



POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGENERIA
DOCTORAL PROGRAMME IN INFORMATION TECHNOLOGY

OPTIMAL KINODYNAMIC PLANNING FOR AUTONOMOUS VEHICLES

Doctoral Dissertation of:
Başak Sakçak

Supervisor:
Prof. Gianni Ferretti

Tutor:
Prof. Luca Bascetta

The Chair of the Doctoral Program:
Prof. Andrea Bonarini

Year – 2017

Acknowledgments

First I would like to thank my advisors, Prof. Gianni Ferretti and Prof. Luca Bascetta, for their generous guidance, availability for numerous discussions and valuable suggestions, for all the interesting research that I had the opportunity to be part of and for giving me the freedom to pursue the work that truly interests me. I would also like to thank Prof. Maria Prandini, for her attentive corrections and suggestions which were indispensable for this work and for her availability and kindness.

I wish to thank Prof. Lucia Pallottino and Prof. Adnan Tahirovic, who reviewed this thesis, for their valuable comments on how to improve the work.

I owe years of support, fruitful discussions and cheering to my fellow PhD students and officemates in Polimi. I would especially like to thank Aida, Alessandro, Daniele, Hafsa, Manuela, Roberto, Riccardo, and Stefano, it would have been a much tougher ride without the company. I am thankful for having great friends, who have supported me throughout this journey, even from overseas; but I am especially indebted to Zhang Hankun, Kizla, Pier and Alejandro, whom I think of as my extended family, for sharing all good and bad times. I wish also to thank Arnau for his ability to make clouds disappear. I am grateful to Sali, for the joy that he brings to my life.

Lastly, I would like to thank my parents and my late grandmother. It is not even possible to state the extent of their contributions to my life.

Abstract

Motion planning is one of the most important factors affecting the functioning of an autonomous vehicle and aims to find a collision-free motion guiding the vehicle from an initial configuration to a final one. In particular, kinodynamic motion planning addresses this very issue, taking also into account the dynamics of the system so that the resulting solution would be executable by the vehicle. This thesis addresses the problem of optimal motion planning with particular emphasis on developing efficient planners while considering underlying realistic dynamic model.

The first contribution is a resolution optimal motion planner, RRT* with Motion Primitives (RRT*_MP), that samples the state space in a grid representation and generates a tree of trajectories using a database of motion primitives. Therefore, the computationally intensive part of solving for a steering action is carried to the preliminary phase of database generation by alleviating the computational load during planning. Furthermore, it allows to compute the true cost-to-go in order to guide the expansion of the tree, improving the convergence properties. The algorithm is proven to be asymptotically optimal as the grid resolution goes to zero and number of nodes goes to infinity. The (sub)optimality caused by the gridding can be tuned to compromise between the size of the database and this performance degradation.

The second part of the thesis focuses on the topological constraints on the kinodynamic motion planning. In particular, elaborates on the decoupled approach of systematically generating homotopy class constraints and imposing these constraints during the planning phase to obtain a set of

distinct local optimal trajectories belonging to different homotopy classes. The contributions in that case is twofold: first, an exact optimal planner based on homotopy class constraints and optimal control is proposed that to decompose the overall trajectory generation problem into simpler sub-problems and, second RRT*_MP is enhanced with homotopy awareness that allow generating distinct good trajectories.

Finally, the proposed algorithms are applied to a particular problem of landmark aware motion planning. For this, the motion planning problem is reformulated as a multi-objective planning problem, where the robot has to keep its connection with at least one landmark at all times, without explicitly interpolating a set of selected ones.

Contents

1	Introduction	1
1.1	Optimal Motion Planning	2
1.2	Optimal Motion Planning with Dynamics	3
1.3	Topology Aware Motion Planning	5
1.4	Landmark Aware Motion Planning	6
1.5	Statement of Contributions	7
1.6	Organization	9
2	Kinodynamic Motion Planning Problem	11
2.1	Kinodynamic Constraints	12
2.2	Free State Space	12
2.3	Problem Definition	13
2.3.1	Feasible Kinodynamic Trajectory Planning	14
2.3.2	Optimal Kinodynamic Trajectory Planning	14
3	Sampling Based Kinodynamic Planning	15
3.1	Basic Components	16
3.1.1	Sampling	16
3.1.2	Distance in State Space	16
3.1.3	Steering	17
3.1.4	Collision Detection	17
3.1.5	Abstract Form of Sampling Based Algorithms	19
3.2	Existing Algorithms	20
3.2.1	Randomized Sampling Based Algorithms	20

3.2.2	Towards Optimality: Optimal Rapidly-exploring Random Trees (RRT*) and Variants	22
3.2.3	Search Based Algorithms	27
3.3	Sampling Based Planning with Motion Primitives	27
3.3.1	Motion Primitives	28
3.3.2	Search Space Design	31
3.3.3	Motion Planning	32
3.4	Implementation Details	41
3.5	Completeness and Optimality Analysis	43
3.5.1	Probabilistic Completeness	44
3.5.2	Optimality of the Solution	45
3.6	Numerical Example	53
3.6.1	Unicycle	54
3.6.2	Results and Discussions	54
3.7	Conclusion	63
4	Planning with Homotopy Class Constraints	65
4.1	Homotopy in Motion Planning	66
4.2	Homotopy Aware Motion Planning Problem	69
4.3	Homotopy Class Identification	71
4.3.1	Partitioning of the Search Space	71
4.3.2	Cell Sequence Selection	73
4.4	Trajectory Generation	79
4.4.1	Exact Optimal Planner with Homotopy Class Constraints	79
4.4.2	Numerical Examples using the Exact Approach	82
4.4.3	Sampling Based Planner with Homotopy Class Constraints	87
4.4.4	Numerical Example using Sampling Based Approach	91
4.5	Conclusion	93
5	Landmark-Aware Motion Planning	95
5.1	Landmark-Aware Motion Planning Problem	96
5.2	Sampling Based Landmark-Aware Planner	99
5.3	Exact Optimal Planner with Connectivity Constraints	105
5.4	Conclusion	107
6	Conclusions and Prospective Works	111
	Bibliography	115

CHAPTER 1

Introduction

Autonomous vehicles have gained considerable attention in the field of robotics as the number of their applications - ranging from service robots to surveillance and to search and rescue missions - keeps increasing. Furthermore, the results of DARPA Urban Challenges show an immense progress for the autonomous navigation in urban areas. Motion planning is one of the most crucial factors affecting the functioning of autonomous vehicles.

In its most basic form, motion planning deals with the problem of finding a collision-free way to guide the vehicle from an initial configuration to a final one. However, with the complexity of the tasks that are in hand, the focus has shifted to even more intriguing aspects such as the introduction of an accurate dynamic model, inclusion of logical rules to govern the motion of an autonomous vehicle or finding an optimal method of reaching its goal when given an objective function.

This thesis addresses the optimal motion planning problem with a strong emphasis on inclusion of the underlying realistic dynamic system as an object of the planning. The proposed approaches consider arbitrary dynamics without imposing strong restrictions on their characteristics. Furthermore, the motion planning problem is also widened to encompass some particular aspects such as the role of the topology on the generated trajectories and

constraints imposed by the landmarks.

1.1 Optimal Motion Planning

Given the dynamics governing the motion of the vehicle, the general definition of *motion planning* is, “finding a trajectory that starts from an initial configuration, reaches the goal region while avoiding obstacles” [21,56,57]. In other words, the aim is to find a collision-free path to the goal and a corresponding velocity profile.

An algorithm is considered *complete* if it can return a solution in finite time, if one exists, or can report the nonexistence. The complete algorithms are called *exact methods* in the literature since they rely on the exact representation of the obstacles in the search space [57]. However, with the exact methods even the simplest problem is found to be PSPACE-hard [79].

Resolution complete algorithms represent this search space as an occupancy grid [1, 14] or a lattice [29, 39, 76] and apply a graph search using optimal graph search algorithms such as Dijkstra Algorithm [43] or A* [73] in order to find the optimal path from an initial state to a goal region. A* has been also used as a basis for improvement for several motion planning applications such as dynamic A* (D*) [87], Theta* [67], anytime D* (AD*) [64] or anytime repairing A* (ARA*) [63]. While efficiently solving an otherwise complex problem, they suffer from branching when the resolution of the grid gets finer and when dealing with high dimensional spaces¹.

Sampling based algorithms, emerged from the need for a planner for high-dimensional spaces as this problem had remained unsolved for many years. They relax the completeness by *probabilistic completeness*, which means that the probability of returning a solution is 1 as the number of iterations go to infinity and avoid the *curse of dimensionality*; however they can not report the nonexistence.

Representing the practical way to solve the motion planning problem, sampling based algorithms have so far dominated the planner debate, without being reserved for high dimensional spaces. Therefore the main sampling based planners Rapidly-exploring Random Trees (RRT) [53, 58] and Probabilistic RoadMap Planner (PRM) [13, 50] have been used effectively to address the motion planning problem for many different mediums ranging from industrial manipulators [53] to autonomous driving [54]. Both approaches return a solution, once it is found, without considering the quality. Several variations have been proposed to improve this aspect [90].

¹As a matter of fact, the search based algorithms presented here consider planning using a 2D grid.

However, optimality of the solution was not argued until the work of Karman et.al. [47] based on the percolation theory [15] and random geometric graphs [74], which not only proposed a new set of algorithms based on RRT and PRM with asymptotic guarantee on optimality but also proves that the probability of obtaining the optimal solution with these algorithms is 0.

Following the emergence of an optimal motion planner, several other approaches have been proposed to improve the convergence to an optimal solution by way of using informed sampling [31, 32], rejecting unpromising samples [42, 49] using Branch and Bound technique [55], propagating information to higher levels if the branch is a promising one (RRT[#]) [3].

1.2 Optimal Motion Planning with Dynamics

Most of the algorithms referred in the previous section have considered dynamics in the simplest form, i.e. holonomic or trivial dynamics. Therefore, the main focus has been on *optimal path planning* other than *optimal trajectory planning*.

Many planners that return optimal trajectories, in particular for dynamic environments, aim to solve the path planning problem first and then adapt a feasible trajectory to this collision-free path. Some algorithms in this class first compute a set of waypoints by applying a planner on a lower dimensional (2D or 3D search space) using any of the path planning methods described before and then impose continuity in the other dimensions - such as orientation, velocity and so on - by interpolating these waypoints using polynomials [80, 93], Bezier curves [33] or splines [23, 27]. Other approaches involve computing a coarse initial guess, in terms of a lower dimensional path and use numerical optimization to compute the optimal trajectory [78, 86].

For many of the dynamics, however, such a decoupled approach of first solving for a collision-free path and then adapting a feasible trajectory is not an easy process. Therefore, as it is first proposed by Donald et.al. [24], *kinodynamic motion planning* addresses the kinematics and the dynamics of the system at once, which involves directly searching the state space. This approach increases the complexity of the motion planning problem with the introduction of higher dimensional spaces and, most importantly, differential constraints imposed by the system dynamics.

Enhancing search based algorithms with kinodynamic constraints, [62] and [77] have further extended the graph search algorithms to search in a state space considering also the proper dynamics of the system. In addition, Dolgov et. al. [23] augmented the A* algorithm with a heuristic to generate

drivable waypoints with respect to the vehicle dynamics that are further interpolated using splines for smoothness.

A considerable amount of research has been dedicated to finding how to use sampling-based algorithms in the context of kinodynamic planning owing to their effectiveness with high dimensional spaces [40, 46, 58, 75, 91]. When it comes to kinodynamic planning, the main focus is on developing efficient steering functions, which correspond to a local trajectory generator without taking into account the obstacles. Since the sampling-based planners randomly sample the state space and later connect these samples to a graph structure, steering, in other words solving the Two Point Boundary Value Problem (TPBVP), is an important subroutine. Computing a steering action corresponds to solving for a trajectory connecting two states, and the solution of this TPBVP can be obtained from the optimal control theory [59] or from numerical optimization [8]. However, for many dynamic systems that are commonly used in autonomous vehicles or mobile robots, solving for a TPBVP is not a trivial task. Therefore many sampling based approaches on kinodynamic planning focus on improving this aspect. For a limited number of dynamic systems, the analytical solution can be embedded in the planning framework using common vehicle models [46] such as double integrator or Dubins vehicle [25] or computing the analytical solution for a system with linear dynamics [91]. Xie et. al. [92] use Sequential Quadratic Programming (SQP) [12] as an optimization tool for solving the TPBVP numerically and implemented it as a subroutine for Batch Informed Trees (BIT*) [31].

There has been a research on avoiding the exact steering, i.e. solving for a TPBVP, by sampling from the control space and forward integrating the system dynamics. This approach is inherent for RRT and Expansive Space Trees (EST) [40] since the methodology of building a tree is based on extending it towards random samples, therefore they sample piecewise constant controls to compute a steering action without solving the TPBVP. However, it is important to highlight that such an approach is not possible for RRT* or any of its variants due to its dependency on an underlying roadmap. Some works involve computing an approximation of the exact steering [34, 41, 75], or avoiding the need for an exact steering using a pruning approach [42] without a provable guarantee on optimality. Based on EST some recent works such as Stable Sparse RRT* (SST*) consider *asymptotic near optimality* by forward propagating the dynamic system [60, 61]. The idea is to iteratively sample a set of controls and final times instead of solving the inverse problem; near optimality is guaranteed by pruning nodes from a neighborhood that is a logarithmic function of the

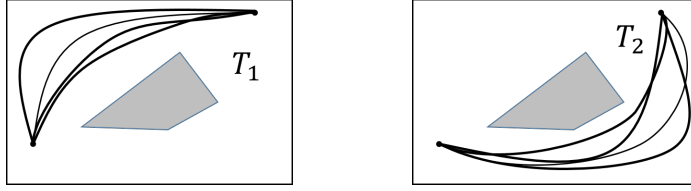


Figure 1.1: T_1 and T_2 define two sets of trajectories that belong to two distinct homotopy classes. All the trajectories in T_1 belong to the same class, the same for T_2 .

tree cardinality.

Exact steering for any arbitrary dynamic system is often more involved compared to sampling from the control space or steering without a guarantee on the final state. However, for some dynamic systems it is crucial to have control over the destination state due to some geometric constraints to be respected [17]. Therefore, one of the aims of this thesis is to contribute to the library of sampling-based planners in terms of developing a planner with an efficient exact steering for any arbitrary dynamic system using the pre-computed solutions of the TPBVPs.

1.3 Topology Aware Motion Planning

Defining the topology, given a search space, has been in the interest of planning with exact (combinatorial) approaches as these methods depend on the explicit representation of the search space [57]. Therefore, the concept has been studied especially for path planning algorithms; however it is generally not favored due to its catastrophic complexity.

A homotopy class in the context of trajectory planning is the set of all trajectories that can be morphed to each other without any discontinuity (Figure 1.1). In other words, trajectories that belong to the same class avoid obstacles in the same way. The use of homotopy information in planning applications is twofolds:

- for a given search space, identifying the distinct homotopy classes and generating distinct trajectories or paths,
- given a homotopy class, constraining the resulting path topologically.

Different methods have been considered to represent the topology of a given search space: *triangulation* [22], *cell-decomposition* [19], *visibility graph* [57] or *reference frame methods* [44]. These methods enable the representation of the topology of the search space in terms of a graph or

a roadmap. Obtaining a graph-based representation allows a systematic generation of homotopy classes using graph search algorithms [37, 44, 71].

On the other hand, some approaches apply a simple search algorithm, such as A* or RRT, on an arbitrary representation of the space and get the topology information in terms of homotopy classes through identification of distinct paths by assigning a class invariant signature [10, 36, 52].

Approaches taking into account the topology of the search space can generate several locally optimal trajectories and select the best one among them [71] or, if given the information about the class containing the optimal solution speed up the computation by focusing on this class. In addition, one might be interested in obtaining *k-shortest paths*. Without the homotopy class constraints these paths would often avoid the obstacles in the same manner; in other words fall into the same class. However, identifying distinct homotopy classes also allow generating *k*-distinct local optimal trajectories [52, 85].

Other than allowing computation of distinct trajectories that are locally optimal, homotopy class constraints can also restrict the trajectory to avoid the obstacles in a specific manner. This can interest a various number of robotics or autonomous vehicle applications, ranging from favoring some areas for surveillance [16], to efficiently deploy a group of agents (such that each would move following different homotopy class) and restricting dangerous zones [82], sharing autonomy where the user makes only high level decisions [52] or for tactical reasoning [36].

