embedded optimization. It consists of a code generation engine capable of providing efficient, statically-allocated, standalone C code with a small memory footprint. FORCESPro can generate codes that are deployable on any embedded platform with a C compiler. However, with the free academic license, a generated code can only be run on a specific desktop platform. FORCESPro has several methods available for solving different types of optmization problems. Its NLP solver is an extension of the previously existing primal-dual interior-point solver for convex QCQPs [Dom+12] and is developed specifically for the mathematical structure of optimal control problems. By exploiting the structure of the problem, the customized solver generated by FORCESPro can perform orders of magnitude faster than a generic NLP solver.

Among several approaches available for solving the OCP (2.1), here we use direct multiple shooting method. Also, based on a detailed comparison between the computation times of different OCP solvers and NLP interfaces for solving several trajectory generation problems, FORCES Pro is our solver of choice for NLPs arising in direct multiple shooting method. FORCESPro allows generating tailored solvers, for a variety of optimization problems such as LPs, QPs and NLPs, from a high-level mathematical description of the problem. Besides optimization variables and constraints, the problem description in FORCESPro client software may include some parameters that can change dynamically during runtime. After the problem is described in the supported form, the client software communicates with the server for code generation and compilation. Once the solver is generated, it can be used as many times as needed to solve different instances of the same optimization problem. This is particularly useful when a problem has to be solved with varying data in real-time (or near real-time).

FORCESPro supports designing solvers via Python and MATLAB scripts. The default

automatic differntiation tool for both Python and MATLAB client is CasADi [AÅD12], which must be added to the PYTHONPATH (or MATLAB path). More details can be found in Appendix , where a simple trajectory generation problem is described in the FORCESPro high-level python interface.

2.4 Simulation Results

In this section we employ direct multiple shooting method to solve different trajectory generation problems. The examples we consider here include go-to-formation maneuver and range-based positioning for multiple AUVs, and autonomous cinematography with multiple drones. We first formulate each problem as an optimal control problem, and then we use FORCES Pro to solve the NLPs resulting from direct multiple shooting. FORCES Pro exploits the special structure of the NLP (See Fig. 2.5) to deliver a customized solver for the problem. We use the generated solver in different instances of the problem, and record the average computation time. The computation times presented below are all obtained on a desktop computer with a 2.60 GHz i7-4510U CPU and 6.00 GB RAM.

In the simulation results presented below, all AUVs are described by the simplified model

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ a \\ \omega \\ r \end{bmatrix}, \tag{2.31}$$

where $\mathbf{p} = [x, y]^T \in \mathbb{R}^2$ is the inertial position, v is the body-speed, ψ is the course angle and ω is the course angle rate. The system input vector $\mathbf{u} = [a, r]^T \in \mathbb{R}^2$ consists of the linear and angular accelerations of the vehicle.

To guarantee that generated trajectories are dynamically feasible, the state and the input of the system must be within given bounds. These constraints are expressed as inequalities of the form

$$\begin{aligned} v_{\min} \leq v(t) \leq v_{\max} \\ a_{\min} \leq a(t) \leq a_{\max} \\ \psi_{\min} \leq \psi(t) \leq \psi_{\max} \\ \omega_{\min} \leq \omega(t) \leq \omega_{\max} \\ r_{\min} \leq r(t) \leq r_{\max} \end{aligned} \tag{2.32}$$

which must hold for all $t \in [0, t_f]$. The lower and upper bounds are given in Table 2.1.

To make sure vehicles maintain a safe distance to obstacles and other vehicles during the mission, collision-avoidance constraints must also be incorporated in the problem. As per the requirements, the constraints may impose temporal or spatial separation between trajectories. The temporal collision avoidance constraint between the i-th and the j-th vehicles can be expressed as

$$c_{\text{col}}\big(\mathbf{p}_i(t), \mathbf{p}_j(t)\big) = \frac{\big(x_i(t) - x_j(t)\big)^2}{r_{ij}^2} + \frac{\big(y_i(t) - y_j(t)\big)^2}{r_{ij}^2} - 1 \geq 0. \quad t \in [0, t_f] \tag{2.33}$$

where r_{ij} is the safe distance to be kept between the i-th and the j-th vehicles. Similar constraints can be added to enforce vehicle-obstacle collision avoidance. As explained before, in

Table 2.1: Upper/lower bounds on states and inputs

	minimum value	maximum value
$v(\frac{\mathrm{m}}{\mathrm{s}})$	0.1	0.7
$\psi(\mathrm{rad})$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$
$a(\frac{\mathrm{m}}{\mathrm{s}^2})$	-0.1	0.1
$\omega(\frac{\mathrm{rad}}{\mathrm{s}})$	-0.349	0.349
$r(\frac{\mathrm{rad}}{\mathrm{s}^2})$	-0.1	0.1

direct multiple shooting method the constraints are evaluated on the grid points used for control discretization. For N shooting intervals, (2.33) translates into N inequality constraints in the problem per each two vehicles. For spatial separation this number would be N^2. For longer distance problems where more shooting intervals is necessary the number of optimization variables and constraints in the problem will grow substantially. Therefore, using efficient and reliable optimization methods and tools is the key to successful implementation of the method.

2.4.1 Go-to-Formation Maneuver of 7 AUVs

In the first example we consider a group of AUVs performing a cooperative mission in a particular formation. In real-world conditions the AUVs are launched from two or more support vessels at different time instances and different positions. Therefore, the initial formation might be completely different from the desired one. In order to start the mission it is necessary to have a set of trajectories that guide the fleet to the desired positions. These trajectories should guarantee simultaneous arrival of the AUVs to the goal positions with desired speeds and orientations. Also, they should minimize a cost function (travel time, energy consumption, ...) while ensuring that the inter-vehicle collisions are avoided.

Due to the environmental conditions at sea and the vehicles' lack of hovering capability, the vehicles may turn or shift position while the optimization problem is being solved. Therefore, the vehicles' positions at the time of executing the trajectory might be different from the position information provided to the solver. The long computational delay between obtaining the initial state information and generating the trajectories can lead to a drastic decrease in performance or even infeasibility of the generated trajectories. Hence, it is imperative to generate the trajectories in a short time following receipt of the position information before the waves and currents scatter the AUVs around.

Fig. 2.8 shows a 7-vehicle problem where the AUVs should reach the mission starting positions, shown with x, with minimum energy usage. Here we pass over the "actual" power consumption from the AUV's batteries and simply define the cost function in the underlying OCP by the integral of a weighted quadratic function of the system control inputs as

$$J = \int_0^{t_f} \mathbf{u}^T \mathrm{Q} \mathbf{u} \; dt \tag{2.34}$$

where Q is the weighting matrix. Given the vehicles' initial positions, shown with x, the goal is to find spatially de-conflicted trajectories such that the AUVs reach their corresponding target positions at the same time, with given speeds and course angles. Figure 2.8 shows the results

for the case where the AUVs should keep a constant speed of $0.5\frac{m}{s}$ along the trajectories. The course angle, course rate, and the input profile for the 7 AUVs are shown in Fig. 2.9.

In this example, the OCP is solved using 90 shooting intervals. The generated solver is used to solve multiple instances of the problem with different initial states. The average computation time using the FORCES Pro interior-point NLP solver is around 16 seconds.

Fig. 2.10 shows the case where the AUVs are allowed to move with varying speed bounded with the minimum and maximum values given in Table 2.1. The AUVs start with a speed of $0.1\frac{m}{s}$ and should reach the target positions with a speed of $0.5\frac{m}{s}$. Fig. 2.11 shows that the speed and course angle of the 7 vehicles are within the required bounds. The average computation time for solving this problem with different initial and final conditions is 23 seconds.

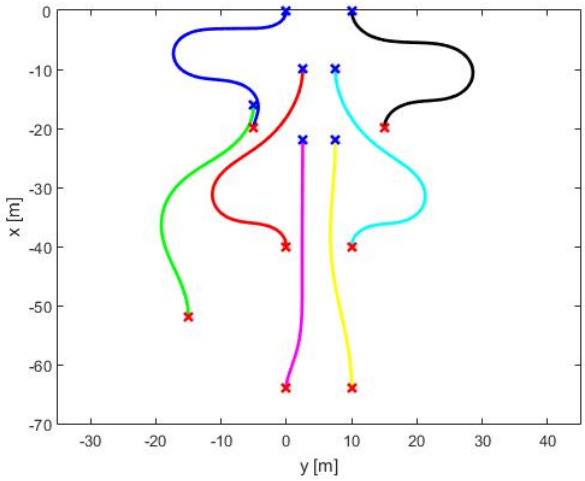


Figure 2.8: Spatially de-conflicted trajectories for 7 vehicles moving from their initial positions (x) to their final positions (x) with a constant speed of $0.5\frac{m}{s}$ and $\psi_0 = \psi_f = 0$.

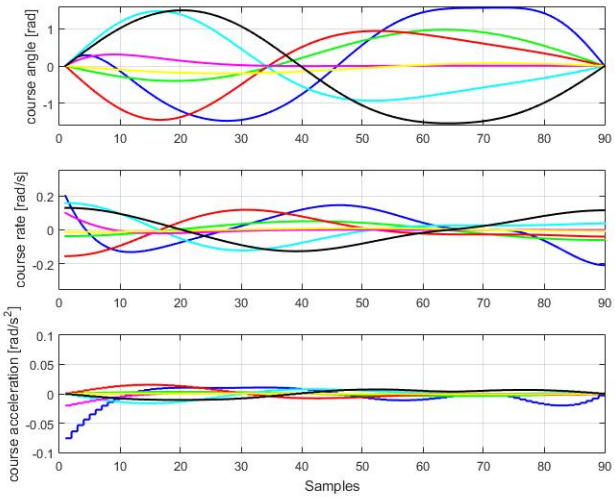


Figure 2.9: The course angle, course rate and angular acceleration of the 7 vehicles are within the given bounds in Table 2.1.

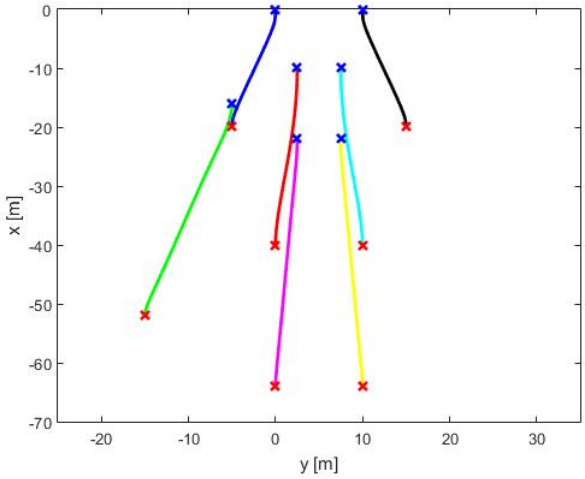


Figure 2.10: Spatially de-conflicted trajectories for 7 vehicles from their initial positions (x) to their final positions (x), with the boundary conditions, $v_0 = 0.1$m/s, $v_f = 0.5$m/s and $\psi_0 = \psi_f = 0$.

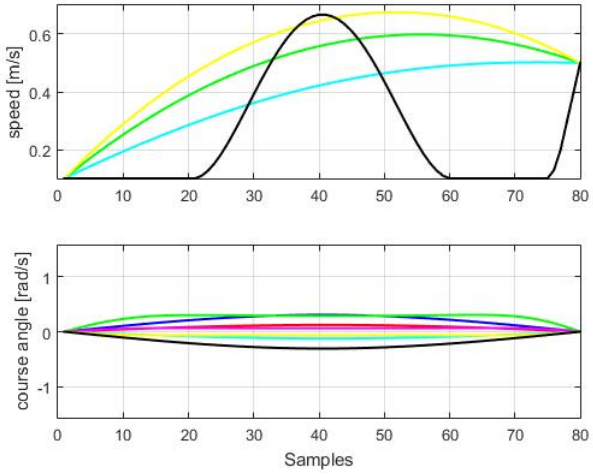


Figure 2.11: The speed and course angle of the 7 vehicles are within the given bounds in Table 2.1.

2.4.2 Minimum time maneuver with collision avoidance

In the second example we consider two vehicles in a cluttered environment (Fig. 2.12). The goal is to generate temporally separated trajectories that minimize the vehicles' travel time, i.e., $J = t_f$. The trajectories should guide the vehicles from the initial positions to the final positions while avoiding static obstacles with known positions. Figure 2.12 shows the trajectories generated with 80 shooting intervals, i.e., $N = 80$. These trajectories ensure that the inter-vehicle and vehicle-obstacle collision avoidance constraints are satisfied for the entire travel time. The speed and the course angle profiles for the two vehicles are shown in Fig. 2.13. The average computation time for solving the same problem with different initial state conditions is 1.59 seconds.

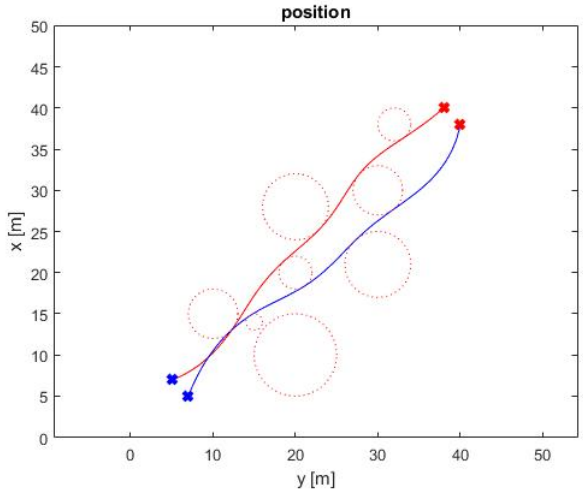


Figure 2.12: Temporally de-conflicted trajectories for two vehicles moving from their initial positions (x) to their final positions (x) in a cluttered environment.

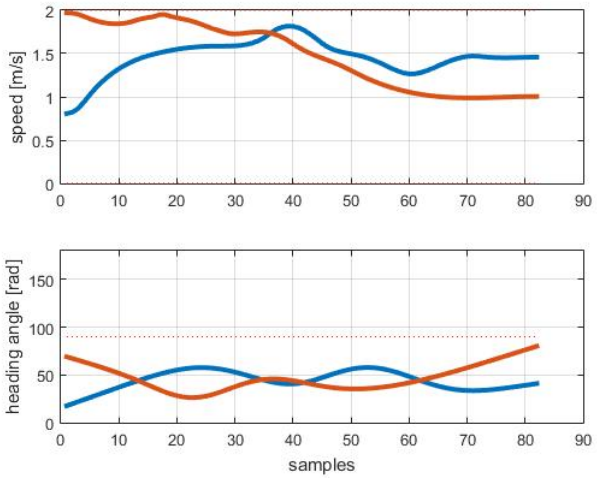


Figure 2.13: The course angle and speed profile for the two vehicles in Figure 2.12

2.4.3 Trajectory optimization for range-based AUV positioning

In the third example, we consider the trajectory generation problem for range-based AUV positioning. In this problem, the AUV's position is estimated using a set of range measurements

from the vehicle to one or more beacons with known positions. In the case of a single beacon, the measurements are obtained as

$$z_k = d_k + w_k \tag{2.35}$$

where $d_k = \|\mathbf{p}_k - \mathbf{p}_b\|$ is the true distance between the AUV and the beacon at time t_k, with $\mathbf{p}_k = [x_k, y_k]^T$ and $\mathbf{p}_b = [p_{b,x}, p_{b,y}]^T$ respectively denoting the position of the AUV and the beacon at time t_k and $w_k \sim \mathcal{N}(0, \sigma^2)$ is additive white Gaussian noise. The goal is to obtain the sequence of measurements that in a well defined sense maximize the information available for AUV positioning by resorting to properly designed position estimation algorithms. This can be done by including directly in the objective function of the NLP a performance measure that quantifies the amount of information available for positioning.

One way to quantify the best possible estimator performance obtained with a given set of data is to use the Cramer-Rao (CR) lower bound in the context of estimation theory. The Cramer-Rao Lower Bound (CRLB) provides a lower bound on the covariance of the estimates that can be achieved with any unbiased estimator. For the unknown deterministic parameter $\theta \in \mathbb{R}^2$, denoting the initial position of the AUV and being estimated from the measurements $Z = (z_0, z_1, \ldots, z_{m-1})^T$, the variance of the unbiased estimator $\hat{\theta}$ is bounded from below by

$$\mathrm{Var}(\hat{\theta}) \geq \frac{1}{mI(\theta)} \tag{2.36}$$

where $I(\theta)$ is the Fisher Information Matrix (FIM) defined as

$$I(\theta) \stackrel{\text{def}}{=} E([\nabla_\theta(\log \mathcal{L}_\theta(Z))][\nabla_\theta(\log \mathcal{L}_\theta(Z))]^T) \tag{2.37}$$

with $\mathcal{L}_\theta(Z)$ being the likelihood function of the measurements with respect to θ. The FIM gives a measure of the information content provided by the set of measurements Z. Therefore, it can be used to define a suitable scalar function for the OCP (2.8). For the range-based localization problem described above, the determinant of the FIM can serve as the objective function of choice. Considering the AUV dynamics and the measurement model (2.35), |FIM| for a "single vehicle-single beacon" problem is obtained as [Bis+10]

$$|\mathrm{FIM}| = \frac{1}{\sigma^4} \sum_{\substack{k,l \\ k>l}} \left((\frac{x_k - p_{b,x}}{d_k})(\frac{y_l - p_{b,y}}{d_l}) - (\frac{y_k - p_{b,y}}{d_k})(\frac{x_l - p_{b,x}}{d_l}) \right)^2 \tag{2.38}$$

which can be used as a term in the objective function to guarantee that the measurements taken along the generated trajectory yield a good estimate of $\mathbf{p}_0$. The main problem with using |FIM| objective function is its dependence on the true position of the AUV. In practice, the optimization and estimation algorithms are run in sequence at each time step. The optimization algorithm will use the most recent position estimate to generate the AUV's trajectory accordingly. This would require the optimization problem to be solved within the (usually short) allotted time. However, solving this optimization problem is a difficult task due to the several local extrema of the objective function |FIM|. This will make the solution of the optimization problem very sensitive to the initial guess provided to the solver. Here, we will use the solution of the unconstrained problem as the initial guess for the constrained problem.

In the first example, we consider one AUV moving from its initial position $\mathbf{p}_0 = [-10, 2]^T$ to the desired final position $\mathbf{p}_f = [10, -2]^T$. Figure 2.14 shows the results obtained with two different objective functions. The trajectory in 6(a) minimizes the overall energy consumption (2.34), with $t_f = 40s$. We then consider a weighted combination of the energy consumption and $-\log|\mathrm{FIM}|$. As Fig. 2.14(b) shows, the trajectory deviates from the straight line in order to increase the determinant of the FIM. This maneuver, however, demands a higher energy usage as shown in 2.14(d).

We next compare the two objective functions explained above in a problem with two AUVs. In Fig. 2.15(a) the trajectories minimize the overall energy consumption while keeping a safe distance of 3m between the AUVs. In Fig. 2.15(b) the determinant of the FIM associated with the AUVs' initial positions $\mathbf{p}_{0,i}, i = 1, 2$ is also included in the objective function. The determinant of the FIM for multiple vehicles and multiple beacons problem can be found in [Mor+16]. In this example, the AUVs measure their distances to the beacon fixed at the origin separately. The generated trajectories guide the AUVs in a circular motion around the beacon in order to increase the information content of the measurements. The results in Fig. 2.14 and Fig 2.15 were generated using a sampling time of 1s. The optimization problem was solved in 6.973 seconds.

2.4.4 Autonomous Drone Cinematography

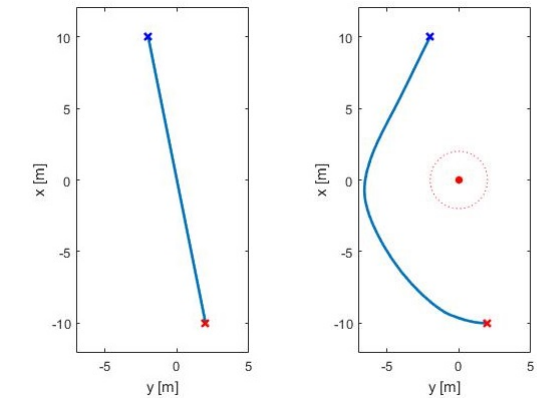
Before we formulate the OCP associated with trajectory generation for autonomous drone cinematography, we give a brief overview of the system onboard the drone cinematographer. The system architecture on board the drone is depicted in Fig. 1.1. The *Target Tracker* module provides the 3D target position $\mathbf{p}_T$ and velocity $\mathbf{v}_T$. We are assuming that targets carry onboard a GPS receiver to communicate their positions to the drone. However, visual-based or other alternative methods could also be used to estimate the target position and velocity. Besides, each drone receives the shot type to be executed from its *Shot Executer* module. Depending on the shot parameters and the target position/velocity, the Shot Executer module computes (and continuously updates) the desired 3D position $\mathbf{p}_D$ and velocity $\mathbf{v}_D$ for the drone. For instance, being asked to perform a lateral shot, the drone should track the target from a specific lateral distance which would be provided as a shot parameter. These objectives are sent to the *Cinematography Planner*, which computes the drone and gimbal movements concurrently. Here, we are not concerned with the definition of the available shot types and their parameters. Nonetheless, our approach to trajectory generation is generic regardless of the geometry of the shot, i.e., orbital, lateral, flyover, etc.

The Cinematography Planner consists of three modules: the *Trajectory Planner*, the *Trajectory Follower* and the *Gimbal Controller*. The Trajectory Planner generates optimal trajectories for the drone considering the constraints and objectives explained below. The cinematography planner aims at generating trajectories that guide the drone to the desired position and velocity given by the Shot Executer while conforming to the video aesthetic quality criteria. Moreover, it must ensure that generated trajectories are collision-free considering the pre-defined no-fly zones and collision avoidance constraints with other drones or obstacles detected during flight. Optimal trajectories are computed periodically in a receding horizon manner and sent to the Trajectory Follower module, which is able to compute 3D velocity commands for the drone to follow those trajectories. The software abstraction layer called *UAL* is the interface with the autopilot, that has velocity controllers for the drone (more details can be found in [Sab+19]). In parallel, the Gimbal Controller generates commands for the gimbal motors in the form of angular rates in order to keep the camera pointing towards the target.

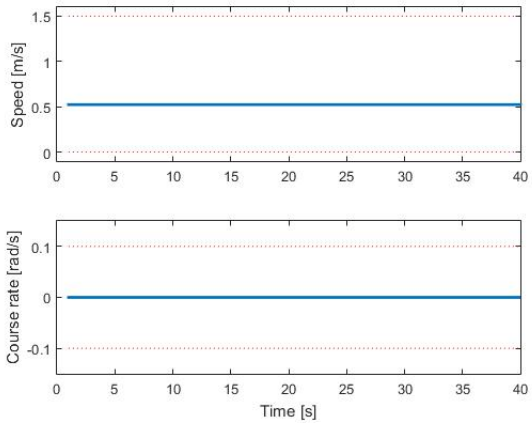
2.4.4.1 Quadrotor model

Let $\{W\}$ denote the world reference frame with origin fixed in the environment and East-North-Up (ENU) orientation. Consider also three additional reference frames: the quadrotor reference frame $\{Q\}$ attached to the vehicle with origin at the center of mass, the camera reference frame $\{C\}$ with z-axis aligned with the optical axis but with opposite sign, and the target reference frame $\{T\}$ attached to the moving target of interest. For simplicity, it is assumed that the origins of $\{Q\}$ and $\{C\}$ coincide. Figure 2.16 depicts the defined reference frames.

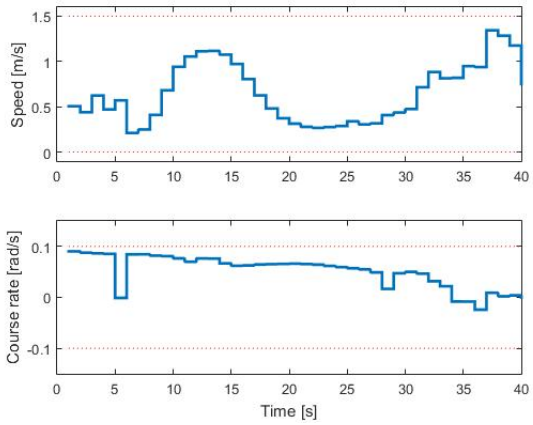
The configuration of $\{Q\}$ with respect to $\{W\}$ is denoted by $(\mathbf{p}_Q, R_Q) \in \mathbb{SE}(3)$, where $\mathbf{p}_Q \in \mathbb{R}^3$ is the position of the origin of $\{Q\}$ expressed in $\{W\}$ and $R_Q \in \mathbb{SO}(3)$ is the rotation



(a) (b)

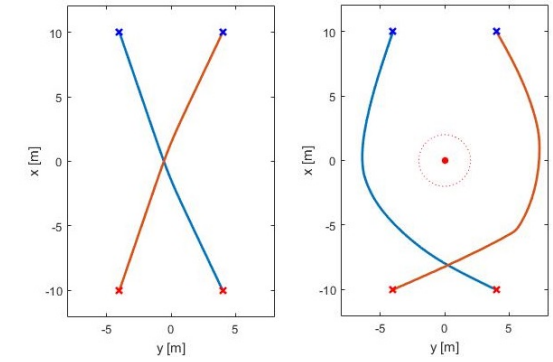


(c)

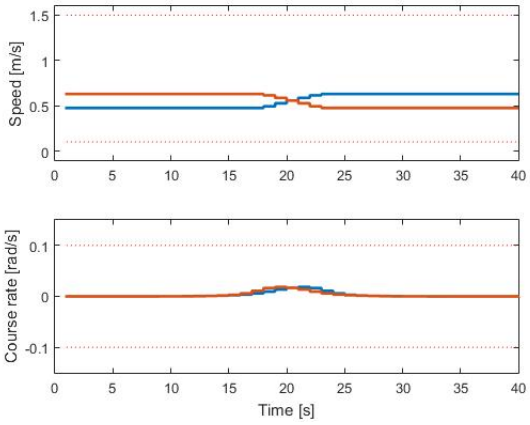


(d)

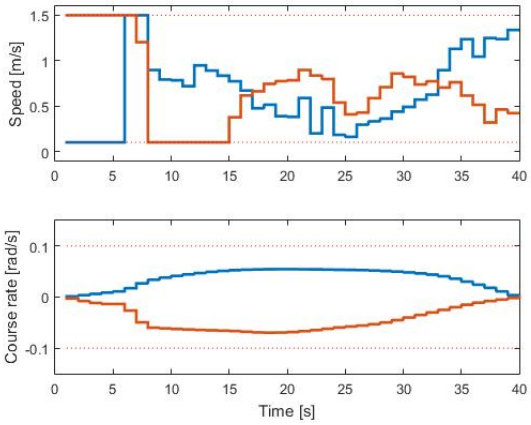
Figure 2.14: (a) and (b) show the AUV's trajectory obtained by minimizing the overall energy consumption and maximizing the $\log|FIM|$, respectively; (c) and (d) are the corresponding course rate and speed profiles for the trajectory in (a) and (b), respectively [SCP18].



(a) (b)



(c)



(d)

Figure 2.15: Two AUVs and a single beacon. The temporally separated trajectories generated in the absence and presence of $\log|FIM|$ in the objective fucntion are shown in (a) and (b), respectively. The input profiles for the trajectories in (a) and (b) are respectively shown in (c) and (d).

matrix from $\{Q\}$ to $\{W\}$. Similarly, the configurations of $\{T\}$ and $\{C\}$ with respect to $\{W\}$ are denoted by $(\mathbf{p}_T, R_T) \in \mathbb{SE}(3)$ and $(\mathbf{p}_C, R_C) \in \mathbb{SE}(3)$, respectively.

In this paper, the quadrotor linear dynamics are described by the following simple double integrator model

$$\begin{aligned} \dot{\mathbf{p}}_Q &= \mathbf{v}_Q \\ \dot{\mathbf{v}}_Q &= \mathbf{a}_Q, \end{aligned} \tag{2.39}$$

where $\mathbf{v}_Q = [v_x \ v_y \ v_z]^T \in \mathbb{R}^3$ is the linear velocity and $\mathbf{a}_Q = [a_x \ a_y \ a_z]^T \in \mathbb{R}^3$ is the linear acceleration. We assume that the linear acceleration $\mathbf{a}_Q$ takes the following form

$$\mathbf{a}_Q = -g\mathbf{e}_3 + R_Q \frac{T}{m}\mathbf{e}_3, \tag{2.40}$$

where m is the quadrotor mass, g is the gravitational acceleration, $T \in \mathbb{R}$ is the scalar thrust, and $\mathbf{e}_3 = [0 \ 0 \ 1]^T$.

For simplicity, we assume as control input the 3D acceleration $\mathbf{a}_Q$. Nonetheless, the thrust T and rotation matrix R_Q could also be recovered from 3D velocities and accelerations. If we restrict the yaw angle ψ_Q to keep the quadrotor's front pointing forward in the direction of motion such that

$$\psi_Q = \text{atan2}(v_y, v_x), \tag{2.41}$$

then the thrust T and the Z-Y-X Euler angles $\boldsymbol{\lambda}_Q = [\phi_Q, \theta_Q, \psi_Q]^T$ can be obtained from $\mathbf{v}_Q$ and $\mathbf{a}_Q$ according to

$$\begin{cases} T = m\|\mathbf{a}_Q + g\mathbf{e}_3\| \\ \psi_Q = \text{atan2}(v_y, v_x) \\ \phi_Q = -\arcsin((a_y \cos(\psi_Q) - a_x \sin(\psi_Q))/\|\mathbf{a}_Q + g\mathbf{e}_3\|) \\ \theta_Q = \text{atan2}(a_x \cos(\psi_Q) + a_y \sin(\psi_Q), a_z + g) \end{cases} \tag{2.42}$$

2.4.4.2 Gimbal angles

Let $\boldsymbol{\lambda}_C = [\phi_C, \theta_C, \psi_C]^T$ denote the Z-Y-X Euler angles that parametrize the rotation matrix R_C, such that

$$R_C = R_z(\psi_C) R_y(\theta_C) R_x(\phi_C). \tag{2.43}$$

The Gimbal Controller provides reference angles such that the camera points towards the target. For simplicity, we consider that the time-scale separation between the "faster" gimbal dynamics and "slower" quadrotor dynamics is sufficiently large to neglect the gimbal dynamics and assume an exact match between the desired and actual orientations of the gimbal. To define R_C, we introduce the relative position

$$\mathbf{q} = \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^T = \mathbf{p}_C - \mathbf{p}_T \tag{2.44}$$

and assume that the quadrotor/camera is always above the target, i.e. $q_z > 0$, and not directly above the target, i.e. $[q_x \ q_y]^T \neq \mathbf{0}$. Then, the gimbal orientation R_C that guarantees the camera is aligned with the horizontal plane and pointing towards the target is given by

$$\begin{aligned} R_C &= \begin{bmatrix} -\dfrac{\mathbf{q} \times \mathbf{q} \times \mathbf{e}_3}{\|\mathbf{q} \times \mathbf{q} \times \mathbf{e}_3\|} & \dfrac{\mathbf{q} \times \mathbf{e}_3}{\|\mathbf{q} \times \mathbf{e}_3\|} & \dfrac{\mathbf{q}}{\|\mathbf{q}\|} \end{bmatrix} \\ &= \begin{bmatrix} * & \frac{q_y}{\sqrt{q_x^2+q_y^2}} & * \\ * & \frac{-q_x}{\sqrt{q_x^2+q_y^2}} & * \\ \frac{\sqrt{q_x^2+q_y^2}}{\sqrt{q_x^2+q_y^2+q_z^2}} & 0 & \frac{q_z}{\sqrt{q_x^2+q_y^2+q_z^2}} \end{bmatrix}. \end{aligned} \tag{2.45}$$

Since the camera is assumed to be aligned with the horizontal plane, the roll angle $\phi_C = 0$. In this case, R_C takes the form

$$R_C = \begin{bmatrix} \cos(\psi_C)\cos(\theta_C) & -\sin(\psi_C) & \cos(\psi_C)\sin(\theta_C) \\ \cos(\theta_C)\sin(\psi_C) & \cos(\psi_C) & \sin(\psi_C)\sin(\theta_C) \\ -\sin(\theta_C) & 0 & \cos(\theta_C) \end{bmatrix}, \tag{2.46}$$

and we obtain

$$\begin{cases} \phi_C = 0 \\ \theta_C = \text{atan2}(-\sqrt{q_x^2 + q_y^2}, q_z) \\ \psi_C = \text{atan2}(-q_y, -q_x) \end{cases} \tag{2.47}$$

For non-aggressive maneuvers, we can assume that accelerations a_x and a_y are small, and hence, we can assume by direct application of (2.42), that the quadrotor roll and pitch angles are also small and $R_x(\phi_Q) \approx R_y(\theta_Q) \approx I_3$. In the following, it will become clear that this is a reasonable assumption, since one of the terms included in the cost function to be minimized is exactly the squared norm of the quadrotor acceleration.

Under this assumption, the orientation matrix of the gimbal with respect to the quadrotor ${}^Q_C R$ can be approximated by

$$\begin{aligned} {}^Q_C R &= (R_Q)^T R_C \\ &\approx R_z(\psi_C - \psi_Q) R_y(\theta_C) R_x(\phi_C), \end{aligned} \tag{2.48}$$

and the relative Euler angles (roll, pitch and yaw) of the gimbal with respect to the quadrotor are obtained as

$$\begin{cases} {}^Q\phi_C = \phi_C = 0 \\ {}^Q\theta_C = \theta_C = \text{atan2}(-\sqrt{q_x^2 + q_y^2}, q_z) \\ {}^Q\psi_C = \psi_C - \psi_Q = \text{atan2}(-q_y, -q_x) - \text{atan2}(v_y, v_x) \end{cases} \tag{2.49}$$

According to (2.42), (2.47) and (2.49), λ_Q, λ_C and ${}^Q\lambda_C$ are completely defined by the trajectories of the quadrotor and the target, and thus can be expresses as explicit functions of $\mathbf{q}$, $\mathbf{v}_Q$, and $\mathbf{a}_Q$.

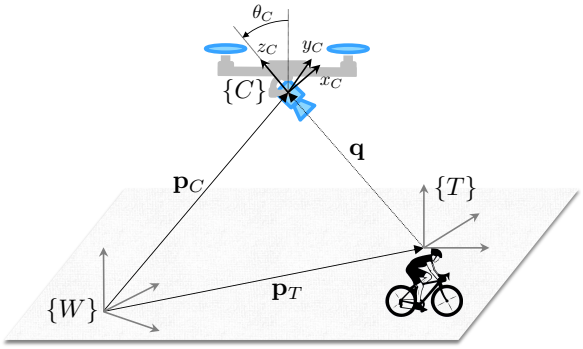


Figure 2.16: Illustration of reference frames. The origin of the camera and the vehicle reference frames coincide.

2.4.4.3 Problem formulation

Here, we address the trajectory generation problem for autonomous cinematography by decoupling gimbal and drone control, that is, we run a gimbal controller in charge of pointing the camera to the target; and in parallel, we solve an optimization problem to generate trajectories for the drone taking into account the gimbal rotation limits. This approach yields a much simpler optimization problem in comparison to using an integrated model of the camera and the vehicle dynamics [Näg+17], while ensuring safe and smooth trajectories for both the drone and the camera on the gimbal.

We formulate the problem such that the generated trajectory guides the drone to a given desired position while filming a moving target. The desired 3D position ($\mathbf{p}_D$) and velocity ($\mathbf{v}_D$) depend on the shot type and are provided by the Shot Executer. We incorporate constraints imposed by the environment and vehicle dynamics as well as those imposed by rotation limits of the gimbal using the equations derived in (2.49). In order to meet requirements on aesthetic quality of the output video, we employ a properly defined objective function that includes camera angle driven terms to limit its rotation speed in favor of smooth shots.

The problem described above is formulated as

$$\underset{\mathbf{x}(t),\mathbf{u}(t)}{\text{minimize}} \int_0^{t_f} (w_1||\mathbf{u}(t)||^2 + w_2\dot{\theta}_C(t)^2 + w_3\dot{\psi}_C(t)^2)dt + w_4 J_N \tag{2.50}$$

$$\begin{aligned}
\text{subject to } & \mathbf{x}(0) = \mathbf{x}_0 && \text{(2.50.a)}\\
& \dot{\mathbf{x}}(t) = f(\mathbf{x}(\mathbf{t}), \mathbf{u}(t)) && \text{(2.50.b)}\\
& \mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max} && \text{(2.50.c)}\\
& \mathbf{v}_{min} \leq \mathbf{v}_Q(t) \leq \mathbf{v}_{max} && \text{(2.50.d)}\\
& \mathbf{p}_Q(t) \in \mathcal{F} && \text{(2.50.e)}\\
& \theta_{min} \leq^Q \theta_C(t) \leq \theta_{max} && \text{(2.50.g)}\\
& \psi_{min} \leq^Q \psi_C(t) \leq \psi_{max} && \text{(2.50.h)}
\end{aligned}$$

where $\mathbf{x} = [\mathbf{p}_Q\ \mathbf{v}_Q]^T$ and $\mathbf{u} = \mathbf{a}_Q$. The first term in the objective function penalizes excessive use of the control inputs while the second and third terms respectively penalize the changes in the gimbal pitch and yaw angles. Having the target position predicted for the future time horizon, these terms can be expressed in terms of $\mathbf{x}$ and $\mathbf{u}$ using (2.49). Finally, the terminal cost is added to drive the drone to the desired position and velocity, and is defined as

$$J_N = ||\mathbf{x}_N - [\mathbf{p}_D\ \mathbf{v}_D]^T||^2 \tag{2.51}$$

Appropriate tuning of the weights $w_1, \ldots, w_4$ will make sure that the generated trajectory creates smooth camera movement during the shot.

In (2.50.a), $\mathbf{x}_0$ denotes the initial state of the vehicle. The system kinematics (2.50.b) along with the bounds on the velocity (2.50.d) and acceleration (2.50.c) ensure the feasibility of the resulting trajectory. The constraint (2.50.e) keeps the drone away from the no-fly zone and obstacles in the surrounding environment. The free space $\mathcal{F}$ is constructed considering the obstacles' positions and the regulatory restricted zones established before the flight to avoid dangerous areas with people, buildings, etc.. Lastly, the constraints on ${}^Q\theta_C$ and ${}^Q\psi_C$, guarantee that the gimbal angles are within the required bounds. The lower/upper bounds are specified by gimbal mechanical limits to rotate around each axis.

2.4.4.4 Simulation results

In the first example, we study the effect of adding the camera angle driven terms to the objective function. We consider a single drone filming a non-moving target. The goal is to find a smooth

T	140s
$\mathbf{a}_{\min}, \mathbf{a}_{\max}$	$\pm[1\ 1\ 1]^T$ m/s
$\mathbf{v}_{\min}, \mathbf{v}_{\max}$	$\pm[1\ 1\ 1]^T$ m/s^2
$\theta_{\min}, \theta_{\max}$	$-\pi/2, -3\pi/8$ rad
$\psi_{\min}, \psi_{\max},$	$-3\pi/4, 3\pi/4$ rad

Table 2.2: Upper/lower bounds used in the simulations.

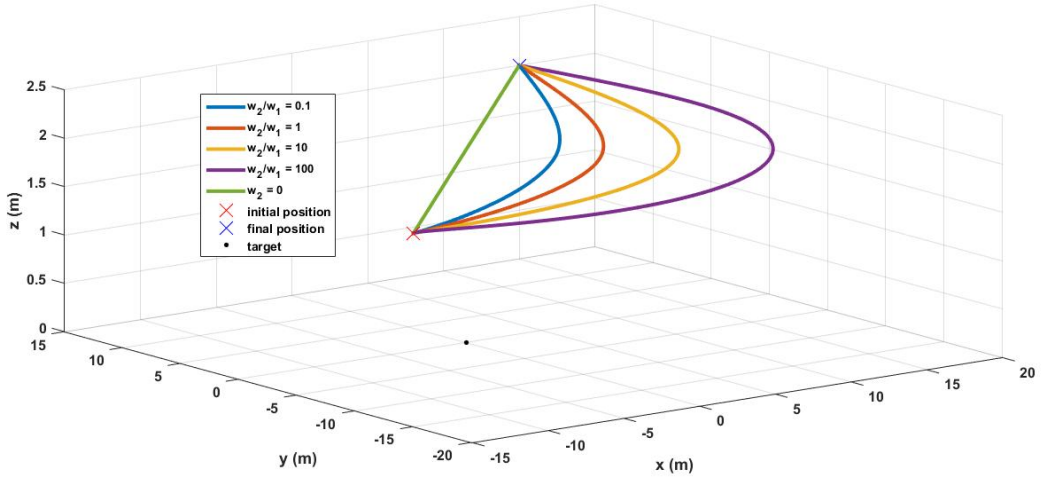


Figure 2.17: Generated trajectories for the single drone-single target example with different relative weights in the objective function (2.50). The trajectories guide the vehicle from its initial position (x) to the final position (x) while the gimbal is pointing towards the target (fixed at the origin).

trajectory that guide the drone from its initial position to a given final position (provided by the shot executor) in a limited amount of time. The OCP (2.50) is solved using the parameter values provided in Table 2.2.

Figure 2.17 shows the trajectories obtained with different relative weights in the above objective function. For $w_2, w_3 = 0$ the solver minimizes the overall energy consumption which results in a straight line from the initial position, shown with x, to the final position (x). The resulting path is obtained without considering any constraint on the gimbal angles. With the increase of $\frac{w_2}{w_1}$, the generated trajectory deviates from the straight line and converge to a circular path to reduce the pitch angle changes. Fig. 2.18 show the corresponding top view (left) and pitch angle (right) of the generated trajectories. As it can be seen the constraint on the pitch angle is satisfied for all trajectories, however, the instantaneous rate of change is decreasing with the increase of w_2. Fig. 2.19 shows the linear velocity and acceleration. It can be observed from Fig. 2.19 that, as w_2 increases, the gimbal pitch angle rate is reduced at the cost of increased acceleration.

2.4.5 Cooperative planning for multiple drones

In the next example, we consider two drones filming a single moving target. In order to obtain collision-free trajectories, the inter-vehicle collision avoidance constraint (2.33) is included in the problem. We also incorporate mutual visibility constraints to make sure that the recorded shot by a camera is unobstructed and does not contain the other flying camera. This is addressed by enforcing the angle between the relative position vector of the cameras, $\mathbf{d} = \mathbf{p}_i - \mathbf{p}_j$, and the view direction of the camera, $\mathbf{q} = \mathbf{p}_i - \mathbf{p}_T$, to be greater than some desired angle determined

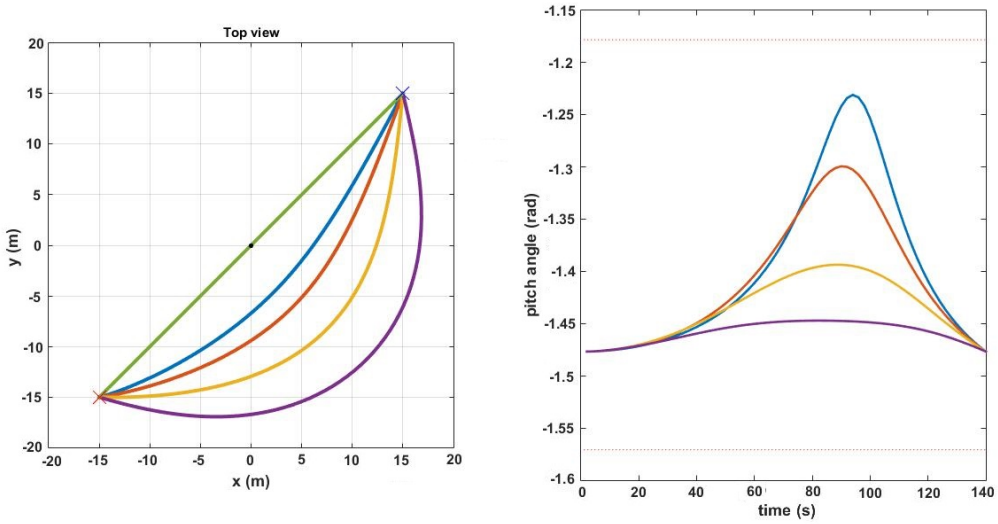


Figure 2.18: The top view (**left**) and the gimbal pitch angle (**right**) of the generated trajectories in Fig. 2.17.

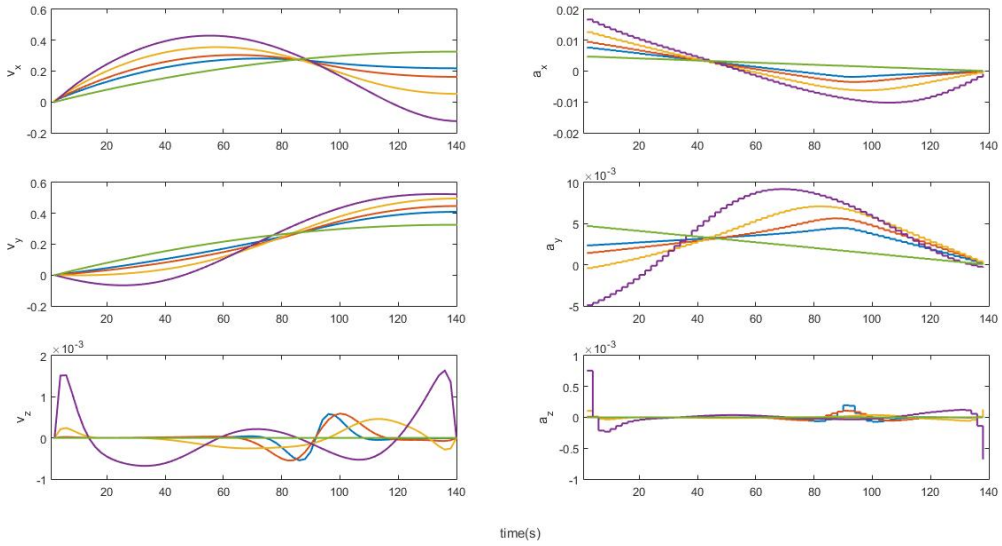


Figure 2.19: The linear velocity and acceleration of the generated trajectories in Fig. 2.17. As w_2 increases the resulting trajectory reduces the gimbal pitch angle rate at the cost of increased energy consumption.

from the camera's field of view. Therefore, the mutual visibility constraint can be expressed as

$$\cos(\beta) \leq \cos(\alpha) \tag{2.52}$$

where

$$\cos(\beta) = \frac{\mathbf{d}.\mathbf{q}}{\|\mathbf{d}\| \, \|\mathbf{q}\|} \tag{2.53}$$

Imposing (2.52) can ensure that the j-th camera is out of the i-th camera's field of view approximated with a cone. Fig. 2.20 is a schematic illustration of the mutual visibility constraint.

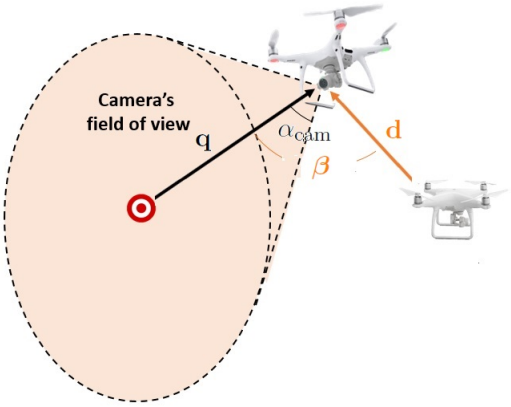
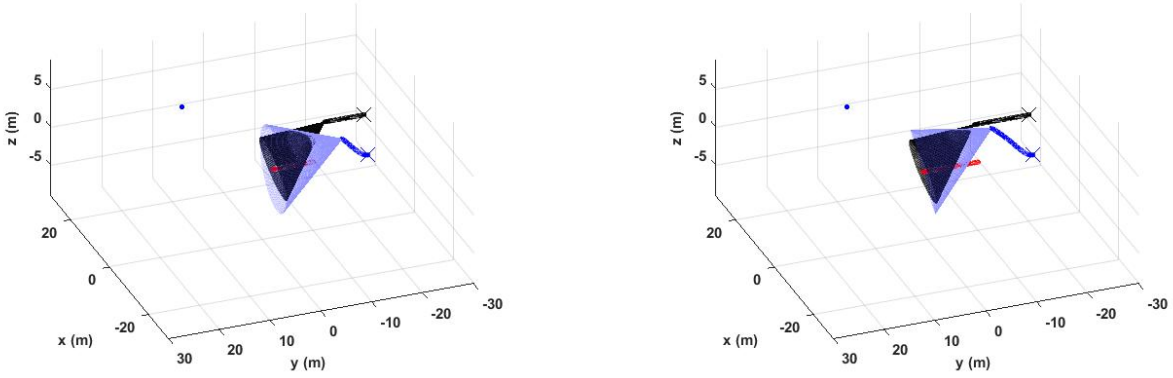
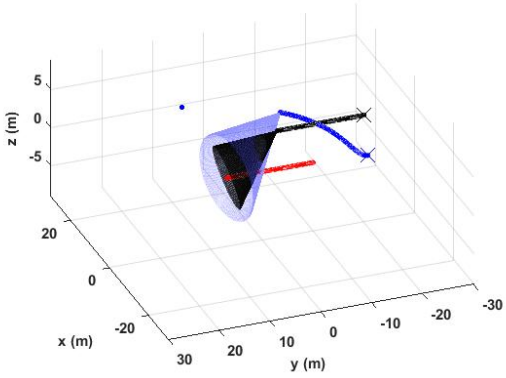


Figure 2.20: Schematic of mutual visibility constraint. Other flying drones must be stayed out of the camera's field of view.

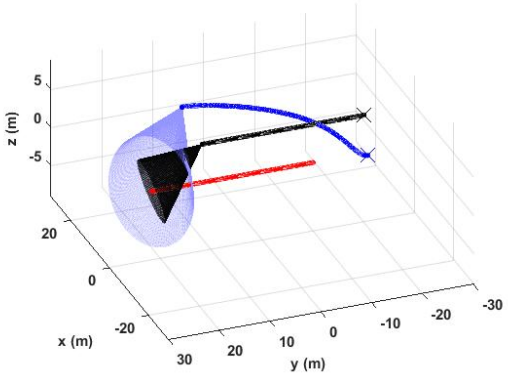


(a) t = 40 s

(b) t = 55 s



(c) t = 86 s



(d) t = 160 s

Figure 2.21: The generated trajectory for two drones (black and blue) at different time instances. The trajectories are generated such that both cameras have unobstructed views of the target moving on the red line.

