10-24-2024

Optimization-based Motion Planning with Homotopy and Homology Class Constraints

Wenbo He
Washington University – McKelvey School of Engineering

WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Electrical & Systems Engineering

Dissertation Examination Committee:
Shen Zeng, Chair
Mohamed Ali Belabbas
ShiNung Ching
Andrew Clark
Jr-Shin Li

Optimization-based Motion Planning with Homotopy and Homology Class Constraints

by
Wenbo He

A dissertation presented to
the McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

December 2024
St. Louis, Missouri

Table of Contents

List of Figures

Acknowledgments

I want to express my deepest gratitude to everyone who has played a role, in shaping my doctoral journey and enhancing this thesis with their guidance, encouragement and friendship. I am especially grateful to my advisor, Shen Zeng for his support and insightful advice that have been crucial throughout my PhD experience. I also want to thank Yunshen Huang for his invaluable collaborations and his remarkable expertise in controller design and quadcopter experiment design. I also want to thank Jie Wang, Jingran Qie and Haoyu Yin for their assistance in paper writing. Furthermore, I am very thankful to my committee members Jr-Shin Li, ShiNung Ching, Andrew Clark and Mohamed Ali Belabbas.

This work was supported by NSF grant CMMI-1933976.

Wenbo He

Washington University in St. Louis
December 2024

ABSTRACT OF THE DISSERTATION

Optimization-based Motion Planning with Homotopy and Homology Class Constraints

by

Wenbo He

Doctor of Philosophy in Electrical Engineering

Washington University in St. Louis, 2024

Professor Shen Zeng, Chair

Motion Planning is a fundamental problem in robotics that aims to find an optimal trajectory for a system to move on while avoiding obstacles in the environment. Often, a feasible trajectory connecting the start and target point with the shortest length is highly desirable. Additionally, in scenarios such as drone racing or surveillance, topology constraints may arise. At the low level, the LQR or PID controller is utilized to steer the agent to move along the designed trajectory. At a high level, optimization-based, search-based, or sample-based algorithms are utilized to synthesize the feasible trajectory. In this thesis, we deal with optimization-based trajectory synthesizing along with a special type of topology constraint named homotopy and homology class constraints.

In the first part of the thesis, we just ignore topology constraints and emphasize how to transform motion planning tasks into optimization tasks while considering start-point end-point constraints and minimal energy or minimal time loss function. Although the loss function is a differential function, the first-order gradient optimization method, such as Adam, shows less ability to find the optimal. However, methods that utilize second-order information, such as the Gaussian-Newton method and the interior point optimizer, can solve the problem quickly and perfectly.

The second part of the thesis emphasizes our proposed optimization method for motion planning with homotopy and homology class constraints. We first introduce the Auxiliary Energy Reduction Technique. The hallmark of our approach is that we first introduce virtual control terms to the original system dynamics that ensure that any preset state trajectory is dynamically feasible with respect to the new extended system. We then gradually shift the contribution of the artificial inputs to the actual original inputs, and in the end, the trajectory will be deformed to the one of the same homotopy class that is now also feasible with respect to the original system. However, the aforementioned method suffers from low efficiency when the required homotopy class is complex. Therefore, in the second method, we deal with two-dimensional obstacles by synthesizing auxiliary trajectories for obstacles then synthesizing optimal trajectories for the agent, and then gradually deforming obstacle trajectories to the original ones and keeping the agent's trajectory optimal, which improves efficiency. To explore the homology class constraints, in the third method, we solve homology class constraints with respect to three-dimensional obstacles by embedding them in two-dimensional. In the fourth method, we combine our method of the third method to extend the second method to deal with special 3-dimensional obstacles with homotopy class constraints.

Part I

Introduction of Motion Planning

Chapter 1

Introduction

1.1 Background and Motivation

The basic task of motion planning is to find a dynamically feasible trajectory connecting a given starting point with a desired target point in a state space that possibly contains one or more obstacles. This topic has been intensively investigated in literature and has been applied to real-world applications, especially in drones [38], automated vehicles [79] and mobile robots [30]. Although most authors divided the whole planning task into global planning and local planning parts, synthesizing control inputs for the dynamical system in the first step helps generate high-fidelity trajectories [21] [51]. When the focus is only on steering complicated dynamics (without the consideration of the constraints), many control techniques can be leveraged to achieve this goal, such as iterative LQR [17], model predictive control [23], sliding mode control [20], and collocation methods [43]. But when obstacles are considered, other methods need to be considered.

If the dynamic system is as simple as mobile robots that can roam freely, the motion planning task is simply finding a path connecting the given start point and the goal point, which falls into the realm of graph searching algorithms. By separating continuous state space into discrete ones, the number of states now becomes limited and the Dijkstra algorithm can be used to find the required path with minimal length [58] [53], which has been applied in real-world challenge [13]. Similarly, other graph searching methods including A* algorithm [85] [22], Field D* [24], Theta* [19], Anytime repairing A* (ARA*) [54] and jump point search [91] have been applied in motion planning with improved performance.

For more intricate systems where free roam is infeasible, the state lattice algorithm is usually utilized. The method separates the state space into the grid and connects vertexes with feasible and smooth curves [67]. A cost function that counts the length and the curvature

assigns weights to connections, and then a search algorithm like A* [52] or D* [71] finds the best path that connects the start state and the goal state without overlapping obstacles. Different from the aforementioned pure graph searching method, the state lattice algorithm carefully selects candidate middle points and feasible potential interconnect trajectories, and synthesizes a smooth path with low computational cost. Thus it has been widely applied in mobile robot [68] and vehicle motion planning tasks [4] [18].

For a highly investigated dynamic system, quadcopter, any smooth curve can be considered as a feasible trajectory due to its holonomic properties. Hence the global motion planner for quadcopter is to synthesize a smooth curve that connects given points and meanwhile be obstacle-free. The workflow is usually getting a set of waypoints that are obstacle-free and marking the contour of the required path, then synthesizing a smooth curve interpolating waypoints [14]. The interpolating curve planners can utilize different techniques to fit waypoints and generate different curves. Lines and circles have been used to generate curves, which also fit the real trajectories of vehicles and airplanes [35]. The spline curve is the piecewise polynomial parametric curve that has been used in motion planning [66] [75]. However, if the spline curve uses waypoints as control points, the curve actually doesn't go across waypoints. Bézier curves mitigate this challenge as Bézier curves will travel across waypoints iteratively. As a type of parametric trajectory, a widely-adopted choice in motion planning [60] [70] [78], the Bézier curve is selected for its simple representation and inherent smoothness [63] [78] [59]. These properties enable fast plannings and easy track for differentially flat systems [76] [27], a large family including mobile robots [62], quadcopters [28].

Sampling-based motion planning is another set of widely applied motion planning methods [21] that can handle environments with obstacles. This approach has its advantages in providing fast solutions for difficult problems by simply sampling in the configure space. However, it has the drawback of finding the suboptimal solution. Randomized Potential Planner (RPP) uses random walk to escape local minima of the potential field planner [6] or only utilizes the random walk for planning [15]. Probabilistic Roadmap Method (PRM) [42] [41] and Rapidly-exploring Random Trees (RRT) [50] are proposed later. Kinodynamic RRT* [82], and LQR-RRT* [65] further provide asymptotic optimality and the ability to process kinodynamic systems.

While the obstacle constraints have been well considered in methods mentioned above, handling the motion planning problem concerning homotopy class constraints is still challenging.

In addition to the aforementioned issues, oftentimes, merely avoiding collisions is insufficient in some topology-sensitive tasks. To illustrate, the constraint of assigning a car to pass the crowd but only on their right side can be considered as a homotopy class constraint. Letting the drone fly around two people once can be transformed into a homology class constraint. Hence, it is important to consider and distinguish among different homotopy classes for generated trajectories. For this consideration, [10] provides a way to classify homotopy classes in higher-dimensional space and proposes a search-based robot path planning method fulfilling topological constraints. Probabilistic roadmaps introduced in [40] are also capable of path generation under homotopy classes constraints. Furthermore, Gaussian process inference is leveraged in [49] to achieve online motion planning involving multiple homotopy classes. However, these methods are again limited to robots with simple dynamics. Although the methods mentioned above are promising in their specified tasks, there are only a few works that address the motion planning problem with both considerations of full-scale nonholonomic dynamical systems and homotopy class constraints. The task of fulfilling the homotopy class constraints introduced by the obstacles is more challenging than mere obstacles, which are aimed at obtaining feasible trajectories belonging to a certain homotopy class. The work in [10] provides a way to classify homotopy classes in higher-dimensional space and proposes a search-based robot path planning method fulfilling topological constraints. However, it still suffers from the limiting application of the simple dynamic system. [7] and [57] provide an elegant way, which is named the affine geometric heat flow, to solve the motion planning problems with homotopy class constraints. Nevertheless, this approach relies on partial differential equation solvers, and the application is confined to the form of control affine system.

After synthesizing the global motion planning, the acquired trajectories may lack control inputs (by graph searching or sample-based methods) or they may not be safe due to environmental disturbances, the movement of obstacles, or the shift of the system's dynamic. Therefore local motion planning methods are needed. Optimal control such as Linear Quadratic Regulators (LQRs), can be utilized to design local controllers, which been widely applied on quadcopters [69] [26] [55]. Model Predictive Controls (MPCs) [90] [37] is another popular control technique that uses the model to compute suitable inputs at each time step. These methods require a known system dynamic, which may fail the control design if the system is not properly modeled [34]. Due to the limitations of model-based approaches, the data-driven design of flight control systems is becoming an increasingly active research area. For instance, neural networks are leveraged to identify the dynamics of quadcopters [5], [16], and

design attitude controllers [11]. Reinforcement learning is also shown to be effective in control UAVs [44], [47], [31]. However, data-driven approaches usually require large computational power and customized flight stack designs [47].

Among the methods above, the optimization-based method shows great ease of use and scalability when further constraints are to be added to the original motion planning task. However, the straightforward way of inserting constraints to optimization tasks often requires constraint terms to be differentiable, while topology constraints are generally not eligible. To mitigate this challenge, we propose several methods in Part 2. In this chapter, a general method, called the iterative method, to solve well-formed differentiable nonlinear programming tasks that we utilized in our works will be introduced.

1.2 Iterative Method

In the optimization-based motion planning method, the planning task is first transformed into a linear or nonlinear programming form, then a solver is assigned to solve the form. Numerous off-the-shelf NLP solvers exist, including SGD, ADAM, Newton's method, and the iterative method. Compared with other methods, the iterative method has great efficiency and scalability and it is the extension of the Gaussian-Newton method [83]. Furthermore, we can modify it to let the trajectory of the agent deform continuously. This feature is helpful when we design motion planning methods with topology constraints. In this section, we show the composition of the iterative method and the process of extending it to solve energy or time optimal motion planning tasks, which will be used in our works later.

1.2.1 On the Time-discretization and Linearization of Nonlinear Control Systems

This section first provides a review of the core idea of our previous contribution to the discretization and linearization of the nonlinear control systems with respect to the flow. For more details, readers are referred to [89].

Now, consider a continuous-time nonlinear system in the general form

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t) \in \mathbb{R}^n, \quad u(t) \in \mathbb{R}^m. \tag{1.1}$$

To discretize both control signal and time, the zero-order hold assumption, i.e., that

$$\forall t \in [k\Delta T, (k+1)\Delta T], \quad u(t) \equiv \bar{u}_k, \quad \bar{u}_k \in \mathbb{R}^m,$$

is often applied. By stacking x, u as ξ_1, ξ_2, we end up with an "autonomized" system, which is represented as:

$$\frac{d}{dt}\begin{pmatrix}\xi_1 \\ \xi_2\end{pmatrix} = \begin{pmatrix} f(\xi_1, \xi_2) \\ 0 \end{pmatrix}, \quad \begin{pmatrix}\xi_1(0) \\ \xi_2(0)\end{pmatrix} = \begin{pmatrix} x(k\Delta T) \\ \bar{u}_k \end{pmatrix}.$$

In this way, particularly for the time interval $t \in [k\Delta T, (k+1)\Delta T]$, a new transformed autonomous system, abbreviated as $\dot{\xi} = F(\xi)$, is built. In more detail, the flow of this system is denoted by $(\Phi_t)_{t\in\mathbb{R}}$, where we simply have

$$\begin{pmatrix} x((k+1)\Delta T) \\ \bar{u_k} \end{pmatrix} = \Phi_{\Delta T}\Big(\begin{pmatrix} x(k\Delta T) \\ \bar{u}_k \end{pmatrix} \Big).$$

By directly truncating the bottom states associated with the u-dynamics, we obtain a constructive formulation of the discrete-time states evolution for the original control system.

1.2.2 Joint Linearization, Discretization and Numerical Integration for Nonlinear Systems via Taylor Series Methods

In this subsection, we introduce a fundamental yet novel approach for the numerical solution of the original differential equations by examining the Taylor series of its solution. More specifically, we consider the expansion of the state around the particular time point t

$$\xi(t+h) = \xi(t) + h\dot{\xi}(t) + \frac{h^2}{2}\ddot{\xi}(t) + \frac{h^3}{3!}\dddot{\xi}(t) + \dots \tag{1.2}$$

to some desired order $p \in \mathbb{N}$. Evidently, a higher value of p theoretically gives a more accurate approximation of $\xi(t+h)$, but introduces more computational complexity at the same time.

To get over this dilemma we can simply recognize the composited derivative denoted as $\Psi_q : \mathbb{R}^n \to \mathbb{R}^n$, regardless the order of the derivative q, is just a vector field. For instance, $\Psi_2 : \xi \mapsto J(F)|_\xi F(\xi)$ is another vector field like $\Psi_1 : \xi \mapsto F(\xi)$. Keep this in mind, it is straightforward to see that the Jacobians Ψ_q obey the following functional recursion

$$\Psi_{k+1} = J(\Psi_k)F, \qquad \Psi_0 = \mathrm{Id}, \tag{1.3}$$

where $\mathrm{Id} : \xi \mapsto \xi$ denotes the identity operator. To derive the symbolic representation of Ψ_k in (1.3), merely a (differentiable) analytic description of the vector field F is required. Moreover, there are many available symbolic differential solving tools can be utilized to avoid tedious hand-work. Eventually, the exact representation of ξ at time point $t+h$ and the corresponding approximation is given by

$$\xi(t+h) = \sum_{q=0}^{\infty} \frac{h^q}{q!} \Psi_q(\xi(t)) \approx \sum_{q=0}^{p} \frac{h^q}{q!} \Psi_q(\xi(t)). \tag{1.4}$$

Having done so, the numerical integration up to the any desired order p is achieved.

Another great features of the proposed method is that the procedure for computing the Taylor series coefficients also directly carries out the Jacobians of the flow as a by-product. It can be seen by denoting ϕ_h as the time-h-map of the autonomous system $\dot{\xi} = F(\xi)$, where h is the time step. Because $\xi(t+h) = \phi_h(\xi(t))$, we can also write

$$\phi_h(\xi) = \Psi_0(\xi) + h\Psi_1(\xi) + \frac{h^2}{2}\Psi_2(\xi) + \dots. \tag{1.5}$$

Thus, we immediately reach the linearized flow

$$\left.\frac{\partial}{\partial \xi}\phi_h\right|_\xi = J(\Psi_0)|_\xi + h\, J(\Psi_1)|_\xi + \frac{h^2}{2}\, J(\Psi_2)|_\xi + \dots. \tag{1.6}$$

By recognizing the terms

$$J(\Psi_0) = I, \;\; J(\Psi_1) = J(F), \;\; J(\Psi_2) = J(J(F)F), \;\; \dots,$$

we can see that these coefficients have already been computed as an intermediate product during the process of obtaining the approximation of $\xi(t+h)$ in (1.4).

1.2.3 Sensitivities on The Time Step Size

We now extend this Taylor series-based approach to inspect the sensitivity of the dynamic flow in terms of the time size h, by merely recognizing h as another free variable alongside the state ξ of the flow, which is denoted as $\phi(\xi, h)$ and has the identical value defined in (1.5). Thus, the linearly approximated flow with respect to the time size is immediately obtained as

$$\left.\frac{\partial}{\partial h}\phi(\xi,h)\right|_h = \sum_{q=1}^{\infty}\frac{h^{q-1}}{(q-1)!}\Psi_q(\xi) \approx \sum_{q=1}^{p}\frac{h^{q-1}}{(q-1)!}\Psi_q(\xi). \tag{1.7}$$

Since every term Ψ_q is already given by previous calculations, the computation of sensitivities on the time window size is virtually for free.

The time size h can be viewed as the time step size for discrete-time systems. Accordingly, the linearly approximation in (1.7) provides an insightful perspective of systems: how the state trajectory evolution is influenced by the tiny variation of the time step size. This thought naturally leads to some interesting topics about solving time related-optimal control problems by manipulating the length of the time step for discrete-time systems. To this end, we will introduce an approach on providing optimal solutions for nonlinear systems in terms of joint energy and time consumption in the next section.

1.2.4 On Computing Energy-optimal Controls for Nonlinear Control Systems

In this section, we consider the practical problem of synthesizing energy-optimal control inputs for general nonlinear control systems by computational means. Our focus will be on

point-to-point minimum-energy steering laws that solve

$$\begin{aligned} \underset{u(\cdot)}{\text{minimize}} \quad & \|u(\cdot)\|^2 = \int_0^T \|u(t)\|^2 \, \mathrm{d}t \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)), \\ & x(0) = x_0, \\ & x(T) = x_{\text{target}}. \end{aligned} \tag{1.8}$$

Instead of directly addressing this continuous-time nonlinearly constrained quadratic program by obtaining the analytical solution, we propose a systematic iterative scheme that slices the entire optimization into unconstrined pieces.

1.2.5 Mapping From Control to State Trajectory

We now consider the discretized control system that is already efficiently computed from the time-continuous version by applying the Taylor series method discussed in 1.2.1. To simplify notations, we write $x_k := x(k\Delta T)$ and $u_k := u(k\Delta T)$. Thus we naturally obtain the discrete-time nonlinear control system as

$$x_{k+1} = \Phi_{\Delta T}(x_k, u_k), \tag{1.9}$$

where for further simplicity, we denote Φ as $\Phi_{\Delta T}$ in the remainder of this subsection. Given a perturbation in both states and inputs

$$(\tilde{x}_k, \tilde{u}_k) = (x_k + \delta x_k, u_k + \delta u_k),$$

the perturbed system evolves according to

$$\begin{aligned} \tilde{x}_{k+1} = \Phi(\tilde{x}_k, \tilde{u}_k) &= \Phi(x_k + \delta x_k, u_k + \delta u_k) \\ &\approx \Phi(x_k, u_k) + \left.\frac{\partial \Phi}{\partial x}\right|_{(x_k, u_k)} \delta x_k + \left.\frac{\partial \Phi}{\partial u}\right|_{(x_k, u_k)} \delta u_k. \end{aligned} \tag{1.10}$$

Since $\Phi(x_k, u_k) = x_{k+1}$ and $\delta x_{k+1} := \tilde{x}_{k+1} - x_{k+1}$, we can thus describe the evolution of the state perturbation in terms of the linear time-varying system

$$\delta x_{k+1} \approx \left.\frac{\partial \Phi}{\partial x}\right|_{(x_k,u_k)} \delta x_k + \left.\frac{\partial \Phi}{\partial u}\right|_{(x_k,u_k)} \delta u_k, \tag{1.11}$$

provided that $(\delta x_k, \delta u_k)$ are sufficiently small. The relevance of this system is due to the fact that (1.11) initialized with $\delta x_0 = 0$ describes the differences between the nominal state trajectory obtained from applying the nominal input signal $U = (u_0, u_1, \ldots, u_{N-1})$ and the perturbed state trajectory obtained from applying the perturbed input signal

$$U + \Delta U = (u_0 + \delta u_0, u_1 + \delta u_1, \ldots, u_{N-1} + \delta u_{N-1}),$$

in a linear manner, provided again that the pertubation is such that $(\delta x_k, \delta u_k)$ are sufficiently small so that the approximation is indeed meaningful. Then, denoting

$$A_k := \left.\frac{\partial \Phi}{\partial x}\right|_{(x_k,u_k)}, \quad B_k := \left.\frac{\partial \Phi}{\partial u}\right|_{(x_k,u_k)},$$

and unfolding the discrete-time system, we obtain the linear relationship between δx_k and ΔU given as

$$\begin{aligned}
\delta x_1 &\approx B_0 \delta u_0, \\
\delta x_2 &\approx A_1 \delta x_1 + B_1 \delta u_1 = A_1 B_0 \delta u_0 + B_1 \delta u_1, \\
&\vdots \\
\delta x_N &\approx A_{N-1} \cdots A_1 B_0 \delta u_0 \\
&\quad + A_{N-1} \cdots A_2 B_1 \delta u_1 \\
&\quad \vdots \\
&\quad + A_{N-1} B_{N-2} \delta u_{N-2} \\
&\quad + B_{N-1} \delta u_{N-1}.
\end{aligned}$$

This shows that if U is a nominal input that drives the system $x_{k+1} = \Phi(x_k, u_k)$ from x_0 to x_1 all the way to x_N, then appyling a slightly perturbed input $U + \Delta U$ will drive x_0 to

$$\tilde{x}_N = x_N + \delta x_N \approx x_N + \mathbb{H} \Delta U, \tag{1.12}$$

where

$$\mathbb{H} = \begin{pmatrix} A_{N-1}\cdots A_1 B_0 & A_{N-1}\cdots A_2 B_1 & \cdots & B_{N-1} \end{pmatrix}. \tag{1.13}$$

This linear mapping forms the basis of the later introduced iterative scheme to syntheses minimum energy steering laws for nonlinear control systems. Moreover, it can be efficiently obtained by applying the Taylor series-based linearizing technique shown in (1.6).

1.2.6 On Synthesizing Energy-optimal Controls

The established basis offers us a powerful tool for designing input signals, such that x_N is driven to be closer to the target x_{target}. However, given the fact that (3.8) is only valid locally, we have to admit that the target steering problem cannot be achieved in one shot but in a gradual manner. For the purpose of achieving both tasks of target steering and energy minimizing for nonlinear control systems, we simultaneously update two control components in each episode to solve both tasks, respectively.

Steering component

During each episode, by updating the steering component ΔU^s in the form of $U^+ \leftarrow U + \Delta U^s$, we desire to see x_N is steered closer to x_{target} than the one recorded from applying the original input signal U. Thanks to the linear mapping represented in (3.8), the solution of this problem can be obtained by equally solving the unconstrained quadratic program

$$\Delta U \| x_N + \mathbb{H}\Delta U - x_{\text{target}}\|^2 + \lambda \|\Delta U\|^2 \tag{1.14}$$

where λ is a regularization parameter that applies a penalty on the increment of δu_k at each time step, which in turn restricts the change of δx_k. The closed-form solution of (1.14) is given as

$$\Delta U^s = (\mathbb{H}^\top \mathbb{H} + \lambda I)^{-1} \mathbb{H}^\top (x_{\text{target}} - x_N). \tag{1.15}$$

Moreover, given the zero-initialized nominal input U and the regularization parameter λ, we may ensure that the updates ΔU^s are chosen in an intuitively economic way in each step. However, it cannot guarantee the energy optimality of the resulting control when the entire

Figure 1.1: Geometric explanation of the update of ΔU^e. The initialized zero input signal is represented as $\mathcal{O}$. Brown contours represent the level set of $\|x_N(U)\|$, where $x_N(U)$ maps U in the control space to X_N in the state space, while the darker one is defined as $\{U \,|\, x_N(U) = x_{\text{target}}\}$. The red solid line and red dash line represent $\ker(\mathbb{H})$ and $\mathrm{C}(\mathbb{H}^T)$, respectively. This 2D figure is normally projected from a much higher dimension of the control space.

iterative process finishes. To this end, we similarly consider updating another component in each episode to fulfill the optimal-energy requirement.

Energy-minimizing component

In order to not disturb the target steering procedure, the terminal state is desired to maintain the same position during the energy minimizing phase. Recalling that (3.8) describes the effect of a small perturbed input $U^+ \leftarrow U + \Delta U$ to the terminal state, we can see that any ΔU that lies in the kernel of $\mathbb{H}$ does not alter the terminal state. Accordingly, we can compute this component by solving the following constrained quadratic optimization problem

$$\Delta U^e \|U + \Delta U^e\|^2 \text{subject to} \quad \mathbb{H}\Delta U^e = 0 \quad \Rightarrow \Delta U^e \in \ker(\mathbb{H}), \tag{1.16}$$

with the additional implicit assumption that ΔU^e again be small. This linearly constrained quadratic program has a geometric interpretation that is illustrated in Figure 1.1. It shows a convenient heuristic way to solve the above linearly constrained quadratic program by merely tilting U towards the orthogonal projection of U onto the orthogonal complement of the kernel space of $\mathbb{H}$ (denoted as $\ker(\mathbb{H})$), which is just the column space of $\mathbb{H}^T$ (denoted as $C(\mathbb{H}^T)$). Such projection is given by

$$P_{C(\mathbb{H}^T)}U = \mathbb{H}^T(\mathbb{H}\mathbb{H}^T)^{-1}\mathbb{H}U.$$

This tilting can be achieved by forming a convex combination of U and $P_{C(\mathbb{H}^T)}U$, i.e.

$$\begin{aligned} U^+ &\leftarrow (1-\alpha)U + \alpha\mathbb{H}^T(\mathbb{H}\mathbb{H}^T)^{-1}\mathbb{H}U \\ &= U + \alpha(\mathbb{H}^T(\mathbb{H}\mathbb{H}^T)^{-1}\mathbb{H} - I)U. \end{aligned}$$

This evidently corresponds to a perturbation

$$\Delta U^e = \alpha(\mathbb{H}^T(\mathbb{H}\mathbb{H}^T)^{-1}\mathbb{H} - I)U, \tag{1.17}$$

where $\alpha \in [0, 1]$ can be adjusted to guarantee a sufficiently small magnitude of ΔU^e so as to satisfy the basic assumption that gives the linearization a valid meaning as an approximation. In this way, the constrained quadratic optimization problem can be relaxed to a non-constrained one and then solved by (1.17).

1.1 also provides a useful graphical intuition that for all $U \in \{U \mid x_N(U) = x_{\text{target}}\}$, the closer of U to the heading local optimum U^*, the more alignment of U with $C(\mathbb{H}^T)$ will be, which in turn means the less component of U lying in $\ker(\mathbb{H})$ defined as

$$P_{\ker(\mathbb{H})}U = U - P_{C(\mathbb{H}^T)}U. \tag{1.18}$$

Therefore, the magnitude of $P_{\ker(\mathbb{H})}U$ can roughly measure the distance from the current solution U to the U^*. Thus, $\|P_{\ker(\mathbb{H})}U\| = 0$ simply means U^* is arrived. Concisely, our proposed method continuously removes $P_{\ker(\mathbb{H})}U$, which has no contribution to the steering task but only wastes energy, by repeatedly updating ΔU^e. In addition, the energy minimization process should be terminated when the reduction of the control energy between two adjacent episodes is too little, which simply means the current solution is close enough to U^*.

In practice, to increase the efficiency of the energy minimizing process, a reasonably large α is preferred. However, the larger α simultaneously leads to larger ΔU^e, which causes the violation of the local linearization, so that the underlying terminal constraint may not hold anymore, i.e., δx_N has a slight bias towards $H\Delta U^e = 0$. Fortunately, it will not cause a noticeable problem by updating ΔU^s and ΔU^e in parallel as we proposed here, since the steering component ΔU^s will naturally compensate the drifting x_N caused by ΔU^e by driving the system closer to x_target. To this end, we need to ensure the drift never exceeds the steering progress contributed by ΔU^s for each episode, so that systems are guaranteed to be steered to the target in an accumulated manner. Therefore, one heuristic way to determine the value of α is to keep decreasing the value of α via $\alpha^+ \leftarrow \beta\alpha$, $\beta \in (0, 1)$, until $||x_N^i - x_{\text{target}}|| < ||x_N^{i-1} - x_{\text{target}}||$, where x_N^i denotes the computed terminal state in i^{th} episode. Once the target is arrived, same strategy can be used to ensure $||x_N^i - x_{\text{target}}|| \leq \epsilon$, where ϵ is small enough. This adopted scheme is quite similar to the backtracking line search used in the gradient method for optimization problems.

In summary, we come up with the following iterative scheme for solving the energy-optimal point-wise steering control problem. Detailed steps are shown in Algorithm 1. Together with the Taylor series-based method in 1.2.1, which computes $\mathbb{H}$ with high accuracy and efficiency, the iterative scheme forms a framework aiming on efficiently synthesizing energy-optimal controls for continuous-time nonlinear systems.

Algorithm 1 Point-to-point minimum-energy control

Require: x_0, x_{target}, initial input U, parameters λ, α

1. Apply the input U to the system and compute the matrices A_k, B_k along the resulting state trajectory $x_0, x_1, \ldots, x_N$ and compute

$$\mathbb{H} = \begin{pmatrix} A_{N-1} \cdots A_1 B_0 & A_{N-1} \cdots A_2 B_1 & \cdots & B_{N-1} \end{pmatrix}.$$

2. Compute the incremental updates

$$\begin{aligned} \Delta U^s &= (\mathbb{H}^\top \mathbb{H} + \lambda I)^{-1} \mathbb{H}^\top (x_{\text{target}} - x_N) \\ \Delta U^e &= \alpha (\mathbb{H}^T (\mathbb{H}\mathbb{H}^T)^{-1} \mathbb{H} - I) U \end{aligned}$$

3. Update α until the steering process is valid.
4. Update control input via $U^+ \leftarrow U + \Delta U^s + \Delta U^e$.
5. Repeat until $\|x_N - x_{\text{target}}\|$ falls below a desired tolerance and the decrease of $\|U\|^2$ is too little.