1.4 Landmark Aware Motion Planning

With the continual increase in the number of tasks that are handled by the help of autonomous vehicles, the role of the planner has been changing from returning optimal trajectories that are collision-free and dynamically feasible to considering advanced tasks that gradually exclude the human operator from the navigation loop.

One of such problems that is also addressed in this thesis is to generate motion plans that consider the connection between the autonomous vehicle and a signal emitting base (Figure 1.2). In particular, the problem restricts the vehicle or the robot to be within a radius from the center of the base and the planner has to guarantee that over course of motion this connection is not broken or a predefined distance is not exceeded while moving from one base to another. The problem can also generalize to any type of landmark that can be defined similarly using a center and a connection radius and that should be perceived by the robot.

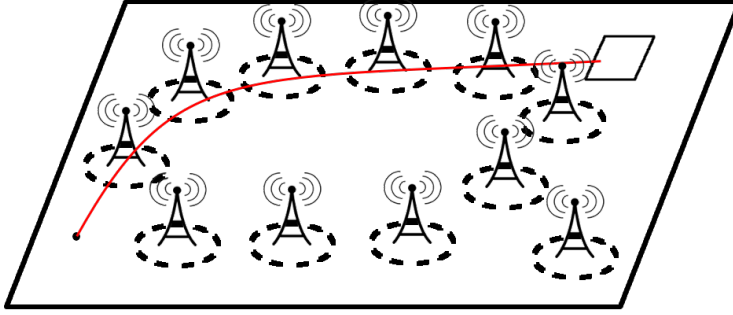


Figure 1.2: *Landmark aware motion planning.*

This problem interests various application areas. To illustrate, for autonomous vehicles used in surveillance or search and rescue missions, it is important that the robot is always in communication with the bases to transmit data or to receive tactical information. A similar case is considered for multi agent path planning by [68], where a group of wireless robots should connect to a radio base to transmit the video obtained from exploration, maintaining single-hop or multi-hop connections.

Furthermore, for mobile robots or autonomous vehicles that work in Global Positioning System (GPS) denied environments, such as indoor robots or autonomous underwater vehicles, it is essential to maintain connection with a signal emitting beacon for improved localization. Without loss of generality, this definition of the problem can be applied to any artificial landmark that can be perceived from a limited range. Some instances that involve selecting a set of artificial landmarks as visual cues for navigating in an GPS denied environment are proposed for unmanned aerial vehicles [4] and indoor mobile robots [6, 94] to generate a trajectory that interpolates a set of selected landmarks. The main difference between the problem considered in this thesis is that landmarks are considered as constraints to satisfy but not waypoints to interpolate, which is inherently more challenging.

1.5 Statement of Contributions

The main contributions of this thesis is on motion planning algorithms that consider the underlying dynamic system precisely, that are computationally effective to be able to re-plan in response to dynamic environments, consider additional constraints imposed by the topology and by the specific

tasks. More detailed, the contributions are as follows:

- A sampling-based motion planning approach is proposed which integrates a precomputed database of trajectories into the sampling-based planning framework to alleviate the computational load introduced by solving TPBVPs. The method is applicable to virtually any kind of dynamic system with various complexity that emerge in the context of motion planning for autonomous vehicles. Furthermore, it can be easily modified to address different problems characterized by various constraints and objectives. This is indeed verified using a challenging problem in the last chapter.
- The analysis of the proposed approach in terms of completeness and optimality is provided. Since the proposed algorithm makes use of a grid-based representation of the state space, it causes some performance degradation. Therefore, an upper bound for this (sub)optimality is computed and it is shown to be asymptotically optimal as the grid resolution goes to infinity.
- In order to generate distinct trajectories in terms of the homotopy classes they belong, a systematic approach to the selection of a number of geometric constraints based on existing decomposition techniques is proposed. Furthermore, the relationship between these geometric constraints and the corresponding homotopy classes is analyzed
- Using the obtained geometric constraints, an exact optimal motion planner is proposed based on the homotopy class constraints. Consequently, the sampling-based motion planning approach is enhanced with homotopy awareness as it allows to impose specific topology characteristics and, at the end, generating homotopically distinct trajectories. Moreover, a homotopy-based informed sampling method is also proposed to improve the convergence of the algorithm
- Finally, both planners are applied to a particular problem of landmark-aware motion planning problem. As a result, the selection of a sequence of landmarks is formalized in the framework of kinodynamic motion planning without restricting them as waypoints and using a coupled approach of generating dynamically feasible trajectories with this specific objective.

1.6 Organization

The rest of the thesis is organized as follows: **Chapter 2** provides the formal definition of the kinodynamic planning problem. In **Chapter 3** a general framework of the sampling based algorithms is identified and the basic concepts structuring this framework are explained. A novel sampling-based algorithm that depends on a database of offline computed solutions to TPBVPs is explained. Consequently, the completeness and optimality properties are analyzed. **Chapter 4** proposes an efficient way of selecting promising homotopy class constraints and an exact algorithm based on homotopy information. It furthermore extends the sampling based approach to include homotopy awareness. **Chapter 5** provides the implementation of both approaches to the landmark aware trajectory planning problem. Finally, **Chapter 6** draws some concluding remarks and explains the prospective future works.

CHAPTER 2

Kinodynamic Motion Planning Problem

Early methods on motion planning considered the geometric aspect of the problem, i.e., the kinematic constraints arouse by the obstacles. For many robots, focusing only on kinematics could result in collision-free paths that are impossible to be executed by the actual system. In particular, for systems that are differentially constrained, such a decoupled approach makes it difficult to turn a collision-free path into a feasible trajectory. In order to tackle this problem, Donald et al. [24] proposed the idea of *kinodynamic planning*, which combines the search for a collision-free path with the underlying dynamics of the system, so that the resulting trajectory would be feasible.

For most motion planning applications, the solution to the planning problem should not only be feasible and collision-free, but also satisfy some properties such as, e.g., reaching the goal in minimum time, minimizing the energy consumption, and maximizing safety. These additional requirements have shifted the focus from simply designing collision-free and feasible trajectories to finding optimal ones.

There are many different approaches that are designed to solve the optimal kinodynamic planning, all having different advantages and drawbacks depending on the constraints imposed by the dynamics and by the planning

environment. However, it is possible to define a general problem that can be addressed using different approaches. Therefore, this chapter is dedicated to the formalization of the feasible and the optimal kinodynamic motion planning problems and explains the peculiar aspects of the kinodynamic motion planning. Note that, throughout the thesis these definitions are modified to accommodate additional specifications while keeping the formalizations here presented, at heart.

2.1 Kinodynamic Constraints

Kinodynamic planning, first introduced by Donald et.al. [24] refers to planning with differential constraints when the solution to the motion planning problem further satisfies the bounds on the velocity and acceleration or higher order derivatives of the configuration. Informally, this corresponds to traversing the state space of a dynamic system respecting the kinodynamic constraints. Kinodynamic constraints can be categorized under three main groups:

Differential constraints: Let the compact sets $\mathcal{S} \subset \mathbb{R}^d$ and $\mathcal{U} \subset \mathbb{R}^m$, with $d \geq 2$ and $m \geq 1$, represent the state and the control spaces, respectively for the following dynamical system

$$\dot{s}(t) = f(s(t), u(t)) \quad (2.1)$$

where $u(t) \in \mathcal{U}$ is the vector of control inputs and $s(t) \in \mathcal{S}$ is the state vector. Then, the differential constraints refer to the system dynamics given by the state transition equation (2.1).

Nonholonomic constraints: Nonholonomic constraints are the constraints that can not be integrated to impose a constraint on the position components, in other words the system that can not be reduced in order.

Hard bounds: This corresponds to virtually any bound on the derivatives, such as the limits on velocity, acceleration or torque.

Note that, kinodynamic planning does not necessarily involve nonholonomic constraints nor bounds. However, since the main focus of this thesis is on mobile robots and autonomous vehicles they are naturally considered as a part of the planning problem.

2.2 Free State Space

Let the compact set $Q \subset \mathbb{R}^{d_q}$, where $2 \leq d_q \leq d$, denote the configuration space of a rigid body. Then, the free configuration space Q_{free} represents

the set of configurations that are collision free, i.e. $Q_{free} := Q/Q_{obs}$ where Q_{obs} represents an open subset of Q comprising the configurations that lead to collisions. The state of the dynamic system, s , comprises the configuration of the robot and the first order derivatives, such that $s = (q, \dot{q})$ and $q \in Q$. The state could also include higher order derivatives, if necessary. Let $\kappa : \mathcal{S} \rightarrow Q$ be a function that maps the state, $s \in \mathcal{S}$ to the corresponding configuration, $q \in Q$, such that $\kappa(s) = q$. Similarly, the free state space, $\mathcal{S}_{free}$, is defined as the set of states that are collision free, i.e. $\mathcal{S}_{free} := \{s \in \mathcal{S} | \kappa(s) \in Q_{free}\}$.

If the state variables have further bounds, namely limits on velocity or bounds on the workspace, the definition of the *free state space* should be modified to include these bounds. Let $\mathbf{h}(s) \leq 0$ be a set of inequalities with cardinality n_h denoting the upper and lower bounds on the state variables, then the *free state space* can be represented as

$$\mathcal{S}_{free} := \{s \in \mathcal{S} \mid \kappa(s) \in Q_{free} \wedge \mathbf{h}(s) \leq 0\}$$

2.3 Problem Definition

The main focus on this thesis is motion planning with kinodynamic constraints and in particular for state spaces that contain first order derivatives, i.e. velocity components, therefore the resulting solution is a trajectory. At the end the returned solution includes a collision free path and a velocity profile simultaneously. Following definition formalizes the description of a trajectory.

Definition 2.1 (Trajectory). *A trajectory $\sigma(t) : [0, T] \rightarrow \mathcal{S}$ of duration T is a function generated by applying a control function $\tilde{u} : [0, T] \rightarrow \mathcal{U}$ by forward integrating the state equation in 2.1 starting from an initial state, such that $\sigma(0) = s_0$.*

Assuming the system has bounds on the actuation, the free control space $U_{free} = \{u(t) \in \Omega \subset U\}$ is introduced, where Ω is the set of admissible controls.

The problem instance is defined as a tuple of free state space, admissible controls, initial state and a set of states defining the goal region $\mathcal{S}_{goal} \in \mathcal{S}_{free}$, i.e., $(\mathcal{S}_{free}, \mathcal{U}_{free}, s_0, \mathcal{S}_{goal})$. Moreover, trajectories can be subjected to geometric constraints expressed in terms of a function $\phi : \mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}^{n_\phi}$ as $\phi_l \leq \phi(s(t), u(t), t) \leq \phi_u$, where $\phi_l, \phi_u \in \mathbb{R}^{n_\phi}$ are represent the lower and upper bounds on the path constraints.

2.3.1 Feasible Kinodynamic Trajectory Planning

The *feasible kinodynamic trajectory planning problem* is formalized as finding a trajectory $\sigma(t) : [0, T] \rightarrow \mathcal{S}$ from a set of dynamic states and controls of a system over a duration $[0, T]$, i.e., $s : [0, T] \rightarrow \mathcal{S}$ and $u : [0, T] \rightarrow \mathcal{U}$, such that the trajectory

- starts from an initial state $\sigma(0) = s_0$;
- reaches the goal region, i.e., $\sigma(T) \in \mathcal{S}_{goal}$;
- avoids collisions with obstacles, and respects the bounds on the state variables, i.e., $\sigma(t) \in \mathcal{S}_{free}, \forall t \in [0, T]$;
- fulfills the actuation bounds, i.e., $u(t) \in \mathcal{U}_{free}$;
- satisfies the algebraic path constraints, if any, i.e., $\phi_l \leq \phi(\sigma(t), u(t), t) \leq \phi_u, \forall t \in [0, T]$.

2.3.2 Optimal Kinodynamic Trajectory Planning

In most of the applications, the solution to the trajectory planning problem has to satisfy some desirable properties, e.g., reaching the goal in minimum time, by minimizing the energy consumption or by maximizing the safety. These requirements shift the focus from finding *any feasible trajectory* to finding the *optimal trajectory* for a given objective function.

Given the same problem instance $(\mathcal{S}_{free}, \mathcal{U}_{free}, s_0, \mathcal{S}_{goal})$ as an input, the *optimal kinodynamic motion planning problem* is formalized as finding a feasible trajectory $\sigma^*(t) : [0, T^*] \rightarrow \mathcal{S}$ from a set of dynamic states and controls of a system over a duration $[0, T^*]$, i.e., $s^* : [0, T^*] \rightarrow \mathcal{S}$ and $u^* : [0, T^*] \rightarrow \mathcal{U}$, that minimizes an objective function in the form

$$\min_{u(t)} J = \int_0^T g(s(t), u(t)) dt \quad (2.2)$$

where g is an arbitrary function, depending on the specific motion planning problem.

Note that, if $g(s(t), u(t)) = 1$ then the problem corresponds to minimum time optimization problem. Indeed, the optimal trajectory duration, T^* , is one of the optimization variables of the problem.

CHAPTER 3

Sampling Based Kinodynamic Planning

Sampling-based approaches first emerged to handle systems with high dimensional configuration spaces. However, they quickly became one of the most popular approaches in the planning literature, representing the practical way to tackle the problem. The basic idea is to sample states (nodes) in the continuous state space and connect these nodes with trajectories in the collision-free space, building a roadmap in the form of a graph or a tree of feasible trajectories. Hence, their practicality is limited to the assumption that solving the planning problem between two nodes without considering any obstacles is trivial. However, this assumption is often not satisfied when it comes to kinodynamic constraints which introduce the need to solve for TPBVPs in order to generate trajectories.

This chapter explains with sampling-based approaches, focusing especially on kinodynamic motion planning. First, basic components of sampling based algorithms are provided, following which the related works and the background on sampling-based approaches are reviewed. Consequently, the novel algorithm of `RRT*_MotionPrimitives` will be introduced with the analysis regarding the specific properties of the planner, which strives to exploit the advantages of a sampling based approach by improving the subprocess of connecting two nodes which requires solving

for a TPBVP. Finally, a numerical validation is provided to show the effectiveness of the algorithm with the corresponding discussions.

3.1 Basic Components

Before introducing the existing algorithms in this class and explaining the proposed algorithm, some basic concepts shared between the sampling-based approaches will be explained and an abstract framework for sampling-based algorithms will be provided.

3.1.1 Sampling

The procedure `Sample()` returns random samples from a selected state space and these samples are assumed to be independent and identically distributed (i.i.d.). In the general case, `Sample( $S_{free}$ )` samples the free state space of the dynamic system. However, as it will be clear in the following sections of this thesis, different subsets of this state space can be purposefully selected to accommodate different needs emerging from the specific motion planner.

3.1.2 Distance in State Space

Almost all the sampling-based algorithms use the notion of *nearness*, which means that, with a given point in the state space, it is necessary to decide the nearest point or a set of nearby points. This necessity requires the definition of a *distance metric*. For planning in Euclidean spaces, the choice for this distance metric can simply be the Euclidean distance between two points. For example, for a rigid body translating on a 2D plane, the shortest distance between two points is the straight line connecting them. However, when the rigid body is allowed to rotate, the configuration space becomes a manifold. There are efficient ways to compute the distances in the case of manifolds; for example, by using the weighted Euclidean distances [57]. For kinodynamic planning, this approach can be further extended to state spaces [58]. However, as the state space gets more complex, like the ones containing the first and/or second order derivatives of the configuration, it becomes more difficult to assign meaningful weights.

The ideal way to measure the distance between two states in the state space is to define a distance function that computes the "*cost-to-go*" from $s \in S$ to $s' \in S$. The distance between s and s' in that case can be calculated by the cost of the trajectory connecting these two states [57]. It should

be noted that this function may not be a true metric since the cost of the trajectory connecting these two states may not be symmetric.

3.1.3 Steering

Steering, as denoted by the procedure $\text{Steer}(s, s')$, refers to computing a trajectory that starts from an initial state s and brings the system towards s' returning s_{new} . Depending on the type of the steering action, s_{new} can be exactly s' or it can be a new state that is close to s' . State based steering solves the inverse problem of interpolating the states s and s' exactly, which corresponds to solving the TPBVP. A second way of steering can be identified as trying a number of admissible controls and choosing the one that brings the system closer to s' , without explicitly interpolating the two states. However, most of the sampling based algorithms that have a guarantee on optimality requires a reasoning over the underlying roadmap or graph and, therefore, require the nodes to be connected exactly and optimally, which refers to *exact optimal steering*.