Fig. 2.21 shows an example where two drones are filming a target moving on a straight line (red line). The first drone, whose trajectory is shown with solid black line, should follow the target while keeping a relative position to the target. This is enforced by adding a term in the objective function that penalizes the deviation of the drone's trajectory from the desired straight line. The second drone should fly from its initial position to the given final position shown with

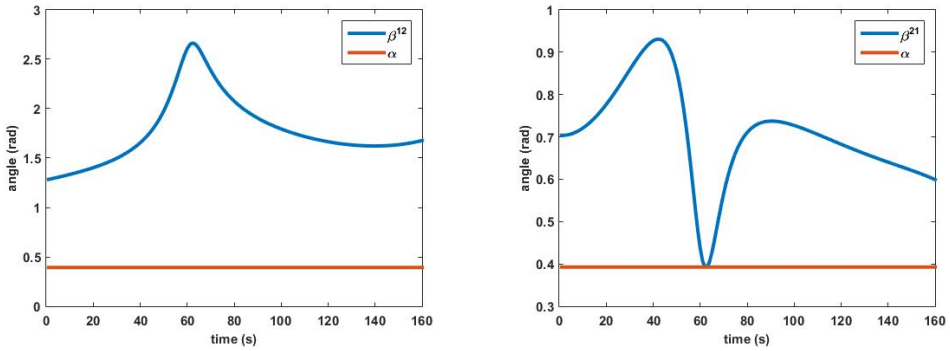


Figure 2.22: The computed β angles for the two cameras show that the mutual visibility constraints are satisfied over the entire time interval.

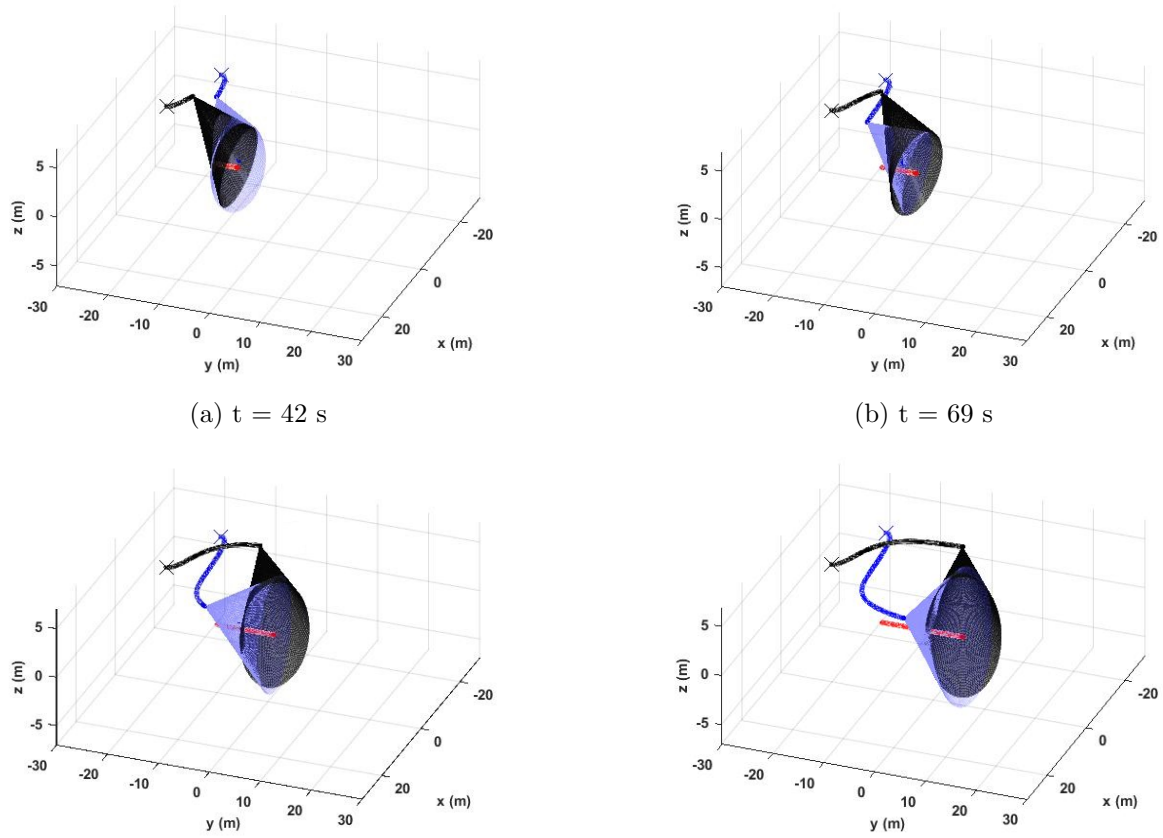
•. The resulting trajectories are shown in Fig. 2.21 with timestamps. The black and blue shaded cones show the field of view of the first and second cameras, respectively. Fig. 2.22 shows β for the two cameras. It can be observed that the mutual visibility constraints are satisfied for the entire flight time.

Fig. 2.23 is another example where two drones should fly from their initial positions to given final positions while filming the moving target. In this example, the drones' final positions are given in terms of relative positions with respect to target's position. Thus, the trajectories must be generated such that in a given period of time, 160s, the drones are located in a certain distance from the target. Fig. 2.23 shows the resulting trajectories and the cameras' fields of view at different time stamps. Fig. 2.24 displays the generated trajectories for the two drones from the top view. Also, the computed β angles are displayed in Fig 2.25 to confirm that both cameras have unobstructed views of the target.

2.4.6 Trajectory re-planning in the receding horizon manner

So far, in the presented simulation results, the trajectories over the entire time interval $[0, t_f]$ were computed in one step by solving a single optimization problem. However, because of disturbances and uncertainties in real-world experiments, it might be necessary to re-plan the trajectories during the flight using the real-time measurements and estimates of the problem parameters. Thus, the problem must be solved in a receding horizon manner. This will convert the trajectory generation problem into solving a reduced-scale optimization problem repeatedly over a moving time horizon. At each time step, a trajectory is computed for a (short) time horizon and the first part of the plan $\mathbf{u}_0$ (or $\mathbf{x}_1$) is provided to the controller. This will be repeated at the next time step with the horizon shifted one step forward. This approach allows using updated information, such as the target and obstacles' positions, while generating the new trajectory.

Here, we consider an example in which two drones are filming a moving target. The first drone is fixed at $\mathbf{p} = [0 - 162]^T$ which is initially estimated by $\hat{\mathbf{p}} = [-2, -17.5, 2]^T$. and the other is flying from its initial position (x) to the final position (x). Both cameras are pointing towards the target whose future path is predicted with a linear model. We assume that the position of the first drone and the target are available with some measurement noise. The goal is to find a trajectory that guide the drone to the given final position in a given time ($t_f = 150s$) while ensuring that both cameras have unobstructed views of the target during the entire flight time. In order to compensate for the errors, the trajectory of the drone is re-planned at each time step using the new measurements of the position of the target and the other drone. Fig. 2.26(a) shows the re-planned trajectories in 40 time steps with a sample interval of. In order to implement the receding horizon scheme, one should make sure that the optimization problem is solved in the limited time between the two steps. The average computation time in this example is



(a) t = 42 s

(b) t = 69 s

(c) t = 114 s

(d) t = 160 s

Figure 2.23: The generated trajectory for two drones (black and blue) at different time instances. The trajectory for the first drone is generated such that both cameras have unobstructed views of the target moving on the red line.

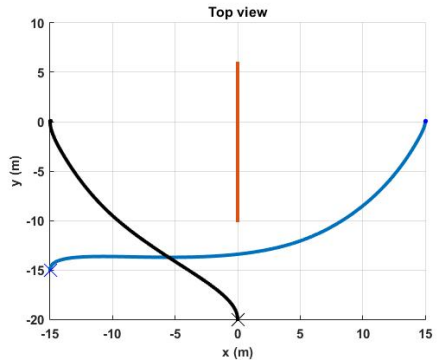


Figure 2.24: The top view of the trajectories shown in Fig. 2.23.

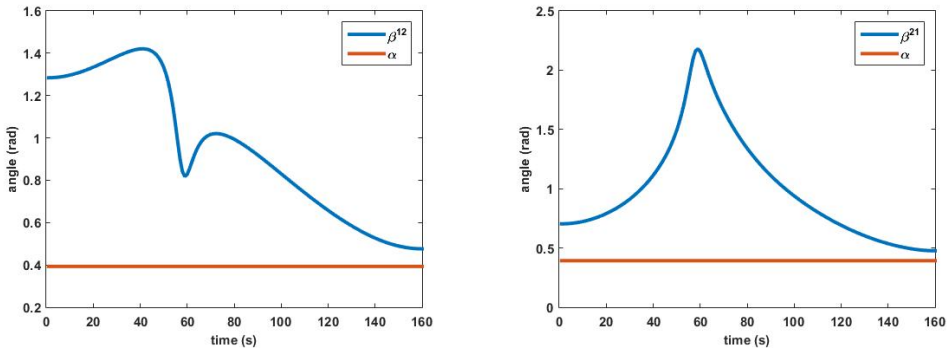


Figure 2.25: The computed β angles show that both cameras have unobstructed views of the moving target.

96 milliseconds. Fig. 2.26(b) compare the initial trajectory to the final trajectory obtained by joining the re-planned segments. It must be noted that incorporating continuity conditions at is necessary for generating smooth trajectories in a receding horizon approach.

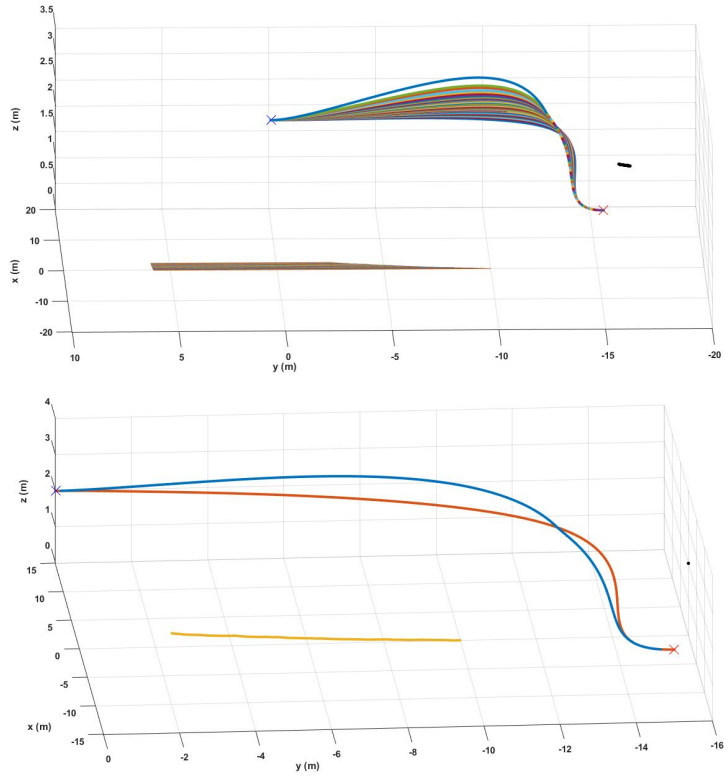


Figure 2.26: (a) Replanning the trajectory in the presence of uncertainty in the target position estimates. The drone trajectory is re-planned at each time step taking into account the most recent measurements of the target and the other drone positions. (b) The initial and final trajectories of the drone. The final trajectory is obtained by applying the first part of the re-planned trajectory at each time step.

Chapter 3

Bézier Curve-based Trajectory Generation Method for Differentially Flat Systems

3.1 Introduction

This chapter is dedicated to motion planning for a particular class of dynamic systems, labeled as *differentially flat* systems, whose special structure can be exploited to develop a real-time efficient trajectory generation framework.

A general statement of the trajectory generation problem that we consider in this thesis is to find the (control) trajectory for a dynamic system of the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ over the time interval $[0, t_f]$, such that it minimizes a cost function, given by

$$J = \int_0^{t_f} L(\mathbf{x}(t), \mathbf{u}(t))dt + E(\mathbf{x}(t_f)), \tag{3.1}$$

and satisfies a set of equality and inequality constraints expressed as

$$\begin{aligned} &\mathbf{h}(\mathbf{x}(t), \mathbf{u}(t)) \leq 0 && t \in [0, t_f] \\ &\mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) = 0 && t \in \{0, t_f\}. \end{aligned} \tag{3.2}$$

The above inequality contains path and actuator constraints and the equality includes initial and final conditions on the state and the input of the system, given by

$$\mathbf{x}(0) = \mathbf{x}_0 \quad \mathbf{x}(t_f) = \mathbf{x}_f \tag{3.3}$$

$$\mathbf{u}(0) = \mathbf{u}_0 \quad \mathbf{u}(t_f) = \mathbf{u}_f. \tag{3.4}$$

Therefore, the trajectory generation problem includes a combination of differential, equality and inequality constraints. As we will see later in this chapter, the differential flatness property of a dynamic system allows canonical transformation to a higher-order form that trivially satisfies the state equations. Therefore, the state equations of the dynamic system, i.e. the differential equations, can be eliminated by transforming the problem into the space of some differentially independent variables called flat outputs. This is clearly advantageous over its alternatives, e.g. using collocation or shooting techniques, to find a trajectory consistent with the system dynamics.

Polynomial-based motion planning methods for differentially flat systems exploit the differential flatness property of a system and parameterize each flat output with a polynomial function. This converts the problem of finding functions in an infinite dimensional space into an approximate one of finding a finite set of coefficients describing the polynomials. The resulting problem, as will be explained in the following, is a semi-infinite optimization problem involving a finite number of variables and an infinite number of constraints. In this chapter, we use Bézier curve for path parameterization and leverage its properties and algorithms to convert the semi-infinite optimization problem into one that is computationally tractable.

3.1.1 Differentially flat systems

Flat systems were first introduced in [Fli+92]. Since then, applications of flatness in engineering problems have grown steadily. In differential algebra, a system is viewed as a differential field generated by the set of states and inputs. A system is said to be differentially flat if there exists a set of variables such that the system is (non-differentially) algebraic over the differential field generated by the set of variables. It is important to mention that many of the systems for which strong nonlinear control techniques are available are in fact flat systems [MMR02]. All controllable linear systems can be shown to be flat. Indeed, any system that can be transformed into a linear system by changes of coordinates, static feedback transformations, or dynamic feedback transformations is also flat [QR10], [MMR03].

Consider the nonlinear system of the form

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}), \tag{3.5}$$

with $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$, $f(0,0) = 0$, and $\operatorname{rank}\frac{\partial f}{\partial \mathbf{u}}(0,0) = m$. The nonlinear system (3.5) is dynamic feedback linearizable if there exist

- A *regular*[1] dynamic compensator:

$$\begin{aligned} \dot{\mathbf{z}} &= a(\mathbf{x}, \mathbf{z}, \boldsymbol{\upsilon}) \\ \mathbf{u} &= b(\mathbf{x}, \mathbf{z}, \boldsymbol{\upsilon}) \end{aligned} \tag{3.6}$$

  where $\mathbf{z} \in \mathbb{R}^q$, $\boldsymbol{\upsilon} \in \mathbb{R}^m$, $a(0,0,0) = 0$, and $b(0,0,0) = 0$;

- A diffeomorphism [2]:

$$\boldsymbol{\xi} = \Xi(\mathbf{x}, \mathbf{z}), \quad (\boldsymbol{\xi} \in \mathbb{R}^{n+q}) \tag{3.7}$$

such that (3.5) and (3.6) become a constant linear controllable system [Fli+95], that is, using the extended coordinate transformation (3.7), the $(n+q)$-dimensional dynamics given by

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, b(\mathbf{x}, \mathbf{z}, \boldsymbol{\upsilon})) \\ \dot{\mathbf{z}} &= a(\mathbf{x}, \mathbf{z}, \boldsymbol{\upsilon}) \end{aligned} \tag{3.8}$$

can be expressed as a linear system of the form

$$\dot{\boldsymbol{\xi}} = F\boldsymbol{\xi} + G\boldsymbol{\upsilon} \tag{3.9}$$

Using a static state feedback and a linear invertible change of coordinates, the above system can be written in Brunovsky canonical form, as expressed by

$$y_j^{(\nu_j)} = \upsilon_j \quad j = 1, \ldots, m \tag{3.10}$$

where $\nu_j, (j = 1, \ldots, m)$ are the controllability indices with $\nu_1 + \cdots + \nu_m = n + q$, and thus, $Y = (y_1, \ldots, y_1^{(\nu_1 - 1)}, \ldots, y_m, \ldots, y_m^{(\nu_m - 1)})$ is another basis for the vector space spanned by the components of $\boldsymbol{\xi}$, and can be stated as

$$Y = T\boldsymbol{\xi} = T\Xi(\mathbf{x}, \mathbf{z}) \tag{3.11}$$

where $T \in \mathbb{R}^{(n+q)\times(n+q)}$ is an invertible matrix. The invertibility of Ξ results in

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} = \Xi^{-1}(T^{-1}Y) \tag{3.12}$$

and thus

$$\mathbf{u} = b(\Xi^{-1}(T^{-1}Y), \boldsymbol{\upsilon}) \tag{3.13}$$

Since $\upsilon_j = y_j^{(\nu_j)}, j = 1, \ldots, m$, $\mathbf{u}$ and $\mathbf{x}$ can be written as analytic functions of the components of $\mathbf{y} = (y_1, \ldots, y_m)$ and a finite number of their derivatives.

[1] The regularity assumption implies the invertibility of (3.6) with input υ and output u.

[2] A diffeomorphism is an invertible function that maps on differentiable manifold such that the function and its inverse are both smooth.

$$\begin{aligned}\mathbf{x} &= \mathcal{A}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(\alpha)}) \\ \mathbf{u} &= \mathcal{B}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(\beta)})\end{aligned} \tag{3.14}$$

The dynamic feedback (3.6) is said to be *endogenous* if and only if the components of $\mathbf{y}$ can be expressed as analytic functions of $\mathbf{x}$, $\mathbf{u}$ and a finite number of its derivatives, i.e.,

$$\mathbf{y} = \mathcal{C}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \ldots, \mathbf{u}^{(\gamma)}) \tag{3.15}$$

The dynamic system (3.5) which is linearizable via such an endogenous feedback is said to be *differentially flat*, and $\mathbf{y}$ is called a *flat output* [Fli+95]. It is important to point out that flatness is a geometric property of a system, and hence it is independent of coordinate choice.

3.1.2 Polynomial parameterization of the flat output

One major property of differentially flat systems, as stated above, is that the state and the input variables can be directly expressed, without integrating any differential equation, in terms of the flat output and a finite number of its derivatives. For any system admitting the alternate form of (3.14), trajectories consistent with the dynamics (3.5) can be planned using the flat output $\mathbf{y}(t)$. The state and input trajectories, $\mathbf{x}(t)$ and $\mathbf{u}(t)$, can be immediately deduced from $\mathbf{y}(t)$ trajectories.

The trajectory generation problem in the space of flat output, where the dynamic constraint is trivially satisfied, can be stated as finding $\mathbf{y}(t)$ such that it minimizes a cost function, $J(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(p)})$, derived from the cost function of the original trajectory generation problem, and satisfies a set of equality and inequality constraints expressed as,

$$\begin{aligned}\bar{\mathbf{h}}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(q_1)}) &\leq 0 \qquad && t \in [0, t_f] \\ \bar{\mathbf{g}}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^{(q_2)}) &= 0 && t \in \{0, t_f\}\end{aligned} \tag{3.16}$$

The above set of constraints is obtained by simply inserting (3.14) in the original constraints (3.2) (See Fig. 3.1).

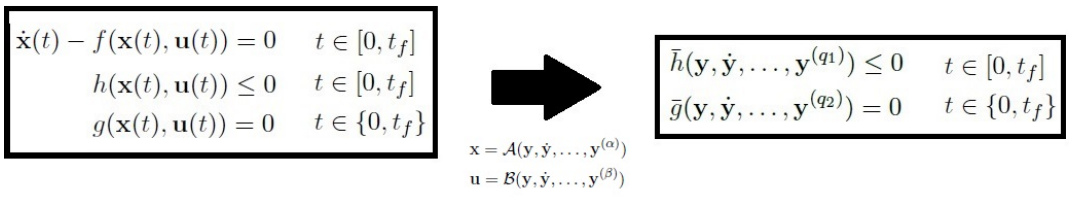


Figure 3.1: Converting the original constraints (left) into equivalent ones in the space of flat output (right).

As indicated above, the components of the flat output $\mathbf{y}(t)$ are differentially independent, i.e., there is no differential relation of the form $R(\mathbf{y}, \dot{\mathbf{y}}, ..., \mathbf{y}^{(r)}) = 0$ [Rig15], and therefore, each can be described by an individual function. In polynomial-based motion planning for flat systems, each $y_j(t)$ is parameterized with a polynomial function as

$$y_j(t) = \sum_{k=0}^{n_j} a_{jk} \Phi_{jk}(t), \quad j = 1, \ldots, m, \tag{3.17}$$

where $a_{jk} \in \mathbb{R}$ are the coefficients and $\Phi_{jk}(t)$ are the polynomial basis functions. Polynomial parameterization of the flat output converts the problem of finding functions in an infinite dimensional space into an approximate one of finding a finite set of coefficients. (Remark 3.1 provides an example where the optimal trajectory is a polynomial). Consequently, the trajectory generation problem is transformed into a semi-infinite optimization problem involving a finite number of variables $a_{jk}, k = 1, \ldots, n_j; j = 1, \ldots, m$ and an infinite number of constraints (3.16).

In order to obtain a computationally tractable optimization problem, different methods have been proposed for dealing with the inequality constraint in (3.16). Earlier work on feasible or optimal motion planning for differentially flat systems does not consider path and actuator constraints [VM98], [AV96], [FA98]. [FAM01] is the first paper investigating trajectory generation for such systems with inequality constraints. It proposes three different techniques for constructing a finite set of constraints that guarantee satisfaction of linear inequalities over the entire time interval $[0, T]$. The proposed techniques are also applied to nonlinear inequality constraints using a convex polytopic approximation (see Fig. 3.3).

The feasible region defined by (3.16) is in general non-convex in the space of $\mathbf{y}$ and its derivatives. However, it can be shown that for linear systems, if the constraint set in the original space is convex, under a linear transformation to the flat space, the set defined by (3.16) is also convex. Moreover, on invoking a transformation to the flat space, linear inequalities will remain linear. Since a convex set can be represented by a system of linear inequalities, the feasible set defined by (3.16) can be expressed as a set of linear inequalities, i.e., a polytope, in the flat space for linear systems with linear inequalities or convex nonlinear inequalities in the original space.

This is not the case for linear systems with general nonlinear inequalities or nonlinear systems. One way to tackle this issue is to approximate the feasible set by a polytope entirely contained within the set, expressed as

$$\bar{h}(t) \approx M_0 \mathbf{y}(t) + M_1 \dot{\mathbf{y}}(t) + \cdots + M_p \mathbf{y}^{(p)}(t) + \mathbf{e} \leq 0 \tag{3.18}$$

where M_i are $l \times m$ matrices for a polytope with l faces, and $\mathbf{e}$ is a constant $l \times 1$ vector [FAM01]. Given a non-convex set S, the main challenge would be to find the best convex approximation by determining $M_i, i = 0, \ldots, p$ such that the resulting polytope encloses the maximum volume within S. This problem is cast as an optimization problem which has to be solved once (off-line) if the constraints do not change in the course of a mission. Using the obtained M_i and substituting (3.17) into (3.18), the constraints can be re-written as linear inequalities on a_{jk}, according to

$$\bar{h}(t) \approx \sum_{i=0}^{p} \sum_{j=1}^{m} \sum_{k=0}^{n_j} M_{i_j} a_{jk} \phi_{jk}^{(i)}(t) + \bar{\mathbf{e}} \leq 0 \tag{3.19}$$

where M_{i_j} is the j-th row of matrix M_i. This inequality must hold for all the time instances in the interval $[0, T]$, therefore, (3.19) represents an infinite number of constraints on the coefficients a_{jk}. The three different schemes, proposed in [VM98], to satisfy (3.19) on the entire interval using only a finite number of constraints are listed below.

- Collocation scheme

  In this scheme a finite number of collocation points is selected within the interval $[0, T]$, and the constraint functions and a finite number of their derivatives are bounded at each of the selected points to ensure that (3.19) is satisfied. Consider a single inequality constraint ($l = 1$) of the form $\bar{h}(t) \leq 0$. Given $\bar{h}(t)$, $\bar{h}(t + \Delta t)$ can be approximated by

$$\bar{h}(t + \Delta t) = \bar{h}(t) + \bar{h}^{(1)}(t)\Delta t + \cdots + \bar{h}^{(r)}(t)\frac{\Delta t^r}{r!}, \tag{3.20}$$

Remark 3.1: Solving the minimum snap trajectory problem

The function $x(t) \in C^1[0, T]$ is an extremum of the functional J defined by an integral of the form

$$J = \int_0^T \mathcal{L}(x^{(n)}, \ldots, \ddot{x}, \dot{x}, x)dt$$

if it satisfies the Euler-Lagrange differential equation:

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{x}}\right) + \frac{d^2}{dt^2}\left(\frac{\partial \mathcal{L}}{\partial \ddot{x}}\right) + \cdots + (-1)^n \frac{d^n}{dt^n}\left(\frac{\partial \mathcal{L}}{\partial x^{(n)}}\right) = 0$$

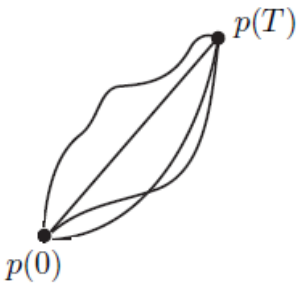


Figure 3.2: Possible paths joining the initial and final points.

Applying the above necessary conditions to the minimum snap trajectory problem, expressed as

$$p^*(t) = \arg\min_{p(t)} \int_0^T (p^{(4)})^2 dt$$

yields

$$p^{(8)} = 0$$

The optimal trajectory will be obtained as a polynomial of degree 7, given by

$$p(t) = c_7 t^7 + \cdots + c_1 t + c_0$$

The coefficients c_7 to c_0 can be determined from the boundary conditions.

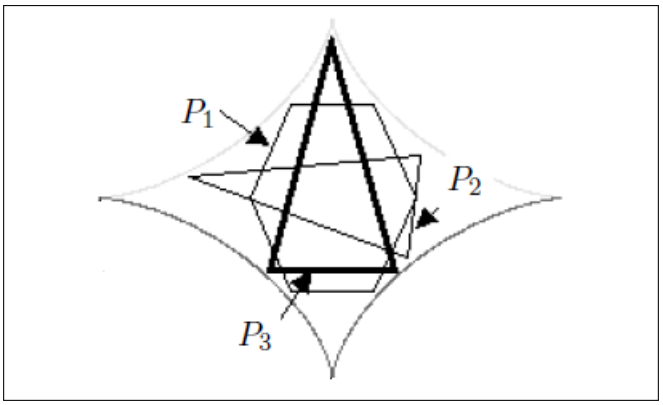


Figure 3.3: Three polytopes embedded within a non-convex set [VM98].

where $0 \leq \Delta t \leq 1$. Higher order terms are considered negligible. If the derivatives satisfy appropriate bounds, i.e., $\bar{h}^{(i)}(t) \leq \bar{h}_i, i = 1, \ldots, r$, then $\bar{h}(t + \Delta t)$ is bounded in the neighborhood of t according to

$$|\bar{h}(t + \Delta t)| \leq \bar{h}_0 + \bar{h}_1 \Delta t + \cdots + \bar{h}_r \frac{\Delta t^r}{r!} \tag{3.21}$$

As a result, given a choice for derivative bounds $\bar{h}_i, i = 1, \ldots, r$, the bound $\bar{h}_0$ can be determined as

$$\bar{h}_0 = -(\bar{h}_1 \Delta t + \cdots + \bar{h}_r \frac{\Delta t^r}{r!}). \tag{3.22}$$

Therefore, if $\bar{h}(t) \leq \bar{h}_0$ and its derivatives are bounded according to $\bar{h}^{(i)}(t) \leq \bar{h}_i, i = 0, \ldots, r$, then $\bar{h}(t + \Delta t) \leq 0$ in the neighborhood of t under the assumption that terms of order Δt^{r+1} and higher are negligible. Following the described scheme, each inequality constraint is evaluated on N equally spaced collocation points $t_1, \ldots, t_N$, chosen such that $t_0 \leq t_1 \leq \cdots \leq t_N \leq T$, with $0 \leq \Delta t \leq 1$ being the spacing between two consecutive points. At each point t_i, appropriate bounds on the derivatives of the constraint must be specified and a modified bound on the constraint is then obtained according to (3.22). If a polytopic approximation of the form (3.19) is available, this approach results in a total of $(N + 1)(r + 1)l$ linear inequalities on the $(k + 1)m$ coefficients.

- Conservative scheme

  Let $\bar{h}(t)$ be parameterized as linear combination of some basis functions $\gamma_i(t)$ so that the inequality constraint $\bar{h}(t) \leq 0$ is written in the form of

$$\alpha_1 \gamma_1(t) + \alpha_2 \gamma_2(t) + \cdots + \alpha_k \gamma_k(t) \leq 0 \tag{3.23}$$

  If the basis functions $\gamma_i(t)$ are chosen such that they are non-negative over the interval $[0, T]$, a sufficient condition for the inequality (3.23) to hold is that the coefficients α_i satisfy

$$\alpha_i \leq 0, \quad i = 1, \ldots, k \tag{3.24}$$

  It should be noted that $\gamma_i(t)$ need not to be the same as the basis functions used in (3.19). For example, one can use the monomial basis, $1, t, t^2, \ldots, t^k$, as $\gamma_i(t)$ over a domain $[0, T]$ regardless of the basis functions of choice for parameterizing the flat output. According to (3.19), the inequalities in (3.24) can be transformed into linear inequalities on a_{jk}.

- Min-max scheme

  To illustrate the idea behind the min-max scheme, consider one such inequality after the solution form has been imposed

$$c_0(t) + c_1(t)a_1 + c_2(t)a_2 + \cdots + c_k(t)a_k \leq 0, \quad t \in [0, T]. \tag{3.25}$$

  where $c_j(t)$ are time-varying coefficients with a minimum and maximum over $[0, T]$. Using these extrema and considering the set of possible coefficient sign distribution, the above inequality can be replaced by a single sufficient condition on a_k. This process should be repeated for every constraint until a set of sufficient conditions for the entire problem is obtained.

With the three schemes described above, the infinite set of inequality constraints in (3.16) is replaced by a finite set characterized by I linear inequalities on the coefficients $a_{jk} \in \mathbb{R}^{mk}$, provided that a convex approximation of $\bar{h}(.)$, as expressed in (3.18), is available. In general, the inequalities arising from these schemes are different; the conservative and min-max schemes might result in sets of inconsistent inequalities, which makes the collocation scheme the most commonly used among the three. The reader is referred to [VM98] for a comparison between these schemes. The major drawback of the above explained approach is that the inner approximation of the original feasible region with a polytope, though speeding up the computations, can lead to overly conservative and suboptimal solutions.

In Sec. 3.2.3 we study other methods for evaluating inequality constraints of the general form (3.16). But before that, we need to review some basic properties of the Bernstein basis polynomials, which are the building blocks for defining these alternative methods.

3.2 Bernstein polynomials and Bézier curves

A polynomial curve p is fully described with its coefficients in the polynomial basis of choice. The set of coefficients together with a value t of the independent variable are the input to algorithms that evaluate $p(t)$. Different polynomial bases, such as the power, Bernstein, Chebyshev, Legendre, etc. , with different properties, can exert influence over the quality and accuracy of the output of algorithms. [FG96] shows that Bernstein basis is *optimally stable*, and yields systematically smaller condition numbers for roots of arbitrary polynomials on a given interval. Hence, it can increase reliability of computations when used properly in floating-point arithmetic [Far08a]. [DS07] demonstrates that Bernstein basis is the best-behaved basis with respect to rounding errors in intersection algorithms. These attractive properties have given Bernstein basis a central role for representing polynomial curves and surfaces in computer-aided design (CAD)-systems.

Using polynomials in Bernstein form, such as Spline and Bézier curves, to parameterize a trajectory in motion planning problems, has also been explored in the literature [VP17c], [Cho+15]. It is proven in [Cic+17] that, for differntially flat systems, the solution to the approximate problem that is obtained by discretizing the OCP (2.8) using Bernstein polynomials converges to the optimal solution of the original problem. These special types of polynomial curves possess properties that, besides allowing for a geometrical interpretation of the design, can significantly reduce the computational complexity of solving the optimization problem. In the following, we give an overview of the most important properties and algorithms related to Bézier curves, that will be later employed in our trajectory-generation framework. A more comprehensive list of properties can be found in ([Far12]).

3.2.1 Bernstein Polynomial: definition and basic properties

Bernstein polynomials form the basis functions for Bézier curves. The Bernstein basis function of degree n is defined over the interval $[0, 1]$ as

$$B_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}(t)^i \quad t \in [0,1], \tag{3.26}$$

where $\binom{n}{i} = \frac{n!}{i!(n-i)!}$. Bernstein basis polynomials of degree $n = 0, \ldots, 5$ are visualized in Figure (3.4). Polynomials in Bernstein form were first introduced by Sergei Natanovich Bernstein in an elegant constructive proof for *Weierstrass approximation theorem*, which guarantees the existence of a polynomial $p_n(t)$ of a certain degree n that uniformly approximates a given function, continuous over its domain $[a, b]$, as closely as desired. The Bernstein polynomial approximation to any continuous function $f(t)$ defined over the interval $[0, 1]$ is specified as

$$p_n(t) = \sum_{i=0}^{n} f(\frac{i}{n})B_{i,n}(t). \tag{3.27}$$

By choosing a sufficiently high degree, $p_n(t)$ can satisfy any prescribed accuracy, i.e., for each $\delta > 0$ there exist an integer n_δ such that

$$|p_n(t) - f(t)| \leq \delta, \qquad t \in [0,1], \quad n \geq n_\delta. \tag{3.28}$$

The convergence behavior of the approximant (3.27) stems from the intrinsic properties of the Bernstein basis functions [Far08b]. In the following, we list some of these properties and the consequent relations between the behavior of the Bernstein polynomial of the form

$$p(t) = \sum_{i=0}^{n} c_i B_{i,n}(t) \tag{3.29}$$

over the interval $[0, 1]$ and its coefficients $c_0, \ldots, c_n \in \mathbb{R}$.

- Non-negativity: All Bernstein basis functions are non-negative on the interval $[0, 1]$.

$$B_{i,n}(t) \geq 0 \quad 0 \leq t \leq 1 \quad i = 0, \ldots, n \tag{3.30}$$

- Partition of unity: The Bernstein polynomials form a Partition of Unity, i.e. the sum of $B_{i,n}(t)$ for all i is equal to 1 at any $t \in [0, 1]$.

$$\sum_{i=0}^{n} B_{i,n}(t) = (1 - t + t)^n = 1 \tag{3.31}$$

- Symmetry: The basis functions $B_{k,n}(t)$ and $B_{n-k,n}(t)$ are symmetric about the interval mid-point $\frac{1}{2}$.

$$B_{n-i,n}(1-t) = B_{i,n}(t). \tag{3.32}$$

- Recursion: The basis polynomial of degree n can be generated using two Bernstein polynomials of degree $n - 1$, according to

$$B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t), \quad i = 0, \ldots, n \tag{3.33}$$

  with $B_{0,0}(t) = 1$ and $B_{i,n}(t) = 0$ for $i < 0$ and $i > n$.

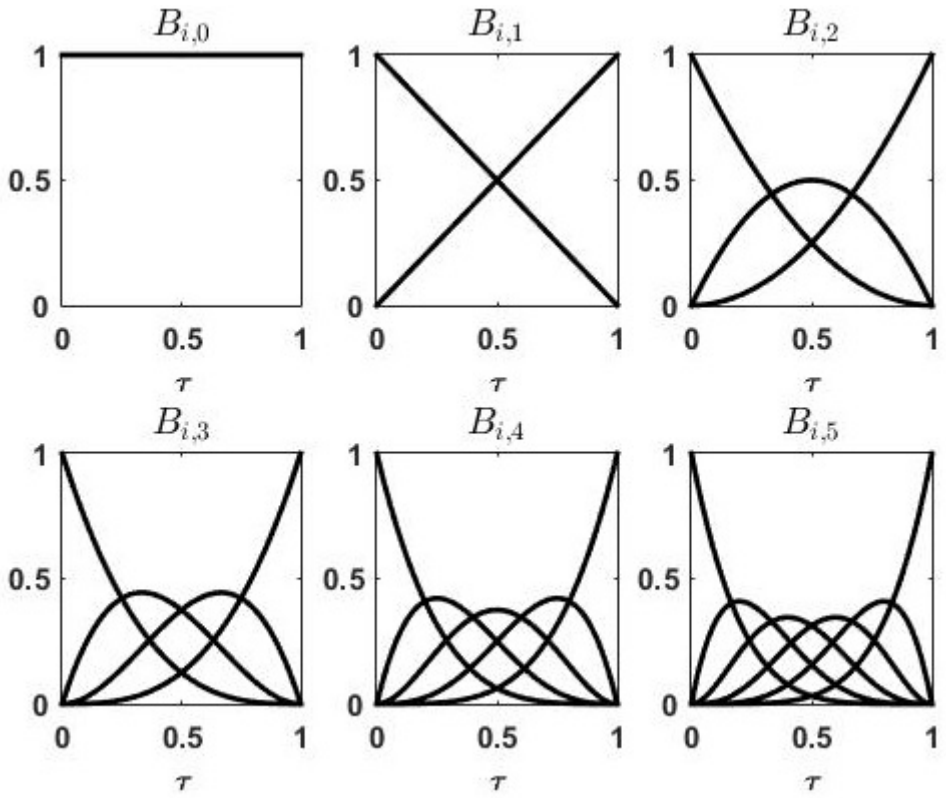


Figure 3.4: Bernstein basis functions of degree 0, 1, 2, 3, 4, and 5.

- Unimodality: The basis function $B_{i,n}(t)$ has an extremum at $t = \frac{i}{n}$. Also, for any parameter value t_* there is a corresponding index i such that

$$B_{0,n}(t_*) \leq \cdots \leq B_{i-1,n}(t_*) \leq B_{i,n}(t_*) \geq B_{i+1,n}(t_*) \geq \ldots B_{n,n}(t_*) \tag{3.34}$$

  i.e. $B_{i,n}(t_*)$ are unimodal with respect to the index i. At $t = t_*$, the coefficient c_i has the greatest influence on (3.29), while the influence of other control points declines monotonically as their indices deviates further from i.

- Upper and lower bounds: The non-negativity and partition of unity properties imply that for any parameter value $t \in [0, 1]$, (3.29) satisfies the inequality

$$\min_{i=0,\ldots,n} c_i \leq p(t) \leq \max_{i=0,\ldots,n} c_i \tag{3.35}$$

- End point values: The Bernstein polynomial (3.29) satisfies

$$p(0) = c_0, \quad p(1) = c_n. \tag{3.36}$$

- Degree elevation: $B_{i,n}(t)$ can be expressed in terms of the Bernstein polynomials of degree $n + 1$, according to

$$B_{i,n}(t) = (1 - \frac{i}{n+1})B_{i,n+1}(t) + \frac{i+1}{n+1}B_{i+1,n+1}(t). \quad i = 0, \ldots, n \tag{3.37}$$

  More generally, $B_{i,n}(t)$ can be written in terms of the basis functions of degree $n + r$ as

$$B_{i,n}(t) = \sum_{j=i}^{i+r} \frac{\binom{n}{i}\binom{r}{j-i}}{\binom{n+r}{j}} B_{j,n+r}(t). \quad i = 0, \ldots, n \tag{3.38}$$

- Variation diminishing property: The number N of real roots of the Bernstein polynomial $p(t)$ on the open interval $t \in (0, 1)$ is less than the number of sign variations in its coefficients by an even amount,

$$N = S(\bar{r}_0, \ldots, \bar{r}_n) - 2K, \tag{3.39}$$

  with K being a non-negative integer, $K \geq 0$.

- Relation to monomial basis: The Bernstein and monomial (power) bases of degree n are related by the expressions

$$t^i = \sum_{j=i}^{n} \frac{\binom{j}{i}}{\binom{n}{i}} B_{j,n}(t), \tag{3.40}$$

$$B_{i,n}(t) = \sum_{j=i}^{n} (-1)^{j-i} \binom{n}{j} \binom{j}{i} t^j. \quad i = 0, \ldots, n \tag{3.41}$$

  For $i = 0$, (3.40) induces the partition of unity property, and for $i = 1$ it yields the *linear precision* property expressed as

$$t = \sum_{i=0}^{n} \frac{i}{n} B_{i,n}(t). \tag{3.42}$$

  The above expression implies that the monomial t can be generated as the weighted sum of Bernstein polynomials of degree n with coefficients equally spaced in the interval $[0, 1]$.

- Mapping to an arbitrary domain: The interval $[0, 1]$ can be mapped to an arbitrary interval $[t_1, t_2]$ through proper scaling of the independent variable. The change of variable, $t \to rt$, maps the interval to $[0, r]$. Performing a binomial expansion, $B_{j,n}(rt)$ can be written as

$$B_{j,n}(rt) = \sum_{k=j}^{n} B_{j,k}(r) B_{k,n}(t), \quad j = 0, \ldots, n \tag{3.43}$$

  which allows the coefficients of $p(t)$ on $[0, r]$, as generated by subdivision at $t = r$ with the de Casteljau's algorithm (See Sec. 3.2.2.2), to be expressed in terms of the coefficients on $[0, 1]$. (3.43) can be generalized to obtain the corresponding basis defined on an arbitrary interval $[t_1, t_2] \subset [0, 1]$:

$$\bar{B}_{j,n}(t) = \binom{n}{j} \frac{(t_2 - t)^{n-j}(t - t_1)^j}{(t_2 - t_1)^n}, \quad j = 0, \ldots, n \tag{3.44}$$

  If $c_0, c_1, \ldots, c_n$ are the coefficients of the Bernstein polynomial $p(t)$ in the basis function on $[0, 1]$, its $\bar{c}_0, \bar{c}_1, \ldots, \bar{c}_n$ coefficients in the analogous basis, $\bar{B}_{j,n}(t)$ defined on the interval $t_1, t_2]$, can be obtained by a matrix multiplication, given by

$$\bar{c}_j = \sum_{i=0}^{n} c_i M_{ij} \tag{3.45}$$

The elements of the matrix, M_{ij}, can be expressed as sums of products of the basis functions $B_{i,n}(t)$ evaluated at t_1 and t_2:

$$M_{ij} = \sum_{k=max(0,i+j-n)}^{min(i,j)} B_{i-k,n-j}(t_1)B_{k,j}(t_2), \quad 0 \leq i,j \leq n \tag{3.46}$$

Since $B_{i-k,n-j}(0) \neq 0$ only when $k = i$ and $B_{k,j}(1) \neq 0$ only when $k = j$, for $[t_1, t_2] = [0, t_0]$ the elements of matrix M reduce to

$$M_{ij} = \begin{cases} B_{i,j}(t_0) & if \quad i \leq j \\ 0 & if \quad i > j \end{cases} \tag{3.47}$$

and for $[t_1, t_2] = [t_0, 1]$, they become

$$M_{ij} = \begin{cases} 0 & if \quad j < i \\ B_{i-j,n-j}(t_0) & if \quad j \geq i \end{cases} \tag{3.48}$$

Therefore, for these subintervals, the coefficients are immediately obtained from the original coefficients, as against the matrix multiplication in (3.45).

- Derivatives and integrals: The Bernstein basis functions satisfy

$$\frac{d}{dt}B_{i,n}(t) = n\big(B_{i-1,n-1}(t) - B_{i,n-1}(t)\big) \quad i = 0, \ldots, n \tag{3.49}$$

where $B_{-1,n-1} \equiv 0$ and $B_{n,n-1} \equiv 0$. Thus, the first derivative of the Bernstein polynomial $p(t)$ is also a Bernstein polynomial, of degree $n-1$, given by

$$p\prime(t) = \sum_{i=0}^{n-1} c_i' B_{i,n-1} \tag{3.50}$$

where $c_i' = n(c_{i+1} - c_i)$. The coefficients for higher-order derivatives are obtained as

$$c_i^{(q)} = (n-q+1)(c_{i+1}^{q-1} - c_i^{q-1}) \quad i = 0, \ldots, n-q \tag{3.51}$$

The indefinite integral of the Bernstein basis function is obtained as

$$\int B_{i,n}(t)dt = \frac{1}{n+1} \sum_{j=i+1}^{n+1} B_{j,n+1}(t). \tag{3.52}$$

Accordingly, the indefinite integral of the Bernstein polynomial $p(t)$ may be expressed as

$$\int p(t)dt = \sum_{i=1}^{n+1} \Big(\frac{1}{n+1} \sum_{j=0}^{i-1} c_j\Big) B_{i,n+1}(t) + const. \tag{3.53}$$

Noting that $\int B_{i,n}(t)dt = \frac{1}{n}$, the definite integral of $p(t)$ over $[0, 1]$ is given by

$$\int_0^1 p(t)dt = \frac{1}{n+1} \sum_{j=0}^{n} c_j \tag{3.54}$$

- Arithmetic operations: The set of Bernstein polynomials is closed under the arithmetic operations of addition, subtraction, multiplication, and composition. To add or subtract two Bernstein polynomials of the same degree, one can simply add or subtract their respective coefficients. Otherwise the degrees of the two polynomials must be matched using the degree elevation before adding or subtracting the coefficients. The coefficients corresponding to adding/subtracting two Bernstein polynomials $r(t)$ and $s(t)$ of degree m and n, $m > n$, with coefficients $r_0, \ldots, r_m$ and $s_0, \ldots, s_n$, are given by

$$c_i = r_i \pm \sum_{j=max(0,i-m+n)}^{min(n,i)} \frac{\binom{m-n}{i-j}\binom{n}{j}}{\binom{m}{i}} s_j, \quad i = 0, \ldots, m \tag{3.55}$$

and the coefficients of their product are obtained according to

$$c_i = \sum_{j=\max(0,i-n)}^{\min(m,i)} \frac{\binom{m}{j}\binom{n}{i-j}}{\binom{m+n}{i}} r_j s_{i-j}, \quad i = 0, \ldots, m+n \tag{3.56}$$

The composition of the two polynomials $r(t)$ and $s(t)$, $p(u) = r(s(u))$, is a Bernstein polynomial of degree mn and its coefficients are obtained through a recursive algorithm that populates a three-dimensional array with $r_{i,j}^k$ using the following expression for $k = 1, \ldots, m$, $i = 0, \ldots, m-k$, and $j = 0, \ldots, kn$

$$r_{i,j}^k = \frac{1}{\binom{kn}{j}} \sum_{l=max(0,j-n)}^{min(j,kn-n)} \binom{kn-n}{l}\binom{n}{j-l}\left((1-s_{j-l})r_{i,l}^{k-1} + s_{j-l}r_{i+1,l}^{k-1}\right) \tag{3.57}$$

where $r_{i,0}^0 = r_i, i = 0, \ldots, m$. Then, the coefficients are specified by

$$c_j = r_{0,j}^m, \quad j = 0, \ldots, mn \tag{3.58}$$

3.2.2 Bézier curves

With the advent of computer graphics, Bernstein polynomials gained popularity in computer aided design in the form of Bézier curves. The coincidental work of two French automotive engineers, Pierre Étienne Bézier of Renault and Paul de Faget de Casteljau of Citroën, who were concerned with developing new mathematical tools for intuitive design and interrogation of complex shapes, such as automobile bodies, led to the adoption of Bernstein form, characterized by what is now called a Bézier curve [Far12]. Besides inheriting simplicity of polynomials, Bézier curves have several other properties that allow designers to analyze and manipulate the curve shape in a simple and intuitive way. In the following, we explore geometric properties and algorithms associated with Bézier curves that make them appealing for describing the trajectories in motion planning problems.

3.2.2.1 Definition and shape features

Given a set of $n+1$ control points r_i, $i = 0, \ldots, n$, the corresponding Bézier curve of degree n is defined as

$$\mathrm{r}(\tau) = \sum_{i=0}^{n} r_i B_{i,n}(\tau). \tag{3.59}$$

where $r_i \in \mathbb{R}^2$ for planar curves and $r_i \in \mathbb{R}^3$ for spatial curves. Figure (3.5) shows a cubic Bézier curve with its control polygon obtained by drawing lines between consecutive control points.

Bézier curves have several geometric and analytical properties that can be particularly useful in trajectory generation applications. We provide a list of the most important ones below.

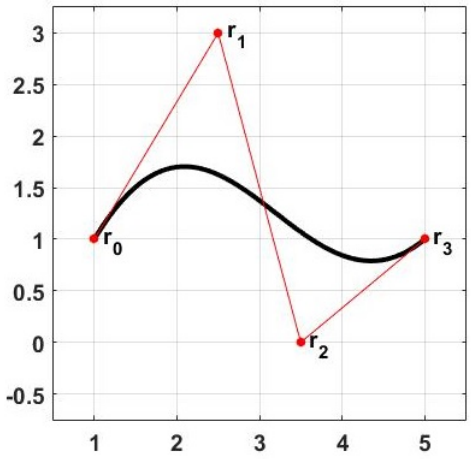


Figure 3.5: A cubic Bézier curve with control polygon

Geometry invariance property: The shape of a Bézier curve does not change under translation and rotation of its control points. This property is directly derived from the partition of unity of the Bernstein basis functions.

Endpoints Geometric property: The first and last control points, $\bar{r}_0$ and $\bar{r}_n$, are the endpoints of the curve, i.e.,

$$\mathrm{r}(0) = r_0, \quad \mathrm{r}(1) = r_n. \tag{3.60}$$

which can be simply verified by inserting $\tau = 0, 1$ in (3.59). Also, a Bézier curves is tangent to its control polygon at the endpoints.

$$\mathrm{r}'(0) = n(r_1 - r_0), \quad \mathrm{r}'(1) = n(r_n - r_{n-1}) \tag{3.61}$$

which is easily observed by taking the first derivative of $r(\tau)$ using (3.49). Figure 3.6 shows a quartic Bézier curve and its control polygon, and its first derivative called hodograph.