1.2.7 Comparison Between Iterative Method and Previous Approaches

Both the newly proposed and the previous optimal control syntheses techniques [87] share the same analyzing basis (utilizing $\mathbb{H}$ to construct the mapping from control to the terminal state) and the strategy of iteratively solving sliced optimizations. But their underlying logic of solving sub-optimization problems is distinct.

The previous approach splits the entire iterative scheme into two separate steps: first iteratively drives the system to a small ball centered at the target state, then gradually minimize the control energy while maintaining the terminal state inside the ball. The calculation of the first part is identical to the steering component ΔU^s (1.14). During the second step, due

to the fact that the terminal state is sufficiently close to the target, a linearly constrained quadratic program can be safely taken into account:

$$\begin{aligned} \underset{\Delta U}{\text{minimize}} \quad & \|U + \Delta U\|^2 + \gamma \|\Delta U\|^2 \\ \text{subject to} \quad & \mathbb{H}\Delta U = x_{\text{target}} - x_N, \end{aligned} \tag{1.19}$$

where γ is a regularization parameter that offers the same purpose as λ (1.14). It can be observed that by repeatedly updating ΔU, the optimal control laws can be eventually achieved. However, because the energy-minimizing process actives only after the target constraints have been satisfied, the total energy need to be minimized is expectedly high, which in turn requires more episodes to reach the energy optimum. In addition, the linear constraint implicitly limits the range of the amount of minimized energy per episode.

In contrast, our latest strategy involving the energy-minimizing component can be simultaneously deployed at the very beginning of the entire iterative scheme. Thus, the first computed control that steers the system to the target consumes comparatively low energy. Additionally, based on the statement in 1.2.6, more aggressive ΔU^e can be safely applied, which leads to the faster convergence to the optimum. Furthermore, we revealed the analytically valuable fact that the distance between the current result to the optimum can be roughly measured based on our new approach. To sum up, the method proposed here surpasses our previous one from the efficiency and analytical point of view.

1.3 Contributions, Structure and Publications

1.3.1 Contribution and Outline of the Thesis

The main focus of this thesis is on designing planning motion for general nonlinear dynamical systems with homotopy or homology class constraints leveraging optimization methods. The main contributions are summarized below.

- In Chapter 2, we introduce the optimization-based technique to deal with continuous curve following and waypoint following tasks. Optimization-based methods are known for easily handling differentiable targets and constraints. However, path following

targets and topology constraints are typically discrete and thus can not be inserted into optimization formulas directly. The work in this section is our initial trial for dealing with hybrid motion planning tasks with optimization-based methods, and the result suggests that the time complexity of the mixed integer approach is not affordable, even though the generated trajectory is acceptable. Hence in the following works, we avoid the usage of integer parts in the optimization formula.

- In Chapter 3, we introduce the so-called Auxiliary Energy Reduction (AER) technique, which is a gradient-based approach to solving motion planning problems with homotopy class constraints for system models with full-scale nonholonomic dynamics. The hallmark of our approach is that we first introduce virtual control terms to the original system dynamics that ensure that any preset state trajectory is dynamically feasible with respect to the new extended system. We then gradually shift the contribution of the artificial inputs to the actual original inputs by solving a sequence of associated quadratic programs. When the contribution of the artificial inputs has been fully removed, the preset trajectory will have been deformed to a trajectory of the same homotopy class that is now also feasible with respect to the original system. The practicality of our method is demonstrated in simulation examples for the Brockett integrator, the unicycle, and a 12-dimensional nonlinear quadcopter model.

- In Chapter 4, we introduce a novel optimal motion planning technique with 2-dimensional homotopy class constraints for general dynamical systems called HMHCC. The previous method in Chapter 3 can handle general dynamical systems and obstacles, but its efficiency is limited as usually tens of seconds is required for synthesizing. In order to thoroughly improve efficiency, HMHCC is proposed. We first initialize an optimal system trajectory regardless of obstacles and homotopy class constraints, and design an auxiliary obstacle trajectory for each obstacle such that the system trajectory belongs to the desired homotopy class regarding these auxiliary obstacle trajectories. During the procedure of deforming the auxiliary obstacle trajectory to the original counterparts, we propose a homotopy method based on nonlinear programming (NLP) such that the synthesized optimal system trajectories fulfill the aforementioned homotopy class constraints. The proposed method is validated with numerical results on two classic nonlinear systems with planar static and moving obstacles.

- In Chapter 5, we propose a novel method to address the shortest path problem with homology class constraints in both 2D and 3D environments. The HMHCC method

shows great efficiency, but its optimization formula limits it to 2D obstacles. Thus in this chapter, we explore a specific class of 3D obstacles that has the potential to be combined with HMHCC. We first define the phase change of the path with respect to 2D obstacles and then apply the same technique to a class of super-toroid obstacles compressed by an embedding map. To synthesize the shortest path, we leverage the visibility graph and the Value Iteration Algorithm (VIA). Finally, we demonstrate the effectiveness of our approach with various simulation examples.

- In Chapter 6, to address the practical issue of higher dimensional obstacles in Chapter 4, we introduce a method to embed specific 3D obstacles to 2D ones as in Chapter 5 and leverage the HMHCC method to synthesize the differentiable trajectories for the agent with homotopy class constraints. Consequently, the proposed method can handle moving obstacles and 3D obstacles while using an optimization framework, which gives it capabilities to apply in real-world scenarios.

1.3.2 Publications

All the work presented is in peer-reviewed journals (denoted as J) and conferences (denoted as C). The publications are listed below in the order of appearance. Submitted publications are marked with 's', while in progress but not submitted publications are marked with 'p'.

The research related to motion planning with homotopy and homology constraints, which concern the topological relations between planned trajectories and obstacles is covered in:

(C1) W. He, Y. Huang, and S. Zeng (2022). "Motion planning with homotopy class constraints via the auxiliary energy reduction technique". In: *2022 American Control Conference (ACC).* IEEE.

(C2) W. He, Y. Huang, J. Qie, and S. Zeng (2023). "Value Iteration Algorithm for Solving Shortest Path Problems with Homology Class Constraints". In: *IEEE Conference on Decision and Control (CDC).* IEEE.

(J1) W. He, Y. Huang, J. Wang, and S. Zeng (2022). "Homotopy Method for Optimal Motion Planning With Homotopy Class Constraints". In: *IEEE Control Systems Letters 7, pp. 1045–1050.* IEEE.

(J2p) W. He, Y. Huang, and S. Zeng (2024). "Homotopy Method for Optimal Motion Planning with Homotopy Class Constraints and 3-dimensional Super-toroid Obstacles".

The research related to data-driven control synthesis techniques is covered in:

(J3s) Y. Huang, M. Vu, W. He, and S. Zeng (2024). "Rapid Attitude Controller Design Enabled by Flight Data", submitted to *2024 Letters in Dynamic Systems and Control*, ASME.

The work related to data-driven or differential dynamic programming-based motion planning techniques is covered in:

(C3) Y. Huang, W. He, Y. Kantaros, and S. Zeng (2024). "Spatiotemporal Co-Design Enabling Prioritized Multi-Agent Motion Planning", submitted to *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE.

(C4) Y. Huang, W, He, and S. Zeng (2023). "A Differential Dynamic Programming-based Approach for Balancing Energy and Time Optimality in Motion Planning", in: *2023 Allerton Conference.* IEEE

Part II

Motion Planning with Homotopy and Homology Class Constraints

Chapter 2

A Unifying Approach to Continuous Curve Following and Waypoint Following via Block Coordinate Descent

2.1 Introduction

Optimization-based motion planning techniques can readily deal with differentiable constraints and optimization targets. However, the aforementioned method is not obvious when discrete parts exist in NLP. In this section, we investigate the optimization-based methods for path-following problems, which are typical hybrid motion planning tasks. This is also our initial trial to investigate the possible method in dealing with homotopy class constraints because these kinds of constraints are also discrete and not differentiable. As one of the fundamental areas within motion control problems [77], the Path Following (PF) problem has attracted extensive studies and found its various applying platforms ranging from unmanned aerial vehicles to unmanned submarine vehicles. In PF problems, one is asked to synthesize the control input to steer a system along a desired geometric path without the need to satisfy explicit time and speed constraints, in contrast to trajectory tracking. Due to the fact that the pre-designed geometric path is normally defined as a curve that is continuous in space, this PF problems can be further categorized as Continuous Curve Following (CCF) problems [32]. To solve such problems, many techniques, e.g., *Feedback Linearization* [2], which benefits from the linearization of the nonlinear systems around a certain region of the state space, *Non-Linear Guidance Law* [64], which adopt a simple geometric strategy that steers the system toward a virtual target point located on the path, and *Backstepping* [48], which achieves the zero tracking-error convergence based on the Lyapunov theory, have demonstrated their effectiveness with many successful applications. In addition, Model Predictive

Control [72], energy shaping [25] and the vector field method [86] are also leveraged to solve CCF problems.

In some other configurations of path guidance tasks, the characteristic motions are defined by a small number of waypoints rather than a geometric continuous curve, where the corresponding PF problems are categorized as Waypoint Following (WF). In contrast to the CCF which requires the desired curve to be aligned with the entire trajectory, the WF problems only expect each waypoint to be visited by the system in a pre-established order, where no constraint between two waypoints exists for the trajectory. Due to the more flexible motions for systems, such configuration is more efficient and thus more preferred [32] in some PF problems. The comparison between problems of CCF and WF is illustrated in Figure 2.1. Several related techniques have been proposed to solve waypoint tracking tasks, e.g., the proposed guidance design method steers the system along straight lines to pass through the preset waypoints [84], the work in [61] manages to find a time-optimal strategy for traveling between waypoints by leveraging Sequential Quadratic Programming (SQP), and in [32], a minimum-effort waypoint-following method is derived as the solution of a linear quadratic optimal control problem.

However, to the best of the authors' knowledge, there is no work managing to solve both problems of CCF and WF in a single framework. Hereby, we propose a novel direction in this section to shed some light on addressing these two PF problems via a unified approach. For the CCF problems, we need to first properly discretize the continuous curve so that we can describe it as a sequence of a large number of waypoints. Then we are able to unify these two distinct PF problems into a single Mixed Integer Nonlinear Programming (MINLP) problem that can be solved by our approach based on Block Coordinate Descent (BCD). Despite the seemingly naive pre-processing that simply converts the CCF problems into the WF problems, our approach solves them individually, as different problems vary the configurations of the formulated optimization problem. As the unified framework is proposed for WF and CCF problems, it is suggested that the mixed problems that involve both WF and CCF parts can also be solved in this framework.

More specifically, the advocated approach finds the optimal solution via iteratively and successively descending along two directions: finding the two optimal alignment functions, whose configurations rely on the PF problem to be solved, and tracking a sequence of waypoints. These two alignment functions are introduced to pair waypoints sequence and trajectory

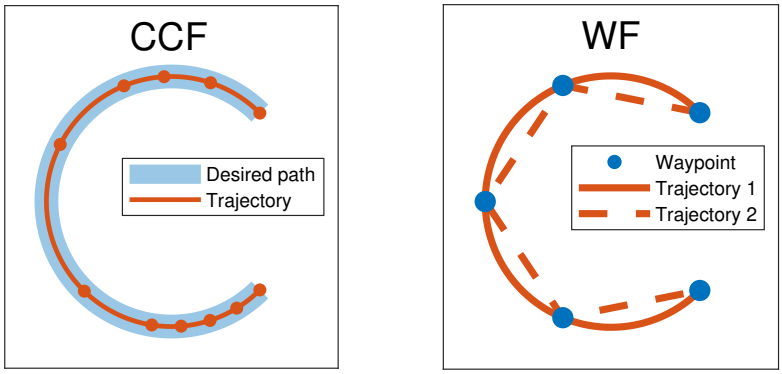


Figure 2.1: Two different sub-problems of the path following problem. **Left**: The goal of the CCF problem is that the trajectory, which is the orange curve with recorded position coordinates as the dots with uniform time intervals, is aligned with the pre-established path, the blue curve, regardless of the velocity of the system. **Right**: In WF problem, a valid trajectory is supposed to pass through the designed waypoints in a certain order. Thus, both trajectories are considered legitimate.

together, which allow us to easily quantify the mismatch between them. Moreover, given a fixed trajectory driven by a pre-determined control input, the two optimal alignment functions, which yield the best trajectory-waypoints alignment, can be analytically obtained by applying DP. In the other descending direction, by leveraging SQP, we are able to find a control input that steers the system to track the waypoints sequence with the alignment functions computed above. By iteratively solving these two sub-problems, we will gradually arrive at the optimal solution that accomplishes the overall PF task.

This section is organized as follows. Section 2.2 defines and formulates the problem. Then we introduce our approach for CCF and WF in Section 2.3. The applicability of the proposed algorithm is presented In Section 2.4. Finally, we conclude the section in Section 2.5.

2.2 Problem Formulation

For the purpose of the computational implementation, we assume the discrete time nonlinear dynamic system

$$x(k+1) = F(x(k), u(k)), \tag{2.1}$$

where state $x \in \mathbb{R}^n$, control input $u \in \mathbb{R}^m$ and F is continuously differentiable. Throughout this section, we consider the state trajectory $X = (x(0), x(1), \cdots, x(N))$ is given by driving the system (2.1) with an input sequence $U = (u(0), u(1), \cdots, u(N-1))$.

Let a sequence of $M+1$ numbers of waypoints $\{p(v)\}_{v=0}^{M}$, where $M \ll N$, be defined as a collection of points in $\mathbb{R}^p$ with a certain order. According to the objective of the WF problems, every waypoint needs to be successively visited by the trajectory, where, in our setup, we aim to associate every waypoint with a certain point of the trajectory. Meanwhile, we define a continuous curve as $\mathbf{p}_c \subset \mathbb{R}^p$. Considering the fact that both waypoints and continuous curves are utilized to regulate the characteristic motions of a system, we attempt to describe the continuous curve obeying the definition of the waypoints. In particular, we consider replacing a continuous curve $\mathbf{p}_c$ with a sequence of $M+1$ numbers of waypoints that is denoted in the same form as $\{p(v)\}_{v=0}^{M}$, whereas M has to be sufficiently large for the accurate discretization so that M is far greater than N. In such a setup, to fulfill the CCF problems, we expect every trajectory point is aligned with an associated point in the waypoints sequence.

It is noted that we always strive to map every point from the shorter sequence to that of the longer one, which is an invariant to the different problems we are treating. In order to unify the formulations of these two problems, we further introduce two injective functions. The first one is defined as

$$\lambda_p : \{0, \ldots, Z\} \to \{0, \ldots, M\}, \quad Z \in \mathbb{N}$$

to assist us in indexing some waypoints of the entire sequence, where the indexed waypoints are denoted as $\{p(\lambda_p(z))\}_{z=0}^{Z}$. Another one has the form of

$$\lambda_x : \{0, \ldots, Z\} \to \{0, \ldots, N\}, \quad Z \in \mathbb{N}$$

that is utilized to index some points of the trajectory which are described as $\{x(\lambda_x(z))\}_{z=0}^{Z}$.

Equipped with the definitions above, for the problem of CCF, we are able to quantify the mismatch between the trajectory and the desired continuous curve (represented in a sequence of waypoints) in the form:

$$\mathrm{d}_{CCF}(X, p, \lambda_p) \coloneqq \frac{1}{2}\sum_{z=0}^{N} \|Cx(z) - p(\lambda_p(z))\|^2,$$

where $C \in \mathbb{R}^{n\times p}$ is a projection matrix. For the feasibility of the PF problem, one has to further ensure that λ_p is non-decreasing, the boundary conditions $\lambda_p(0) = 0$, $\lambda_p(N) = M$, and that the increment of λ_p is relatively small to prevent a big jump of the path within a small trajectory segment. As for the WF problems, the mismatch between the trajectory points and the desired waypoints can be formulated as

$$\mathrm{d}_{WF}(X, p, \lambda_x) \coloneqq \frac{1}{2}\sum_{z=0}^{M} \|Cx(\lambda_x(z)) - p(z)\|^2,$$

where we have similar constraints of that λ_x is non-decreasing, $\lambda_x(0) = 0$ and $\lambda_x(M) = N$. Because both λ_p and λ_x are used to create alignments between the trajectory and the waypoints sequence, they are called alignment functions in the rest of this section.

Having done so, *both types of path following problems* can be addressed by solving a single MINLP

$$
\begin{aligned}
& & L(U,\lambda_x,\lambda_p) &= \frac{1}{2}\sum_{z=0}^{Z}\|Cx(\lambda_x(z)) - p(\lambda_p(z))\|^2 \\
& \underset{(U,\,\lambda_x,\,\lambda_p)}{\text{minimize}} & &+ \frac{1}{2}\sum_{n=0}^{N} u(n)^T R_u u(n) \\
& & &+ \frac{1}{2}(x(N) - x_{\text{target}})^\top Q_f (x(N) - x_{\text{target}}) \\
& \text{subject to} & & u_{\min} \le u(k) \le u_{\max}, \\
& & & x(0) = x_{\text{start}}, x(k+1) = F(x(k), u(k)), \\
& & & \lambda_x(0) = 0, \lambda_x(Z) = N, \lambda_x(z) \in \mathbb{N}, \\
& & & \lambda_p(0) = 0, \lambda_p(Z) = M, \lambda_p(z) \in \mathbb{N}, \\
& & & 0 \le \alpha_x \le \lambda_x(z+1) - \lambda_x(z) \le \beta_x, \\
& & & 0 \le \alpha_p \le \lambda_p(z+1) - \lambda_p(z) \le \beta_p,
\end{aligned}
\tag{2.2}
$$

where $z \in \{0, \cdots, Z\}$; $k \in \{0, \cdots, N\}$; $\alpha_x, \alpha_p, \beta_x, \beta_p$ are pre-defined hyper-parameters to regulate the increment of λ_x and λ_p. In addition, $x_{\text{target}} \in \mathbb{R}^n$ and $Q_f \in \mathbb{R}^{n\times n}$ terms are introduced to regulate the terminal state of the system, which are required when terminal angle constraints [73] or terminal velocity constraints [46] are considered. Meanwhile, we have

$$
\begin{aligned}
Z = N, \quad & \alpha_x = \beta_x = 1, \\
Z = M, \quad & \alpha_p = \beta_p = 1,
\end{aligned}
$$

for the problems of CCF and WF, respectively. Figure 2.2 illustrates different configurations of alignment functions in different problems.

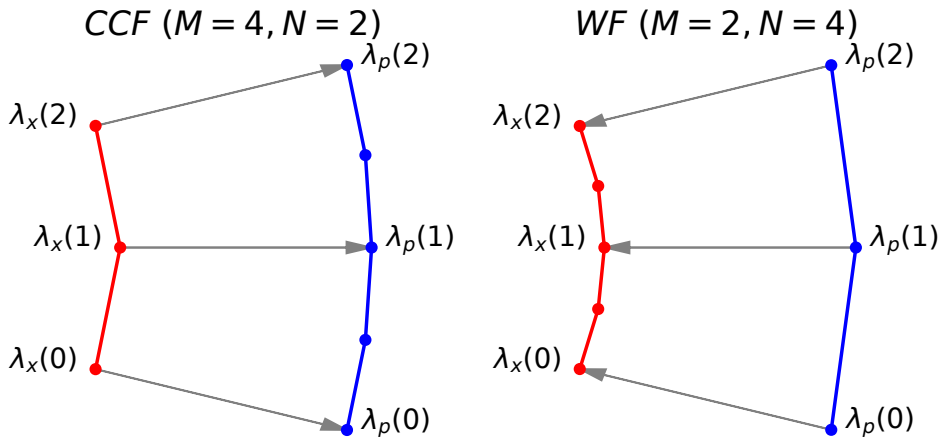


Figure 2.2: Graphical representation of the alignment functions in different tasks. The injective functions λ_x and λ_p are utilized to index some points of trajectories (in the red curves) and waypoints sequence (in blue curves), respectively, and thus pair these points together. It is noted that, λ_x and λ_p are forced to be identity functions in CCF and WF problems, respectively.

2.3 Path Following by Block Coordinate Descent

Directly solving MINLP (2.2) is challenging, as the input sequence U and the alignment function λ_x are mutually correlated. In order to make the problem more tangible and manageable, we propose an algorithm to decouple decision variables by leveraging the idea of BCD. More specifically, the proposed methodology contains two consecutive steps in each iteration. Step 1 is to find out the optimal alignment given a fixed control input, which will be introduced in Section 2.3.1. Section 2.3.2 describes Step 2 which is to update control inputs to minimize the mismatch of the trajectory-waypoints alignment, while keeping the alignment functions fixed. A summary of our approach will be provided in Section 2.3.3.

2.3.1 Optimal Alignment via Dynamic Programming

Once the control input U^* is given, the trajectory of the system X driven by U^* is determined and fixed, where we assume that $x(0)$ is known. Meanwhile, the optimization problem that

solely relies on the alignment functions can be reformulated as

$$\underset{\lambda_x,\,\lambda_p}{\text{minimize}} \quad L_{U^*}(\lambda_x,\lambda_p)=\frac{1}{2}\sum_{z=0}^{Z}\|Cx(\lambda_x(z))-p(\lambda_p(z))\|^2, \tag{2.3}$$

which has the same constraints as (2.2). Problem (2.3) is to find the optimal alignment functions that match the trajectory $\{x(k)\}_{k=0}^{N}$ and the waypoints sequence $\{p(v)\}_{v=0}^{M}$ the best, and can be solved by addressing dependent subproblems, where we attempt to find the best alignment between the subsequence of $\{x(k)\}_{k=0}^{N}$ and the subsequence of $\{p(v)\}_{v=0}^{M}$. Likewise, the problems of finding the best alignments between these subsequences can also be broken down into aligning subsequences of themselves. Due to the existing recursive pattern, we can start from solving the "smallest" subproblems, i.e., aligning $\{x(k)\}_{k=0}^{0}$ and $\{p(v)\}_{v=0}^{0}$, and build up to solve problem (2.3) eventually, which is referred to as DP. To this end, we need to store the distance of every possible pair of trajectory point and waypoint, as well as the information of the optimal value of each subproblems.

In particular, we consider building a local loss matrix $\in \mathbb{R}^{(M+1)\times(N+1)}$ with entries

$$_{i,j}=\frac{1}{2}\|Cx(j)-p(i)\|^2,$$

which stores the squared distance between the current trajectory to the desired waypoints. Based on , an accumulated loss matrix $\in \mathbb{R}^{(M+1)\times(N+1)}$ can be formed as

$$\begin{aligned} _{0,0} &= _{0,0},\\ _{i,j} &= \min_{(r_i,r_j)\in R_{i,j}}\{_{r_i,r_j}\}+_{i,j}, \end{aligned}$$

where $R_{i,j}=\{(r_i,r_j)|\alpha_p\le i-r_i\le\beta_p,\alpha_x\le j-r_j\le\beta_x\}$. For those elements whose indices are out of range, i.e., $i<0$, $j<0$, $i>M$ or $j>N$, we treat the corresponding $_{i,j}=+\infty$.

Proposition 1 *Each entry of $_{i,j}$ represents the minimum loss of matching a subsequence of waypoints $\{p(v)\}_{v=i}^{0}$ and a subsequence of trajectory $\{x(k)\}_{k=j}^{0}$ for all feasible alignment with the constraints of end-to-end alignment and the limited increment described in (2.2), where any $_{i,j}$ with infinity value simply means the corresponding problem is infeasible.*

Proof: For the index out of bound, there is no feasible solution, thus the loss value is $+\infty$. For $_{0,0}$, there is only one feasible alignment $\lambda_x(0)=\lambda_p(0)=0$ with the loss value $M_{0,0}$. Then consider an arbitrary entry $_{i,j}$ and the length of the corresponding alignment denoted as z,

the constraints of the problem suggest $\alpha_p \leq i - \lambda_p(z-1) \leq \beta_p$ and $\alpha_x \leq j - \lambda_x(z-1) \leq \beta_x$, thus the minimum loss of the corresponding sub-problem is $_{i,j} = \min_{(r_i,r_j)\in R_{i,j}}\{_{r_i,r_j}\} + _{i,j}$. Moreover, according to the constraints of either CCF or WF, we have $\alpha_x + \alpha_p > 0$, which suggests $i+j > r_i+r_j$ for all $(r_i, r_j) \in R_{i,j}$. Therefore, the proposed algorithm can recurrently evaluate $_{i,j}$ bottom-up. ■

An auxiliary directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is introduced to store the information of indices for the optimal alignment. More specifically, if $_{ij}$ is finite, we add (i, j) to $\mathcal{V}$ and add the directed link $(i, j) \to (\hat{r}_i, \hat{r}_j)$ to $\mathcal{E}$, where $(\hat{r}_i, \hat{r}_j) = \operatorname{argmin}_{(r_i,r_j)\in R_{i,j}}\{_{r_i,r_j}\}$. Having done so, the optimal alignment can be obtained by searching in $\mathcal{G}$:

$$\begin{aligned} &(\lambda_x^*(Z), \lambda_p^*(Z)) = (N, M), \\ &(\lambda_x^*(z+1), \lambda_p^*(z+1)) \to (\lambda_x^*(z), \lambda_p^*(z)) \in \mathcal{E}. \end{aligned} \tag{2.4}$$

Corollary 2.3.1 *The solution λ^* found in (2.4) is the optimal solution of the optimization problem (2.3).*

Proof: Given a preset path and determined trajectory, the associated optimal loss is shown in $_{M,N}$ which relies on other entries of , where $\mathcal{G}$ provides the information of which alignment of the sub-problem gives that optimal value and thus should be chosen. By iterating in such a way back to the starting point of the trajectory, the optimal alignment $(\lambda_x^*, \lambda_p^*)$ is constructed and is thus the optimal solution of the optimization problem (2.3). ■

According to the method proposed above, the time and the space complexity of building matrices , are $O(MN)$. One way to increase efficiency is to simply omit the computation of the entries whose indices conflict with the constraints of the alignment functions. More specifically, the computation of the entry of and with index (i, j) that belongs to the union of the following sets can be ignored

$$\begin{aligned} \mathcal{S}_1 &= \{(i,j) |\ \lfloor i/\alpha_x \rfloor < \lceil j/\beta_p \rceil\}, \\ \mathcal{S}_2 &= \{(i,j) |\ \lceil i/\beta_x \rceil > \lfloor j/\alpha_p \rfloor\}, \\ \mathcal{S}_3 &= \{(i,j) |\ \lfloor (N-i)/\alpha_x \rfloor < \lceil (M-j)/\beta_p \rceil\}, \\ \mathcal{S}_4 &= \{(i,j) |\ \lceil (N-i)/\beta_x \rceil > \lfloor (M-j)/\alpha_p \rfloor\}. \end{aligned} \tag{2.5}$$

The corresponding constraint window is similar to Itakura Parallelograms [39] in Dynamic Time Warping.

2.3.2 Optimal Input via Sequential Quadratic Programming

In this subsection, given fixed alignment functions λ_p and λ_x, we propose an approach to compute the optimal control signal in an iterative scheme. Because $Q_f = R_f^\top R_f$ and $R_n = R_s^\top R_s$ in (2.2) are symmetric and positive semi-definite, which transforms problem (2.2) into a more compact form:

$$\begin{aligned} \underset{U}{\text{minimize}} \quad & L_{\lambda^*}(U) = \frac{1}{2}\|R(U)\|^2 \\ \text{subject to} \quad & u_{\min} \leq u(k) \leq u_{\max}, \end{aligned} \tag{2.6}$$

where

$$R(U) = \begin{bmatrix} Cx(\lambda_x^*(0)) - p(\lambda_p^*(0)) \\ \vdots \\ Cx(\lambda_x^*(Z)) - p(\lambda_p^*(Z)) \\ R_f(x(N) - x_{\text{target}}) \\ R_s u(0) \\ \vdots \\ R_s u(N) \end{bmatrix},$$

and the trajectory X is obtained by steering the system (2.1) with the input U. Problem (2.6) can be solved by Sequential Quadratic Programming (SQP). More specifically, we consider solving the following optimization problem iteratively

$$\begin{aligned} \underset{\Delta U}{\text{minimize}} \quad & \nabla_U L_{\lambda^*}^\top \Delta U + \frac{1}{2}\Delta U^\top \nabla_U^2 L_{\lambda^*} \Delta U \\ \text{subject to} \quad & u_{\min} \leq u(k) + \delta u \leq u_{\max}, \end{aligned} \tag{2.7}$$

where $\nabla_U L_{\lambda^*}$ is the gradient vector, and $\nabla_U^2 L_{\lambda^*}$ is the Hessian matrix. For nonlinear programs with a least-squares objective function like problem (2.6), the Hessian matrix can be approximated using Gauss-Newton method for the purposes of reasonable time complexity [12]. Hereby, we arrive at

$$\nabla_U^2 L_{\lambda^*} \approx \frac{\partial R}{\partial U}^\top \frac{\partial R}{\partial U} + \gamma I, \tag{2.8}$$

where γI is the regularization term in case the term of $\frac{\partial R}{\partial u}^\top \frac{\partial R}{\partial u}$ is not invertible. By solving quadratic program (2.7), the updated input is given by $U = U + \Delta U$.