Definition 3.1 (Exact Optimal Steering). *Given a cost function that assigns a non-zero cost to each non-trivial feasible trajectory, i.e. $\text{Cost}(\sigma) : \Sigma \rightarrow \mathbb{R}_{\geq 0}$, where Σ is the set of feasible trajectories, the state based optimal steering is the problem of finding a feasible trajectory $\sigma^*(t) : [0, T^*] \rightarrow \mathcal{S}$ from a set of dynamic states and controls of a system over a duration $[0, T^*]$, i.e. $s^* : [0, T^*] \rightarrow \mathcal{S}$ and $u^* : [0, T^*] \rightarrow \mathcal{U}$, such that the trajectory*

- *starts from an initial state $\sigma(0) = s$;*
- *reaches the final state $\sigma(T^*) = s'$;*
- *fulfills the actuation bounds, i.e., $u(t) \in \mathcal{U}_{\text{free}}$;*
- *minimizes the cost function $\text{Cost}(\sigma)$*

By definition, computing a steering action implies solving the motion planning without considering any obstacles, which is assumed to be a trivial operation. Therefore, most algorithms reviewed in the following section focus on improving the exact optimal steering procedure, as it is the most crucial subroutine for the algorithms in this class.

3.1.4 Collision Detection

The most significant advantage of sampling-based algorithms is that the obstacles are not explicitly represented in the state space. Instead, they

avoid the exact definition of it by using a collision checking module. The procedure `isCollisionFree( $\sigma$ )` is necessary to decide if the trajectory, σ , connecting two states, $s \in \mathcal{S}_{free}$ and $s' \in \mathcal{S}_{free}$ lies entirely in $\mathcal{S}_{free}$.

It is clear that collision detection plays an important part in the performance of a sampling-based algorithm as it is called several times over the course of planning. However, the role of a collision detection algorithm over the functioning of the sampling-based algorithms is modular. In other words, it does not rely on a particular collision detection algorithm and, therefore, this subroutine can be substituted by any other algorithm depending on the problem specifications, for example the number of obstacles, whether or not they are dynamic or the complexity of the objects to check for collisions.

Two of the most common approaches that verify if a trajectory is collision-free are: using Bounding Volume Hierarchies (BVH) and using spatial partitioning techniques. The former one adopts a hierarchical representation of the object partitions to be checked for collisions in the object level. They represent each object by means of simpler geometric shapes and determine a hierarchy of collision checks within these primitive shapes. In other words, they represent the object by means of bounding volumes and define a order of collision checking within these hierarchy of volumes. Two of the most commonly used collision checkers of this class are Proximity Query Package (PQP) [35] and Flexible Collision Library (FCL) [70]. On the other hand, the latter one corresponds to representing the space by simpler shapes, for example, using grid-based or voxel-based representations, Octrees etc.

Other than how the collision instance is represented, collision detection methods differ also based on the approach that they take on how they check for collisions. In particular, two different approaches exist: the first one checks multiple discrete instances along the trajectory (e.g. [40]), while the second one uses the *Swept Volume*, i.e., the full volume occupied by an object while moving from one state to another (e.g., [28, 83]). Furthermore, some approaches focus also on modifying the functioning of a sampling-based planner to reduce the number of calls to the collision detection subroutine, for example lazy approaches that delay edge verification [13] or those that apply distance queries to avoid checks for trajectories that are unlikely to be in collision [11].

The numerical examples provided throughout this thesis using the sampling based approach exploits a grid representation of the environment in terms of a binary occupancy grid. The occupancy grid is inflated by the size of the vehicle and the feasibility is checked at discrete intervals. Based

Algorithm 1: An abstract sampling based algorithm

```

1  $V_i \leftarrow s_0, E \leftarrow \emptyset, i \leftarrow 0$ 
2 while  $i \leq N$  do
3    $s_{rand} \leftarrow \text{Sample}(\mathcal{S}_{free})$ 
4    $V_{near} \leftarrow \text{NearNodes}(V, s_{rand})$ 
5    $(s_{best}, e_{best}) \leftarrow \text{FindBestParent}(s_{rand}, V_{near})$ 
6   if  $s_{best} \neq 0$  then
7      $(V_{i+1}, E_{i+1}) \leftarrow \text{UpdateConnections}(s_{best}, e_{best}, V_{near}, s_{rand})$ 
8 return  $\text{BestTraj}(V_{goal})$ 

```

on the observations made by Sánchez and Latombe in [81], the collision check starts from the midpoint of the trajectory as it is argued that it is more likely for a trajectory to have collisions at the midpoint other than at the boundaries.

3.1.5 Abstract Form of Sampling Based Algorithms

All the algorithms presented here keep a graph or a tree structure as a set of nodes and edges, i.e. $G = (V, E)$ or $\mathcal{T} = (V, E)$ respectively. Each node is a state of the dynamic system, i.e. $s \in V$, and they are connected with edges. An edge, $e = \{s, s'\}$, connecting two nodes s and s' , corresponds to the trajectory σ that starts from s and reaches s' , such that $\sigma(0) = s$ and $\sigma(T) = s'$. The notations $\mathcal{T}_i = (V_i, E_i)$ or $G_i = (V_i, E_i)$ with the subscript i are used to describe the set of nodes and the edges in the graph at i -th iteration. We use n to denote the number of nodes in the graph and in order to explicitly refer to the graph with n number of nodes we use the subscript n .

An abstract class of sampling based algorithms is given in Algorithm 1, which iteratively takes the steps defined by the abstract procedures `Sample`, `NearNodes`, `FindBestParent` and `UpdateConnections`, assuming that each procedure has access to the current state of the tree, i.e., the set of nodes and edges. The proposed algorithms and all the reviewed relevant algorithms in this section keeps a tree structure, limiting the connections of a node to a single parent. Therefore, the procedure `Parent(s)` returns a single node in the tree, namely s_{parent} .

The notation $\sigma_{(s \rightarrow s')}$ is used to represent the trajectory starting from s and reaching s' while the function $\text{Cost}(\sigma_{(s \rightarrow s')})$ returns the cost of this trajectory. Every node $s \in V$ is connected to s_0 via a single sequence of intermediate nodes $s_k \in V$, $k = 0, \dots, n - 1$, $n \leq |V|$, where $|V|$ is the cardinality of the tree. One can then associate to this sequence a cumulative

cost $\text{Cost}(\rightarrow s_k)$ that can be computed as

$$\text{Cost}(\rightarrow s_k) = \sum_{i=1}^k \text{Cost}(\sigma_{(s_{k-1} \rightarrow s_k)})$$

and $\sigma_{(s_{k-1} \rightarrow s_k)}$ corresponds to edge $e_k = \{s_{k-1}, s_k\}$. V_{goal} represents the set of all nodes that are in the goal region such that

$$V_{goal} := \{s \in V \mid s \in S_{goal}\}$$

Let $\text{dist}(s, s')$ be a function that returns the distance between two nodes s and s' based on a selected distance metric, the procedure $\text{NearNodes}(V, s_{rand})$ computes neighbor nodes, V_{near} that are in the tree and such that the distance between s_{rand} and any node in V_{near} is less than a threshold, d_{th} , i.e.,

$$V_{near} := \{s \in V \mid \text{dist}(s_{rand}, s) \leq d_{th}\}$$

Note that, for the systems with non-Euclidean dynamics, the distance function is not symmetric, i.e. $\text{dist}(s, s') \neq \text{dist}(s', s)$, which is the case for the systems that are in the scope of this thesis.

Once the nearby nodes are defined, the FindBestParent procedure returns the best node, s_{best} , in V_{near} to connect s_{rand} to the tree and the corresponding edge, $e_{best} = \{s_{best}, s_{rand}\}$ such that $\text{Cost}(\rightarrow s_{rand})$ yields the lowest cost. Consequently, UpdateConnections procedure updates the tree when a new node is added.

Finally, when the algorithm iterations are stopped, the $\text{BestTraj}(V_{goal})$ procedure returns the minimum cost trajectory as a concatenation of trajectories $\{\sigma_1, \sigma_2, \dots, \sigma_{goal}\}$ that interpolates the sequence of nodes, $\{s_0, s_1, \dots, s_{goal}\}$ such that $s_{goal} \in V_{goal}$ is the one with the lowest cumulative cost, $\text{Cost}(\rightarrow s_{goal})$, among all the nodes that reach the goal region.

3.2 Existing Algorithms

This section reviews the existing algorithms that use a sampling-based approach with a particular focus on the randomized sampling-based algorithms, highlighting the optimal sampling based algorithm RRT*. Moreover, due to their relevance, search-based algorithms using a deterministic sampling strategy on the discretized space are also reviewed.

3.2.1 Randomized Sampling Based Algorithms

Considering the literature on planning techniques, sampling based algorithms are still the most popular approaches, representing the practical way

to tackle the problem. Their basic idea is to sample nodes in the continuous state space and connect these nodes with trajectories in the collision-free space, building a roadmap in the form of a graph or a tree. These algorithms overcome the complexity issue by relaxing the *completeness* requirement with the *probabilistic completeness*, stated in the following definition.

Definition 3.2 (Probabilistic completeness). *An algorithm is probabilistically complete, if, the probability that the graph computed by the algorithm contains a node in the goal region approaches 1 as the number of iterations goes to infinity, i.e.,*

$$\liminf_{i \rightarrow \infty} \mathbb{P}(\exists s_{goal} \in V_i \cap S_{goal}) = 1$$

PRM introduced by Kavraki et. al. [50], and RRT [53] introduced by Kuffner et. al., were the first popular sampling-based planners. Both algorithms are *probabilistically complete* with proofs that state finding a solution, if one exists, approaches 1 exponentially fast with the number of iterations.

PRM first creates a graph in the free space by randomly sampling states and using a local planner to connect a randomly sampled node to the already existing ones in the graph. This graph is then traversed to answer multiple queries. However, for kinodynamic planning connecting two nodes entails solving a TPBVP. This means that PRM and its optimal counterpart Optimal Probabilistic RoadMap Planner (PRM*) [47] are limited to symmetric costs and to those systems for which the cost associated to a TPBVP is conserved when the boundary pairs are swapped. Note that, nonholonomic systems do not belong to this class.

RRT, on the other hand, incrementally builds a tree for each single query, returning the solution as soon as it finds one. In particular in [58], the authors have extended the algorithm to consider kinodynamic planning. The outline of the algorithm is given in Algorithm 2, which shares the same structure also with the kinematic counterpart. Making reference to the abstract class given in Algorithm 1, the `NearNodes` procedure is replaced with `NearestNeighbor` which returns the closest node in the tree, $s_{nearest}$, other than a set of states. The Lines 4-6 correspond to the `Extend` procedure in RRT terminology [53, 58] and is the procedure of finding the best parent, $s_{nearest}$ to extend the tree from. Note that the algorithm does not employ an exact steering algorithm. Hence, instead of expanding the tree to include s_{rand} , a new node s_{new} is created by the steering procedure. Moreover, Line 7 can be interpreted as the `UpdateConnections`, where the tree is simply updated with the inclusion of a new edge and a node. Finally, once

Algorithm 2: RRT

```

1  $V \leftarrow s_0, E \leftarrow \emptyset, i \leftarrow 0, V_{goal} \leftarrow \emptyset$ 
2 while  $i \leq N$  do
3    $s_{rand} \leftarrow \text{Sample}(S_{free})$ 
4    $s_{nearest} \leftarrow \text{NearestNeighbor}(V, s_{rand})$ 
5    $s_{new}, \sigma_{new} \leftarrow \text{Steer}(s_{nearest}, s_{rand})$ 
6   if  $\text{isCollisionFree}(\sigma_{new})$  then
7      $V \leftarrow V \cup s_{new}, E \leftarrow E \cup \{s_{nearest}, s_{new}\}$ 
8     if  $V_{goal} \neq \emptyset$  then
9       return  $\text{BestTraj}(V_{goal})$ 

```

the tree is expanded to include a node in the goal region, i.e., $V_{goal} \neq \emptyset$, $\text{BestTraj}(V_{goal})$ returns the trajectory that reaches the goal region.

RRT provides fast, on-line implementations and has been effectively used for various different problems; however, the resulting trajectories are generally jerky and not desirable. Some variants are proposed (see e.g., [90]) to improve the solution introducing heuristics, but it is proven that the probability of finding the optimal solution is zero [47].

3.2.2 Towards Optimality: RRT* and Variants

With the introduction of RRT* and PRM* [47], an important progress has been made towards the improvement of the solution quality that guarantees *asymptotical optimality* using the random graph theory, such that the probability of finding the optimal solution is 1 as the number of iterations goes to infinity.

Definition 3.3 (Asymptotic optimality). *An algorithm is asymptotically optimal if the following is true for the cost of the best cost trajectory that reaches a state in the goal region, i.e. c_i ,*

$$\mathbb{P}(\{\limsup_{i \rightarrow \infty} c_i = c^*\}) = 1$$

where c^* is the cost of the optimal trajectory for a given motion planning problem.

The idea behind optimality is to check connections within a neighborhood, ensuring that the graph is connected with the optimal edges at all times. Karaman and Frazzoli point out that using a neighborhood whose volume is a logarithmic function of the graph cardinality creates the perfect balance between graph connectivity, hence the asymptotic optimality, and the computational complexity [45].

Algorithm 3: FindBestParent procedure for RRT*

```

1 Input:  $V_{near}, s_{rand}$ 
2  $c_{best} \leftarrow \infty$ 
3 for  $\forall s_{near} \in V_{near}$  do
4    $\sigma_{near} = \text{Steer}(s_{near}, s_{rand})$ 
5   if  $\text{isCollisionFree}(s_{near}, s_{rand})$  then
6     if  $C(\rightarrow s_{near}) + \text{Cost}(\sigma_{near}) < c_{best}$  then
7        $c_{best} = C(\rightarrow s_{near}) + \text{Cost}(\sigma_{near})$ 
8        $s_{best} = s_{near}$ 
9        $e_{best} = \{s_{rand}, s_{best}\}$ 
10 return  $s_{best}, e_{best}$ 

```

The RRT* Algorithm

Similar to RRT, RRT* also keeps a tree data structure; however, for every time a random state is sampled, a set of nodes in the tree is searched to ensure that the tree is connected with the best edges at all times. In this way, RRT* is more similar to PRM than it is to RRT since it reasons over a graph. Algorithms 3 and 4 show the adapted procedures of FindBestParent and UpdateConnections for RRT*, both of which consider an *exact optimal steering* action where the resulting trajectory, σ_{near} , of the $\text{Steer}(s_{rand}, s_{near})$ or vice versa, corresponds to solving the TPBVP by minimizing an objective function. Once a node, s_{rand} , is sampled and a neighborhood is searched to ensure that s_{rand} is connected to the tree with the lowest cost possible. Once a node is added to the tree using UpdateConnections procedure (Line 2), the tree is *rewired* to check the local neighborhood if an already existing node in the tree can be reached from s_{rand} with a lower cost. If a node in $s_{near} \in V_{near}$ is reached with a lower cost from s_{rand} compared to its previous parent, the tree is rewired; i.e., the edge connecting s_{near} with its parent is replaced with the edge $\{s_{rand}, s_{near}\}$ (Lines 3-9).

To ensure asymptotic optimality without sacrificing computational performance, the samples are connected as selected within hyperspheres which have a volume parametrized by the actual tree cardinality, n . Therefore, for each sampled node s_{rand} , the set of near nodes is defined as the set of nodes within a volume centered in s_{rand} . For holonomic systems, the state space is Euclidean and the distance between two states can be represented by Euclidean distances, and then this volume is defined by using a d -dimensional ball of radius

$$\gamma_{ball} = \gamma_{RRT^*}(\log(n)/n)^{1/d}$$

Algorithm 4: UpdateConnections procedure for RRT*

```

1 Input:  $V_{near}, s_{rand}, e_{best}, s_{best}$ 
2  $V \leftarrow V \cup s_{rand}, E \leftarrow E \cup \{s_{best}, s_{rand}\}$ 
3 for  $\forall \{s_{near} \in S_{near}\} \setminus s_{best}$  do
4    $\sigma_{near} = \text{Steer}(s_{rand}, s_{near})$ 
5   if  $\text{isCollisionFree}(s_{rand}, s_{near})$  then
6     if  $C(\rightarrow s_{rand}) + C(\sigma_{near}) < C(\rightarrow s_{near})$  then
7        $s_{parent} \leftarrow \text{Parent}(s_{near})$ 
8        $E \leftarrow E \setminus \{s_{parent}, s_{near}\}$ 
9        $E \leftarrow E \cup \{s_{rand}, s_{near}\}$ 
10 return  $V, E$ 

```

where γ_{RRT^*} is a suitable constant selected as

$$\gamma_{RRT^*} > 2(1 + 1/d)^{1/d} (\mu((Q_{free})/\zeta_d)^{1/d})$$

with $\mu(Q_{free})$ and ζ_d denoting the volume of the free configuration space and of the unit ball respectively, in a d -dimensional Euclidean space [47].