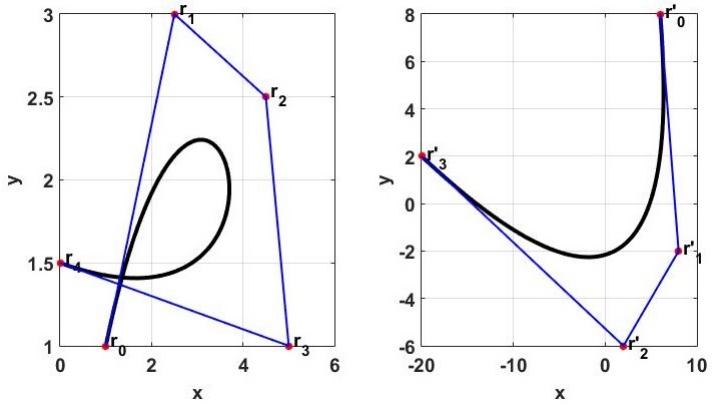


Figure 3.6: A quartic Bézier curve (left) and its first derivative (right). The derivative at the endpoints depend only on the first two and last two control points.

Convex hull property: A Bézier curve is contained within the convex hull of its control polygon, that is expressed as

$$\mathrm{r}(\tau) \in CH(\bar{R}), \quad \tau \in [0,1], \tag{3.62}$$

where the convex hull $CH(\bar{R})$ defined by the set of control points $\bar{R}$ is the boundary of the smallest convex set that contains all the points, that is,

$$CH(\bar{R}) = \{a_0 r_0 + \cdots + a_n r_n | a_0 + \cdots + a_n = 1, a_i \geq 0\}. \tag{3.63}$$

The convex hull property states that the entire Bézier curve is inside a computable region. The convex hull property for a quartic Bézier curve is visualized in Figure (3.7).

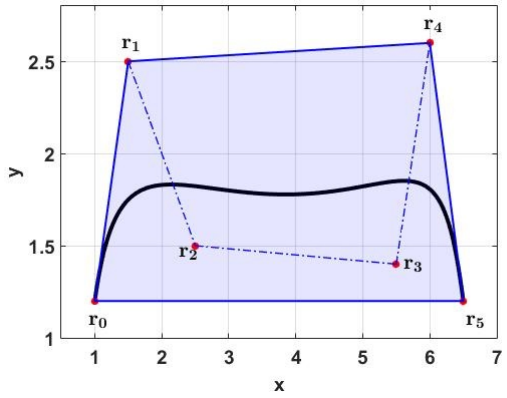


Figure 3.7: A cubic Bézier curve contained within the convex hull defined by its 4 control points. The control points are shown in red circles and control polyline in dashed line segments.

Variation diminishing property: A straight line may not intersect a planar Bézier curve more times than it intersects the control polygon. The same property holds true for a plane and a spatial Bézier curve. This implies that a Bézier curve cannot be more complex than its control polygon, i.e., the control polygon turns and twists more frequently than the Bézier curve itself. This property is illustrated in Figure (3.8).

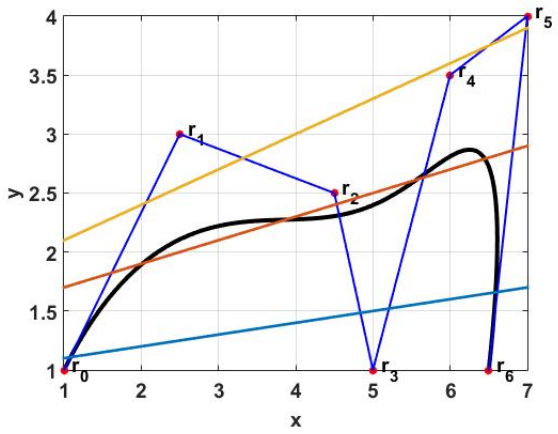


Figure 3.8: A Bézier curve oscillates no more than the piece-wise linear interpolant to its control points.

Symmetry: Relabeling the control points as $\bar{r}_i = r_{n-i}$ and using the symmetry property of the Bernstein basis polynomials yield

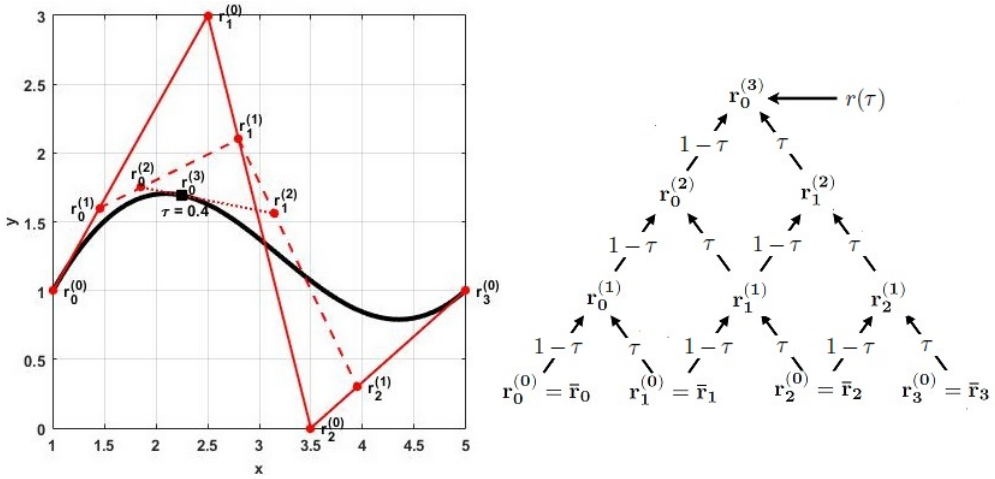


Figure 3.9: A cubic Bézier curve is evaluated $\tau = 0.4$ (left). The pyramid scheme of the de Casteljau's algorithm (right) for 3 iterations. The right and left sides of the pyramid show the control points for the new control polygons.

$$\sum_{i=0}^{n} r_i B_{i,n}(t) = \sum_{i=0}^{n} \bar{r}_i B_{i,n}(1-t) \tag{3.64}$$

3.2.2.2 Algorithms

De Casteljau's Algorithm:

Using the definition in (3.59) to evaluate a point on a Bézier curve can cause numerical instability for high degree curves. The de Casteljau's algorithm has established an alternative method to find $\mathrm{r}(\tau)$ using only the control points r_i. Given a specific value of $\tau \in [0,1]$, the algorithm starts by dividing each polyline between two consecutive control points r_i and r_{i+1}, $i = 0, \ldots, n-1$, in a ratio of $\tau : 1-\tau$ to obtain n new points, indicated by $r_i^{(1)}$. This process is repeated using the new set of points to obtain $r_i^{(2)}$, $i = 0, \ldots, n-2$. Repeating the subdivision n times yields a single point $r_0^{(n)}$ which can be proved to be the point on the curve corresponding to τ. Figure (3.9) provides a geometrical interpretation of the algorithm for a cubic curve with 4 control points and $\tau = 0.4$.

The above procedure can be expressed as a recursive formula. Initialized with $r_i^{(0)} = r_i$, at each iteration j a set of $n-j$ points is calculated by linear interpolations of the points in the previous iteration, according to

$$\begin{aligned} r_i^{(j)} &= (1-\tau_0) r_i^{(j-1)} + \tau_0 r_{i+1}^{(j-1)} \qquad && i = 0, \ldots, n-j \\ & && j = 1, \ldots, n \end{aligned} \tag{3.65}$$

The single point obtained at $j = n$ is the point on the curve corresponding to τ_0, that is,

$$r_0^{(n)} = \mathrm{r}(\tau_0). \tag{3.66}$$

The de Casteljau's algorithm also provides a subdivision scheme for Bézier curves. The two sets of $\{r_0^{(k)} | \quad k = 0, \ldots, n\}$ and $\{r_{n-k}^{(k)} | \quad k = 0, \ldots, n\}$, corresponding to the two sides of the de Casteljau's pyramid (Fig. 3.9), form the control points for two Bézier curves of degree n, defined over $[0, \tau_0]$ and $[\tau_0, 1]$, that divide the original curve $r(\tau)$ at τ_0. Figure (3.10) illustrates the application of the de Casteljau's algorithm for splitting a cubic Bézier curve into two pieces at different points on the curve.

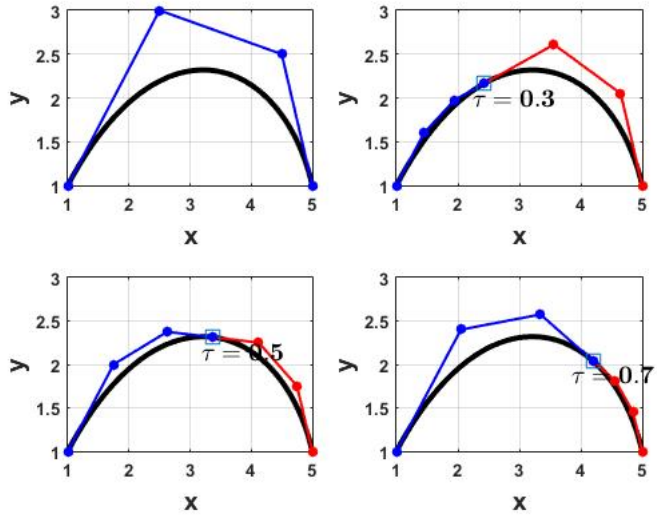


Figure 3.10: A cubic Bézier curve is divided into two Bézier curves of the same degree, $n = 3$, at $\tau = 0.3, 0.5, 0.7$, using the de Casteljau's algorithm.

Degree Elevation:

The Degree Elevation algorithm is incredibly useful for applications involving two or more Bézier curves with different degrees. It allows expressing the Bézier curve $\mathrm{r}(\tau)$, of true degree n, in the Bernstein basis of degree $n + r$, for all $r > 0$, without changing its shape. In order to obtain the higher degree Bézier curve, one should find the set of control points $r_0^{n+r}, \ldots, r_{n+r}^{n+r}$ such that (3.59) can be written as,

$$\mathrm{r}(\tau) = \sum_{i=0}^{n+1} r_i^{n+1} B_{i,n+1}(\tau). \tag{3.67}$$

The first and last control points must obviously be in the new set since the higher degree representation is supposed to retain the shape. The rest of the control points can be achieved by substituting (3.37) into (3.59). The following formula gives the new $n + 2$ control points for the unit degree elevation:

$$r_i^{n+1} = \frac{i}{n+1} r_{i-1}^n + (1 - \frac{i}{n+1}) r_i^n \quad i = 1, \ldots, n \tag{3.68}$$

where $r_0^{n+1} = r_0^n$ and $r_{n+1}^{n+1} = r_n^n$. It can be implied from (3.68) that each polyline will exactly contain one new control point. The control points for an m-fold degree elevation can be determined by using (3.69) as

$$r_i^{n+m} = \sum_{j=\max\{0,i-m\}}^{\min\{n,i\}} \frac{\binom{n}{j}\binom{m}{i-j}}{\binom{n+m}{i}} r_j^n. \quad i = 0, \ldots, n+m \tag{3.69}$$

Fig. 3.11 shows a cubic Bézier curve expressed in terms of basis functions of degree 4, 6, and 15 with the new set of control points obtained using (3.68).

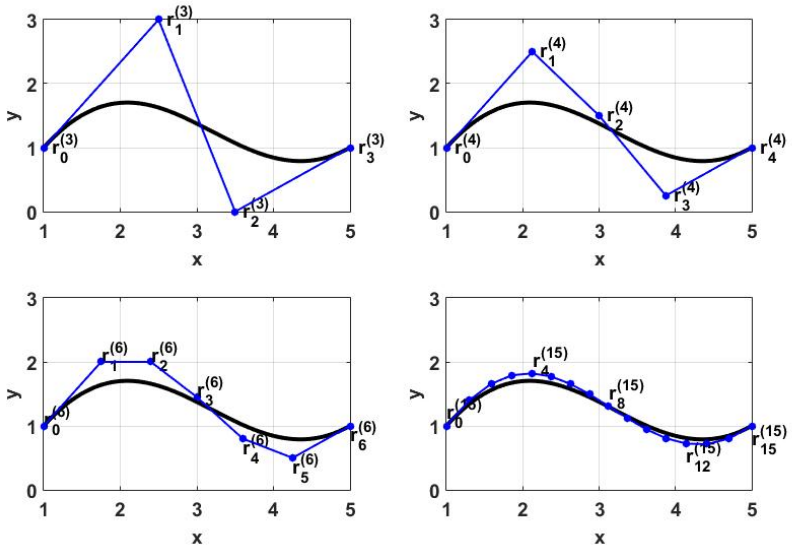


Figure 3.11: A cubic Bézier curve is expressed in terms of higher degree basis functions, $n = 4, 6, 15$, using the degree elevation algorithm.

Degree reduction:

The true degree of a Bézier curve can not be immediately determined from its control points. If $\mathrm{r}(\tau)$ is of the true degree $n - m$ then the power coefficients in monomial basis must satisfy the following conditions

$$\begin{aligned} &a_{n-1} \neq 0 \\ &a_n = \cdots = a_{n-m+1} = 0 \end{aligned} \tag{3.70}$$

Using (3.40), the coefficients a_k can be expressed in terms of the control points as,

$$a_k = \sum_{j=0}^{k} (-1)^{k-j} \binom{n}{k} \binom{k}{j} r_j, \quad k = 0, \ldots, n. \tag{3.71}$$

Assuming that the above conditions on a_k hold, the control points in the Bernstein basis of degree $n - m$ can be obtained from

$$r_k^{n-m} = \sum_{j=0}^{k} (-1)^{k-j} \frac{\binom{k-j+m-1}{m-1}\binom{n}{j}}{\binom{n-m}{k}} r_k^n, \quad k = 0, \ldots, n-m \tag{3.72}$$

Continuity Algorithm:

Complex curves can be represented with piece-wise Bézier curves, formed by joining several Bézier curves end-to-end, as an alternative to using higher degree curves. The continuity algorithm addresses geometric and parameter continuity between consecutive curves. Position continuity, i.e. G^0 continuity, requires the endpoints of two consecutive curves, $\mathrm{r}_i(\tau)$ and $\mathrm{r}_{i+1}(\tau)$, to coincide, that is

$$\mathrm{r}_i(1) = \mathrm{r}_{i+1}(0), \tag{3.73}$$

which is equivalent to $r_{n_i,i} = r_{0,i+1}$. Tangent continuity or G^1 can be enforced through the equality constraints given by

$$\begin{aligned}\dot{\mathrm{r}}_i(1) &= n_i(\|r_{n_i,i} - r_{n_i-1,i}\|)\mathbf{t} \\ \dot{\mathrm{r}}_{i+1}(0) &= n_{i+1}(\|r_{1,i+1} - r_{0,i+1}\|)\mathbf{t}\end{aligned} \tag{3.74}$$

where $\mathbf{t}$ is a common unit vector ensuring $r_{n_i-1,i}$, $r_{n_i,i} = r_{0,i+1}$, and $r_{1,i+1}$ are on the same line. The more stringent parametric continuity conditions, C^k, requires the k-th derivative and all lower derivatives of the consecutive segments to be equal at the joining point. In other words,

$$\frac{d^k \mathrm{r}_i(1)}{dt^k} = \frac{d^k \mathrm{r}_{i+1}(0)}{dt^k} \quad k \in \{0, \ldots, k\} \tag{3.75}$$

If we assume that the global parameter t runs over the interval $[t_i, t_{i+1}]$ for the i-th segment of the composite curve, then $\mathrm{r}_i(t)$ is defined as

$$\mathrm{r}_i(t) = \sum_{k=0}^{n_i} \bar{r}_{k,i} B_{k,n_i}(\tau_i(t)) \tag{3.76}$$

where the local parameter $\tau_i(t)$ is related to the global parameter by

$$0 \leq \tau_i = \zeta_i(t) = \frac{t - t_i}{t_{i+1} - t_i} \leq 1 \tag{3.77}$$

For the i-th and $i+1$-th segments, the first order parametric continuity or C^1 continuity can be stated as

$$\Delta t_{i+1} n_i (r_{n_i,i} - r_{n_i-1,i}) = \Delta t_i n_{i+1} (r_{1,i+1} - r_{0,i+1}) \tag{3.78}$$

where $\Delta t_i = t_{i+1} - t_i$. Likewise, the C^2 continuity condition can be expressed as

$$\begin{aligned}\frac{\Delta t_{i+1}}{\Delta t_i}(r_{n_i-1,i} - r_{n_i-2,i}) + n_i(n_{i+1} - 1) r_{n_i-1,i} + n_i r_{n_i,i} = \\ \frac{\Delta t_i}{\Delta t_{i+1}}(r_{1,i+1} - r_{2,i+1}) + n_{i+1}(n_i - 1) r_{1,i+1} + n_{i+1} r_{0,i+1}\end{aligned} \tag{3.79}$$

Higher-order parametric continuity conditions can be obtained likewise.

3.2.3 Evaluating Inequality constraints using B-spline and Bézier curves properties

The main reason for adopting Bernstein polynomials, i.e. B-spline and Bézier curves, for describing trajectories is that their properties and algorithms can be exploited to develop reliable algorithms for finding intersections between trajectories. The convex hull property and subdivision algorithms, in particular, have been extensively used in collision detection and trajectory generation problems. In the following, we first review existing algorithms and then we propose an efficient method for evaluating the inequality constraints, including collision-avoidance constraints, whose functions can be expressed in Bézier form. As we will explain in detail the proposed method converts the semi-infinite optimization problem associated with polynomial path parameterization into a standard finite dimensional optimization problem.

3.2.3.1 Literature Review

Finding the minimum distance between two parametric curves $\mathrm{c}_1(u)$ and $\mathrm{c}_2(v)$ can be turned into finding the roots of a system of nonlinear equations, expressed as

$$\begin{aligned} \mathrm{f}_u(u,v) &= 0, \\ \mathrm{f}_v(u,v) &= 0, \end{aligned} \tag{3.80}$$

where $\mathrm{f}(u,v)$ is the squared distance between $\mathrm{c}_1(u)$ and $\mathrm{c}_2(v)$, i.e.,

$$\mathrm{f}(u,v) = \big(\mathrm{c}_1(u) - \mathrm{c}_2(v)\big)^2. \tag{3.81}$$

There exist different methods for solving the system of nonlinear equations (3.80) (See [EK01], [PM02], [EG08], and [LS02]); however, as stated in [JC98], root finding methods have low robustness and efficiency. Alternative methods for computing the minimum distance between parametric curves and surfaces of general form can be found in [LT95], [LM95], and [LS02]. For NURBS, B-spline and Bézier curves, more efficient and robust algorithms are respectively proposed in [Che+10], [Che+09], and [Cha+11]. These methods are not only geometrically intuitive, but can also find the minimum distance to an arbitrary tolerance by using the unique properties of Bernstein basis polynomials.

The recursive algorithm proposed in [Cha+11] uses a pruning scheme to find the minimum distance between two Bézier curves (or surfaces) by repeatedly testing an exclusion criteria and subdividing the two curves. At each iteration, the algorithm computes a global upper bound of the minimum distance which converges to the actual minimum distance $d_{\min}$. The algorithm then subdivides the two curves, using the de Casteljau's algorithm, and finds lower bounds of the minimum distance between each pair of the curve segments. If the computed lower bound is greater than the global upper bound, then the pair will be excluded. Otherwise, subdivision of the curve segments will be repeated until the two bounds are arbitrarily close. For two given Bézier curves $\mathrm{r}(\tau)$ of degree n and $\mathrm{s}(\tau)$ of degree m, and a given tolerance ϵ, the algorithm is summarized below.

Algorithm 1 Computing the minimum distance between two Bézier curves $\mathrm{r}(\tau)$ and $\mathrm{s}(\tau)$ [Cha+11]

Input: $\{r_0, \ldots, r_n\}$,$\{s_0, \ldots, s_m\}$
Input: $\overline{d}$='Initial estimate of the upper bound'
if upperBound$(\mathrm{r},\mathrm{s}) \leq \overline{d}$ **then**
  $\overline{d} \leftarrow$ upperBound(r,s)
end if
if lowerBound$(\mathrm{r},\mathrm{s}) \geq \overline{d}(1-\epsilon)$ **then**
else
  Subdivide r into $\mathrm{r}^{(1)}$ and $\mathrm{r}^{(2)}$
  Subdivide s into $\mathrm{s}^{(1)}$ and $\mathrm{s}^{(2)}$
  $\overline{d} \leftarrow \min(\overline{d}, \text{Algorithm 1}(\mathrm{r}^{(1)}, \mathrm{s}^{(1)}, \overline{d}))$
  $\overline{d} \leftarrow \min(\overline{d}, \text{Algorithm 1}(\mathrm{r}^{(1)}, \mathrm{s}^{(2)}, \overline{d}))$ ▷ Only for the spatial minimum distance
  $\overline{d} \leftarrow \min(\overline{d}, \text{Algorithm 1}(\mathrm{r}^{(2)}, \mathrm{s}^{(1)}, \overline{d}))$ ▷ Only for the spatial minimum distance
  $\overline{d} \leftarrow \min(\overline{d}, \text{Algorithm 1}(\mathrm{r}^{(2)}, \mathrm{s}^{(2)}, \overline{d}))$
end if
return $\overline{d}$

The upper bound, $\overline{d}$, can be obtained by sampling any two points on the curves; however, making use of the end point interpolation property, the distance between the closest pair of endpoints can be selected as $\overline{d}$, i.e.,

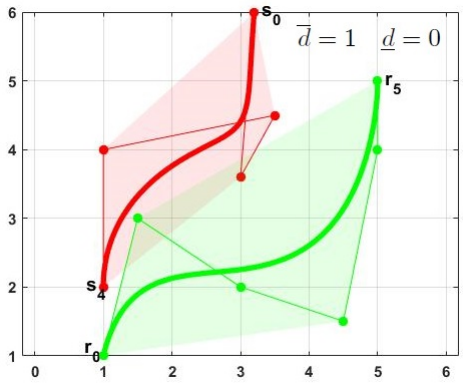


Figure 3.12: A Bézier curve of degree 4 (red) and a Bézier curve of degree 5 (green). The solid lines are the control polylines and the shaded areas are the convex hulls of the curves. According to (3.82), the upper bound of the minimum distance between the two curves is $\|r_0 - s_4\| = 1$, and the lower bound, i.e., the distance between the convex hulls, is 0.

$$\overline{d} = \min\{\|r_0 - s_0\|, \|r_0 - s_m\|, \|r_n - s_0\|, \|r_n - s_m\|\} \tag{3.82}$$

where r_0, r_n, s_0, and s_m are the first and last control points of $r(t)$ and $s(t)$, respectively. The lower bound of the minimum distance, $\underline{d}$, is estimated by the minimum distance between the convex hulls of the given Bézier curves, which can be computed using the Gilbert-Johnson-Keerthi (GJK) distance algorithm of convex shapes. Since the GJK algorithm only requires a supporting vertex of the convex hull in a given directional vector, there is no need to explicitly construct the convex hulls (The GJK algorithm is explained in the next Chapter). An example for the upper bound and lower bound of the minimum distance between two Bézier curves is shown in Fig. 3.12.

Fig. 3.13 shows three examples of the obtained minimum spatial and temporal distance computation between two planar Bézier curves of different degrees using Algorithm 1 with $\epsilon = 10^{-8}$. In order to find the temporal minimum distance with Algorithm 1, the upperBound subroutine must only compare the two values $\|r_0 - s_0\|$ and $\|r_n - s_m\|$, and the two curves must be subdivided at the same parameter value (Line 6 and Line 7). Algorithm 1 can also be modified to find the extrema of a Bézier curve.

This Algorithm is used in [Kie+22] to ensure spatial or temporal separation of flight trajectories. The proposed framework in [Kie+22] enforces the minimum distance between two trajectories, expressed as Bézier curves, to be greater than the minimum safe distance. This approach can circumvent the problem associated with time gridding, and does ensure the satisfaction of collision-avoidance constraints at all time instances [Cho17], yet it suffers from a major drawback; incorporating the non-smooth function for computing $d_{\min}$, i.e. Algorithm 1, in the trajectory generation problem calls for non-smooth optimization techniques such as sub-gradient and bundle methods, which in turn cause a significant increase in computational costs of generating trajectories.

A more efficient method for treating the collision-avoidance constraint in trajectory generation problems is proposed in [MVP16] for B-spline curves. This method can be used for evaluating any inequality constraint in the problem, provided that it is expressed as a B-spline. For the spline function, given by

$$\mathrm{r}(t) = \sum_{i=0}^{n} r_i N_{i,k}(t), \tag{3.83}$$

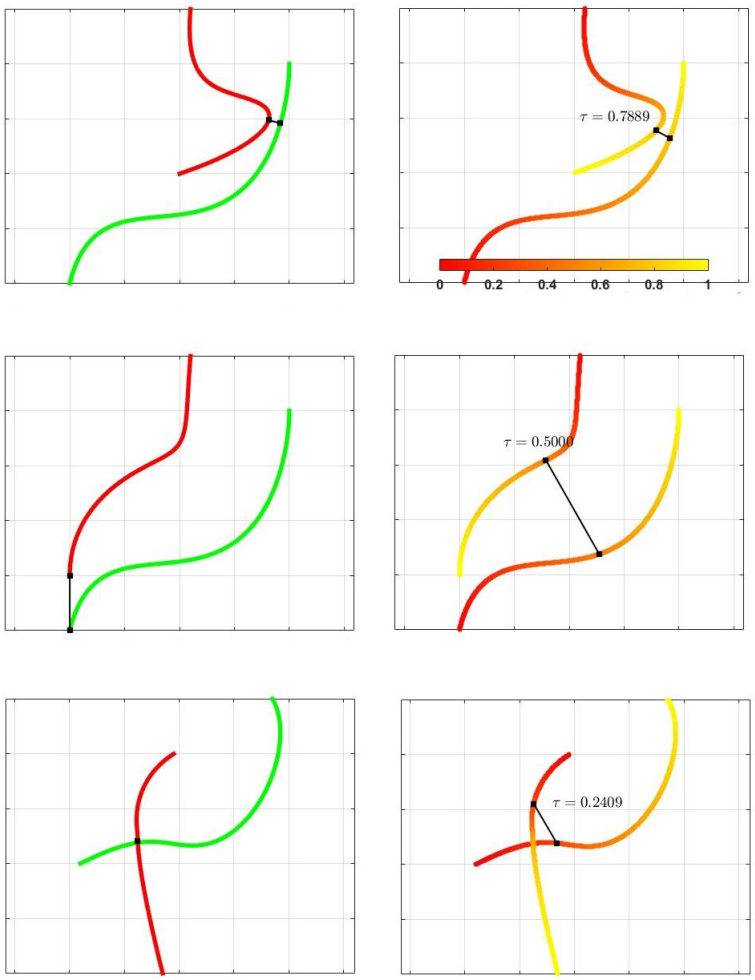


Figure 3.13: Comparing the minimum spatial (**left**) and temporal (**right**) distance between a Bézier curve of degree 4 (red) and a Bézier curve of degree 5 (green). The parameter value is indicated in the color bar to show the temporal evolution of the curves. The results are computed using Algorithm 1 with $\epsilon = 10^{-8}$ in < 4 ms.

where $N_{i,k}$ is the B-spline basis function of degree k, any inequality constraint of the form $\mathrm{r}(t) \leq 0$ is replaced with the finite set of sufficient conditions on its control points, i.e.,

$$r_i \leq 0, i = 0, \ldots, n \tag{3.84}$$

The above set of constraints is directly derived from the fact that a B-spline curve is contained in the convex hull of its control points (or de Boor points). (The reader is referred to Appendix C for a brief review of the propoerties of the B-spline basis functions). This method, known as B-spline relaxation, transforms the semi-infinite optimization problem, involving (3.16), into an efficiently solvable finite dimensional problem. However, the distance between the control polygon and the B-spline curve itself introduces conservatism, which can be reduced by representing the curve in a higher dimensional B-spline bases that includes the original one [VP17c]. Such bases can be obtained by inserting extra knots. Introducing a new knot is followed by removing $k-1$ of the original control points and replacing them with k new control points. Considering that a single knot $\bar{t} \in [t_l, t_{l+1})$ is inserted into the knot vector $T = [t_0, t_1, \ldots, t_l, t_{l+1}, \ldots]$, the B-spline curve (3.83) can be expressed in the new basis functions as

$$\mathrm{r}(t) = \sum_{i=0}^{n+1} \bar{r}_i \bar{N}_{i,k}(t), \tag{3.85}$$

where

$$\begin{aligned} \bar{N}_{i,k}(t) &= N_{i,k}(t), && i = 0, \ldots, l-k-1, \\ \bar{N}_{i,k}(t) &= N_{i-1,k}(t), && i = l+2, \ldots, n+1. \end{aligned} \tag{3.86}$$

The new control points can be obtained from the Boehm's algorithm [PBP02] as

$$\bar{r}_i = (1-\alpha_i) r_{i-1} + \alpha_i r_i, \tag{3.87}$$

where α_i is the ratio of dividing the affected knot span and is obtained as

$$\alpha_i = \begin{cases} 1 & i \leq l-k \\ 0 & i \geq l+1 \\ \frac{\bar{t}-t_i}{t_{i+k}-t_i} & l-k+1 \leq i \leq l \end{cases} \tag{3.88}$$

More general algorithms, such as Oslo algorithm, for inserting several knots into the knot vector exist. These algorithms can be used to represent a B-spline curve in a higher dimensional basis for reducing the conservatism in the constraint set (3.84). However, it also increases the number of control points, and thus the number of constraints. Therefore, it is necessary to make a trade-off between conservatism and computational burden. Overall, B-spline relaxaion is an efficient method for evaluating inequality constraints whose functions can be expressed in B-spline basis. Yet, a few important questions have remained unanswered in [VP17c] including where the new knots should be inserted or how many knot insertions are necessary so that the new control points are sufficiently close to the B-spline curve.

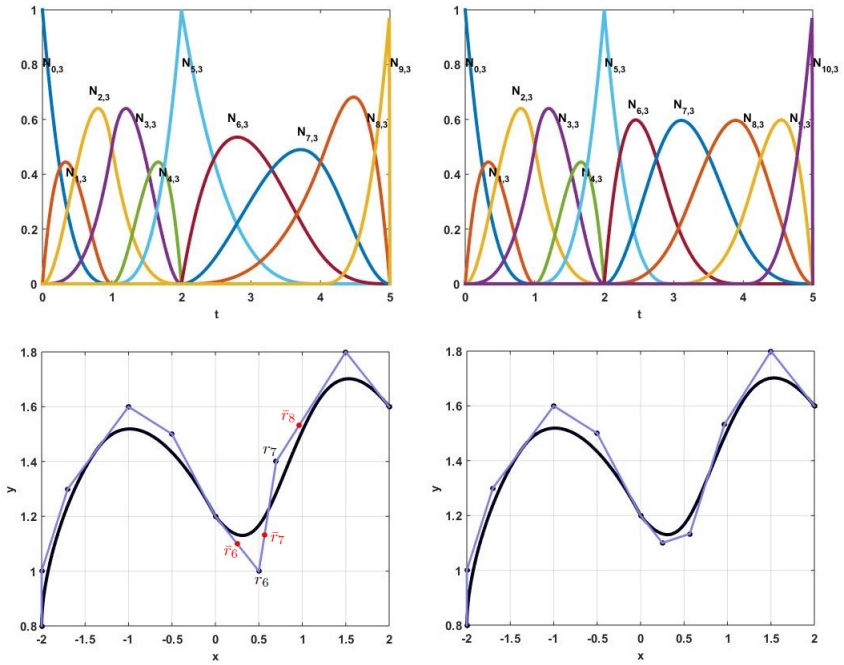


Figure 3.14: A B-spline curve of degree 3 (bottom left) and the corresponding B-splines defined over the knot vector $T = [0, 0, 0, 0, 1, 1, 2, 2, 2, 4, 5, 5, 5, 5]$ (**top left**). Figures on the right show the effect of inserting a single knot at $t = 3$ on the B-splines and the control polyline. The three new control points, generated with Boehm's algorithm and shown with red dots, replace the original control points r_6 and r_7.

3.2.3.2 Evaluating inequality constraints in Bernstein form

Here, we leverage Bézier curve properties and algorithms to efficiently evaluate inequality constraints, and convert the semi-infinite optimization problem involving (3.16) into a computationally tractable one containing a countable number of constraints. Without loss of generality, we assume that $\bar{h}(.)$ in (3.16) is a scalar-valued function. If this function can be represented as a Bernstein polynomial of the form

$$\bar{h}(\tau) = \sum_{i=0}^{n_h} \bar{h}_i B_{i,n_h}(\tau), \tag{3.89}$$

with control points, $\bar{h}_i \in \mathbb{R}, i = 0, \ldots, n_h$, then the inequality constraint $\bar{h}(\tau) \leq 0, \tau \in [0, 1]$ can be replaced by a finite set of constraints on the control points,

$$\bar{h}_i \leq 0 \quad for \quad i = 0, \ldots, n_h. \tag{3.90}$$

Therefore, owing to non-negativity and partition of unity of Bernstein basis functions, the original (infinitely many) inequality constraint is converted into a finite set of constraints. The resulting set of constraints will guarantee that $\bar{h}(\tau) \leq 0$ is satisfied over the entire domain $[0, 1]$. It should be noted that $\bar{h}_i$ are functions of the optimization variables, i.e. the coefficients a_{jk} in (3.17).

The main problem with the above approach is that it can lead to an overly conservative set of constraints due to the existing gap between the control points and the actual curve. The conservatism can be reduced by refining the representation of $\bar{h}(\tau)$ with closer control

points to the curve obtained from degree elevation or subdivision (using the de Casteljau's algorithm). The new control points, obtained with either method, are convex combinations of the original ones resulting in numerically stable algorithms. Reducing the conservatism using a refined control polygon of $\bar{h}(\tau)$ increases the number of constraints, and thus, making a trade-off between conservatism and computational cost is essential. Nonetheless, the optimization variables remain the same with repeated degree elevation or subdivision [SCP20].

As we will show below, the sequence of control polygons generated with repeated degree elevation or subdivision converges to the underlying Bézier curve, $\bar{h}(\tau)$, yet, while the former converges linearly, the latter can converge quadratically with respect to the subdivision level. Furthermore, doubling the degree of $\bar{h}(\tau)$ by repeated degree elevation from $2^k n_h$ to $2^{k+1} n_h$ requires $\binom{2^{k+1}}{2^{n_h}+1} - \binom{2^k}{2^{n_h}+1} \approx 3\binom{2^k}{2^{n_h}+1}$ additions and multiplications, while generating the same number of control points with subdivision at midpoints only costs $2^k\binom{n_h}{2}$ operations [NPL98]. More importantly, subdivision with the de Casteljau's algorithm allows refining the control polygon locally, whereas degree elevation affects the entire control polygon (cf. Fig. 3.10 and Fig. 3.11). Therefore, throughout the thesis, we use subdivision using the de Casteljau's algorithm to obtain refined control polygons of a Bézier curve.

Using (3.65), the Bézier control polygon of $\bar{h}(\tau)$ can be obtained over any number of adjacent intervals, $[0, \tau_0], [\tau_0, \tau_1], \ldots, [\tau_{k-1}, 1]$, by repeated subdivision. The control polygons together construct the composite control polygon of $\bar{h}(\tau)$ over $[0, 1]$ with $kn_h + 1$ distinct vertices. For the Bézier representation of the curve segment over the sub-interval $[\bar{\tau}, \bar{\tau} + n_h\Delta\tau]$ with control points $\bar{h}_{i,[\bar{\tau},\bar{\tau}+n_h\Delta\tau]}, i = 0, \ldots, n_h$, there is a constant c independent of $\bar{\tau}$ such that [PBP02]

$$\max_i \left|\bar{h}(\bar{\tau} + i\Delta\tau) - \bar{h}_{i,[\bar{\tau},\bar{\tau}+n_h\Delta\tau]}\right| \leq c\Delta\tau^2 \tag{3.91}$$

As we will study in the next section, the lowest possible constant c for which the above estimate holds can be obtained for infinity norm. It can be observed from (3.91) that the Bézier control polygon of a small segment of the curve is a fairly good approximation of the segment. It can also be shown that the distance between the Bézier segment and its linear interpolant, given by

$$\mathrm{l}(\tau) = \bar{h}_{0,[\bar{\tau},\bar{\tau}+n_h\Delta\tau]}(1-\tau) + \bar{h}_{n_h,[\bar{\tau},\bar{\tau}+n_h\Delta\tau]}\tau, \tag{3.92}$$

is bounded by $\frac{1}{8}n_h(n_h-1)\max\left\{\left|\Delta_2\bar{h}_{i,[\bar{\tau},\bar{\tau}+n_h\Delta\tau]}\right| \mid i = 0, \ldots, n_h-2\right\}$. Therefore, if

$$\max_{i=0,\ldots,n_h-2}\left|\Delta_2\bar{h}_{i,[\bar{\tau},\bar{\tau}+n_h\Delta\tau]}\right| \leq \epsilon \tag{3.93}$$

the Bézier control polygon over the sub-interval $[\bar{\tau}, \bar{\tau} + n_h\Delta\tau]$ is close to a line segment. The above bound offers a simple measure of the straightness of the Bézier control polygon using the second forward differences of the control points, and can be used as a stopping criteria for an algorithm that computes an approximant Bézier polygon of a curve with repeated subdivision.

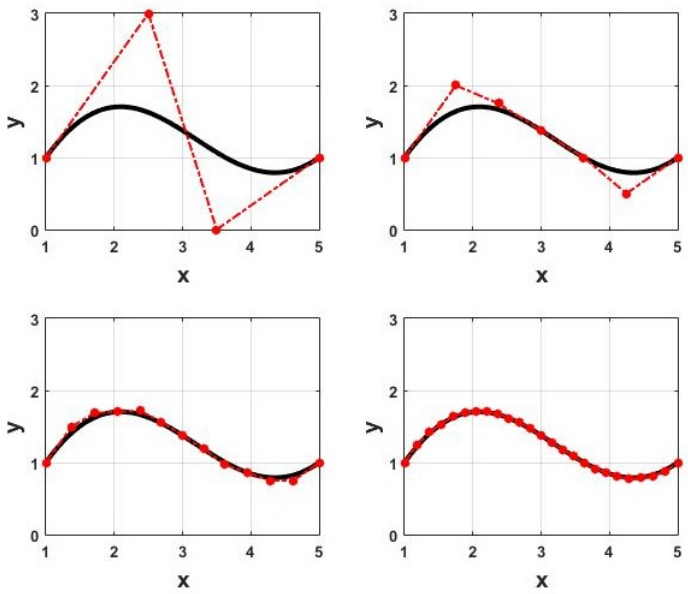


Figure 3.15: A cubic Bézier curve (**top left**) is subdivided into two Bézier curves of the same degree (**top right**) using the de Casteljau's algorithm. Successive refinement of the original control polygon after 2 (**bottom left**) and 3 (**bottom right**) subdivisions. The composite control polygon generated by repeated subdivisions converges to the Bézier curve.

3.2.4 Quantitative bounds on the distance between a Bézier curve and its control polygon

In this section, we employ the techniques and results in [NPL99] to compute the distance between a scalar-valued polynomial in Bézier form and its control polygon. As we will see, quantitative bounds on the maximum distance can be obtained from the sequence of the control points and some constant that depend on the degree of the polynomial [MZ06]. We will show that these bounds can be used in combination to obtain a polygonal approximation of Bézier curves with improved error bound compared to the original control polygon. We will also study the convergence behavior of repeated subdivision and degree elevation. Finally, we will show that the composite control polygon generated with subdivision (or degree elevation) can be locally and adaptively refined to meet a given tolerance.

3.2.4.1 Bézier control polygon

Consider the one dimensional Bézier curve of degree n, given by

$$\mathrm{r}(\tau) = \sum_{i=0}^{n} r_i B_{i,n}(\tau) \tag{3.94}$$

where $r_i \in \mathbb{R}, i = 0, \ldots, n$. The control polygon $l(\tau)$ of $\mathrm{r}(\tau)$ is obtained by connecting the points (τ_k, r_k) where the first coordinates are the Greville abscissae, i.e., $\tau_k \coloneqq \frac{k}{n}$. The k-th piece of the control polygon, $l_{[\tau_k, \tau_{k+1}]}$, is a line segment over the interval $[\tau_k, \tau_{k+1}]$, and is defined as

$$\begin{aligned} l_{[\tau_k,\tau_{k+1}]} &= r_k \frac{\tau_{k+1} - \tau}{\tau_{k+1} - \tau_k} + r_{k+1} \frac{\tau - \tau_k}{\tau_{k+1} - \tau_k} \\ &= r_k(k + 1 - n\tau) + r_{k+1}(n\tau - k). \end{aligned} \tag{3.95}$$

Therefore, on the interval $[\tau_k, \tau_{k+1}]$, we have

$$\mathrm{r}(\tau) - l(\tau) = \sum_{i=0}^{n} r_i \alpha_{ki}(\tau), \tag{3.96}$$

where

$$\alpha_{ki}(\tau) \coloneqq B_{i,n}(\tau) - \begin{cases} k+1-n\tau & \text{if } i = k \\ n\tau - k & \text{if } i = k+1 \\ 0 & \text{else} \end{cases} \tag{3.97}$$

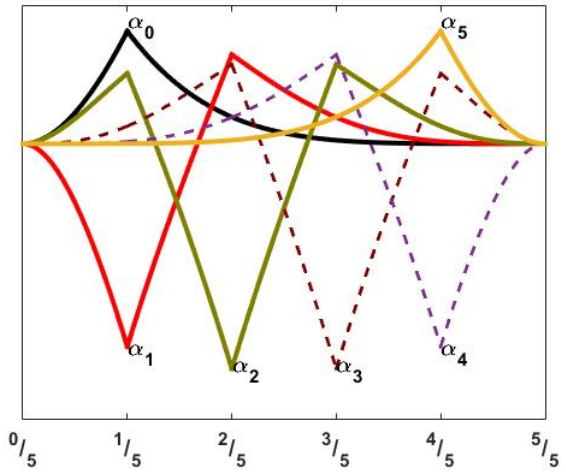


Figure 3.16: $\alpha_i(\tau)$ for $n = 5$ and $i = 0, \ldots, 5$. For any partcular value of τ, the sum of α_i is 0, i.e., $\sum_{i=0}^{n} \alpha_i(\tau) = 0$.

The partition of unity property of Bernstein bases implies that

$$\sum_{i=0}^{n} \alpha_{ki} = 0, \tag{3.98}$$

and from Eq. (3.40), we obtain

$$\sum_{i=0}^{n} i\alpha_{ki} = 0. \tag{3.99}$$

As a result, α_{ki} satisfies

$$\sum_{j=0}^{i} (i-j)\alpha_{kj} = \sum_{j=i}^{n} (j-i)\alpha_{k,j}, \tag{3.100}$$

Accordingly, $\alpha_{ki}(\tau)$ can be expressed as the centered second differences of the non-negative functions

$$\beta_{ki}(\tau) \coloneqq \sum_{j=0}^{i} (i-j)\alpha_{kj}(\tau) = \begin{cases} \sum_{j=0}^{i} (i-j)B_{j,n}(\tau) & \text{for } 0 \le i \le k \\ \sum_{j=i}^{n} (j-i)B_{j,n}(\tau) & \text{for } k+1 \le i \le n \end{cases} \tag{3.101}$$

that is

$$\alpha_{ki} = \Delta_2 \beta_{ki} = \beta_{k,i+1} - 2\beta_{ki} + \beta_{k,i-1} \quad \text{for } 1 \leq i \leq n \tag{3.102}$$

assuming that $\beta_{k,-1}(\tau) = \beta_{k,n+1}(\tau) = 0$.

It can be observed from (3.101) that $\beta_{k0}(\tau) = \beta_{kn}(\tau) = 0$. Fig. 3.17 shows $\beta_i(\tau)$ for $d = 5$ and $i = 1, 2, 3, 4$. $\beta_i(\tau) = \beta_{ki}(\tau)$ on $[t_k, t_{k+1}]$, and have a maximum at $\tau_i = \frac{i}{n}$.

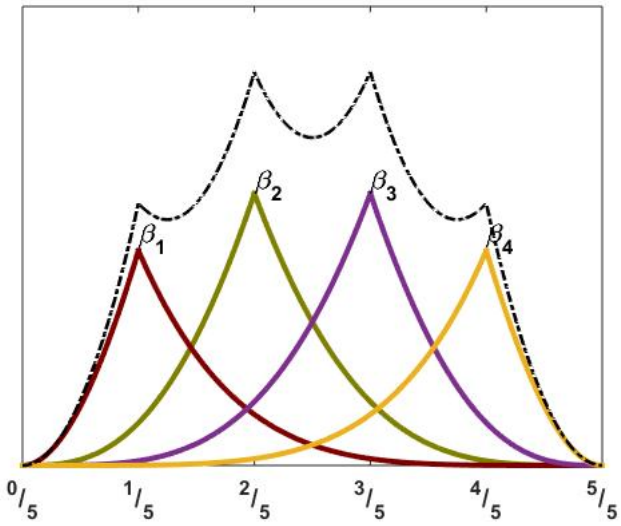


Figure 3.17: $\beta_i(\tau)$ for $n = 5$ and $i = 1, 2, 3, 4$. $\beta_i(\tau)$ is monotonically increasing on $[0, \tau_i]$ and decreasing on $[\tau_i, 1]$, where $\tau_i = \frac{i}{n}$. The dashed line shows their piece-wise sum, i.e., $\sum_i \beta_{ki}(\tau)$.

As we will see below, using the second anti-differences functions, $\beta_{ki}(\tau)$, leads to bounds that include second differences of the control points. Choosing m-th differences, $m > 2$, of the control point sequence does not result in better bounds than those of second difference [NPL98]. Here, we also study the result for the first difference, i.e., $m = 1$.

The functions $\alpha_{ki}(\tau)$ can be expressed as the first differences of the functions $\gamma_{ki}(\tau)$, given by

$$\gamma_{ki}(\tau) \coloneqq \sum_{j=0}^{i} \alpha_{kj}(\tau) \coloneqq \sum_{j=0}^{i} B_{j,n}(\tau) - \begin{cases} 0, & \text{if } i < k \\ k + 1 - n\tau, & \text{if } i = k \\ 1, & \text{if } i > k \end{cases} \tag{3.103}$$

that is

$$\alpha_{ki} = \Delta \gamma_{ki} \coloneqq \gamma_{ki} - \gamma_{k,i-1} \tag{3.104}$$

It can be readily noted from (3.103) that, over the interval $[\tau_k, \tau_{k+1}]$, γ_{ki} is positive for $i < k$ and negative for $i > k$ while γ_{kk} changes sign over the interval. For $d = 5$ and $i = 0, \ldots, 4$, $\gamma_i(\tau) = \gamma_{ki}(\tau)$ on $[\tau_k, \tau_{k+1}]$ are displayed in Fig. 3.18.

3.2.4.2 Bounding functions

The distance between the Bézier curve $\mathrm{r}(\tau)$ and its control polygon over the interval $[t_k, t_{k+1}]$ is measured by

$$\|\mathrm{r} - l\|_{p,[\tau_k,\tau_{k+1}]} = \max_{\tau \in [\tau_k, \tau_{k+1}]} \|\mathrm{r}(\tau) - l(\tau)\|_p, \tag{3.105}$$

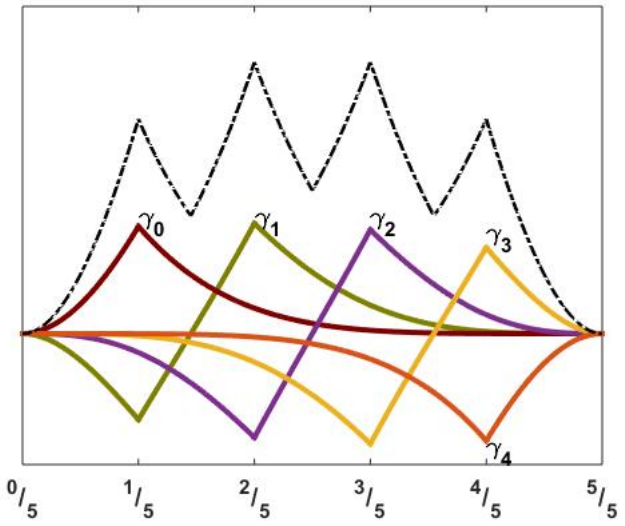


Figure 3.18: The first anti-differences of α_{ki}, $\gamma_{ki}(\tau)$, for $n = 5$ and $i = 0, \ldots, 4$. The dashed line shows the piece-wise sum of the absolute of γ_i, i.e., $\sum_i |\gamma_{ki}|$.

where $\|.\|_p, p \geq 1$ is the p-norm. Sharp and easily computable upper bounds of the distance for $\tau \in [0, 1]$ can be expressed in terms of the p-norm second differences of the control points sequence and constants depending on the degree [NPL99], i.e.,

$$\|\mathrm{r}(\tau) - l(\tau)\|_{p,[0,1]} = \max_{0 \leq k \leq n-1} \max_{\tau \in [\tau_k, \tau_{k+1}]} \|\mathrm{r}(\tau) - l(\tau)\|_p \leq N_p(d) \|\Delta_2 r\|_p. \tag{3.106}$$

In the following, we first consider $p = \infty$, and compare the two cases of first and second differences.