In the following part, we will show how to compute the $\frac{\partial X}{\partial U}$, which is critical in calculation of $\nabla_U L_{\lambda^*}$. Given the discrete nonlinear system (2.1), a nominal control signal U and the corresponding nominal state trajectory X, discrete-time linearization of (2.1) can be obtained via perturbing the entire trajectory and control signals. The resulting dynamics of the perturbed state are given as

$$\delta x(k+1) \approx A_k \delta x(k) + B_k \delta u(k), \quad \delta x(0) = 0, \tag{2.9}$$

where $A_k = \frac{\partial F}{\partial x}(x(k), u(k))$ and $B_k = \frac{\partial F}{\partial u}(x(k), u(k))$ are Jacobian matrices of the flow with respect to the state and the input, respectively. Iterating (3.7) from $\delta x(1)$ to $\delta x(N)$, we arrive at

$$\begin{aligned}
\delta x(1) &\approx B_0 \delta u(0) \\
\delta x(2) &\approx A_1 B_0 \delta u(0) + B_1 \delta u(1) \\
&\vdots \\
\delta x(N) &\approx A_{N-1} \dots A_1 B_0 \delta u(0) + \dots + B_{N-1} \delta u(N-1).
\end{aligned}$$

It is noted that, these linear approximations are only valid for sufficiently small control perturbations, i.e., $\|\Delta U\| = \|(\delta u(0), \delta u(1), \dots, \delta u(N-1))\|$ sufficiently small. To condense the approximation above, the variation of the entire state trajectory caused by small ΔU can be rewritten as

$$\Delta X := \begin{bmatrix} \delta x(0) \\ \delta x(1) \\ \vdots \\ \delta x(N) \end{bmatrix} \approx \begin{bmatrix} H_0 \Delta U \\ H_1 \Delta U \\ \vdots \\ H_N \Delta U \end{bmatrix} = H \Delta U, \tag{2.10}$$

where

$$H := \begin{bmatrix} 0 & 0 & \cdots & 0 \\ B_0 & 0 & \cdots & 0 \\ A_1 B_0 & B_1 & & \vdots \\ \vdots & \vdots & & 0 \\ A_{N-1} \cdots A_1 B_0 & A_{N-1} \cdots A_2 B_1 & \cdots & B_{N-1} \end{bmatrix},$$

and the matrices H_i represent the ith horizontal block of H, respectively. The obtained matrix H is the desired term $\frac{\partial X}{\partial U}$.

2.3.3 Summary and Practical Setups

As discussed in Section 2.2, both problems of CCF and WF can be described in a unified MINLP with a loss function $L(U, \lambda_x, \lambda_p)$. Based on the previous results, we propose a BCD-based approach to solve this MINLP by considering the alignment functions λ_x and λ_p as a block of decision variables and the control input U as another one. Our approach then sequentially minimizes $L(U, \lambda_x, \lambda_p)$ in terms of each block of variables while keeping the other one fixed. The general method is summarized in Algorithm 2.

Algorithm 2 BCD for Path Following

Input: Initial input U, desired terminal state x_{target} and waypoints p.

1: Apply U to the system and store the trajectory X.
2: Update the alignment λ_x and λ_p by solving (2.4).
3: compute H as defined in (3.8).
4: Solve the quadratic program (2.7) for ΔU.
5. Update the control input $U = U + \Delta U$.
6. Repeat step $1 - 5$ until $L(U, \lambda_x, \lambda_p)$ converges.

In practice, carefully selected hyper-parameters are helpful to acquire better performance. For CCF tasks, we set $\alpha_p = 0, \beta_p = \lceil M/N \cdot \mu \rceil$, where μ is a hyper-parameter, enabling dynamical systems to stay in place and meanwhile not jump over a large portion of the path. For waypoint tracking tasks, we set $\alpha_x = \lceil N/M/\mu \rceil, \beta_x = \lceil N/M \cdot \mu \rceil$ to prevent aligning adjacent trajectory points to waypoints that are far from each other, which usually causes an unacceptable local minimum of the optimization problem. Moreover, when the system is initialized at x_{start}, our approach tends to ignore the first small portion of the alignment functions, which may lead to slow convergence. To mitigate this challenge, inspired by the observation that updating the control input with uniform alignment functions, which are defined as

$$\lambda_x^{\text{unif}}(z) = \lceil z/Z \cdot N \rceil, \ \lambda_p^{\text{unif}}(z) = \lceil z/Z \cdot M \rceil,$$

will steer the trajectory to spread uniformly and make use of whole length of alignment functions, we hereby propose Algorithm 3 to provide proper initialization of control inputs. Experiments suggest that the proposed initialization is effective as shown in Figure 2.5.

Algorithm 3 Proper Initialization of U

Input: Initial input U, desired terminal state x_{target}, waypoints p and the required iteration d.

1: Apply U to the system and store the trajectory X.
2: compute H as defined in (3.8).
3: Solve Quadratic Programming (2.7) for ΔU with the uniform alignment λ_x^{unif} and λ_p^{unif}.
4. Update the control input $U = U + \Delta U$.
5. Repeat step $1 - 4$ for d iterations.

2.4 Illustrative Examples

Throughout this section, the theoretical results are evaluated through intensive numerical simulation studies, where the proposed algorithm is applied to a quadcopter for both CCF and WF problems. The adopted quadcopter is modeled by a state space representation $\dot{x} = f(x, u)$ with 12 states and 4 inputs that represent the rotating speed of four motors [74]. The 4th Order Runge Kutta method is leveraged to transfer it to a discrete-time dynamical system. In the following tasks, the position parts of x_{start} and x_{target} are equal to the first and the last points of the path, respectively, while the rest of the entries are zero. We additionally set the time horizon to 6 seconds, the sampling rate to be 50 Hz, $u_{\min} = 0$, $u_{\max} = 700$, $Q_f = 10I_{12}$, $R_n = 10^{-6}I_4$,$\gamma = 0.0001 \cdot L(U, \lambda_x, \lambda_p)$ and $\mu = 2$.

Application to CCF Tasks: Given the quadcopter model above, we leverage the proposed method to find the input sequence that drives the drone to follow two different paths, a star-shaped path in XY-plane and a square-shaped one in three-dimensional space, where we utilize two sequences of 2000 and 1500 waypoints to represent these two continuous curves, respectively. For both tasks, our approach converges to the optimal solutions within 60 iterations, where the proper initialization of control inputs is adopted with $d = 5$. Figure 2.3 and Figure 2.4 illustrate the results of these two tasks, including trajectories in a two-dimensional view, three-dimensional view, control inputs, and alignment functions, where the pre-designed paths are shown to be followed well.

To illustrate the effectiveness of the initialization process for PF tasks, we consider the CCF task with the same setup as the one shown in Figure 2.3 except the time horizon is 10 seconds. In Figure 2.5, we compare the rates of convergence and the alignments of the last iteration with different initialization iterations. The results suggest that the initialization process is necessary for CCF and a few iterations are sufficient.

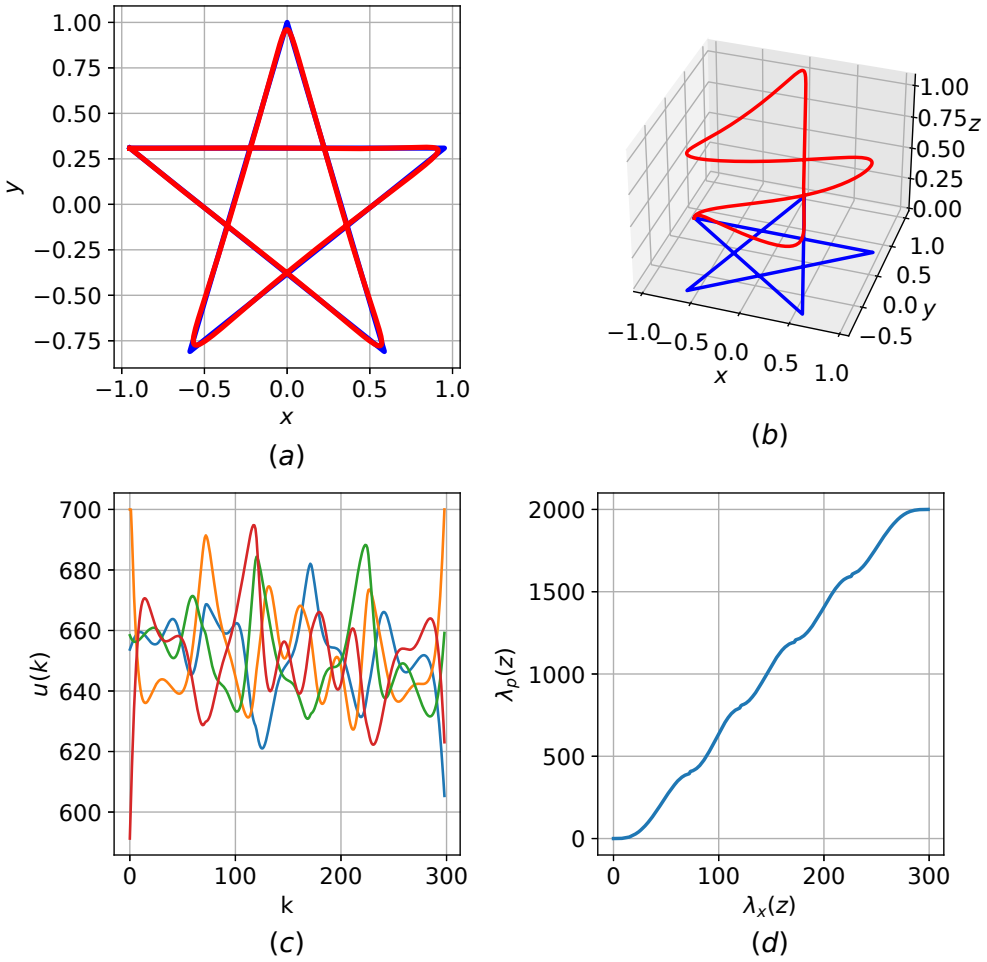


Figure 2.3: The quadcopter following a two-dimensional star-shaped continuous curve. **(a)**: The pre-established path and the resulting trajectory in XY-plane. **(b)**: The pre-established path and the resulting trajectory in three-dimensional space. **(c)**: Control inputs. **(d)**: Alignment of the trajectory and the continuous curve of the last iteration.

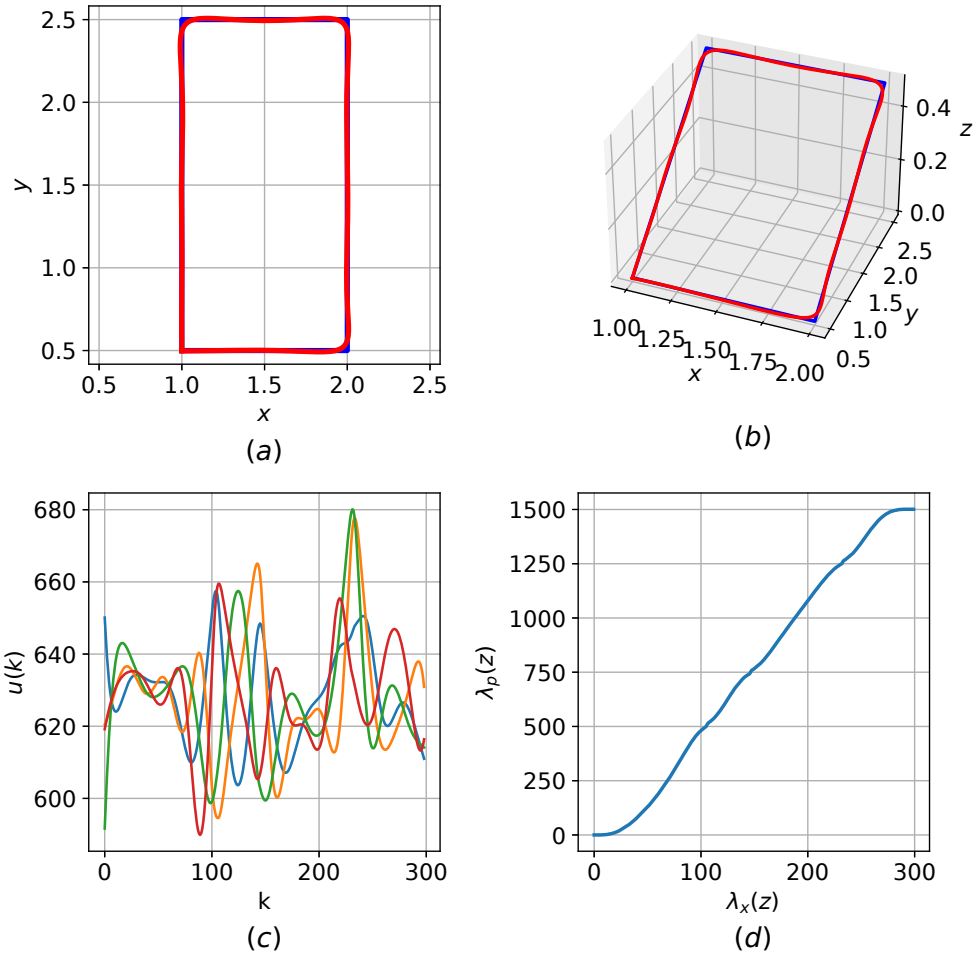


Figure 2.4: The quadcopter following the three-dimensional square-shaped continuous curve. **(a)**: The pre-established path and the resulting trajectory in XY-plane. **(b)**: The pre-established path and the resulting trajectory in three-dimensional space. **(c)**: Control inputs. **(d)**: Alignment of the trajectory and the continuous curve of the last iteration.

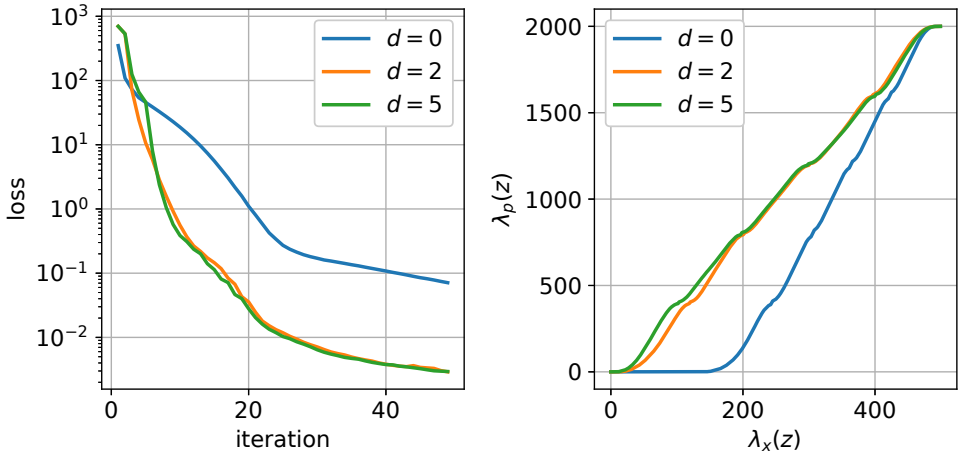


Figure 2.5: Comparison between different iterations of initialization in terms of the loss function and the alignment. **Left**: The value of $L(U, \lambda_x, \lambda_p)$ in each iteration. **Right**: Alignment of the trajectory and the continuous curve of the last iteration.

Application to WF Tasks: Here we leverage a two-dimensional eight-shaped and a three-dimensional star-shaped waypoint sequence to illustrate the performance of the proposed method on WF tasks. For both tasks, our approach converges to the optimal solutions within 60 iterations, where the proper initialization strategy is not adopted. Figure 2.6 and Figure 2.7 illustrate the results of these two tasks, including trajectories in two-dimensional view and three-dimensional view, control inputs, and alignment functions.

2.5 Conclusion

In this section, we introduced a novel approach based on block coordinate descent with Dynamic Programming and Sequential Quadratic Programming to address Continuous Curve Following (CCF) and Waypoint Following (WF) problems. The advocated approach first formulates both problems into a Mixed Integer Nonlinear Program, and then solves it by leveraging the idea of block coordinate descent. We further provided some discussions on the practical setups of the approach for better performance. We have additionally demonstrated the advantage and the practicability of the proposed method in a quadcopter model with various CCF and WF tasks. In addition, it is suggested that the mixed problems that involve

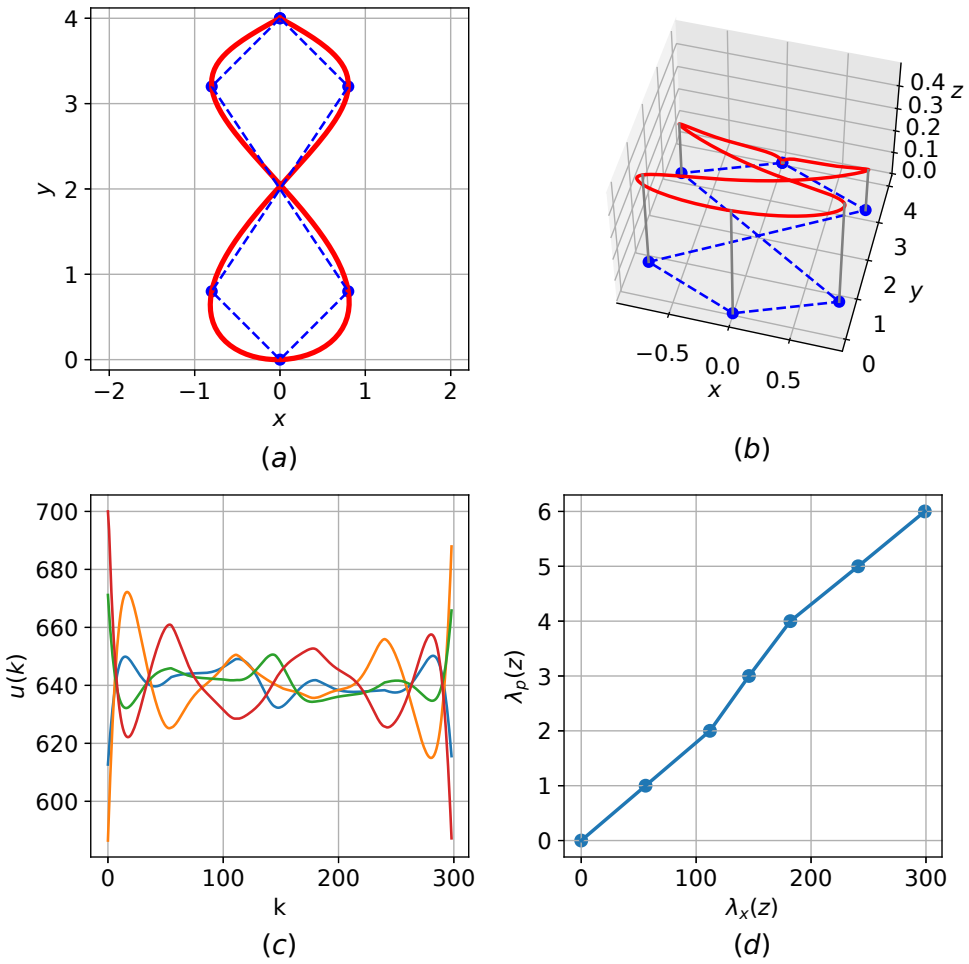


Figure 2.6: Quadcopter following the two-dimensional eight-shaped waypoints. Blue dash lines demonstrate the sequence of waypoints. **(a)**: Waypoints and the trajectory in XY-plane. **(b)**: Waypoints and the trajectory in 3-dimensional space. **(c)**: Control inputs. **(d)**: Alignment of the trajectory and the waypoints of the last iteration.

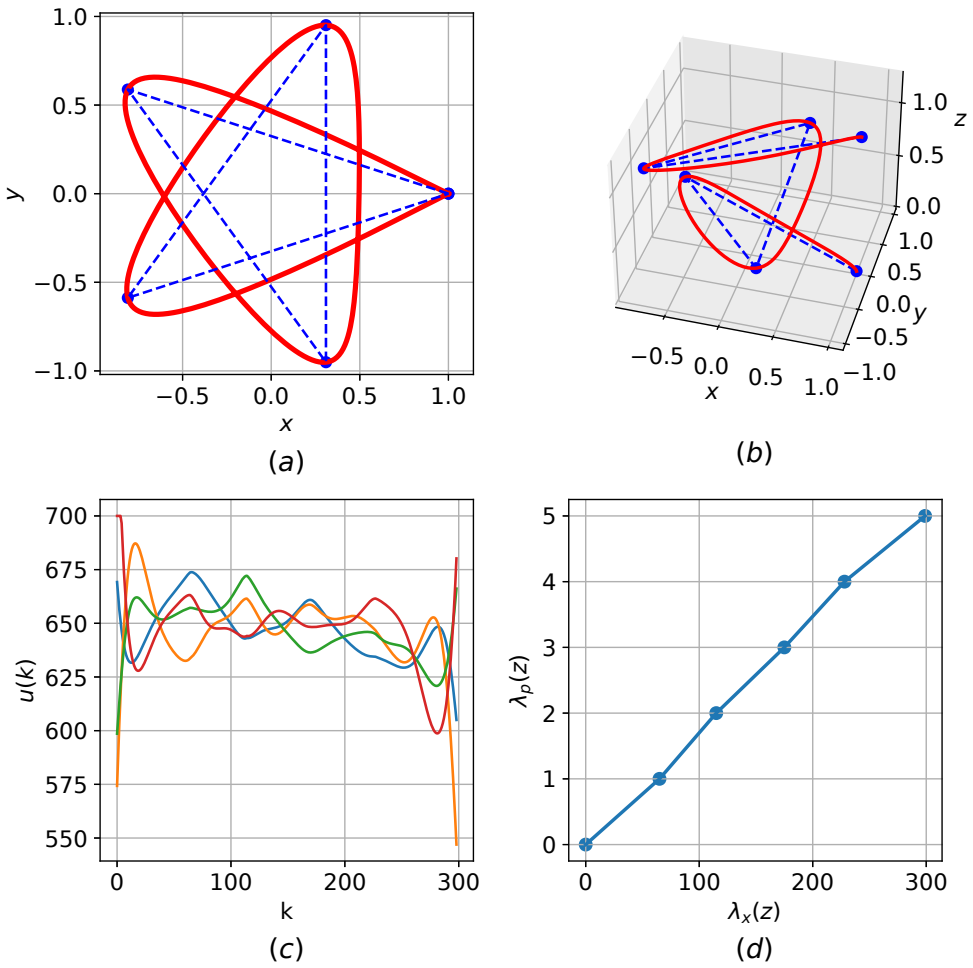


Figure 2.7: Quadcopter following a three-dimensional star-shaped waypoints. Blue dash lines demonstrate the sequence of waypoints. **(a)**: Waypoints and the trajectory in XY-plane. **(b)**: Waypoints and the trajectory in 3-dimensional space. **(c)**: Control inputs. **(d)**: Alignment of the trajectory and the waypoints of the last iteration.

both WF and CCF parts can also be solved in this framework, which we will investigate in future work.

In investigating this approach, we also realized that transforming hybrid optimization-based motion planning tasks to mixed integer nonlinear programming is a solution. However, the time complexity of MINLP is exponential and far from real-time. It also suggests that to deal with discrete constraints, the number of sample points needs to be minimized, which is not parallel with the requirement of high-fidelity trajectory generation. Hence we tend not to use mixed integral optimization techniques in our works of the following sections.

Chapter 3

Motion Planning with Homotopy Class Constraints via the Auxiliary Energy Reduction Technique

3.1 Introduction

The basic task of motion planning is to find a dynamically feasible system trajectory connecting a given starting point with a desired target point in a state space that possibly contains one or more obstacles. However, merely avoiding collisions is insufficient in some topology-sensitive tasks, e.g., a vehicle having to stay on the right side of the road. Hence, it is important to consider and distinguish among different homotopy classes for generated trajectories. For this consideration, [10] provides a way to classify homotopy classes in higher-dimensional space and proposes a search-based robot path planning method fulfilling topological constraints. Probabilistic roadmaps introduced in [40] are also capable of path generation under homotopy classes constraints. Furthermore, Gaussian process inference is leveraged in [49] to achieve online motion planning involving multiple homotopy classes. However these methods are again limited to robots with simple dynamics.

There are few prior works on motion planning that allow for the simultaneous considerations of full-scale nonlinear nonholonomic dynamics and homotopy class constraints. Recent works that made important contributions towards this direction are [7], [57] and [56]. These works introduce the so-called affine geometric heat flow, which is a partial differential equation that evolves an arbitrary differentiable path between an initial and final state to a path that meets additional constraints imposed on the problem. A potential drawback is the reliance of this approach on the numerical solution of the involved partial differential equation.

In this section, we propose a novel direction for addressing motion planning problems with homotopy class constraints that can be applied to general, possibly high-dimensional, nonlinear dynamical systems. The approach first adds an auxiliary control term to the original system, which turns a preset reference trajectory that is dynamically infeasible for the original system to a feasible one for the new extended system. Afterwards, the approach *gradually* (cf. [80], [81]) eliminates the influence of the auxiliary control term so as to let the original input slowly take over the control of the system. As a result, the dynamically infeasible trajectory is gradually deformed to a feasible one that the original system is fully capable to track. In addition, the advocated approach is able to preserve the homotopy class for generated trajectories throughout the iterative synthesis process.

The general idea underlying our proposed technique is reminiscent of the process by which young children first learn how to ride a bicycle, where in the beginning, the children often require external assistance, e.g., via parents actively holding or via a pair of training wheels to mitigate the challenging unstable dynamical component. After having gained sufficient experience under the externally facilitated steady conditions, the external assistance/forces can be gradually removed during subsequent phases of the learning process.

This section is organized as follows. Section 6.2, introduces the precise problem setup. Then we develop our method for generating feasible trajectories in Section 3.3, and its extension that concerns homotopy class constraints in Section 3.4. The applicability of the advocated technique is presented in Section 3.5. Finally, we conclude the section in Section 6.5.

3.2 Problem Formulation

We consider the general nonlinear dynamical system

$$\dot{x}(t) = f(x(t), u(t)), \tag{3.1}$$

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$. The feasible trajectory of the system above can be represented as a tuple (x, u), which satisfies the dynamical constraint (3.1) and terminal-point constraints $x(0) = x_{\text{start}}$, $x(T) = x_{\text{target}}$, where T is the terminal time. We denote $x \in \mathbb{X}_{\text{dynamic}}$, if there exists a corresponding admissible input u that steers (3.1) along x. We

take the parallel parking problem as an example to illustrate feasible and infeasible state trajectories as shown in 3.1.

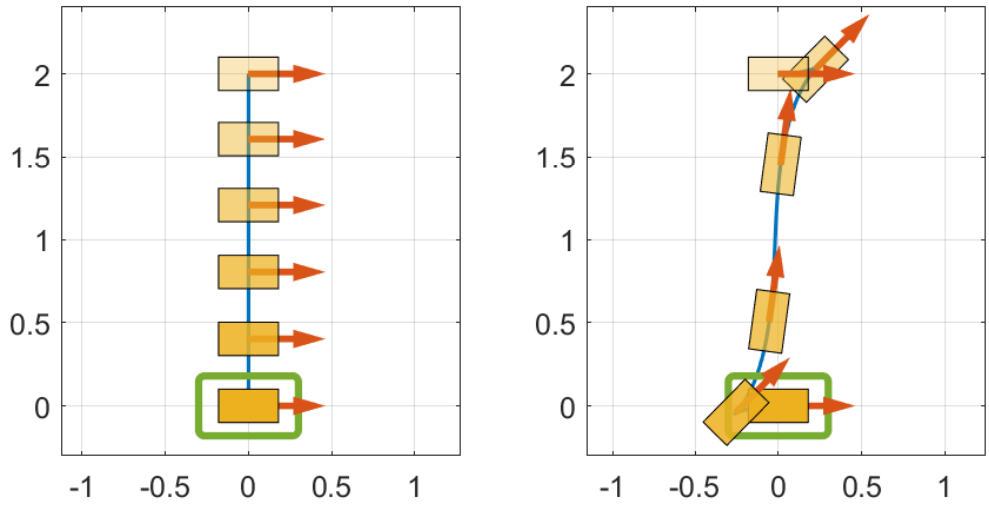


Figure 3.1: Parallel parking problem from the current car position $(0, 2)$ to the parking spot that is shown as the green rectangular and centered at $(0, 0)$. The trajectory on the left shown as the blue straight line is dynamically infeasible, which can be observed by the fact that the car's facing directions (shown as the red arrows) are perpendicular to the path. In contrast, the trajectory on the right is feasible, as the car's facing directions are tangent to the path.

Moreover, we denote $\mathcal{H}$ as the homotopy class which satisfies the homotopy class constraints described by a set of obstacles. The definition of homotopy class is given in Definition 1. A graphical illustration is presented in 5.1. A more detailed definition is referred to [10].

Definition 1 *Two trajectories belong to the same homotopy class if and only if they have identical start and target points and they can be smoothly deformed into one another without intersecting any obstacles.*

Given a nominal state trajectory, the objective of this work is to find a feasible trajectory $x \in \mathbb{X}_{\text{dynamic}} \cap \mathcal{H}$ and the corresponding admissible control u, with the property $x \in \mathcal{H}$ being preserved throughout the iterative computation process. It is noted that the initially chosen state trajectories are *not* required to be dynamically feasible.