For dynamic systems that have a non-Euclidean state spaces, the distance between two states is represented by the optimal cost of the trajectory that connects them [46, 48], and the near nodes are selected as the nodes that can be reached from s_{rand} or that can reach s_{rand} as follows

$$V_{near} = \{s \in S : \text{Cost}(\sigma_{s_{rand}, s}) \leq d_{th}(n) \vee \text{Cost}(\sigma_{s, s_{rand}}) \leq d_{th}(n)\} \quad (3.1)$$

where $d_{th}(n)$ is a cost threshold which is a logarithmic function of the actual tree cardinality, n .

RRT* Variants

Two domain dependent components are needed in order to build the tree: the first one is the definition of a distance metric and the second is a steering function. In [20], the authors highlight the importance of a correct distance metric for the convergence of RRT based algorithms, arguing that naive implementations such as using Euclidean distances may not describe the true dynamics. In fact, this is often the case for kinodynamic motion planning instances. For a given efficient steering function, in [48] Karaman and Frazzoli propose a conservative estimate of the neighbors based on differential geometry that helps defining a Euclidean metric accurately approximating the proper dynamics of the system, hence improving computational effectiveness. Their approach is based on the control-affine formalization of the

system dynamics and approximating reachable set of states within a local neighborhood by means of approximation using an axis aligned box. However, this improvement also assumes that an efficient steering function is available.

Other than effective computation of neighbors, the main difficulty in using RRT based approaches for optimal kinodynamic planning lies within the definition of a steering function which should output the optimal trajectory connecting the two states. For holonomic robots, this connection can be represented by straight lines. However, for many other dynamic systems, this corresponds to solving a TPBVP. The main benefit of the sampling-based planners is the assumption that solving a single problem is much easier than solving the global planning problem; however, computing optimal trajectories for dynamic systems is generally not a trivial task. There are some dynamic systems which have available solutions that can efficiently steer the system from one state to another such as Dubins vehicle [25] or 2D Double Integrator [30]. Therefore, some works focus on such specific systems (e.g., [46]).

To deal with the computational complexity originated from the steering procedure, some effort has been made to develop more effective steering functions for different types of dynamic systems. Webb and Berg [91] have obtained the closed-form analytical solution of the optimization problem for a fixed final state free final time for systems with linear dynamics and extended it to non-linear dynamics using their first-order Taylor approximations. Other works [34, 75] focus on approximating the solution of the TPBVP for systems with linearizable dynamics by locally linearizing the system and applying linear quadratic regulation (LQR). However, their approach does not have a guarantee on reaching the exact final state, hence the optimality is not proven to be guaranteed. Furthermore, these approaches are system dependent and can apply only to a limited variety of dynamic systems.

Some recent attempts have been made towards optimality without formulating and solving a TPBVP, as well. For example, algorithms like Stable Sparse RRT* (SST*) [61] have proved asymptotic optimality given access only to a forward propagation model. The idea is to iteratively sample a set of controls and final times instead of explicitly solving the TPBVP. Similarly, a variant of RRT* [42] uses a shooting method without an exact steering function to improve the solution by pruning branches from the tree. If a sampled node has a lower cost compared to another one that is close by and that shares the same parent, the preexisting node is pruned from the tree and its branches are connected to the newly sampled node, or they are

pruned completely if they are not collision-free. This approach generates feasible but inherently suboptimal solutions.

Other than having arguable guarantee on optimality, not having an exact steering action is sometimes undesirable for some types of dynamic systems. Humanoid robots set an example for that since it is crucial to have control over the intermediate nodes in order to satisfy some geometric constraints (eg. keeping one foot on the ground). A similar approach that embeds the idea of motion primitives into the sampling based planning framework is recently proposed by [84] for combining the motion planning for locomotion and the upper body (eg. pick up an object) by combining different sets of motion primitives. However, their approach does not avoid solving for the TPBVPs.

Nevertheless, solving the TPBVP for an arbitrary nonlinear system remains challenging and typically calls for numerical solvers. The algorithms that account for a nonlinear optimization tool, like for example ACADO toolkit [38], GPOPS-II [72], etc., commonly use *Sequential Quadratic Programming* (SQP) [12] for solving the TPBVPs numerically, and embed it as a subroutine in the sampling based planning framework such as in RRT* [88] or in Batch-Informed-Trees star (BIT*) [31]. Although allowing to address a large variety of problems, embedding a numerical solver within the sampling-based framework is far from being applicable to online application where the plans need to be updated relatively fast.

The main contribution of this thesis to the literature of sampling-based kinodynamic planners is the the algorithm called RRT*_MP, which extends RRT* by introducing a database of pre-computed motion primitives in order to avoid the online solution of a constrained TPBVP for the edge computation. The database is composed of a set of trajectories, each one connecting an initial state to a final one in a suitably defined grid. By sampling in the gridded state space, the implementation of the steering function adopted for growing and rewiring the RRT* tree reduces to the search of a motion primitive in a pre-computed Look Up Table (LUT).

The main difference between the proposed approach and the existing sampling-based algorithms is that it is applicable to any dynamical system described by differential equations and subject to analytical constraints, for which edge design can be formulated as a TPBVP. However, unlike the algorithms that can handle a large variety of dynamics, solving for a steering action is carried to the offline phase removing the overwhelming procedure of calling for a nonlinear solver over the course of planning. Notably, when a model of the vehicle is not available, the database can be derived directly from experimental trajectories.

3.2.3 Search Based Algorithms

A different class of algorithms, aiming at optimality, is based on graph search and adopt a gridding approach. The main idea is to discretize the state space, building a grid, and compute a graph. The motion planning problem is then recast into finding the best sequence of motions by traversing this graph with an optimal search algorithm like A* [73]. This graph is often represented by a state lattice, a set of states distributed in a regular pattern, where the connections between states are provided by feasible/optimal trajectories [77]. [62] improved the idea of state lattice by using a multi-resolution lattice such that the portion of the graph that is close to either the initial or the goal state has a higher resolution than the other parts. These approaches have been successfully applied to several robotic systems and found to be effective for dynamic environments [23, 62]. However, these algorithms are *resolution optimal*, such that the optimality is guaranteed up to the grid resolution. Furthermore, their computational effectiveness is highly related to the resolution: the finer is the grid, the higher the branching factor, and thus the computational time and the required memory to execute a graph traversal algorithm.

The main difference of the proposed approach, with respect to existing search-based algorithms, is that it leverages on a dynamic tree whose size depends only on the number of samples, but not on the number of motion primitives that affects the accuracy in the approximation of the robot kinematic and dynamic characteristics. As a consequence, memory consumption to store the tree and computation time to determine a solution on a given tree can be bounded selecting an appropriate maximum number of samples, without introducing undesired constraints in the robot action space. Search based approaches, instead, have to strongly limit the action space, keeping the number of motion primitives low, as the branching factor of the graph, i.e., the number of edges generated expanding each node, depends on the size of the database.

3.3 Sampling Based Planning with Motion Primitives

The approach here proposed is based on previous works on search-based [62, 77], and sampling-based [46, 47] methods, and combines them in a novel way. In particular, it relies on a uniform discretization of the state space, and on the computation of a finite set of motion primitives by solving a constrained optimization problem with boundary conditions on the grid points of a smaller (uniform) grid. The motion primitives are then

embedded in the RRT* algorithm, where they are used to compute the distance between two states and to connect the randomly generated nodes to the tree, thus eliminating the need of solving online challenging and time consuming TPBVPs. Therefore, all the computations for respective operations are reduced into a search in the precomputed database that can be easily represented as a Look Up Table (LUT).

In the context of motion planning, system (2.1) represents the robot equations of motion and, consequently, its state vector s includes the robot position with respect to a given absolute reference frame. Therefore, the state is represented as tuple $s \in \mathcal{S}$, $s = [\pi, \dots]^T$ such that π denotes the position components of the vehicle. The following property of a dynamic system is essential for the rest of the algorithm that allows the use of a database of precomputed trajectories.

Assumption 3.1 (translation invariance property). *The state s contains the position of the vehicle in some coordinate system. If the system dynamics (2.1) and the state initialization are written in a new coordinate system obtained via translation of the current one, the state evolution obtained by applying some input $u(\cdot)$ is the translated version of the one in the original coordinates. Also, the satisfaction of algebraic path constraints and the value taken by the cost criterion are not affected by the translation of the coordinate system.*

It is worth mentioning that the above assumption is common for both mobile robots and autonomous vehicles and it has been used by many works in the literature of motion planning (see e.g., [23, 62, 77]). The rest of this section will explain the idea of motion primitives and explicitly define the sampling-based motion planning algorithm that makes use of a database of precomputed trajectories.

3.3.1 Motion Primitives

As the name suggests motion primitives refer to a set of precomputed motions for a given dynamic system. These primitives are solutions to sub-problems that can be concatenated in order to obtain a final trajectory that solves the motion planning problem given in Section 2.3.

The database of motion primitives is built by gridding the continuous state space in order to obtain a finite set of boundary conditions (initial and final states), and by solving offline a constrained boundary value optimization problem for each pair. The resulting set of optimal trajectories is then used repeatedly online, implementing a procedure that, when an edge con-

necting two nodes is requested by the planner, simply picks up from the database a suitable trajectory.

Given a state tuple $\mathbf{s} \in \mathcal{S}$, $\mathbf{s} = [\boldsymbol{\pi}^T, \dots]^T$ where $\boldsymbol{\pi}$ represents the vehicle position, motion primitives are computed for each pair of initial and final states $(\tilde{s}_0, \tilde{s}_f)$ with $\tilde{s}_0 = [\boldsymbol{\pi}^T, \dots]^T$. Given a boundary value pair $(\tilde{s}_0, \tilde{s}_f)$, a motion primitive is computed by solving the following optimization problem, for the optimal final time and control input

$$\min_{u(\cdot), \tau} J = \int_0^\tau g(\mathbf{s}(t), u(t)) dt \quad (3.2)$$

subject to

$$\dot{\mathbf{s}}(t) = f(\mathbf{s}(t), u(t)) \quad (3.3)$$

$$u(t) \in \Omega \text{ for } t \in [0, \tau] \quad (3.4)$$

$$\mathbf{s}(t) \in \mathcal{S} \quad (3.5)$$

$$\mathbf{s}(0) = \mathbf{s}_0, \mathbf{s}(\tau) = \mathbf{s}_f \quad (3.6)$$

$$\phi_l \leq \phi(\mathbf{s}(t), u(t), t) \leq \phi_u \quad (3.7)$$

where $u(t)$ is the vector of control inputs where Ω denotes the admissible controls and $\mathbf{s}(t)$ is the state vector.

Finally the database, $\mathbb{Z}$, is generated by considering distinct pairs of initial and final states $(\tilde{s}_0^i, \tilde{s}_f^i)$, $i = 1, \dots, N_{MP}$, and computing the corresponding motion primitives $\mathbb{Z} = \{\mathbf{z}_i \in \mathcal{S} \times \Omega \times \mathbb{R}_{\geq 0}, \mathbf{z}_i = (\sigma_i^*(t), u_i^*(t), \tau_i^*)\}$ where $\sigma_i^*(t), u_i^*(t)$ for $t \in [0, \tau_i^*]$ represent optimal trajectories, optimal control inputs and optimal final time respectively. Similarly, $\mathbb{C} = \{J_i^*\}$ denotes the set of costs of these optimal trajectories.

Note that, thanks to the translation invariance property introduced in Assumption 3.1, the size of the database can be kept small while covering a wide range of the space where the robot is moving. Indeed, one can, without loss of generality, set the initial position $\tilde{\boldsymbol{\pi}}_0$ to $\tilde{\boldsymbol{\pi}}_0 = 0$ when building the database, and recover the optimal trajectory for an arbitrary initial position $\boldsymbol{\pi}_0$ by simply centering the coordinate system in $\boldsymbol{\pi}_0$.

Example 3.1. Consider, as an example, a planning problem for a unicycle robot characterised by a 3D search space (x, y, θ) , including the position (x, y) and the orientation θ , and by a 2D actuation space (v, ω) , constituted by the linear velocity v and the angular velocity ω .

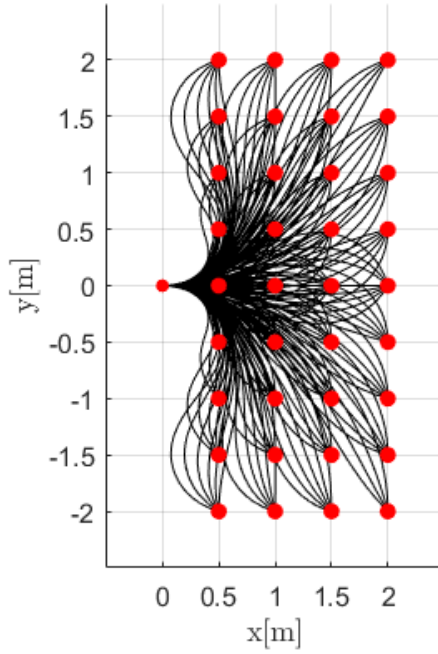


Figure 3.1: A subset of the motion primitives computed for a 3D search space (x, y, θ) . Red dots correspond to the Cartesian coordinates (x, y) and black lines represent the resulting trajectories for different final orientations (θ) .

Motion primitives are computed solving the following TPBVP

$$\begin{aligned}
 & \underset{v(\cdot), \omega(\cdot), \tau}{\text{minimize}} && \int_0^{\tau} (1 + 0.5v(t)^2 + 0.5\omega(t)^2) dt \\
 & \text{subject to} && \dot{x}(t) = v(t) \cos(\theta(t)) \\
 & && \dot{y}(t) = v(t) \sin(\theta(t)) \\
 & && \dot{\theta}(t) = \omega(t) \\
 & && v(t) \in [0, 2], t \in [0, \tau] \\
 & && \omega(t) \in [0, 2], t \in [0, \tau] \\
 & && x(0) = \bar{x}_0, y(0) = \bar{y}_0, \theta(0) = \bar{\theta}_0 \\
 & && x(\tau) = \bar{x}_f, y(\tau) = \bar{y}_f, \theta(\tau) = \bar{\theta}_f
 \end{aligned}$$

for different initial and final poses.

Figure 3.1 shows a subset of these motion primitives, characterized by

trajectories starting from $\bar{x}_0 = \bar{y}_0 = \bar{\theta}_0 = 0$.

3.3.2 Search Space Design

In order to take advantage of the precomputed database of motion primitives in sampling-based planning, the search space of the planner has to be defined appropriately so that every time the planner needs to connect two nodes, the corresponding optimal trajectory can be found in the database. To guarantee that this is indeed the case, the search space of the planner is uniformly gridded as the region where motion primitives are built. The translation invariance property¹ (Assumption 3.1) can then be exploited, as any optimal trajectory which connects a pair of initial and final states (in the discretized search space) can be computed by first shifting the initial and final states so that the initial robot position corresponds to the zero position, then picking a suitable motion primitive in the database, and finally shifting the motion primitive so as to get back to the original reference coordinate system. Translation invariance, jointly with uniform gridding, allow to cover the entire (discretized) search space with a reduced number of motion primitives.

Let $\mathcal{S}^\Delta \subset \mathcal{S}$ define the discretized state space and let $\tilde{\mathcal{S}}^\Delta \subset \mathcal{S}$ be the discrete state space used to construct the database. Consecutively, let $\text{BoundingBox}(\pi_0)$ denote the bounding box that includes all the boundary value pairs used for the computation of the motion primitives, with its center having the same position as the initial states used in the database generation, π_0 . For any $s' \in \mathcal{S}^\Delta$ a set of states can be defined as the ones contained in the bounding box translated to π' , i.e., $\mathcal{S}' = \mathcal{S}^\Delta \cap \text{BoundingBox}(\pi')$. Denote T as the function that translates π' to π_0 , i.e., $T : \pi' \rightarrow \pi_0$. In order to use a particular database of motion primitives it is necessary that applied a translation using T , any point in $\mathcal{S}'$ should be strictly mapped to a point in the database. This means that the same database can be queried for different discrete search spaces, as long as the resulting translation is a subset of $\tilde{\mathcal{S}}^\Delta$.