- Upper bound including $\|\Delta_2 r\|_\infty$

For $p = \infty$, the distance between the scalar Bézier curve r and its control polygon l on a given interval $[\tau_k, \tau_{k+1}]$ is defined by

$$\|\mathrm{r}(\tau) - l(\tau)\|_{\infty,[\tau_k,\tau_{k+1}]} \coloneqq \max_{\tau \in [\tau_k, \tau_{k+1}]} |\mathrm{r}(\tau) - l(\tau)| \tag{3.107}$$

Over the interval $[0, 1]$, a bound on the distance can be computed using the Hölder's inequality (3.125) as

$$\begin{aligned} \|\mathrm{r} - l\|_{\infty,[0,1]} &= \max_{0 \leq k \leq n-1} \| \sum_{i=0}^{n} r_i \alpha_{ki} \|_k \\ &= \max_{0 \leq k \leq n-1} \| \sum_{i=0}^{n} \Delta_2 \beta_{ki} r_i \|_k \\ &= \max_{0 \leq k \leq n-1} \| \sum_{i=0}^{n} \beta_{ki} \Delta_2 r_i \|_k \\ &\leq \max_{0 \leq k \leq n-1} \| \|\beta_{ki}(.)\|_1 \|_k \|\Delta_2 r\|_\infty \end{aligned} \tag{3.108}$$

where $\|.\|_k = \|.\|_{\infty,[\tau_k,\tau_{k+1}]}$, and $\|\Delta_2 r\|_\infty$ is the maximum absolute value of the centered second differences of the sequence of control points, given by

$$\|\Delta_2 r\|_\infty \coloneqq \max_{0\le i\le n} |\Delta_2 r_i| \quad \text{and} \quad \Delta_2 r_i \coloneqq r_{i-1} - 2r_i + r_{i+1}. \tag{3.109}$$

Using the conversion to monomial bases formula (3.40), $\|\beta_{ki}(.)\|_1$ can be computed as

$$\begin{aligned}\|\beta_{ki}(.)\|_1 = \sum_{i=0}^{n} |\beta_{ki}(\tau)| = \sum_{i=1}^{n-1} \beta_{k,i}(\tau) &= \sum_{i=0}^{n-1}\sum_{j=0}^{i}(i-j)\alpha_{k,j}(\tau) = \sum_{j=0}^{n}\sum_{i=j}^{n-1}(i-j)\alpha_{k,j}(\tau) \\ &= \sum_{j=0}^{n}\Big(\sum_{i=0}^{n-1-j} i\Big)\alpha_{k,j}(\tau) = \sum_{j=0}^{n}\binom{n-j}{2}\alpha_{k,j}(\tau) \\ &= \sum_{j=0}^{n}\binom{j}{2}\alpha_{k,j}(\tau) = \sum_{j=2}^{n}\binom{j}{2}\alpha_{k,j}(\tau) \\ &= \sum_{j=2}^{n}\binom{j}{2}B_{j,n}(\tau) + \frac{k}{2}(k+1-2n\tau) \\ &= \binom{n}{2}\tau^2 + \frac{k}{2}(k+1-2n\tau)\end{aligned} \tag{3.110}$$

Therefore, $\sum_{i=0}^{n}\beta_{ki}$ is a positive quadratic polynomial over the interval $[\tau_k, \tau_{k+1}]$, and it attains a maximum at τ_k or τ_{k+1}, that is

$$\begin{aligned}\max_{\tau_k\le\tau\le\tau_{k+1}} \sum_{i=0}^{n}\beta_{k,i}(\tau) &= \max\Big\{\sum_{i=0}^{n}\beta_{ki}(\tau_k), \sum_{i=0}^{n}\beta_{ki}(\tau_{k+1})\Big\} \\ &= \binom{n}{2}\frac{k^2}{n^2} - \binom{k}{2} = \frac{k}{2n}(n-k)\end{aligned} \tag{3.111}$$

Hence,

$$\max_{0\le k\le n-1}\max_{\tau_k\le\tau\le\tau_{k+1}} \sum_{i=0}^{n}\beta_{k,i}(\tau) = \max_{0\le k\le n}\binom{n}{2}\frac{k^2}{n^2} - \binom{k}{2} = \frac{k}{2n}(n-k) = \frac{\lfloor\frac{n}{2}\rfloor\lceil\frac{n}{2}\rceil}{2n} \tag{3.112}$$

Accordingly, the bound on $\|\mathrm{r} - l\|_{\infty,[0,1]}$ can be expressed as [NPL98]

$$\|\mathrm{r} - l\|_{\infty,[0,1]} \le N_\infty(n)\|\Delta_2 r\|_\infty, \tag{3.113}$$

where

$$N_\infty(n) = \frac{\lfloor\frac{n}{2}\rfloor\lceil\frac{n}{2}\rceil}{2n}. \tag{3.114}$$

- Upper bound including $\|\Delta r\|_\infty$

A bound on the distance between $\mathrm{r}(\tau)$ and its control polygon can be similarly obtained using the first differences as

$$\begin{aligned}\|\mathrm{r} - l\|_{\infty,[0,1]} &= \max_k \|\sum_{i=0}^{n}\alpha_{k,i}r_i\|_k \\ &= \max_k \|\sum_{i=0}^{n}\Delta\gamma_{k,i}r_i\|_k = \max_k\|\sum_{i=0}^{n}-\gamma_{k,i}\Delta r_i\|_k \\ &\le \max_k\|\sum_{i=0}^{n}|\gamma_{k,i}|\|_k\|\Delta r\|_\infty\end{aligned} \tag{3.115}$$

where

$$\|\Delta r\|_\infty \coloneqq \max_{0\le i\le n} |\Delta r_i| \quad \text{and} \quad \Delta r_i \coloneqq r_{i+1} - r_i. \tag{3.116}$$

Considering that $\gamma_{ki} > 0$ for $i < k$, the sum $\sum_{i=0}^{k-1} |\gamma_{ki}|$ can be obtained as

$$\sum_{i=0}^{k-1} |\gamma_{ki}| = \sum_{i=0}^{k-1} \gamma_{ki} = \sum_{i=0}^{k-1}\sum_{j=0}^{i} B_{j,n} = \sum_{j=0}^{k-1}\sum_{i=j}^{k-1} B_{j,n} = \sum_{j=0}^{k-1}(k-j)B_{j,n} = \beta_{kk} \tag{3.117}$$

and for $i > k$, using the partition of unity property, we get

$$\begin{aligned}\sum_{i=k+1}^{n} |\gamma_{ki}| &= -\sum_{i=k+1}^{n} \gamma_{ki} = -\sum_{i=k+1}^{n}\Big(\sum_{j=0}^{i} B_{j,n} - 1\Big) = \sum_{i=k+1}^{n}\sum_{j=i+1}^{n} B_{j,n} \\ &= \sum_{j=k+2}^{n}\sum_{i=k+2}^{j} B_{j,n} = \sum_{j=k+2}^{n}(j-k-1)B_{j,n} = \sum_{j=k+1}^{n}(j-k-1)B_{j,n} = \beta_{k,k+1}.\end{aligned} \tag{3.118}$$

and finally, making use of (3.40), γ_{kk} can be written as

$$\begin{aligned}\gamma_{kk} &= \sum_{j=0}^{k} B_{j,n} - (k+1) + n\tau = \sum_{j=0}^{k} B_{j,n} - (k+1)\sum_{j=0}^{n} B_{j,n} + \sum_{j=0}^{n} jB_{j,n} \\ &= \sum_{j=0}^{k}(j-k)B_{j,n} + \sum_{j=k+1}^{n}(j-k-1)B_{j,n} = -\beta_{kk} + \beta_{k,k+1}\end{aligned} \tag{3.119}$$

Considering the above Bézier representation of γ_{kk} with monotonically increasing sequence of $n+1$ control points, $\{-k, \ldots, 0, 0, \ldots, n-k-1\}$, the sum $\sum_{i=0}^{n} |\gamma_{ki}|$ can be computed as

$$\sum_{i=0}^{n} |\gamma_{ki}| = \sum_{i=0}^{k-1} |\gamma_{ki}| + |\gamma_{kk}| + \sum_{i=k+1}^{n} |\gamma_{ki}| = \begin{cases} 2\beta_{kk}, & \tau \le \tau_0, \\ 2\beta_{k,k+1}, & \tau > \tau_0 \end{cases} \tag{3.120}$$

where $\tau_0 \in [\tau_k, \tau_{k+1}]$ is the corresponding parameter value to the unique zero of γ_{kk}. Since $\sum_{i=0}^{n} |\gamma_{ki}|$ is a non-negative convex function over $[\tau_k, \tau_{k+1}]$ (see Fig. 3.18), we get

$$\max_{\tau_k \le \tau \le \tau_{k+1}} \sum_{i=0}^{n} |\gamma_{k,i}(\tau)| = 2\max\{\beta_{kk}(\tau_k), \beta_{k,k+1}(\tau_{k+1})\} \tag{3.121}$$

Therefore,

$$\begin{aligned}\max_k \|\sum_{i=0}^{n} |\gamma_{ki}|\|_k &= 2\max_k \beta_{kk}(\tau_k) \\ &= 2\max_k \sum_{j=0}^{k}(k-j)B_{j,n}(\tau_k) \\ &= 2\max_k \frac{(n-k)k}{n} B_{k,n}(\frac{k}{n}) \\ &= 2\frac{\lfloor \frac{n}{2} \rfloor \lceil \frac{n}{2} \rceil}{n} B_{\lceil \frac{n}{2} \rceil, n}(\frac{\lceil \frac{n}{2} \rceil}{n}) = 4N_\infty(n) B_{\lceil \frac{n}{2} \rceil, n}(\frac{\lceil \frac{n}{2} \rceil}{n})\end{aligned} \tag{3.122}$$

That being the case, the bound on $\|\mathrm{r} - l\|_{\infty,[0,1]}$ including $\|\Delta r\|_\infty$ is given by [NPL98]

$$\|\mathrm{r} - l\|_{\infty,[0,1]} \leq L_\infty(n)\|\Delta r\|_\infty, \tag{3.123}$$

where

$$L_\infty(n) = 4N_\infty(n)B_{\lceil \frac{n}{2} \rceil, n}(\frac{\lceil \frac{n}{2} \rceil}{n}). \tag{3.124}$$

3.2.4.3 Bound alternatives

Bounds involving p-norm of $\Delta_2 r$ or Δr can be computed by making proper choices of p and q in the Hölder's inequality. To obtain N_∞ and L_∞, the inequality

$$\sum_{i=1}^{n-1} \beta_{ki}\Delta_2 r_i \leq \|\beta_{ki}\|_q \|\Delta_2 r\|_p, \quad q^{-1} + p^{-1} = 1, \tag{3.125}$$

was used with $p = \infty$ and $q = 1$. Among alternative estimates, the two cases, which $p = 1$ and $q = \infty$, and $p = q = 2$, warrant particular attention.

- $p = 1$ and $q = \infty$

The distance from the scalar-valued polynomial $\mathrm{r}(\tau)$ of degree n to its control polygon $l(\tau)$ is bounded by

$$\|\mathrm{r}(\tau) - l(\tau)\|_{\infty,[0,1]} \leq N_1(n)\|\Delta_2 r\|_1, \tag{3.126}$$

where

$$N_1(n) = \max_k \|\max_i \beta_{ki}\|_k = \beta_{\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil}(\tau^*) = 2N_\infty(n)B^n_{\lceil \frac{n}{2} \rceil}(\tau^*) \tag{3.127}$$

where $\tau^* := \lceil \frac{n}{2} \rceil / n$ [NPL98].

Similarly, we can derive

$$\|\mathrm{r}(\tau) - l(\tau)\|_{\infty,[0,1]} \leq L_1(n)\|\Delta r\|_1, \tag{3.128}$$

where

$$L_1(n) = \max_k \|\max_i |\gamma_{ki}|\|_k = \gamma_{k^*}(\frac{k^* + 1}{n}) \tag{3.129}$$

where $k^* := \lfloor \frac{n-2}{3} \rfloor$.

The bounds in (3.113), (3.126), and (**??**), containing the norm of the second difference vector, can be interpreted as measuring an estimate of the maximum, total, and average curvature, respectively. As we will see below, in the first case sharpness is obtained when the curvature is distributed most evenly, while sharpness in the second is attained when curvature is distributed most unevenly.

3.2.4.4 Sharpness of the bounds

The above estimated bounds are sharp for all degrees [NPL99], [KKK04]. For quadratic curves, i.e. $n = 2$, (3.108) becomes strict equality, i.e.,

$$\|\mathrm{r}(\tau) - l(\tau)\|_{\infty,[0,1]} = \|\Delta_2 r_1\|_\infty N_\infty(2) \tag{3.130}$$

where

$$N_\infty(2) = \max_{0 \leq k \leq 1} \max_{\tau \in [\tau_k . \tau_{k+1}]} \beta_{k1}(\tau) \tag{3.131}$$

Therefore, the bound (3.113) is sharp for degree 2. For the m-fold degree elevated representation of a quadratic curve, $\mathrm{r}^{(m)}(\tau)$, the second differences of the control points are all equal, i.e.

$$\Delta_2 r_i^{(m)} = \Delta_2 r_1^{(m)}, \quad i = 1, \ldots, m+1. \tag{3.132}$$

and we get equality in (3.108)

$$\|\mathrm{r}^{(m)}(\tau) - l(\tau)\|_{\infty,[0,1]} = |\Delta_2 r_1^{(m)}| \max_{0\le k\le m+1} \max_{\tau\in[\tau_k,\tau_{k+1}]} \sum_{i=1}^{m+1} \beta_{ki}(\tau) \tag{3.133}$$

and thus

$$N_\infty(2+m) = \max_{0\le k\le m+1} \max_{\tau\in[\tau_k,\tau_{k+1}]} \sum_{i=1}^{m+1} \beta_{ki}(\tau), \quad \|\Delta_2 r^{(m)}\|_\infty = |\Delta_2 r_1^{(m)}| \tag{3.134}$$

Therefore, the bound (3.113) is sharp for all degrees, if r has no inflection point, i.e. the sequence of second differences has no sign changes. (cf. Fig. 3.19.)

The above result can be extended to (3.126). Similarly, for quadratic curves (3.126) is sharp and

$$N_1(2) = \max_{0\le k\le n-1} \max_{\tau\in[\tau_k.\tau_{k+1}]} \max_{1\le i\le n-1} \beta_{ki}(\tau) = \max_{0\le k\le 1} \max_{\tau\in[\tau_k.\tau_{k+1}]} \beta_{k1}(\tau) = N_\infty(2) \tag{3.135}$$

Also, for a Bézier curve whose control polygon has the shape of an angle, i.e.,

$$\Delta_2 r_{\lceil \frac{n}{2} \rceil} \neq 0 \qquad \Delta_2 r_i = 0 \quad \text{for} \quad i \neq \lceil \frac{n}{2} \rceil, \tag{3.136}$$

the inequality (3.126) turns into equality

$$\mathrm{r}(\tau^*) - l(\tau^*) = \Delta_2 r_{\lceil \frac{n}{2} \rceil} \beta_{\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil}(\tau^*) = \|\Delta_2 r\|_1 N_1(n). \tag{3.137}$$

Therefore, (3.126) is sharp for all degrees if the sequence of second differences has one non-zero entry at $i = \lceil \frac{n}{2} \rceil$, where $\sum_i \beta_{ki}$ has a maximum.

3.2.4.5 Bound improvement at the end points

Considering that $\mathrm{r}(0) = r_0$ and $\mathrm{r}(1) = r_n$, and that the Bézier curve is tangent to its control polygon over the intervals $[0, \frac{1}{n}]$ and $[\frac{n-1}{n}, 1]$, the above bounds can be improved at the end-points [NPL98]. The first-order Taylor expansion of r at $\tau = 0$, $\mathrm{r}(0) + \mathrm{r}'(0)\tau$, agrees with the first leg of the control polygon, $(1 - n\tau)r_0 + n\tau r_1$. Hence, for $\tau \in [0, 1/n]$ and $\xi(\tau) \in (0, 1)$, the bound (3.113) can be refined as

$$\begin{aligned} \left|\mathrm{r}(\tau) - \Big((1-n\tau)r_0 + n\tau r_1\Big)\right| &= \left|\frac{r''(\xi(\tau))}{2}\tau^2\right| \\ &\le \frac{\tau^2}{2} n(n-1)\|\Delta_2 r\|_\infty \\ &\le \frac{n-1}{2}\|\Delta_2 r\|_\infty \tau. \end{aligned} \tag{3.138}$$

The bound improvement for $\tau \in [\frac{n-1}{n}, 1]$ can be obtained by replacing τ with $1 - \tau$ in the right-hand side of (3.138). Similarly, the bound (3.126) can be refined over $[0, 1/n]$ as

$$\left|\mathrm{r}(\tau) - \Big((1-n\tau)r_0 + n\tau r_1\Big)\right| \le \frac{n-1}{2}\|\Delta_2 r\|_1 \tau. \tag{3.139}$$

3.2.4.6 Polygonal Envelopes

The above estimated bounds can be used to construct polygonal envelopes of the scalar-valued polynomial r. The envelope for a constant c, provided by the right-hand side of one of the above inequalities that bounds the distance between the Bézier curve and its control polygon, is given by

$$\mathrm{E}(\mathrm{r}(\tau); p, c) = l(\tau) \oplus \{q \in \mathbb{R} | \quad |q| \leq c\} \tag{3.140}$$

which is the Minkowski sum of the control polygon and a closed set. Fig. (3.19) compares the polygonal envelopes constructed with different bounds. Fig. (3.19) shows sharpness of the bound implied by N_∞ bound for the first and second (From top) Bézier curves. The third example and illustrates the effect of sign changes in the sequence of second differences, $\Delta_2 r$, on the sharpness of the bound, and the fourth demonstrate a Bézier curve with hat-shaped control polygon for which (3.126) is sharp.

It should be noted that all bounding regions can be improved at the end points. For N_∞, in particular, using (3.138), the envelope on $[0, 1/n]$ is confined to the region enclosed by the triangle with vertices

$$r_0, \quad \text{and} \quad r_1 \pm \frac{n-1}{2n} \|\Delta_2 r\|_\infty. \tag{3.141}$$

The envelope can be rendered tighter with combining bounds, e.g. taking the intersection with the control polygon [NPL98]. Also, improvements can be made by using bounds that are more local to each segment. In (3.113), for example, the bound can be refined by replacing the maximum value of $\sum_i \beta_{ki}$ over $[0, 1]$ with its maximum over a particular interval.

Table 3.1: The smallest possible constant that bounds the distance between a Bézier curve and its control polygon, for 1, 2, and ∞-norm, and $n = 2, \ldots, 8$.

n	2	3	4	5	6	7	8
$N_\infty(n)$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{3}{5}$	$\frac{3}{4}$	$\frac{6}{7}$	1
$N_1(n)$	0.2500	0.2963	0.3750	0.4147	0.4688	0.5036	0.5469
$N_2(n)$	0.2500	0.2986	0.3853	0.4331	0.5015	0.5480	0.6079

3.2.4.7 Convergence under subdivision

Here, we use the above computed bounds to evaluate the rate of convergence of the sequence of Bézier control polygons, generated with repeated subdivision or degree elevation, to a curve.

As we saw previously, adaptive refinement of the control point sequence of $\mathrm{r}(\tau)$ can be achieved by generating control polygons over the sub-intervals of the domain with repeated subdivision. The curve segment restricted to the sub-interval $[0, \tau_0], \tau_0 \in (0, 1)$, can be expressed as

$$\mathrm{r}_{[0,\tau_0]}(\tau) \coloneqq \sum_{i=0}^{n} r_{i,[0,\tau_0]} B_{i,n}(t), \tag{3.142}$$

where $r_{i,[0,\tau_0]}$ are the coefficients computed with the de Casteljau's recursive formula (3.65), and can be expressed as

$$r_{i,[0,\tau_0]} = r_0^{(i)} = \sum_{k=0}^{i} B_{k,i}(\tau_0) r_k^{(0)} = \sum_{k=0}^{i} B_{k,i}(\tau_0) r_k \quad i = 0, \ldots, n-2. \tag{3.143}$$

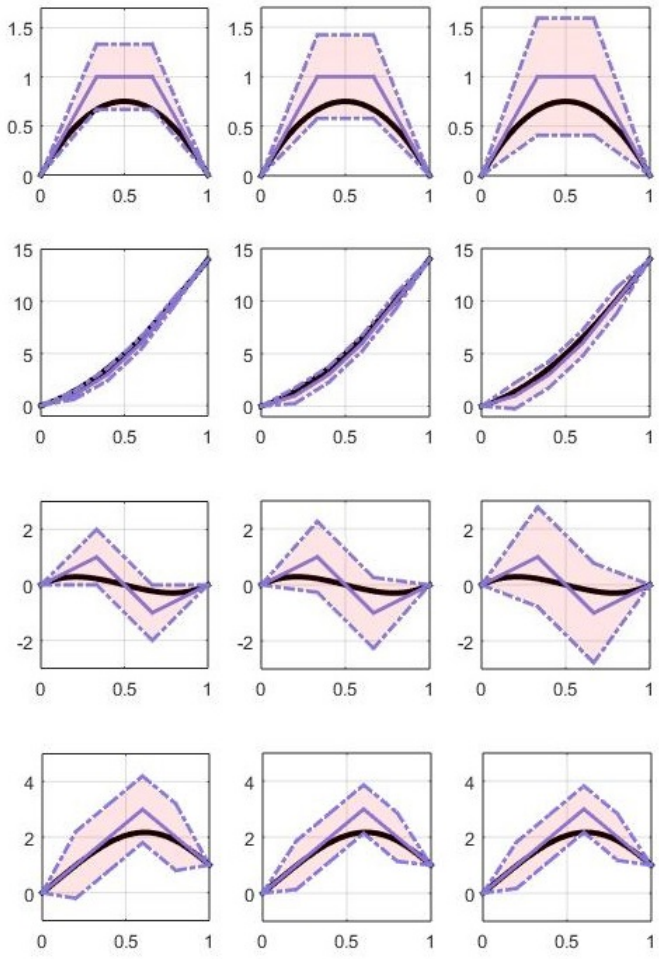


Figure 3.19: One dimensional Bézier curve (solid black line), its control polyline (solid blue line), and the envelope (dashed line) constructed with the bound implied by (from left to right) (1) N_∞, (2) N_2, (3) N_1. The control point sequence for the Bézier curve is (from top to bottom) (1) $[0, 1, 1, 0]$, (2) $[0, 1, 3, 6, 10, 14]$, (3) $[0, 1, -1, 0]$, (4) $[0, 1, 2, 3, 2, 1]$.

and thus, the second differences of the control points is obtained as

$$\begin{aligned}\Delta_2 r_{i+1,[0,\tau_0]} &= r_{i,[0,\tau_0]} - 2r_{i+1,[0,\tau_0]} + r_{i+2,[0,\tau_0]} \\ &= \tau_0^2(r_0^{(i)} - 2r_1^{(i)} + r_2^{(i)}) \\ &= \tau_0^2(\Delta_2 r_1^{(i)}) = \tau_0^2\Big(\sum_{j=0}^{i} B_{j,i}(\tau_0)\Delta_2 r_{j+1}\Big).\end{aligned} \tag{3.144}$$

Therefore, the distance between $\mathrm{r}_{[0,\tau_0]}(\tau)$, and its control polygon $l_{[0,\tau_0]}(\tau)$, is bounded by

$$\|\mathrm{r}_{[0,\tau_0]}(\tau) - l_{[0,\tau_0]}(\tau)\|_{\infty,[0,\tau_0]} \leq \tau_0^2 N_\infty(n)\|\Delta_2 r\|_\infty \tag{3.145}$$

where $\|\Delta_2 r\|_\infty$ is the maximum absolute second difference of the original control points. By symmetry, the bound for the segment over the interval $[\tau_0, 1]$ is

$$\|\mathrm{r}_{[\tau_0,1]}(\tau) - l_{[\tau_0,1]}(\tau)\|_{\infty,[\tau_0,1]} \leq (1-\tau_0)^2 N_\infty(n)\|\Delta_2 r\|_\infty \tag{3.146}$$

Therefore, the bounds on the two segments generated with the subdivision of r are scaled versions of the original bound. It can be concluded, from (3.145) and (3.146), that the distance between the polynomial r and the control polygon generated with subdivision at τ_0 is bounded by

$$\|\mathrm{r}(\tau) - l_{[0,\tau_0]\cup[\tau_0,1]}(\tau)\|_{\infty,[0,1]} \leq \bar{\tau}^2 N_\infty(n)\|\Delta_2 r\|_\infty \tag{3.147}$$

where $l_{[0,\tau_0]\cup[\tau_0,1]}$ is the union of the control polygons of $\mathrm{r}_{[0,\tau_0]}$ and $\mathrm{r}_{[\tau_0,1]}$, and $\bar{\tau} \coloneqq \max\{\tau_0, 1-\tau_0\}$ [NPL98]. The upper bound can be refined by repeated subdivision. The distance between r to $l^{(m)}$, the control polygon after m-fold subdivision at the local parameter τ_0, is bounded by

$$\|\mathrm{r}(\tau) - l^{(m)}(\tau)\|_{\infty,[0,1]} \leq \bar{\tau}^{2m} N_\infty(n)\|\Delta_2 r\|_\infty \tag{3.148}$$

The above bound establishes the quadratic convergence rate of the refined control polygon to the curve segment under subdivision. It follows directly from (3.148) that the distance of $\mathrm{r}(\tau)$ to the composite control polygon generated with repeated subdivision at

$$[0, \frac{1}{2^m}, \frac{2}{2^m}, \ldots, 1] \tag{3.149}$$

is reduced by $\frac{1}{4}$, and is, asymptotically, as good an approximation as the polyline connecting the grid points

$$\big(\tau_i, \mathrm{r}(\tau_i)\big) \qquad i = 0, \ldots, n2^m \tag{3.150}$$

where $\tau_i = \frac{i}{n2^m}$ [PBP02].

Considering (3.144), the optimal subdivision parameter, $\tau_0 \in (0,1)$, for minimizing the distance between the Bézier curve $\mathrm{r}(\tau)$ and the generated control polygon after subdivision, can be obtained by solving the following problem

$$\min_{\tau_0\in(0,1)} \max_{i=1,\ldots,n-1} \{|\Delta_2 r_{i,[0,\tau_0]}|, |\Delta_2 r_{i,[\tau_0,1]})|\} \tag{3.151}$$

$$= \min_{\tau_0\in(0,1)} \max_{i=1,\ldots,n-1} \Big\{\tau_0^2\Big|\sum_{j=0}^{i} B_{j,i}(\tau_0)\Delta_2 r_{k+1}\Big|, (1-\tau_0)^2\Big|\sum_{j=0}^{i} B_{j,i}(1-\tau_0)\Delta_2 r_{n-1-j}\Big|\Big\} \tag{3.152}$$

- For $n = 2$, after scaling by $\Delta_2 r_0$, the problem becomes

$$\min_{0<\tau_0<1} \max\{\tau_0^2, (1-\tau_0)^2\}, \tag{3.153}$$

  and $\tau_0 = \frac{1}{2}$ is optimal.

- For $n = 3$, assuming that $\Delta_2 r_i \neq 0$, and $\Delta_2 r_1 = 1 + \delta$ and $\Delta_2 r_2 = 1$, the second difference can be normalized by $\Delta_2 r_2$, and the problem becomes

$$\min_{0<\tau_0<1} \max\{\tau_0^2|1+\delta|, (1-\tau_0)^2, \tau_0^2|1+\delta-\delta\tau_0|, (1-\tau_0)^2|1+\delta(1-\tau_0)|\}, \tag{3.154}$$

  and $\tau_0 = 0.43$, is the solution.

It should be noted that for $n = 2$ and $n = 3$, the optimal subdivision parameter τ_0, minimizing the distance, does not coincide with the point of maximum curvature, which is usually used for adaptive subdivision of Bézier curves.

3.2.4.8 Convergence under degree elevation

With degree elevation, the number of coefficients increases by one and since the new coefficients are obtained as convex combinations of the original coefficients, it is possible to show convergence of the sequence of the control polygons corresponding to repeated degree elevation to the graph of the polynomial on $[0, 1]$.

For unit degree elevation, we have

$$\sum_{i=0}^{n} r_i B_{i,n}(\tau) = \sum_{i=0}^{n+1} r_i^{n+1} B_{i,n+1}(\tau). \tag{3.155}$$

Differentiating twice yields,

$$n(n-1)\sum_{i=0}^{n-2} \Delta_2 r_{i+1}^n B_i^{n-2} = (n+1)n \sum_{i=0}^{n-1} \Delta_2 r_{i+1}^{n+1} B_i^{n-1} \tag{3.156}$$

Considering (3.69), the maximum $\|\Delta_2 r^{n+1}\|_\infty$ is obtained when all second differences $\Delta_2 r_i$ are equal, implying

$$\|\Delta_2 r^{n+1}\|_\infty \leq \frac{n-1}{n+1} \|\Delta_2 r\|_\infty. \tag{3.157}$$

The distance between the control polygon l^{n+1} of the degree elevated representation and the Bézier curve r is therefore

$$\|\mathrm{r} - l^{(n+1)}\|_{\infty,[0,1]} \leq K(n, n+1) N_\infty(n) \|\Delta_2 r\|_\infty, \tag{3.158}$$

where

$$K(n, n+1) \coloneqq \frac{n-1}{n+1} \frac{N_\infty(n+1)}{N_\infty(n)} \tag{3.159}$$

Similarly, for elevation to the degree $2^k n$

$$K(n, 2^k n) = \frac{n(n-1)}{2^k n(2^k n+1)} \frac{N_\infty(2^k n)}{N_\infty(n)} = \frac{1}{2} \left\{ \begin{array}{ll} \frac{n-1}{n+1/2} & \text{if n is even} \\ \frac{n^2}{(n+1/2)(n+1)} & \text{if n is odd} \end{array} \right\} \leq \frac{1}{2}. \tag{3.160}$$

Therefore, the distance between the Bézier curve of degree n and the control polygon of the curve elevated to the degree $2^k n$, is bounded by

$$\|\mathrm{r} - l^{(2^k n)}\|_{\infty,[0,1]} \leq \frac{1}{2} N_\infty(n) \|\Delta_2 r\|_\infty, \tag{3.161}$$

where $\|\Delta_2 r\|_\infty$ is the maximum absolute second difference of the original control point sequence [NPL98].

The above inequality implies that the distance between a Bézier curve and the generated polygon with degree elevation is (asymptotically) reduced by $\frac{1}{2}$. Therefore, while an approximant of a Bézier curve can be obtained from the sequence of control polygons generated with either subdivision and degree elevation, the former converges faster.

3.3 Case study

In this section we put the performance and the computational efficiency of the proposed method into test through different simulation examples. First, we consider the go-to-formation maneuver, described in the previous chapter, and write the constraints (2.32) as functions of the flat output. Next, we study the more challenging problem of generating collision-free trajectories for drones flying in confined spaces. We derive a set of collision avoidance constraints that explicitly takes into account the drone's orientation. We show that the obtained set of constraints can be expressed as Bézier curves, and thus the method proposed in (3.2.3.2) can be applied to solve the problem.

3.3.1 Go-to-Formation maneuver

Here, we use the Bézier curve-based method proposed in this chapter to solve the multiple vehicle motion planning problem studied in Sec. 2.4.1. Recalling from Chapter 2, the vehicle's model is described with

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ a \\ \omega \\ r \end{bmatrix}. \tag{3.162}$$

With the choice of $x(t)$ and $y(t)$ as flat outputs, the trajectory for each vehicle is parameterized with a n-degree planar Bézier curve given by

$$\mathbf{p}_i(\tau) = \begin{bmatrix} x_i(\tau) \\ y_i(\tau) \end{bmatrix} = \sum_{k=0}^{n} \mathbf{p}_{i,k} B_{k,n}(\tau) \quad i = 1, \ldots, N_v \tag{3.163}$$

where $\mathbf{p}_{i,k} \in \mathbb{R}^2$ are the control points that need to be determined by solving an optimization problem. The independent variable $\tau \in [0, 1]$ is defined as

$$\tau = \zeta(t) = \frac{t}{t_f} \tag{3.164}$$

A subset of the control points can be readily determined from the initial and final conditions on the states and inputs of the vehicle's model. Particularly, considering (3.61), the first and last control points are obtained as

$$\mathbf{p}_{i,0} = \begin{bmatrix} x_i(0) \\ y_i(0) \end{bmatrix} = \begin{bmatrix} x_{i,0} \\ y_{i,0} \end{bmatrix} \qquad \mathbf{p}_{i,n} = \begin{bmatrix} x_i(1) \\ y_i(1) \end{bmatrix} = \begin{bmatrix} x_{i,f} \\ y_{i,f} \end{bmatrix} \tag{3.165}$$

where $(x_{i,0}, y_{i,0})$ and $(x_{i,f}, y_{i,f})$ denote respectively the initial and final positions of the i-th vehicle. The rest of the control points will be determined such that they satisfy the constraints,

$$\begin{aligned} &v_{\min} \leq v(t) \leq v_{\max} \\ &a_{\min} \leq a(t) \leq a_{\max} \\ &\sin(\psi_{\min}) \leq \sin(\psi(t)) \leq \sin(\psi_{\max}) \\ &\omega_{\min} \leq \omega(t) \leq \omega_{\max}, \end{aligned} \tag{3.166}$$

for $\forall t \in [0, t_f]$, and minimize the cost function

$$J = \int_0^1 \left\| \mathrm{Q}^{\frac{1}{2}} \mathbf{u}(\tau) \right\|^2 d\tau + \rho t_f, \tag{3.167}$$

where $\mathbf{u} = [a, r]^T$, and Q is the weighting matrix. The above cost function and constraints can be re-written in terms of the flat outputs, $x_i(\tau)$ and $y_i(\tau)$, and their derivatives. Considering the vehicle's model, the speed and acceleration are obtained as

$$\begin{aligned} v(t) &= \frac{1}{t_f} \sqrt{x'^2 + y'^2} \\ a(t) &= \frac{1}{t_f^2} \frac{x'x'' + y'y''}{\sqrt{x'^2 + y'^2}} \end{aligned} \tag{3.168}$$

where $x' = \frac{d}{d\tau}\big(x(\zeta(t))\big)$ and $x'' = \frac{d^2}{d\tau^2}\big(x(\zeta(t))\big)$. We drop the subscript i for simplicity. The course angle and the course rate can also be re-written as

$$\begin{aligned} \psi(\tau) &= \arcsin \frac{y'}{x'^2 + y'^2} \\ \omega(t) &= \frac{1}{t_f} \frac{x'y'' - y'x''}{x'^2 + y'^2} \end{aligned} \tag{3.169}$$

Also, the collision avoidance constraint between the i-th and the j-th vehicles, given by

$$\left\| \mathbf{p}_i(\tau) - \mathbf{p}_j(\tau) \right\|^2 \geq \mathrm{R}^2 \qquad \forall \tau \in [0,1] \quad i, j \in \{1, \ldots, N_v\} \tag{3.170}$$

is immediately expressed in terms of the flat outputs. The inequality (3.170) guarantees temporal separation of the two trajectories. In the following, we will show how the spatial separation between trajectories can be handled.

Spatial collision-avoidance constraints as Bézier Surface

The spatial collision-avoidance constraint between the i-th and the j-th vehicles, whose trajectories, $\mathbf{p}_i$ and $\mathbf{p}_j$, are parameterized with Bézier curves of degree n and m, is expressed as

$$\mathrm{R}^2 \leq \left\| \mathbf{p}_i(u) - \mathbf{p}_j(v) \right\|^2 \qquad \forall u \in [0,1] \text{ and } \forall v \in [0,1]. \tag{3.171}$$

The right-hand side of the above inequality can be defined as a bivariate function,

$$S(u,v) = \Big(\sum_{k=0}^{n} \mathbf{p}_{i,k} B_{k,n}(u) - \sum_{l=0}^{m} \mathbf{p}_{j,l} B_{l,m}(v) \Big)^2, \tag{3.172}$$

From the Bernstein polynomial properties, we have

$$\begin{aligned} \| \sum_{k=0}^{n} \mathbf{p}_{i,k} B_{k,n}(u) \|^2 &= \sum_{i=0}^{2n} r_i B_{i,2n}(u) = \sum_{i=0}^{2n} r_i B_{i,2n}(u) \sum_{j=0}^{2m} B_{j,2m}(v) \\ \| \sum_{l=0}^{m} \mathbf{p}_{j,l} B_{l,m}(v) \|^2 &= \sum_{j=0}^{2m} s_j B_{j,2m}(v) = \sum_{j=0}^{2m} s_j B_{j,2m}(v) \sum_{i=0}^{2n} B_{i,2n}(u) \end{aligned} \tag{3.173}$$

where

$$\begin{aligned} r_i &= \sum_{k=\max\{0,i-n\}}^{\min\{i,n\}} (\mathbf{p}_{i,k}{}^T \mathbf{p}_{i,i-k}) \frac{\binom{n}{k}\binom{n}{i-k}}{\binom{2n}{i}} \\ s_j &= \sum_{l=\max\{0,j-m\}}^{\min\{j,m\}} (\mathbf{p}_{j,l}{}^T \mathbf{p}_{j,j-l}) \frac{\binom{m}{l}\binom{m}{j-l}}{\binom{2m}{j}} \end{aligned} \tag{3.174}$$

Also, the inner product of $\mathbf{p}_i(u)$ and $\mathbf{p}_j(v)$ can be obtained as

$$\begin{aligned} &\sum_{k=0}^{n} \mathbf{p}_{i,k} B_{k,n}(u) . \sum_{l=0}^{m} \mathbf{p}_{j,l} B_{l,m}(v) \\ &= \Bigg(\sum_{i=0}^{2n} \Big(\sum_{k=\max\{0,i-n\}}^{\min\{i,n\}} \mathbf{p}_{i,k} \frac{\binom{n}{k}\binom{n}{i-k}}{\binom{2n}{i}} \Big) B_{i,2n} \Bigg)^T \Bigg(\sum_{j=0}^{2m} \Big(\sum_{l=\max\{0,j-m\}}^{\min\{j,m\}} \mathbf{p}_{j,l} \frac{\binom{m}{l}\binom{m}{j-l}}{\binom{2m}{j}} \Big) B_{j,2m} \Bigg) \\ &= \sum_{i=0}^{2n} \sum_{j=0}^{2m} q_{i,j} B_{i,2n}(u) B_{j,2m}(v), \end{aligned} \tag{3.175}$$

where

$$q_{i,j} = \Big(\sum_{k=\max\{0,i-n\}}^{\min\{i,n\}} \mathbf{p}_{i,k} \frac{\binom{n}{k}\binom{n}{i-k}}{\binom{2n}{i}} \Big)^T \Big(\sum_{l=\max\{0,j-m\}}^{\min\{j,m\}} \mathbf{p}_{j,l} \frac{\binom{m}{l}\binom{m}{j-l}}{\binom{2m}{j}} \Big). \tag{3.176}$$

Therefore, $S(u,v)$ can be written as a Bézier surface

$$S(u,v) = \sum_{i=0}^{2n} \sum_{j=0}^{2m} c_{i,j} B_{j,2m}(v) B_{i,2n}(u) \tag{3.177}$$

whose control points are

$$c_{i,j} = r_i + s_j - 2q_{i,j}. \tag{3.178}$$

The proposed Bernstein relaxation and refinement for evaluating inequality constraints on Bézier curves can be extended to Bézier surfaces. Therefore, the spatial collision-avoidance constraint (3.171) can be replaced by a set of constraints on the control points $c_{i,j}$.

Bézier surfaces: de Casteljau's Algorithm

The de Casteljau's algorithm can be extended to handle Bézier surfaces. The Bézier surface $S(u, v)$ defined as

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} c_{i,j} B_{j,m}(v) B_{i,n}(u) \tag{3.179}$$

can be re-written as

$$S(u,v) = \sum_{i=0}^{n} d_i(v) B_{i,n}(u) \tag{3.180}$$

where

$$d_i(v) = \sum_{j=0}^{m} c_{i,j} B_{j,m}(v) \tag{3.181}$$

For a given parameter value $\bar{v}$, $d_i(\bar{v}), i = 0, \ldots, n$, is a point on the Bézier curve defined by the set of $m+1$ control points $c_{i,0}, \ldots, c_{i,m}$. Also, from (3.180), $S(\bar{u}, \bar{v})$ can be computed as a point on a Bézier curve with $n+1$ control points $d_0(\bar{v}), \ldots, d_n(\bar{v})$. Therefore, for the given parameter value $(\bar{u}, \bar{v})$, the corresponding point on a Bézier surface can be found by applying the de Casteljau's algorithm several times. This procedure is summarized in Algorithm 2 for a Bézier surface defined by a $n \times m$ control net.

Algorithm 2 Computing $S(\bar{u}, \bar{v})$ using the de Casteljau's algorithm

Input: $\bar{u}$ and $\bar{v}$
Output: $S(\bar{u}, \bar{v})$

1: **for** $i = 1 : n$ **do**
2: $\quad d_i(\bar{v}) \leftarrow$ Apply the de Casteljau's algorithm to the i-th row of the control net with $\bar{v}$;
3: **end for**
4: $S(\bar{u}, \bar{v}) \leftarrow$ Apply the de Casteljau's algorithm to $d_i(\bar{v}), \quad i = 0, \ldots, n$ with $\bar{u}$;

Having obtained the set of constraints in terms of the flat output, $x_i(\tau)$ and $y_i(\tau)$, the trajectory generation problem is converted to an optimization problem with the control points $\mathbf{p}_{i,k}, i = 1, \ldots, N_v, k = 0, \ldots, n$ as decision variables. Once the optimization problem is solved, the trajectories as well as the input profiles can be deduced from the obtained control points. In the examples presented below, the optimization problems are all solved with the FORCES Pro NLP solver.

Recalling that the sum and the product of two Bernstein polynomials are also a Bernstein polynomial, the constraints (3.166), (3.170), and (3.171) can be immediately expressed in Bézier form, and thus they can be replaced by a set of constraints on their control points. Below, we compare the computational efficiency of the proposed Bernstein relaxation and refinement method for evaluating inequality constraints with the method explained in Algorithm 1 and the time gridding approach.

3.3.1.1 Simulation Results

The first example consists of 5 vehicles in a go-to-formation maneuver. Given the vehicles' initial positions, we require a set of 5 collision-free trajectories, parameterized as planar Bézier curves, such that they guide the vehicles to the desired positions. The initial and final positions are shown in Fig. 3.20a with • and ×, respectively. The trajectories should guarantee that all vehicles reach their final positions simultaneously with desired speed and orientation. In this example, the final course angle and speed of all vehicles are set to 0° and $0.7\frac{m}{s}$, respectively. The

generated trajectories should also satisfy the bounds on the states and inputs, given in Table 3.2, while minimizing the cost function with $\mathrm{Q} = \mathrm{diag}(1, 0.5)$ and $\rho = 10$. A minimum distance of 5 meters should be kept among the vehicles during the entire travel time.

Fig. 3.20a shows the trajectories generated using Bézier curves of degree 6. The obtained travel time to final positions is 83.87 seconds, i.e. $t_f^* = 83.87$s. The speed, acceleration, course angle, and course rate of the vehicles are displayed in Fig. 3.20b. The results confirm that the states and inputs are within the required bounds. The average solution time, obtained on a desktop computer with 2.60 GHz i7-4510U CPU and 6.00 GB RAM, is 365 milliseconds. The same problem is also solved using the direct multiple shooting method described in the previous chapter with 180 shooting intervals. While the generated trajectories with the two methods are almost identical, the Bézier curve-based method ensures constraint satisfaction for the entire travel time, and results in much shorter computation times, which was to be expected due to the reduced number of decision variables and constraints in the underlying optimization problem. We've also employed different approaches for evaluating inequality constraints and compared the recorded computation times in Table 3.3. It should be noted that while Algorithm 1 yields non-smooth optimization problems, and non-smooth optimization methods must be, technically, used to handle them, the results in Table 3.3 are obtained with standard optimization methods, and finite difference for approximating Jacobian of the constraint functions. Using non-smooth approaches such as bundle methods would result in longer computation times. A similar problem is solved for 5 drones using spatial Bézier curves and the obtained trajectories are shown in Fig. 3.33.

In the next example, we consider two vehicles moving in an environment with static and moving obstacles. The goal is to generate smooth trajectories that guide the vehicles to their desired final positions, shown with $\times$ in Fig. 3.22, while avoiding the obstacles. Also, a minimum distance of 2 m must be kept between the two vehicles during the entire travel time. In order to address the uncertainties in the environment, the trajectories are re-planned as new obstacles are detected. At each re-planning step, an optimization problem is solved to generate a Bézier curve, from the position of the vehicles at the current time instant to the final positions, taking into account the most recent measurements of the obstacles' positions. Therefore, it is necessary to consider the continuity conditions between consecutive Bézier segments at the joining points to ensure that the generated trajectories are smooth.

Fig. 3.22 shows the trajectories generated at different time instances using planar Bézier curve of degree 6. At $t = 0$, the trajectory is generated from the drone's initial position to its final position such that collisions with the 3 known obstacles are avoided. At $t = 5.6s$ and $t = 6.8s$, the trajectories are re-planned to avoid collision with the new obstacles detected along the trajectory. The average computation time for solving the optimization problems, using the proposed Bernstein relaxation and refinement method, is 146 ms. A comparison of the computation times obtained with different approaches of evaluating the inequality constraints is given in Table 3.3. In all simulation results presented in this chapter, a non-conservative set of constraints is made with repeated subdivision such that the generated trajectories match those obtained with the multiple shooting method over fine discretization grids. As the recorded computation times presented verify, the proposed Bézier curve-base method, with smaller number of decision variables and constraints, can generate collision-free trajectories in much shorter time without compromising on the performance of the trajectories.

We also compare the computation times of solving the autonomous cinematography problem (2.50) using different approaches discussed in this chapter. Expressing the mutual visibility constraint (2.52) as well as the constraints (2.50.c-2.50.g) as Bernstein polynomials is straightforward. The bound constraints on the gimbal yaw angle with respect to the quadrotor, $|^{Q}\psi_C| \leq \pi/2$, can be expressed as a Bernstein polynomial considering that

$$\cos(^{Q}\psi_C) = \frac{\mathbf{v}_{XY}{}^{T}\mathbf{q}_{XY}}{\|\mathbf{v}_{XY}\|\|\mathbf{q}_{XY}\|} \tag{3.182}$$

Table 3.2: Upper and lower bounds on the states and inputs of the vehicle's model

	minimum value	maximum value
$\nu(\frac{m}{s})$	0.1	1.5
$\psi(rad)$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$
$a(\frac{m}{s^2})$	-0.1	0.1
$\omega(\frac{rad}{s})$	-0.349	0.349

where $\mathbf{v}_{XY} = [v_x, v_y]^T$ and $\mathbf{q}_{XY} = [q_x, q_y]^T$. The trajectories in Fig. 2.26 and Fig. 2.23 are recreated using the proposed Bézier curve-based method in this chapter and the computation times are presented in Table 3.3.

Table 3.3: Recorded computation times for generating trajectories using different approaches to evaluating inequality constraints.

			Computation Time (ms)		
		Number of Vehicles	**Bézier curve-based method with Bernstein relaxation and refinement**	**Bézier curve-based method with the GJK-based approach in Alg. 1**	**Direct multiple shooting method**
go-to-formation	**maneuver**	5 (Fig. 3.20)	365	546	1017
		5 (Fig. 3.33)	647	934	2184
		2 (Fig. 3.22)	146	421	818
autonomous	**cinematography**	1 (Fig. 2.26)	63	96	312
		2 (Fig. 2.23)	117	342	651

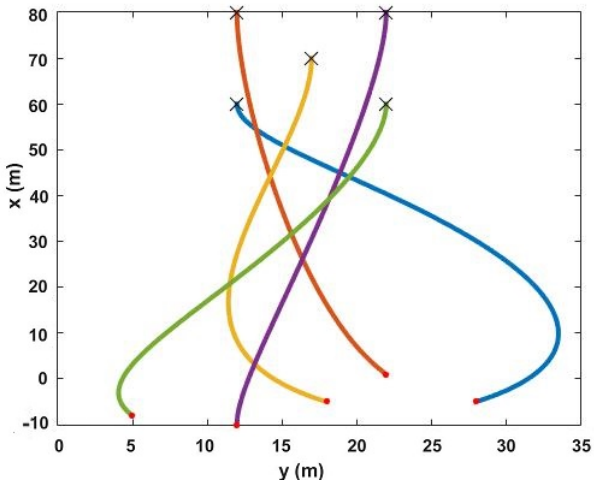
3.3.2 Collision-avoidance constraints for an ellipsoid model of the drone body

3.3.2.1 Quadrotor model

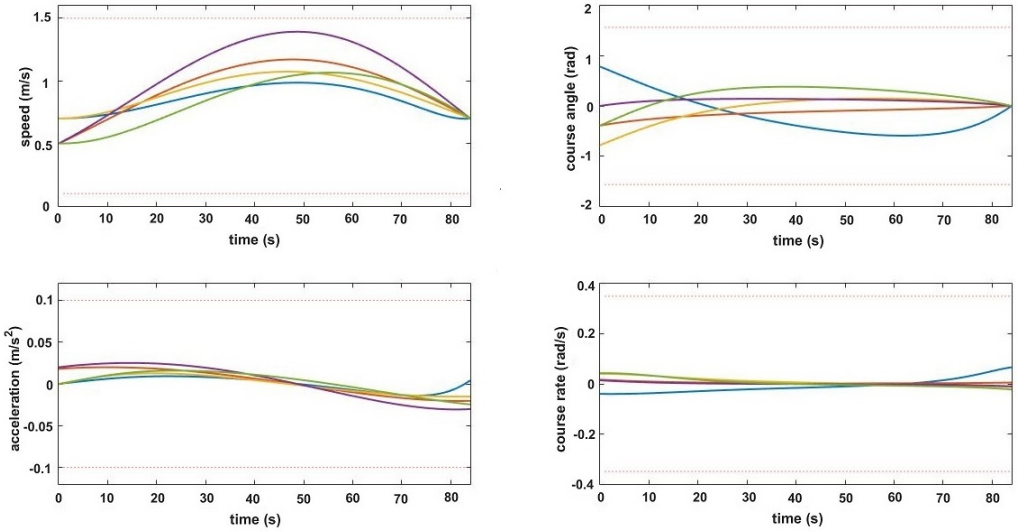
Here, the simplified quadrotor equations of motion are described by

$$m\ddot{\mathbf{p}} = mg\mathbf{e}_3 + f, \tag{3.183}$$

where $\mathbf{p} \in \mathbb{R}^3$ is the position and m is the mass of the quadrotor. In addition, $g = 9.8\,\frac{\mathrm{m}}{\mathrm{s}^2}$ is the gravitational acceleration, and $\mathbf{e}_3 = [0\ 0\ 1]^T$. The first term on the right-hand side of (3.183) is gravity in the $\mathbf{z}_I$ direction, and the second term, $f \in \mathbb{R}^3$, is the thrust force aligned with the body's $\mathbf{z}$-axis.



(a) Temporally de-conflicted trajectories generated for 5 vehicles using Bézier curves of degree 6. The trajectories are obtained in 365 milliseconds.



(b) The speed, acceleration, course angle and course rate of the 5 vehicles are within the required bounds.

Figure 3.20: The go-to-formation maneuver for 5 vehicles. Trajectories are generated with the Béier curve-based method proposed in this chapter.