Notation:

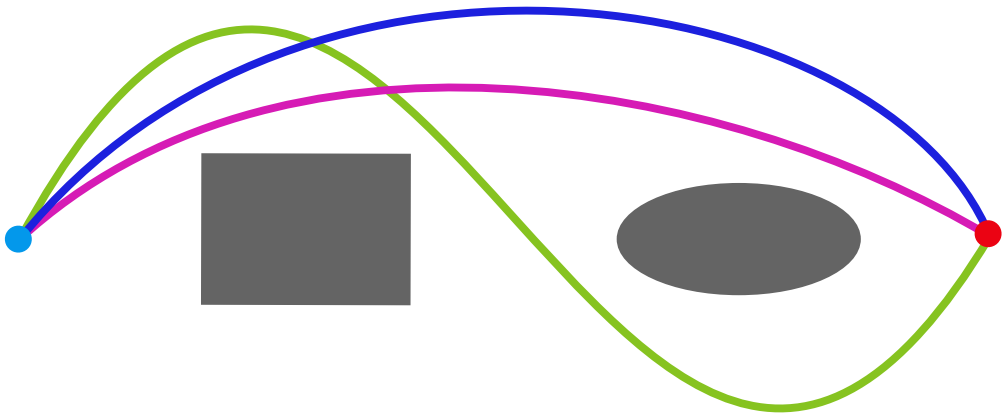


Figure 3.2: Three trajectories connecting identical starting and target points. According to the definition of homotopy class, the blue trajectory is homotopy equivalent with the purple one, but belongs to the different homotopy class of the green one.

1. For conciseness, $(x_0, x_1, \ldots, x_N)$ will represent the vector $(x_0^\top, x_1^\top, \ldots, x_N^\top)^\top \in \mathbb{R}^{n(N+1)}$.
2. By default, $\|\cdot\|$ stands for $\|\cdot\|_2$ in this section.

3.3 Motion Planning by Auxiliary Energy Reduction (AER)

In this section, we first introduce the general structure of the AER method in the absence of a cluttered environment. In Section 3.3.1, we transform the motion planning problem to an energy-minimization problem by virtue of the framework centered around the auxiliary control terms. The resulting nonlinear optimization problem to eliminate the virtually-added control energy is then solved in an iterative fashion, which is described in more detail in Section 3.3.2.

3.3.1 Auxiliary control inputs and the Extended System

For the purpose of the computational implementation, we assume that the continuous-time nonlinear system

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t) \in \mathbb{R}^n, \quad u(t) \in \mathbb{R}^m, \tag{3.2}$$

has been appropriately discretized into

$$x_{k+1} = F(x_k, u_k). \tag{3.3}$$

In order to track an arbitrary, even dynamically infeasible, nominal trajectory, we introduce an auxiliary (virtual) control component $\hat{u}_k$ to the discrete-time nonlinear system (4.7)

$$x_{k+1} = F(x_k, u_k) + \hat{u}_k, \tag{3.4}$$

where $x_k \in \mathbb{R}^n, u_k \in \mathbb{R}^m$, and $\hat{u}_k \in \mathbb{R}^n$. Moreover, for any pair of $X = (x_0, x_1, ..., x_N)$ and $U = (u_0, ..., u_{N-1})$, not necessarily feasible for the original system (4.7), there always exists a corresponding auxiliary control

$$\hat{u}_k = x_{k+1} - F(x_k, u_k), \tag{3.5}$$

such that (3.4) initialized at x_0 and driven with U and $\hat{U}$ traces out the given trajectory X in the state space.

In our Auxiliary Energy Reduction framework, the problem of generating feasible trajectories for the original system can be then cast as an energy minimization problem:

$$\begin{aligned} \underset{(U,\hat{U})}{\text{minimize}} \quad & \|\hat{U}\|^2 \\ \text{subject to} \quad & x_0 = x_{\text{start}}, \\ & x_N = x_{\text{target}}, \\ & x_{k+1} = F(x_k, u_k) + \hat{u}_k, \end{aligned} \tag{3.6}$$

where the start and end point of the given predetermined trajectory have been fixed. It is clear that if this optimization problem admits a solution where the auxiliary control term reaches zero, i.e., $\|\hat{U}\|^2 = 0$, the virtually-added input no longer impacts the system (4.7), i.e., we obtain the admissible control that steers the system (4.7) from x_{start} to x_{target} along with the corresponding feasible trajectory.

The purpose of our overall approach to first introduce an auxiliary term and then to work towards reducing it is to have a feasible starting point (for the extended system) in the computational optimal control setup, which in particular also allows for the fixing of known

and desired start and end points. Furthermore, in some instances, a user may already have a sense of how the desired trajectory of a system may end up looking like, and incorporate this information in terms of a useful prior design of the desired state trajectory, with the parts of the prior that are dynamically infeasible for the original system being outsourced to the auxiliary control terms. Another key is that we subsequently solve the nonlinear optimization problem (4.8) in an iterative fashion which is very efficient in that it results in a sequence of quadratic programs that need to be solved.

3.3.2 Auxiliary Energy Reduction

Given the discrete nonlinear system (3.4), a nominal state trajectory X, a nominal control signal U, and the derived auxiliary input signal $\hat{U}$, we obtain the discrete-time linearization of (3.4) via perturbing the entire trajectory (except x_0) and control signals, which yields the dynamics of the pertubation of the state as

$$\delta x_{k+1} \approx A_k \delta x_k + B_k \delta u_k + \hat{B}_k \delta \hat{u}_k, \quad \delta x_0 = 0, \tag{3.7}$$

where $A_k = \frac{\partial F}{\partial x}(x_k, u_k)$, $B_k = \frac{\partial F}{\partial u}(x_k, u_k)$, $\hat{B}_k = I_n$ are Jacobian matrices of the flow with respect to the state and the input, respectively. Unfolding (3.7), we arrive at

$$\begin{aligned}
\delta x_1 &\approx B_0 \delta u_0 + \hat{B}_0 \delta \hat{u}_0 \\
\delta x_2 &\approx A_1 B_0 \delta u_0 + B_1 \delta u_1 + A_1 \hat{B}_0 \delta \hat{u}_0 + \hat{B}_1 \delta \hat{u}_1 \\
&\vdots \\
\delta x_N &\approx A_{N-1} \dots A_1 B_0 \delta u_0 + \dots + B_{N-1} \delta u_{N-1} \\
&\quad + A_{N-1} \dots A_1 \hat{B}_0 \delta \hat{u}_0 + \dots + \hat{B}_{N-1} \delta \hat{u}_{N-1}.
\end{aligned}$$

It is noted that, these linear approximations are only valid for small control perturbations, i.e., $\|\Delta U\| = \|(\delta u_0, \delta u_1, \dots, \delta u_{N-1})\|$ and $\|\Delta \hat{U}\| = \|(\delta \hat{u}_0, \delta \hat{u}_1, \dots, \delta \hat{u}_{N-1})\|$ are sufficiently small. To condense the approximation above, the variation of the entire state trajectory

caused by small ΔU and $\Delta\hat{U}$ can be rewritten as

$$\Delta X := \begin{bmatrix} \delta x_1 \\ \vdots \\ \delta x_N \end{bmatrix} \approx \begin{bmatrix} H_1\Delta U + \hat{H}_1\Delta\hat{U} \\ \vdots \\ H_N\Delta U + \hat{H}_N\Delta\hat{U} \end{bmatrix} = H\Delta U + \hat{H}\Delta\hat{U}, \tag{3.8}$$

where

$$H := \begin{bmatrix} B_0 & 0 & \cdots & 0 \\ A_1B_0 & B_1 & & \vdots \\ \vdots & \vdots & & 0 \\ A_{N-1}\cdots A_1B_0 & A_{N-1}\cdots A_2B_1 & \cdots & B_{N-1} \end{bmatrix},$$

$$\hat{H} := \begin{bmatrix} \hat{B}_0 & 0 & \cdots & 0 \\ A_1\hat{B}_0 & \hat{B}_1 & & \vdots \\ \vdots & \vdots & & 0 \\ A_{N-1}\cdots A_1\hat{B}_0 & A_{N-1}\cdots A_2\hat{B}_1 & \cdots & \hat{B}_{N-1} \end{bmatrix},$$

and the matrices H_i and $\hat{H}_i$ represent the ith horizontal block of H and $\hat{H}$, respectively. Moreover, the corresponding state trajectory driven by the slightly drifted nominal control can thus be quantified as

$$X(U + \Delta U, \hat{U} + \Delta\hat{U}) \approx X(U, \hat{U}) + H\Delta U + \hat{H}\Delta\hat{U}. \tag{3.9}$$

With these insights, we can tackle the solution of the optimization problem (4.8) in a iterative manner where in each step of the iteration, we are looking at a drastically simpler quadratic program

$$\begin{aligned} \underset{(\Delta U, \Delta\hat{U})}{\text{minimize}} \quad & \|\hat{U} + \Delta\hat{U}\|^2 + \gamma_1\|\Delta U\|^2 + \gamma_2\|\Delta\hat{U}\|^2 \\ \text{subject to} \quad & x_N + H_N\Delta U + \hat{H}_N\Delta\hat{U} = x_{\text{target}}, \end{aligned} \tag{3.10}$$

for gradually reducing the auxiliary energy. Here γ_1, γ_2 are regularization parameters that penalize large values of $\|\Delta U\|$ and $\|\Delta\hat{U}\|$, which ensures the validity of the first order Taylor expansion applied in (3.8). By iteratively updating U and $\hat{U}$ by solving (3.10), the auxiliary control energy $\hat{U}$ will be steadily reduced. Furthermore, if we arrive at $\hat{U} = 0$, this would mean that the original input U has taken over full control of the system in steering the

state on a trajectory connecting x_{start} and x_{target}. The general structure of AER which is summarized as below.

Algorithm 4 Auxiliary Energy Reduction (AER)

Require: A nominal trajectory X connecting x_{start} and x_{target}, as well as a pair of nominal inputs $(U, \hat{U})$, such that $(U, \hat{U})$ steers the extended system along X.

1: Apply the input $(U, \hat{U})$ to the system and calculate H, $\hat{H}$.
2: Solve for $(\Delta U^*, \Delta \hat{U}^*)$ of the optimization problem (3.10).
3: Update the control input via $U \leftarrow U + \Delta U^*$ and $\hat{U} \leftarrow \hat{U} + \Delta \hat{U}^*$.
4: Repeat step $1-3$ until $\|\hat{U}\|^2 \leq \epsilon_{2,\text{tol}}$.

As the dimension of the input of the extended system $\dim(U) + \dim(\hat{U}) = m + n$ is larger than the dimension of the state, the extended system is an over-actuated system, and more than one pair of $(u_k, \hat{u}_k)$ can drive the system from x_k to x_{k+1}. To decrease the total iteration times needed when employing the AER method, an initial $\hat{U}$ with less energy is preferred. In the next part, we show how this may be achieved by manipulating the way U is initialized.

Without loss of generality, we focus on minimizing the energy of the initial auxiliary control term in the kth time step of the trajectory by considering (3.5) through choosing the initial input u_k according to $\min_{u_k} ||x_{k+1} - F(x_k, u_k)||^2$. Given a nominal trajectory, a nonlinear regression approach can be utilized to update u_k:

$$u_k \leftarrow u_k + \lambda B_k^{\dagger}(x_{k+1} - F(x_k, u_k)), \tag{3.11}$$

where λ is the step size which is small enough to ensure a steady convergence, and $B_k^{\dagger}$ is the left pseudo inverse of B_k. This procedure can be generalized to the entire time steps for properly initialization that is concluded in Algorithm 5.

Algorithm 5 Proper initialization of $U, \hat{U}$

Require: A nominal trajectory X from x_{start} to x_{target}.

1: Initialize $U = (u_0, ..., u_{N-1})$ as a sequence of small random numbers.
2: Calculate B_k for every k, and update u_k by (3.11) until converged.
3: Set the auxiliary control input via $\hat{u}_k = x_{k+1} - F(x_k, u_k)$, and $\hat{U} = (\hat{u}_0, ..., \hat{u}_{N-1})$.

We note that the AER method without optimizing the initial U is fast enough for simpler systems, such that the energy minimization step is not necessary in general, in which case step 2 of Algorithm 5 can be skipped.

3.4 Motion Planning with Homotopy Class Constraints

In this section, we illustrate how the AER method can be extended to preserve the homotopy class of generated trajectories when tackling motion planning problems involving cluttered environments and homotopy constraints. To this end, we integrate the AER method with two new components: anchors that are to identify the obstacles and the associated anchor loss that is added to the objective function for maintaining the topological properties of the trajectory.

Anchors are obstacles with regular shapes, such as points, lines and planes. In addition, obstacles with irregular shapes can be approximated by tightly arranging multiple anchors around their boundaries. An anchor is defined as a tuple (C, a), where $C \in \mathbb{R}^{p\times n}$ is the linear operator for dimension selection and rotation, and $a \in \mathbb{R}^p$. As a result, the distance between a point x and the anchor can be measured as $\|Cx - a\|$. Moreover, a set of anchors is defined as a collection of tuples: $\mathcal{A} = \{(C_1, a_1), ..., (C_M, a_M)\}$.

To keep a given trajectory $X(U, \hat{U}) = (x_0, x_1, ..., x_N)$ that is obtained by an extended system (3.4) starting from x_{start} and driven by inputs $(U, \hat{U})$ away from an anchor set $\mathcal{A}$, we introduce the so-called anchor loss $L(X, \mathcal{A}) \in \mathbb{R}^{M(N+1)}$ by utilizing the framework of barrier functions. Within our particular implementation, the anchor loss is defined to be logarithmic-like, whose $(i + kM)$th element is expressed as

$$L(X, \mathcal{A})_{i+kM} = \begin{cases} 0, & \text{if } \|C_i x_k - a_i\|^2 \geq \mu, \\ -\log(\|C_i x_k - a_i\|^2), & \text{otherwise,} \end{cases} \tag{3.12}$$

where (C_i, a_i) denotes the ith anchor of $\mathcal{A}$, $\mu \in (0, 1]$ and x_k represents the state at the kth time step of the trajectory. The logarithm distance of the form $-\log(d^2)$ is preferred here because its derivative will not vanish as $-d^2$ or expand too fast as $1/d^2$ when d approaches zero.

In order to prevent the trajectories from crossing through the anchor set during the trajectory updates, we need to ensure that there is no anchor standing inside the smooth deformation from $X(U, \hat{U})$ to $X(U + \Delta U, \hat{U} + \Delta\hat{U})$. To this end, we consider the following. If there is an intersection between (C_i, a_i) and the interpolated trajectory segment x_k to x_{k+1}, we have $\|C_i x_k - a_i\|^2 \leq \|C_i(x_{k+1} - x_k)\|^2$, which implies that there exists x_k, x_{k+1} and an

anchor $(C_i, a_i) \in \mathcal{A}$, such that $L(X, \mathcal{A})_{i+kM} \geq -\log(\|C_i(x_{k+1} - x_k)\|^2)$, where we assume that the sampling rate of the discretization from (3.2) to (4.7) is sufficiently fast so that $\|C_i(x_{k+1} - x_k)\|^2 \leq \mu$ holds for every k. In addition, by denoting $X^\alpha = X(U + \alpha\Delta U, \hat{U} + \alpha\Delta\hat{U}), \alpha \in [0, 1]$, we can leverage $X^0 \mapsto X^1$ to present a smooth deformation from $X(U, \hat{U})$ to $X(U + \Delta U, \hat{U} + \Delta\hat{U})$. We conclude that, as long as one can ensure

$$\|L(X^\alpha, \mathcal{A})\|_\infty < \min_{i,k} -\log(\|C_i(x_{k+1} - x_k)\|^2) = \Xi, \tag{3.13}$$

and $X_N^\alpha = X_N^0$ for all $\alpha \in [0, 1]$, the trajectory driven by updated inputs $X(U+\Delta U, \hat{U}+\Delta\hat{U})$ belongs to the same homotopy class of $X(U, \hat{U})$.

Proof: According to the definition of the anchor loss, $\|L(X^\alpha, \mathcal{A})\|^2 < \Xi^2$ implies the interpolated X^α will not intersect any obstacle in the state space. Therefore X^0 to X^1 presents a smooth deformation from $X(U, \hat{U})$ to $X(U + \Delta U, \hat{U} + \Delta\hat{U})$ without intersecting any obstacle, which indicates that they belong to the same homotopy class. For instance, as shown in Figure 3.3, the necessary condition of the trajectory $\hat{x}$ intersects the obstacle located at a is there exists a sample point $\hat{x}_k$, such that $\|C\hat{x}_k - a\| < \|C\hat{x}_k - C\hat{x}_{k+1}\|$, where C is the matrix that map the configure space of the agent to the work space. ■

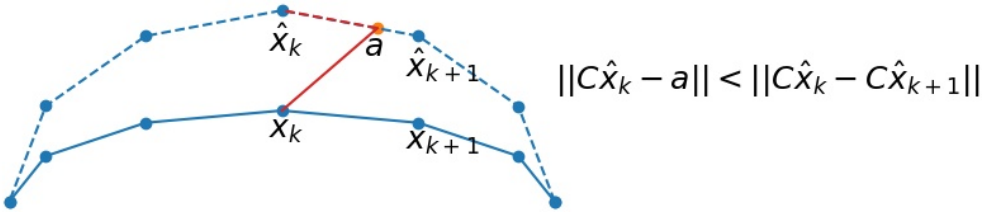


Figure 3.3: Case when blue trajectory intersect the obstacle marked as a.

It is natural to consider an optimization problem minimizing the norm of the anchor loss so as to fulfill the condition in (3.13). Before we take the further step towards the optimization formulation, we first investigate the linear approximation of the anchor loss with respect to $(U, \hat{U})$. Based on (3.8) and (3.12), the $(i + kM)$th row of the Taylor expansion's first-order term regarding $L(X, \mathcal{A})$ is expressed as

$$\Delta L(X, \mathcal{A})_{i+kM} = \begin{cases} 0, & \text{if } \|C_i x_k - a_i\|^2 \geq \mu, \\ M_{i+kM}\Delta U + \hat{M}_{i+kM}\Delta\hat{U}, & \text{otherwise,} \end{cases}$$

where

$$M_{i+kM} = \frac{-2(C_i x_k - a_i)^T C_i}{\|C_i x_k - a_i\|^2} H_k,$$
$$\hat{M}_{i+kM} = \frac{-2(C_i x_k - a_i)^T C_i}{\|C_i x_k - a_i\|^2} \hat{H}_k.$$

Having done so, the following quadratic program is considered for maintaining the auxiliary energy and decreasing the anchor loss simultaneously,

$$\begin{aligned} \underset{\Delta U}{\text{minimize}} \quad & \|L(X, \mathcal{A}) + M\Delta U\|^2 + \gamma \|\Delta U\|^2 \\ \text{subject to} \quad & x_N + H_N \Delta U = x_{\text{target}}, \end{aligned} \tag{3.14}$$

where $\gamma \geq 0$ is the regularization parameter.

We note that it is in general cumbersome to choose γ_1 and γ_2 in (3.10) for each iteration such that the condition (3.13) is satisfied. Instead, one could simply use the same γ_1 and γ_2 in Algorithm 4, and choose an appropriate update step size α to update the controls: for ΔU^* and $\Delta \hat{U}^*$ obtained from (3.10), we employ a line search strategy to find the largest update step size $\alpha^* \in (0, 1]$, such that $\|L(X^\alpha, \mathcal{A})\|_\infty < \xi$ for all $\alpha \in [0, \alpha^*]$, where $\xi \leq \Xi$ is a predetermined threshold. This strategy has been found to be particularly effective in dealing with highly nonlinear systems.

To summarize, by incorporating anchors, we endow AER method with the capability of preserving the topology class for the generated trajectories through out iterations. The overall procedure is summarized in Algorithm 6.

Algorithm 6 AER with anchors

Require: A nominal trajectory X connecting x_{start} and x_{target}, as well as a pair of corresponding inputs $(U, \hat{U})$.

1: Apply the input $(U, \hat{U})$ to the system and calculate $H, \hat{H}$.

2: Solve for ΔU^* and $\Delta \hat{U}^*$ of the optimization problem (3.10), and find the largest step size α^* using line search.

3: Update $U \leftarrow U + \alpha^* \Delta U$ and $\hat{U} \leftarrow \hat{U} + \alpha^* \Delta \hat{U}$.

4: Calculate $L(X, \mathcal{A})$ and M, update U according to (3.14).

5: Repeat step 4 until $\|L(X, \mathcal{A})\|^2 \leq \epsilon_A$.

6: Repeat step $1 - 5$ until $\|\hat{U}\|^2 \leq \epsilon_{2,\text{tol}}$.

3.5 Illustrative Examples

We first demonstrate the effectiveness of the proposed AER method on two well-studied benchmark platforms: Brockett integrator and a unicycle that operates on a plane. Then a more complex and highly nonlinear model of the quadcopter in 3-D space is evaluated.

3.5.1 Brockett Integrator Model

As a classical nonholonomic control system, the Brockett integrator

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ x_1 u_2 - x_2 u_1 \end{pmatrix}$$

has been widely studied.

We consider the setup in which $x_0 = (0, 0, 0)^\top$, $x_{\text{target}} = (0, 0, 1)^\top$, $T = 2$, and the preset state trajectory is set as a naive straight line between these two points. We set $\gamma_1 = \gamma_2 = 0.05\|\hat{U}\|^2$, and the sampling rate to be 20 Hz. By leveraging the geometric interpretation of the Brockett integrator in terms of the connection to the sector area of the curve in the xy-plane (cf. [88]), the value of x_3 is known to be a function of the area of the (x_1, x_2) curve. Therefore, this preset reference is clearly dynamically infeasible. 3.4 illustrates the deformation from the reference trajectory to a dynamically feasible one obtained by AER, where the x_0 and x_{target} are represented in red and blue dots, respectively.

3.5.2 Unicycle Model with Homotopy Class Constraints

The unicycle model is another popular nonholonomic testbed for illustrating path generation methods, cf. [57]. Its corresponding system dynamics are described by the following system of nonlinear differential equations

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ \phi \end{pmatrix} = \begin{pmatrix} v\cos\phi \\ v\sin\phi \\ \omega \end{pmatrix},$$

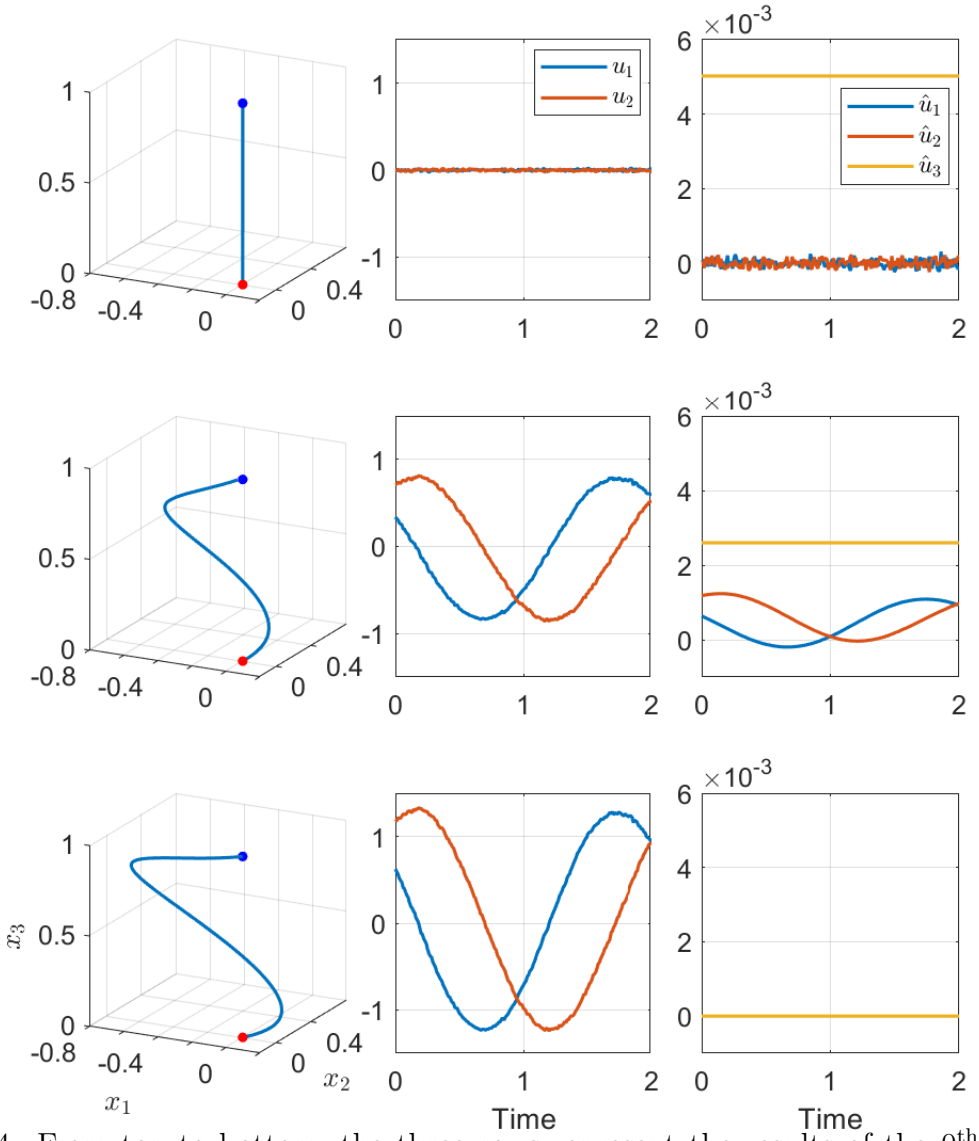


Figure 3.4: From top to bottom, the three rows represent the results of the 0^{th}, 40^{th} and 55^{th} iteration of the computation. In the left column, the state trajectory of the Brockett integrator evolves from being dynamically infeasible to being feasible. The corresponding control signals U and auxiliary input signals $\hat{U}$ are shown in the middle and the right column.

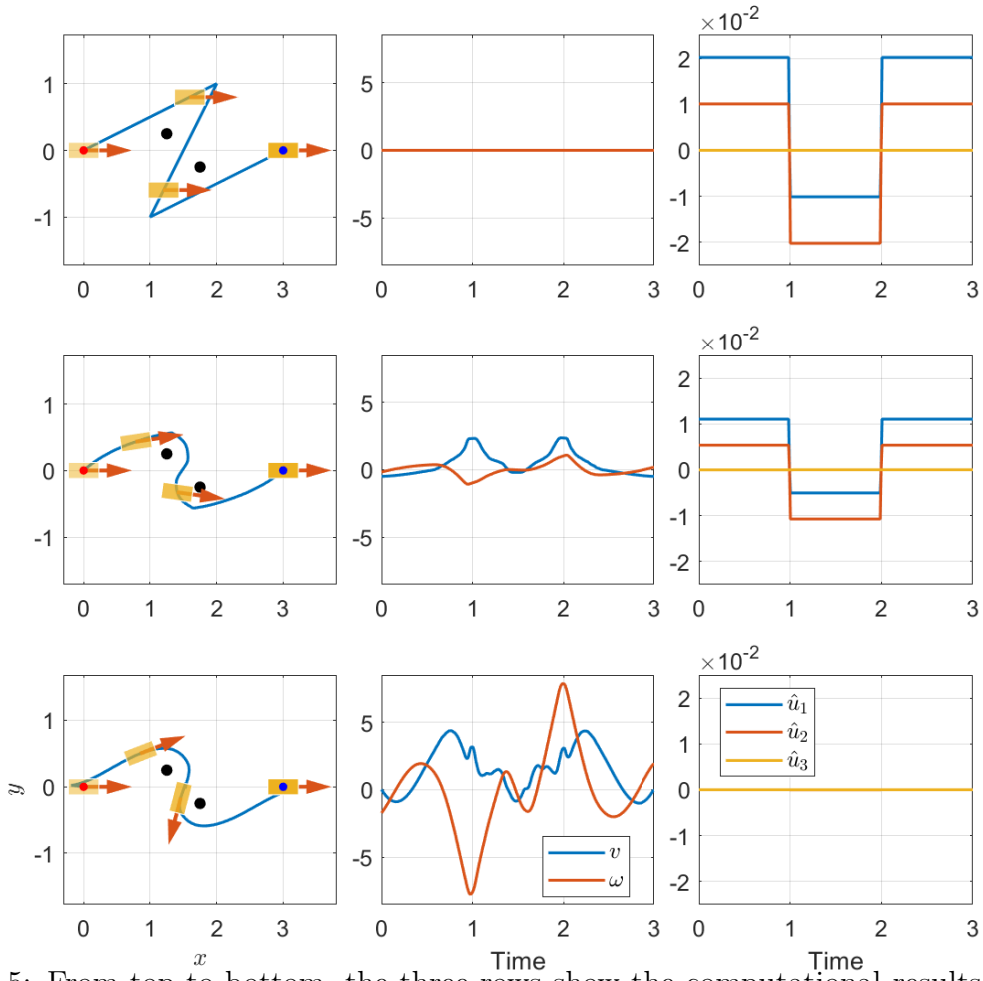


Figure 3.5: From top to bottom, the three rows show the computational results of the 0^{th}, 11^{th} and 42^{th} iteration, respectively. In the left column, the generated paths tend to be more dynamically feasible. Additionally, the corresponding control U and the virtually added input $\hat{U}$ are shown in the middle and the right column, respectively.

where ϕ represents the facing angle of the unicycle, v stands for the moving velocity, and ω denotes the steering velocity.

We set $x_0 = (0, 0, 0)^\top$, $x_{\text{target}} = (3, 0, 0)^\top$, $T = 3$. We set $\gamma_1 = \gamma_2 = 0.0001\|\hat{U}\|^2$, $\gamma = 0.005$, and the sampling rate of the discretization to be 20 Hz. In order to illustrate AER method's capability of keeping the consistency of homotopy class for generated trajectories, two anchors are placed at $(1.25, 0.25)$ and $(1.75, -0.25)$, respectively. 3.5 shows snapshots of the unicycle's movements within three representative iterations, as well as the corresponding input signals.