Determining optimal state space discretization depends on the specific application, and is out of the scope of this work. We shall assume here that the state space grid should include the initial state s_0 , at least a grid point in the goal region, and few points in the free space $\mathcal{S}_{free}$. Moreover, in order to find a solution that reaches the goal region, the algorithm should be able to search within all homotopy classes. In other words, one should be able to represent in the grid space all sets of trajectories in the continuous state

¹This approach can be easily extended in case stronger invariance properties hold.

Algorithm 5: RRT*__{MotionPrimitives}

```

1  $V_i \leftarrow \{s_0\}, E_i \leftarrow \emptyset, i \leftarrow 0$ 
2 while  $i \leq N$  do
3    $s_{\text{rand}} \leftarrow \text{Sample}(S_{\text{free}}^\Delta)$ 
4    $V_{\text{near}} \leftarrow \text{NearNodesToCome}(s_{\text{rand}})$ 
5    $s_{\text{best}}, e_{\text{best}} \leftarrow \text{FindBestParent}(V_{\text{near}}, s_{\text{rand}})$ 
6   if  $s_{\text{rand}} \in V$  then
7      $(V_{i+1}, E_{i+1}) \leftarrow \text{UpdateExisting}(s_{\text{best}}, e_{\text{best}}, s_{\text{rand}})$ 
8   else if  $s_{\text{rand}} \notin V \wedge s_{\text{best}} \neq \emptyset$  then
9      $(V_{i+1}, E_{i+1}) \leftarrow \text{UpdateNew}(s_{\text{best}}, e_{\text{best}}, s_{\text{rand}})$ 
10 return  $\text{BestTraj}(V_{\text{goal}})$ 

```

space that can be obtained applying a smooth transformation and lead to the goal region. Missing a homotopy class could highly deteriorate performance in terms of achieved cost. In Section 3.6 some numerical examples are provided, in which different grids are used for solving the same case study and results are compared.

Recall that $\kappa(s) = q$ is a function that maps a state to the corresponding configuration. Similar to the continuous case, the *discrete free state space* can be defined as

$$\mathcal{S}_{\text{free}}^\Delta := \{s \in \mathcal{S}^\Delta \mid \kappa(s) \in \mathcal{Q}_{\text{free}} \wedge \mathbf{h}(s) \leq 0\}$$

where $\mathcal{Q}_{\text{free}}$ is the free configuration space and $\mathbf{h}(s) \leq 0$ denotes a set of inequalities representing the bounds on the state variables. The discrete goal region

$$\mathcal{S}_{\text{goal}}^\Delta := \{s \in \mathcal{S}^\Delta \mid s \in \mathcal{S}_{\text{goal}}\}$$

is also introduced as the set of states that are contained in the goal region.

3.3.3 Motion Planning

This section introduces RRT*__{MotionPrimitives} (RRT*__{MP}) (Algorithm 5), the proposed variant of RRT* integrating the database of motion primitives for the computation of a collision-free optimal trajectory.

As in RRT*, RRT*__{MP} follows the abstract framework given in Algorithm 1 and it is based on the construction of a random tree $\mathcal{T} = (V, E)$, whose *nodes* $s \in V$ represent a state of the system sampled from the discrete state space, $s \in \mathcal{S}^\Delta$, and *edges* $e \in E$ are optimal trajectories, each one connecting a pair of origin and destination nodes and solving the TPBVP in (3.7).

Algorithm 6: NearNodesToGo

```

1 input:  $s_{rand}$ 
2  $V_{near} \leftarrow \emptyset$ 
3  $\pi_{rand} \leftarrow \text{GetPosition}(s_{rand})$ 
4 for  $\forall s \in V$  do
5   if  $s \in \text{BoundingBox}(\pi_{rand})$  then
6      $\sigma_{(s_{rand} \rightarrow s)} \leftarrow \text{FindTrajectory}(s_{rand}, s)$ 
7     if  $\text{Cost}(\sigma_{(s_{rand} \rightarrow s)}) \leq d_{th}(n)$  then
8        $V_{near} \leftarrow V_{near} \cup \{s\}$ 
9 return  $V_{near}$ 

```

Tree growing is based on four main steps – random sampling, finding near nodes, extending the tree, and updating connections – that are described in the following.

Random sampling

A random state s_{rand} is sampled from the free state space $\mathcal{S}_{free}$ according to a uniform distribution, using the function $\text{Sample}(\mathcal{S}_{free}^\Delta)$. Unlike the original RRT* algorithm, however, the node is not sampled from the continuous state space, but from its discretization according to a uniform grid. For this reason, there is also a non zero probability that the same state s_{rand} is sampled again in the next iterations of the algorithm.

Near nodes (Algorithms 6-7)

In RRT* framework, hence in RRT*_MP, a random state s_{rand} can be connected only to a node that is within the set of its near nodes. Furthermore, once connected to the tree it can contribute to update the tree only for the nodes that are within this local neighborhood.

Defining a local neighborhood requires a distance metric to be computed, taking advantage of the motion primitives, the distance between two states is easily defined as the cost of the optimal trajectory connecting them, i.e. $\text{dist}(s, s') = \text{Cost}(\sigma_{(s \rightarrow s')})$. In other words, the database of motion primitives allows the usage of the real *cost-to-go* as the distance metric, which is the ideal case.

Depending on the queried connection – extending the tree towards the sample or rewiring – the definition of this neighborhood changes, since computing the distance between two states, i.e., the cost of the optimal trajectory, implies solving for a TPBVP and it is often not symmetric. In

Algorithm 7: NearNodesToCome

```

1 input:  $s_{rand}$ 
2  $V_{near} \leftarrow \emptyset$ 
3  $\pi_{rand} \leftarrow \text{GetPosition}(s_{rand})$ 
4 for  $\forall s \in V$  do
5   if  $s \in \text{BoundingBox}(\pi_{rand})$  then
6      $\sigma_{(s \rightarrow s_{rand})} \leftarrow \text{FindTrajectory}(s, s_{rand})$ 
7     if  $\text{Cost}(\sigma_{(s \rightarrow s_{rand})}) \leq d_{th}(n)$  then
8        $V_{near} \leftarrow V_{near} \cup \{s\}$ 
9 return  $V_{near}$ 

```

particular, two different procedures are defined, named NearNodesToGo and NearNodesToCome (Algorithms 6 - 7), in order to compute a local neighborhood, V_{near} , which has a different meaning depending on its purpose within the functioning of the algorithm. For a sampled state s_{rand} , NearNodesToCome(s_{rand}) returns the nodes in the tree that are in the neighborhood when a backward motion is considered, i.e. a trajectory reaching s_{rand} . This case is used when a new node, s_{rand} has to be added to the tree or when an existing one is sampled again. On the other hand, NearNodesToGo(s_{rand}) returns the nodes that are in the neighborhood when a forward motion is considered, i.e. a trajectory initiating from s_{rand} . In that case, V_{near} is used for rewiring the tree.

Given s_{rand} the set of reachable nodes in the neighborhood of s_{rand} is defined as the set of states that can be reached from s_{rand} or that can reach s_{rand} with a cost lower than a threshold as follows

$$V_{reachToGo} = \{s \in V : \text{dist}(s_{rand}, s) \leq d_{th}(n)\} \quad (3.8)$$

where $d_{th}(n)$ is a distance threshold that decreases logarithmically over the iterations of the algorithm such that $d_{th}(n) = \gamma(\log n/n)$ and γ is a suitable constant, determining the size of the neighborhood, independent of the tree cardinality. It should be selected such that a Euclidean ball of volume $\gamma_l^d(\log n/n)$ is contained within $V_{reachToGo}$, where γ_l is a constant, and it should have non-empty interior when the gridding is concerned. This ensures that for a selected cost threshold there are reachable states without leaving a local neighborhood.

In RRT*_{MP}, the set $V_{reachToGo}$ has to be further constrained to ensure that there is a precomputed trajectory in the database for each connection

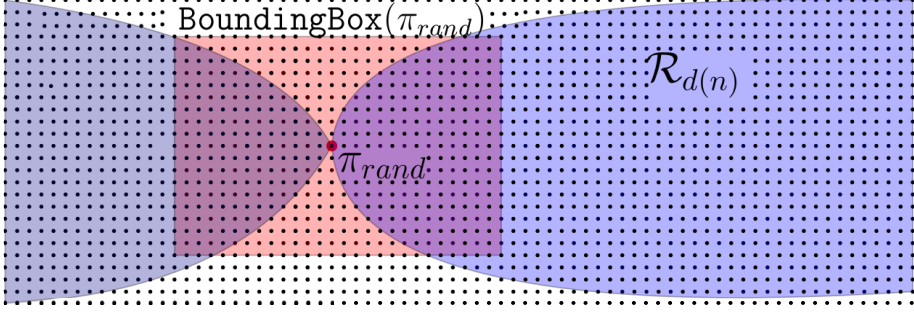


Figure 3.2: For a random state whose position corresponds to π_{rand} , the $\text{BoundingBox}(\pi_{rand})$ is defined. $\mathcal{R}_{d(n)}$ represents the states that can reach or can be reached from s_{rand} below a threshold. Therefore any vertex that falls into the overlapping area (purple) is taken as a member of $V_{nearToGo}$ or $V_{nearToCome}$ depending on if the set comprises the nodes that can reach or can be reached from s_{rand} .

in the set of near nodes, that is thus redefined as follows

$$V_{nearToGo} = V_{reachToGo} \cap \text{BoundingBox}(\pi_{rand})$$

where $\text{BoundingBox}(\pi_{rand})$ is the box of grid points in the state space S adopted for the database construction, with the origin of the reference coordinate system shifted from $\pi_0 = 0$ to π_{rand} which is the vehicle position associated to the state s_{rand} and obtained using `GetPosition` (Figure 3.2).

Similarly, the nodes that can reach s_{rand} below a threshold are computed as

$$V_{reachToCome} = \{s \in V : \text{dist}(s, s_{rand}) \leq d_{th}(n)\} \quad (3.9)$$

and the set of nearby nodes when a motion towards s_{rand} is concerned is computed as:

$$V_{nearToCome} = V_{reachToCome} \cap \text{BoundingBox}(\pi_{rand})$$

Find best parent (Algorithm 8)

In order to keep the tree connected with the best possible edges, it is necessary to find the best parent, s_{best} , to connect the sampled node, s_{rand} . The parent is computed using the `FindBestParent` procedure and the minimum cost collision-free trajectory is selected to connect s_{rand} to the tree such that $\text{Cost}(\rightarrow s_{rand})$ results in the lowest cumulative cost. In order to find s_{best} , a set of neighboring nodes, V_{near} , computed using `NearNodesToCome` procedure, are queried to find the resulting trajectory $\sigma(s_{best} \rightarrow s_{rand})$ and its cost such that combined with the cumulative cost of

Algorithm 8: FindBestParent for RRT*_MotionPrimitives

```

1 Input:  $s_{rand}, V_{near}$ 
2  $c_{best} \leftarrow \infty$ 
3 for  $s_{near} \in V_{near}$  do
4    $\sigma_{near}, c_{near} \leftarrow \text{FindTrajectory}(s_{near}, s_{rand})$ 
5   if  $\text{Cost}(\rightarrow s_{near}) + c_{near} < c_{best}$  then
6     if  $\text{isCollisionFree}(\sigma_{near})$  then
7        $s_{best} \leftarrow s_{near}, c_{best} \leftarrow \{s_{near}, s_{rand}\}$ 
8        $c_{best} \leftarrow \text{Cost}(\rightarrow s_{near}) + c_{near}$ 
9 return  $s_{best}, c_{best}$ 

```

reaching s_{best} , would allow s_{rand} to be connected to the tree with the lowest cost possible.

More precisely, s_{best} is found as follows

$$s_{best} = \underset{s \in V_{feasible}}{\operatorname{argmin}} (\text{Cost}(\rightarrow s) + \text{Cost}(\sigma_{s, s_{rand}}))$$

where $V_{feasible} \subseteq V_{near}$ is the set of nodes that belong to V_{near} and such that the corresponding trajectories are collision free, i.e.,

$$V_{feasible} = \{s \in V_{near} | \text{isCollisionFree}(\sigma_{(s \rightarrow s_{rand})})\}$$

The selection of a trajectory from the database connecting s to s_{rand} , is a peculiarity of RRT*_MP, and is performed by the FindTrajectory function as explained in the following.

Find trajectory

Given a set of boundary conditions, the database of motion primitives is queried using the FindTrajectory procedure. For each query, a series of geometric transformations is often required (Figure 3.3, see Section 3.3.1 for further details). In particular, for each edge that connects two nodes s_0, s_f , the corresponding trajectory and the cost are extracted from the database, $\mathbb{Z}$, as follows:

1. The pair of initial and final states (s_0, s_f) (Figure 3.3a) are normalized by a translation, obtaining the normalized pair of states $(\tilde{s}_0, \tilde{s}_f)$, such that the resulting $\tilde{s}_0$ has the position $\tilde{\pi}_0$ corresponding to the null vector (Figure 3.3b).
2. A query is executed to compute an index, index , corresponding to the normalized boundary pair and on the database to look for the tuple

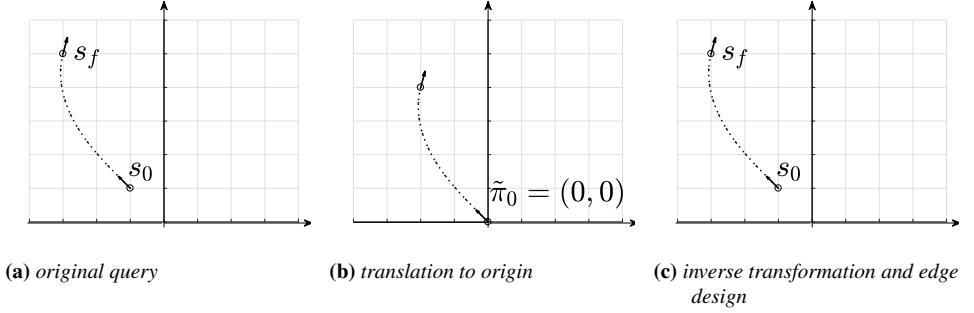


Figure 3.3: Steps involved in the coordinate transformation during the FindTrajectory procedure.

$z_{\text{index}} \in \mathbb{Z}$ for the trajectory σ_{index} and the cost $c_{\text{index}} \in \mathbb{C}$ such that $c_{\text{index}} = \text{Cost}(\sigma_{\text{index}})$.

3. The inverse of the previous translation is applied in order to recover the trajectory connecting the actual pair of boundary values (s_0, s_f) . Hence, the edge $e = \sigma_{(s_0 \rightarrow s_f)}$ is determined (Figure 3.3c).

Update connections (lines 7-9 in Algorithm 5)

Since $\text{Sample}(S_{\text{free}}^\Delta)$ returns samples from a set of finite number of states and sampled states are not discarded, there is a non-zero probability that a state which is already a node in the tree can be sampled again. Therefore, Algorithm 1 should be modified to adapt updating the connections in the tree. Algorithm 5 shows the adapted version of the abstract class. In particular, the UpdateConnections procedure is changed and divided into two distinct procedures; while UpdateNew deals with the scenario when a state is sampled for the first time, UpdateExisting updates the tree in the case of re-sampling a state that is already a member of the tree. The functionality of both procedures will be explained in detail further in this chapter.

Two versions of the algorithm are proposed depending on how the information propagates in the tree, that both share the same steps and differ in UpdateConnections procedures. The following two sections explain two different modes of RRT*_{MP} in terms of updating the tree, in particular depending on how a change in the tree is communicated through the neighborhood. One of the main procedures of RRT* that leads to optimality is the rewiring, which ensures that each node has always the best parent within a local neighborhood. Having a finite sampling set allows to

Algorithm 9: UpdateExisting for RRT*_MotionPrimitives without rewiring

```

1 Input:  $s_{rand}, s_{best}, e_{best}$ 
2  $s_{parent} \leftarrow \text{Parent}(s_{rand})$ 
3 if  $s_{parent} \neq s_{best}$  then
4    $e_{prev} \leftarrow \{s_{parent}, s_{rand}\}$ 
5    $E \leftarrow \{E \setminus e_{prev}\} \cup \{e_{best}\}$ 
6 return  $V, E$ 

```

re-sample the same state throughout the planning. Therefore, this requirement can be satisfied also without rewiring by altering the connections, if necessary, once the node is re-sampled. However, this affects how the information propagates along the tree and, hence, the cost evolution. To this end, both approaches are implemented and later compared by numerical validation with respect to the cost evaluation and time taken.