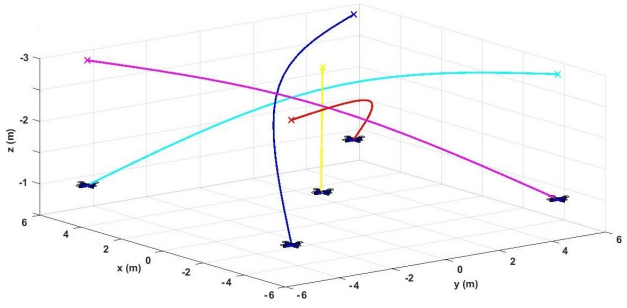
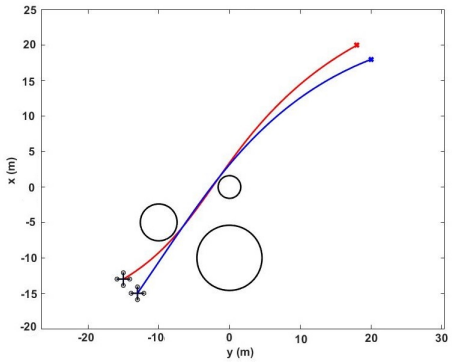
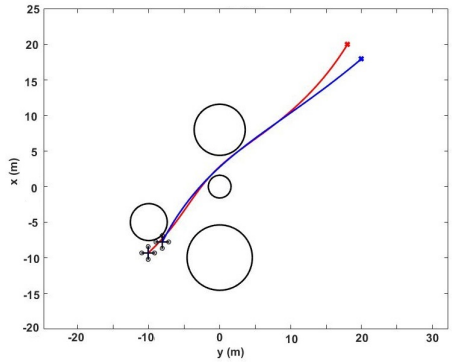


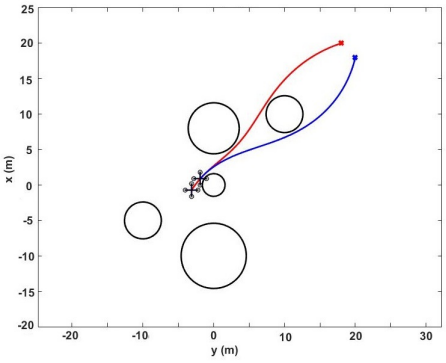
Figure 3.21: Collision-free trajectories for 5 drones generated with Bézier curves of degree 8.



(a) $t = 0$



(b) $t = 5.6s$



(c) $t = 6.8s$

Figure 3.22: To deal with the uncertainties in the obstacles' positions, trajectories for two drones are re-planned at different time instances. The trajectories are generated using Bézier curves of degree 8. Imposing continuity constraints at the joining point of consecutive segments does guarantee smoothness of the overall trajectory.

$$f = -T\,{}^{I}\mathbf{z}_B \tag{3.184}$$

where $T \in \mathbb{R}$ is the net thrust, ${}^{I}\mathbf{z}_B = R^B\mathbf{z}_B = R\mathbf{e}_3$ is the body frame $\mathbf{z}$-axis expressed in $\{I\}$, and $R \equiv {}^{I}_{B}R \in SO(3)$ is the rotation matrix from the body frame $\{B\}$, centered at the quadrotor's center of gravity, to the fixed inertial frame $\{I\}$. For simplicity, we drop the superscript ^{I} and consider $\mathbf{z}_B = {}^{I}\mathbf{z}_B$. Figure 3.23 is a graphical representation of the quadrotor and the associated reference frames.

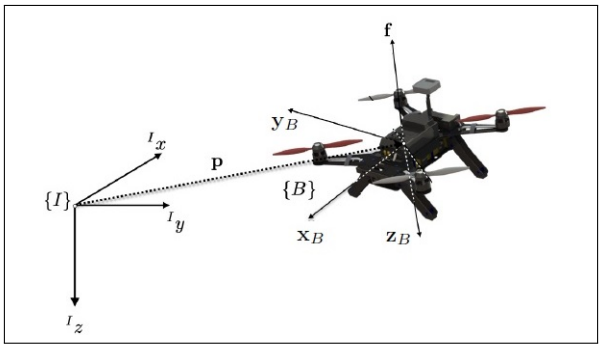


Figure 3.23: The quadrotor reference frames.

Trajectory Parameterization

The quadrotor dynamics (3.183) with the four inputs is differentially flat [MK11], and therefore the state and the input of the system can be expressed as functions of the flat outputs and a finite number of its derivatives. The position vector together with the yaw angle can be selected as flat outputs of the system. Here, $\mathbf{p} \in \mathbb{R}^3$ is parameterized as a Bézier curve, given by

$$\mathbf{p}(\tau) = \sum_{k=0}^{n} \mathbf{p}_k B_{k,n}(\tau) \tag{3.185}$$

where $\mathbf{p}_k \in \mathbb{R}^3$ are the control points. The linear velocity, $\mathbf{v} = \dot{\mathbf{p}}$, and linear acceleration, $\mathbf{a} = \ddot{\mathbf{p}}$, can be expressed as parametric Bézier curves through the first and second derivative of $\mathbf{p}$ with respect to time, yielding

$$\begin{aligned} \mathbf{v}(t) &= \sum_{k=0}^{n-1} \mathbf{v}_k B_{k,n-1}(\zeta(t)) \\ \mathbf{a}(t) &= \sum_{k=0}^{n-2} \mathbf{a}_k B_{k,n-2}(\zeta(t)) \end{aligned} \tag{3.186}$$

where the control points $\mathbf{v}_k$ and $\mathbf{a}_k$ are obtained as

$$\begin{aligned} \mathbf{v}_k &= \frac{n}{t_f}\big(\mathbf{p}_{k+1} - \mathbf{p}_k\big) && k = 0, \ldots, n-1 \\ \mathbf{a}_k &= \frac{n(n-1)}{t_f^2}\big(\mathbf{p}_{k+2} - 2\mathbf{p}_{k+1} + \mathbf{p}_k\big) && k = 0, \ldots, n-2 \end{aligned} \tag{3.187}$$

The thrust T and rotation matrix R can also be expressed as functions of the flat output and its derivatives. The net thrust T can be written as

$$T = m\|\ddot{\mathbf{p}} - g\mathbf{e}_3\|. \tag{3.188}$$

Assuming that the rotation matrix $R = [\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B]$ is parameterized by the Z-Y-X Euler angles $\boldsymbol{\lambda} = [\phi, \theta, \psi]^T$ as

$$R = R_z(\psi)R_y(\theta)R_x(\phi), \tag{3.189}$$

then the columns of the rotation matrix are extracted from

$$\mathbf{z}_B = \frac{\mathbf{g}\mathbf{e_3} - \ddot{\mathbf{p}}}{\|\mathbf{g}\mathbf{e_3} - \ddot{\mathbf{p}}\|} \quad \mathbf{x}_B = \frac{\mathbf{r} \times \mathbf{z}_B}{\|\mathbf{r} \times \mathbf{z}_B\|}, \quad \mathbf{y}_B = \mathbf{z}_B \times \mathbf{x}_B \tag{3.190}$$

where the unit vector $\mathbf{r}$ is defined as

$$\mathbf{r} = [-\sin\psi, \cos\psi, 0]^T \tag{3.191}$$

The above equations declare that the vehicle's orientation can be fully determined from the second derivative of the trajectory and the yaw angle. As mentioned before, the yaw angle ψ is a component of the flat output, and therefore it can be controlled independently without affecting the trajectory generation. Using the differential flatness property of the system, trajectories consistent with dynamics can be planned in the space of flat outputs, where (3.183) is trivially satisfied and the original input and state constraints are transformed into constraints on the flat output and its derivatives.

3.3.2.2 Collision Avoidance Constraint

Using the conventional way of modeling the drone body as a sphere can be too conservative and might lead to an infeasible problem for the examples described above. As explained before, approximating the drone body shape by an ellipsoid allows the incorporation of the drone attitude in the trajectory generation problem. This is a requisite for generating collision-free trajectories in small spaces. However, as opposed to spherical shapes, there is no simple algebraic inequality for detecting collisions between two ellipsoid-shaped bodies. In this section we use the separating hyperplane theorem [BBV04] of convex sets (See Remark 3.2, Theorem 2) to derive collision avoidance constraints between two drones.

The drone body is approximated by an ellipsoid centered at $\mathbf{c} = \mathbf{p}(t)$ with its principal axes aligned in the direction of the body frame axes.

$$E \equiv \{\mathbf{x} \in \mathbb{R}^3 | (\mathbf{x} - \mathbf{c})^T L L^T (\mathbf{x} - \mathbf{c}) \leq 1\} \tag{3.192}$$

where $L = R\Lambda^{-1}$ and Λ is a diagonal matrix with diagonal elements equal to the length of the principal semi-axis.

$$\Lambda = \begin{bmatrix} r_{\mathbf{D}} & 0 & 0 \\ 0 & r_{\mathbf{D}} & 0 \\ 0 & 0 & h_{\mathbf{D}} \end{bmatrix} \tag{3.193}$$

A separating hyperplane for two ellipsoids E_1 and E_2 is a hyperplane that has E_1 on one side of it and E_2 on the other side, that is to say the hyperplane H defined as,

$$H \equiv \{\mathbf{x} \in \mathbb{R}^3 | \boldsymbol{\alpha}^T \mathbf{x} - \beta = 0\} \tag{3.194}$$

is a separating hyperplane for E_1 and E_2 if,

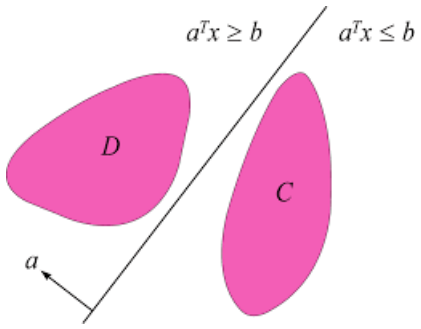
$$\begin{aligned} \boldsymbol{\alpha}^T \mathbf{x} - \beta \leq 0 \quad \forall \mathbf{x} \in E_1 \\ \boldsymbol{\alpha}^T \mathbf{x} - \beta > 0 \quad \forall \mathbf{x} \in E_2 \end{aligned} \tag{3.195}$$

The ellipsoid E_1 can be equivalently represented as,

Remark 3.2: Separating hyperplane theorem for convex sets

The separating hyperplane theorem is one of the most fundamental theorems about convex sets.

Theorem 1: Let C and D be two convex sets in $\mathbb{R}^n$ that do not intersect, i.e., $C \cap D = \emptyset$. Then, there exist $a \in \mathbb{R}^n, a \neq 0$ and $b \in \mathbb{R}$, such that $a^T x \leq b$ for all $x \in C$ and $a^T x \geq b$ for all $x \in D$ [BBV04].



Neither inequality in the theorem can be made strict. If the sets C and D are strictly separated, as is the case in the above picture, then $\exists a, b \; s.t. a^T x < b, \forall x \in C$ and $a^T x > b, \forall x \in D$. A special case of Theorem 1 with strict separation is as follows:

Theorem 2: Let C and D be two convex sets in $\mathbb{R}^n$ that do not intersect, i.e., $C \cap D = \emptyset$, and at least one of them is closed. Then, there exist $a \in \mathbb{R}^n, a \neq 0$ and $b \in \mathbb{R}$, such that $a^T x < b$ for all $x \in C$ and $a^T x > b$ for all $x \in D$.

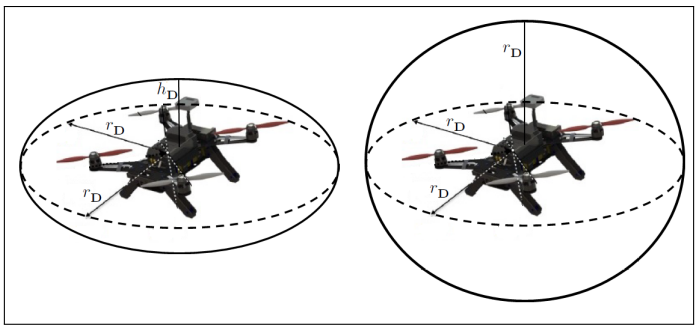


Figure 3.24: The quadrotor body can be represented as a sphere with radius $r_{\mathbf{D}}$ (**right**), or an ellipsoid aligned with the axes of the body frame (**left**). Approximating the drone body with an ellipsoid allows considering the quadrotor's rotational motion.

$$E_1 \equiv \{\mathbf{x} \in \mathbb{R}^3 | \mathbf{y}^T \mathbf{y} \leq 1, \mathbf{y} \equiv L_1^T (\mathbf{x} - \mathbf{c}_1)\} \tag{3.196}$$

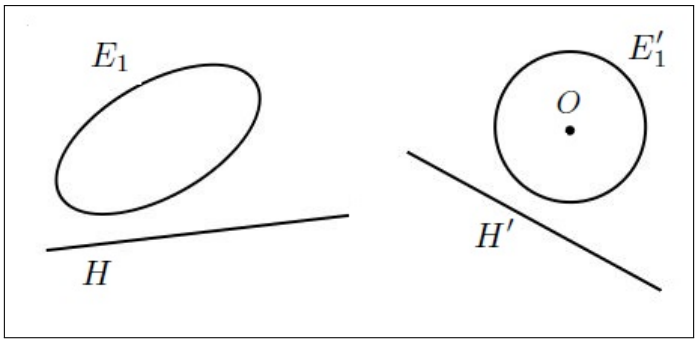


Figure 3.25: 2D sketch of an ellipsoid E_1 and a hyperplane H in the original space (**left**) and the corresponding sketch in the transformed space (**right**) in which E_1 is transformed into a unit ball at the origin.

With the transformation $\mathbf{y} \equiv L_1^T(\mathbf{x} - \mathbf{c}_1)$, E_1 is transformed into a unit ball at the origin and H into a hyperplane H',

$$H' \equiv \{\mathbf{y} \in \mathbb{R}^3 | \boldsymbol{\alpha}_1^T \mathbf{y} - \beta_1 = 0\} \tag{3.197}$$

where

$$\boldsymbol{\alpha}_1 = L_1^{-1}\boldsymbol{\alpha}, \quad \beta_1 = \beta - \boldsymbol{\alpha}^T \mathbf{c}_1 \tag{3.198}$$

If the following inequality holds,

$$|\beta - \boldsymbol{\alpha}^T \mathbf{c}_1| \geq \|L_1^{-1}\boldsymbol{\alpha}\| \tag{3.199}$$

then the unit ball and H' do not intersect and so do not E_1 and H in the original space. Accordingly, the following set of inequalities makes that H is a separating hyperplane for E_1 and E_2 centered at $\mathbf{p}_1(\tau)$ and $\mathbf{p}_2(\tau)$ [SCP21],

$$\begin{aligned} &(\boldsymbol{\alpha}(\tau)^T \mathbf{p}_1(\tau) - \beta(\tau))(\boldsymbol{\alpha}(\tau)^T \mathbf{p}_2(\tau) - \beta(\tau)) < 0 \\ &|\beta(\tau) - \boldsymbol{\alpha}(\tau)^T \mathbf{p}_i(\tau)| \geq \|L_i^{-1}(\tau)\boldsymbol{\alpha}(\tau)\|, \quad i = 1, 2. \end{aligned} \tag{3.200}$$

The normal vector $\boldsymbol{\alpha}(\tau)$ and the offset $\beta(\tau)$ are also parameterized as Bézier curves to describe the time evolution of the separating hyperplane,

$$\boldsymbol{\alpha}(\tau) = \sum_{i=0}^{n_\alpha} \alpha_i B_{i,n_\alpha}(\tau), \quad \beta(\tau) = \sum_{i=0}^{n_\beta} \beta_i B_{i,n_\beta}(\tau) \tag{3.201}$$

This approach is more efficient and adds on fewer number of optimization variables to the problem compared to the time gridding approach that requires finding one separating hyperplane at each sample time. Having $\boldsymbol{\alpha}$, β and $\mathbf{p}_i, i = 1, 2$ parameterized as Bézier curves and noting that

$$L_i^{-T} L_i^{-1} = R_i\big(r_{\mathbf{D}}^2 I - (r_{\mathbf{D}}^2 - h_{\mathbf{D}}^2)\mathbf{e_3}\mathbf{e_3}^T\big)R_i^T \tag{3.202}$$

$\|L_i^{-1}(\tau)\boldsymbol{\alpha}(\tau)\|^2$ can be expressed as a Bézier curve according to

$$\begin{aligned} \|L_i^{-1}(\tau)\boldsymbol{\alpha}(\tau)\|^2 =& r_{\mathbf{D}}^2\big(\boldsymbol{\alpha}(\tau)^T\boldsymbol{\alpha}(\tau)\big) \\ &- (r_{\mathbf{D}}^2 - h_{\mathbf{D}}^2)\big(\boldsymbol{\alpha}(\tau)^T \mathbf{z}_B(\tau)\big)^2. \end{aligned} \tag{3.203}$$

where $\mathbf{z}_{B,i} = R_i\mathbf{e}_3$ is fully obtained from $\ddot{\mathbf{p}}_i$ as stated in (3.190). Therefore, the inequalities in (3.200) can be effortlessly written as Bézier curves and evaluated with the approach described in Sec. (3.2.3.2). Note that the constraint for collision avoidance between an ellipsoid and a polygon can be easily derived in the same fashion as (3.200).

3.3.2.3 Simulation results

In this section, the efficacy of the proposed method for generating feasible and collision-free trajectories are assessed through different simulations involving one or more drones navigating narrow gaps. Here, the drones are assumed to have a hub-to-hub length of 360 mm, a height of 222 mm, and a propeller length of 230 mm. Therefore, the drone body is approximated by an ellipsoid with $r_D = 300$ mm and $h_D = 110$ mm. The cost function in all of the simulations below is given by

$$J = \int_0^1 \left\|\mathbf{p}^{(4)}(\tau)\right\|^2 d\tau + \rho t_f, \tag{3.204}$$

where t_f is the total flight time, and ρ is the scaling coefficient. A smaller ρ results in longer flight times. The lower and upper bounds on the first, second, third, and fourth derivatives of the trajectory are $7\frac{m}{s}$, $10\frac{m}{s^2}$, $50\frac{m}{s^3}$, and $60\frac{m}{s^4}$ respectively.

In the first example, we consider a single drone traversing a 120×48cm gap with $\phi_{gap} = 45°$ (Fig. 3.26a). Flying through such a gap while maintaining a safe distance of 80 mm away from the frame edges would not be possible without proper adjustments of pitch and roll angles. In order to traverse the narrow gap, the trajectory shown with the solid line is generated with the method proposed above which uses separating hyperplanes between the ellipsoid and edges to avoid collisions with the gap frame, without directly imposing constraints on attitude angles. The trajectory is generated using a Bézier curve of degree 12, and the inequalities are evaluated as explained in Sec. (3.2.3.2). The optimal solution is computed in 306 ms. The generated trajectory is compared to the one obtained with the method proposed in [Fal+17], which consists of a quadratic polynomial piece for traversing the gap, and an approach trajectory that guides the drone from its initial position to the starting position of the traverse trajectory. The approach trajectory in [Fal+17] is generated such that it also enables state estimation, while here it is obtained by minimizing the cost function (3.204); yet the traverse trajectory is generated the exact same way as in [Fal+17].

The traverse trajectory lies in the plane Π, that passes through the gap's center and is orthogonal to the gap while being parallel to the longest edge (See Fig 3.27). The drone's trajectory along Π is described as

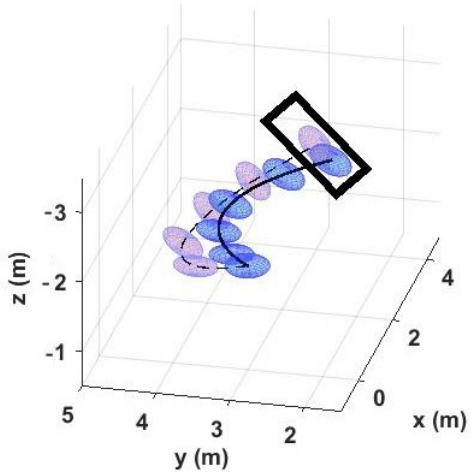
$$\mathbf{p}(t) = \mathbf{p}_1 + \mathbf{v}_1 t + \frac{1}{2}\mathbf{g}_\Pi t^2 \tag{3.205}$$

where $\mathbf{g}_\Pi = g\mathbf{e}_3 - g(\mathbf{e}_3^{T\Pi}\mathbf{e}_3)^\Pi\mathbf{e}_3$ and

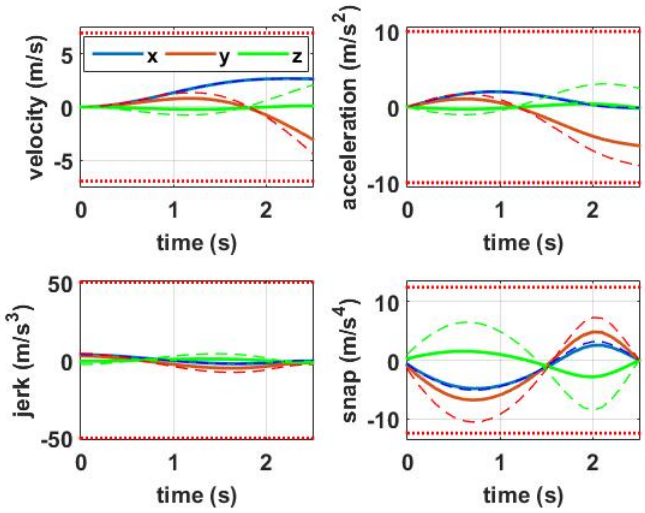
$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}(t_1) = \mathbf{p}_G - l_1{}^\Pi\mathbf{e}_1 - l_2{}^\Pi\mathbf{e}_2, \\ \mathbf{v}_1 &= \mathbf{v}(t_1) = \left(\frac{l_1}{t_c} - \frac{1}{2}\mathbf{g}_{\Pi,1}t_c\right)^\Pi\mathbf{e}_1 + \left(\frac{l_2}{t_c} - \frac{1}{2}\mathbf{g}_{\Pi,2}t_c\right)^\Pi\mathbf{e}_2 \end{aligned} \tag{3.206}$$

The unit vectors ${}^\Pi\mathbf{e}_1$ and ${}^\Pi\mathbf{e}_2$, spanning the plane Π, are determined from the gap's position and orientation. l_1 and l_2 denote respectively the distance of the starting position, $\mathbf{p}_1$, to the center of the gap along ${}^\Pi\mathbf{e}_1$ and ${}^\Pi\mathbf{e}_2$, and are obtained by solving an optimization problem that minimizes t_c defined as

$$t_c = \sqrt{\frac{2l_1}{|\mathbf{g}_{\Pi,1}|}} \tag{3.207}$$



(a) Trajectories guiding a drone through a narrow gap inclined at 45° with respect to the horizontal plane. The ellipsoids show the flight attitude at different time samples along the trajectories. The objective function values for the solid line and dashed line are 1.0071×10^3 and 2.7777×10^3 respectively.



(b) The velocity, acceleration, jerk, and snap along the x-y-z axis for the above generated trajectories. The dotted lines show the lower and upper bounds.

Figure 3.26: Comparing the proposed method in the thesis to the method in [Fal+17] for generating a trajectory that guides a drone through a gap inclined at 45°.

It can be readily observed from the trajectories and the input profiles in Fig. 3.26 that enforcing the trajectory to lie on the plane Π can ensure collision avoidance with the gap frame, but it does not yield the optimal solution. The proposed method outperforms the method in [Fal+17] in the sense that it yields a less conservative trajectory, offering a reduced value of the cost function.

The results in Fig. 3.28, Fig. 3.29, and Fig. 3.30 display the trajectories, thrust and input profiles obtained with the proposed method for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a 120mm × 550mm gap inclined at 30°, 45°, and 90° respectively. Fig. 3.31 compares the trajectory generated with the proposed method for a gap with a frame size of 180mm × 550mm inclined at 60° to the one obtained from the frequently used approach that aligns the drone's orientation with that of the gap as flying past its center. The following inequality constraint can ensure the proper alignment of the body z-direction with a desired direction $\mathbf{d}$,

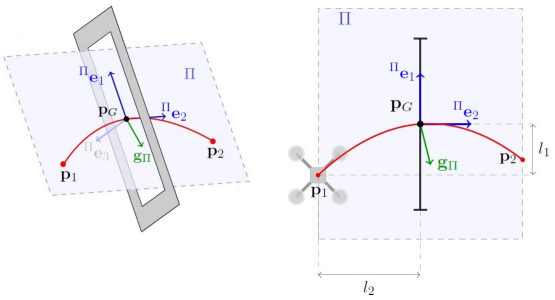
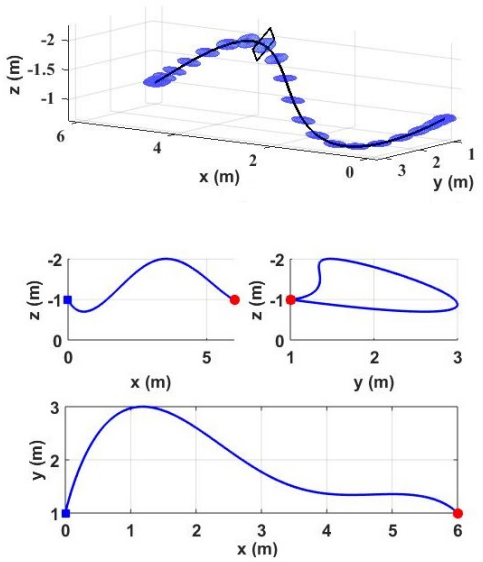


Figure 3.27: The orthogonal plane Π to an inclined gap [Fal+17] (**left**). The drone's trajectory must pass through the center of the gap, $\mathbf{p}_G$, while lying in the plane Π. A view of the traverse trajectory in the direction of the normal vector to the plane $^{\Pi}\mathbf{e}_3$ (**right**).

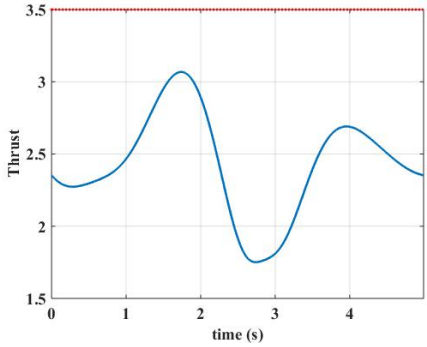
$$\cos\alpha_d \leq \mathbf{d}^T\mathbf{z}_B \leq 1 \tag{3.208}$$

where α_d is the permissible angular deviation from $\mathbf{d}$. Both $\mathbf{d}$ and α_d are determined according to the size of the drone and the actual orientation and width of the gap. It should be noted that the inequality constraint (3.208) should be imposed at specified times when the drone is flying through the gap center. Also, while using (3.208) minimizes the collision risk with the gap frame, it might yield an increase in the cost value. This can be clearly observed from the input profile corresponding to each trajectory shown in (Fig. 3.31). The proposed method is also evaluated and demonstrated successfully in real experiments involving a single drone flying through a narrow gap inclined at different orientations. Fig. 3.32 shows the experimental results for a gap inclined at 60°.

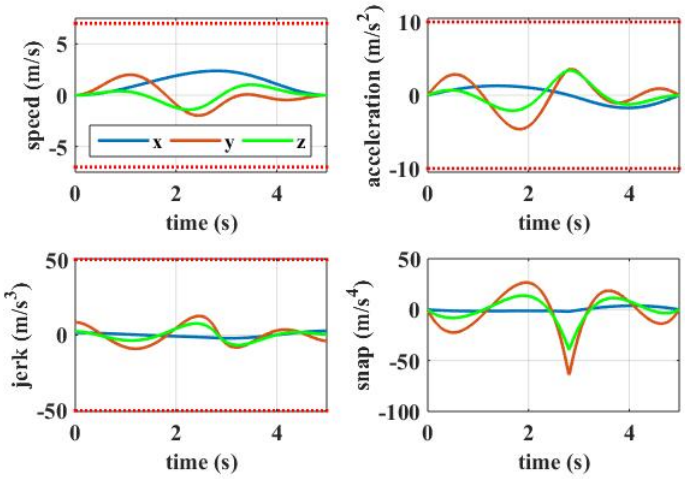
The performance of the proposed method is also evaluated in trajectory generation problems for multiple vehicles. In the next example, we consider two drones switching their positions. Figure (3.33) compares the trajectories generated with an ellipsoid model of the drone body to those generated with the widely used sphere model. Clearly, the sphere model takes no notice of the drone's shape and orientation, and imposes a conservative bound on the minimum distance between two drones. This results in an extended arc-length as shown in Fig. 3.33. The main advantage of using an ellipsoid model, however, becomes evident when dealing with confined spaces where a sphere model yields an infeasible problem. This is better shown in the next two examples. Fig. 3.34 and Fig. 3.35 show, respectively, the generated trajectories for two and four drones flying through a small gap in a wall towards their desired positions on the other side of the wall. The set of inequality constraints (4.9) with separating hyperplanes, rotating and translating over time, is incorporated into the optimization problem to guarantee collision avoidance between the ellipsoids. Both examples are solved with given fixed final times, i.e., $t_f = 1.5$s and $t_f = 3.5$s, respectively. The average computation time for obtaining the optimal trajectories in Fig. 3.35 and Fig. 3.34, parameterized as Bézier curves of degree 12 and degree 9, is 712 ms and 481 ms, respectively.



(a) The generated trajectory is visualized from different angles. The initial and final positions are shown with • and •, respectively.

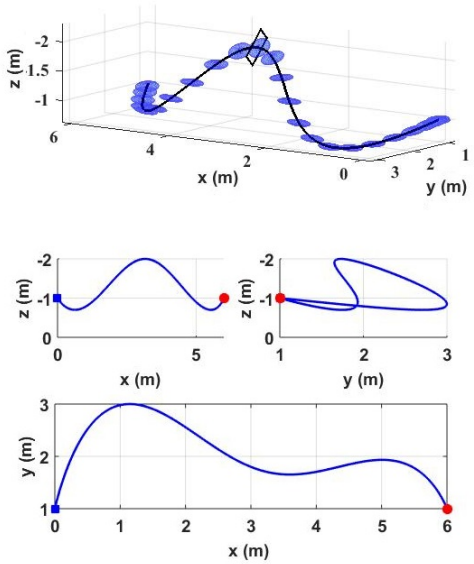


(b) The net thrust required for executing the trajectory in (a).

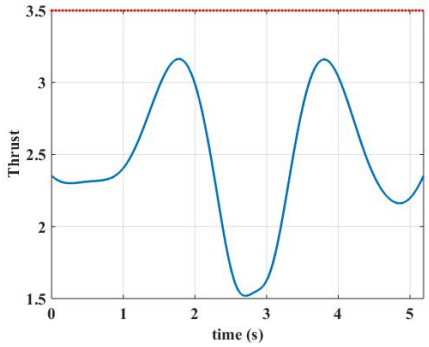


(c) The speed, acceleration, jerk, and snap along the x-y-z axis for the generated trajectory in (a). The dotted lines show the lower and upper bounds.

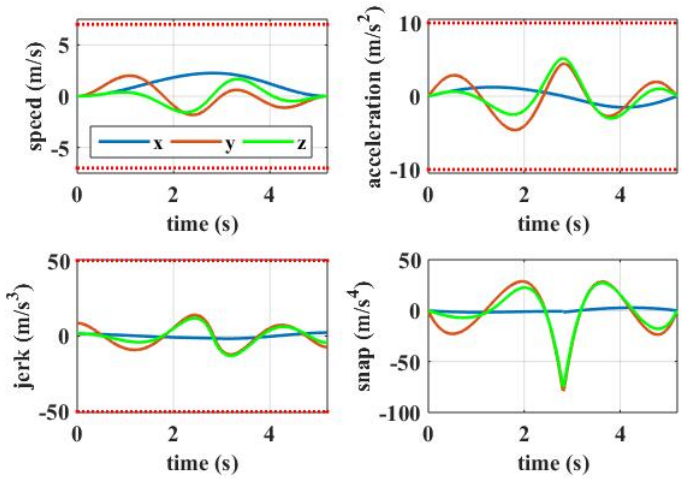
Figure 3.28: The generated trajectory for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a gap with the frame size of 120mm × 550mm, centered at $[3.4, 1.6, -2]$ and inclined at 30°.



(a) The generated trajectory is visualized from different angles. The initial and final positions are shown with • and •, respectively.

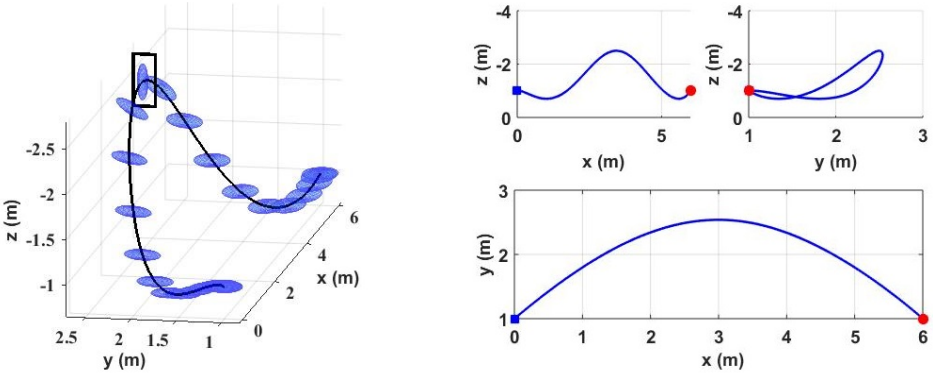


(b) The net thrust required for executing the trajectory in (a).

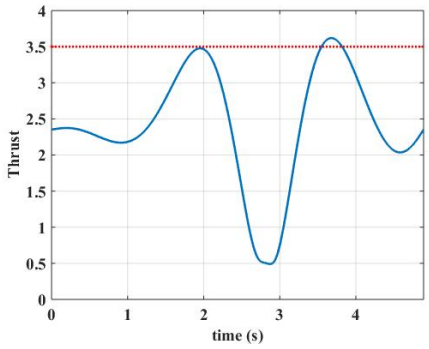


(c) The speed, acceleration, jerk, and snap along the x-y-z axis for the generated trajectory in (a). The dotted lines show the lower and upper bounds.

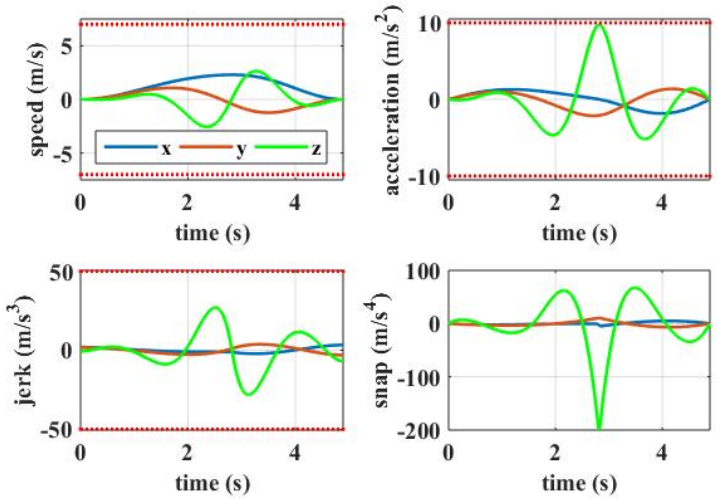
Figure 3.29: The generated trajectory for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a gap with the frame size of 120mm $\times$ 550mm, centered at $[3.3, 1.7, -2]$ and inclined at 45°.



(a) The generated trajectory is visualized from different angles. The initial and final positions are shown with • and •, respectively.

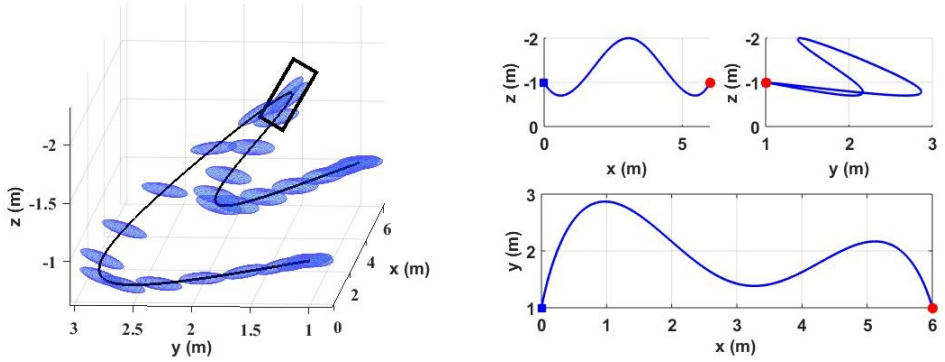


(b) The net thrust required for executing the trajectory in (a).

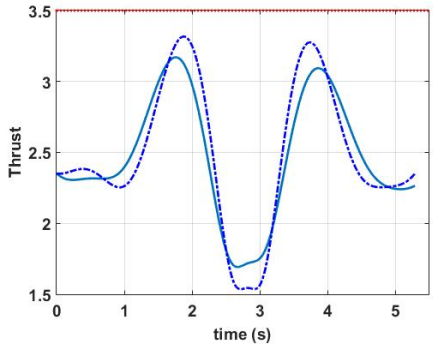


(c) The speed, acceleration, jerk, and snap along the x-y-z axis for the generated trajectory in (a). The dotted lines show the lower and upper bounds.

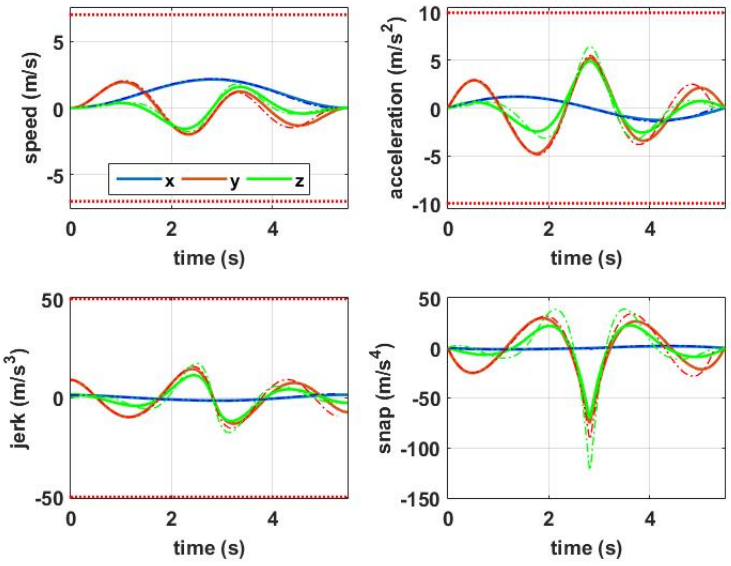
Figure 3.30: The generated trajectory for flying a drone with $h_{\mathbf{D}} = 30$ mm and $r_{\mathbf{D}} = 150$ mm through a gap with the frame size of 120mm × 550mm, centered at $[3.5, 2, -2.5]$ and inclined at 90°.



(a) The trajectory generated with the proposed method is visualized from different angles. The initial and final positions are shown with • and •, respectively.

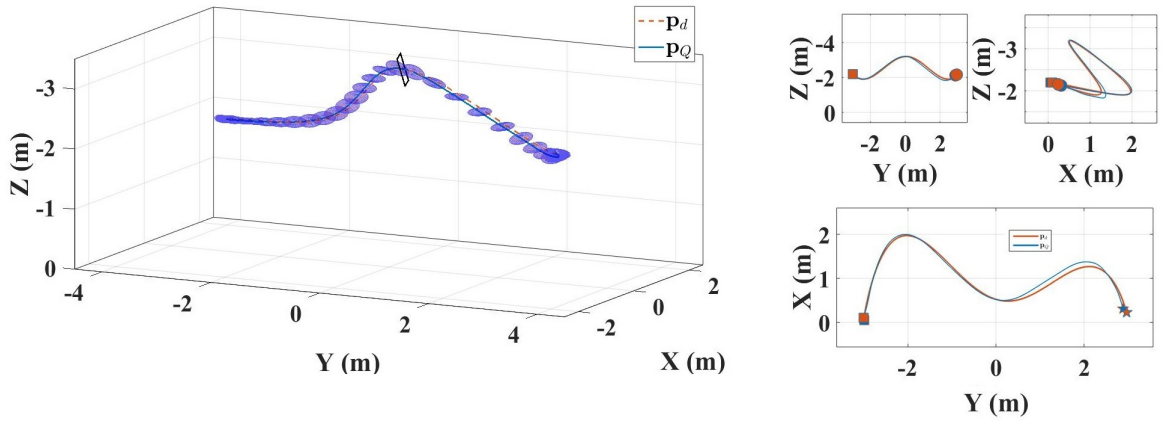


(b) The net thrust required for the trajectories generated with the two methods.



(c) The speed, acceleration, jerk, and snap along the x-y-z axis obtained with the two methods. The dotted lines show the lower and upper bounds.

Figure 3.31: Comparing the trajectory generated with the proposed method in the thesis (solid line) to the one generated using (3.208) (dashed line) for flying a drone through a 180mm × 550mm gap centered at $[3, 1.4, -2]$ inclined at 60°.



(a) The generated trajectory (red) and the trajectory executed in the experiment (blue).

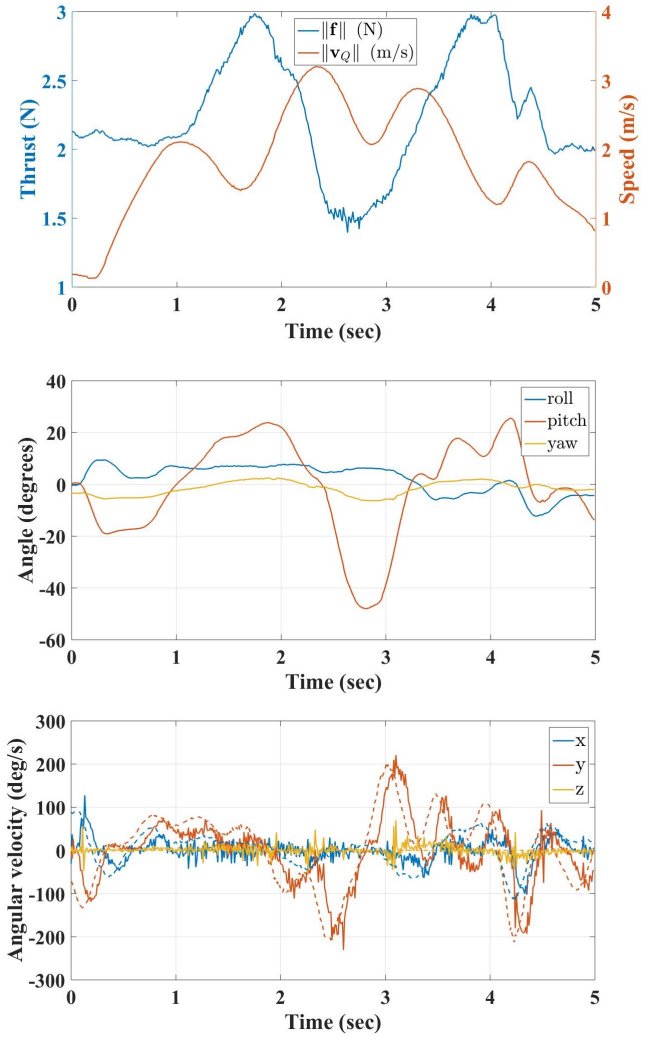


Figure 3.32: The experimental results for a single drone flying through a gap inclined at 60°.

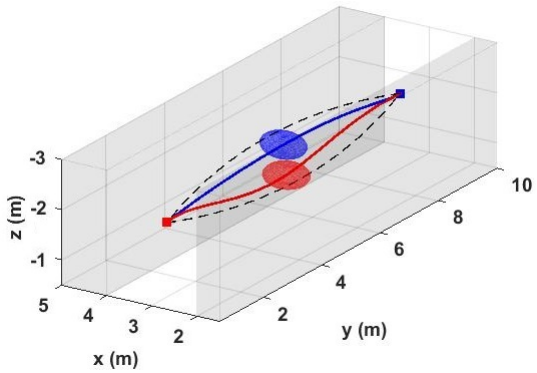


Figure 3.33: Comparing collision-free trajectories, for two drones switching positions, generated with an ellipsoid model of the drone body (solid lines) and a sphere model (dashed lines).

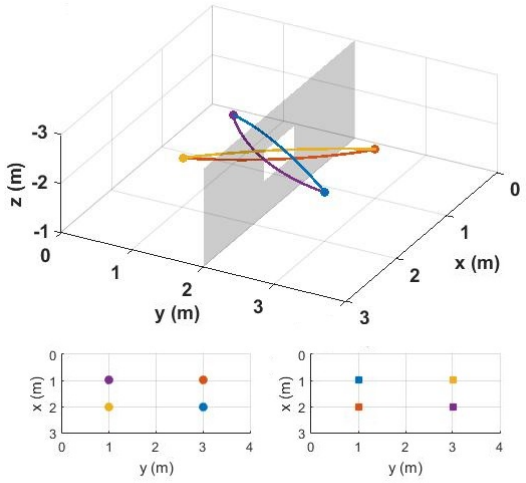


Figure 3.34: Generated trajectories for 4 drones flying through a gap and switching positions in a given time. The initial (**left**) and final (**right**) positions of the drones are shown in the XY-plane. Using a sphere model of the drone body yields an infeasible problem.

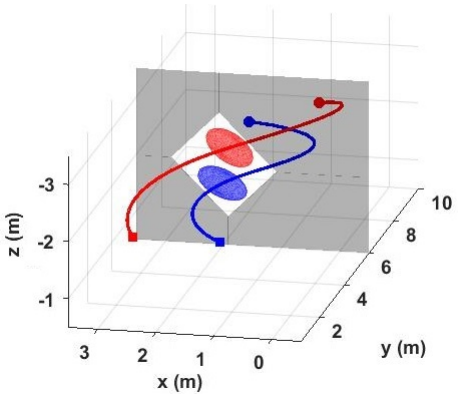


Figure 3.35: Collision-free trajectories for steering two drones through a narrow gap in a wall towards their desired position on the other side. The initial and final positions are shown with squares and circles respectively.

Chapter 4

Distributed Algorithm for Real-time Multi-drone Trajectory Re-planning

In this chapter we develop a distributed trajectory generation framework, with low computation and communication demands, for multiple quadrotors flying in (relatively) close proximity to each other. We specifically address the shortcomings of approximating the drone body with a disc (or sphere) for generating feasible collision-free trajectories for large groups of drones. A sphere model, used in most existing distributed collision avoidance schemes, may be overly conservative in confined spaces since it invalidates trajectories whose feasibility depends on the consideration of the flight attitude. Instead, we model the drone body with an ellipsoid, and employ the Voronoi partitioning of space to derive local collision avoidance constraints that take into account the drone's real size and orientation. The approach presented here can be integrated into any other distributed schemes that utilize separating hyperplanes for decoupling collision avoidance constraints. Yet the main reason for adopting Voronoi diagram is that using time invariant boundary hyperplanes determined prior to solving a sub-problem, despite being more conservative, can significantly reduce communication and computational load, allowing for higher re-planning rates.

Incorporating the resulting set of constraints into the optimization sub-problems, solved by each vehicle, allows finding collision-free trajectories for guiding a group of drones through confined spaces by proper adjustment of attitude angles. Also, we show that the obtained set of constraints can be expressed as Bézier curves, and hence can be efficiently evaluated with the proposed method in the previous chapter. Hence, it can be guaranteed that inter-vehicle collision avoidance requirements are met at any instant of time even over a long planning horizon.

In the synchronous distributed scheme presented here, each vehicle uses the position information of its neighbors, updated at each sampling time, and solves a sub-problem to generate its trajectory inside (a subset of) its Voronoi cell towards the closest point (in the cell) to its goal position. We also present an efficient method to compute this point, which is needed to appropriately define the terminal constraint and cost in the sub-problem. A sequence of sub-problems are then solved in a receding-horizon manner until the vehicles reach their goal positions. We show that the proposed method has a higher success rate at finding collision-free trajectories for larger groups of quadrotors compared to other Voronoi diagram-based methods. We also show that it can effectively reduce the total flight time required to perform point-to-point maneuvers. We illustrate that the computation time required to generate trajectories with the proposed method satisfies timing constraints of real-time applications.

4.1 Literature Review

For a large group of vehicles, the trajectory generation problem translates into an optimization problem that involves a large number of constraints and decision variables. The computational cost of solving such an optimization problem centrally can be prohibitively high. To reduce the computational complexity, a multitude of distributed schemes have been proposed for decomposing the optimization problem into smaller sub-problems that can be solved locally by each vehicle. The major challenge is to ensure that local decisions do also satisfy the coupling collision avoidance constraints. This is mainly addressed by exchanging information among the vehicles on their current states, future input sequences, etc. Depending on the communication strategy, the sub-problems might be solved sequentially or concurrently, with possibly several iterations of optimization and communication to achieve the required performance.

In [KH11], the collision avoidance constraint, usually expressed in terms of the two-norm of a relative position vector, is approximated by a set of linear constraints. The sub-problem for each vehicle is then formulated as a mixed-integer linear programming (MILP) that includes the vehicle's individual variables as well as variables of a subset of neighbors. This enables cooperation among vehicles by allowing a vehicle to make feasible perturbations to neighboring vehicles' decisions. The sub-problems are solved sequentially by each vehicle, and the algorithm iterates over the group of vehicles until convergence, during each cycle of a model predictive

control (MPC) scheme.

Sequential Convex Programming (SCP)-based methods have also been used for solving distributed multiple vehicle trajectory generation problems [ASD12],[MCH14]. [CCH15] addresses the infeasiblity of intermediate problems in decoupled-SCP methods, arising from convex approximation of collision-avoidance constraints, i.e. linearizing them, and proposes incremental SCP (iSCP) which tightens collision constraints incrementally. Compared to sequential approaches in [Kuw+07], [CHL10], [Ted+10], that cast the trajectory of anterior vehicles as dynamic obstacles for a posterior vehicle, the methods proposed in [CCH15] and [KH10] result in less constrained intermediate problem and faster convergence rate, yet, similar to most MPC-SCP-based methods, they would require the vehicles to exchange a full representation of their decisions to neighboring vehicles over a communication network.

The synchronous approach in [VP17a] extends the ADMM-based distributed MPC (dMPC) scheme, developed in [VP17b] for formation control, to problems with inter-vehicle collision avoidance constraints. These constraints are decoupled using separating hyperplanes, which enforces each vehicle to stay within one half-space of a time-varying plane for a certain time horizon. The resulting sub-problems are solved simultaneously by vehicles, while the normal vector and offset shared between a vehicle and a neighbor, for characterizing their separating hyperplane, are updated at each cycle of the dMPC, using the interchanged information about generated trajectories at the previous cycle.

In the decentralized trajectory planner proposed in [TH21], vehicles re-plan their trajectories asynchronously, independent of the planning status of other vehicles. At each iteration, a vehicle considers trajectories assigned to neighboring vehicles as constraints, and solves an optimization problem including as decision variables the normal and offset of planes that separate the outer polyhedral representation of its trajectory and those of its neighbors. A check-recheck scheme is then performed to ensure that the generated trajectory does not collide with trajectories other vehicles have committed to during the optimization time. Therefore, to guarantee deconfliction between vehicles, the planner requires a vehicle to broadcast its computed trajectory to its neighboring vehicles at the end of each re-planning iteration.