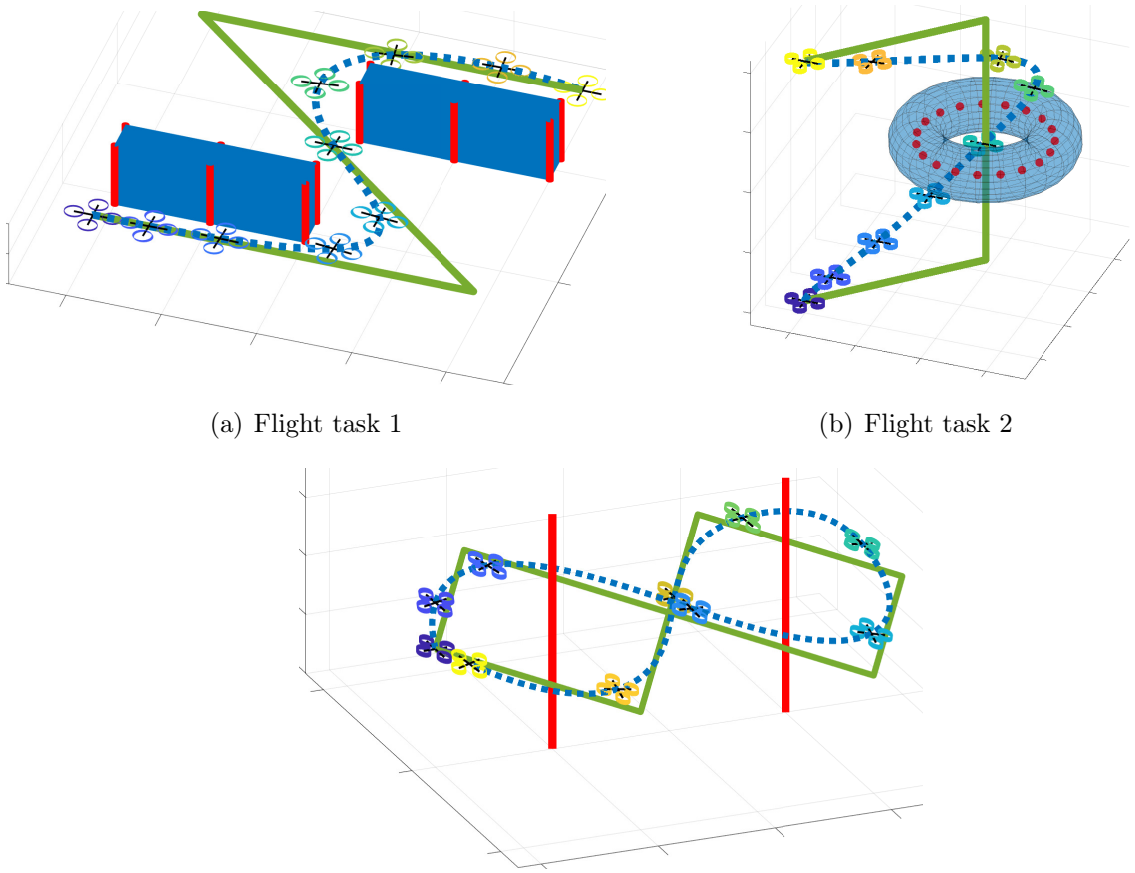
3.5.3 Quadcopter Model with Homotopy Class Constraints

Here we consider a quadcopter model with full-scale nonlinear dynamics to highlight the effectiveness of the proposed AER method to generate a feasible tracking trajectory, which at the same time preserves a desired homotopy class. The adopted dynamic model of the quadcopter system is described by a system of nonlinear ODEs with 12 states and 4 control inputs representing the rotating speed of four motors in rotation per minute (rpm) and takes the form

$$\frac{d}{dt}x = f_d(x) + \sum_{i=1}^{4} f_i(x)u_i^2,$$

where f_d and f_i are nonlinear differentiable functions. Readers are referred to [74] for the detailed dynamics structure.

Given the quadcopter model above, we leverage the AER method to plan feasible trajectories with homotopy constraints for three flight tasks in different environments. The simulation setup and results are shown in 3.6. In these tasks, we set $\gamma_1 = \gamma_2 = 0.0001\|\hat{U}\|^2$, $\gamma = 0.005$, the sampling rate to be 20 Hz, and total time horizon of each task to be 5, 6 and 8 seconds, respectively. Moreover, we heuristically initialize U such that the quadcopter hovers at the starting point, while the initial $\hat{U}$ then would act as the invisible force that drives the quadcopter along the preset trajectory.



(a) Flight task 1

(b) Flight task 2

(c) Flight task 3

Figure 3.6: For each flight task, the cluttered environment contains different types of obstacles that are identified by a set of anchors shown in red points/lines. The states with regard to the position of the preset nominal reference trajectory is presented as green polylines, while the rest, i.e. translational velocities, altitudes, angular velocities, are set to zero. AER method has transformed the preset reference to a dynamically-feasible one shown in the blue dashed curve for the quadcopter to track. Meanwhile, the homotopy class constraints are observed to be fulfilled. The sequential snapshots demonstrate the tracking procedure of the quadcopter starting from the darker blue color to lighter yellow color gradually.

3.6 Conclusion

In this section, we introduced a new motion planning technique called the auxiliary energy reduction (AER) method. The AER method obtains a dynamically feasible trajectory through minimizing the energy of the auxiliary control term which is artificially added to

the original dynamics. By incorporating the so-called anchor and anchor loss, the method further preserves the homotopy class for generated trajectories. We have demonstrated the practicability of the method in three different simulation examples. However, the efficiency of the proposed method is limited. We observed that when the dynamic of the agent or the required homotopy class is complicated, the updating of the trajectory may oscillate and eventually converge after tens of seconds. We believe it happens because of the contradiction of the AER optimization and the anchor loss optimization. To mitigate this challenge, we proposed a more efficient method that eliminate this contradiction, which will be introduced in the next chapter.

Chapter 4

Homotopy Method for Optimal Motion Planning with Homotopy Class Constraints

4.1 Introduction

In the previous chapter, we introduce the AER method to solve the motion planning problem with homotopy class constraints. However, the proposed algorithm includes two alternating parts, the AER part and the anchor loss part. The AER part pulls the trajectory to a more feasible shape, while the anchor loss part pushes the trajectory away from obstacles. When these alternating updates point in opposite directions, the trajectory will oscillate and the convergence will be slow. To mitigate this challenge, we refrain from updating the trajectory from infeasible to feasible, instead, we keep the trajectory always feasible while updating the environment from trivial to the original one. By implementing this method, the opposite update directions are eliminated and the efficiency is improved dramatically.

In this section, we propose a novel direction that addresses the optimal motion planning task with 2-dimensional homotopy class constraints for general nonlinear nonholonomic dynamical systems with both static and moving obstacles. We first initialize an optimal trajectory of the dynamical system without considering obstacles, then we design the auxiliary trajectories of obstacles such that the initial system trajectory is collision-free and belongs to the desired homotopy class regarding the auxiliary obstacle trajectories. Next, we iteratively deform auxiliary obstacle trajectories to the original ones and determine the optimal system trajectory accordingly. Having done so, the resulting trajectory is feasible and satisfies homotopy class constraints regarding the original obstacles.

This section is organized as follows. Section 6.2 introduces the problem setup and mathematical preliminaries. We then provide the method of auxiliary obstacle trajectory synthesizing in Section 4.3. The overall algorithm is given in Section 4.4. We present the numerical results in Section 6.4 and conclude the section in Section 6.5.

4.2 Mathematical Preliminaries and Problem Statement

4.2.1 Differentiable Obstacle Representations

Consider M number of moving obstacles which can be of any shape or size, in a 2-dimensional cluttered environment. An often-faced challenge is non-differentiable shapes such as a square and triangle that can not be directly handled by nonlinear programming. To address this issue, we approximate non-differentiable obstacles using super-ellipses [8], which in turn yields the collision avoidance constraints

$$\left(\frac{x(t) - z_{ix}(t)}{r_{ix}}\right)^{k_i} + \left(\frac{y(t) - z_{iy}(t)}{r_{iy}}\right)^{k_i} - {R_i}^{k_i} \geq 0, \tag{4.1}$$

where $z(t) = [z_{ix}(t), z_{iy}(t)]$ denotes the center of the i^{th} obstacle at time t. $r_{ix}, r_{iy} \geq 1$ represent the object's spatial extensions, and $[x(t), y(t)]$ is the trajectory of the dynamical system. $R_i > 0$ is a size constant and the exponent k_i is a positive even number. If $k_i = 2$, the obstacle is a circle or an ellipse, otherwise a rounded square, as shown in Figure 4.1.

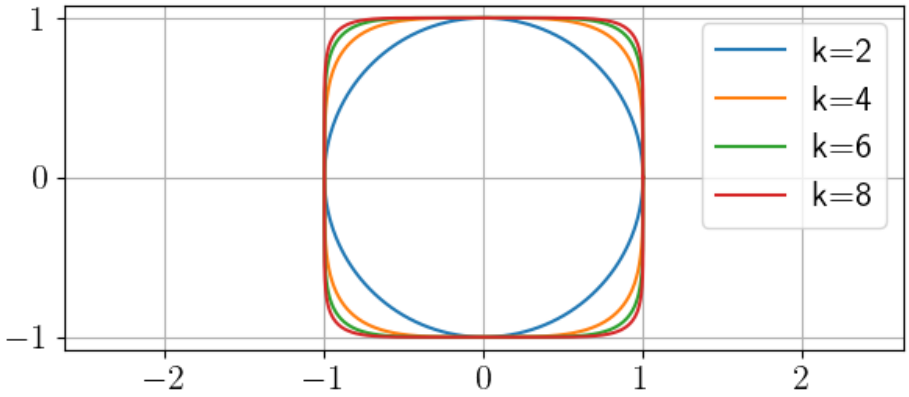


Figure 4.1: One obstacle approximation with $(z_x, z_y) = (0, 0)$, $r_x = r_y = R = 1$. As k increases, the obstacle turns more toward a rounded square.

4.2.2 Optimal Motion Planning

The basic task of motion planning is to find a dynamically feasible trajectory that connects the given starting and target points while satisfying some obstacle constraints imposed by the environment. The task of optimal motion planning further requires the synthesized trajectory to be optimal, or locally optimal, with respect to a certain metric, such as energy consumption. Generally, the optimal trajectory can be obtained by solving the following nonlinear program:

$$\begin{aligned} \underset{(x(\cdot),u(\cdot))}{\text{minimize}} \quad & \Phi(x(\cdot),u(\cdot)) \\ \text{subject to} \quad & x(0)=x_{\text{start}},\ x(T)=x_{\text{target}}, \\ & \dot{x}(t)=f(x(t),u(t)), \\ & G(x(t))\geq 0, \quad \forall t\in[0,T], \end{aligned} \tag{4.2}$$

where $x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m$ are the state and control input respectively. Φ is the loss function, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ describes the system dynamics, and $G(x(t)) \geq 0$ is the control barrier function which defines a collision-free space. In this section, we focus on energy-optimal motion planning such that $\Phi(x(\cdot), u(\cdot)) = \int_0^T \|u(t)\|^2 dt$ while the total time T is pre-given, but it is straightforward to modify the nonlinear program to further accommodate the time-optimal requirement.

4.2.3 Homotopy Class Constraints for Movable Obstacles

Collision avoidance of movable obstacles is critical in some topology-restricted tasks, e.g., a vehicle properly avoids other cars while merging off a highway. In this example, an appropriate route fulfilling the homotopy class constraints for movable obstacles, i.e., other in-moving cars, is desired. The definition of homotopy class is given in Definition 1 and it has been well explained in Section 6.2.

Given a nominal 2-dimensional trajectory, one homotopy class is specified and such a constraint should be preserved throughout the iterative optimization process. However, obstacles being movable imposes more challenges to the homotopy class constraint guarantee as both obstacles and the trajectory of the given dynamical system are changing over time. We hence

first define the homotopy class constraints under movable obstacle scenarios here and further prove that the proposed approach can preserve the desired homotopy class in Section 4.3.

By virtue of the Residue Theorem [1], if the analytic function $F : \mathbb{C} \to \mathbb{C}$ does not have poles, and $c(t)$ is a continuous and closed curve on the complex plane that never intersects a point $z_0 \in \mathbb{C}$, then the following line integral holds,

$$\oint_c \frac{F(z)}{z - z_0} dz = 2\pi j F(z_0) n(c, z_0), \tag{4.3}$$

where j is the imaginary unit, $n(c, z_0)$ is the winding number representing the total number of times that the curve c encircles counterclockwise around z_0. Therefore we have the followings.

Lemma 4.2.1 *Two continuous trajectories c_1 and c_2 in the complex plane with identical start and target points belong to the same homotopy class regarding the obstacle at z_0, if and only if*

$$\oint_{c_1 \sqcap c_2^-} \frac{F(z)}{z - z_0} dz = 0,$$

where $c_1 \sqcap c_2^-$ is a closed trajectory concatenating c_1 and inverse c_2, which starts from the start point of c_1 to the end point of c_1 along the trajectory c_1 and then travel from it to the start point along the trajectory c_2.

The proof can be seen in Lemma 1 of [9]. Note that for any trajectory $c(t)$, we have $\oint_c \frac{F(z)}{z-z_0} dz = \oint_{c'} \frac{F(z+z_0)}{z} dz$, where $c'(t) = c(t) - z_0$. In addition, we are able to heuristically generalize the criterion to a moving obstacle. More specifically, trajectories $c_1(t)$ and $c_2(t)$ belong to the same homotopy class regarding a obstacle trajectory $z_0(t)$, if and only if

$$\oint_{c_1' \sqcap c_2'^-} \frac{1}{z} dz = 0,$$

where $c_1'(t) = c_1(t) - z_0(t)$, $c_2'(t) = c_2(t) - z_0(t)$ and $F(z)$ is simply chosen as a constant map to 1. Note that if the obstacle trajectory is time-invariant, i.e. $z_0(t) \equiv z_0(0)$, this criterion also degenerates to the form of Lemma 4.2.1. Moreover, in a cluttered environment with

multiple obstacles, this criterion becomes

$$\begin{bmatrix} \oint_{c'_{1,1} \sqcap c'^{-}_{2,1}} \frac{1}{z} dz \\ \vdots \\ \oint_{c'_{1,M} \sqcap c'^{-}_{2,M}} \frac{1}{z} dz \end{bmatrix} = \mathbf{0}_{M \times 1}, \tag{4.4}$$

where $c'_{1,i}(t) = c_1(t) - z_i(t)$ and $c'_{2,i}(t) = c_2(t) - z_i(t)$, and $z_i(t)$ is the trajectory of the i-th obstacle.

4.3 Auxiliary Obstacle Trajectory Synthesis

In this section, we propose a method of auxiliary obstacle trajectory synthesizing that guarantees the satisfaction of homotopy class constraints of the initial system trajectory towards these modified obstacles. In addition, we provide the condition under which this homotopy property holds while trajectories are deforming. All trajectories in this section are discussed on the complex plane for concise proof, but the results can be directly adopted in the $\mathbb{R}^2$ space. Let $z_i(t) = z_{ix}(t) + j \cdot z_{iy}(t)$ be the known trajectory of the center of the i-th movable obstacle and $r(t)$ be another given trajectory to specify the desired homotopy class. $r(t)$ is designed to never intersect the obstacles, i.e. $\|r(t) - z_i(t)\| > 0$ holds for all $i = 1, 2, \ldots, M$ and $t \in [0, T]$. A system trajectory $p(t)$ is acquired without concerning homotopy class constraints and obstacles but has the same start and target points as the ones of $r(t)$. Based on the notations, we define the auxiliary obstacle trajectory parameterized by s as

$$\hat{z}_i(t; s) = z_i(t) + s \frac{z_i(t) - r(t)}{\|z_i(t) - r(t)\|}, \tag{4.5}$$

where $s \in [0, \infty)$ is called the push distance. As $\|r(t) - z_i(t)\|_2 \geq R_i > 0$, $\forall t \in [0, T]$ because $r(t)$ is collision-free, we can choose a large-enough s_0 so that the conditions hold

$$\begin{aligned} \min_t \|\hat{z}_i(t; s_0)\| &> \max_t \|r(t)\| \\ \min_t \|\hat{z}_i(t; s_0)\| &> \max_t \|p(t)\|. \end{aligned} \tag{4.6}$$

A graphical example is demonstrated in Figure 4.2. We now show, according to (4.6), the sufficient condition that $p(t)$ and $r(t)$ belong to the same homotopy class with respect to $\hat{z}_i$.

Lemma 4.3.1 *Given two continuous and closed trajectories $c_0(t)$ and $c_1(t)$ in the complex plane which never intersect the origin for $t \in [0,T]$, if there exists a continuous function $H(v,t)$ such that $H(0,t) = c_0(t)$, $H(1,t) = c_1(t)$, $\|H(v,t)\| \neq 0, \forall v \in [0,1], \forall t \in [0,T]$, and $H(v,t)$ is the continuous and closed trajectory with any fixed $v \in [0,1]$, then $\oint_{c_0} \frac{1}{z} dz = \oint_{c_1} \frac{1}{z} dz$.*

Proof: Because $H(v,t)$ is continuous and never intersects the origin, $\theta(v) \coloneqq \oint_{H(v,\cdot)} \frac{1}{z} dz$, $v \in [0,1]$ is also continuous. According to the Residue Theorem [1], $\theta(v) = 2\pi j \cdot n(H(v,\cdot),0)$ where $n(H(v,\cdot),0) \in \mathbb{Z}$ is the winding number. Therefore the function $\theta(v)$ has to be both continuous and discrete, which implies $\theta(v)$ is constant, so that $\theta(0) = \theta(1)$. ■

Corollary 4.3.2 *Given three trajectories $c_0(t)$, $c_1(t)$ and $\hat{z}(t)$ on the complex plane, where $c_0(0) = c_1(0)$, $c_0(T) = c_1(T)$, if $\min_t \|\hat{z}(t)\| > \max_t \|c_0(t)\|$ and $\min_t \|\hat{z}(t)\| > \max_t \|c_1(t)\|$, then $\oint_c \frac{1}{z} dz = 0$, where $c = (c_0 - \hat{z}) \sqcap (c_1 - \hat{z})^-$.*

Proof: We define the continuous function

$$H_1(v,t) = \begin{cases} v \cdot c_0(t) - \hat{z}(t), & 0 \leq t \leq T \\ v \cdot c_1(2T-t) - \hat{z}(2T-t), & T < t \leq 2T \end{cases}$$

where $v \in [0,1]$, $t \in [0,2T]$. Therefore, $H_1(v,t)$ is the closed trajectory with any fixed $v \in [0,1]$. For $v = 0$, $H_1(0,t)$ is a trivial trajectory that moves forth and back along $-\hat{z}$ such that $\oint_{H_1(0,\cdot)} \frac{1}{z} dz = 0$. $\|H_1(v,t)\| \geq \|\hat{z}(t)\| - s\|c_0(t)\| \geq \|\hat{z}(t)\| - \|c_0(t)\| > 0$ for $t \in [0,T]$, and similarly $\|H_1(v,t)\| > 0$ for $t \in [T,2T]$. Therefore $H_1(v,t)$ never intersects the original point. Hence according to Lemma 4.3.1, $\oint_{H(1,\cdot)} \frac{1}{z} dz = \oint_{H(0,\cdot)} \frac{1}{z} dz = 0$, where $H(1,\cdot)$ is actually the closed curve concatenating $c_0 - \hat{z}$ and inverse $c_1 - \hat{z}$. ■

By virtue of Lemma 4.2.1 and Corollary 6.3.2, with a large s_0 that pushes the obstacle far enough so as to satisfy the conditions in (4.6), $p(t)$ and $r(t)$ are within the same homotopy class concerning the auxiliary obstacle trajectory $\hat{z}_i(t;s_0)$. The same idea can be directly generalized to the environment with multiple obstacles $\{z_1(t),...,z_M(t)\}$, where a suitable s_0 is chosen for all obstacles. More specifically, the initialized system trajectory $p(t)$ and the pre-defined $r(t)$ belong to the same homotopy class with respect to a set of auxiliary obstacles trajectories $\{\hat{z}_1(t;s_0),...,\hat{z}_M(t;s_0)\}$ as long as the conditions in (4.6) holds for every obstacle.

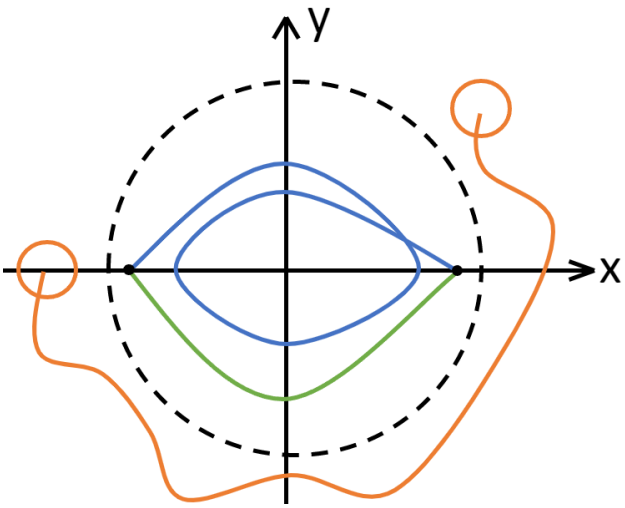


Figure 4.2: Orange curve represents the auxiliary obstacle trajectory $\hat{z}(t)$. Trajectories $c_0(t)$ and $c_1(t)$ in blue and green, respectively, have the same start and target point. As the conditions $\min_t \|\hat{z}(t)\| > \max_t \|c_0(t)\|$ and $\min_t \|\hat{z}(t)\| > \max_t \|c_1(t)\|$ hold, they belong to the same homotopy class regarding the orange obstacle.

In the following, we aim to show how to obtain the trajectory that belongs to the same homotopy class of the given trajectory $r(t)$ towards original obstacle trajectories by deforming $p(t)$ and $\hat{z}_i(t;s)$ continuously.

Corollary 4.3.3 *Given three trajectories $c_0(t;0)$, $c_1(t;0)$ and $\hat{z}(t;0)$ on the complex plane that continuously deform to $c_0(t;1)$, $c_1(t;1)$ and $\hat{z}(t;1)$ for $t \in [0,T]$, respectively. If $c_0(t;0)$ and $c_1(t;0)$ are homotopy equivalent towards $\hat{z}(t;0)$, $c_0(0;v) = c_1(0;v), c_0(T;v) = c_1(T;v), \forall v \in [0,1]$, and $c_0(t;v) \neq \hat{z}(t;v), c_1(t;v) \neq \hat{z}(t;v), \forall v \in [0,1], \forall t \in [0,T]$ then $c_0(t;1)$ and $c_1(t;1)$ are homotopy equivalent towards $\hat{z}(t;1)$.*

Proof: We define the continuous function

$$H_2(v,t) = \begin{cases} c_0(t;v) - \hat{z}(t;v), & 0 \leq t \leq T \\ c_1(2T-t;v) - \hat{z}(2T-t;v), & T < t \leq 2T \end{cases}$$

where $v \in [0,1]$, $t \in [0,2T]$, then $H_2(v,t)$ is the closed trajectory with any fixed $v \in [0,1]$. $H_2(v,t)$ never reaches the origin because $c_0(t;v) \neq \hat{z}(t;v)$ and $c_1(t;v) \neq \hat{z}(t;v)$ always hold. Therefore according to the Lemma 4.3.1, $c_0(t;1)$ and $c_1(t;1)$ are homotopy equivalent towards $\hat{z}(t;1)$. ■

According to Corollary 4.3.3, during the continuous deformation of the auxiliary obstacle trajectories $\hat{z}_i(t;s)$ and the system trajectory $p(t)$, $p(t)$ remains within the same homotopy class of $r(t)$ if $\hat{z}_i(t;s)$ never intersects $p(t)$ and $r(t)$. Based on the definition (4.5), the deformation of $\hat{z}_i(t;s)$ can be controlled by $s \in [0,\infty)$. Therefore, by gradually decreasing s from s_0 to zero, i.e. $\hat{z}_i(t;s_0)$ is deformed to $\hat{z}_i(t;0) = z_i(t)$, and deforming $p(t)$ accordingly with no intersection, the resulting $p(t)$ also satisfies the homotopy class constraints regarding the original obstacles $z_i(t)$. It is noted that $\hat{z}_i(t;s)$ never intersects the pre-defined trajectory $r(t)$ in this deformation, which can be seen below:

$$\|\hat{z}_i(t;s) - r(t)\| = \left(1 + \frac{s}{\|z_i(t) - r(t)\|}\right) \|z_i(t) - r(t)\| > 0.$$

4.4 Implemention with Optimal Control Criterion

In this section, we propose an algorithm to solve the problem of preserving the homotopy property from the last section by reformulating it into a sequence of nonlinear programming problems (NLP). The resulting NLP can be efficiently addressed by mature and off-the-shelf solvers and toolboxes such as CasADi [3]. In the following, we first describe a process of achieving optimal motion planning tasks with known obstacles using NLP. Then we show how to prevent the system trajectories from intersecting the obstacles by transferring this task into sequential optimal motion planning problems.

4.4.1 Nonlinear Programming

To use the multiple shooting method, a continuous-time nonlinear dynamical system is discretized to the form

$$\begin{aligned} x(k+1) &= F(x(k), u(k)), \\ p(k) &= g(x(k)), \end{aligned} \tag{4.7}$$

where $x(k) := x(k\Delta T)$, $u(k) := u(k\Delta T)$. The obstacle trajectories are also discretized as $\hat{z}_i(k;s) := \hat{z}_i(k\Delta T;s)$. Although the full state trajectory $x(k)$ can be high-dimensional, homotopy class constraints are only applied to $p(k)$, which is a 2-dimensional system trajectory.

Therefore, the optimal motion planning problem can then be framed as the NLP

$$\begin{aligned} \underset{(U,X)}{\text{minimize}} \quad & \|U\|^2 \\ \text{subject to} \quad & x(0) = x_{\text{start}},\ x(N) = x_{\text{target}}, \\ & x(k+1) = F(x(k), u(k)), \\ & G(g(x(k)), \hat{z}_i(k; s)) \geq 0, \\ & \forall k \in \{0, ..., N\}, i \in \{1, ..., M\}, \end{aligned} \tag{4.8}$$

where $X = [x(0), ..., x(N)]$, $U = [u(0), ..., u(N-1)]$, and $G(g(x(k)), \hat{z}_i(k; s))$ is the control barrier function described in constraint (6.3). In our following experiments, the interior point optimizer is utilized to address such NLP (4.8). And the convergence analysis of the solver can be seen in [29].

4.4.2 Homotopy Method

As mentioned in Section 4.3, resolving the motion planning problems with non-differentiable homotopy class constraints is equivalent to resolving the sequential differentiable motion planning problems. The auxiliary obstacle trajectory initialization can be simply achieved by choosing a large enough s_0, and the challenging task is to ensure that the system trajectory and obstacles remain untouched during the continuous deformation of the auxiliary obstacle trajectory by decreasing the push distance s, as specified in Corollary 4.3.3.

To address the challenge, we adopt the idea of the homotopy method [36] which first solves the simpler problems that are deformed from the original and complicated problem, and then gradually leverages the intermediate results to solve the ones all the way back to the original problem. In this process, we take advantage of the fact that if the local optimal trajectory is already synthesized, then obstacles are moved by a small distance and the solver uses the last result as initial values, the solver tends to provide the nearest local optimal solution. To illustrate the adapted homotopy method to our problem, we utilize a simple motion planning problem shown in Figure 4.3. In this example, we attempt to solve an energy-optimal motion planning problem concerning a simple system $\dot{x} = u$ and a circle obstacle that is centered at $(4.5, 0)$ with the radius of 0.5, such that the system starts from $(0, -1.5)$

to $(0, 1.5)$ and bypasses the obstacle on its right side. Instead of directly solving this hard-to-address problem, we first solve an easier problem that places the obstacle center at $(0, 0)$. Based on this intermediate result, we then solve the problem that moves the obstacle a bit right to $(0.1, 0)$. By iterating this procedure, the problem will be gradually deformed to the original one that can be easily solved based on results from the previous iteration.

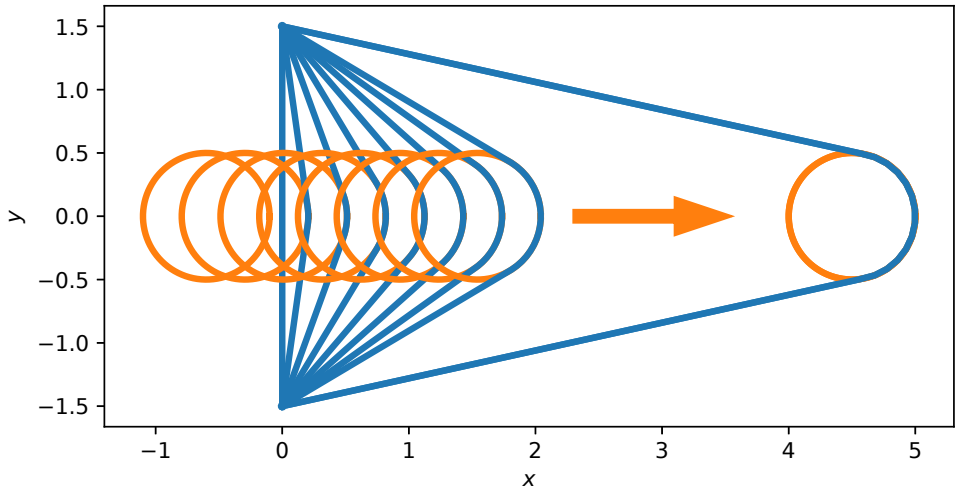


Figure 4.3: Orange circles represent the obstacles and blue curves represent the synthesized shortest path in each iteration. In each iteration, the computed optimal trajectory is seen to stay on the right side of the obstacle, which fulfills the control barrier constraints.