RRT*_MotionPrimitives without rewiring

One of the significant procedures of RRT* is rewiring the tree, in other words propagating the new connection information towards a set of neighborhood nodes. This is necessary to adjust the way the tree is connected, ensuring that each node is reached with the lowest possible cumulative cost. This is because once a node is sampled and added to tree it will not be sampled again while searching the continuous state space. However, using a discrete state space there is a non-zero probability that a state can be sampled again even though it is added as a node to the tree.

Taking advantage of the discrete representation of the state space and the possibility of re-sampling a particular state, the first approach builds the tree using the FindBestParent procedure only. This implies that even though a node is not connected in the best way to the tree, FindBestParent will find a better parent when that state is sampled again – provided that a better way exists. In other words, any change that happens to the tree concerns only s_{rand} and the information does not affect the neighbors.

In particular, if the sampled node s_{rand} is not a node in the tree, UpdateNew (line 9 in Algorithm 5) procedure is similar to RRT (Algorithm 2), adding s_{rand} and the corresponding edge, e_{best} , to the tree, i.e. $V \leftarrow V \cup s_{rand}$ and $E \leftarrow E \cup \{s_{best}, s_{rand}\}$. On the other hand, if s_{rand} is already a node in the tree, and if the best parent to connect, s_{best} returned by FindBestParent is different from the previous parent², s_{parent} ,

²Since FindBestParent returns the best parent to connect if s_{best} is not equal to s_{parent} , it means that s_{best} is a better parent, that was not a member of the tree when s_{rand} was sampled the previous time.

Algorithm 10: UpdateNew for RRT*_MotionPrimitives with rewiring

```

1 Input:  $s_{rand}, s_{best}, e_{best}$ 
2  $V \leftarrow s_{rand}$ 
3  $E \leftarrow E \cup \{e_{best}\}$ 
4  $V_{near} \leftarrow \text{NearNodesToGo}(s_{rand})$ 
5  $V, E \leftarrow \text{Rewire}(V_{near}, s_{rand})$ 
6 return  $V, E$ 

```

then the previous edge, UpdateExisting removes the previous edge, $e_{prev} = \{s_{parent}, s_{rand}\}$, from the set of edges and adds the new better edge, e_{best} (Algorithm 9).

RRT*_MotionPrimitives with rewiring

Similar to RRT*, the second version of RRT*_MP conveys the information about any updates to the tree towards the nearby nodes. Therefore, if a node is connected to the tree or if the cost of a node is changed, a set of neighborhood nodes is checked to determine whether there are other possible modifications.

If s_{rand} is sampled for the first time, firstly, UpdateNew (see Algorithm 10) procedure adds the corresponding node and the edge to the tree. Consequently, similar to RRT* framework, a set of nearby nodes are searched to check if any of them can be reached with a lower cumulative cost from s_{rand} , using the Rewire procedure. In that case, the set of nearby nodes, V_{near} , is computed using $\text{NearNodesToGo}(s_{rand})$ that computes the local neighborhood of s_{rand} when a forward motion is considered. Therefore, $\forall s_{near} \in V_{near}$ if $e = \{s_{rand}, s_{near}\}$ is collision free and

$$\text{Cost}(\rightarrow s_{rand}) + \text{Cost}(\sigma_{s_{rand} \rightarrow s_{near}}) \leq \text{Cost}(\rightarrow s_{near})$$

the tree is rewired, i.e., $E = \{E \setminus e_{prev}\} \cup \{e\}$, where e_{prev} is the previous edge connecting the node s_{near} to the tree.

Similar to the previous approach, UpdateExisting (Algorithm 11) deals with the case when a node is re-sampled and a better parent is found using FindBestParent. Initially, it changes the edge that connects s_{rand} to the tree by connecting it to a better parent. Consequently, it *rewires* the tree to propagate this change to the local neighborhood using Rewire (Algorithm 12).

Remark 3.1. The lowest cost solution found at iteration, i , using RRT*_MP when a rewiring step is applied is less than the lowest cost returned by the

Algorithm 11: UpdateExisting for RRT*_MotionPrimitives with rewiring

```

1 Input:  $s_{rand}, s_{best}, e_{best}$ 
2  $s_{parent} \leftarrow \text{Parent}(s_{rand})$ 
3 if  $s_{parent} \neq s_{best}$  then
4    $e_{prev} \leftarrow \{s_{parent}, s_{rand}\}$ 
5    $E \leftarrow \{E \setminus e_{prev}\} \cup \{e_{best}\}$ 
6    $V_{near} \leftarrow \text{NearNodesToGo}(s_{rand})$ 
7    $V, E \leftarrow \text{Rewire}(V_{near}, s_{rand})$ 
8 return  $V, E$ 

```

Algorithm 12: Rewire for RRT*_MotionPrimitives with rewiring

```

1 Input:  $V_{near}, s_{rand}$ 
2 for  $s_{near} \in V_{near}$  do
3    $\sigma_{near}, c_{near} \leftarrow \text{FindTrajectory}(s_{rand}, s_{near})$ 
4   if  $\text{Cost}(\rightarrow s_{rand}) + c_{near} < \text{Cost}(\rightarrow s_{near})$  then
5     if  $\text{CollisionFree}(\sigma_{near})$  then
6        $s_{parent} \leftarrow \text{Parent}(s_{near})$ 
7        $e_{prev} \leftarrow \{s_{parent}, s_{rand}\}$ 
8        $e \leftarrow \{s_{rand}, s_{near}\}$ 
9        $E \leftarrow \{E \setminus e_{prev}\} \cup \{e\}$ 
10 return  $V, E$ 

```

RRT*_MP without a rewiring step, i.e.:

$$c^* \leq \text{Cost}(\rightarrow s_{goal,i})^{RW} \leq \text{Cost}(\rightarrow s_{goal,i})^{noRW} \quad (3.10)$$

where c^* is the truly optimal cost and $\text{Cost}(\rightarrow s_{goal,i})^{RW}$ and $\text{Cost}(\rightarrow s_{goal,i})^{noRW}$ are the costs of the trajectory returned by $\text{BestTraj}(V_{goal,i})$ for the two different modes of the algorithm. This is due to the fact that, at each iteration, the information about a cost change propagates also to the neighbors, effecting a larger area. However, attaining a lower cost at a lower number of iterations does not mean a lower computation time since the computation time is also related to the operations carried out at each iteration.

Termination and best path selection

Algorithm terminates when a maximum number of iterations are reached, i.e., $i \rightarrow N$. Consequently, the best trajectory among all the trajectories that reach the goal region is selected using BestTraj procedure. Notably, using a discretized search space limits the number of nodes that can be sampled.

In other words, there is a maximum number of nodes $|V| = |S_{free}^\Delta|$ and, once all these nodes are included in the tree, the cardinality will not change; however, since the algorithm is allowed to re-sample, the way the tree is connected changes even though the maximum cardinality is reached.

It is also noteworthy that, there is a maximum number of iterations such that the best sequence of nodes reaching the goal region will be sampled and, after that, the minimum cost trajectory returned by the algorithm will not change regardless of the time given.

3.4 Implementation Details

In this section, some details are highlighted that improve the performance of the algorithm in terms of computational performance and memory needed.

Assumption 3.1 allows us to use the motion primitives by translating the trajectories to match the new boundary conditions obtained over the course of planning. However, considering the translation invariance property only, the database of motion primitives also includes the trajectories that correspond to a set of initial states that share the same position; i.e., π_0 equal to null vector. The size of the database can be further reduced, depending on the dynamic system at hand if stronger invariance properties hold. In this case, the optimal solutions of two different sets of boundary conditions can be obtained from each other if the initial and final configuration pairs are related with an affine transformation that preserves distances, extending the invariance property from translation to rotation and/or flipping (mirror rotation). Note that, this condition is limited to the state variables that correspond to the configuration and does not extend to include the derivatives such as velocity components.

As can be seen in Figure 3.1, with the dynamical system and cost function considered in Example 3.1, motion primitives, corresponding to boundary conditions that are symmetric with respect to the x -axis, are symmetric. A further analysis reveals that the same property holds for the y -axis as well, and that symmetric primitives are characterized by the same cost.

When for the considered dynamical system and cost criterion stronger invariance properties hold, like the axis symmetry in Example 3.1, the size of the database can be further reduced by storing only a few “reference trajectories”, and generating all the others using the invariance transformation. Figure 3.4 shows an example, referred to the TPBVP considered in Example 3.1, where the trajectories represented by the pairs $(s_0, s_f)_1$, $(s_0, s_f)_2$, $(s_0, s_f)_3$ are characterized by the same cost and can be easily mapped to a

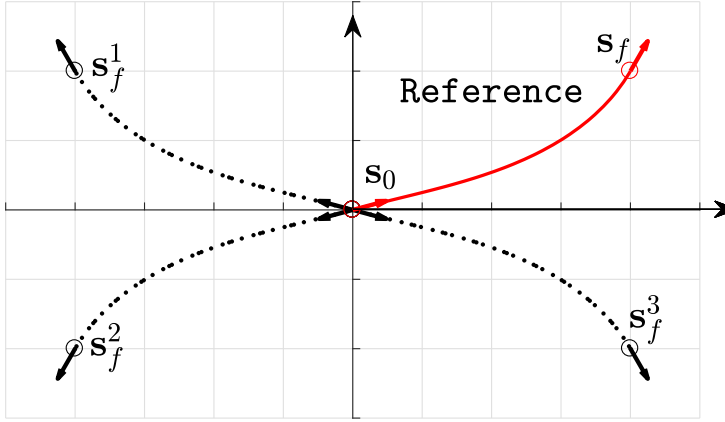


Figure 3.4: “Reference trajectory” corresponding to (s_0, s_f) (red solid line), and symmetric trajectories defined by the pair of boundary conditions of $(s_0, s_f)_1$, $(s_0, s_f)_2$, $(s_0, s_f)_3$, projected on 2D plane.

“reference trajectory” corresponding to the boundary value pair (s_0, s_f) . In this case, the size of the database can be further reduced storing only the “reference trajectory”.

Note that, if the “reference trajectory” technique is applied, it is necessary to augment the FindTrajectory procedure to account for further geometric transformations such as rotation or flipping. Although applying transformations does not have a significant effect on the computational performance, one might still consider this choice as a trade-off between the memory needed to store the database and the additional computational time needed for each search in the database.

In order to improve the database search efficiency, a LUT is created mapping the boundary conditions queried for the optimal trajectory to the corresponding indexes of the trajectories in the database. For this purpose, boundary conditions are quantized and a D -dimensional array is created to carry the proper indexes in the database. Consequently, the search for a particular index is reduced into extracting the corresponding element from this D -dimensional array using the quantized boundary conditions as the subscripts for each dimension. Since the database is created taking π_0 as the null vector, the size of the LUT is smaller with respect to the total number of boundary conditions, i.e., $D = 2d - n$, where n is the length of π_0 and d is the size of the search space. For example the database of motion primitives computed for a 3D state space (Figure 3.1) can be queried using

a 4D LUT.

3.5 Completeness and Optimality Analysis

In this section, the probabilistic completeness of the algorithm and optimality of the solution are discussed. Furthermore, some results on how close is the obtained solution in the discretized state space using motion primitives to the optimal one in continuum are also provided.

Let $\mathcal{G}_{free} = (V_G, E_G)$ be a graph with a fixed set of nodes and edges. The nodes of the graph correspond to the largest subset of $\mathcal{S}_{free}^\Delta$ such that for any $s \in V_G$ there exist at least a sequence of motion primitives that starts from s_0 and reaches $s \in V_G$ and lies in $\mathcal{S}_{free}$. The set of edges, E_G corresponds to a set of motion primitives built as follows: starting from s_0 include all the motion primitives, which lie in $\mathcal{S}_{free}$ and connect s_0 to all possible other states in V_G . Then, continue with the same strategy with all new discovered nodes iteratively until there is no other way of expanding the graph. Lastly, all the edges created this way form E_G .

Similarly to RRT*, RRT*_MP also generates a tree by extending it towards random samples in $\mathcal{S}_{free}$. However unlike the former one, the nodes and edges added to the tree belong respectively to V_G and E_G , so that the obtained tree $\mathcal{T}$ is a subgraph of $\mathcal{G}_{free}$, i.e., $\mathcal{T} \subset \mathcal{G}_{free}$. Recall that $\mathcal{T}_i$ corresponds to the tree after i number of iterations and c_i^Δ is the lowest cost trajectory represented in the tree $\mathcal{T}_i$.

Following analyses are based on the assumption that the motion planning problem using the grid representation admits a solution, i.e., there exist a way of reaching a state in $\mathcal{S}_{goal}^\Delta$ starting from s_0 with the available motion primitives in terms of intermediate nodes of the graph. In other words, when there exist a sequence of nodes in $\mathcal{G}_{free}$ that connects the initial state to the goal region. Assuming that $\mathcal{G}_{free}$ admits a solution to the motion planning problem, i.e. $V_G \cap \mathcal{S}_{goal}^\Delta \neq \emptyset$, then there exist a sequence

$$SQ^* := \{s_0^*, s_1^*, \dots, s_k^*\} \quad (3.11)$$

representing a *resolution optimal Δ -trajectory*, a minimum cost trajectory that starts at the initial state s_0 and ends in the goal region, i.e., $s_k^* \in \mathcal{S}_{goal}^\Delta$, such that $\forall s_j^* \in SQ^*, s_j^* \in \mathcal{S}_{free}^\Delta$ and $e_{s_{j-1}^*, s_j^*} \in E_{free}$. Let $c^{*\Delta}$ denote the optimal cost of this optimal trajectory, i.e., $\text{Cost}(\rightarrow s_k^*) = c^{*\Delta}$ which will be called as the *resolution optimal Δ -cost*.

Finally, the goal of RRT*_MP can be reformulated as to generate a tree that contains the SQ^* . Note that, one could in principle apply an exhaustive search on G_{free} . However, this is still an issue due to the combinatorial

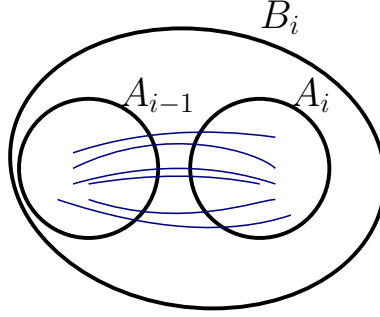


Figure 3.5: Attraction sequence defined with the properties: 1) Any point in A_{i-1} is closer to any point in A_i than any point outside of B_i 2) There exist motion primitives that can connect any point in B_i to a point in A_i .

nature of the problem, in particular due to the branching caused by the dimensionality of the state space and the number of nodes contained in the graph.

3.5.1 Probabilistic Completeness

Recall that the probabilistic completeness is defined as the probability of an algorithm to return a feasible solution to the motion planning problem, if the problem admits such a solution, tending to 1, as the number of iterations, i , reach infinity. This section analyses the probabilistic completeness of RRT*_MP in a more general way aligned with the proofs of RRT and RRT*. However, in the following section we prove this aspect differently, considering the specific properties of RRT*_MP.

Let a sequence of sets with finite number of elements, $A = \{A_0, A_1, \dots, A_k\}$ be a sequence of subsets of $\mathcal{S}_{free}^\Delta$ defined as an *attraction sequence* [58], where $A_0 = \{s_0\}$, $A_k = \mathcal{S}_{goal}^\Delta$ and the rest of the sequence is defined as follows

1. For each A_i there exists a basin B_i such that for any $x \in A_{i-1}$, $y \in A_i$ and $z \in \mathcal{S}_{free}^\Delta \setminus B_i$ in terms of the selected distance metric x is closer to y than it is to z , i.e., $\text{dist}(x, y) \leq \text{dist}(x, z)$;
2. For all $x \in B_i$ there exist a motion primitive that can connect x with a point $y \in A_i$.

Given such a sequence let p denote the lower bound on the probability of

selecting a random node in a particular A_k , that corresponds to

$$p = \min_{i \in \{1, 2, \dots, k\}} \left(\frac{|A_i|}{|\mathcal{S}_{free}^\Delta|} \right)$$

where $|\cdot|$ represents the cardinality of the set.

On condition that an attraction sequence exists, and the motion planning problem admits a solution, the proof of the following theorem follows directly from the probabilistic completeness of RRT [58] and RRT* [47].