The on-demand approach to local collision avoidance, proposed in [LS19], imposes constraints only at specific time instances when collisions between a vehicle and its neighbors are predicted. Predicting collisions along a time horizon, however, relies on an accurate knowledge of the neighbors' future actions which must be communicated at every sampling time. The dMPC scheme in [LVS20] for distributed trajectory generation is based on this predict-avoid paradigm and an event-trigered replanning strategy, and has been shown to result in less conservative trajectories, but at the cost of voiding the collision avoidance guarantees for all time instances over the horizon. To capture the downwash effect of quadrotor's propellers, the collision avoidance constraint in [LVS20] is modified with a diagonal scaling matrix, which approximates the quadrotor body with a translating ellipsoid elongated along the vertical axis, yet it ignores the quadrotor's rotational motion.

Reciprocal velocity obstacle (RVO) and its variants have been widely used in distributed collision avoidance [VLM08], [Ber+11b], [Ber+11a], [Alo+12], [Sna+11]. At each time step, RVO [VLM08] builds the set of all relative velocities leading to a collision between a vehicle and its neighbors, and chooses a new constant velocity outside this set, and closest to the desired value, to avoid collisions. Therefore, RVO requires the position and velocity information to be communicated, or sensed, between nearby neighbors. Other variants such as Acceleration Velocity Obstacle (AVO), which addresses the instantaneous change of velocity in RVO by taking into account acceleration constraints, need further information like acceleration to be interchanged. Reciprocally-Rotating Velocity Obstacle (RRVO) [GLA14] uses rotation information to mitigate deadlocks caused by symmetries of representing vehicles with translating discs in RVO. It relies on the assumption that neighbors may rotate equally (or equally opposite), bounded by a maximum value, to compute an approximation of swept areas for rotating polygon-shaped vehicles,

and use them for constructing Velocity Obstacles. A new velocity and rotation is then selected at each time step to avoid collisions.

Another approach to distributed collision avoidance is to construct the Voronoi diagram of the group of vehicles and generate the trajectory for each vehicle so that it's entirely within the vehicle's Voronoi cell [Bor00], [Gar+06], [BG08], [Cor+04]. Since Voronoi cells do not overlap, it can be guaranteed that the generated trajectories are collision-free. To consider the physical size of a vehicle, modified Voronoi cell used in [BCH14],[Zho+17] retracts boundary hyperplanes of the cell by a safety radius for disc-shaped vehicles. At each sampling time, upon receiving the relative position information, trajectories are re-planned to conform to the updated Voronoi diagram. The resulting sub-problems can be solved simultaneously, in a receding horizon manner until vehicles reach their final positions. The Voronoi-based approaches only require the vehicles to know relative positions to neighboring vehicles, and therefore is well suited to applications where vehicles only have relative position sensing and no communication network [ŞHA19].

4.2 problem formulation

The multiple vehicle trajectory generation problem addressed in this chapter can be defined as finding optimal trajectories that guide a group of vehicles from their initial positions to some desired final positions. The generated trajectories should jointly minimize a cost function, corresponding to the accomplishment of mission goals and objectives, and satisfy a set of local and coupling constraints, so that they are dynamically-feasible and collision-free. For N_v vehicles, this problem can be formulated as the following optimal control problem.

$$\min_{\substack{\mathbf{u}_i(.)\\ i\in[N_v]}} \sum_{i\in[N_v]} J(\mathbf{x}_i(.),\mathbf{u}_i(.)) \tag{4.1}$$

$$\begin{aligned}
s.t.\quad & \dot{\mathbf{x}}_i(t) = f(\mathbf{x}_i(t),\mathbf{u}_i(t)) && \text{(Dynamics)}\\
& \mathbf{x}_i(0) = \mathbf{x}_{i,0} && \text{(Initial state)}\\
& \mathbf{x}_i(t_f) = \mathbf{x}_{i,f} && \text{(Final state)}\\
& c(\mathbf{x}_i(t),\mathbf{x}_j(t)) \le 0 \quad j\in[N_v]\backslash\{i\} && \text{(collision avoidance)}\\
& \mathbf{x}_i(t) \in \mathcal{X}_i && \text{(State Constraints)}\\
& \mathbf{u}_i(t) \in \mathcal{U}_i && \text{(Input constraints)}
\end{aligned}$$

where $[N_v] = \{1,\dots,N_v\}$. The cost to be minimized is the sum of the vehicles' individual costs, J, given by the functional,

$$J[\mathbf{u}_i(.)] = \int_0^{t_f} L(\mathbf{x}_i,\mathbf{u}_i)dt \tag{4.2}$$

where $\mathbf{x}_i(t) \in \mathbb{R}^{n_x}$ and $\mathbf{u}_i(t) \in \mathbb{R}^{n_u}$ are the state and the input vectors of the vehicle's model described by an ODE, and, $\mathbf{x}_{i,0}$ and $\mathbf{x}_{i,f}$ are the initial and final values of the state of the i-th vehicle, respectively. $\mathcal{X}_i$ and $\mathcal{U}_i$ denote the set of admissible states and inputs for the i-th vehicle derived from limits imposed by vehicle dynamics and the surrounding environment.

In order to reduce the computational complexity of solving (4.1) for large N_v with increased numbers of constraints and variables, one can divide the problem into a set of small-scale sub-problems. Here, the sub-problems are formulated such that each involves only a vehicle's individual costs and constraints, and hence can be solved independently by vehicles. The sub-problems must include constraints to ensure that the trajectory generated locally by a vehicle does satisfy the coupling collision avoidance constraints.

The key idea to ensure inter-vehicle collision avoidance is to decompose the environment into non-overlapping regions, provided by a Voronoi diagram, and generate the trajectory for

each vehicle such that it's entirely within its partition. The Voronoi diagram is updated at each sampling time according to the relative positions of vehicles, and a sequence of sub-problems is solved in a receding horizon manner until the vehicles reach their final positions. For the i-th vehicle, the problem that has to be solved at the time instant t_k can be expressed as,

$$\begin{aligned} \min_{\mathbf{x}_{i,k}(.),\mathbf{u}_{i,k}(.)} \quad & J(\mathbf{x}_{i,k}(.),\mathbf{u}_{i,k}(.)) && (4.3)\\ s.t. \quad & \dot{\mathbf{x}}_{i,k}(t) = f(\mathbf{x}_{i,k}(t),\mathbf{u}_{i,k}(t)) && \text{(Dynamics)}\\ & \mathbf{x}_{i,k}(t_k) = \hat{\mathbf{x}}_{i,k} && \text{(Initial state)}\\ & \mathbf{x}_{i,k}(t) \in \mathcal{C}_{i,k}(\bar{\mathbf{x}}_k) && \text{(collision avoidance)}\\ & \mathbf{x}_{i,k}(t) \in \mathcal{X}_{i,k} && \text{(State Constraints)}\\ & \mathbf{u}_{i,k}(t) \in \mathcal{U}_{i,k} && \text{(Input constraints)} \end{aligned}$$

where $\mathbf{x}_{i,k}(t)$ and $\mathbf{u}_{i,k}(t)$ are the state and the input profiles of the vehicle over the time interval $[t_k, t_k + t_h]$, with t_h being the planning horizon, and $\hat{\mathbf{x}}_{i,k}$ denotes its state at the time instant t_k. The cost function in the above sub-problem is modified as,

$$J[\mathbf{u}_{i,k}(.)] = \int_{t_k}^{t_k+t_h} L(\mathbf{x}_{i,k},\mathbf{u}_{i,k})dt + \phi(\mathbf{x}_{i,k}(t_k+t_h)) \tag{4.4}$$

where the second term is added to penalize the distance, at $t_k + t_h$, to the point in the Voronoi partition that is closest to the goal position.

In the optimization problem (4.3), $\mathcal{C}_{i,k}$ may denote the Voronoi partition of the i-th vehicle. The Voronoi diagram is updated for each sub-problem according to the vehicles' configuration at that time instant, i.e. $\bar{\mathbf{x}}_k = \{\hat{\mathbf{x}}_{j,k}\}_{j\in[N_v]}$. Since Voronoi partitions are disjoint and the assigned trajectory to the each vehicle for the time horizon t_h is contained within its partition, it can be guaranteed that there is no collision between the trajectories over the time interval $[t_k, t_k + t_h]$.

The distributed trajectory generation framework is summarized in Algorithm 3. In Sec. 4.2.1, we study how to define the Voronoi diagram for a group of vehicles, and then we modify $\mathcal{C}_{i,k}$ to explicitly take into account the orientation while generating collision-free trajectories for multiple drones.

Algorithm 3 Distributed Trajectory Generation Framework

1: $k = 0$
2: $\hat{\mathbf{x}}_{i,0} \leftarrow$ Initial position of the i-th vehicle
3: **repeat**
4: Receive position information from neighbors
5: Broadcast own position to neighbors
6: Update Voronoi partition
7: Compute the closest point in the Voronoi partition to the goal position ▷ Sec. 4.2.2
8: Set the cost function (4.4)
9: Set the constraints ▷ Sec. 4.2.1, Sec. 4.2.3
10: Solve the optimization sub-problem
11: **until** $\hat{\mathbf{x}}_{i,k} = \mathbf{x}_{i,f}$.

4.2.1 Decoupling the inter-vehicle collision avoidance constraint

In this section, we present a Voronoi diagram-based approach to decoupling inter-vehicle collision avoidance constraints. Although the presence of obstacles, interpreted as non-decision-making

agents, is not explicitly considered here, incorporating vehicle–obstacle collision avoidance constraints into the problem simply amounts to taking into account the obstacles' position when updating the Voronoi partition (step 6 of Algorithm 3).

The widely used approach in the literature to avoid collisions between a drone and obstacles is to model the drone body as a sphere with radius $r_{\mathbf{D}}$, and then simply building the collision-free space, $\mathcal{C}_{free}$, by inflating the obstacles with a factor $r_{\mathbf{D}}$. As a result, collision-free trajectories can be obtained by enforcing the vehicle, which is now treated as a point in the space, to be inside $\mathcal{C}_{free}$ [LaV06]. Considering now the collision avoidance between the i-th and j-th drones, the corresponding constraint can be derived similarly by

$$\|\mathbf{p}_i - \mathbf{p}_j\| \geq 2r_{\mathbf{D}} \tag{4.5}$$

where $\|.\|$ denotes the Euclidean distance. Ignoring the real shape and orientation of the drone, and approximating its body with a sphere, invalidates trajectories that are feasible upon considering the flight attitude. For this reason, the above approach might be too conservative for trajectory generation in confined spaces and can even fail to find feasible collision-free trajectories when multiple drones are involved.

Approximating the drone body with an ellipsoid, whose principal axis is aligned with the body frame axes, allows for considering the drone orientation while inspecting for collisions against other vehicles. The collision-avoidance constraints for two ellipsoid-shaped drones, as proposed in the previous chapter, can be derived using separating hyperplanes. The ellipsoids E_i and E_j, corresponding to the i-th and j-th drones, defined as

$$E_i \equiv \{\mathbf{p} \in \mathbb{R}^3 | \mathbf{p} = \mathbf{p}_i + R_i \Lambda \mathbf{w}, \|\mathbf{w}\| \leq 1\}, \tag{4.6}$$

do not intersect if they satisfy

$$\begin{aligned} \boldsymbol{\alpha}^T \mathbf{p} - \beta \leq 0 \quad \forall \mathbf{p} \in E_i \\ \boldsymbol{\alpha}^T \mathbf{p} - \beta > 0 \quad \forall \mathbf{p} \in E_j \end{aligned} \tag{4.7}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^3$ and $\beta \in \mathbb{R}$ are, respectively, the normal vector and offset of the separating hyperplane for E_i and E_j, defined as $H \equiv \{\mathbf{p} \in \mathbb{R}^3 | \boldsymbol{\alpha}^T \mathbf{p} - \beta = 0\}$.

Since

$$-\|\Lambda R^T \boldsymbol{\alpha}\| \leq \boldsymbol{\alpha}^T R \Lambda \mathbf{w} \leq \|\Lambda R^T \boldsymbol{\alpha}\| \tag{4.8}$$

the set of inequalities (4.7) holds if and only if,

$$\begin{aligned} \boldsymbol{\alpha}^T \mathbf{p}_i - \beta \geq \|\Lambda R_i^T \boldsymbol{\alpha}\| \\ \boldsymbol{\alpha}^T \mathbf{p}_j - \beta \leq -\|\Lambda R_j^T \boldsymbol{\alpha}\|. \end{aligned} \tag{4.9}$$

Satisfying the set of constraints (4.9) will guarantee that the two ellipsoids E_i and E_j, centered at $\mathbf{p}_i$ and $\mathbf{p}_j$ and rotated with R_i and R_j, do not intersect, and hence there is no collision between the i-th and j-th drones.

For multiple vehicle trajectory generation, collision-avoidance constraints, either in the form of the inequality constraint (4.5), for spheres, or the set of constraints (4.9), for ellipsoids, must be incorporated in the optimization problem for each pair of vehicles. As the number of vehicles involved in a mission grows, the resulting increase in the number of constraints would inevitably exacerbate the computational issues of finding collision-free trajectories in a centralized manner.

A. Collision avoidance constraint for algorithm 3

We now use (4.9) and the Voronoi partitioning of space to derive collision-avoidance constraints that can be integrated into the distributed trajectory generation framework of algorithm 3. Incorporating the resulting constraints into the vehicle's sub-problem (4.3) can guarantee that the generated trajectory is entirely within (a subset of) the vehicle's Voronoi cell, for the time interval t_h, taking into account the shape and orientation of drones. Since Voronoi cells are pairwise disjoint, the locally generated trajectories are collision-free for all future time before t_h.

Each Voronoi cell in an n-dimensional space is a convex polytope bounded by a number of $(n-1)$-dimensional convex polytopes. For a group of vehicles in a 3-dimensional space, the general Voronoi cell of the i-th vehicle is defined as

$$\mathcal{V}_i = \{\mathbf{p} \in \mathbb{R}^3 | {\mathbf{p}_{ij}}^T \big(\mathbf{p} - \frac{1}{2}(\mathbf{p}_i + \mathbf{p}_j)\big) \leq 0, \forall j \in [N_v] \backslash \{i\}\} \tag{4.10}$$

where $\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i$, and $\mathbf{p}_i$ and $\mathbf{p}_j$ are the position of the i-th and j-th vehicles at the current time instant. Note that, $\mathcal{V}_i$ is the intersection of half-spaces corresponding to hyper-planes with $\alpha = \mathbf{p}_{ij}$ and $\beta = \frac{1}{2}\mathbf{p}_{ij}^T(\mathbf{p}_i + \mathbf{p}_j)$. An arbitrary point in $\mathcal{V}_i$ is closer to the i-th vehicle than any other vehicle [Cor+04], i.e.,

$$\|\mathbf{p} - \mathbf{p}_i\| \leq \|\mathbf{p} - \mathbf{p}_j\|, \quad \forall \mathbf{p} \in \mathcal{V}_i \quad \& \quad j \neq i \tag{4.11}$$

The boundary of the Voronoi cell, $\partial\mathcal{V}_i$, is the union of multiple faces, each of which include points in the space that are equidistant to the i-th vehicle and a neighboring vehicle.

In order to account for the size of a vehicle, the Buffered Voronoi Cell (BVC) proposed in [Zho+17] retracts the boundary of the general Voronoi cell by a safety radius, so that if the vehicle's center is inside the BVC, its body, approximated by a sphere of radius $r_{\mathbf{D}}$, will be entirely within its Voronoi cell. The BVC of the i-th vehicle, denoted by $\bar{\mathcal{V}}_i$, is defined as

$$\bar{\mathcal{V}}_i = \{\mathbf{p} \in \mathbb{R}^3 | \quad \mathbf{p}_{ij}^T(\mathbf{p} - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) + r_{\mathbf{D}}\|\mathbf{p}_{ij}\| \leq 0, \forall j \in [N_v] \backslash \{i\}\} \tag{4.12}$$

For a group of vehicles in a collision-free configuration, the BVC defined as above has the following properties,

1. $\bar{\mathcal{V}}_i \subset \mathcal{V}_i$.
   According to the BVC definition, for any point $\mathbf{p}' \in \bar{\mathcal{V}}_i$, we have

   $$\mathbf{p}_{ij}^T(\mathbf{p}' - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) \leq -r_{\mathbf{D}}\|\mathbf{p}_{ij}\| \leq 0 \tag{4.13}$$

   which also satisfies (4.10), and thus $\mathbf{p}' \in \mathcal{V}_i$.

2. $\forall \mathbf{p}' \in \bar{\mathcal{V}}_i$ and $\forall \mathbf{q}' \in \bar{\mathcal{V}}_j$, $\|\mathbf{p}' - \mathbf{q}'\| \geq 2r_{\mathbf{D}}$.
   According to (4.12), we have

   $$\begin{aligned} \mathbf{p}_{ij}^T(\mathbf{p}' - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) &\leq -r_{\mathbf{D}}\|\mathbf{p}_{ij}\| \\ \mathbf{p}_{ji}^T(\mathbf{q}' - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) &\leq -r_{\mathbf{D}}\|\mathbf{p}_{ji}\|. \end{aligned} \tag{4.14}$$

   Adding the above two inequalities, we get

   $$\mathbf{p}_{ij}^T(\mathbf{p}' - \mathbf{q}') \leq -2r_{\mathbf{D}}\|\mathbf{p}_{ij}\| \tag{4.15}$$

   Using the Cauchy-Schwarz inequality, we can conclude that

   $$\|\mathbf{p}' - \mathbf{q}'\| \geq \frac{\|\mathbf{p}_{ij}^T(\mathbf{p}' - \mathbf{q}')\|}{\|\mathbf{p}_{ij}\|} \geq \frac{2r_{\mathbf{D}}\|\mathbf{p}_{ij}\|}{\|\mathbf{p}_{ij}\|} = 2r_{\mathbf{D}}. \tag{4.16}$$

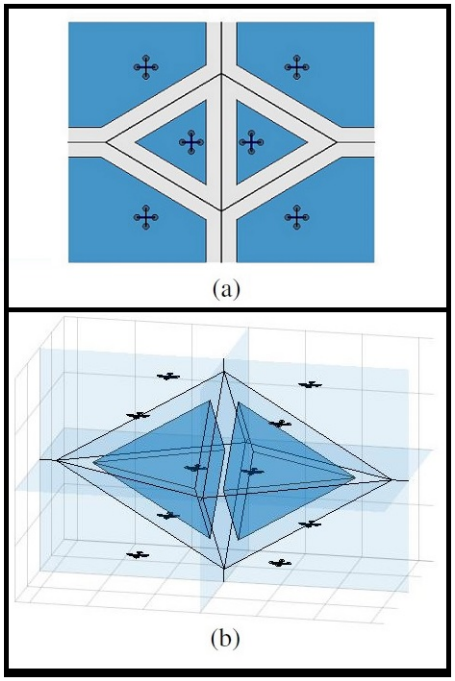


Figure 4.1: (**a**) The Voronoi diagram for six drones in 2D space. The Voronoi boundary edges are shown with solid black lines, and the buffered Voronoi cells are shaded in dark blue. (**b**) The Voronoi diagram for 10 drones in a collision-free configuration in 3D space. The Voronoi boundary $\partial\mathcal{V}$ is shaded in light blue, and the buffered Voronoi cells $\bar{\mathcal{V}}$ for two neighboring drones in the center are shown in dark blue.

3. $\bar{\mathcal{V}}_i \cap \bar{\mathcal{V}}_j = \emptyset, \forall i \neq j$.

   For an arbitrary point $\mathbf{p}' \in \bar{\mathcal{V}}_i$, we have

$$\mathbf{p}_{ji}^T(\mathbf{p}' - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) + r_{\mathbf{D}}\|\mathbf{p}_{ji}\| \geq r_{\mathbf{D}}\|\mathbf{p}_{ij}\| + r_{\mathbf{D}}\|\mathbf{p}_{ij}\| = 2r_{\mathbf{D}}\|\mathbf{p}_{ij}\|, \tag{4.17}$$

   which contradicts to the definition of $\bar{\mathcal{V}}_j$. Similarly, for an arbitrary point $\mathbf{q}' \in \bar{\mathcal{V}}_j$, we get $\mathbf{q}' \notin \bar{\mathcal{V}}_i$. Therefore, the two sets $\mathcal{V}_i$ and $\mathcal{V}_j$ are disjoint.

Considering the above properties, the vehicles are guaranteed to avoid collisions due to the buffer region of $r_{\mathbf{D}}$ along $\partial\mathcal{V}_i$. Fig. 4.1b shows the Voronoi diagram for 10 drones in a collision-free configuration and the BVC for two adjacent drones.

The BVC is defined based on a symmetrical approximation of the vehicle's body with a translating disc. In order to reduce the conservatism and avoid infeasibility issues due to ignoring the real shape and orientation of the vehicle, we approximate the drone with an ellipsoid (4.6), and propose $\mathcal{C}_i$ in problem (4.3), to be defined as [SCP22]

$$\mathcal{C}_i = \{(\mathbf{p}, \ddot{\mathbf{p}}) \in \mathbb{R}^6 | \mathbf{p}_{ij}^T(\mathbf{p} - \frac{1}{2}(\mathbf{p}_j + \mathbf{p}_i)) + \|\Lambda R^T \mathbf{p}_{ij}\| \leq 0, \forall j \in [N_v]\backslash\{i\}\} \tag{4.18}$$

If the trajectory of the i-th drone $\mathbf{p}_i(t)$ satisfies the above set of local collision avoidance constraints for all $t \in [t_k, t_k + t_h]$, then the ellipsoid representing the drone body is within the Voronoi cell for the entire time horizon; that is the ellipsoid centered at $\mathbf{p}_i$ and aligned with the columns of R_i does not intersect the Voronoi boundary, stated mathematically

$$\|\partial E_i - \partial\mathcal{V}_i\| \geq 0 \tag{4.19}$$

Noting that

$$h_{\mathbf{D}}\|\mathbf{p}_{ij}\| \leq \|\Lambda R^T \mathbf{p}_{ij}\| \leq r_{\mathbf{D}}\|\mathbf{p}_{ij}\|, \tag{4.20}$$

it can be induced that,

$$\bar{\mathcal{V}}_i(r_{\mathbf{D}}) \subset \text{proj}_{XYZ}\mathcal{C}_i \subset \bar{\mathcal{V}}_i(h_{\mathbf{D}}) \subset \mathcal{V}_i \tag{4.21}$$

where $\text{proj}_{XYZ}\mathcal{C}_i$ is the projection of $\mathcal{C}_i$ onto the 3 dimensional subspace spanned by $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$.

Therefore, incorporating (4.18) into the optimization problem (4.3) will ensure that the generated trajectories are collision-free while alleviating infeasibility problems by taking orientations into account. Also, since

$$\|\Lambda R^T \mathbf{p}_{ij}\|^2 = r_{\mathbf{D}}^2\|\mathbf{p}_{ij}\|^2 - (r_{\mathbf{D}}^2 - h_{\mathbf{D}}^2)(\mathbf{p}_{ij}^T \mathbf{z}_B)^2 \tag{4.22}$$

and considering that $\mathbf{z}_B$ is fully obtained from $\ddot{\mathbf{p}}$ (3.190), the above set of local collision avoidance constraints can be expressed as constraints in Bézier form, and thus, they can be evaluated efficiently using the method proposed in Sec. 3.2.3.2.

4.2.2 Finding the closest point to the goal position

As explained above, at each time instant the Voronoi cell $\mathcal{V}_i$ is updated according to the relative position of the i-th vehicle to other vehicles. The optimization problem (4.3) is then solved to generate a trajectory, for a time horizon t_h, that guides the vehicle towards the point in the cell closest to the goal position. This process is repeated until the vehicle reaches its final position. At each sampling time, the closest point must be found prior to solving the trajectory generation problem. Therefore, having an efficient scheme for finding the closest point is critically important to avoid long computational delays between updating the Voronoi cell and re-planning the trajectory.

The point in a convex polytope that is closest to a query point $\mathbf{q}$ is either $\mathbf{q}$ itself or a point on the boundary of the polytope. A naive way to find the closest point in a convex polytope in a 3-dimensional space, represented by $P = (\mathbb{F}, \mathbb{E}, \mathbb{V})$, where $\mathbb{F}$ is the set of faces, $\mathbb{E}$ is the set of edges and, $\mathbb{V}$ is the set of vertices, is to check the distance between $\mathbf{q} \in \mathbb{R}^3$ to all faces, edges, and vertices for finding the minimum. However, for complex polytopes the computation time is not negligible.

The geometric approach proposed in [Zho+17] for a polygon in a 2-dimensional space calculates the barycentric coordinates and an angle from the query point to the two vertices of each edge to find out the closest point. Since this approach iterates over all edges, its computational complexity can significantly increase as the number of Voronoi neighbors of a vehicle increases. Here, we make use of the GJK distance algorithm and devise an approach that can efficiently determine whether the query point is inside the polytope, i.e. $\mathbf{q} \in P$, in which case the closest point is $\mathbf{q}$ itself. Otherwise, the presented algorithm returns the closest feature of P to $\mathbf{q}$, and the closest point can be obtained by projecting $\mathbf{q}$ onto it. The proposed approach is not limited to distance queries between a point and a polytope, and can also be used when the final constraint in (4.3) is relaxed to a small box or sphere around the goal position, and used in conjunction with a terminal cost term.

In the following we review the building blocks of the GJK algorithm. The original GJK algorithm is very versatile and can be used for distance queries between convex objects in general [Eri04]. However, as we will explain below, the algorithm can be implemented more efficiently for polytopes.

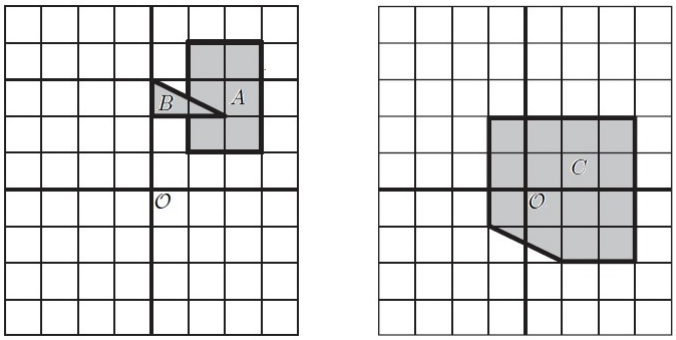


Figure 4.2: The Minkowski difference of two convex polygons A and B. Since A and B intersect, C contains the origin.

4.2.2.1 Overview

The GJK distance algorithm, or simply GJK algorithm, is an iterative algorithm that casts the distance query between two convex shapes into a point-simplex polytope distance query in a configuration space where a unique solution exists. The algorithm relies on a support mapping function to incrementally build simplices that are closer to the query point [GJK88]. GJK has been extensively used for collision detection between generic convex shapes [Van03], [Eri04]. The original algorithm, however, can be used to compute the minimum distance, and also the respective pair of (closest) points, between two convex shapes [TCF13].

4.2.2.2 Configuration Space Obstacle

The Minkowski difference between two convex sets A and B, referred to as the *translational configuration space obstacle* [Eri04], is defined as

$$\begin{aligned} C &= A \ominus B \\ &= A \oplus (-B) = \{c : c = a - b; a \in A, b \in B\} \end{aligned} \tag{4.23}$$

which is a set of vectors in an affine space. C can be visualized as the area swept by A translated to every point in $-B$, the reflection of B over the origin. It can be proven that C is convex. More particularly, if A and B are convex polygons, then C is also a convex polygon and its vertices are the differences of the vertices of A and B.

The distance query between A and B can be translated into finding the point with minimum norm in C, indicated with $\nu(C)$, i.e.,

$$\begin{aligned} \text{distance } (A, B) &= \min \{\|a - b\| : a \in A, b \in B\} \\ &= \min \{\|c\| : c \in C\} = \|\nu(C)\| \end{aligned} \tag{4.24}$$

If A and B have one or more points in common then C includes the origin and $\|\nu(C)\| = 0$ (See Fig. 4.2), otherwise A and B are disjoint and their distance is greater than zero (See Fig. 4.3). Since C is convex, its point of minimum norm, $\nu(C)$, is unique.

There exists algorithms for building the Minkowski difference, however, computing the entire Minkowski difference is computationally infeasible at runtime. As we will see below, the GJK algorithm can be implemented without building the entire C explicitly, as it only samples the points in C using a support mapping function.

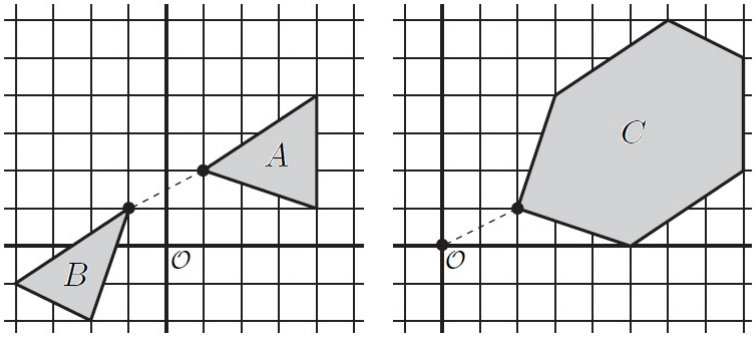


Figure 4.3: Finding the minimum distance between two convex bodies A and B is equivalent to finding the minimum distance of their Minkowski difference, C, to the origin.

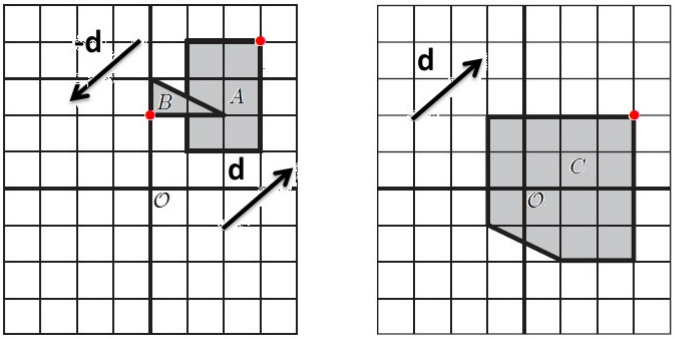


Figure 4.4: The support point of the Minkowski difference can be determined by subtracting the support points of the two shapes.

4.2.2.3 Support Mapping Function

GJK relies on the so-called support mapping function to construct a new simplex. A support mapping function $s_C(\mathbf{d})$ of the convex set C maps a given vector $\mathbf{d}$ to a point in the set, called the support point, according to

$$\mathbf{d}^T\mathbf{w} = \mathbf{d}^T s_C(\mathbf{d}) = \max\{\mathbf{d}^T\mathbf{p}; \mathbf{p} \in C\} \tag{4.25}$$

Therefore, the support point $\mathbf{w}$ can be interpreted as the furthest point in C along the direction $\mathbf{d}$. The support point is not necessarily unique. For $\mathbf{d} = \mathbf{0}$ any point in C may be returned as a support point. However, the support mappings in GJK is imposed to return a point on the boundary of the object.

The support mapping function defined above is the maximum over a linear function, and can be expressed in terms of the support mappings for A and B (See Fig. 4.4), i.e.,

$$s_C(\mathbf{d}) = s_A(\mathbf{d}) - s_B(-\mathbf{d}) \tag{4.26}$$

Therefore, points from the Minkowski difference, C, can be computed, on demand, from the supporting points of A and B.

A support point of a convex polytope can be computed efficiently. For a polytope P, the support point is a vertex of P,

$$\mathbf{w} = s_P(\mathbf{d}) \in \text{vert}(P) \tag{4.27}$$

obtained such that

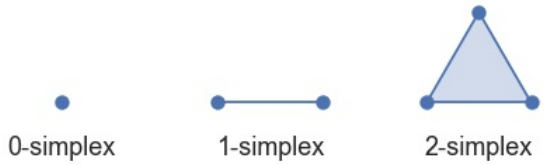


Figure 4.5: Simplices in $\mathbb{R}^2$.

$$\mathbf{d}^T s_P(\mathbf{d}) = \max\{\mathbf{d}^T \mathbf{v}; \mathbf{v} \in \text{vert}(P)\}. \tag{4.28}$$

Therefore, for polytopes, the support point can be uniquely determined by simply scanning through the list of vertices for the vertex that is the most extreme in the search direction $\mathbf{d}$. Therefore, the computation time is linear in the number of vertices of P. For complex polytopes, the vertices adjacency information and the coherence between consecutive calls to support mapping functions can be exploited to find the support point with almost constant time complexity [Van03].

4.2.2.4 Simplices

A m-simplex in $\mathbb{R}^n$, is the convex hull of a set of $m+1$ ($m \leq n$) affinely independent vertices in $\mathbb{R}^n$. Fig. 4.5 illustrates the simplices in $\mathbb{R}^2$. In $\mathbb{R}^3$, a simplex can be a point, a line, a triangle or a tetrahedron with 1, 2, 3, and 4 vertices, respectively. A simplex in $\mathbb{R}^3$ has $2^{m+1}-1$ vertices, edges, faces, and volume, each of which associated with a Voronoi region in the three-dimensional space. The Voronoi region of a feature of the simplex is the set of points that are at least closer to one point of the feature than any other point of the simplex not included in the feature. Examples of Voronoi regions for 1-simplex and 2-simplex in $\mathbb{R}^2$ are shown in Fig. 4.6. For a simplex indicated with $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_{m+1}\}$, the Voronoi region associated to a feature of the simplex is denoted by $\mathcal{V}_\iota$, where ι is an ordered tuple listing the indices of the vertices specifying the feature.

The GJK algorithm builds simplices that are subsets of the Minkowski difference C by selecting the vertices on the boundary of C. This follows immediately from the fact that C is a compact set and every convex combination of the points in C is a point in C. Fig. 4.10 shows simplices in a representative C in three-dimensional space.

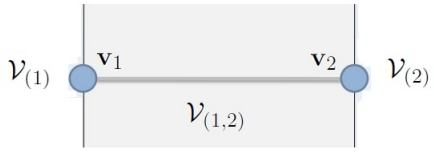
The GJK algorithm uses the results of the Carathéodory's theorem, which says each point of convex body, $H \in \mathbb{R}^n$, can be expressed as the convex combination of $n+1$ or fewer points of H. This allows expressing a point in a simplex as a convex combination of its vertices,

$$\mathbf{c} = \sum_{i=1}^{m+1} \lambda_i \mathbf{v}_i, \qquad \lambda_i \geq 0, \sum_{i=1}^{m} \lambda_i = 1. \tag{4.29}$$

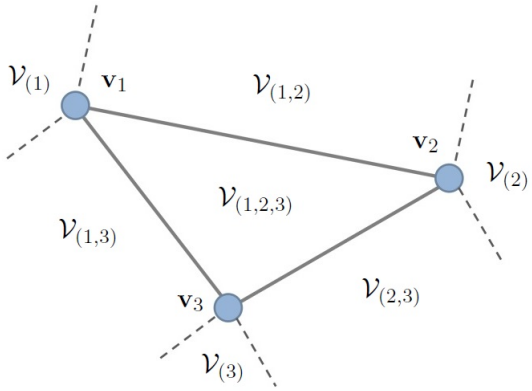
More specifically, a point lying on a feature of the simplex specified with V_ι, is expressed as

$$\mathbf{c} = \sum_{i \in \iota} \lambda_i \mathbf{v}_i, \qquad \lambda_i > 0, \sum_{i \in \iota} \lambda_i = 1. \tag{4.30}$$

The coefficients λ_i are called the barycentric coordinates.



(a) Voronoi regions of a 1-simplex



(b) Voronoi regions of a 2-simplex

Figure 4.6: Each feature of a simplex is linked to a Voronoi region.

4.2.2.5 Convergence and termination

In order to obtain the minimum distance between two general convex bodies A and B, the GJK algorithm approximates the closest point in their Minkowski difference to the origin, i.e. $\nu(C)$, with an iterative search. The algorithm relies on the support mapping function to incrementally build simplices in C that are closer to the origin [GJK88]. At each iteration, a search direction is set as

$$\mathbf{d}_k = -\boldsymbol{\nu}_k \tag{4.31}$$

where $\boldsymbol{\nu}_k$ is minimum norm point of the current simplex whose vertices are indicated with V_k, i.e.,

$$\boldsymbol{\nu}_k = \nu(\text{conv}(V_k)) \tag{4.32}$$

A support point $\mathbf{w}_k = s_C(-\boldsymbol{\nu}_k)$ is added as a vertex to the current simplex. V_{k+1} is then updated such that it only contains the smallest set of vertices in $Y_k = V_k \cup \{\mathbf{w}_k\}$ that supports $\boldsymbol{\nu}_{k+1} = \nu(\text{conv}(Y_k))$, and earlier vertices that do not back ν_{k+1} are discarded. (Algorithms for computing $\boldsymbol{\nu}$ and updating V are explained in the next section.)

Since $V_k \subset Y_k$ it may be concluded that

$$\|\nu(V_{k+1})\| = \|\nu(V_k \subset \{\mathbf{w}_k\})\| \leq \|\nu(V_k)\|, \tag{4.33}$$

and thus, $\boldsymbol{\nu}_k$ calculated at each iteration is closer to the origin than the previous one, More specifically

$$\|\boldsymbol{\nu}_{k+1}\| \leq \|\boldsymbol{\nu}_k\| \tag{4.34}$$

with equality only if $\boldsymbol{\nu}_k = \nu(C)$ [GJK88]. Fig. 4.7 illustrates the descent nature of the GJK algorithm. It is shown in [Van03] that $\|\boldsymbol{\nu}_k\|^2 - \boldsymbol{\nu}_k^T \mathbf{w}_k \geq 0$, with equality only if $\boldsymbol{\nu}_k = \nu(C)$. Although $\mathbf{w}_k$ is not necessarily unique, $\|\boldsymbol{\nu}_k\|^2 - \boldsymbol{\nu}_k^T \mathbf{w}_k$ is uniquely defined for each $\boldsymbol{\nu}_k$, and provides an upper bound for the squared distance between $\boldsymbol{\nu}_k$ and $\nu(C)$, that is

$$\|\boldsymbol{\nu}_k - \nu(C)\|^2 \leq \|\boldsymbol{\nu}_k\|^2 - \boldsymbol{\nu}_k^T \mathbf{w}_k \tag{4.35}$$

Also, the absolute error in $\|\boldsymbol{\nu}_k\|$ is bounded by

$$\|\boldsymbol{\nu}_k\| - \|\nu(C)\| \leq \|\boldsymbol{\nu}_k - \nu(C)\| \leq \gamma_k \tag{4.36}$$

where $\gamma_k = \text{sqrt}(\|\boldsymbol{\nu}_k\|^2 - \boldsymbol{\nu}_k^T \mathbf{w}_k)$ [Van03]. Since γ_k is continuous in $\boldsymbol{\nu}_k$ and $\gamma_k = 0$ for $\boldsymbol{\nu}_k = \nu(C)$, it can be inferred that the error bound converges to zero as $\boldsymbol{\nu} \to \nu(C)$. Therefore, for a given positive absolute error tolerance ϵ_{abs}, there exist a k such that $\gamma_k \leq \epsilon_{\text{abs}}$.

Algorithm 4 describes the GJK algorithm [Van03] for general convex shapes. The algorithm may terminate if $|V| = 4$ or $\|\boldsymbol{\nu}\|^2 \leq \epsilon_{\text{tol}} \|\mathbf{y}_{\max}\|^2$, both of which establish that $\nu(\text{aff}(V)) = 0$; Or it may terminate if $\boldsymbol{\nu}_k$ is sufficiently close to $\nu(C)$, that is

$$\|\boldsymbol{\nu}\|^2 - \boldsymbol{\nu}^T \mathbf{w} \leq \epsilon_{\text{rel}}^2 \|\boldsymbol{\nu}\|^2 \tag{4.37}$$

for a given tolerance value $\epsilon_{\text{rel}} > 0$. The GJK algorithm, as described in Alg. 4, depends heavily on the computation of $\boldsymbol{\nu}_k$ for testing the termination condition and finding the search direction. The original GJK algorithm employs the *Johnson Distance Subalgorithm* [GJK88] (explained in the next section) to compute $\boldsymbol{\nu}_k$ at each iteration.

Here, we exploit unique features of polytopes and present a faster way to evolve simplices in the GJK algorithm. Instead of computing $\boldsymbol{\nu}_k$ at each iteration, we employ an easily computable search direction $\mathbf{d}_k \uparrow\downarrow \boldsymbol{\nu}_k$ to find the support point, and only compute the minimum norm point when the algorithm terminates. The termination condition is also modified appropriately such that it is independent of $\boldsymbol{\nu}_k$.

The following proposition offers a criterion for deciding weather or not $\boldsymbol{\nu}_k = \nu(\text{conv}(V_k)) = \nu(C)$ without explicitly computing $\boldsymbol{\nu}_k$.

Proposition 1. $\mathbf{d}_k^T(\mathbf{w}_k - \mathbf{v}_1) \geq 0$, *with equality only if* $\boldsymbol{\nu}_k = \nu(\text{conv}(V_k)) = \nu(C)$.

Proof. (i) From the definition of the support mapping function we have

$$\mathbf{d}_k^T \mathbf{w}_k \geq \mathbf{d}_k^T \mathbf{c} \quad \forall \mathbf{c} \in C \tag{4.38}$$

In particular, for $\mathbf{c} = \mathbf{v}_1$, we get

$$\mathbf{d}_k^T \mathbf{w}_k \geq \mathbf{d}_k^T \mathbf{v}_1 \tag{4.39}$$

and thus, $\mathbf{d}_k^T(\mathbf{w}_k - \mathbf{v}_1) \geq 0$.

(ii) Now, assuming that $\mathbf{d}_k^T(\mathbf{w}_k - \mathbf{v}_1) = 0$, and expressing the minimum norm point as $\boldsymbol{\nu}_k = -\alpha \mathbf{d}_k, \alpha > 0$, we derive

$$\begin{aligned}
\|\boldsymbol{\nu}_k\|^2 &\leq \|\boldsymbol{\nu}_k\|^2 + \|\boldsymbol{\nu}_k - \mathbf{c}\|^2 \\
&= \|\mathbf{c}\|^2 - 2(\boldsymbol{\nu}_k^T \mathbf{c} - \|\boldsymbol{\nu}_k\|^2) \\
&= \|\mathbf{c}\|^2 - 2\alpha(\mathbf{d}_k^T \boldsymbol{\nu}_k - \mathbf{d}_k^T \mathbf{c}) \\
&= \|\mathbf{c}\|^2 - 2\alpha(\mathbf{d}_k^T \mathbf{v}_1 - \mathbf{d}_k^T \mathbf{c}) \\
&= \|\mathbf{c}\|^2 - 2\alpha(\mathbf{d}_k^T \mathbf{w}_k - \mathbf{d}_k^T \mathbf{c}) \\
&\leq \|\mathbf{c}\|^2,
\end{aligned} \tag{4.40}$$

From (4.40), $\boldsymbol{\nu}_k$ is closer to the origin than any other point $\mathbf{c} \in C$ and thus, is the unique point of minimum norm in C, i.e., $\boldsymbol{\nu}_k = \nu(C)$.

□

In (4.40), we make use of the fact that the search direction $\mathbf{d}_k$ is perpendicular to the convex hull of V_k, and thus we have

$$\mathbf{d}_k^T(\boldsymbol{\nu}_k - \mathbf{v}) = 0 \quad \forall \mathbf{v} \in \text{conv}(V_k). \tag{4.41}$$

A natural choice, as used above, is $\mathbf{v}_1 \in V_k$, yielding $\mathbf{d}_k^T \boldsymbol{\nu}_k = \mathbf{d}_k^T \mathbf{v}_1$.

For a polytope P, GJK arrives at the actual $\nu(P)$ in a finite number of iterations [Van03], and thus, the stop criterion for the conditional loop in the algorithm can be replaced by

$$\mathbf{d}_k^T(\mathbf{w}_k - \mathbf{v}_1) \leq 0 \tag{4.42}$$

If (4.42) holds then V_k represents the closest feature of the polygon to the origin, i.e., $\nu(\text{conv}(V_k)) = \nu(P)$, and the point of minimum norm in P can be obtained by projecting the origin onto the convex hull of V_k as

$$\nu(P) = \frac{\mathbf{d}_k^T \mathbf{v}_1}{\|\mathbf{d}_k\|^2} \mathbf{d}_k. \tag{4.43}$$

Algorithm 5 summarizes GJK with all necessary modifications to have it run efficiently for polytopes. The algorithm may terminate if $|V| = 4$, in which case aff(V) is the whole space and $\nu(P)$ must be the origin; or it may return V_k representing a vertex, an edge or a face of P if (4.42) holds. Figure 4.8 shows possible outputs of Algorithm 5 for a representative polyhedron.

To derive the termination condition (4.42) we used the fact that the search direction $\mathbf{d}_k$ is in the opposite direction of $\boldsymbol{\nu}_k$. In Sec. 4.2.2.6.A we explain the algorithm for updating $\mathbf{d}_k$ at each iteration.

4.2.2.6 Johnson's Distance Sub-algorithm

The Johnson's distance sub-algorithm is an algorithm for computing the point of minimum norm, $\boldsymbol{\nu}_{k+1} = \nu(\text{conv}(V_k \cup \{\mathbf{w}_k\}))$, and the smallest subset, $V_{k+1} \subset V_k \cup \{\mathbf{w}_k\} = Y_k$, supporting it [JC98]. (Here, we drop the superscript for simplicity). Since the GJK algorithm updates the simplices by adding a single vertex at a time, the maximum cardinality of $Y \subset \mathbb{R}^n$ is $n+1$ and, thus, Y specifies the vertices of a m-simplex in $\mathbb{R}^n$. For $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_{m+1}\}$, $\boldsymbol{\nu}$ can be expressed as

$$\boldsymbol{\nu} = \sum_{i=1}^{m+1} \lambda_i \mathbf{v}_i \quad \text{where} \quad \sum_{i=1}^{m+1} \lambda_i = 1 \text{ and } \lambda_i \geq 0. \tag{4.44}$$

The smallest subset $V \subset Y$, such that $\boldsymbol{\nu} \in \text{conv}(V)$ is obtained by discarding all $\mathbf{y}_i$ for which $\lambda_i = 0$, i.e.,

$$V = \{\mathbf{y}_i : \lambda_i > 0\} \tag{4.45}$$

It can be concluded immediately that $\boldsymbol{\nu}$ is also the closest point of the affine hull of V to the origin, i.e., $\boldsymbol{\nu} = \nu(\text{conv}(V)) = \nu(\text{aff}(V))$. Therefore, the set $V = \{\mathbf{y}_i : i \in I_V\}$, where $I_V \subset \{1, \ldots, m+1\}$, must satisfy

$$\boldsymbol{\nu} = \nu(\text{aff}(V)) = \sum_{i \in I_V} \lambda_i \mathbf{y}_i \quad \text{where} \quad \sum_{i \in I_V} \lambda_i = 1 \text{ and } \lambda_i > 0, \tag{4.46}$$

and

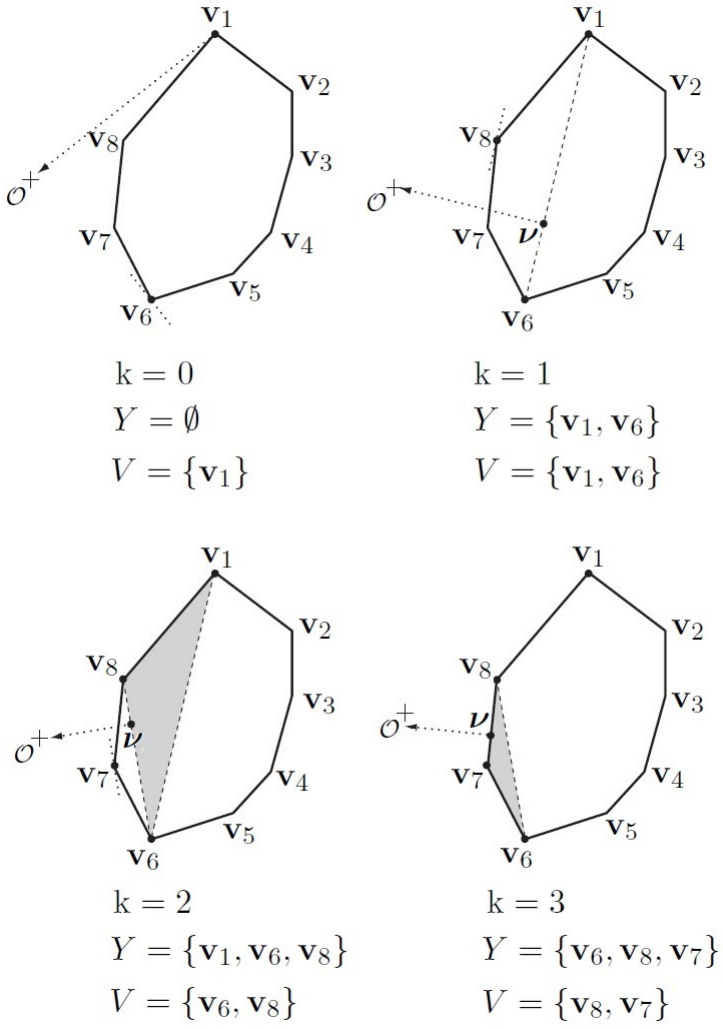


Figure 4.7: Finding the closest point to the origin on a polygon using the GJK algorithm. The set of vertices $Y = V_k \cup \{\mathbf{w}_k\}$ and $V_{k+1} \subset Y_k$ are shown at each iteration. The iterative search descends such that the generated simplex at each iteration offers a better approximation of the $\nu(C)$ that the previous one.