Algorithm 7 Homotopy Method for Homotopy Class Constraints

Require: Obstacle trajectories $\{z_1(k), ..., z_M(k)\}$ and pre-established trajectory $r(k)$ which defines the required homotopy class.

1: Initialize inputs U^0 and state trajectory X^0 by solving (4.8) without considering obstacles.
2: Synthesis trajectories for obstacles $\{\hat{z}_1(k; s), ..., \hat{z}_M(k; s)\}$ according to (4.5) with s large enough.
3: Set iteration step $m = 0$.
4: Set $s = \max(s - \Delta s, 0)$, $m = m + 1$
5: Solve optimal motion planning problem (4.8) with obstacles and initial $(U, X) = (U^{m-1}, X^{m-1})$, and the solution of it is noted as (U^m, X^m).
6: Repeat step $4 - 5$ until $s = 0$.

The proposed approach is summarized in Algorithm 7. To choose the initial value of s, we first set it as zero and gradually increase it until no collision happens and the initial system trajectory $p(t)$ and the given trajectory $r(t)$ are in the same homotopy class towards auxiliary obstacle trajectories by checking whether the criterion (4.4) is satisfied. Corollary 6.3.2

guarantees the existence of such s, while it is worth noting that a small value of s will decrease the efficiency of the method. To find a suitable value of Δs, we may start from a large value of it and update obstacle trajectories and the system trajectory. If the homotopy class constraints are not satisfied with the updated trajectory, we can return to the last step and try a smaller Δs. But setting $\Delta s = \min_i R_i/5$ is found to be good in practice. To prevent obstacles from passing through the discrete trajectory and moving to another homotopy class, sample points of the system trajectory should be dense enough by choosing suitable ΔT.

4.5 Numerical Results

We present numerical results on two classic dynamical systems: the unicycle and the quadcopter. The results demonstrate that the proposed approach can iteratively synthesize optimal trajectories with homotopy class constraints for nonholonomic unicycle systems and highly nonlinear quadcopter systems.

4.5.1 Unicycle

We first adopt the unicycle model

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ \phi \end{pmatrix} = \begin{pmatrix} v\cos\phi \\ v\sin\phi \\ \omega \end{pmatrix},$$

as a classical nonholonomic system to illustrate optimal trajectory generation under homotopy class constraints. Here x and y indicate the position of the unicycle, and ϕ denotes the facing angle of the unicycle. Moreover, v and ω are control inputs that stand for the moving and steering velocity, respectively.

In the following demonstrations, the homotopy class constraints are applied to the position states. The facing angle ϕ is set to zero at both the start and target points. Figure 4.4 shows the results of the advocated approach that addresses the energy-optimal motion planning problems for a unicycle system, where the unicycle is requested to encircle two stationary

obstacles before reaching the target. The total time is 10 seconds and the discrete time interval is set as $\Delta T = 0.05$ second. In addition, the push distance s is decreased from 2.5 to 0 with a step of $\Delta s = 0.1$. Another simulation with one obstacle and a different homotopy class is carried out on the same system and setups except that s is initialized as 1.0, whose results are presented in Figure 4.5.

We additionally compare the computational efficiency of our proposed method with the MIP-based method, which is shown in Figure 4.6. We attempt to leverage both methods on the same motion planning problems shown in Figure 4.5 but with varying sample points number, $N = T/\Delta T$, by changing ΔT. Because of the larger gap between each adjacent sample points of the trajectory, smaller N tends to result in a trajectory that violates the obstacle constraints. Due to the exponentially-like growing computation complexity of the MIP-based method, our method enjoys the fulfillment of the obstacle constraints with reasonable computation time.

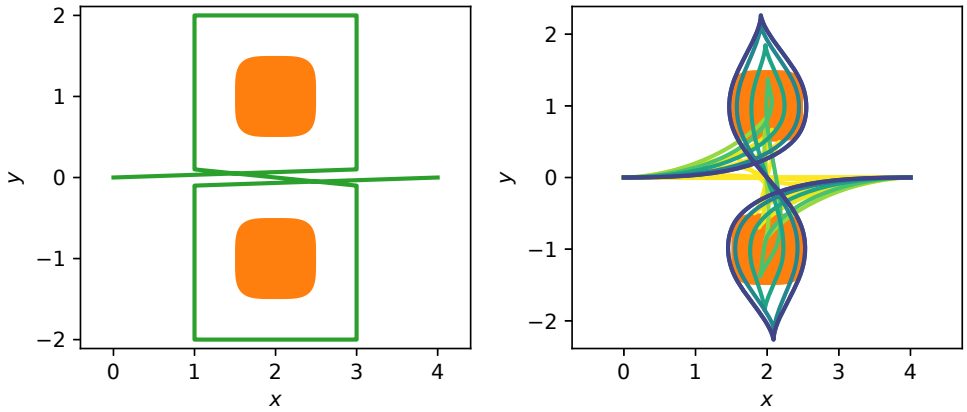


Figure 4.4: The optimal trajectory is synthesized for a unicycle system with two stationary obstacles centered at $(2, -1)$ and $(2, 1)$ with $R = 0.5$ and $k = 4$. The initial and target points of the system are set to $(0, 0)$ and $(4, 0)$, respectively. The left figure shows the stationary obstacles (orange) and the given trajectory (green) in the $x - y$ plane. The right figure illustrates how the initial system trajectory (yellow) gradually deforms into the optimal trajectory (blue) that obeys the homotopy class constraints.

4.5.2 Quadcopter

Here we consider a holonomic but highly-nonlinear quadcopter model that is described by the nonlinear dynamics with 12 states and 4 control inputs denoting the rotating speed of

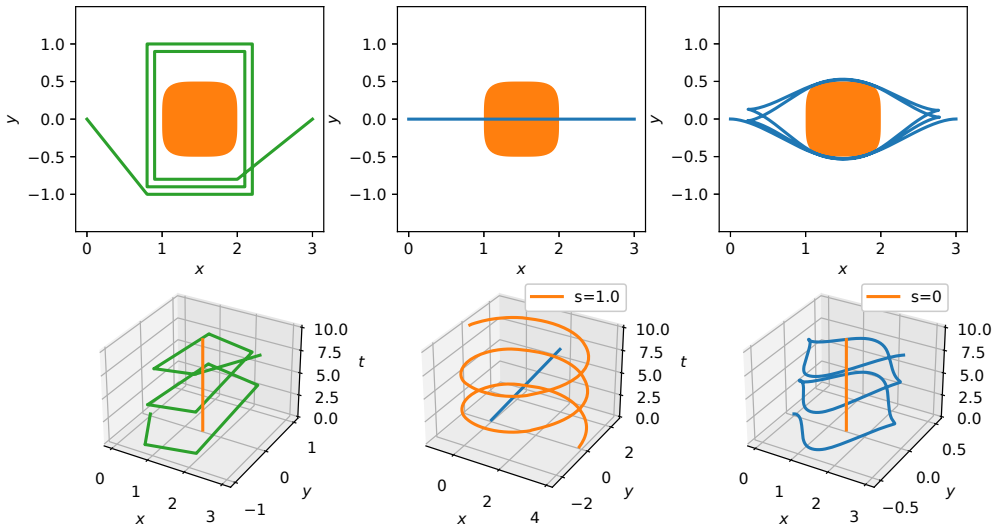


Figure 4.5: The optimal trajectory is synthesized for a unicycle system with one stationary obstacle centered at $(1.5, 0)$ with $R = 0.5$ and $k = 4$. The start and target point of the system are $(0, 0)$ and $(3, 0)$, respectively. The upper row shows the given trajectory (green), the system trajectory (blue) and an obstacle (orange) in the $x-y$ plane, while the lower row presents the corresponding trajectory in the $x-y-t$ space. The middle column indicates the initial trajectories with $s = 1.0$, and the right column shows the final results.

four motors in rotation per minute and takes the form

$$\frac{d}{dt}x = f_d(x) + \sum_{i=1}^{4} f_i(x)(u_i + u_{eq})^2,$$

where $u = [u_1, u_2, u_3, u_4]^T$ are control inputs, f_d and f_i are differentiable functions. u_{eq} is the hovering motor speed, such that $(x, u) = (\mathbf{0}_{12\times1}, \mathbf{0}_{4\times1})$ is the equilibrium point of the dynamical system. Readers are referred to [74] for the detailed dynamics structure. According to the definition of the homotopy class in Section 4.2.3, the constraints considered here are only applied to the $x-y$ plane, which means the altitude of the quadcopter is not concerned. As for the initial and target points, all states besides x_1 and x_2 are set to zero.

Figure 4.7 presents the numerical results for a quadcopter flight in a cluttered environment with four stationary obstacles. The total time needed is set as $T = 10s$ and the time interval as $\Delta T = 0.05s$. In addition, the push distance is gradually decreased from $s = 2.5$ to $s = 0$ with $\Delta s = 0.05$. In Figure 4.8, the algorithm is applied to the same system with the same experimental setups, whereas two moving obstacles are considered instead. The results

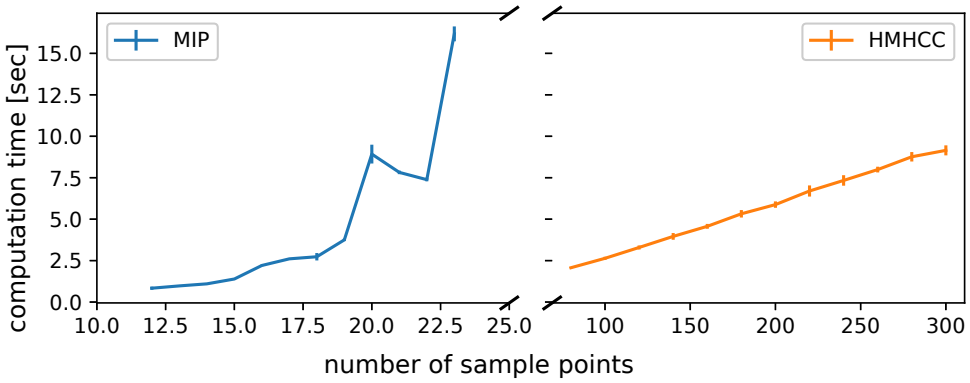


Figure 4.6: Comparison of the computation time between two methods. The error distribution at each number of sample points is estimated based on 5 simulations.

suggest that the proposed method is further capable of addressing the homotopy-constrained optimal motion planning problems with moving obstacles.

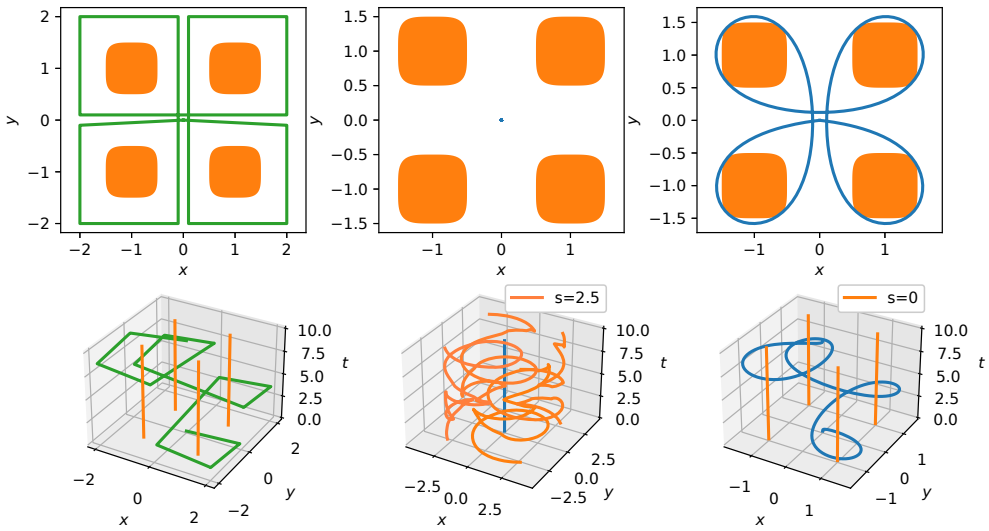


Figure 4.7: The optimal trajectory is synthesized for a quadcopter system with four stationary obstacles centered at $(\pm 1, \pm 1)$ with $R = 0.5$ and $k = 4$. The start and target point of the system trajectory are set at $(0, 0)$. The layout and explanation of the figure are identical to Figure 4.5, except that the push distance is initialized as $s = 2.5$.

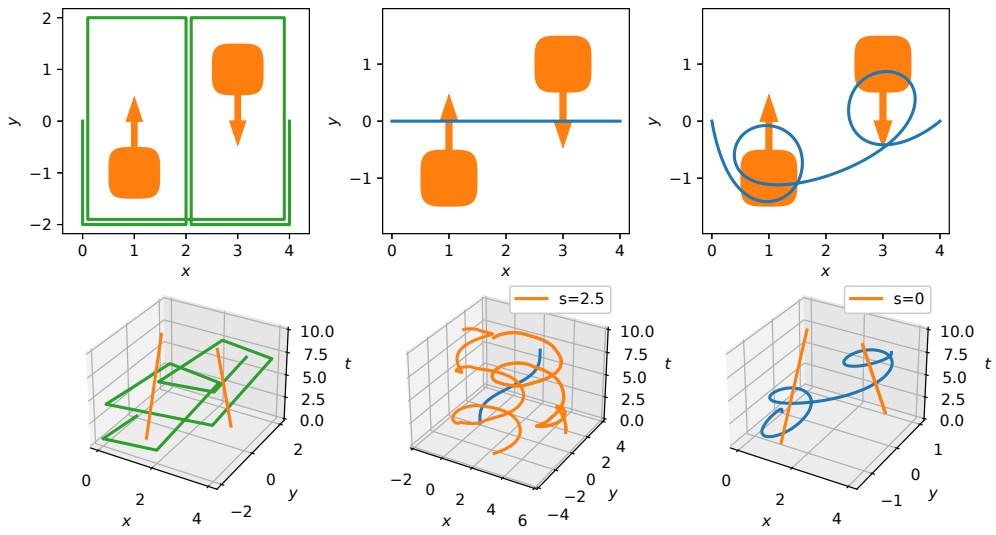


Figure 4.8: The optimal trajectory is synthesized for a quadcopter system with two moving obstacles, which are originally centered at $(1, -1)$ and $(3, 1)$ with $R = 0.5$ and $k = 4$ with a constant moving speed at 0.1 m/s, and the moving direction is indicated by the orange arrow. The start and target point of the system are set at $(0, 0)$ and $(4, 0)$, respectively. The layout and explanation of the figure are identical to Figure 4.5, except that the push distance is initialized as $s = 2.5$.

4.6 Conclusion

In this section, we introduced a novel optimal motion planning technique with 2-dimensional homotopy class constraints. The proposed method first forces the initialized dynamical system trajectory to belong to the desired homotopy class regarding the auxiliary obstacle trajectories rather than the original obstacles. By gradually deforming the auxiliary ones to their original counterparts, the dynamical system trajectory, which fulfills the homotopy property, and the corresponding inputs are eventually synthesized. We have demonstrated the practicability of the proposed method on nonholonomic systems with both static obstacles and movable obstacles. However, the current method is limited to the planer homotopy class constraints. To find the possible method that extends it to 3D obstacles, we explore the embedding method, which will be shown in the next section.

Chapter 5

Value Iteration Algorithm for Solving Shortest Path Problems with Homology Class Constraints

5.1 Introduction

In the last chapter, we proposed HMHCC which is limited to the planer homotopy. To find possible methods that can handle 3D obstacles in 2D space, we explore the embedding method and utilize it for shortest path problems with homology class constraint tasks. class constraints Generally speaking, two trajectories belong to the same homology class if they are homotopy equivalent regarding every single obstacle. Again, in drone racing, trajectories that connect the same initial and target locations and pass through every gate in arbitrary order constitute a homology class. Although homology class constraints can be considered as the loose representation of homotopy class constraints, the corresponding optimization is much harder to address, as the feasible solution set is no longer compact. Therefore, the global search-based algorithm is preferable. Pioneering methods based on H-signature are proposed to generate optimal trajectories under homology class constraints for both 2D [45] and 3D [10] environments. Despite the capability of dealing with obstacles with general shapes, H-signature-based methods have to be defined differently depending on the environmental dimensionality, which hinders efficiency. In this section, we proposed a novel approach that solves the constrained shortest path problem for both 2D and 3D environments in a unified manner. For 3D cases, we first compress the super-toroid-shaped obstacles into a 2D counterpart. We then classify the homology class of the 2D trajectory by adopting the idea of phase change. Having done so, we are able to compute the shortest path fulfilling the topological requirement using a visibility graph and VIA. The analysis of our method

suggests the potential generalization to higher dimensional environments with obstacles that are properly described.

This section is organized as follows. Section 6.2 defines the homotopy and homology class, as well as obstacles in different dimensions. Then we introduce the proposed method in both 2D and 3D environments and the Value Iteration Algorithm in Section 5.3. Numerical Results are shown in Section 6.4 and we conclude the section in Section 6.5.

5.2 Mathematical Preliminaries and Problem Statement

5.2.1 Homotopy and Homology Classes for Static Obstacles

In this section, we begin with the definition of homotopy class for trajectories and depict the connection between homotopy and homology. Two continuous trajectories belong to the same homotopy class if and only if they have identical start and target points and they can be smoothly deformed into one another without intersecting any obstacles, as shown in Figure 5.1. Although the definition of homotopy class is given in Definition 1, a more formal definition that will be used in the later part of this section is shown below:

Definition 2 *Two continuous trajectories* $r_1(t) : [0,1] \to \mathbb{R}^n$ *and* $r_2(t) : [0,1] \to \mathbb{R}^n$ *belong to the same homotopy class if and only if* $r_1(0) = r_2(0)$, $r_1(1) = r_2(1)$, *and there exists a continuous function* $H(s,t) : [0,1] \times [0,1] \to \mathbb{R}^n$ *which satisfies* $H(0,t) = r_1(t)$, $H(1,t) = r_2(t)$ *and* $H(s,t)$ *never intersects any obstacles for all* $s \in [0,1]$ *and* $t \in [0,1]$.

Considering a 2-dimensional space with a single obstacle, the homotopy class of a trajectory can be quantified from the perspective of phase change. Given a trajectory $r(t) : [0,1] \to \mathbb{R}^2$ and a fixed point o_i representing the center point of the obstacle, the responding phase change regarding the obstacle is defined as

$$p(r) = \theta(r(1) - o_i) - \theta(r(0) - o_i), \tag{5.1}$$

where $\theta : \mathbb{R}^2 \to \mathbb{R}$ is the unwrapped angle, which is the unwrapped version of arctan : $\mathbb{R}^2 \to [0, 2\pi)$ that ensure $\theta(p(\cdot))$ is continuous if $p(\cdot)$ is continuous. It implies that two trajectories

with identical start and target points belong to the same homology class if they own the same phase change.

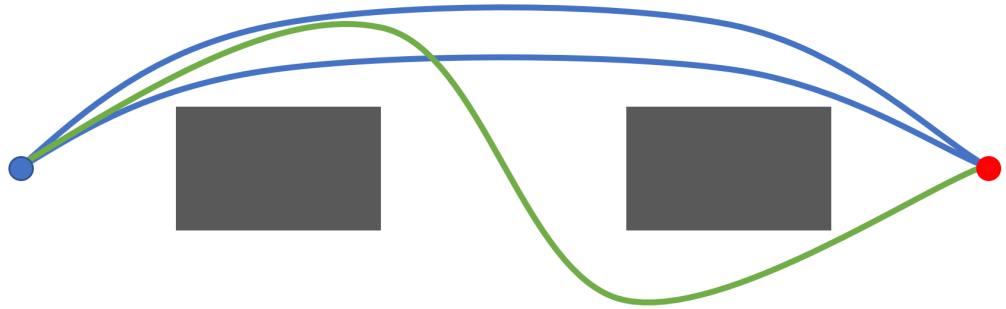


Figure 5.1: The three curves are the trajectories connecting identical starting and target points, and grey rectangles are obstacles. According to the definition of the homotopy class, the two blue trajectories are homotopy equivalent, but the green one belongs to a different homotopy class.

In a cluttered environment, the phase change with respect to n obstacles is given by $p(r) = [p_1(r), \ldots, p_n(r)]^T$, where $p_i(r)$ is the phase change regarding the ith obstacle. Generally, for an arbitrary dimensional space with multiple obstacles, trajectories belong to the same homology class if they have the same $p(r)$, i.e., they belong to the same homotopy class regarding every individual obstacle. However, having the same phase change is only a necessary condition for homolopy as shown in Figure 5.2, and the formal definition is given below.

Definition 3 *Two continuous trajectories $r_1(t) : [0, 1] \to \mathbb{R}^n$ and $r_2(t) : [0, 1] \to \mathbb{R}^n$ belong to the same homology class if and only if $r_1(0) = r_2(0)$, $r_1(1) = r_2(1)$, and $r_1(t)$ together with $r_2(t)$ with the opposite orientation forms the complete boundary of a 2-dimensional manifold embedded in $\mathbb{R}^n$ not containing any obstacles.*

Although homology is a loose representation of homotopy in many applications, finding the optimal trajectory belonging to a specified homology class is even more challenging than that of the homotopy counterpart. It can be seen from the fact that a homotopy class can be treated as a compact set, as every element can continuously deform to one another. Therefore, starting from a random feasible trajectory, one can reach the local or even global optimum of such a homotopy class by leveraging the gradient information. Nevertheless, a homology class is usually a disconnected set that contains various homotopy classes, which prevents the implementation of the gradient method, and thus a global searching mechanism is needed. The objective of this section is to design the shortest trajectory that is obstacle-free and belongs to a particular homology class in both 2D and 3D space.

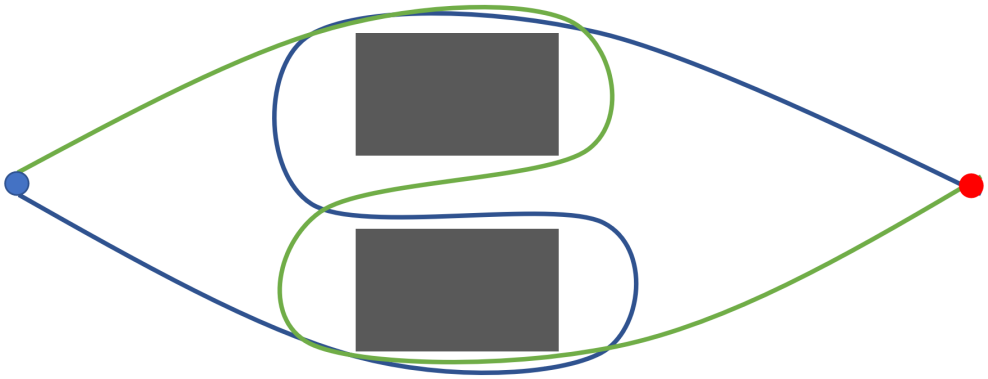


Figure 5.2: The two curves are the trajectories connecting identical starting and target points. They belong to the same homology class but different homotopy classes.

5.2.2 Static Obstacle Representations

For practical purposes, we consider M numbers of static obstacles in i) a 2-dimensional (2D) and ii) a 3-dimensional (3D) cluttered environment. In case i), though the proposed method can handle obstacles with arbitrary shapes, we approximate them using super-ellipse [8] that is aligned with the case of the 3D environment. In case ii), we consider a type of obstacle with a hole connecting the two opposite facets, which emulates the obstacle that a moving object needs to pass through.

In 2D environments, the collision avoidance constraints regarding the ith obstacle can be written as

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} := M_i \, \mathrm{diag}(e_{ix}, e_{iy}) \begin{bmatrix} x \\ y \end{bmatrix} + b_i, \tag{5.2}$$

$$\hat{x}^{k_i} + \hat{y}^{k_i} - {R_i}^{k_i} \geq 0, \tag{5.3}$$

where $[x, y]$ denotes the allowed spatial trajectory of the dynamical system, M_i is the rotation matrix, $e_{ix}, e_{iy} \geq 1$ represent the spatial expansion factor along each axis, b_i is the offset of the center of the obstacle, $R_i > 0$ is a size constant and the exponent k_i is a positive even number, which regulates the shape of the obstacle. Particularly, when $k_i = 2$, the obstacle is a circle or ellipse, otherwise, it would be a rounded square.

In 3D environments, we consider a group of obstacles with super-toroid shapes, where the ith obstacle is described by

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} := M_i \operatorname{diag}(e_{ix}, e_{iy}, e_{iz}) \begin{bmatrix} x \\ y \\ z \end{bmatrix} + b_i, \tag{5.4}$$

$$\hat{z}^{m_i} + (R_i - (\hat{x}^{n_i} + \hat{y}^{n_i})^{\frac{1}{n_i}})^{m_i} - {r_i}^{m_i} \geq 0, \tag{5.5}$$

where R_i is the distance from the center of the tube to the center of the obstacle, r_i is the radius of the tube, and the rest of the parameters are defined in a similar way to that in (6.4). It is noted that m_i and n_i have to be positive even numbers. Some examples of the obstacle are shown in Figure 6.1

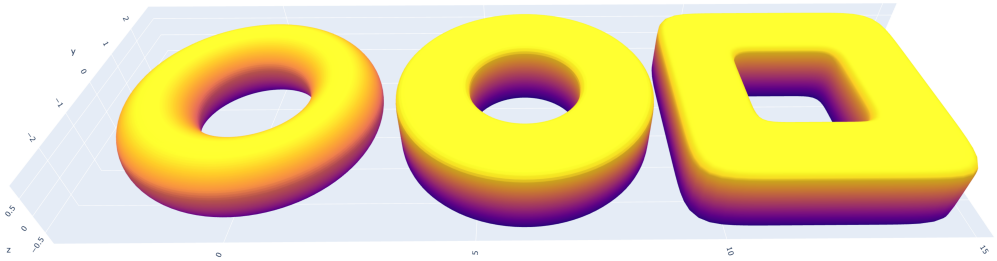


Figure 5.3: 3-dimensional obstacles with $(n, m) = (2, 2), (2, 8), (8, 8)$ from left to right, respectively.

5.3 Methods

In this section, we describe our strategy to synthesize the shortest obstacle-free path within a specific homology class. Our method relies on the phase change representation of the homology class that is well-defined in 2D space, which will be illustrated in Section 5.3.1. In Section 5.3.2, we prove that the interested type of 3D obstacles can be simply compressed into a 2D space. In both cases, we transform feasible paths into a graph and find the shortest one using the value iteration shown in Section 5.3.3.

5.3.1 2D Environment

To better present the corresponding homotopy class of a path, we associate a phase change with each point of the path. More specifically, given a path and M obstacles, we define a

node of the path as a vector $(x, y, p_1, \ldots, p_M)$, where (x, y) represents the position of the node and $p_i \in \mathbb{R}$ denotes the corresponding phase change regarding the ith obstacle. Because of the conservation of the phase change, given a specified path, the phase change regarding an obstacle only relies on the number of the encirclement around it that must be an integer. Therefore, the difference in the phase change of the two paths with the same ending points has to be an integral multiple of 2π. As a result, we are able to define a straight line, which connects the given start point $\mathbf{x}_{\text{start}}$ and (x, y), as a base path with a base phase change defined as $(\bar{p}_1, \ldots, \bar{p}_M)$. It is noted that such a base path does not have to be feasible in terms of obstacles. According to the base path, the node located at (x, y) of any feasible path, which starts from $\mathbf{x}_{\text{start}}$, can can be expressed as $(x, y, \bar{p}_1 + s_1 2\pi, \ldots, \bar{p}_M + s_M 2\pi),\ s_i \in \mathbb{Z}$. For simplicity, we use $(x, y, s_1, \ldots, s_M)$ to define the state in the rest of the section and $(s_1, \ldots, s_M)$ is the homology class label, where an example is shown in Figure 5.4. Likewise, if the line connects $(x^a, y^a, s_1^a, \ldots, s_M^a)$ and $(x^b, y^b, s_1^b, \ldots, s_M^b)$, we define the label-wise phase change as $(\Delta s_1, \ldots, \Delta s_M) = (s_1^a - s_1^b, \ldots, s_M^a - s_M^b)$.

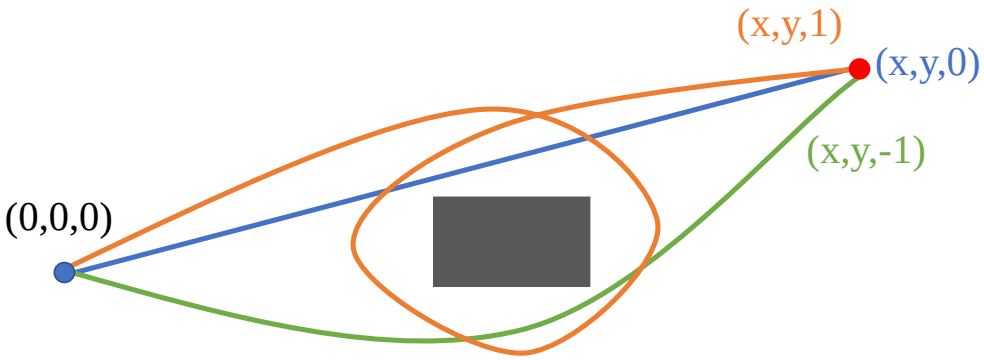


Figure 5.4: Three different paths connecting $(0, 0)$ and (x, y) belong to different homology classes, where the corresponding nodes at (x, y) are marked as well.