Theorem 3.1. *RRT*_MP is probabilistically resolution complete. If an attraction sequence of length k exists, the probability of not finding a solution decreases exponentially as the number of iterations, i , increases, i.e.*
 $\mathbb{P}(V_i \cap \mathcal{S}_{goal}^\Delta = \emptyset) \leq e^{-\frac{1}{2}(pi-2k)}$

3.5.2 Optimality of the Solution

In this section the quality of the solution obtained by RRT*_MP is analyzed by addressing the following questions:

1. resolution optimality: if there exists a *resolution optimal* Δ -trajectory defined as SQ^* , is it possible to obtain such a trajectory?
2. asymptotic optimality: how close is the *resolution optimal* Δ -cost to the cost of the optimal trajectory, as the grid resolution increases and the grid converges to the continuous state space?

Assuming that there exists a sequence of nodes starting from s_0 and reach $\mathcal{S}_{goal}^\Delta$, the following is true for the cost of the trajectory returned by RRT*_MP, in comparison to the resolution optimal trajectory represented on $\mathcal{G}_{free}$.

Theorem 3.2. *As the number of iterations goes to infinity the cost of the trajectory returned by RRT*_MP converges to the resolution optimal Δ -cost with a probability equal to 1, i.e.,*

$$\mathbb{P} \left(\left\{ \lim_{i \rightarrow \infty} c_i^\Delta = c^{*\Delta} \right\} \right) = 1.$$

Moreover, the expected number of iterations required to converge to $c^{*\Delta}$ is upper bounded by $k|\mathcal{S}_{free}^\Delta|$, where k is the length of an optimal branch of $\mathcal{G}_{free}$.

Proof. RRT*_{MP} returns the resolution optimal solution if it discovers the optimal sequence, SQ^* , defined on $\mathcal{G}_{free}$. Assuming that the constant defining the cost threshold for selecting nearby nodes, γ , is selected large enough, (e.g. γ can be chosen so that $V_{reachToGo}$ and $V_{reachToCome}$ defined in (3.8) and (3.9) respectively, contains the $\text{BoundingBox}(\pi_{rand})$ for $n = |\mathcal{S}_{free}^\Delta|$), the algorithm determines an edge $e = \{s_{j-1}^*, s_j^*\}$ in order to extend the sequence SQ^* , in one of the following two ways:

1. if s_{j-1}^* is already a node in the tree, i.e., $s_{j-1}^* \in V_i$, any time s_j^* is sampled it is connected to s_{j-1}^* by the FindBestParent procedure. In fact, thanks to the monotonicity of the cost function, as $e = \{s_{j-1}^*, s_j^*\}$ is part of the optimal sequence, there is no better way of reaching s_j^* other than e .
2. if s_j^* is sampled before s_{j-1}^* and connected to an arbitrary node s_{prev} , such that $s_j^* \in V_i$ and $e = \{s_{prev}, s_j^*\} \in E_i$, the edge $e = \{s_{j-1}^*, s_j^*\}$ is selected when s_j^* is re-sampled once the previous part of the sequence is formed, i.e., $\exists \{s_0^*, s_1^* \dots s_{j-1}^*\}$, thanks to the UpdateExisting procedure, as the near nodes of s_{j-1}^* would include s_j^* , i.e., $s_j^* \in \text{NearNodesToGo}(s_{j-1}^*)$.

Therefore, two consecutive nodes that belong to the optimal sequence SQ^* will eventually be connected regardless of their sampling order. This means that obtaining the resolution optimal trajectory can be interpreted as advancing in the optimal sequence.

This property allows to model the process of determining the sequence SQ^* as an *absorbing Markov chain*. As a consequence, there is a positive probability $\mathcal{P}_j$ of advancing in the sequence, i.e., moving from s_{j-1}^* to s_j^* , and a probability $1 - \mathcal{P}_j$ of staying at the same state. Considering that the nodes are sampled i.i.d., the probability of sampling a node from $\mathcal{S}_{free}^\Delta$ can be seen as a Bernoulli trial, such that each vertex has at least a probability $\frac{1}{|\mathcal{S}_{free}^\Delta|}$ of being sampled at each iteration. Hence, in the worst case there is a probability $\mathcal{P}_j = \frac{1}{|\mathcal{S}_{free}^\Delta|}$ of advancing in the *Markov chain* and, since all states are transient states apart from s_k^* which is the absorbing state, the probability that the process is absorbed by s_k^* tends to 1 as i tends to infinity [7]. Moreover, for an absorbing Markov chain that is defined in this way, there is a finite number of expected iterations, $k|\mathcal{S}_{free}^\Delta|$, before the process is absorbed, i.e., before RRT*_{MP} returns the resolution optimal solution. $\square$

Note that, a third way of advancing in the sequence exists when the Rewire procedure is used, in that case if s_j^* is sampled before s_{j-1}^* and

connected to an arbitrary node s_{prev} , the edge $e = \{s_{j-1}^*, s_j^*\}$ is selected any time s_{j-1}^* is sampled, thanks to the Rewire procedure. This allows advancing in the sequence faster than the counterpart.

Clearly, the number of expected iterations increase with the depth of the solution, k , and the number of states represented on the grid. However, one advantage of RRT*_MP is the possibility of obtaining a solution fast and improving the quality within the allowed computing time.

Corollary 3.1. *RRT*_MP is probabilistically resolution complete, as the number of iterations goes to infinity the algorithm will return a solution to the motion planning problem, if there exist one in $\mathcal{G}_{free}^\Delta$, with a probability 1.*

Proof. The same methodology can also be used to prove the *probabilistic completeness* of RRT*_MP, modeling a sequence of nodes starting from s_0 and reaching the goal region while every node is connected with its best parent as a sequence to be discovered. Therefore, similar to the proof of Theorem 3.2, the probability of discovering this sequence as the number of iterations goes to infinity is 1. $\square$

The remaining of this section deals with the relation between the resolution optimal trajectory returned by RRT*_MP and the truly optimal trajectory in continuum. In order to simplify such a reasoning, the problem as given in Section 2.3.2 is assumed to be without any actuation and path constraints. Furthermore, the following assumptions, regarding the properties of the dynamical system and the existence of a solution, are enforced.

Assumption 3.2. *The following properties hold for the dynamical system in (2.1)*

- *the system is Small-Time Locally*

Attainable (STLA) [21]: A system is STLA from a state $s \in S$ if $\forall T > 0$ the reachable set of states from s in time $0 < t < T$, i.e., $\mathcal{R}^N(s, \leq T)$, contains a d -dimensional subset of $\mathcal{N}$, where $\mathcal{N}$ denote set of neighborhood states (Figure 3.6).

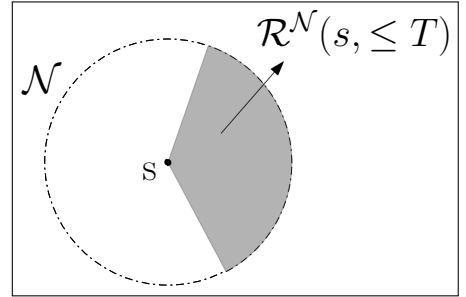


Figure 3.6: *Shape of a STLA dynamic system. $\mathcal{R}^N(s, \leq T)$ is the reachable set of states from s and it contains a subset of its neighborhood $\mathcal{N}$ which has the same dimensionality as the state space.*

- function $f(\cdot)$ representing the system dynamics is Lipschitz continuous with Lipschitz constant $\mathcal{K}_f$.
- function $\text{Cost}(\cdot)$ that assigns a cost to each trajectory is Lipschitz continuous with Lipschitz constant $\mathcal{K}_c$. Specifically, it is true that given two trajectories $\sigma_{(s_i \rightarrow s_j)}$ and $\sigma_{(\tilde{s}_i \rightarrow \tilde{s}_j)}$ the cost function satisfies

$$|\text{Cost}(\sigma_{(s_i \rightarrow s_j)}) - \text{Cost}(\sigma_{(\tilde{s}_i \rightarrow \tilde{s}_j)})| \leq \mathcal{K}_c \left\| \begin{vmatrix} s_i \\ s_j \end{vmatrix} - \begin{vmatrix} \tilde{s}_i \\ \tilde{s}_j \end{vmatrix} \right\|$$

As this thesis is mainly focused on state spaces that are manifolds, the following assumption is introduced concerning a property of these manifolds.

Assumption 3.3. *The state space manifold for a system with d number of states is a subspace of the d -dimensional Euclidean space, $\mathbb{R}^d$, and, therefore can be locally treated as $\mathbb{R}^d$.*

The above assumptions allow us to make use of Euclidean norms following in this section. Such a property is required for state spaces that include the derivatives of the configuration components, or for systems with configuration spaces that are already manifolds. However, the approach presented here naturally generalizes to state spaces that are already Euclidean.

In order to compare the resolution optimal trajectory returned by RRT^*_{MP} with the optimal trajectory in continuum, trajectories whose points are all away from obstacles by a certain distance are considered. For this reason, the definition of obstacle clearance of a trajectory, i.e., the minimum distance between obstacles and points belonging to the trajectory, has to be introduced.

Definition 3.4 (ϵ -obstacle clearance). *Given a trajectory $\sigma(t)$, $t \in [0, T]$, if any ball $\mathcal{B}_\epsilon(\sigma(t))$ of radius ϵ and centred at any point along $\sigma(t)$ is strictly inside $\mathcal{S}_{\text{free}}$, the ϵ -obstacle clearance property holds for σ .*

Definition 3.5 (ϵ -free space). *Let $\sigma(t) : [0, T] \rightarrow \mathcal{S}_{\text{free}}$ be a trajectory which has ϵ -obstacle clearance, the ϵ -free space along σ is given by*

$$\mathcal{S}_{\sigma_\epsilon} := \bigcup_{t \in [0, T]} \mathcal{B}_\epsilon(\sigma(t))$$

Then, a set of trajectories that are called to be ϵ -similar to σ and can be introduced.

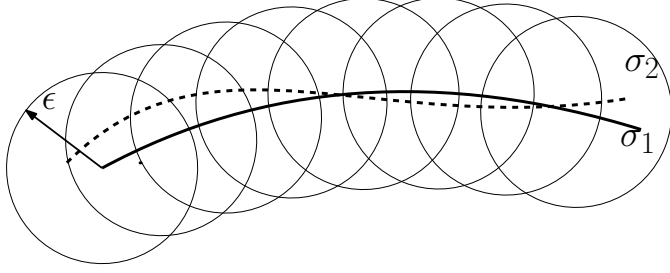


Figure 3.7: Covering σ_1 with ϵ -balls it is shown that σ_2 is ϵ -similar to σ_1

Definition 3.6 (ϵ -similarity). Any trajectory $\tilde{\sigma}(t) : [0, \tilde{T}] \rightarrow \mathcal{S}_{free}$ is said to be ϵ -similar to σ if it lies in the $\mathcal{S}_{\sigma_\epsilon}$ volume, i.e., if

$$\forall t \in [0, \tilde{T}] : \tilde{\sigma}(t) \in \mathcal{S}_{\sigma_\epsilon}$$

Figure 3.7 shows two ϵ -similar trajectories. However, having an ϵ -free space along a trajectory is not sufficient to guarantee the existence of ϵ -similar trajectories, as this existence depends also on the properties of the dynamical system (2.1). The following definition relates the ϵ -similarity with the ϵ -free space through the system dynamics.

Definition 3.7 (ϵ -dynamic clearance). Given a trajectory $\sigma(t)$, where $t \in [0, T]$, consider two time instants t_1, t_2 , with $t_1 < t_2$, and the corresponding states $s_1 = \sigma(t_1)$ and $s_2 = \sigma(t_2)$. If for any $t_1 < t_2$ there exists a set of states, inside a ball of radius $\alpha\epsilon$, for $0 < \alpha < 1$, centered at s_2 , that are reachable from s_1 without leaving the ϵ -free space around $\sigma(t)$, then σ has ϵ -dynamic clearance.

Existence of an ϵ -dynamic clearance is essential for a sampling-based algorithm to find a solution since it ensures that there are reachable set of states, characterized as the ones inside a ball of radius $\alpha\epsilon$, that can contribute to obtain ϵ -similar trajectories.

Let Σ_ϵ denote all the trajectories that solve the *feasible motion planning problem* given in Section 2.3.1³ and that have at least ϵ -obstacle and dynamic clearance. Let c_ϵ^* denote the minimum cost over all Σ_ϵ which corresponds to the ϵ -optimal trajectory, σ_ϵ^* (Figure 3.8). Note that, as ϵ goes to zero, i.e., $\epsilon \rightarrow 0$, the ϵ -optimal trajectory converges to the truly optimal trajectory, σ^* , solving the *optimal kinodynamic motion planning problem* defined in Section 2.3.2. Furthermore, the cost of the truly optimal trajectory is denoted as c^* .

³With a slight abuse of the previously introduced formalization, here, the feasible motion planning problem refers to the problem without any actuation and path constraints

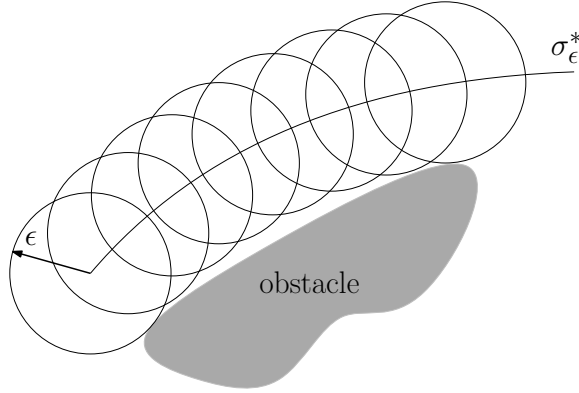


Figure 3.8: σ_ϵ^* is the optimal trajectory with at least ϵ -obstacle and dynamic clearance.

Due to the discretized nature of RRT*_MP it is not possible to converge to the optimal trajectory in continuum, however we are going to show that RRT*_MP will eventually generate an ϵ -similar trajectory, σ_ϵ^Δ , to the optimal trajectory with a particular ϵ -clearance, σ_ϵ^* . Consequently, this result will be used to show that as the resolution of the grid increases and as ϵ converges to zero, the *resolution optimal* Δ -cost will converge to the truly optimal cost, i.e., c^* .

Theorem 3.3. *Let $\epsilon > 0$ be a clearance such that the motion planning problem admits an ϵ -optimal trajectory, then,*

- *for a sufficiently fine gridding, $\mathcal{G}_{free}$ contains an ϵ -similar trajectory, σ_ϵ^Δ to the ϵ -optimal one, σ_ϵ^* ,*
- *as the grid resolution increases the resolution optimal Δ -cost, $c^{*\Delta}$, converges to c^* .*

Proof. Let $\mathcal{S}_{\sigma_\epsilon^*}$ be the ϵ -free space around σ_ϵ^* . Since by definition σ_ϵ^* has ϵ -dynamic clearance, a set of states contained in a ball of radius $\alpha\epsilon$ that are reachable from any $s \in \sigma_\epsilon^*$ without leaving $\mathcal{S}_{\sigma_\epsilon^*}$ can be determined. Therefore one can define a sequence of balls, $\{\mathcal{B}_\delta(s_m)\}$, for $m = 0, 1, \dots, M-1$, characterized by a radius δ and a center determined in such a way that $\text{Cost}(\sigma_{s_m \rightarrow s_{m+1}}) = l$ (Figure 3.9).

From Assumption 3.2, in particular from the Lipschitz continuity of the system dynamics, it follows that a sequence of balls, $\{\mathcal{B}_\delta(s_m)\}$, of radius

$$\delta = \frac{\alpha\epsilon}{2\mathcal{K}_f} \quad (3.12)$$

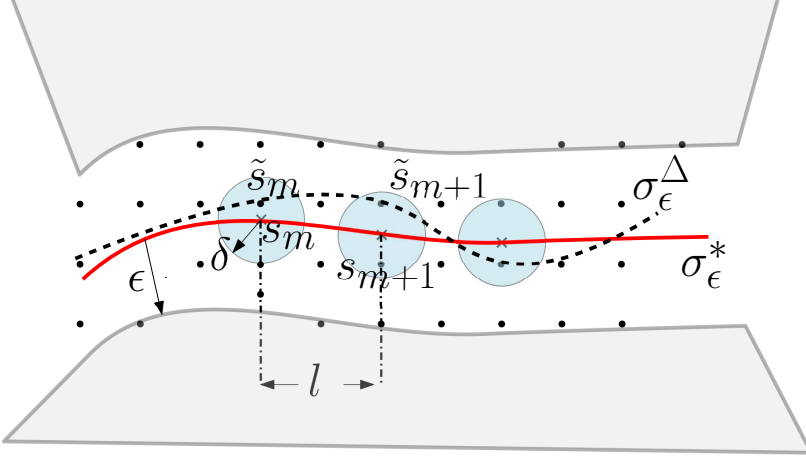


Figure 3.9: Tiling σ_ϵ^* with δ balls, such that each consecutive center is l away. The ϵ -similar trajectory that passes through these δ balls lies in the ϵ -free space along σ_ϵ^*

exists⁴, such that any state within $\mathcal{B}_\delta(s_m)$ can be attracted to $\mathcal{B}_\delta(s_{m+1})$ (see [51]), and any such trajectory would be in $\mathcal{B}_\epsilon(s_m)$. Therefore, tiling σ_ϵ^* with balls of radius δ a sequence is created so that, if RRT*_MP samples within this sequence of balls it generates an ϵ -similar trajectory to the ϵ -optimal one.