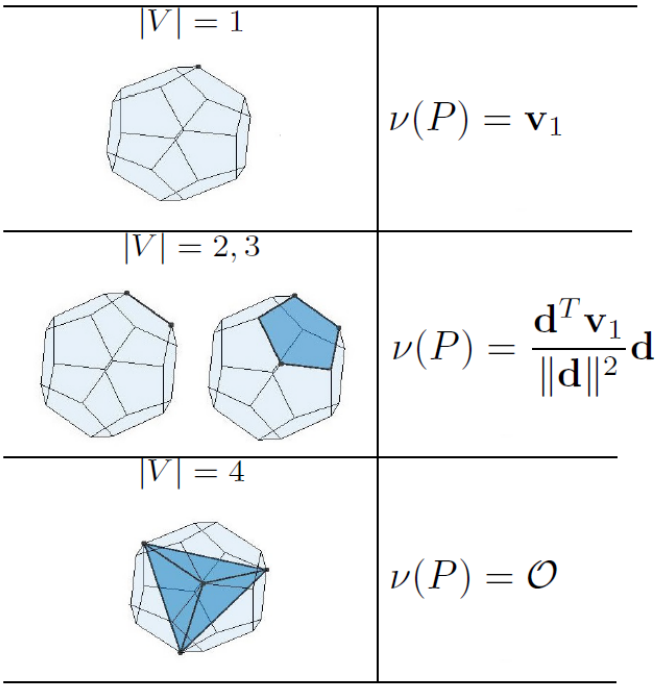


Figure 4.8: Examples of the closest feature of a polyhedron to the origin are shown above. The closest feature can be a vertex, an edge or a face with $|V| = 1, 2, \text{or } 3$, respectively. If $|V| = 4$ the origin is contained in the interior of P. Once the closest feature specified with V is obtained, the closest point, i.e., $\nu(P)$, can be determined as shown above.

$$\nu(\text{aff}(V \cup \{\mathbf{y}_j\})) = \sum_{i \in I_V \cup \{j\}} \lambda_i \mathbf{y}_i \quad \text{where} \quad \sum_{i \in I_V \cup \{j\}} \lambda_i = 1 \text{ and } \lambda_j \leq 0, \forall j \notin I_V. \tag{4.47}$$

The subset V identifies a face of the simplex Y (or its interior). To determine whether a face of Y, $Y_\iota \subset Y, \iota \subset \{1, \ldots, m+1\}$, is the smallest subset supporting $\boldsymbol{\nu}$, it is necessary to compute λ_is representing $\nu(\text{conv}(Y_\iota))$, and check if they comply with (4.46) and (4.47). Assuming that Y_ι is a face of the simplex defined with $Y_\iota = \{\mathbf{y}_i : i \in \iota\} = \{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$, and bearing in mind that the closest point of Y_ι to the origin is perpendicular to aff(Y_ι), then

$$(\mathbf{v}_j - \mathbf{v}_l)^T \nu(\text{conv}(Y_\iota)) = 0, \qquad l \neq j \quad j, l \in \{1, \ldots, r\} \tag{4.48}$$

Therefore, the barycentric coordinates can be computed by solving a system of linear equations that embeds the above orthogonality condition, that is

$$\begin{bmatrix} 1 & \ldots & 1 \\ (\mathbf{v}_2 - \mathbf{v}_1)^T \mathbf{v}_1 & \ldots & (\mathbf{v}_2 - \mathbf{v}_1)^T \mathbf{v}_r \\ \vdots & & \vdots \\ (\mathbf{v}_r - \mathbf{v}_1)^T \mathbf{v}_1 & \ldots & (\mathbf{v}_r - \mathbf{v}_1)^T \mathbf{v}_r \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_r \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.49}$$

A solution to the above algebraic system can be obtained by using Cramer's rule and a cofactor expansion around the first row as,

$$\lambda_j = \frac{-1^{1+j} \det A_{1j}}{\det A} \tag{4.50}$$

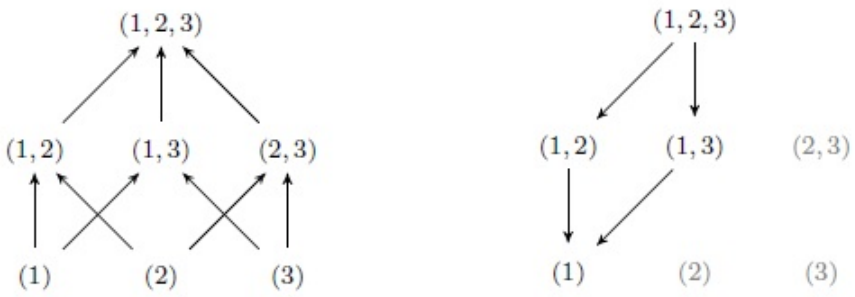


Figure 4.9: The bottom-up approach to searching all $2^{m+1} - 1$ non-empty subsets of Y for a 2-simplex used in Johnson's distance subalgorithm (**left**). The method presented in Alg. 6 to 8 conducts a search only through 2^m subsets whose Voronoi region can possibly contain the origin (**right**).

However, since the cardinality of the smallest subset supporting $\boldsymbol{\nu}$ is not known a priori, the Johnson's distance algorithm solves (4.49) for all the non-empty subsets of Y, and recursively computes $\Delta_i(Y_\iota)$ defined as

$$\Delta_i(Y_\iota) = -1^{(1+j)} \det A_{1j} \tag{4.51}$$

for $\mathbf{v}_j = \mathbf{y}_i, i \in \iota$, to express

$$\nu(\text{aff}(Y_\iota)) = \sum_{i \in \iota} \lambda_i \mathbf{y}_i \quad \text{where } \lambda_i = \frac{\Delta_i(Y_\iota)}{\det A} \tag{4.52}$$

Starting from the $m+1$ singletons, $Y_\iota = \{\mathbf{y}_i\}$, $\Delta_i(Y_\iota)$ is trivially obtained as

$$\Delta_i(\{\mathbf{y}_i\}) = 1 \tag{4.53}$$

whereas for $Y_\iota \cup \{\mathbf{y}_j\}, j \notin \iota$

$$\Delta_j(Y_\iota \cup \{\mathbf{y}_j\}) = \sum_{i \in \iota} \Delta_i(Y_\iota)\big((\mathbf{v}_1 - \mathbf{y}_j)^T \mathbf{y}_i\big). \tag{4.54}$$

Since $\det A = \sum_{i \in \iota} \Delta_i(Y_\iota)$, λ_i is obtained as

$$\lambda_i = \frac{\Delta_i(Y_\iota)}{\sum_{i \in \iota} \Delta_i(Y_\iota)}. \tag{4.55}$$

Since for affinely independent subset, $\det A > 0$, the sign of λ_i can be determined from the sign of $\Delta_i(Y_\iota)$. Thus, if Y_ι satisfies

$$\Delta_i(Y_\iota) > 0 \quad \forall i \in \iota \qquad \text{and} \qquad \Delta_j(Y_\iota \cup \{\mathbf{y}_j\}) \leq 0 \quad \forall j \notin \iota \tag{4.56}$$

then λ_i for Y_ι also complies with (4.46) and (4.47), and, thus, Y_ι is the unique face of the simplex supporting $\boldsymbol{\nu}$, and $V = Y_\iota$. The Johnson's distance algorithm inspects all the $2^{m+1} - 1$ non-empty subsets (all the faces and the interior of the simplex) to find the one that satisfies (4.56). Fig. 4.9 illustrates the bottom-up approach used in the Johnson's distance algorithm for searching through all the non-empty subsets of a 2-simplex.

In practice the GJK algorithm tends to form degenerate simplices with vertices that are close to being affinely dependent. This can affect the numerical stability of the Johnson's algorithm since $\det A$ is close to 0 for such simplices. Therefore, the GJK algorithm may return erroneous results as $\det A$ approaches machine precision. This issue is addressed in [MPB17], where a more robust distance algorithm, named the signed volumes method, is proposed. This method

is based on the fact that λ_is are invariant to affine transformations and, thus, can be computed in a lower dimensional space and be used in the original space of the simplex. Accordingly, the signed volumes method project the vertices into a lower dimensional space to get rid of their affine dependency. The barycentric coordinates are then obtained by solving a simpler well-conditioned system of equations. More details on irregularities in the GJK algorithm and measures to preserve precision in the computations can be found in [Van03].

A. Efficient approach to finding the smallest subset V supporting $\boldsymbol{\nu}$

An alternative way of implementing the distance algorithm is to search through the Voronoi regions of the features of a simplex for the one that contains the origin. This method is based on the fact that $\boldsymbol{\nu} = \nu(Y) \in \text{conv}(V)$ if and only if $\mathcal{O} \in \mathcal{V}_V$. Accordingly, the algorithm considers all the non-empty subsets of Y, and tests their Voronoi regions for containment of the origin using sets of inequalities. The barycentric coordinates and the point of minimum norm can be computed once the feature, specified by the unique subset satisfying the inequalities, is identified.

The algorithm, as explained in [Eri04] and [Van03], search through the subsets one by one, in order of increasing size. For example, for a 3-simplex with $Y = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4\}$, the algorithm inspects a total of 15 subsets corresponding to 4 vertices, 6 edges, 4 faces and the interior of the simplex. However, ordering the vertices of Y such that $\mathbf{y}_1$ is the last added point to Y, i.e., $\mathbf{y}_1 = \mathbf{w}_k$, we can conclude that only the 8 subsets, containing $\mathbf{y}_1$ as an element, can possibly satisfy the containment test. This is immediately followed from the fact that $\mathbf{y}_1$ is the support point obtained from searching in the direction of $-\nu(\{\mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4\})$. That being the case, we can immediately invalidate 7 features, and test the remaining features for containment using inequality sets specific to each one.

Fig. 4.10 shows the list of 2^{m} Voronoi regions of a m-simplex that can possibly contain the origin.

Defining the vector $\mathbf{n}_{\mathbf{y_c y_b y_a}}$ prependicular to the face specified with $\{\mathbf{y}_a, \mathbf{y}_b \mathbf{y}_c\}$ as

$$\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} = (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1) \tag{4.57}$$

and bearing in mind that,

$$(\mathbf{y}_1^T \mathbf{n}_{\mathbf{y}_4\mathbf{y}_3\mathbf{y}_2})(\mathbf{y}_i^T \mathbf{n}_{\mathbf{y}_4\mathbf{y}_3\mathbf{y}_2}) < 0 \quad i \neq 1 \tag{4.58}$$

the set of inequalities for determining whether the interior of a 3-simplex contains the origin is obtained as

$$\begin{aligned} (\mathbf{y}_1^T \mathbf{n}_{\mathbf{y}_3\mathbf{y}_2\mathbf{y}_1})(\mathbf{y}_4^T \mathbf{n}_{\mathbf{y}_3\mathbf{y}_2\mathbf{y}_1}) < 0 \\ (\mathbf{y}_1^T \mathbf{n}_{\mathbf{y}_4\mathbf{y}_3\mathbf{y}_1})(\mathbf{y}_2^T \mathbf{n}_{\mathbf{y}_4\mathbf{y}_3\mathbf{y}_1}) < 0 \\ (\mathbf{y}_1^T \mathbf{n}_{\mathbf{y}_4\mathbf{y}_2\mathbf{y}_1})(\mathbf{y}_3^T \mathbf{n}_{\mathbf{y}_4\mathbf{y}_2\mathbf{y}_1}) < 0 \end{aligned} \tag{4.59}$$

The above set simply considers the 3 faces passing through $\mathbf{y}_1$ and checks if the origin is between the plane supporting the face and a parallel plane passing through the vertex not in the face. Therefore, if (4.59) holds then $\mathcal{O} \in \mathcal{V}_{(1,2,3,4)}$.

The set of inequalities for testing the Voronoi region associated with $\mathbf{y}_1$ can be defined as

$$\begin{aligned} \big((\mathcal{O} - \mathbf{v}_1)^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}\big)\big((\mathbf{v}_4 - \mathbf{v}_1)^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}\big) \leq 0 \\ (\mathcal{O} - \mathbf{v}_1)^T \big(\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} \times (\mathbf{v}_2 - \mathbf{v}_1)\big) \leq 0 \\ (\mathcal{O} - \mathbf{v}_1)^T (\mathbf{v}_2 - \mathbf{v}_1) \leq 0 \end{aligned} \tag{4.60}$$

If the above inequalities are satisfied, then the origin lies in $\mathcal{V}_{(1)}$, and $V = \{\mathbf{y}_1\}$. Analogous sets of inequalities can be obtained for testing the Voronoi regions of the edges and faces of the simplex. It should be noted that most of the computed quantities in the resulting set of inequalities for testing different Voronoi regions are the same and need not be recomputed. This allows an efficient implementation of the test as summarized in S1D, S2D, or S3D subroutines, presented in Alg. 6 to 8, for $|Y| = 1, 2,$ or 3, respectively.

The presented case-based approach to identifying the smallest subset $V \subset Y$ such that $\boldsymbol{\nu} \in \text{conv}(V)$, differs from the Johnson's distance algorithm in the sense that, instead of computing λ_i (or Δ_i) for each of the $2^{m+1} - 1$ non-empty subsets of Y and checking whether they satisfy (4.46) and (4.47), it identifies the unique set of vertices $\mathbf{y}_i$ that satisfy (4.46) and (4.47) by searching through 2^m subsets, and only then computes λ_i or $\boldsymbol{\nu}$ if necessary. However, as we explained before we do not compute $\boldsymbol{\nu}$ until the very end, and thus Alg. 6 to 8 only return a search direction which is almost readily available from the previously computed quantities. It can be observed from Alg. 6 to 8 that the search direction is simply a vector pointing from $\text{conv}(V_{k+1})$ to the origin.

Algorithm 4 The numerical GJK distance algorithm (adapted from [Van03])

$\mathbf{v}$ = "Arbitrary point in $C = A - B$"
$\boldsymbol{\nu} = -\mathbf{v}$
$V = \emptyset$
$Y = \emptyset$
repeat
  $\mathbf{w} = s_C(-\boldsymbol{\nu})$;
  if $\mathbf{w} \in Y$ or $\|\boldsymbol{\nu}\|^2 - \boldsymbol{\nu}^T\mathbf{w} \leq \epsilon_{\text{rel}}^2\|\boldsymbol{\nu}\|^2$ **then**
    $\boldsymbol{\nu}$ is close enough to $\nu(C)$
    return $\boldsymbol{\nu}$
  end if
  $Y \leftarrow V \cup \mathbf{w}$;
  $[V, \lambda] \leftarrow$ DistanceSubalgorithm(Y);
  $\boldsymbol{\nu} = \sum \lambda_i \mathbf{y}_i : \mathbf{y}_i \in V, i \in I_V$
until $|V| = 4$ or $\|\boldsymbol{\nu}\|^2 \leq \epsilon_{\text{tol}} \max\{\|\mathbf{y}_i\|^2; \mathbf{y}_i \in V\}$;
return $\boldsymbol{\nu} = \mathcal{O}$

Algorithm 5 GJK for polygons

$\mathbf{v}$ = "Arbitrary point in vert(P)"
$\mathbf{d} = -\mathbf{v}$
$V = \{\mathbf{v}\}$
$Y = \emptyset$
repeat
  $\mathbf{w} = s_P(\mathbf{d})$;
  if $\mathbf{d}^T(\mathbf{w} - \mathbf{v}_1) \leq 0$ **then**
    V represents the closest face of P to $\mathcal{O}$
    return $\nu(V)$
  end if
  $Y \leftarrow V \cup \mathbf{w}$;
  $[V, \mathbf{d}] \leftarrow$ CallSmD(Y);
until $|V| = 4$;
P contains $\mathcal{O}$
return $\boldsymbol{\nu} = \mathcal{O}$

Algorithm 6 Sub-routine for $|Y| = 2$

function S1D($\{\mathbf{v}_2, \mathbf{v}_1\}$) [1]
  if $\mathbf{v}_1^T \mathbf{v}_{12} \geq 0$ **then**
    $V \leftarrow \{\mathbf{v}_1\}$
    $\mathbf{d} \leftarrow -\mathbf{v}_1$
  else
    $V \leftarrow \{\mathbf{v}_2, \mathbf{v}_1\}$
    $\mathbf{d} \leftarrow -\mathbf{v}_{12} \times \mathbf{v}_1 \times \mathbf{v}_{12}$
  end if
end function

Algorithm 7 Sub-routine for $|Y| = 3$

function S2D ($\{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$)
  $\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} = \mathbf{v}_{12} \times \mathbf{v}_{13}$
  if $\mathbf{v}_1^T(\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} \times \mathbf{v}_{12}) \geq 0$ **then**
    $[V, \mathbf{d}] \leftarrow$ S1D($\{\mathbf{v}_2, \mathbf{v}_1\}$)
  else
    if $\mathbf{v_1}^T(\mathbf{v}_{13} \times \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}) \geq 0$ **then**
      $[V, \mathbf{d}] \leftarrow$ S1D($\{\mathbf{v}_3, \mathbf{v}_1\}$)
    else
      if $\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} \geq 0$ **then**
        $V \leftarrow \{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$
        $\mathbf{d} \leftarrow -\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}$
      else
        $V \leftarrow \{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$
        $\mathbf{d} \leftarrow \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}$
      end if
    end if
  end if
end function

4.2.3 Continuity conditions

As explained before, at each sampling time, a trajectory, expressed as a parametric Bézier curve, is generated for the time horizon $[t_k, t_k + t_h]$, and the trajectory for the entire flight time is formed by joining segments of these Bézier curves end-to-end. The smoothness of the resulting composite trajectory must be guaranteed by enforcing continuity at the joining points of two consecutive segments up to a certain derivative. Assuming that the time horizon is equal to $\Delta t = t_k - t_{k-1}$, the trajectory generated for the i-th vehicle at t_k must satisfy

$$\frac{d^r \mathbf{p}_{i,k}(0)}{dt^r} = \frac{d^r \mathbf{p}_{i,k-1}(1)}{dt^r} \quad r \in \{0, \ldots, \mathrm{r}\} \tag{4.61}$$

for a desired r. The first and second-order parameteric continuity conditions are given in (3.78) and (3.79), respectively. Similar conditions can be derived for higher-order continuity between two Bézier curves. It should be noted that in practice the time horizon is usually greater than Δt, in which case a Bézier curve describing the segment over the time interval $[t_{k-1}, t_k]$ can be obtained by subdividing $\mathbf{p}_{k-1}(.)$ at t_k with the de Casteljau's algorithm.

Algorithm 8 Sub-routine for $|Y| = 4$

function S3D($\{\mathbf{v}_4, \mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$)
  $\mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1} = \mathbf{v}_{12} \times \mathbf{v}_{13}$
  if $(\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1})(\mathbf{v}_{14}^T \mathbf{n}_{\mathbf{v}_3\mathbf{v}_2\mathbf{v}_1}) \geq 0$ **then**
    $[V, \mathbf{d}] \leftarrow$ S2D($\{\mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$)
  else
    $\mathbf{n}_{\mathbf{v}_4\mathbf{v}_3\mathbf{v}_1} = \mathbf{v}_{13} \times \mathbf{v}_{14}$
    if $(\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_3\mathbf{v}_1})(\mathbf{v}_{12}^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_3\mathbf{v}_1}) \geq 0$ **then**
      $[V, \mathbf{d}] \leftarrow$ S2D($\{\mathbf{v}_4, \mathbf{v}_3, \mathbf{v}_1\}$)
    else
      $\mathbf{n}_{\mathbf{v}_4\mathbf{v}_2\mathbf{v}_1} = \mathbf{v}_{12} \times \mathbf{v}_{14}$
      if $(\mathbf{v}_1^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_2\mathbf{v}_1})(\mathbf{v}_{13}^T \mathbf{n}_{\mathbf{v}_4\mathbf{v}_2\mathbf{v}_1}) \geq 0$ **then**
        $[V, \mathbf{d}] \leftarrow$ S2D($\{\mathbf{v}_4, \mathbf{v}_2, \mathbf{v}_1\}$)
      else
        $V \leftarrow \{\mathbf{v}_4, \mathbf{v}_3, \mathbf{v}_2, \mathbf{v}_1\}$
      end if
    end if
  end if
end function

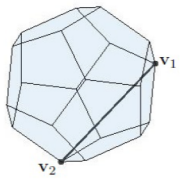
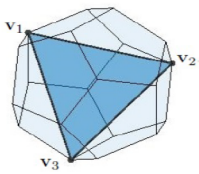
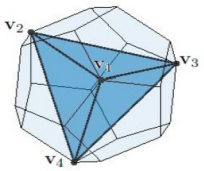
1-simplex 	$\mathcal{V}_{(1)}$ $\mathcal{V}_{(1,2)}$
2-simplex 	$\mathcal{V}_{(1)}$ $\mathcal{V}_{(1,2)}, \mathcal{V}_{(1,3)}$ $\mathcal{V}_{(1,2,3)}$
3-simplex 	$\mathcal{V}_{(1)}$ $\mathcal{V}_{(1,2)}, \mathcal{V}_{(1,3)}, \mathcal{V}_{(1,4)}$ $\mathcal{V}_{(1,2,3)}, \mathcal{V}_{(1,2,4)}, \mathcal{V}_{(1,3,4)}$ $\mathcal{V}_{(1,2,3,4)}$

Figure 4.10: A m-simplex is linked to $2^{m+1} - 1$ Voronoi regions associated to its vertices, edges, faces, and volume. The list of 2^m Voronoi regions that can possibly contain the origin is given in this table. It should be noted that $\mathbf{v}_1$ is the latest vertex added to V.

4.3 Simulation Results

In this section, the efficacy of the proposed method for generating feasible and collision-free trajectories in (vehicle-) dense environments are assessed through different simulation examples.

We compare the resulting trajectories to those generated with the well-studied BVC approach. We specifically test the capability of the two methods to generate trajectories that ensure all drones involved in a simulation example reach their final positions, and compare the flight time, obtained with each of them, to complete point-to-point transition missions. We also present the recorded computation time for executing the proposed algorithm in this paper to emphasize its suitability for real-time applications.

In the simulations presented below, we assume all drones have the same size, and their BVC (4.12) is defined with the safety radius $r_{\mathbf{D}} = 0.30$ m. To specify the set (4.18), we approximate the drone body with an oblate spheroid with $\Lambda = \text{diag}([0.30\text{ m}, 0.30\text{ m}, 0.11\text{ m}])$. In both methods, trajectories are parameterized with Bézier curves. Upper and lower bounds on the speed and acceleration are assumed to be $\pm 2.3\,\frac{\text{m}}{\text{s}}$ and $\pm 7.1\,\frac{\text{m}}{\text{s}^2}$ respectively. At each replanning step, the planner finds the closest point in the updated Voronoi cell to the goal position using the algorithm in Section 4.2.2. The computed point is then used to define the terminal cost term. The first term of the cost function in all subproblems is defined as $\int_0^1 \|\mathbf{p}_{i,k}^{(4)}(\tau)\|^2 d\tau$. The time horizon and the replanning step are also considered to be the same for both methods. The obtained solution at the previous replanning step is used to set the initial guess for the current sub-problem. We use FORCES Pro [DJ19] to generate solvers for the resulting sub-problems. The sub-problems, involving the set of control points $\bar{p}_{i,k}$ as decision variables, can be reformulated to match the supported classes of problem in FORCES Pro. Here, all computations are executed on a single desktop computer, with 2.60 GHz i7-4510U CPU and 6.00 GB RAM; however, in practice, the resulting independent sub-problems can be solved in parallel.

As mentioned before in the paper, in Voronoi-based methods, a vehicle only requires the position information from its neighboring vehicles to generate its trajectory. Therefore, they are more suitable for implementation when vehicles have limited communication capability, and have to rely solely on onboard sensing. In reality, the position sensor noise can impact the planner performance, yet this is more pronounced when estimating other information, such as velocity, from noise-corrupted measurements is needed. Therefore, Voronoi-based planners are more robust when there is no communication network. Nevertheless, in the following simulations, we assume that accurate position information is available with no delay at the replanning time.

In the first example, we consider five drones flying from their initial positions to given final positions. This example is similar to one in [Zho+17] where a random offset is added to break the symmetry in the drones' initial and final configurations. Figure 4.11 (right) shows collision-free trajectories generated with the distributed scheme described above, with a replanning rate of 20 Hz. For this particular example, the resulting trajectories match those generated with BVC with a flight time of 11.6782 s. Figure 4.11 (left) shows collision-free trajectories obtained from the centralized solution, which delivers a total flight time of 9.4347 s, yet, while the central solution is obtained in 601 ms, the average computation time for solving the sub-problems in the decentralized scheme is only 49 ms.

In the next example, we consider 18 drones switching positions in a 3 m $\times$ 5 m $\times$ 2 m space, with a maximum speed and acceleration of $\pm 4.7\,\frac{\text{m}}{\text{s}}$ and $\pm 9.8\,\frac{\text{m}}{\text{s}^2}$, respectively. Figure 4.12a shows the initial and final configurations, and Figure 4.12b displays collision-free trajectories generated with the proposed distributed scheme in the paper implemented at 10 Hz. While both methods could find collision-free trajectories for guiding the team of drones from their initial positions to their goal positions, the flight time achieved with the proposed method is markedly shorter than the time obtained with BVC. We also performed a trial simulation with 34 drones in a similar configuration. Table 4.1 compares the success rate and the flight time to complete the transition using BVC and the proposed method.

In the third example, we consider 100 drones flying in an 8 m $\times$ 8 m $\times$ 3.5 m space. The initial and final positions for the drones are displayed with dot and square markers in Figure 4.13. We test both methods in 30 different trials. In each trial, final positions are randomly assigned to drones. A trial is considered successful if all drones could reach their final positions within the

stipulated time. The proposed method with 23 successful trials and an average total flight of $1s$ outperforms the BVC with only 16 completed trials. It should be noted that using well-devised deadlock prevention strategies or loosening time constraints can improve the success rate of both methods. Figure 4.13 shows collision-free trajectories generated with the proposed method for one of the trials at different time steps. The average computation time for solving sub-problems in this example was around 115 milliseconds. In addition, compared to the geometric algorithm in [Zho+17], the closest point in a Voronoi cell to the goal position was obtained at least 10 times faster with the proposed algorithm in Section 4.2.2. The computation time for finding the closest point, and solving the optimization problem, depends on the number of neighboring drones (See Table 4.2 for recorded computation times in simulation examples with 18, 34, and 100 drones). In most applications, with typical Voronoi diagrams, the number of boundary planes, i.e., the number of Voronoi neighbors, is small. Thus, the proposed distributed algorithm is scalable to arbitrary numbers of vehicles.

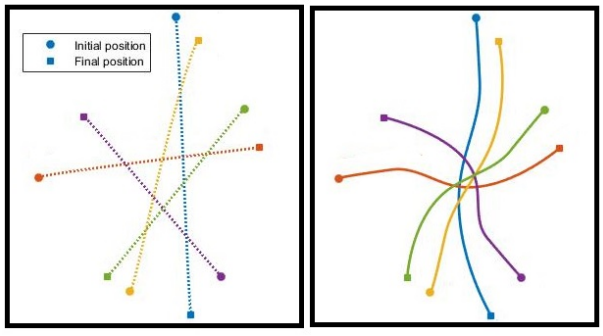
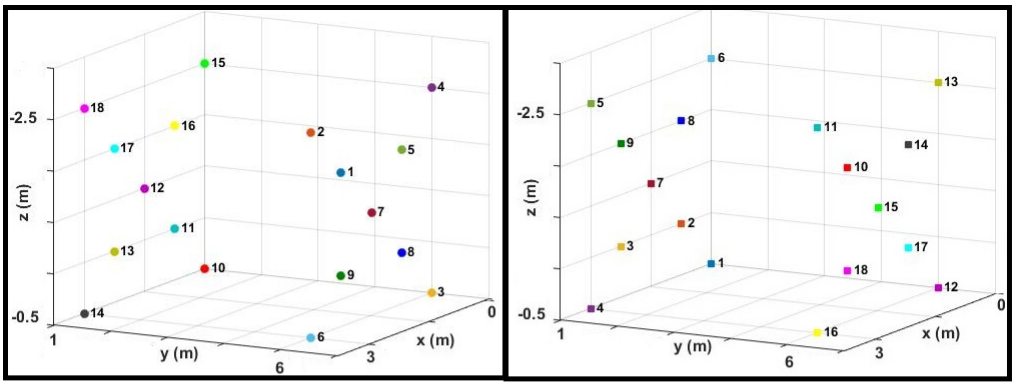
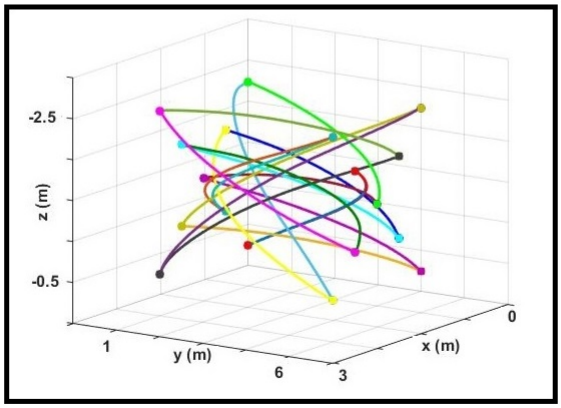


Figure 4.11: Comparing collision-free trajectories generated with the centralized solution (**left**) and the proposed decentralized approach (**right**) for five drones flying from their initial positions to given final positions. While the central solution yields a shorter flight time, its computation time is significantly longer than average time required to solve the sub-problems in the distributed method.



(a) Initial (left) and final (right) configurations



(b) Collision-free trajectories

Figure 4.12: (**a**) Initial (left) and final (right) position configurations for 18 drones. Each drone is assigned a unique color and a number next to it. (**b**) Collision−free trajectories for 18 drones switching their positions in a 3 m × 5 m × 2 m space. The total flight time for the drones to reach their final positions is 5.1 s using the proposed method, which is shorter than the 6.3 s flight time obtained with the BVC.

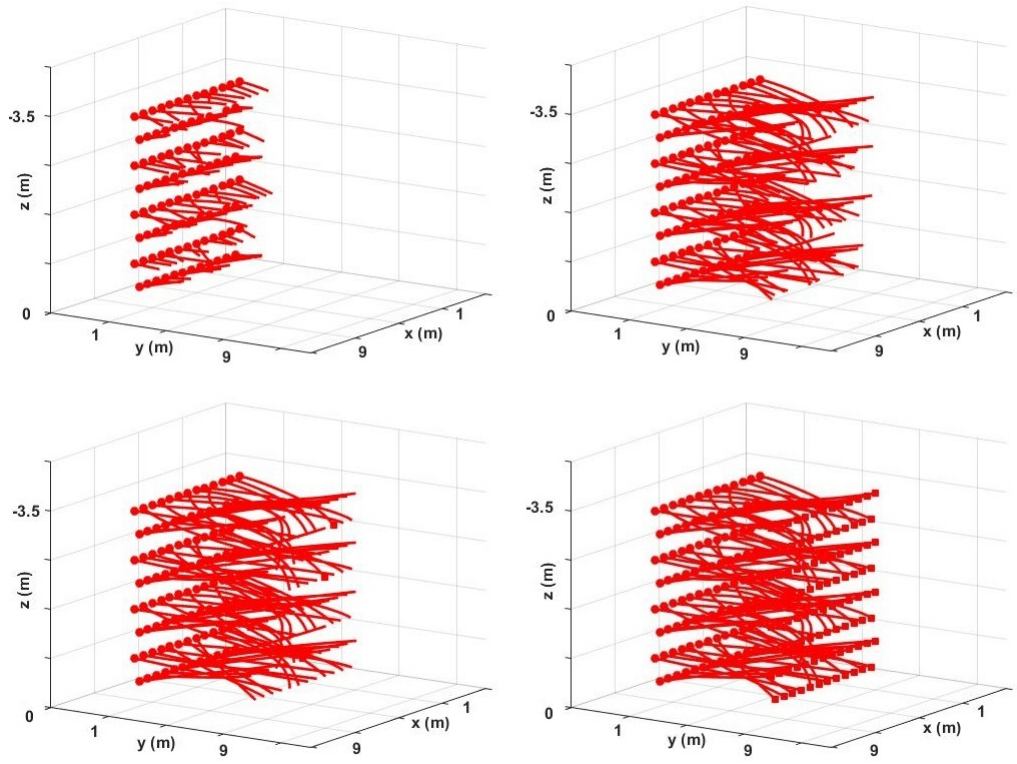


Figure 4.13: Collision−free trajectories for 100 drones flying from their initial positions (dots) to randomly specified final positions (squares) at different replanning steps.

Table 4.1: Comparing the number of successful trials and the average flight time achieved with the BVC and the proposed method in the paper.

Number of Drones	**BVC**		**Proposed Method**	
	Flight Time	**Completed Trials**	**Flight Time**	**Completed Trials**
18	6.812 s	5/5	5.327 s	5/5
34	8.105 s	7/10	6.625 s	8/10
100	14.573 s	16/30	11.462 s	23/30

Table 4.2: Recorded computation times for finding the closest point in a Voronoi cell to the goal position and solving the optimization problem in simulation examples with 18, 34, and 100 drones.

Number of Drones	**Computation Time (ms)**	
	Finding the Closest Point	**Solving the Sub-Problem**
18	<0.1	77.562
34	<0.1	98.330
100	0.171	121.633

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we developed a Bézier curve based trajectory generation method for multiple autonomous vehicles, motivated by practical problems that arise in emerging applications of networked aerial vehicles. We proposed an efficient method for evaluating collision-avoidance constraints, and demonstrated its suitability for real-time applications with strict time constraints. We also addressed the shortcomings of the existing trajectory generation methods to consider the rotational motion of a drone and derived collision avoidance constraints taking into account the drone's orientation along a trajectory. We also developed a distributed trajectory generation framework, with low computation and communication demands, which allows for real-time re-planning of trajectories for large groups of drones flying in a common workspace. In this chapter we summarize the contributions of the thesis and provide suggestions for future research work.

5.1.1 Summary

In Chapter 2 we formulated the optimal trajectory generation problem for autonomous drone cinematography and AUV range-based positioning, and employed the direct multiple shooting method in conjunction with a structure-exploiting solver to efficiently solve the resulting NLPs. The simulation results presented in Chapter 2 verified the applicability of numerical optimal control approaches to generate trajectories in multi-vehicle missions with complex constraints and objectives.

In Chapter 3 we leveraged the unique properties of Bézier curves and developed a computationally efficient trajectory generation framework for differentially flat systems. We addressed the issue of evaluating inequality constraints in the semi-infinite optimization problem associated with polynomial path parameterization. Such problems involve constraints that bound functions of a finite number of control points over an entire time interval. In order to obtain a computationally tractable optimization problem, we replaced inequality constraints, whose functions are represented as Bernstein polynomials, with constraints on their control points, yielding a finite-dimensional problem with a small number of constraints. We also proposed a method to reduce the conservatism in the resulting set of constraints, that might be caused by the gap between a Bernstein polynomial and its control polygon. The proposed method is based on the de Casteljau's algorithm to find a more refined Bézier control polygon of the Bernstein polynomial with repeated subdivision. We showed that the proposed method would allow refining the representation of the polynomial locally with control points that are arbitrarily close to the curve. We also presented quantitative bounds on the distance between a Bernstein polynomial and its Bézier control polygon and used the result to derive an appropriate criterion for deciding whether the composite control polygon obtained with repeated subdivision is a good approximant of the Bernstein polynomial. We compared the proposed method to other approaches for evaluating inequality constraints and showed that it can effectively speed up the solution time of generating and re-planning trajectories.

We also addressed the motion planning problem for drones flying in small spaces, where finding feasible trajectories is impossible unless the flight attitude angles are taken into account. In order to include the attitude angles in the trajectory generation problem, we modeled the drone body with an ellipsoid and utilized the separating hyperplane theorem to derive collision avoidance constraints between two ellipsoids. We showed that incorporating the resulting set of constraints would enable the planner to generate trajectories that can safely navigate a drone through narrow gaps, or maneuver multiple drones in a tight space without collisions. We also showed that the resulting constraints can be expressed as inequalities in Bézier form and thus can be integrated seamlessly into the proposed Bézier curve-based trajectory generation framework.

In Chapter 4, we presented a distributed algorithm to generate collision-free trajectories for a group of quadrotors flying through a common workspace. In the proposed setup, each vehicle

would replan its trajectory, in a receding horizon manner, by solving a small-scale optimization problem that only involves its own individual variables. We adopted the Voronoi partitioning of space to derive local constraints that guarantee collision avoidance with all neighbors for a certain time horizon. The collision avoidance constraints were obtained taking into account the orientation to avoid infeasiblity issues caused by ignoring the quadrotor's rotational motion. It was shown that the resulting set of constraints can be expressed as Bézier curves, and thus can be evaluated efficiently, without discretization, using the method proposed in Chapter 3. This would ensure that collision avoidance requirements are satisfied at any time instant, even for an extended planning horizon. It was shown through extensive simulations, with up to 100 drones, that the proposed method has a higher success rate at finding collision-free trajectories for large groups of drones compared to the widely used BVC method.

We also presented an efficient approach to implementing the GJK distance algorithm for polytopes, which is used to compute the closest point of the vehicle's Voronoi cell to the vehicle's goal position prior to solving the optimization problem at each re-planning step. The presented algorithm is different from the original GJK algorithm in the sense that: (i) it does not rely on the computation of the point of minimum norm at each iteration, instead it uses an easily computable search direction to find new support points and to check a modified termination condition; (ii) it does not inspect all Voronoi regions to evolve simplices, alternatively it searches through the least number of regions by discarding those that cannot contain the origin. Once the closest feature of the polytope to the origin is determined, the point of minimum norm is obtained by projecting the origin onto the feature. It was shown that the proposed approach can effectively reduce the computational delays between updating the vehicles' Voronoi diagram and re-planning trajectories.

5.2 Future Work

5.2.1 Rational Bézier curves

In Chapter 3, we proposed an efficient method for evaluating an inequality constraint provided that its function can be expressed as a scalar-valued Bézier curve. Extending the proposed method to constraints with functions in the form of rational Bézier curves can be an interesting topic for future research work. This would require new shape control tools that are more efficient than the classical way of manipulating rational Bézier curves.

A rational Bézier curve of degree n is defined with a set of $n+1$ control points $P_0, \ldots, P_n$ as

$$\mathrm{r}(\tau) = \sum_{i=0}^{n} R_{i,n}(\tau) P_i \tag{5.1}$$

where $R_{i,n}(\tau)$ is defined as

$$R_{i,n}(\tau) = \frac{w_i B_{i,n}(\tau)}{\sum_{j=0}^{n} w_j B_{j,n}(\tau)} \tag{5.2}$$

with the weights $w_i > 0$. The numerator and denominator of $\mathrm{r}(\tau)$ behave like ordinary Bézier curves, and thus the de Casteljau's algorithm can be applied to each separately for the evaluation of $\mathrm{r}(\tau)$. However, this approach has a time complexity of $\mathbf{O}(n^2)$. More efficient de Casteljau-type evaluation algorithms for rational Bézier curve can be found in [Far83], [SJ15] and [WC20]. Alternatively, a rational Bézier curve can be expressed in barycentric form over a set of distinct nodes $\tau_0, \ldots, \tau_n$ as [RH21],

$$\mathrm{r}(\tau) = \frac{\sum_{i=0}^{n} (-1)^i \frac{\beta_i}{t - t_i} Q_i}{\sum_{i=0}^{n} (-1)^i \frac{\beta_i}{t - t_i}} \tag{5.3}$$

where $Q_i = \mathrm{r}(\tau_i)$ are the interpolation points and β_i are non-zero weights defined as

$$\beta_i = (-1)^{n+i}\alpha_i z(\tau_i) \tag{5.4}$$

with the Lagrange weights $\alpha_i = \prod_{j=0, j\neq i}^{n} \frac{1}{\tau_i - \tau_j}, i = 0, \ldots, n$ and $z(\tau) = \sum_{i=0}^{n} w_i B_{i,n}(\tau)$.

Representing a rational Bézier curve as above allows direct control over the shape of the curve, i.e., the curve can be enforced to pass through some desired points. The curve flatness can also be adjusted by modifying the weights β_i. Furthermore, degree elevation can be performed simply and efficiently by adding new interpolation points $Q_j = \mathrm{r}(\tau_j), \tau_j \in [0,1]\backslash\{\tau_0, \ldots, \tau_n\}$ and adapting the weights accordingly [RH21]. Therefore, to benefit from the simplicity of the barycentric form, while making use of the properties of the Bézier form, it may be desirable to convert a curve from Bézier form to barycentric form for editing its control points and weights as required by the trajectory generation problem.

5.2.2 Computation delay compensation for real-time implementation

In chapter 4, we introduced a distributed algorithm for generating collision-free trajectories for multiple drones, taking into account their orientation. In order to avoid substantial communication between drones, we adopted Voronoi-based space partitioning and derived local constraints that guarantee collision avoidance with neighboring vehicles for an entire time horizon. As we explained, at each re-planning step, upon receiving (or sensing) the new position information, a vehicle would find the closest point in its Voronoi cell to the goal position, and solve an optimization problem, using as the initial condition the current state of the vehicle, to generate its trajectory for the time horizon. However, it should be noted that the computation time to find the optimal solution can lead to a (significant) delay between updating the position information and executing the trajectory. Therefore, in practice, the computational delay must be explicitly considered to avoid performance degradation (or even failure) of the planner in real-time applications.

This issue can be addressed by setting the initial condition in the optimization sub-problem solved at the time instant t_k as

$$\mathbf{x}_{i,k}(0) = \mathbf{x}_{i,k-1}(t_k + \delta t) \tag{5.5}$$

where δt is an estimation of the computation time required to find the closest point and solve the optimization problem. Assuming that the time for computing the closest point is negligible, it is imperative to re-plan the trajectory in an allocated time $< \delta t$. If the solver cannot find the optimal solution in less than δt, the best feasible solution is executed. That being the case, it is of utmost importance to use a solver that can guarantee intermediate solutions are feasible.

Appendix A

Structure exploiting NLP solver

The NLP resulting from the direct multiple shooting method can be solved efficiently by exploiting its special structure. In the following, we briefly outline the block Cholesky procedure which was first developed in [WB09] for block structured QPs and later extended to nonlinear non-convex problems in [Dom+12] and [Zan+17].

A general NLP formulation is given by

$$\min_{\mathbf{z}} F(\mathbf{z}) \tag{A.1}$$
$$\text{s.t. } G(\mathbf{z}) = 0, \tag{A.1a}$$
$$H(\mathbf{z}) \leq 0. \tag{A.1b}$$

where $G : \mathbb{R}^{n_\mathbf{z}} \to \mathbb{R}^{n_{\text{eq}}}$ and $H : \mathbb{R}^{n_\mathbf{z}} \to \mathbb{R}^{n_{\text{ineq}}}$. The Karush-Kuhn-Tucker (KKT) optimality conditions for the above NLP are

$$\begin{aligned} \nabla_\mathbf{z} \mathcal{L}(\mathbf{z}^*, \mu^*, \lambda^*) &= 0 \\ G(\mathbf{z}^*) &= 0, \\ H(\mathbf{z}^*) &\leq 0, \\ \lambda^* &\geq 0 \\ H(\mathbf{z}^*)^T \lambda^* &= 0. \end{aligned} \tag{A.2}$$

where $\mathcal{L}(.)$ is the Lagrangian defined as

$$\mathcal{L}(\mathbf{z}, \lambda, \mu) = F(\mathbf{z}) + G(\mathbf{z})^T \mu + H(\mathbf{z})^T \lambda, \tag{A.3}$$

and $\mu \in \mathbb{R}^{n_{\text{eq}}}$ and $\lambda \in \mathbb{R}^{n_{\text{ineq}}}$ are the Lagrange multipliers associated with equality and inequality constraints respectively. One approach to solving the NLP (A.1) is via interior point (IP) methods which replace the non-smooth conditions in (A.2) by smoothed version, expressed as

$$\begin{aligned} \nabla_\mathbf{z} \mathcal{L}(\mathbf{z}^*, \mu^*, \lambda^*) &= 0, \\ G(\mathbf{z}^*) &= 0, \\ H(\mathbf{z}^*) + \mathbf{s} &= 0, \\ \Lambda S &= 0, \end{aligned} \tag{A.4}$$

where $\mathbf{s} \in \mathbb{R}^{n_{\text{ineq}}}$ is the vector of slack variables, $S = \text{diag}(\mathbf{s})$ and $\Lambda = \text{diag}(\lambda)$. The conditions in (A.4) can be summarized as

$$R(\mathbf{w}) = 0. \tag{A.5}$$

where $\mathbf{w} = (\mathbf{z}, \mu, \lambda, \mathbf{s})^T$. Newton's method can be applied to solve the above system of equations by linearizing around the current guess $\mathbf{w}^j = (\mathbf{z}^j, \mu, \lambda, \mathbf{s})^T$ and generating a sequence of points by solving

$$R(\mathbf{w}^j) + \nabla R(\mathbf{w}^j)(\Delta \mathbf{w}) = 0 \tag{A.6}$$

where $\Delta \mathbf{w} = \mathbf{w}^{j+1} - \mathbf{w}^j$ and

$$\nabla R = \begin{bmatrix} \mathcal{H} & J_{\text{eq}}^T & J_{\text{ineq}}^T & 0 \\ J_{\text{eq}}(\mathbf{z}) & & & \\ J_{\text{ineq}}(\mathbf{z}) & & & I \\ & & S & \Lambda \end{bmatrix} \tag{A.7}$$

In (A.7), J_{eq} and J_{ineq} denote respectively the Jacobian matrices of the equality constraints and inequality constraints, and $\mathcal{H}$ is obtained as

$$\mathcal{H} = \nabla_{\mathbf{z}}^2 F + \sum_{i=1}^{n_{\text{eq}}} \boldsymbol{\mu}_i^T \nabla_{\mathbf{z}}^2 G_i + \sum_{i=1}^{n_{\text{ineq}}} \boldsymbol{\lambda}_i^T \nabla_{\mathbf{z}}^2 H_i \tag{A.8}$$

The above linear system can be reduce by eliminating $\Delta \mathbf{s}$ and the Lagrange multipliers $\Delta \lambda$ using

$$\Delta \mathbf{s} = \Lambda^{-1}(-r_{\mathbf{s}} - S\Delta\lambda), \tag{A.9}$$

and

$$\Delta \lambda = S^{-1}\Lambda(r_I + J_{\mathbf{ineq}}(\mathbf{z})\Delta \mathbf{z}) - S^{-1} r_{\mathbf{s}}, \tag{A.10}$$

where $r_{\mathbf{s}} = \Lambda S$ and $r_I = H(\mathbf{z}) + \mathbf{s}$. The resulting augmented system can be expressed as

$$\begin{bmatrix} \Phi & J_{\text{eq}}(\mathbf{z})^T \\ J_{\text{eq}}(\mathbf{z}) & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{z} \\ \Delta \mu \end{bmatrix} = - \begin{bmatrix} r_d \\ r_E \end{bmatrix}, \tag{A.11}$$

where

$$\begin{aligned} \Phi &= \mathcal{H}(\mathbf{z}, \mu, \lambda) + J_{\text{ineq}}(\mathbf{z})^T S^{-1} \Lambda J_{\text{ineq}}(\mathbf{z}), \\ r_d &= \nabla_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \mu, \lambda) + J_{\text{ineq}}(\mathbf{z})^T S^{-1} r_{\mathbf{s}} - J_{\text{ineq}}(\mathbf{z})^T S^{-1} \Lambda r_I, \\ r_E &= G(\mathbf{z}). \end{aligned} \tag{A.12}$$

The above form of the KKT system is called the symmetric indefinite form [WB06]. By forming the Schur complement of the matrix in (A.11), a more compact and symmetric system can be obtained as [Dom+12]

$$Y \Delta \mu = \beta. \tag{A.13}$$

where

$$\begin{aligned} Y &= J_{\text{eq}}(\mathbf{z}) \Phi^{-1} J_{\text{eq}}(\mathbf{z})^T, \\ \beta &= r_E - J_{\text{eq}}(\mathbf{z}) \Phi^{-1} r_d. \end{aligned} \tag{A.14}$$

Once (A.13) is solved the search direction can be computed as

$$\Delta \mathbf{z} = \Phi^{-1}(-r_d - J_{\text{eq}}(\mathbf{z})^T \Delta \mu). \tag{A.15}$$

Solving the KKT system and finding the search direction is the most computationally expensive step in the interior point method. For a generic NLP, the system of linear equations (A.13) (or (A.11)) is solved by first computing the LDL^T factorization of the matrix Y. However, for the NLP obtained via direct multiple shooting discretization, Y can be factored efficiently by block-wise Cholesky factorization. This is due to the fact that the direct multiple shooting method yields a block diagonal Hessian matrix and a block banded Jacobian matrix (See sec. 2.3.2), and thus, the resulting augmented Hessian matrix Φ and its inverse exhibit block diagonal structure. Consequently, the matrix Y has a block tri-diagonal structure, expressed as

$$Y = \begin{bmatrix} Y_{11} & Y12 & 0 & \dots & 0 \\ Y_{12}^T & Y_{22} & Y_{23} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & Y_{N-2,N-1}^T & Y_{N-1,N-1} & Y_{N-1,N} \\ 0 & \dots & 0 & Y_{N-1,N}^T & Y_N \end{bmatrix}. \tag{A.16}$$

The Cholesky factor of $Y = L_Y L_Y^T$, expressed as

$$L_Y = \begin{bmatrix} L_{11} & 0 & 0, \dots & 0 & \\ L_{21} & L_{22} & 0, \dots & 0 & \\ 0 & L_{32} & L_{33} & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & L_{N,N-1} & L_{N,N} \end{bmatrix}, \tag{A.17}$$

can be computed by solving [Dom+12]

$$\begin{aligned} Y_{11} &= L_{11} L_{11}^T, \\ Y_{k,k+1} &= L_{k,k} L_{k+1,k}^T, \quad 1 \le k \le N, \\ Y_{k,k} - L_{k,k-1} L_{k,k-1}^T &= L_{k,k} L_{k,k}^T, \quad 2 \le k \le N. \end{aligned} \tag{A.18}$$

It must be noted that the matrix to be decomposed, Y, can be obtained efficiently using the Cholesky factors of $\Phi = L_\Phi L_\Phi^T$ [Zan+17].

Appendix B

FORCES Pro

B.1 FORCES Pro High-Level Interface

Throughout the thesis, we have used FORCES Pro high-level interface to generate solvers for the optimization problems. Here, we explain how to express an optimization problem in FORCES Pro python client, through a simple example of a single vehicle, described with the model (2.31), moving from its initial position to a given final position while avoiding a static obstacle. The FORCES Pro NLP solver takes in (potentially) non-convex, finite-time nonlinear OCPs with horizon length of N in the following form,

$$
\begin{aligned}
\text{minimize} \quad & \sum_{k=1}^{N-1} f_k(z_k, p_k) && \text{(separable objective)} \\
\text{subject to} \quad & z_1(\mathcal{I}) = z_{\text{init}} && \text{(initial equality)} \\
& E_k z_{k+1} = c_k(z_k, p_k) && \text{(inter-stage equality)} \\
& z_N(\mathcal{N}) = z_{\text{final}} && \text{(final equality)} \\
& \underline{z}_k \leq z_k \leq \bar{z}_k && \text{(upper-lower bounds)} \\
& F_k z_k \in [\underline{z}_k, \bar{z}_k] \cap \mathbb{Z} && \text{(integer variables)} \\
& \underline{h}_k \leq h_k(z_k, p_k) \leq \bar{h}_k && \text{(nonlinear constraints)}
\end{aligned}
$$

Figure B.1: Supported problem in FORCES Pro high-level interface [DJ19]

where $z_k \in \mathbb{R}^{n_k}$ is the vector of optimization variables and usually consists of all inputs and states in the problem; $p_k \in \mathbb{R}^{l_k}$ is the vector of real-time data; $f_k : \mathbb{R}^{n_k} \times \mathbb{R}^{l_k} \to \mathbb{R}$ is the stage cost function; $c_k : \mathbb{R}^{n_k} \times \mathbb{R}^{l_k} \to \mathbb{R}^{n_{eq}}$ is the stage transition function which, along with the matrix E_k, connect variables of $k+1$-th stage to those of k-th stage; and $h_k : \mathbb{R}^{n_k} \times \mathbb{R}^{l_k} \to \mathbb{R}^{n_{ineq}}$ expresses the (potentially non-convex) inequality constraints. $\mathcal{I}$ and $\mathcal{N}$ indicate respectively the indices of the optimization variables that are assigned initial and final values. All parameters highlighted in red can be changed in real-time while invoking the generated solver.