Equipped with the definition above, we proposed a graph search-based method to find the shortest path for moving an agent from the start point to the target, where the agent is assumed to be holonomic, i.e., any path is dynamically feasible to the agent. Specifically, we consider a visibility graph that is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of sampled states and $\mathcal{E}$ contains obstacle-free connections between these states. Given an interested point (x, y), we obtain $(x, y, s_1, \ldots, s_M)$ as sampled states with $s_i \in \{-\Gamma, -\Gamma + 1, \ldots, \Gamma\}$ and $\Gamma \in \mathbb{Z}^+$ is the hyper-parameter that determines the size of the sample space. Likewise, we say a state is legal iff $-\Gamma \leq s_i \leq \Gamma$ for all $i \in \{1, \ldots, M\}$. As such, given N interested points, the cardinality of the sample space is $|\mathcal{V}| = N(2\Gamma + 1)^M$, whose value significantly

impacts the time complexity of graph search-based algorithms. To reduce the number of interested points without highly degenerating the performance of the proposed method, we sample locations around obstacles rather than discretize the space into a grid. Around the obstacles described in Section 5.2.2, we adopt the sampling strategy shown below:

$$\begin{aligned}\begin{bmatrix} x \\ y \end{bmatrix}_{\text{sample}} =& \text{diag}(1/e_{ix}, 1/e_{iy}) M_i^{-1} \\ & \left(\frac{R_i + \delta}{(\cos(\theta)^{k_i} + \sin(\theta)^{k_1})^{1/k_i}} \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} - b_i \right),\end{aligned} \tag{5.6}$$

where $\theta \in \{0, 2\pi/N_i, \ldots, 2(N_i - 1)\pi/N_i\}$ is the sampling angle, N_i is the number of samples around the obstacle, and $\delta > 0$ is a hyper-parameter that ensures the sampled points residing outside the obstacle. The algorithm for building the visible graph is summarized in Algorithm 8. Note that the graph assumes that each state is self-connected. Given the graph $\mathcal{G}$, the shortest path can be obtained by value iteration that will be described in Section 5.3.3.

Algorithm 8 Construct Visibility Graph

Require: Obstacle description and hyper-parameter Γ
Initialize an empty graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a potions set $\mathcal{L}$
Sampled points according to (5.6), and store them in $\mathcal{L}$
Push the start and target points into $\mathcal{L}$
for each pair of points $((x^a, y^a), (x^b, y^b))$ from $\mathcal{L}$
    if the straight line connecting the pair is infeasible
        continue
    Get phase change $(\Delta s_1, \ldots, \Delta s_M)$ of the straight line
    for each legal $(s_1, \ldots, s_M)$
        if $(s_1 + \Delta s_1, \ldots, s_M + \Delta s_M)$ is also legal
            Add the edge between $(x^a, y^a, s_1, \ldots, s_M)$
            and $(x^b, y^b, s_1 + \Delta s_1, \ldots, s_M + \Delta s_M)$ to $\mathcal{G}$
return $\mathcal{G}$

5.3.2 3D Environment

It is noted that the graph-based method proposed above depends on the phase change for classifying homology classes. Hereby, it is severely restricted and thus only applicable to planar systems. To generalize the method to 3-dimensional, as defined in Section 5.2.2, we consider a certain type of 3D obstacle equipped with a hole, where the agent can pass

through. Because the coordinate transformation between $(\hat{x}, \hat{y}, \hat{y})^\top$ and $(x, y, z)^\top$ is invertible in (6.1), in this section and without loss of generality, we only discuss normalized obstacles

$$z^{m_i} + (R_i - (x^{n_i} + y^{n_i})^{\frac{1}{n_i}})^{m_i} - {r_i}^{m_i} \geq 0. \tag{5.7}$$

We consider an embedding mapping $f_i : \mathbb{R}^3 \to \mathbb{R}^2$

$$f_i\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} (x^{n_i} + y^{n_i})^{\frac{1}{n_i}} \\ z \end{bmatrix} := \begin{bmatrix} x_i^e \\ y_i^e \end{bmatrix}, \tag{5.8}$$

which compresses the ith 3D obstacle to a super-ellipse centered at $(R_i, 0)$ with the radius r_i and the exponent m_i. As a result, the phase change technique proposed in Section 5.3.1 can be safely applied.

Lemma 5.3.1 *Given a trajectory $r(t) : [0, 1] \to \mathbb{R}^3$ and the super-toroid obstacle, the trajectory is obstacle-free iff the embedded trajectory $(x_i^e(t), y_i^e(t))^\top = f_i(r(t))$ is obstacle-free for all $t \in [0, 1]$ in the embedding space:*

$$(R_i - x_i^e(t))^{m_i} + y_i^e(t)^{m_i} - r_i^{m_i} \geq 0.$$

Proof: Replacing $x_i^e(t)$ and $y_i^e(t)$ by $(x(t)^{n_i} + y(t)^{n_i})^{\frac{1}{n_i}}$ and $z(t)$ in (6.4), respectively, the obstacle-free condition for 2D super-ellipse is identical to the one of that in 3D space. ■

As such, one can easily evaluate the phase change of a trajectory and further the homotopy relationship of multiple trajectories in the embedding space, which is shown in Figure 6.3. Having the same phase change in the embedding space is the necessary and sufficient condition for two trajectories to be within the same homotopy class when only one obstacle exists as shown in the corollaries below. If trajectories are homotopy equivalent towards every single obstacle, then they belong to the same homology class.

Corollary 5.3.2 *If two continuous trajectories $r_1(t) : [0, 1] \to \mathbb{R}^3$ and $r_2(t) : [0, 1] \to \mathbb{R}^3$ belong to the same homotopy class regarding the super-toroid obstacle, then their embedding trajectories $r_1^e(t)$ and $r_2^e(t)$ are also homotopy equivalent regarding the embedded obstacle.*

Proof: There exist a continuous function $H(s, t) : [0, 1] \times [0, 1] \to \mathbb{R}^3$ that $H(s, t)$ never inside the obstacle, $H(0, t) = r_1(t)$ and $H(1, t) = r_2(t)$ because r_1 and r_2 are homotopy

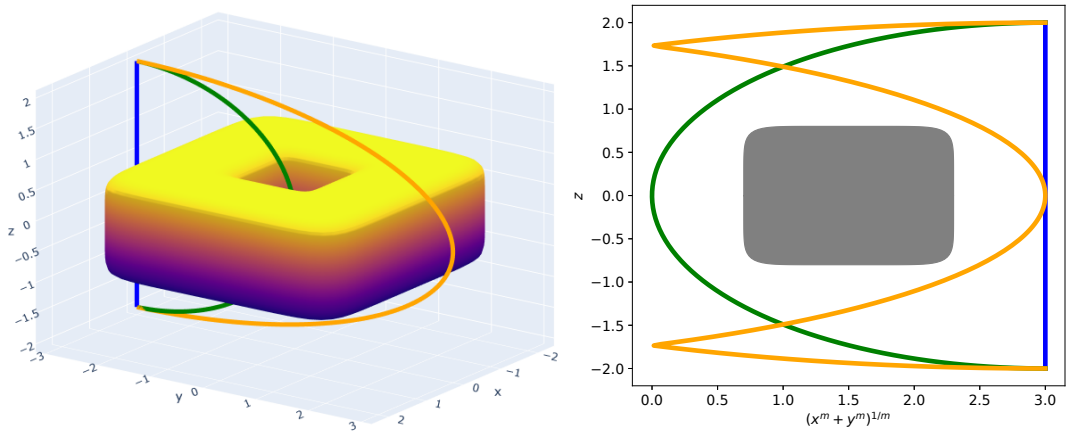


Figure 5.5: A super-toroid obstacle and three trajectories are shown in the left figure and their shapes in the embedding space are shown in the right figure, where the homotopy property of three trajectories still holds.

equivalent. We define $H^e(s,t) = f_i\left(H(s,t)\right)$, then $H^e(0,t) = r_1^e(t)$ and $H^e(1,t) = r_2^e(t)$. According to Lemma 6.3.1, $H^e(s,t)$ is obstacle-free, and the function $H^e(s,t)$ is still continuous because the embedding function f_i is continuous. Therefore $r_1^e(t)$ and $r_2^e(t)$ are also homotopy equivalent. ■

Corollary 5.3.3 *If two continuous trajectories $r_1(t) : [0,1] \to \mathbb{R}^3$ and $r_2(t) : [0,1] \to \mathbb{R}^3$ have the same initial and terminal points, and their embedded trajectories $r_1^e(t)$ and $r_2^e(t)$ are homotopy equivalent regarding the compressed obstacle, then $r_1(t)$ and $r_2(t)$ belong to the same homotopy class towards the original 3-dimensional super-toroid obstacle.*

Proof: We define the complementary embedding mapping

$$f_i^c\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} (x^{n_i} + y^{n_i})^{\frac{1}{n_i}} \\ z \\ \arctan(y,x) \end{bmatrix} := \begin{bmatrix} x_i^e \\ y_i^e \\ z_i^c \end{bmatrix},$$

then f_i^c is the generalized cylindrical coordinate transformation. We kindly assume $r_1(t)$ and $r_2(t)$ never attach the z-axis, then their complementary trajectories $r_1^c(t) = f_i^c(r_1(t))$ and $r_2^c(t) = f_i^c(r_2(t))$ are also continuous and well-defined. Because $r_1^e(t)$ and $r_2^e(t)$ are homotopy equivalent, there exists a continuous function $H(s,t)$ that is obstacle-free and connects $r_1^e(t)$

and $r_2^e(t)$. Then we define

$$H^c(s,t) = \bar{f}_i^c \left(\begin{bmatrix} H(s,t)_1 \\ H(s,t)_2 \\ (1-s) \cdot z_i^c(r_1(t)) + s \cdot z_i^c(r_2(t)) \end{bmatrix} \right),$$

where $\bar{f}_i^c : \mathbb{R}^3 \to \mathbb{R}^3$ is the mapping from the generalized cylindrical coordinate to the original Cartesian coordinate, thus $H^c(0,t) = r_1(t)$ and $H^c(1,t) = r_2(t)$. According to Lemma 6.3.1, $H^c(s,t)$ is obstacle-free. Because $\bar{f}_i^c$ is continuous, $H^c(s,t)$ is also continuous. Hence $r_1(t)$ and $r_2(t)$ are homotopy equivalent. ■

Similar to the 2D case, the state in a 3D environment is defined as $(x, y, z, s_1, \ldots, s_M)$, where (x, y, z) is the space location and $(s_1, \ldots, s_M)$ is the homology class label calculated in the 2-dimensional embedding space. To decrease the number of sampled states, we uniformly sample nodes around the obstacle as shown in Algorithm 9, and then construct the graph in the same way as Algorithm 8 except that the trajectory and obstacles need to be transformed to the embedding space according to (6.6).

5.3.3 Optimization Method: Value Iteration

Given an undirected graph built in the previous sections, the Value Iteration Algorithm (VIA) will then be implemented to yield the shortest path. Although VIA has been widely applied in stochastic shortest path problems, it is less efficient compared with the Dijkstra algorithm in deterministic situations. But we will show that the format of VIA can be embarrassingly paralleled and therefore outperform the Dijkstra algorithm on multi-core CPU or GPU.

Algorithm 9 Sample locations around the 3D obstacle

Require: obstacle parameters, hyper-parameter δ

Initialize empty set S

for $\theta \in \{\theta_1, \ldots, \theta_n\}$ and $s \in \{s_1, \ldots, s_n\}$

$\lambda \leftarrow (r_i + \delta)/((\cos(s)^{m_i} + \sin(s)^{m_i})^{1/m_i}$

$\mu \leftarrow ((\cos(\theta)^{n_i} + sin(\theta)^{n_i})^{1/n_i}$

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} \leftarrow \begin{bmatrix} (R_i + \lambda \sin(s)) \cos(\theta)/\mu \\ (R_i + \lambda \sin(s)) \sin(\theta)/\mu \\ \lambda \cos(s) \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \leftarrow diag(1/e_{ix}, 1/e_{iy}, 1/e_{iz}) M_i^{-1} \left(\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} - b_i \right)$$

Put $(x, y, z)^T$ to S

return S

Shortest path problems based on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be converted to reinforcement learning (RL) problems, which are infinite-horizon Markov decision processes (MDP) with a terminal state. To begin with, we use the vertex set $\mathcal{G}(\mathcal{V})$ to represent the state set of RL, and $A(\mathbf{s}_i)$ to denote the feasible actions set of the state $\mathbf{s}_i$. An action $a_{i,j} \in A(\mathbf{s}_i)$ will transfer the state from $\mathbf{s}_i$ to $\mathbf{s}_j$. Moreover, the return function of the state transfer is:

$$R(\mathbf{s}_i, a_{i,j}) = \begin{cases} \Lambda - E(\mathbf{s}_i, \mathbf{s}_j), & \text{if } \mathbf{s}_j \text{ is } \mathbf{s}_T \\ -E(\mathbf{s}_i, \mathbf{s}_j), & \text{else} \end{cases} \tag{5.9}$$

where $\mathbf{s}_T$ is the terminal state, $\Lambda \in \mathbb{R}$ is a large enough value and $E(\mathbf{s}_i, \mathbf{s}_j)$ is the Euclidean distance of two states' locations. A policy $\pi(a|\mathbf{s})$ describes a deterministic desired action for each state. Starting from an initial state $\mathbf{s}_1$, the policy π provides an infinite state sequence $\{\mathbf{s}_1, \mathbf{s}_2, \dots\}$ with return value sequence $\{r_1, r_2, \dots\}$. We define $\gamma \in (0, 1)$ as the discount factor, then the value function is defined as

$$V^{\pi}(\mathbf{s}_1) = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots = r_1 + \gamma V^{\pi}(\mathbf{s}_2) \tag{5.10}$$

Under the optimal policy π' which maximize the value function $V^{\pi'}$ for every states, we have $V^{\pi'}(\mathbf{s}_T) = \Lambda/(1-\gamma)$ and $V^{\pi'}(\mathbf{s}_0) = \sum_{i=0}^{n} \gamma^i E(\mathbf{s}_i, \mathbf{s}_{i+1}) + \gamma^{n+1} V^{\pi'}(\mathbf{s}_T)$, where $\{\mathbf{s}_i\}_{i=0}^{n}$ forms a feasible trajectory from $\mathbf{s}_0$ to $\mathbf{s}_T$ for any $\mathbf{s}_0 \in \mathcal{V}$. If we choose γ close enough to 1, then

$\{\mathbf{s}_i\}_{i=0}^n$ is the shortest path, and $V^{\pi'}(\mathbf{s}_0)$ approaches $V^{\pi'}(\mathbf{s}_T)$ minus the length of the shortest path from $\mathbf{s}_0$ to $\mathbf{x}_T$. Note that Λ should be larger than the longest shortest path from every state, otherwise the optimal state sequence of some states would trivially stay in place. The optimal trajectory is hereby given by

$$\mathbf{s}_{i+1} = \underset{\mathbf{s}_j}{argmax} \left(-E(\mathbf{s}_i, \mathbf{s}_j) + \gamma V^{\pi'}(\mathbf{s}_j) \right), \tag{5.11}$$

where $\mathbf{s}_i$ and $\mathbf{s}_j$ should be connected directly in the graph and therefore $a_{i,j} \in A(\mathbf{s}_i)$.

To obtain the optimal value function $V^{\pi'}$, we start from a initial estimated value function V_0^π and update it iteratively:

$$V_{n+1}^\pi(\mathbf{s}_i) = \underset{a_{i,j} \in A(\mathbf{s}_i)}{max} \left(R(\mathbf{s}_i, a_{i,j}) + \gamma V_n^\pi(\mathbf{s}_j) \right), \tag{5.12}$$

which is referred to as Value Iteration as Algorithm 10. Although it's proven that V_n^π will converge to $V^{\pi'}$ for any initial estimate, the reasonable initial value will dramatically reduce the iterations it needs. In practice, we choose $V_0^\pi(\mathbf{s}_T) = \Lambda/(1-\gamma)$ and $V_0^\pi(\mathbf{s}) = 0$ for other states. Within each iteration, the updating processes of states are independent, therefore no effort is needed to separate the updating into a number of parallel tasks and reach linear speedup, which is referred to as embarrassingly parallel.

Algorithm 10 Value Iteration Algorithm (VIA)

Require: a small number θ
1: Initialize V_0^π, $n \leftarrow 0$
2: **do**
3: $\quad \Delta \leftarrow 0$
4: $\quad$ **for each** $\mathbf{s}_i \in \mathcal{V}$:
5: $\quad\quad$ Update $V_{n+1}^\pi(\mathbf{s}_i)$ according to (5.12)
6: $\quad\quad \Delta \leftarrow \Delta + |V_{n+1}^\pi(\mathbf{s}_i) - V_n^\pi(\mathbf{s}_i)|$
7: $\quad n \leftarrow n + 1$
8: **while** $\Delta > \theta$
3. **return** V_n^π

5.4 Numerical Results

We evaluate the performance of the proposed method on various 2D and 3D cases, and compare the efficiency with cutting-edge algorithms [10]. For all simulations, we set the sample distance $\delta = 0.5$ in (5.6) and Algorithm 9, $\Gamma = 1$ for sampling candidate states, and $\Lambda = 1000$, $\gamma = 0.999$ in Algorithm 10. Figure 5.6 demonstrates the resulting shortest trajectories with respect to the given homology class. The 3-dimensional case is shown in Figure 5.7 with two obstacles.

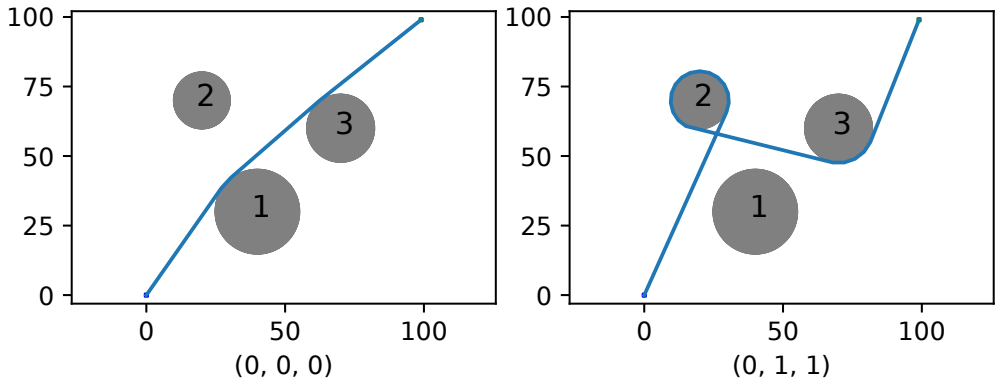


Figure 5.6: Shortest path with respect to the given homology class. The labels of obstacles are marked on it and the target homology label (s_1, s_2, s_3) is shown below each figure.

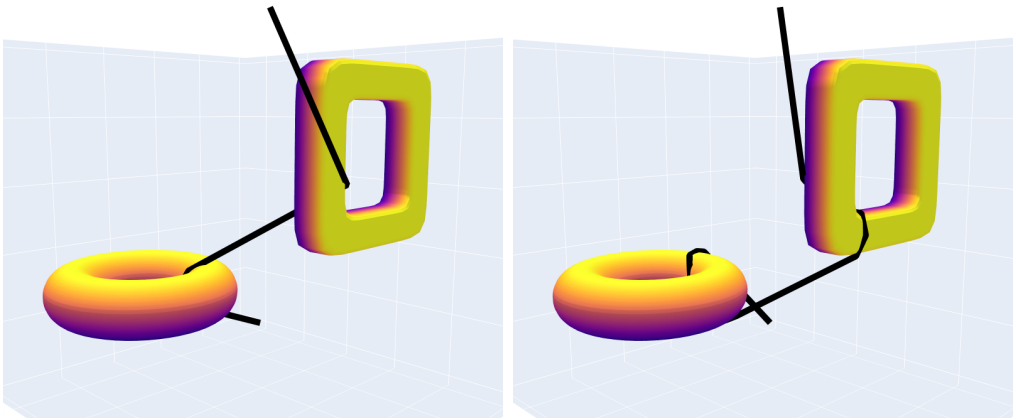


Figure 5.7: Four trajectories start from $(0, 0, 0)$ to $(100, 100, 100)$ with the terminal homology class label $(s_1, s_2) = (1, 1)$ and $(-1, -1)$, respectively.

The proposed method can be separated into two steps: graph building and then finding the shortest path using VIA. Given the graph, one can assign different states as the final states and then design the shortest path accordingly. Therefore, we demonstrate the efficiency

of these two steps separately. During building the graph, there exists a trade-off between precision and efficiency. However, given a fixed number of obstacles, the number of $\mathcal{G}(\mathcal{V})$ will be squared with an increased number of sampled points. Fortunately, Algorithm 8 can also be paralleled and, hereby, dramatically decrease the computational time.

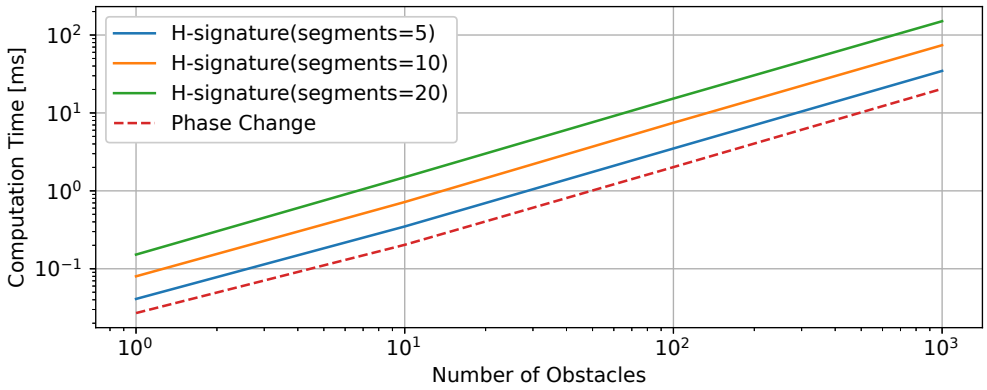


Figure 5.8: Comparison of the computation time of getting the homology class label in a 3D environment with random obstacles between H-signature and the proposed phase-change-based method.

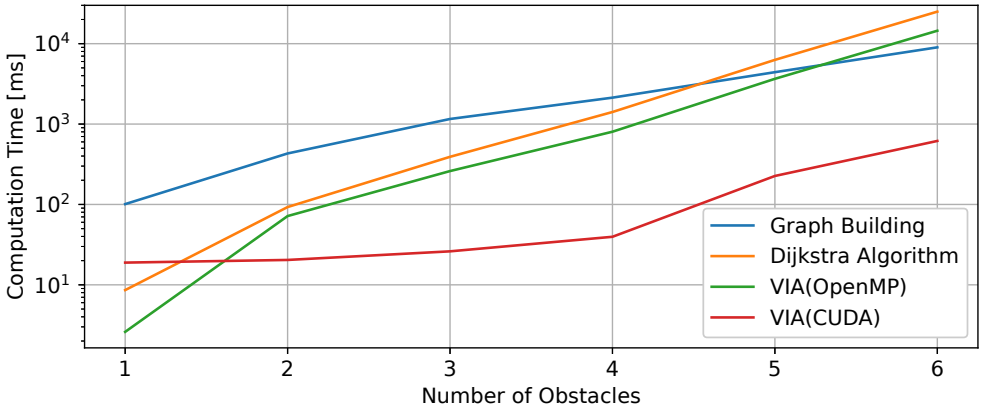


Figure 5.9: Comparison of the computation time in a 3D environment with random obstacles between Dijkstra Algorithm and VIA. The complete computation time is the sum of the graph-building time and the searching time.

Figure 5.8 demonstrates the cost time of getting homology labels in 3D environments. H-signature methods are required to discretize the skeleton of obstacles to line segments, and the low number of segments may lead to inaccurate modeling and wrong labels. The proposed method can achieve fast labeling without discretizing obstacles. Figure 5.9 illustrates the computation time of the proposed method to find the shortest path from $(0, 0, 0)$ to $(1, 1, 1)$ with the homology class label $(1, \ldots, 1)$. Obstacles are randomly generated in the 3D space,

and 100 locations are sampled for each obstacle. For parallel computing VIA using OpenMP, the 6-core Intel 9400F CPU is used. For the case using CUDA, the NVIDIA GTX 1660 GPU is leveraged. As the number of sampled locations increases linearly, the number of possible homology classes and the number of sampled states increase exponentially, therefore the calculation time also increases exponentially.

5.5 Conclusion

In this section, we introduced a novel path-planning method with homology class constraints that can handle 2-dimensional and 3-dimensional environments in a unified manner. The idea of the proposed method is to first sample states around obstacles and connect them according to the phase change to build the graph. For 3-dimensional super-toroid obstacles, we defined the embedding space in which the phase change can be calculated in a manner similar to the case of 2-dimensional obstacles. Then Value Iteration Algorithm is leveraged to find the shortest path in the synthesized graph. The analysis of the embedding function suggests that even higher dimensional obstacles can also be mapped to 2-dimensional embedding space and therefore be processed by the proposed method if it has a proper formulation. This is planned to be investigated in future work. On the other hand, the proposed global searching method lacks efficiency when the number of possible homology classes increases, which is also planned to be addressed in the future.

Chapter 6

Homotopy Method for Optimal Motion Planning with Homotopy Class Constraints and 3-dimensional Super-toroid Obstacles

6.1 Introduction

In Chapter 4, we introduced the HMHCC method that is able to efficiently deal with motion planning task with homotopy class constraints. However, it is limited to 2D obstacles. In Chapter 5, we explore features of super-toroid obstacles and utilize them to find the shortest path in 3D space with specific 3D obstacles and homology class constraints. In this section, we further utilize another feature of the super-toroid obstacle formula, differentiability, to propose a novel direction for addressing motion planning problems with homotopy class constraints that can be applied to general, possibly high-dimensional, nonlinear dynamical systems. Especially, we consider 3-dimensional super-toroid obstacles by embedding them into 2-dimensional space. The approach first adds an auxiliary control term to the original system, which turns a preset reference trajectory that is dynamically infeasible for the original system into a feasible one for the new extended system. Afterward, the approach *gradually* (cf. [80], [81]) eliminates the influence of the auxiliary control term so as to let the original input slowly take over the control of the system. As a result, the dynamically infeasible trajectory is gradually deformed to a feasible one that the original system is fully capable of tracking. In addition, the advocated approach is able to preserve the homotopy class for generated trajectories throughout the iterative synthesis process.

6.2 Mathematical Preliminaries and Problem Statement

6.2.1 Differentiable Super-toroid Obstacles Representations

In 3D environments, we consider a group of obstacles with super-toroid shapes, where the ith obstacle is described by

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} := M_i \, \mathrm{diag}(e_{ix}, e_{iy}, e_{iz}) \begin{bmatrix} x \\ y \\ z \end{bmatrix} + b_i, \tag{6.1}$$

$$\hat{z}^{m_i} + (R_i - (\hat{x}^{n_i} + \hat{y}^{n_i})^{\frac{1}{n_i}})^{m_i} - {r_i}^{m_i} \geq 0, \tag{6.2}$$

where R_i is the distance from the center of the tube to the center of the obstacle, r_i is the radius of the tube, and the rest of the parameters are defined in a similar way to that in (6.4). It is noted that m_i and n_i have to be positive even numbers. Some examples of the obstacle are shown in Figure 6.1

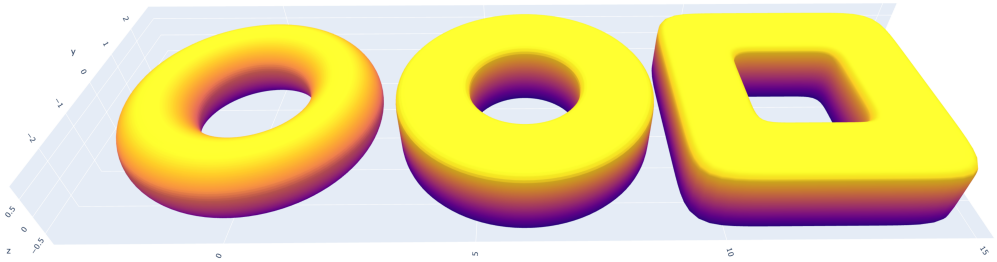


Figure 6.1: 3-dimensional obstacles with $(n, m) = (2, 2), (2, 8), (8, 8)$ from left to right, respectively.

According to equation (6.2), 3-dimensional super-toroid obstacles can be embedded into 2-dimensional space (x, y) by $x = \hat{z}$ and $y = (\hat{x}^{n_i} + \hat{y}^{n_i})^{\frac{1}{n_i}}$, which in turn yields the collision avoidance constraints in 2-dimensional embedding space

$$\left(\frac{x(t) - z_{ix}(t)}{r_{ix}}\right)^{k_i} + \left(\frac{y(t) - z_{iy}(t)}{r_{iy}}\right)^{k_i} - {R_i}^{k_i} \geq 0, \tag{6.3}$$

where $z(t) = [z_{ix}(t), z_{iy}(t)]$ denotes the center of the i^{th} obstacle at time t. $r_{ix}, r_{iy} \geq 1$ represent the object's spatial extensions, and $[x(t), y(t)]$ is the trajectory of the dynamical

system. $R_i > 0$ is a size constant and the exponent k_i is a positive even number. If $k_i = 2$, the obstacle is a circle or an ellipse, otherwise a rounded square, as shown in Figure 6.2.