In order to generate a solver for the above optimization problem, all functions and the corresponding dimensions must be defined as explained below.

B.1.1 Expressing the optimization problem in python

Once the FORCES Pro client has been downloaded and the requirements have been installed, the FORCES Pro Python client must be added to the Python path. This will allow the user to import forcespro and/or forcespro.nlp in Python scripts and to use the FORCES Pro functionality. To

make the FORCES Pro client available, the user must add FORCES Pro client directory to the PYTHONPATH or to sys.path before importing it in the script, as follows:

```
import sys
sys.path.append('C:/path/to/forces-pro-client/')
```

FORCES pro requires all functions and their derivatives (Jacobians) to be evaluated in each iteration. This can be done by providing external function evaluation in C (used frequently throughout the thesis) or using the automatic differentiation tool CasADi integrated with FORCES Pro. For the CasADi-based approach, the package must be imported at the beginning of the script:

```
import numpy as np
import casadi
import forcespro
import forcespro.nlp
```

In FORCES Pro python client, optimization problems are described with objects of different types. For the CasADi-based approach, one should use the SymbolicModel with no argument or with a single argument, denoting the number of stages, if the dimensions, cost function or other quantities vary in different stages of the problem. The dimension of the optimization variables, the number of equality and inequality constraints, and the number of real-time parameters should be defined as:

```
# Problem dimensions
Nstages = 80  # number of stages
model = forcespro.nlp.SymbolicModel(Nstages)
model.nvar = 7 #  number of variables
model.neq = 5 # number of equality constraints
model.nh =  1  # number of inequality constraint
model.npar = 2 # number of runtime parameters
```

The cost function can be defined using a handle to a Python function with the optimization variables as its first argument:

```
# Objective function
model.objective = lambda z: z[0]**2 + z[1]**2
```

where the optimization variables, z, indexed from 0 are the collection of the 2 inputs and 5 states of the vehicle's model (2.31).

In order to define the equality constraints imposed by the continuous-time model of the vehicle dynamics, one must first describe the model of the system as a python function:

```
def continuous_dynamics(x, u):
    return casadi.vertcat(  x[2] * casadi.cos(x[3]),
                            x[2] * casadi.sin(x[3]),
                            u[0],
                            x[4],
                            u[1])
```

and then use an integrator function, such as the explicit RK4 (the default integrator), and set its options, like the integration interval, and then set model.eq as:

```
integrator_stepsize = 0.1
model.eq = lambda z: forcespro.nlp.integrate(continuous_dynamics, z[2:7], z
    [0:2],
                         integrator=forcespro.nlp.integrators.RK4,
                         stepsize=integrator_stepsize)
model.E = np.concatenate([np.zeros((5,2)), np.eye(5)], axis=1)
```

The selection matrix E links the stage optimization variables to the states and inputs of the continuous dynamics function.

The nonlinear inequality constraints can be similarly defined with Python functions. For example, the collision avoidance constraint between the vehicle and a static obstacle and the corresponding upper and lower bounds should be expressed as:

```
#  Inequality constraints
model.ineq = lambda z,p: (z[2] - p[0]) ** 2 + (z[3] - p[1]) ** 2

# Upper/lower bounds for the inequality constraint
model.hu = +np.inf
model.hl = 1**2
```

The position of the obstacle is defined with the run-time parameter p which is declared during call to the solver.

The upper and lower bounds on the optimization variables, i.e., inputs and states of the vehicle's model, should be defined as:

```
#  upper/lower variable bounds
model.lb = np.array([-0.1, -0.1,  0,   0,   0,     -np.pi/2,  np.pi])
model.ub = np.array([+0.1, +0.1,  100,   100,   0.7,  np.pi/2,  np.pi])
```

For varying dimensions, parameters, constraints, or functions, the quantities defined above must be expressed as Python lists of length 'Nstages' , each item of which is linked to a stage of the problem.

Finally, the initial and final conditions on the optimization variables must be set as follows:

```
# Initial and final conditions
# Initial condition on vehicle's position, speed and course angle
xinit = np.array([0.5, 1, 0, np.deg2rad(45)])
model.xinitidx = range(2,6)
# Final condition on on vehicle's position, speed and course angle
xfinal = np.array([2, 5.5, 0.6, np.deg2rad(0)])
model.xfinalidx = range(2,6)
```

The indices also need to be specified to indicate on which variables the initial and final conditions are imposed.

Having defined the different elements of the optimization problem as above, the next code snippet sets the solver options and generates the FORCES solver.

```
 # Set solver options
    codeoptions = forcespro.CodeOptions('FORCESNLPsolver')
    codeoptions.maxit = 500     # Maximum number of iterations
    codeoptions.printlevel = 0
    codeoptions.optlevel = 0    # 0 no optimization, 1 optimize for size,
    #                             2 optimize for speed, 3 optimize for size &
    speed
    codeoptions.nlp.bfgs_init = 3.0*np.identity(7) # initialization of the
    Hessian approximation
    codeoptions.noVariableElimination = 1.

    # Creates code for symbolic model formulation given above, then connects to
    the
    # server to generate new solver
    solver = model.generate_solver(options=codeoptions)
```

Once the solver is generated, several files, including the compiled solver and MATLAB/Python interfaces for calling it, are downloaded to the current working directory. In order to use the generated solver for solving an instance of the optimization problem, an initial guess, initial and final conditions, and runtime parameters must be defined as follows:

```
x0i = (model.lb + model.ub) / 2.0
x0 = np.transpose(np.tile(x0i, (1, model.N)))
xinit = np.transpose(np.array([1, 1, 0, np.deg2rad(30)]))
xfinal = np.transpose(np.array([2, 6, 0.5, np.deg2rad(0)]))

problem = {"x0": x0,
           "xinit": xinit,
           "xfinal": xfinal}
```

```
# Set runtime parameters
params = np.array([1.5, 3.5])
problem["all_parameters"] = np.transpose(np.tile(params,(1,model.N)))
```

Here, the initial guess is set to be in the middle of upper and lower bounds; however, the FORCES Pro NLP solver finds the solution to local optimality and the initialization can significantly impact the quality of the resulting solution. Therefore, one should provide another initialization point if a better guess is available. Once the real-time parameters are set for all stages, the solver can be invoked using its MEX interface:

```
output, exitflag, info = solver.solve(problem)

# Make sure the solver has exited properly.
assert exitflag == 1, "bad exitflag"
```

The exitflag shoud always be checked before using the solution to make sure the solver has exited without an error. For more information about using FORCES Pro the reader is referred to [DJ19].

Appendix C

B-spline Curves

C.1 B-spline basis functions: definition and properties

Let T be a set of $m+1$ non-decreasing points, $t_0 \leq t_1 \leq \cdots \leq t_m$. The points $t_j \quad j = 0, \ldots, m$ are called knots, the set T is called the knot vector and the half-open interval $[t_j, t_{j+1})$ is the j-th knot span. Computing B-spline basis functions requires specification of a knot vector , T, along with the degree k [PT96]. Given a specific T, the associated B-spline basis functions of degree k are defined recursively as

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t) \quad k \geq 1 \tag{C.1}$$

where $N_{i,0}$ is a step function and is defined as

$$N_{i,0}(t) = \begin{cases} 1 & \text{for} \quad t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{C.2}$$

Fig. C.1 shows B-spline basis functions of degree 0 over the knot interval $T = \{0, 0.25, 0.5, 0.75, 1\}$.

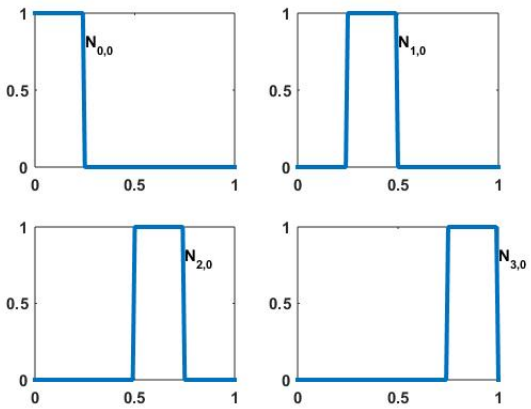
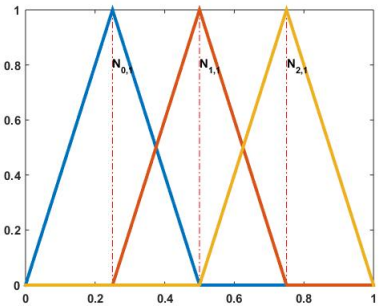


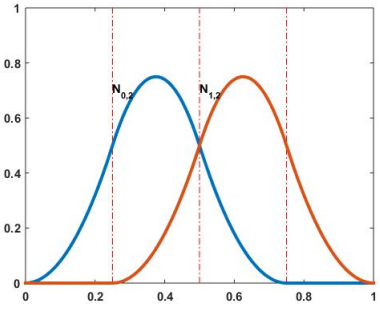
Figure C.1: B-spline basis functions of degree 0, $T = \{0, 0.25, 0.5, 0.75, 1\}$.

B-spline basis functions of degree 1 and 2 are computed below using Eq. (C.1). Fig. C.2 and Fig. C.3 display $N_{i,1}$ and $N_{i,2}$ respectively.



$$
\begin{array}{|l|ll|}
\hline
N_{0,1}(t) = & \begin{cases} 4t & 0 \le t < 0.25 \\ 2(1-2t) & 0.25 \le t < 0.5 \end{cases} \\
\hline
N_{1,1}(t) = & \begin{cases} 4t-1 & 0.25 \le t < 0.5 \\ 3-4t & 0.5 \le t < 0.75 \end{cases} \\
\hline
N_{2,1}(t) = & \begin{cases} 2(2t-1) & 0.5 \le t < 0.75 \\ 4(1-t) & 0.75 \le t < 1 \end{cases} \\
\hline
\end{array}
$$

Figure C.2: B-spline basis functions of degree 1, $T = \{0, 0.25, 0.5, 0.75, 1\}$.



$$
\begin{array}{|l|ll|}
\hline
N_{0,2}(t) = & \begin{cases} 8t^2 & 0 \le t < 0.25 \\ -1.5 + 12t - 16t^2 & 0.25 \le t < 0.5 \\ 4.5 - 12t + 8t^2 & 0.5 \le t < 0.75 \end{cases} \\
\hline
N_{1,2}(t) = & \begin{cases} -0.5 - 4t + 8t^2 & 0.25 \le t < 0.5 \\ -1.5 + 8t - 8t^2 & 0.5 \le t < 0.75 \\ 8(1-t)^2 & 0.75 \le t < 1 \end{cases} \\
\hline
\end{array}
$$

Figure C.3: B-spline basis functions of degree 2, $T = \{0, 0.25, 0.5, 0.75, 1\}$

.

Since knots are not necessarily distinct, Eq.(C.1) can yield zero in the quotient, i.e. $t_i = t_{i+k}$ or $t_{i+1} = t_{i+k+1}$, in which case the quotient is considered to be zero. B-splines basis functions of degree 0, 1, and 2 are computed below using the zero convention for $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$, and are displayed in Fig. C.4 and Fig. C.5 respectively.

B-splines defined above have the following properties,

- $N_{i,k}(t)$ is a piece-wise polynomial of degree k, and is $k-r$ times continuously differentiable at a knot, where r is the multiplicity of the knot.
- $N_{i,k}(t) \ge 0$ for all i, k, and t.
- $N_{i,k}(t)$ is zero outside of $[t_i, t_{i+k+1})$,

$N_{0,0}(t) = 0$	for all t
$N_{1,0}(t) = 0$	for all t
$N_{2,0}(t) = 1$	$0 \le t < 0.3$
$N_{3,0}(t) = 1$	$0.3 \le t < 0.5$
$N_{4,0}(t) = 0$	for all t
$N_{5,0}(t) = 1$	$0.5 \le t < 0.6$
$N_{6,0}(t) = 1$	$0.6 \le t < 1$
$N_{7,0}(t) = 0$	for all t
$N_{8,0}(t) = 0$	for all t

Table C.1: B-spline basis functions of degree 0, $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$.

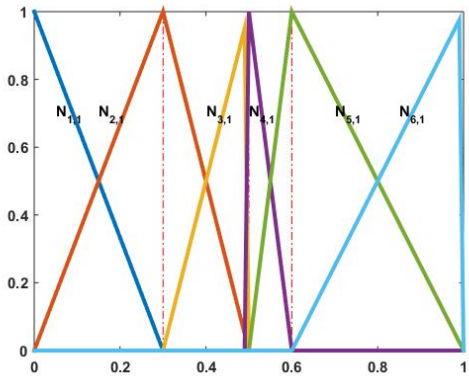


Figure C.4: B-spline basis functions of degree 1, $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$.

$N_{0,1}(t) = 0$	for all t
$N_{1,1}(t) = 1 - \frac{10}{3}t$	$0 \le t < 0.3$
$N_{2,1}(t) = \begin{cases} \frac{10}{3}t \\ 2.5(1-2t) \end{cases}$	$0 \le t < 0.3$ $0.3 \le t < 0.5$
$N_{3,1}(t) = 5t - 1.5$	$0.3 \le t < 0.5$
$N_{4,1}(t) = 6 - 10t$	$0.5 \le t < 0.6$
$N_{5,1}(t) = \begin{cases} 10t - 5 \\ 2.5(1-t) \end{cases}$	$0.5 \le t < 0.6$ $0.6 \le t < 1$
$N_{6,1}(t) = 2.5t - 1.5$	$0.6 \le t < 1$
$N_{7,1}(t) = 0$	for all t

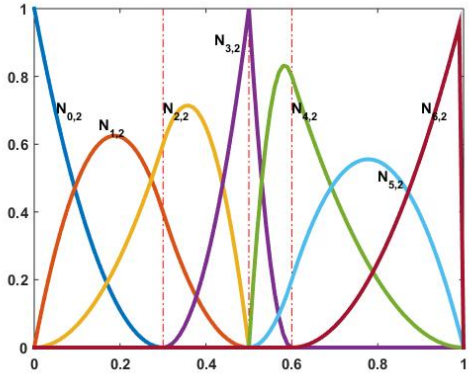


Figure C.5: B-spline basis functions of degree 2, $T = \{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$.

$N_{0,2}(t) = (1 - \frac{10}{3}t)^2$	$0 \le t < 0.3$
$N_{1,2}(t) = \begin{cases} \frac{20}{3}(t - \frac{8}{3}t^2) \\ 2.5(1-2t)^2 \end{cases}$	$0 \le t < 0.3$ $0.3 \le t < 0.5$
$N_{2,2}(t) = \begin{cases} \frac{20}{3}t^2 \\ -3.75 + 25t - 35t^2 \end{cases}$	$0 \le t < 0.3$ $0.3 \le t < 0.5$
$N_{3,2}(t) = \begin{cases} (5t - 1.5)^2 \\ (6 - 10t)^2 \end{cases}$	$0.3 \le t < 0.5$ $0.5 \le t < 0.6$
$N_{4,2}(t) = \begin{cases} 20(-2 + 7t - 6t^2) \\ 5(1-t)^2 \end{cases}$	$0.5 \le t < 0.6$ $0.6 \le t < 1$
$N_{5,2}(t) = \begin{cases} 20t^2 - 20t + 5 \\ -11.25t^2 + 17.5t - 6.25 \end{cases}$	$0.5 \le t < 0.6$ $0.6 \le t < 1$
$N_{6,2}(t) = 6.25t^2 - 7.5t + 2.25$	$0.6 \le t < 1$

- In any knot span $[t_j, t_{j+1})$, at most $k+1$ of the $N_{i,k}$ are non-zero, namely $N_{j-k,k}, \ldots, N_{j,k}$.
- For any knot span $[t_j, t_{j+1})$, $\sum_{i=j-k}^{j} N_{i,k}(t) = 1$.
- A knot vector of the form

$$T = \{\underbrace{0, \ldots, 0}_{\text{k+1}}, \underbrace{1, \ldots, 1}_{\text{k+1}}\} \tag{C.3}$$

  yield the Bernstein polynomials of degree k.

C.2 B-spline curves: definition and properties

Given $n+1$ control points, $c_0, \ldots, c_n$, and a knot vector T, a B-spline curve of degree k is defined by

$$c(t) = \sum_{i=0}^{n} c_i N_{i,k}(t), \quad n \ge k-1, \quad t \in [t_k, t_{m-k}]. \tag{C.4}$$

which is a piece-wise polynomial curve defined over the interval $[t_0, t_f]$ for a clamped knot vector of the form,

$$T = \{\underbrace{t_0, \ldots, t_0}_{\text{k+1}}, t_{k+1}, \ldots, t_{m-k-1}, \underbrace{t_f, \ldots, t_f}_{\text{k+1}}\} \tag{C.5}$$

The degree k of a B-spline curve, the number of its control points, $n+1$, and the number of knots, $m+1$, are related by

$$m = k + n + 1 \tag{C.6}$$

Fig. C.6 and Fig. C.7 show examples of B-spline curves together with their corresponding control polygon.

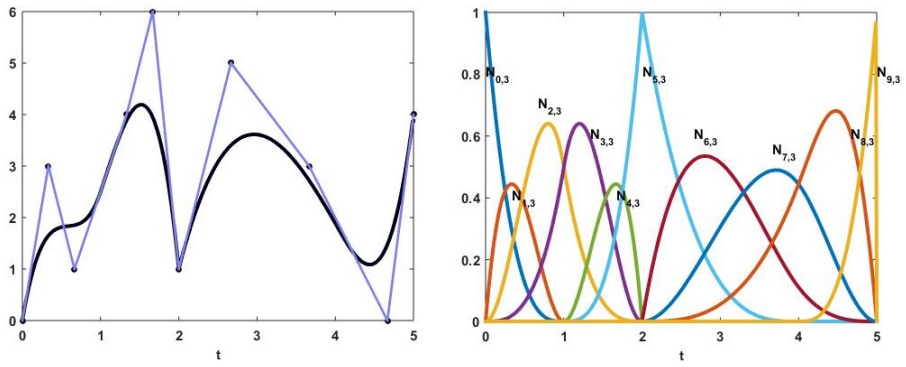


Figure C.6: A cubic B-spline curve and its control polygon (left), the corresponding B-spline basis functions(right) with $T = \{0,0,0,0,1,1,2,2,2,4,5,5,5,5\}$. The control polygon is obtained by connecting the points with coordinates $(\bar{t}_i, c_i), i = 0, \ldots, n$, where $\bar{t}_i = \frac{t_{i+1}+\cdots+t_{i+k}}{k}$.

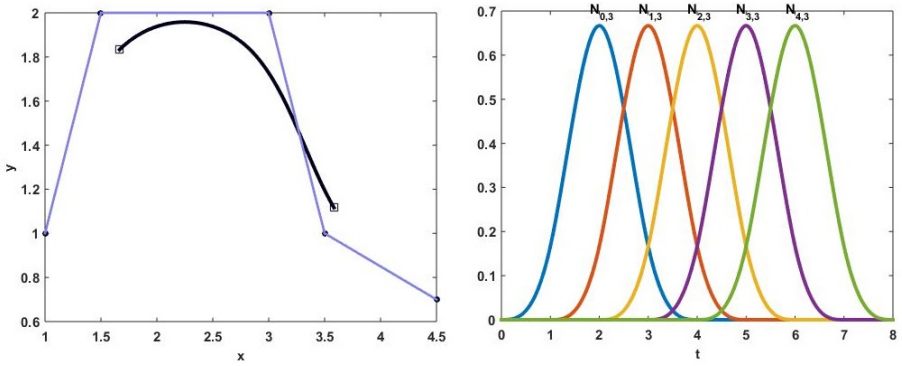


Figure C.7: A cubic B-spline curve and its control polygon (left), the corresponding B-spline basis functions(right) with unclamped $T = \{0,1,2,3,4,5,6,7,8\}$. The curve is not clamped at the end points, and is only defined over the interval $t \in [t_k, t_{m-k}] = [3,5]$

Most algorithms for Bézier curves have a generalized form for B-spline curves. (See [PT96] and [PBP02] for a full list of properties and algorithms). The de Casteljau's algorithm, in particular, can be realized through a special multiple knot Insertion. On accounts of the fundamental equality (C.6), introducing a new knot into T without changing the shape of the curve, i.e., the so-called knot insertion, must be followed by increasing n, i.e. adding a new control point, or increasing k. Adding a new control point is usually done by removing some of the existing control points and replacing them with new ones by corner cutting, which changes the control polygon

locally over the affected knot spans. Increasing the degree of the curve k, as an alternative, would change the control polygon of the curve globally.

C.2.1 Convergence under knot insertion

The sequence of control polygons generated with repeated knot insertion into the knot vector T converges to the B-spline curve $c(t)$, and the convergence rate is quadratic in the maximum knot distance, i.e.,

$$\max\|c(\bar{t}_i) - c_i\| = O(h^2) \tag{C.7}$$

where $\bar{t}_i = \frac{t_{i+1}+\ldots,t_{i+k}}{k}$ and

$$h = \max\{\Delta t_i | [t_i, t_{i+1}] \subset [t_0, t_f]\} \tag{C.8}$$

The maximum is taken over all i such that $[t_{i+1}, t_{i+k}] \subset [t_0, t_f]$ [PBP02].

References

[HR71] GA Hicks and WH Ray. "Approximation methods for optimal control synthesis". In: *The Canadian Journal of Chemical Engineering* 49.4 (1971), pp. 522–528.

[SS78] RWH Sargent and GR Sullivan. "The development of an efficient optimal control package". In: *Optimization Techniques*. Springer, 1978, pp. 158–168.

[Far83] Gerald Farin. "Algorithms for rational Bézier curves". In: *Computer-aided design* 15.2 (1983), pp. 73–77.

[DP85] Rina Dechter and Judea Pearl. "Generalized best-first search strategies and the optimality of A". In: *Journal of the ACM (JACM)* 32.3 (1985), pp. 505–536.

[Kha86] Oussama Khatib. "The potential field approach and operational space formulation in robot control". In: *Adaptive and Learning Systems*. Springer, 1986, pp. 367–377.

[GJK88] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203.

[RK88] Elon Rimon and Daniel E Koditschek. "Exact robot navigation using cost functions: the case of distinct spherical boundaries in E^n". In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE. 1988, pp. 1791–1796.

[Fli+92] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. "Sur lessystèmesnon linéaires différentiellement plats". In: *CR Acad. Sci. Paris* (1992), p. 619.

[Don+93] Bruce Donald, Patrick Xavier, John Canny, and John Reif. "Kinodynamic motion planning". In: *Journal of the ACM (JACM)* 40.5 (1993), pp. 1048–1066.

[DX95] Bruce Randall Donald and Patrick G. Xavier. "Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open-chain manipulators". In: *Algorithmica* 14.6 (1995), pp. 480–530.

[EKR95] Gamal Elnagar, Mohammad A Kazemi, and Mohsen Razzaghi. "The pseudospectral Legendre method for discretizing optimal control problems". In: *IEEE transactions on Automatic Control* 40.10 (1995), pp. 1793–1796.

[Fli+95] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. "Flatness and defect of non-linear systems: introductory theory and examples". In: *International journal of control* 61.6 (1995), pp. 1327–1361.

[LT95] Anis Limaiem and Francois Trochu. “Geometric algorithms for the intersection of curves and surfaces”. In: *Computers & graphics* 19.3 (1995), pp. 391–403.

[LM95] Ming C Lin and Dinesh Manocha. “Fast interference detection between geometric models”. In: *The visual computer* 11.10 (1995), pp. 542–561.

[MRS95] Richard M Murray, Muruhan Rathinam, and Willem Sluis. “Differential flatness of mechanical control systems: A catalog of prototype systems”. In: *ASME international mechanical engineering congress and exposition*. Citeseer. 1995.

[AV96] SK Agrawal and T Veeraklaew. “A higher-order method for dynamic optimization of a class of linear systems”. In: (1996).

[FG96] R Farouki and T Goodman. “On the optimal stability of the Bernstein basis”. In: *Mathematics of computation* 65.216 (1996), pp. 1553–1566.

[Kav+96] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[PT96] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996.

[AF98] SK Agrawal and N Faiz. “Optimization of a class of nonlinear dynamic systems: new efficient method without lagrange multipliers”. In: *Journal of optimization theory and applications* 97.1 (1998), pp. 11–28.

[Bet98] John T Betts. “Survey of numerical methods for trajectory optimization”. In: *Journal of guidance, control, and dynamics* 21.2 (1998), pp. 193–207.

[FA98] Nadeem Faiz and Sunil K Agrawal. “Optimal control of 2-input chained systems using higher-order method”. In: *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207)*. Vol. 1. IEEE. 1998, pp. 6–7.

[JC98] David E Johnson and Elaine Cohen. “A framework for efficient minimum distance computations”. In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*. Vol. 4. IEEE. 1998, pp. 3678–3684.

[NPL98] D Nairn, J Peters, and D Lutterkort. “Sharp, Quantitative Bounds on the Distance Between a Bezier Curve and its Control Polygon”. In: (1998).

[VM98] Michiel J Van Nieuwstadt and Richard M Murray. “Real-time trajectory generation for differentially flat systems”. In: *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* 8.11 (1998), pp. 995–1020.

[NPL99] David Nairn, Jörg Peters, and David Lutterkort. “Sharp, quantitative bounds on the distance between a polynomial piece and its Bézier control polygon”. In: *Computer Aided Geometric Design* 16.7 (1999), pp. 613–631.

[BK00] Robert Bohlin and Lydia E Kavraki. “Path planning using lazy PRM”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on*

Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). Vol. 1. IEEE. 2000, pp. 521–528.

[Bor00] Scott A Bortoff. "Path planning for UAVs". In: *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*. Vol. 1. 6. IEEE. 2000, pp. 364–368.

[KL00] James J Kuffner and Steven M LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.

[EK01] Gershon Elber and Myung-Soo Kim. "Geometric constraint solver using multivariate rational spline functions". In: *Proceedings of the sixth ACM symposium on Solid modeling and applications*. 2001, pp. 1–10.

[FAM01] Nadeem Faiz, Sunil K Agrawal, and Richard M Murray. "Trajectory planning of differentially flat systems with dynamics and inequalities". In: *Journal of Guidance, Control, and Dynamics* 24.2 (2001), pp. 219–227.

[Jad01] Ali Jadbabaie. "Receding horizon control of nonlinear systems: A control Lyapunov function approach". PhD thesis. Citeseer, 2001.

[LS02] Christian Lennerz and Elmar Schomer. "Efficient distance computation for quadratic curves and surfaces". In: *Geometric Modeling and Processing. Theory and Applications. GMP 2002. Proceedings*. IEEE. 2002, pp. 60–69.

[MMR02] Ph Martin, Richard M Murray, and Pierre Rouchon. "Flat systems". In: (2002).

[PM02] Nicholas M Patrikalakis and Takashi Maekawa. *Shape interrogation for computer aided design and manufacturing*. Vol. 15. Springer, 2002.

[PBP02] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Vol. 6. Springer, 2002.

[MMR03] Phillipe Martin, Richard M Murray, and Pierre Rouchon. "Flat systems, equivalence and trajectory generation". In: (2003).

[RF03] I Michael Ross and Fariba Fahroo. "Legendre pseudospectral approximations of optimal control problems". In: *New trends in nonlinear dynamics and control and their applications*. Springer, 2003, pp. 327–342.

[Van03] Gino Van Den Bergen. *Collision detection in interactive 3D environments*. CRC Press, 2003.

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Cor+04] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. "Coverage control for mobile sensing networks". In: *IEEE Transactions on robotics and Automation* 20.2 (2004), pp. 243–255.

[Eri04] Christer Ericson. *Real-time collision detection*. Crc Press, 2004.

[KKK04] Menelaos I Karavelas, Panagiotis D Kaklis, and Konstantinos V Kostas. "Bounding the distance between 2D parametric Bézier curves and their control polygon". In: *Computing* 72.1 (2004), pp. 117–128.

[Kir04] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[DBS05] Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. "A real-time iteration scheme for nonlinear optimization in optimal feedback control". In: *SIAM Journal on control and optimization* 43.5 (2005), pp. 1714–1736.

[FLS05] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. "A guide to heuristic-based path planning". In: *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*. 2005, pp. 9–18.

[FDF05] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. "Maneuver-based motion planning for nonlinear systems with symmetries". In: *IEEE transactions on robotics* 21.6 (2005), pp. 1077–1091.

[GMS05] Philip E Gill, Walter Murray, and Michael A Saunders. "SNOPT: An SQP algorithm for large-scale constrained optimization". In: *SIAM review* 47.1 (2005), pp. 99–131.

[SKI05] Shingo Shimoda, Yoji Kuroda, and Karl Iagnemma. "Potential field navigation of high speed unmanned ground vehicles on uneven terrain". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE. 2005, pp. 2828–2833.

[Die+06] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. "Fast direct multiple shooting algorithms for optimal robot control". In: *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93.

[Gar+06] Santiago Garrido, Luis Moreno, Mohamed Abderrahim, and Fernando Martin. "Path planning for mobile robot navigation using voronoi diagram and fast marching". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 2376–2381.

[KP06] Maciej Kalisiak and Michiel van de Panne. "RRT-blossom: RRT with a local flood-fill behavior". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 1237–1242.

[LaV06] Steven M LaValle. "Planning algorithms". In: (2006).

[MZ06] 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*. Springer. 2006, pp. 157–174.

[WB06] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[Bed+07] Nazareth Bedrossian, Sagar Bhatt, Mike Lammers, and Louis Nguyen. "Zero-Propellant Maneuver [TM] Flight Results for 180 deg ISS Rotation". In: *Proceedings of the 20th International Symposium on Space Flight Dynamics*. 2007.

[DS07] Tor Dokken and Vibeke Skytt. "Intersection algorithms and CAGD". In: *Geometric Modelling, Numerical Simulation, and Optimization*. Springer, 2007, pp. 41–90.

[Kuw07] Yoshiaki Kuwata. "Trajectory planning for unmanned vehicles using robust receding horizon control". PhD thesis. Massachusetts Institute of Technology, 2007.

[Kuw+07] Yoshiaki Kuwata, Arthur Richards, Tom Schouwenaars, and Jonathan P How. "Distributed robust receding horizon control for multivehicle guidance". In: *IEEE Transactions on Control Systems Technology* 15.4 (2007), pp. 627–641.

[LRG07] L Ryan Lewis, I Michael Ross, and Qi Gong. "Pseudospectral motion planning techniques for autonomous obstacle avoidance". In: *2007 46th IEEE Conference on Decision and Control*. IEEE. 2007, pp. 5997–6002.

[BG08] Priyadarshi Bhattacharya and Marina L Gavrilova. "Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path". In: *IEEE Robotics & Automation Magazine* 15.2 (2008), pp. 58–66.

[CCE08] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. "Path planning based on bézier curve for autonomous ground vehicles". In: *Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*. IEEE. 2008, pp. 158–166.

[EG08] Gershon Elber and Tom Grandine. "Hausdorff and Minimal Distances between Parametric Freeforms in $\mathbb{R}^2$ and $\mathbb{R}^3$". In: *International conference on geometric modeling and processing*. Springer. 2008, pp. 191–204.

[Far08a] Rida T Farouki. *Pythagorean-hodograph curves: algebra and geometry inseparable, Geometry and Computing*. 2008.

[Far08b] Rida T Farouki. "The Bernstein basis". In: *Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable* (2008), pp. 249–260.

[MM08] MH Mabrouk and CR McInnes. "Solving the potential field local minimum problem using internal agent states". In: *Robotics and Autonomous Systems* 56.12 (2008), pp. 1050–1060.

[VLM08] Jur Van den Berg, Ming Lin, and Dinesh Manocha. "Reciprocal velocity obstacles for real-time multi-agent navigation". In: *2008 IEEE international conference on robotics and automation*. Ieee. 2008, pp. 1928–1935.

[Che+09] Xiao-Diao Chen, Linqiang Chen, Yigang Wang, Gang Xu, Jun-Hai Yong, and Jean-Claude Paul. "Computing the minimum distance between two Bézier curves". In: *Journal of Computational and Applied Mathematics* 229.1 (2009), pp. 294–301.

[WB09] Yang Wang and Stephen Boyd. "Fast model predictive control using online optimization". In: *IEEE Transactions on control systems technology* 18.2 (2009), pp. 267–278.

[Bis+10] Adrian N Bishop, Barış Fidan, Brian DO Anderson, Kutluyıl Doğançay, and Pubudu N Pathirana. "Optimality analysis of sensor-target localization geometries". In: *Automatica* 46.3 (2010), pp. 479–492.

[CHL10] Georgios Chaloulos, Peter Hokayem, and John Lygeros. "Distributed hierarchical MPC for conflict resolution in air traffic control". In: *Proceedings of the 2010 American Control Conference*. IEEE. 2010, pp. 3945–3950.

[Che+10] Xiao-Diao Chen, Weiyin Ma, Gang Xu, and Jean-Claude Paul. "Computing the Hausdorff distance between two B-spline curves". In: *Computer-Aided Design* 42.12 (2010), pp. 1197–1206.

[GT10] Elena Glassman and Russ Tedrake. "A quadratic regulator-based heuristic for rapidly exploring state space". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 5021–5028.

[GKM10] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. "A survey of motion planning algorithms from the perspective of autonomous UAV guidance". In: *Journal of Intelligent and Robotic Systems* 57.1 (2010), pp. 65–100.

[KH10] Yoshiaki Kuwata and Jonathan P How. "Cooperative distributed robust trajectory optimization using receding horizon MILP". In: *IEEE Transactions on Control Systems Technology* 19.2 (2010), pp. 423–431.

[QR10] Alban Quadrat and Daniel Robertz. "Controllability and differential flatness of linear analytic ordinary differential systems". In: *Proceedings of 19th International Symposium on Mathematical Theory of Networks and Systems, Budapest (Hungary),(05-09/07/10)*. 2010.

[Rao+10] Anil V Rao, David A Benson, Christopher Darby, Michael A Patterson, Camila Francolin, Ilyssa Sanders, and Geoffrey T Huntington. "Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method". In: *ACM Transactions on Mathematical Software (TOMS)* 37.2 (2010), pp. 1–39.

[Ted+10] Francesco Tedesco, Davide M Raimondo, Alessandro Casavola, and John Lygeros. "Distributed collision avoidance for interacting vehicles: a command governor approach". In: *IFAC Proceedings Volumes* 43.19 (2010), pp. 293–298.

[Ber+11a] Jur van den Berg, Stephen J Guy, Jamie Snape, Ming C Lin, and Dinesh Manocha. "Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation". In: *See https://gamma. cs. unc. edu/RVO2* (2011).

[Ber+11b] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. "Reciprocal n-body collision avoidance". In: *Robotics research*. Springer, 2011, pp. 3–19.

[Cha+11] Jung-Woo Chang, Yi-King Choi, Myung-Soo Kim, and Wenping Wang. "Computation of the minimum distance between two Bézier curves/surfaces". In: *Computers & Graphics* 35.3 (2011), pp. 677–684.

[DG11] Moritz Diehl and Sébastien Gros. "Numerical optimal control". In: *Optimization in Engineering Center (OPTEC)* (2011).

[HFD11] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range". In: *Automatica* 47.10 (2011), pp. 2279–2285.

[KF11] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[KH11] Yoshiaki Kuwata and Jonathan P How. "Cooperative distributed robust trajectory optimization using receding horizon MILP". In: *IEEE Transactions on Control Systems Technology* 19.2 (2011), pp. 423–431.

[MK11] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 2520–2525.

[Sna+11] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. "The hybrid reciprocal velocity obstacle". In: *IEEE Transactions on Robotics* 27.4 (2011), pp. 696–706.

[Alo+12] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. "Reciprocal collision avoidance for multiple car-like robots". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 360–366.

[AÅD12] Joel Andersson, Johan Åkesson, and Moritz Diehl. "CasADi: A symbolic package for automatic differentiation and optimal control". In: *Recent advances in algorithmic differentiation*. Springer, 2012, pp. 297–307.

[ASD12] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach". In: *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1917–1922.

[BW12] Christof Büskens and Dennis Wassel. "The esa nlp solver worhp". In: *Modeling and optimization in space engineering*. Springer, 2012, pp. 85–110.

[Dom+12] Alexander Domahidi, Aldo U Zgraggen, Melanie N Zeilinger, Manfred Morari, and Colin N Jones. "Efficient interior point methods for multistage problems arising in receding horizon control". In: *2012 IEEE 51st IEEE conference on decision and control (CDC)*. IEEE. 2012, pp. 668–674.

[Far12] Rida T Farouki. "The Bernstein polynomial basis: A centennial retrospective". In: *Computer Aided Geometric Design* 29.6 (2012), pp. 379–419.

[GGJ12] Alexandra Grancharova, Esten I Grøtli, and Tor A Johansen. "Distributed MPC-based path planning for UAVs under radio communication path loss constraints". In: *IFAC Proceedings Volumes* 45.4 (2012), pp. 254–259.

[Häu+12] Andreas J Häusler, Alessandro Saccon, A Pedro Aguiar, John Hauser, and António M Pascoal. "Cooperative motion planning for multiple autonomous marine vehicles". In: *IFAC Proceedings Volumes* 45.27 (2012), pp. 244–249.

[KNN12] Moharam Habibnejad Korayem, Mostafa Nazemizadeh, and Hamed Rahimi Nohooji. "Smooth jerk-bounded optimal path planning of tricycle wheeled mobile manipulators in the presence of environmental obstacles". In: *International Journal of Advanced Robotic Systems* 9.4 (2012), p. 105.

[Per+12] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2537–2542.

[TCF13] Vasyl Tereshchenko, Sergii Chevokin, and Andriy Fisunenko. "Algorithm for finding the domain intersection of a set of polytopes". In: *Procedia Computer Science* 18 (2013), pp. 459–464.

[Vuk+13] Milan Vukov, Alexander Domahidi, Hans Joachim Ferreau, Manfred Morari, and Moritz Diehl. "Auto-generated algorithms for nonlinear model predictive control on long and on short horizons". In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 5113–5118.

[BCH14] Saptarshi Bandyopadhyay, Soon-Jo Chung, and Fred Y Hadaegh. "Probabilistic swarm guidance using optimal transport". In: *2014 IEEE Conference on Control Applications (CCA)*. IEEE. 2014, pp. 498–505.

[Fer+14] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. "qpOASES: A parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363.

[GLA14] Andrew Giese, Daniel Latypov, and Nancy M Amato. "Reciprocally-rotating velocity obstacles". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3234–3241.

[HK14] Toru Hirata and Makoto Kumon. "Optimal path planning method with attitude constraints for quadrotor helicopters". In: *Proceedings of the 2014 International Conference on Advanced Mechatronic Systems*. IEEE. 2014, pp. 377–381.

[MCH14] Daniel Morgan, Soon-Jo Chung, and Fred Y Hadaegh. "Model predictive control of swarms of spacecraft using sequential convex programming". In: *Journal of Guidance, Control, and Dynamics* 37.6 (2014), pp. 1725–1740.

[PR14] Michael A Patterson and Anil V Rao. "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming". In: *ACM Transactions on Mathematical Software (TOMS)* 41.1 (2014), pp. 1–37.

[WD14] Peng Wang and Baocang Ding. "A synthesis approach of distributed model predictive control for homogeneous multi-agent system with collision avoidance". In: *International Journal of Control* 87.1 (2014), pp. 52–63.

[CCH15] 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)*. IEEE. 2015, pp. 5954–5961.

[Cho+15] Ronald Choe, Javier Puig, Venanzio Cichella, Enric Xargay, and Naira Hovakimyan. "Trajectory generation using spatial Pythagorean Hodograph Bézier curves". In: *AIAA Guidance, Navigation, and Control Conference*. 2015, p. 0597.

[Hau15] Kris Hauser. "Lazy collision checking in asymptotically-optimal motion planning". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 2951–2957.

[LDM15] Alexander Liniger, Alexander Domahidi, and Manfred Morari. "Optimization-based autonomous racing of 1: 43 scale RC cars". In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647.

[OG15] Hao Yi Ong and J Christian Gerdes. "Cooperative collision avoidance via proximal message passing". In: *2015 American Control Conference (ACC)*. IEEE. 2015, pp. 4124–4130.

[Rig15] Gerasimos G Rigatos. *Nonlinear control and filtering using differential flatness approaches: applications to electromechanical systems*. Vol. 25. Springer, 2015.

[SJ15] Zbynek Sir and Bert Jüttler. "On de Casteljau-type algorithms for rational Bézier curves". In: *Journal of Computational and Applied Mathematics* 288 (2015), pp. 244–250.

[CLS16] Jing Chen, Tianbo Liu, and Shaojie Shen. "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1476–1483.

[Cic+16] Venanzio Cichella, Ronald Choe, Syed Bilal Mehdi, Enric Xargay, Naira Hovakimyan, Vladimir Dobrokhodov, Isaac Kaminer, Antonio M Pascoal, and A Pedro Aguiar. "Safe coordinated maneuvering of teams of multirotor unmanned aerial vehicles: A cooperative control framework for multivehicle, time-critical missions". In: *IEEE Control Systems Magazine* 36.4 (2016), pp. 59–82.

[GM16] Mathieu Geisert and Nicolas Mansard. "Trajectory generation for quadrotor based systems using numerical optimal control". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 2958–2964.

[Loi+16] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU". In: *IEEE Robotics and Automation Letters* 2.2 (2016), pp. 404–411.

[MVP16] Tim Mercy, Wannes Van Loock, and Goele Pipeleers. "Real-time motion planning in the presence of moving obstacles". In: *2016 European Control Conference (ECC)*. IEEE. 2016, pp. 1586–1591.

[Mor+16] D Moreno-Salinas, N Crasta, M Ribeiro, B Bayat, AM Pascoal, and J Aranda. "Integrated motion planning, control, and estimation for range-based marine vehicle positioning and target localization". In: *IFAC-PapersOnLine* 49.23 (2016), pp. 34–40.

[RBR16] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments". In: *Robotics research*. Springer, 2016, pp. 649–666.

[Cho17] Ronald Choe. "Distributed cooperative trajectory generation for multiple autonomous vehicles using pythagorean hodograph Bézier curves". PhD thesis. University of Illinois at Urbana-Champaign, 2017.

[Cic+17] Venanzio Cichella, Isaac Kaminer, Claire Walton, and Naira Hovakimyan. "Optimal motion planning for differentially flat systems using Bernstein approximation". In: *IEEE Control Systems Letters* 2.1 (2017), pp. 181–186.

[Dai+17] Li Dai, Qun Cao, Yuanqing Xia, and Yulong Gao. "Distributed MPC for formation of multi-agent systems with collision avoidance and obstacle avoidance". In: *Journal of the Franklin Institute* 354.4 (2017), pp. 2068–2085.

[Fal+17] 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)*. IEEE. 2017, pp. 5774–5781.

[Gar+17] Divya Garg, Michael Patterson, William Hager, Anil Rao, David Benson, and Geoffrey Huntington. "An overview of three pseudospectral methods for the numerical solution of optimal control problems". In: (2017).

[Liu+17a] Chang Liu, Seungho Lee, Scott Varnhagen, and H Eric Tseng. "Path planning for autonomous vehicles using model predictive control". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 174–179.

[Liu+17b] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. "Search-based motion planning for quadrotors using linear quadratic minimum time control". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 2872–2879.

[MVP17] Tim Mercy, Ruben Van Parys, and Goele Pipeleers. "Spline-based motion planning for autonomous guided vehicles in a dynamic environment". In: *IEEE Transactions on Control Systems Technology* 26.6 (2017), pp. 2182–2189.

[MPB17] Mattia Montanari, Nik Petrinic, and Ettore Barbieri. "Improving the GJK algorithm for faster and more reliable distance queries between convex objects". In: *ACM Transactions on Graphics (TOG)* 36.3 (2017), pp. 1–17.

[Näg+17] Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora, and Otmar Hilliges. "Real-time planning for automated multi-view drone cinematography". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–10.

[Pre+17] James A Preiss, Karol Hausman, Gaurav S Sukhatme, and Stephan Weiss. "Trajectory Optimization for Self-Calibration and Navigation." In: *Robotics: Science and Systems*. Vol. 13. 2017.

[VP17a] Ruben Van Parys and Goele Pipeleers. "Distributed model predictive formation control with inter-vehicle collision avoidance". In: *2017 11th Asian Control Conference (ASCC)*. IEEE. 2017, pp. 2399–2404.

[VP17b] Ruben Van Parys and Goele Pipeleers. "Distributed MPC for multi-vehicle systems moving in formation". In: *Robotics and Autonomous Systems* 97 (2017), pp. 144–152.

[VP17c] Ruben Van Parys and Goele Pipeleers. "Spline-based motion planning in an obstructed 3D environment". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 8668–8673.

[Zan+17] Andrea Zanelli, Alexander Domahidi, J Jerez, and Manfred Morari. "FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs". In: *International Journal of Control* (2017), pp. 1–17.

[Zho+17] Dingjiang Zhou, Zijian Wang, Saptarshi Bandyopadhyay, and Mac Schwager. "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1047–1054.

[Gao+18] Fei Gao, William Wu, Yi Lin, and Shaojie Shen. "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 344–351.

[Liu+18] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. "Search-based motion planning for aggressive flight in se (3)". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2439–2446.

[NFK18] Yuanbo Nie, Omar Faqir, and Eric C Kerrigan. "ICLOCS2: Try this optimal control problem solver before you try the rest". In: *2018 UKACC 12th international conference on control (CONTROL)*. IEEE. 2018, pp. 336–336.

[SCP18] Bahareh Sabetghadam, Rita Cunha, and António Pascoal. "Cooperative motion planning with time, energy and active navigation constraints". In: *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*. IEEE. 2018, pp. 1–6.

[HSS19] Shlomi Hacohen, Shraga Shoval, and Nir Shvalb. "Probability navigation function for stochastic static environments". In: *International Journal of Control, Automation and Systems* 17.8 (2019), pp. 2097–2113.

[LS19] Carlos E Luis and Angela P Schoellig. "Trajectory generation for multiagent point-to-point transitions via distributed model predictive control". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 375–382.

[Sab+19] Bahareh Sabetghadam, Alfonso Alcantara, Jesus Capitan, Rita Cunha, Anibal Ollero, and Antonio Pascoal. "Optimal trajectory planning for autonomous drone cinematography". In: *2019 European Conference on Mobile Robots (ECMR)*. IEEE. 2019, pp. 1–7.

[ŞHA19] Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. "Robust trajectory execution for multi-robot teams using distributed real-time replanning". In: *Distributed autonomous robotic systems*. Springer, 2019, pp. 167–181.

[Tan+19] 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)*. IEEE. 2019, pp. 1133–1138.

[TLH19] 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, pp. 1934–1940.

[Zho+19] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. "Robust and efficient quadrotor trajectory generation for fast autonomous flight". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3529–3536.

[BB20] Tim Breitenbach and Alfio Borzi. "The Pontryagin maximum principle for solving Fokker–Planck optimal control problems". In: *Computational Optimization and Applications* 76.2 (2020), pp. 499–533.

[LVS20] Carlos E Luis, Marijan Vukosavljev, and Angela P Schoellig. "Online trajectory generation with distributed model predictive control for multi-robot motion planning". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 604–611.

[Ros20] Isaac M Ross. "Enhancements to the DIDO optimal control toolbox". In: *arXiv preprint arXiv:2004.13112* (2020).

[SCP20] Bahareh Sabetghadam, Rita Cunha, and António Pascoal. "Real-time trajectory generation for multiple drones using bézier curves". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 9276–9281.

[TH20] Jesus Tordesillas and Jonathan P How. "MINVO basis: Finding simplexes with minimum volume enclosing polynomial curves". In: *arXiv preprint arXiv:2010.10726* (2020).

[Váz+20] José L Vázquez, Marius Brühlmeier, Alexander Liniger, Alisa Rupenyan, and John Lygeros. "Optimization-based hierarchical motion planning for autonomous racing". In: *arXiv preprint arXiv:2003.04882* (2020).

[WC20] Paweł Woźny and Filip Chudy. "Linear-time geometric algorithm for evaluating Bézier curves". In: *Computer-Aided Design* 118 (2020), p. 102760.

[ZNS20] Vera Zeidan, Chadi Nour, and Hassan Saoud. "A nonsmooth maximum principle for a controlled nonconvex sweeping process". In: *Journal of Differential Equations* 269.11 (2020), pp. 9531–9582.

[RH21] Andriamahenina Ramanantoanina and Kai Hormann. "New shape control tools for rational Bézier curve design". In: *Computer Aided Geometric Design* 88 (2021), p. 102003.

[SCP21] Bahareh Sabetghadam, Rita Cunha, and António Pascoal. "Trajectory Generation for Drones in Confined Spaces using an Ellipsoid Model of the Body". In: *IEEE Control Systems Letters* 6 (2021), pp. 1022–1027.

[TH21] Jesus Tordesillas and Jonathan P How. "MADER: Trajectory planner in multiagent and dynamic environments". In: *IEEE Transactions on Robotics* (2021).

[Kie+22] Calvin Kielas-Jensen, Venanzio Cichella, Thomas Berry, Isaac Kaminer, Claire Walton, and Antonio Pascoal. "Bernstein Polynomial-Based Method for Solving Optimal Trajectory Generation Problems". In: *Sensors* 22.5 (2022), p. 1869.

[SCP22] Bahareh Sabetghadam, Rita Cunha, and António Pascoal. "A distributed algorithm for real-time multi-drone collision-free trajectory replanning". In: *Sensors* 22.5 (2022), p. 1855.

[DJ19] Alexander Domahidi and Juan Jerez. *FORCES Professional*. 2014–2019. URL: `https://embotech.com/FORCES-Pro`.

[20] *MultiDrone @ONLINE*. 2016-2020. URL: `https://multidrone.eu/`.