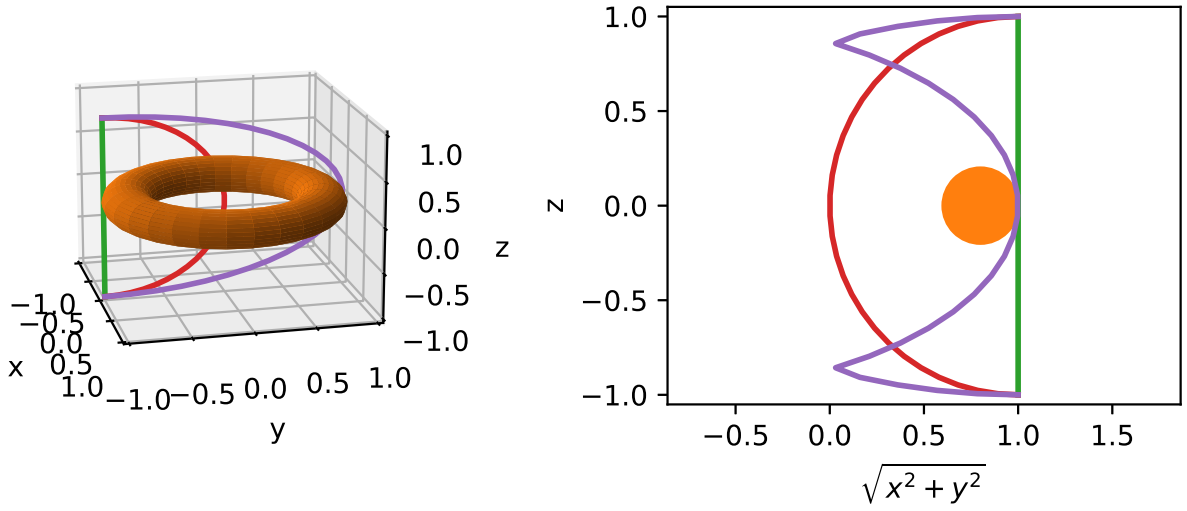


Figure 6.2: Investigation of three trajectories and a doughnut obstacle in 3-dimensional space. **Left:** In the $x-y-z$ view, according to the definition of the homotopy class, the green curve and purple curve belong to the same homotopy class, and the red curve belongs to another homotopy class. **Right:** In the 2-dimensional $\sqrt{x^2+y^2}-z$ view, the 3-dimensional doughnut obstacle is reformatted into a 2-dimensional circular obstacle, where the homotopy property of three trajectories still holds.

In 2D environments, the collision avoidance constraints regarding the ith obstacle can be written as

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} := M_i \operatorname{diag}(e_{ix}, e_{iy}) \begin{bmatrix} x \\ y \end{bmatrix} + b_i, \tag{6.4}$$

$$\hat{x}^{k_i} + \hat{y}^{k_i} - {R_i}^{k_i} \geq 0, \tag{6.5}$$

where $[x, y]$ denotes the allowed spatial trajectory of the dynamical system, M_i is the rotation matrix, $e_{ix}, e_{iy} \geq 1$ represent the spatial expansion factor along each axis, b_i is the offset of the center of the obstacle, $R_i > 0$ is a size constant and the exponent k_i is a positive even number, which regulates the shape of the obstacle. Particularly, when $k_i = 2$, the obstacle is a circle or ellipse, otherwise, it would be a rounded square.

6.3 Homotopy Method for 3-dimensional Obstacles

In the previous section, we provide the algorithm for synthesizing feasible trajectories with homotopy class constraints. However, the general auxiliary obstacle moving strategy is only adaptable to 2-dimensional space. In order to deal with 3-dimensional obstacles, such as gate-like ones, we describe the embedding method of 3-dimensional super-toroid obstacles which transfer obstacle space to 2-dimensional. In this section, we will describe the embedding method and prove it is homotopy equivalent after embedding.

The shape of super-toroid obstacles is defined in (6.1). We consider an embedding mapping $f_i : \mathbb{R}^3 \to \mathbb{R}^2$

$$f_i\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} (x^{n_i} + y^{n_i})^{\frac{1}{n_i}} \\ z \end{bmatrix} := \begin{bmatrix} x_i^e \\ y_i^e \end{bmatrix}, \tag{6.6}$$

which embeds the i^{th} super toroid obstacle to a super-ellipse centered at $(R_i, 0)$ with the radius r_i and the exponent m_i. Similarly, the dynamical system's trajectory will also be embedded in 2D space using f_i. According to Lemma 6.3.1, the continuous deforming and obstacle-free properties still hold in this embedding space as shown in Figure 6.3. Thus in the deforming process, if the trajectory never intersects the obstacle in embedding space, it also never intersects in the original 3D space.

Lemma 6.3.1 *Given a trajectory $r(t) : [0, 1] \to \mathbb{R}^3$ and the super-toroid obstacle, the trajectory is obstacle-free iff the embedded trajectory $(x_i^e(t), y_i^e(t))^\top = f_i(r(t))$ is obstacle-free for all $t \in [0, 1]$ in the embedding space:*

$$(R_i - x_i^e(t))^{m_i} + y_i^e(t)^{m_i} - r_i^{m_i} \geq 0.$$

Proof: Please see [33]. ■

Moreover, it leads to a mutual homotopy equivalent relationship between 2D embedding space and original 3D space. If the dynamical system's trajectory satisfies homotopy class constraints at the beginning, and the trajectory and obstacles never intersect in the embedding space during the deforming process, then it still satisfies homotopy class constraints. The aforementioned assertion is based on Corollary 6.3.4. The proof of Corollary 6.3.2 and Corollary 6.3.3 can be seen in [33].

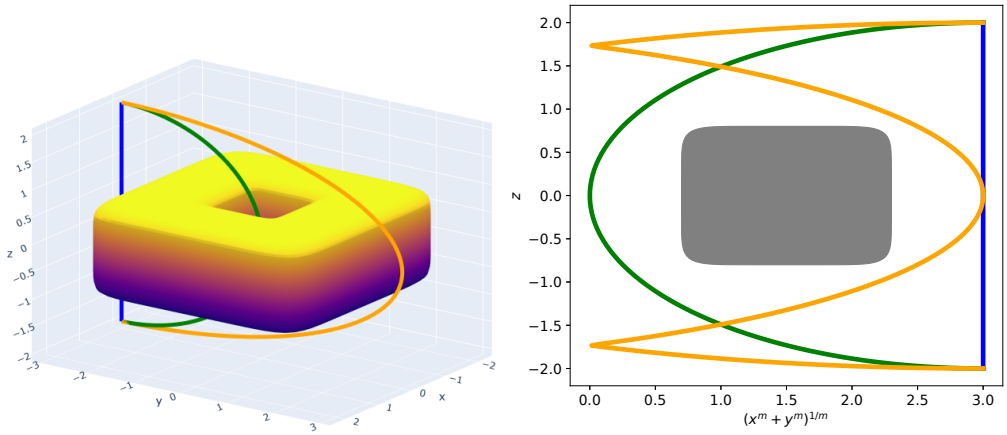


Figure 6.3: A super-toroid obstacle and three trajectories are shown in the left figure and their shapes in the embedding space are shown in the right figure, where the homotopy property of three trajectories still holds.

Corollary 6.3.2 *If two continuous trajectories* $r_1(t) : [0,1] \to \mathbb{R}^3$ *and* $r_2(t) : [0,1] \to \mathbb{R}^3$ *belong to the same homotopy class regarding the super-toroid obstacle, then their embedding trajectories* $r_1^e(t)$ *and* $r_2^e(t)$ *are also homotopy equivalent regarding the embedded obstacle.*

Corollary 6.3.3 *If two continuous trajectories* $r_1(t) : [0,1] \to \mathbb{R}^3$ *and* $r_2(t) : [0,1] \to \mathbb{R}^3$ *have the same initial and terminal points, and their embedded trajectories* $r_1^e(t)$ *and* $r_2^e(t)$ *are homotopy equivalent regarding the compressed obstacle, then* $r_1(t)$ *and* $r_2(t)$ *belong to the same homotopy class towards the original 3-dimensional super-toroid obstacle.*

Corollary 6.3.4 *If two continuous trajectories* $r_1(t) : [0,1] \to \mathbb{R}^3$ *and* $r_2(t) : [0,1] \to \mathbb{R}^3$ *have the same initial and terminal points are homotopy equivalent towards original multiple 3D super-toroid obstacles, and their embedded trajectories are keeping homotopy equivalent regarding the embedding obstacle while* ***deforming****, then they still belong to the same homotopy class towards the original 3-dimensional obstacles.*

Proof: According to Corollary 6.3.2 and Corollary 6.3.3, $r_1(t)$ and $r_1(t)$ never intersect 3D obstacles because they are always homotopy equivalent regarding the embedding obstacle. Hence they can **deform** to each other in an obstacle-free manner. According to Definition 2, the homotopy relationship holds for two trajectories in the original space. ■

Therefore, similar to the 2D case, we first synthesize the initial system trajectory p_0 by ignoring obstacle constraints using an NLP solver. Then we design embedding mapping

according to 6.6 for each obstacle, and the 3D task has been transformed into the 2D task with super-ellipse obstacles. The HMHCC method is then applied to the embedding space and acquires the optimal trajectory for the original task by iteratively solving the following NLP sequentially:

$$\begin{aligned} \underset{(U,X)}{\text{minimize}} \quad & \|U\|^2 \\ \text{subject to} \quad & x(0) = x_{\text{start}},\ x(N) = x_{\text{target}}, \\ & x(k+1) = F(x(k), u(k)), \\ & \|f_i(g(x(k))) - f_i(\hat{z}_i(k;s))\|^2 \geq R_i^2, \\ & \forall k \in \{0, ..., N\}, i \in \{1, ..., M\}, \end{aligned} \tag{6.7}$$

where f_i is the embedding mapping. Because f_i are differentiable functions, they can be directly integrated into the optimization-based framework and be solved using off-the-shelf solvers, such as the primal-dual interior point method. In our following experiments, the toolbox CasADi is utilized which provides the interface to the interior point solver and it shows more robustness than sequential quadratic programming solvers. The overall algorithm is shown in Algorithm 11.

Algorithm 11 Homotopy Method for Homotopy Class Constraints with Super-toroid Obstacles

Require: Obstacle trajectories $\{z_1(k), ..., z_M(k)\}$ and pre-established trajectory $r(k)$ which defines the required homotopy class.

1: Initialize inputs U^0 and state trajectory X^0 by solving (4.8) without considering obstacles.
2: Design embedding functions f_i according to (6.6).
3: Synthesis trajectories for obstacles $\{f_1(\hat{z}_1(k;s)), ...\}$ according to (4.5) with s large enough.
4: Set iteration step $m = 0$.
5: Set $s = \max(s - \Delta s, 0)$, $m = m + 1$
6: Solve optimal motion planning problem (6.7) with obstacles and initial $(U, X) = (U^{m-1}, X^{m-1})$, and the solution of it is noted as (U^m, X^m).
7: Repeat step $5 - 6$ until $s = 0$.

6.4 Numerical Results

In this section, we present our method for integrator models. The result demonstrates that our iterative synthesizing method can generate optimal and smooth trajectories while

satisfying homotopy class constraints. The model we adopt is shown below:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ dx \\ dy \\ dz \end{pmatrix} = \begin{pmatrix} dx \\ dy \\ dz \\ u_x \\ u_y \\ u_z \end{pmatrix},$$

where x,y and z indicate the position of the system, and u_x, u_y and u_z are control inputs. As a class dynamical model, the integrator model with an optimal energy target can generate a smooth trajectory, which can be utilized in the online path following the algorithm to steer real-world systems.

In the demonstrations, we apply the homotopy class constraints to the 3-dimensional position states. The facing angle ϕ is set to zero at both the start and target points. The total time is 10 seconds and the discrete-time interval is set as $\Delta T = 0.1$ second. In addition, the push distance s is decreased from 1.5 to 0 with a step of $\Delta s = 0.1$. The results are shown in Figure 6.4. Another simulation with two obstacles and a different homotopy class is carried out on the same system and setups except that s is initialized as 2.5, whose results are presented in Figure 6.5.

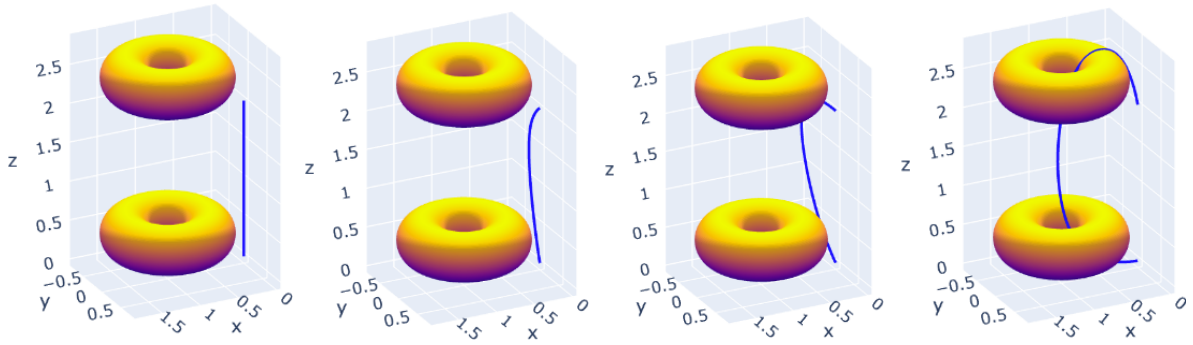


Figure 6.4: The trajectory of the dynamical system gradually deforms to the optimal one while satisfying homotopy class constraints with $s = 1.5, 1.0, 0.5, 0$, respectively.

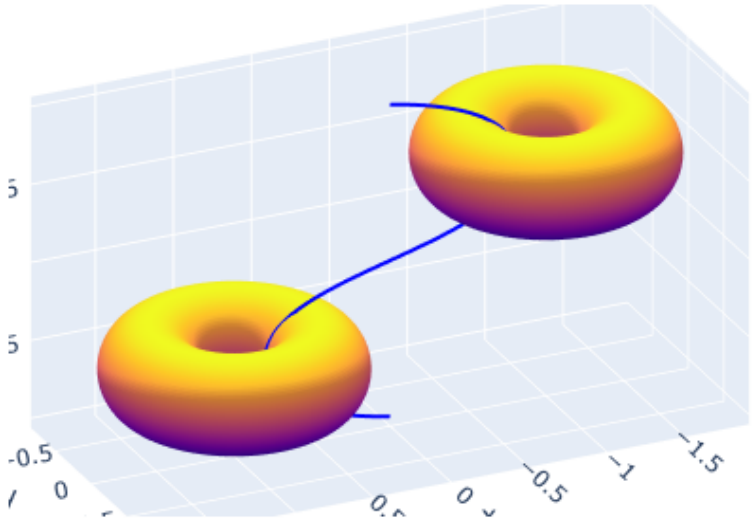


Figure 6.5: The trajectory of the dynamical system eventually deforms to the optimal one while satisfying homotopy class constraints.

6.5 Conclusion

In this section, we introduced the homotopy method for homotopy class constraints (HMHCC). It first synthesizes the initial trajectory for the dynamical system without considering the obstacles-free condition and homotopy class constraints. By choosing a large enough pushing distance, the homotopy property holds initially. Then the pushing distance gradually decreases and the system's trajectory deforms accordingly until the pushing distance reaches zero, and the resulting trajectory is optimal satisfying homotopy class constraints. To extend the aforementioned method to the 3D environment, we proposed the embedding method of gate-like 3D super-toroid obstacles. The proof of the embedding method is given and numerical results demonstrate that the proposed can correctly handle 3D homotopy class constraints. However, the current method is limited to specified super-toroid obstacles, and the problem of finding the general embedding formula is to be investigated in the future.

References

[1] M. J. Ablowitz, A. S. Fokas, and A. S. Fokas. *Complex variables: introduction and applications*. Cambridge University Press, 2003.

[2] A. Akhtar, S. L. Waslander, and C. Nielsen. Fault tolerant path following for a quadrotor. In *52nd IEEE Conference on Decision and Control*, pages 847–852. IEEE, 2013.

[3] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[4] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz. Receding-horizon lattice-based motion planning with dynamic obstacle avoidance. In *2018 IEEE conference on decision and control (CDC)*, pages 4467–4474. IEEE, 2018.

[5] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin. Learning quadrotor dynamics using neural network for flight control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4653–4660. IEEE, 2016.

[6] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[7] M. A. Belabbas and S. Liu. New method for motion planning for non-holonomic systems using partial differential equations. In *2017 American Control Conference (ACC)*, pages 4189–4194. IEEE, 2017.

[8] K. Bergman and D. Axehill. Combining homotopy methods and numerical optimal control to solve motion planning problems. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 347–354. IEEE, 2018.

[9] S. Bhattacharya. Search-based path planning with homotopy class constraints. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 1230–1237, 2010.

[10] S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, 2012.

[11] J. D. Biggs and H. Fournier. Neural-network-based optimal attitude control using four impulsive thrusters. *Journal of Guidance, Control, and Dynamics*, 43(2):299–309, 2020.

[12] H. G. Bock. Recent advances in parameteridentification techniques for ode. *Numerical treatment of inverse problems in differential and integral equations*, pages 95–121, 1983.

[13] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield. Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge. *Journal of Field Robotics*, 25(9):598–614, 2008.

[14] M. Brezak and I. Petrović. Real-time approximation of clothoids with bounded error for path planning applications. *IEEE Transactions on Robotics*, 30(2):507–515, 2013.

[15] S. Carpin and G. Pillonetto. Motion planning using adaptive random walks. *IEEE Transactions on Robotics*, 21(1):129–136, 2005.

[16] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh. Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots. *IEEE Robotics and Automation Letters*, 7(2):2819–2826, 2022.

[17] J. Chen, W. Zhan, and M. Tomizuka. Autonomous driving motion planning with constrained iterative LQR. *IEEE Transactions on Intelligent Vehicles*, 4(2):244–254, 2019.

[18] M. Cirillo. From videogames to autonomous trucks: A new algorithm for lattice-based motion planning. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 148–153. IEEE, 2017.

[19] K. Daniel, A. Nash, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[20] C. Edwards and S. Spurgeon. *Sliding mode control: theory and applications*. Crc Press, 1998.

[21] M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

[22] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang, and Z. Dawei. An improved a-star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems*, 17(5):1729881420962263, 2020.

[23] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli. Real-time motion planning of legged robots: A model predictive control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 577–584. IEEE, 2017.

[24] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[25] J. Ferguson, R. Shima, and K. Fujimoto. Path following via kinetic-potential energy shaping. In *59th IEEE Conference on Decision and Control, CDC 2020, Jeju Island, South Korea, December 14-18, 2020*, pages 763–768. IEEE, 2020.

[26] P. Foehn and D. Scaramuzza. Onboard state dependent lqr for agile quadrotors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6566–6572. IEEE, 2018.

[27] L. Greco, H. Mounier, and M. Bekcheva. An approximate characterisation of the set of feasible trajectories for constrained flat systems. *Automatica*, 144:110484, 2022.

[28] M. Greiff, A. Vinod, S. Nabi, and S. Di Cairano. Quadrotor motion planning in stochastic wind fields. In *2023 American Control Conference (ACC)*, pages 4619–4625. IEEE, 2023.

[29] I. Griva, D. F. Shanno, and R. J. Vanderbei. Convergence analysis of a primal-dual interior-point method for nonlinear programming. *Optimization Online (http://www. optimizationonline. org/DB HTML/2004/07/913. html)*, 2004.

[30] S. Han, B. Choi, and J. Lee. A precise curved motion planning for a differential driving mobile robot. *Mechatronics*, 18(9):486–494, 2008.

[31] X. Han, Z. Zheng, L. Liu, B. Wang, Z. Cheng, H. Fan, and Y. Wang. Online policy iteration adp-based attitude-tracking control for hypersonic vehicles. *Aerospace Science and Technology*, 106:106233, 2020.

[32] S. He, C.-H. Lee, H.-S. Shin, and A. Tsourdos. Minimum-effort waypoint-following guidance. *Journal of Guidance, Control, and Dynamics*, 42(7):1551–1561, 2019.

[33] W. He, Y. Huang, J. Qie, and S. Zeng. Value iteration algorithm for solving shortest path problems with homology class constraints. In *2023 IEEE 62nd IEEE Conference on Decision and Control (CDC)*. IEEE, 2023.

[34] J. P. Hespanha, D. Liberzon, and A. S. Morse. Overcoming the limitations of adaptive control by means of logic-based switching. *Systems & control letters*, 49(1):49–65, 2003.

[35] J. Horst and A. Barbera. Trajectory generation for an on-road autonomous vehicle. In *Unmanned systems technology VIII*, volume 6230, pages 866–877. SPIE, 2006.

[36] R. Horst and P. M. Pardalos. *Handbook of global optimization*, volume 2. Springer Science & Business Media, 2013.

[37] H. Hu, X. Feng, R. Quirynen, M. E. Villanueva, and B. Houska. Real-time tube mpc applied to a 10-state quadrotor model. In *2018 Annual American Control Conference (ACC)*, pages 3135–3140. IEEE, 2018.

[38] Y. Huang. *Control Design and Motion Planning for Unmanned Aerial Vehicles: A Data-Driven Scheme*. PhD thesis, Washington University in St. Louis, 2024.

[39] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on acoustics, speech, and signal processing*, 23(1):67–72, 1975.

[40] R. Kala. Homotopic roadmap generation for robot motion planning. *Journal of Intelligent & Robotic Systems*, 82(3-4):555–575, 2016.

[41] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for path planning: Articulated robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 3, pages 1764–1771. IEEE, 1994.

[42] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[43] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[44] H. Kim, M. Jordan, S. Sastry, and A. Ng. Autonomous helicopter flight via reinforcement learning. *Advances in neural information processing systems*, 16, 2003.

[45] S. Kim, K. Sreenath, S. Bhattacharya, and V. Kumar. Optimal trajectory generation under homology class constraints. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3157–3164. IEEE, 2012.

[46] Y. Kim, N. Cho, J. Park, and Y. Kim. Design framework for optimizing waypoints of vehicle trajectory considering terminal velocity and impact angle constraints. *Engineering Optimization*, pages 1–15, 2021.

[47] W. Koch, R. Mancuso, R. West, and A. Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21, 2019.

[48] P. Kokotovic and H. J. Sussmann. A positive real condition for global stabilization of nonlinear systems. *Systems & Control Letters*, 13(2):125–133, 1989.

[49] K. Kolur, S. Chintalapudi, B. Boots, and M. Mukadam. Online motion planning over multiple homotopy classes with gaussian process inference. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2358–2364, 2019.

[50] J. J. Kuffner and S. 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)*, volume 2, pages 995–1001. IEEE, 2000.

[51] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn. Path planning and obstacle avoidance for autonomous mobile robots: A review. In *Knowledge-Based Intelligent Information and Engineering Systems: 10th International Conference, KES 2006, Bournemouth, UK, October 9-11, 2006. Proceedings, Part II 10*, pages 537–544. Springer, 2006.

[52] A. Kushleyev and M. Likhachev. Time-bounded lattice for efficient planning in dynamic environments. In *2009 IEEE International Conference on Robotics and Automation*, pages 1662–1668. IEEE, 2009.

[53] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*, 14(6):912–925, 1998.

[54] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.

[55] H. Liu, G. Lu, and Y. Zhong. Robust lqr attitude control of a 3-dof laboratory helicopter for aggressive maneuvers. *IEEE Transactions on Industrial Electronics*, 60(10):4627–4636, 2013.

[56] S. Liu, Y. Fan, and M. Belabbas. Geometric motion planning for affine control systems with indefinite boundary conditions and free terminal time. *CoRR*, abs/2001.04540, 2020.

[57] S. Liu, Y. Fan, and M.-A. Belabbas. Affine geometric heat flow and motion planning for dynamic systems. *IFAC-PapersOnLine*, 52(16):168–173, 2019.

[58] F. M. Marchese. Multiple mobile robots path-planning with mca. In *International Conference on Autonomic and Autonomous Systems (ICAS'06)*, pages 56–56. IEEE, 2006.

[59] T. Marcucci, P. Nobel, R. Tedrake, and S. Boyd. Fast path planning through large collections of safe boxes. *arXiv preprint arXiv:2305.01072*, 2023.

[60] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *Int. Robotics Research*, 31(5):664–674, 2012.

[61] G. Moon and Y. Kim. Optimum flight path design passing through waypoints for autonomous flight control system. In *AIAA guidance, navigation, and control conference and exhibit*, page 5334, 2003.

[62] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME Int. Mech. Eng. Congress and Exposition*, 1995.

[63] J. Park and H. J. Kim. Fast trajectory planning for multiple quadrotors using relative safe flight corridor. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 596–603. IEEE/RSJ, 2019.

[64] S. Park, J. Deyst, and J. P. How. Performance and lyapunov stability of a nonlinear path following guidance method. *Journal of guidance, control, and dynamics*, 30(6):1718–1728, 2007.

[65] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542, 2012.

[66] A. Piazzi, C. L. Bianco, M. Bertozzi, A. Fascioli, and A. Broggi. Quintic g/sup 2/-splines for the iterative steering of vision-based autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 3(1):27–36, 2002.

[67] M. Pivtoraiko and A. Kelly. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7. Munich Germany, 2005.

[68] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[69] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, and J. Guichard. Lqr control for a quadrotor using unit quaternions: Modeling and simulation. In *CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing*, pages 172–178. IEEE, 2013.

[70] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics research*, pages 649–666. Springer, 2016.

[71] M. Rufli and R. Siegwart. On the application of the d* search algorithm to time-based planning on lattice graphs. In *Proc. of The 4th European Conference on Mobile Robotics (ECMR)*, pages 105–110. Eidgenössische Technische Hochschule Zürich, 2009.

[72] A. Rupenyan, M. Khosravi, and J. Lygeros. Performance-based trajectory optimization for path following control using bayesian optimization. In *60th IEEE Conference on Decision and Control, CDC 2021, Austin, TX, USA, December 14-17, 2021*, pages 2116–2121. IEEE, 2021.

[73] C.-K. Ryoo, H. Cho, and M.-J. Tahk. Optimal guidance laws with terminal impact angle constraint. *Journal of Guidance, Control, and Dynamics*, 28(4):724–732, 2005.

[74] F. Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation. (Masters Degree Project, KTH Royal Institute of Technology), 2015.

[75] Z. Shiller, Y.-R. Gwo, et al. Dynamic motion planning of autonomous vehicles. *IEEE Transactions on Robotics and Automation*, 7(2):241–249, 1991.

[76] H. Sira-Ramirez and S. K. Agrawal. *Differentially flat systems*. Crc Press, 2018.

[77] D. Soetanto, L. Lapierre, and A. Pascoal. Adaptive, non-singular path-following control of dynamic wheeled robots. In *42nd IEEE international conference on decision and control (IEEE Cat. No. 03CH37475)*, volume 2, pages 1765–1770. IEEE, 2003.

[78] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How. FASTER: Fast and safe trajectory planner for navigation in unknown environments. *IEEE Trans. Robotics*, 38(2):922–938, 2021.

[79] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of field Robotics*, 25(8):425–466, 2008.

[80] M. Vu and S. Zeng. Iterative optimal control synthesis for nonlinear switching systems. In *2021 American Control Conference (ACC)*, pages 998–1003. IEEE, 2021.

[81] M. Vu, S. Zeng, and H. Fang. Health-aware battery charging via iterative nonlinear optimal control syntheses. *IFAC-PapersOnLine*, 53(2):12485–12490, 2020.

[82] D. J. Webb and J. Van Den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE international conference on robotics and automation*, pages 5054–5061. IEEE, 2013.

[83] R. W. Wedderburn. Quasi-likelihood functions, generalized linear models, and the gauss—newton method. *Biometrika*, 61(3):439–447, 1974.

[84] I. H. Whang and T. W. Hwang. Horizontal waypoint guidance design using optimal control. *IEEE Transactions on Aerospace and Electronic Systems*, 38(3):1116–1120, 2002.

[85] J. Yao, C. Lin, X. Xie, A. J. Wang, and C.-C. Hung. Path planning for virtual human motion using improved a* star algorithm. In *2010 Seventh international conference on information technology: new generations*, pages 1154–1158. IEEE, 2010.

[86] W. Yao, B. Lin, and M. Cao. Integrated path following and collision avoidance using a composite vector field. In *58th IEEE Conference on Decision and Control, CDC 2019, Nice, France, December 11-13, 2019*, pages 250–255. IEEE, 2019.

[87] S. Zeng. Iterative optimal control syntheses illustrated on the brockett integrator. In *Proc. 11th IFAC Symposium on Nonlinear Control Systems*, 2019. submitted, PDF available at https://cpb-us-w2.wpmucdn.com/sites.wustl.edu/dist/d/928/files/2019/03/paper-1srvhy1.pdf.

[88] S. Zeng. Iterative optimal control syntheses illustrated on the brockett integrator. *IFAC-PapersOnLine*, 52(16):138–143, 2019.

[89] S. Zeng. On linearization, discretization, and numerical integration of nonlinearcontrol systems with applications to optimal control design. In *IFAC-PapersOnLine*, volume 52, pages 138–143, 2019.

[90] W. Zhao and T. H. Go. Quadcopter formation flight control combining mpc and robust feedback linearization. *Journal of the Franklin Institute*, 351(3):1335–1355, 2014.

[91] K. Zhou, L. Yu, Z. Long, and S. Mo. Local path planning of driverless car navigation based on jump point search method under urban environment. *Future Internet*, 9(3):51, 2017.