Since any node sampled within $\mathcal{B}_\delta(s_{m+1})$ can be connected to $\mathcal{B}_\delta(s_m)$ and assuming that the discretization is fine enough so that none of the $\mathcal{B}_\delta(s_m)$ are empty, there exist a sequence of nodes represented in $\mathcal{G}_{free}$, connecting s_0 with a node in the goal region. Therefore, if RRT*_MP discovers this sequence, it will generate an ϵ -similar trajectory to the ϵ -optimal one.

Consequently, similar to the proof of Theorem 3.2, since by assuming the existence of such a ball sequence, if $\tilde{s}_m$ is a node in the tree, once $\tilde{s}_{m+1}$ is sampled, it will be connected to $\tilde{s}_m$. Lastly, when the goal node, $\tilde{s}_{M-1}$ is reached, the process will be absorbed and as the number of iterations goes to infinity RRT*_MP will generate such a sequence with probability 1. Thus, it will eventually generate an ϵ -similar trajectory to the ϵ -optimal one.

Assume that the discretization is small enough to allow generating an ϵ -similar trajectory to the ϵ -optimal one, then, the relation between the cost of this ϵ -similar trajectory, c_ϵ^Δ and the cost of the ϵ -optimal trajectory, c_ϵ^* can be established.

The cost of the σ_ϵ^* can be defined as the sum of the costs of the trajec-

⁴Without loss of generality it is assumed that $\mathcal{K}_f \geq 1$.

ries connecting consecutive $\mathcal{B}_\delta(s_m)$ balls, i.e.:

$$c_\epsilon^* = \sum_{m=0}^{M-1} \text{Cost}(\sigma_{s_m \rightarrow s_{m+1}}) \quad (3.13)$$

and similarly the cost of the σ_ϵ^Δ is

$$c_\epsilon^\Delta = \sum_{m=0}^{M-1} \text{Cost}(\sigma_{\tilde{s}_m \rightarrow \tilde{s}_{m+1}}). \quad (3.14)$$

where c_ϵ^Δ is the cost of ϵ -similar trajectory. Subtracting (3.13) from (3.14), the difference can be written as:

$$c_\epsilon^\Delta - c_\epsilon^* = \sum_{m=0}^{M-1} (\text{Cost}(\sigma_{\tilde{s}_m \rightarrow \tilde{s}_{m+1}}) - \text{Cost}(\sigma_{s_m \rightarrow s_{m+1}})) \quad (3.15)$$

By the Lipschitz continuity of the cost function given in Assumption 3.2 it follows that

$$c_\epsilon^\Delta - c_\epsilon^* = \sum_{m=0}^{M-1} \mathcal{K}_c \delta \quad (3.16)$$

where $\mathcal{K}_c$ is the Lipschitz constant for the cost function. Therefore:

$$c_\epsilon^\Delta \leq c_\epsilon^* + \mathcal{K}_c M \delta \quad (3.17)$$

Finally, considering that the maximum number of balls for a particular ϵ -ball sequence can be upper bounded by

$$M \leq \frac{c_\epsilon^*}{l}$$

from (3.17) it follows that

$$c_\epsilon^\Delta \leq \left(1 + \frac{\mathcal{K}_c \delta}{l}\right) c_\epsilon^* \quad (3.18)$$

Finally, recall from (3.12) that δ is a function of ϵ then,

$$c_\epsilon^\Delta \leq \left(1 + \frac{\mathcal{K}_c \alpha \epsilon}{2\mathcal{K}_f l}\right) c_\epsilon^* \quad (3.19)$$

which shows that the cost of the ϵ -similar trajectory for a specific gridding is upper bounded by the cost of the ϵ -optimal trajectory and the ϵ -clearance. Finally, since by Theorem 3.2, as $i \rightarrow \infty$, RRT^*_{MP} will eventually generate

a *resolution optimal* Δ -trajectory with a cost $c^{*\Delta}$ and since it is the cost of the *resolution optimal* Δ -trajectory, which corresponds to the optimal trajectory that can be obtained given a particular discretization, then it is also true that

$$c^{*\Delta} \leq c_\epsilon^\Delta \quad (3.20)$$

$$\leq \left(1 + \frac{\mathcal{K}_c \alpha \epsilon}{2\mathcal{K}_f l}\right) c_\epsilon^* \quad (3.21)$$

which also provides an upper bound on the *resolution optimal* Δ -cost.

Lastly, as $\mathcal{K}_f$ and $\mathcal{K}_c$ are constants and l is fixed and by the assumption on the ϵ -dynamic clearance such that there exist an α for any ϵ , one can pick a gridding such that as the resolution of the grid increases and goes to infinity the ϵ can converge to zero. Then, as ϵ goes to zero the *resolution optimal* Δ cost will converge to the cost of the ϵ -optimal trajectory, i.e.,

$$\lim_{\epsilon \rightarrow 0} c^{*\Delta} = c^* \quad (3.22)$$

where c^* is the cost of the ϵ -optimal trajectory when ϵ is zero, i.e., cost of the truly optimal trajectory. $\square$

Theorem 3.3 states that the asymptotic optimality is reached when the number nodes in $\mathcal{G}_{free}$ goes to infinity, i.e., when the uniform gridding converges to the continuous state space. In that case as the number of iterations goes to infinity the cost of trajectory returned by RRT*_MP will converge to the cost of the optimal trajectory. Furthermore, it establishes an upper bound as a function of ϵ -optimal trajectory and the ϵ -clearance such that as the grid resolution increases the upper bound on the *resolution optimal* Δ -cost will decrease. This owes to the fact that, as the grid resolution gets finer, there exist ϵ -similar trajectories to the ϵ -optimal one represented in $\mathcal{G}_{free}$ for smaller values of ϵ , guaranteeing that none of the balls tiling the optimal trajectory is empty. However, as seen from Theorem 3.2, as the number of discrete states increase RRT*_MP will take more iterations to return the *resolution optimal* Δ -trajectory. Therefore, depending on the problem at hand, one should make the best compromise between the computing time (and size of the database) and the performance in terms of cost.

3.6 Numerical Example

In this section a numerical example is presented to show the effectiveness of the proposed algorithm.

3.6.1 Unicycle

A 4D state-space (x, y, θ, v) representing a unicycle like vehicle moving on a planar surface is considered. The vehicle is described by the following dynamics

$$\begin{cases} \dot{x}(t) = v(t) \cos \theta(t) \\ \dot{y}(t) = v(t) \sin \theta(t) \\ \dot{\theta}(t) = w(t) \\ \dot{v}(t) = a(t) \end{cases} \quad (3.23)$$

where the (x, y) is the position of the vehicle and θ the orientation with respect to a global frame, v and w are the linear and angular velocity, respectively. The control input is given by $u = [w, a]^T$, where a is the linear acceleration.

In order to obtain a database, the motion primitives are computed for pairs of initial and final states, $s_0 = [0, 0, \theta_0, v_0]$ and $s_f = [x_f, y_f, \theta_f, v_f]$. For each pair s_0 and s_f a trajectory is precomputed solving the TPBVP by minimizing the cost function

$$J(u, \tau) = \int_0^{t_f} [1 + u(t)^T R u(t)] dt$$

that minimizes the total time of the trajectory, t_f , at the same time penalizing the total actuation effort using the weights given by R . The differential constraints representing the system dynamics are taken as in (3.23).

3.6.2 Results and Discussions

The test environment is bounded by $x \in [0, 13]$ m, and $y \in [0, 10.25]$ m. The other states are bounded as $\theta \in [0, 2\pi]$ rad and $v \in [0, 4]$ m/s. The control variables a and w are equally penalized using $R = 0.5I_2$, and bounded as $a \in [-3, 3]$ m/s², $w \in [-3, 3]$ rad/s.

In order to solve the TPBVPs, MATLAB toolbox GPOPS [72], a non-linear optimization tool based on the Gauss pseudo-spectral collocation method, was used.

Three different databases with different gridding properties have been considered. For all of them the initial states have the same position $(x_0, y_0) = (0, 0)$. The first database is based on a low resolution uniform square grid (Figure 3.10a), where $(x_f, y_f) \in [-2, 0) \cup (0, 2] \times [-2, 0) \cup (0, 2]$

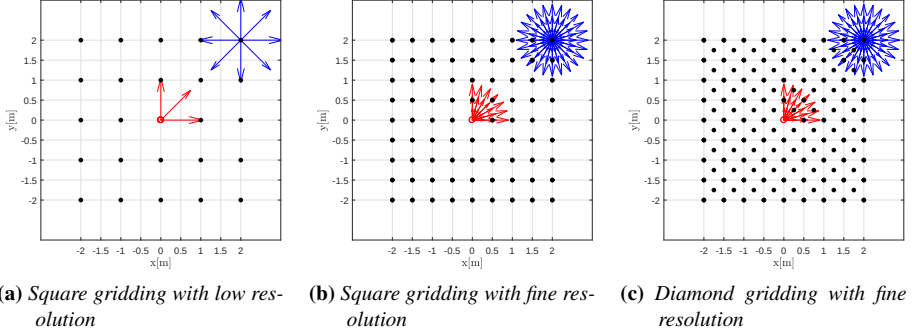


Figure 3.10: Different samplings for the database generation. Red arrows represent the initial headings and velocities, blue arrows are the final ones. Different arrow sizes correspond to different velocities. The database contains a reference trajectory from each initial state (red) to every final grid point (blue).

and each cell has a size of one meter. The initial orientation, θ_0 is selected among three values $\{0, \pi/4, \pi/2\}$ rad, the final orientation θ_f can take 8 equally spaced values in the range $[0, 2\pi)$ rad. For the initial and final velocities, v_0 and v_f , a minimum and a maximum velocity of 1 m/s and 4 m/s are considered.

The second database is based on a fine resolution uniform square grid (Figure 3.10b), where $(x_f, y_f) \in [-2, 0) \cup (0, 2] \times [-2, 0) \cup (0, 2]$ and each square cell has a size of half a meter. The initial and final orientations can take 24 equally spaced values in the range $[0, 2\pi)$ rad. The initial and final velocities are selected among 5 equally spaced values in the range $[0, 4]$ m/s.

Finally, the third database is based on a uniform diamond grid (Figure 3.10c), characterised by the same initial and final states as the previous one, plus some additional final states at $(x_f, y_f) \in [-1.75, 1.75] \times [-1.75, 1.75]$ with a discretization step of half a meter in each direction. These additional states are characterized by the same orientation and velocity as the rest of the database.

Simulations are performed on an IntelCore i7 @2.40 GHz personal computer with 8GB RAM and the algorithm is implemented in MATLAB. For a virtual indoor environment the vehicle has an initial state $s_0 = [1, 0, \pi/2, 0]$, and the goal area is defined as a 0.5m square centered at (9, 11). It is also considered that the vehicle should stop at the end of the trajectory, i.e., $v_f = 0$ m/s. Figures (3.11-3.13) show the trees built and the corresponding optimal trajectories obtained for various cardinalities. The first obtained

Table 3.1: Number of vertices in the state space grid obtained by applying different resolutions together with the branching factor introduced by each of the vertex represented in the corresponding grid (only the minimum and the maximum are shown). Square gridding with low resolution (LR), square gridding with fine resolution (FR), diamond gridding with fine resolution (DR).

Type of the grid	Number of vertices in the grid	Branching factor	
		min	max
LR	2,460	370	324
FR	68,000	7,300	9,100
DR	131,000	12,000	15,000

solutions, the first trajectory that reaches the goal region for each discretization strategy, are also shown.

In order to state the impact of the resolution to the size of the search space, the number⁵ of vertices of the grid used for each database resolution is shown in Table 3.1. Moreover the minimum and the maximum branching factors, i.e., number of neighbors that each vertex is connected to, are also provided. This highlights the issue in the combinatorial nature of the problem of explicitly building the graph of motion primitives and applying a graph search.

Figure 3.14 shows the velocity and the actuation profiles for the optimal trajectories given in Figure 3.11(d) and Figures 3.12-3.13(f), clearly demonstrating that the velocity constraint and bounds on actuation (w, a) are satisfied. Note that the velocity profile that corresponds to the coarse resolution square gridding exhibits a jerky acceleration behaviour, due to the fact that the velocity at each node is constrained to be exactly one of the values in the database. This demonstrates that the velocity discretization step has to be accurately selected if a smoother velocity profile is required.

The same planning problem has been solved for 10 independent simulation runs. Figure 3.15 shows the average cost evolutions related to the low resolution, fine resolution and diamond gridding, as the number of iterations increases. As expected, the cost reduces increasing the number of iterations, and converges to the *resolution optimal Δ -cost*: once this minimum is achieved the solution will not further improve.

As can be easily seen, motion primitives computed using a denser grid provide lower cost plans. Moreover, the resolution optimal Δ -cost achieved using the fine resolution square and diamond grids are similar, demonstrat-

⁵The numbers are rounded to 3 significant digits.

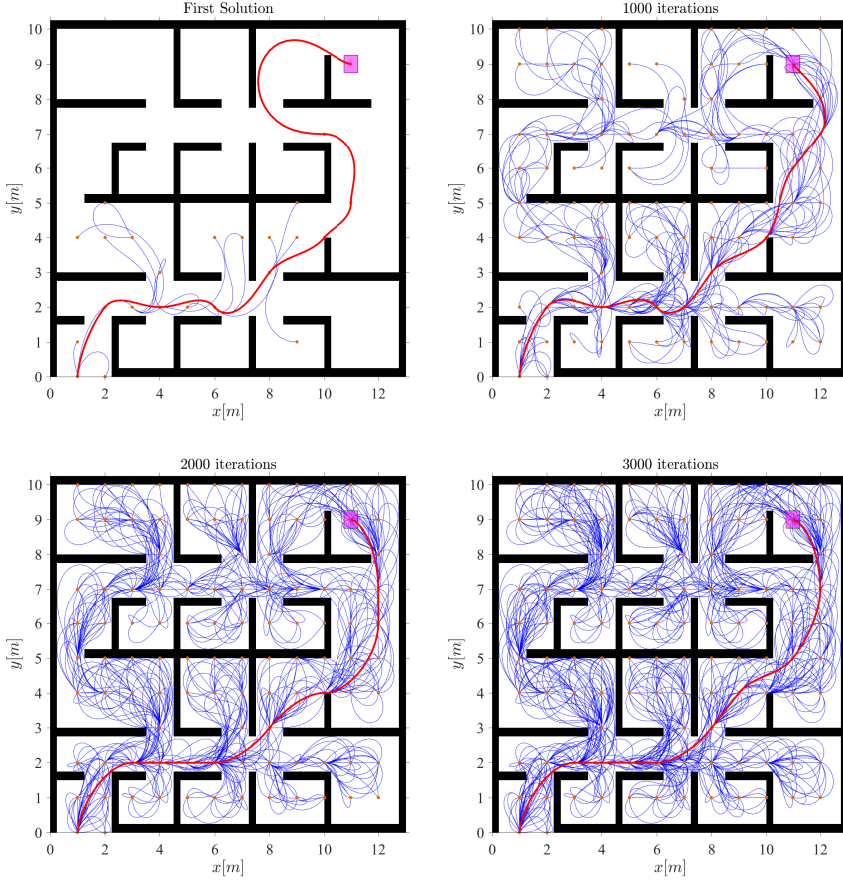


Figure 3.11: Simulation results using an indoor map and square gridding with low resolution for various number of iterations including the first solution. Blue lines represent the edges connecting the vertices of the tree (orange). Red color represents the lowest cost trajectory. Magenta area is the goal region while green area marks the initial state of the vehicle.

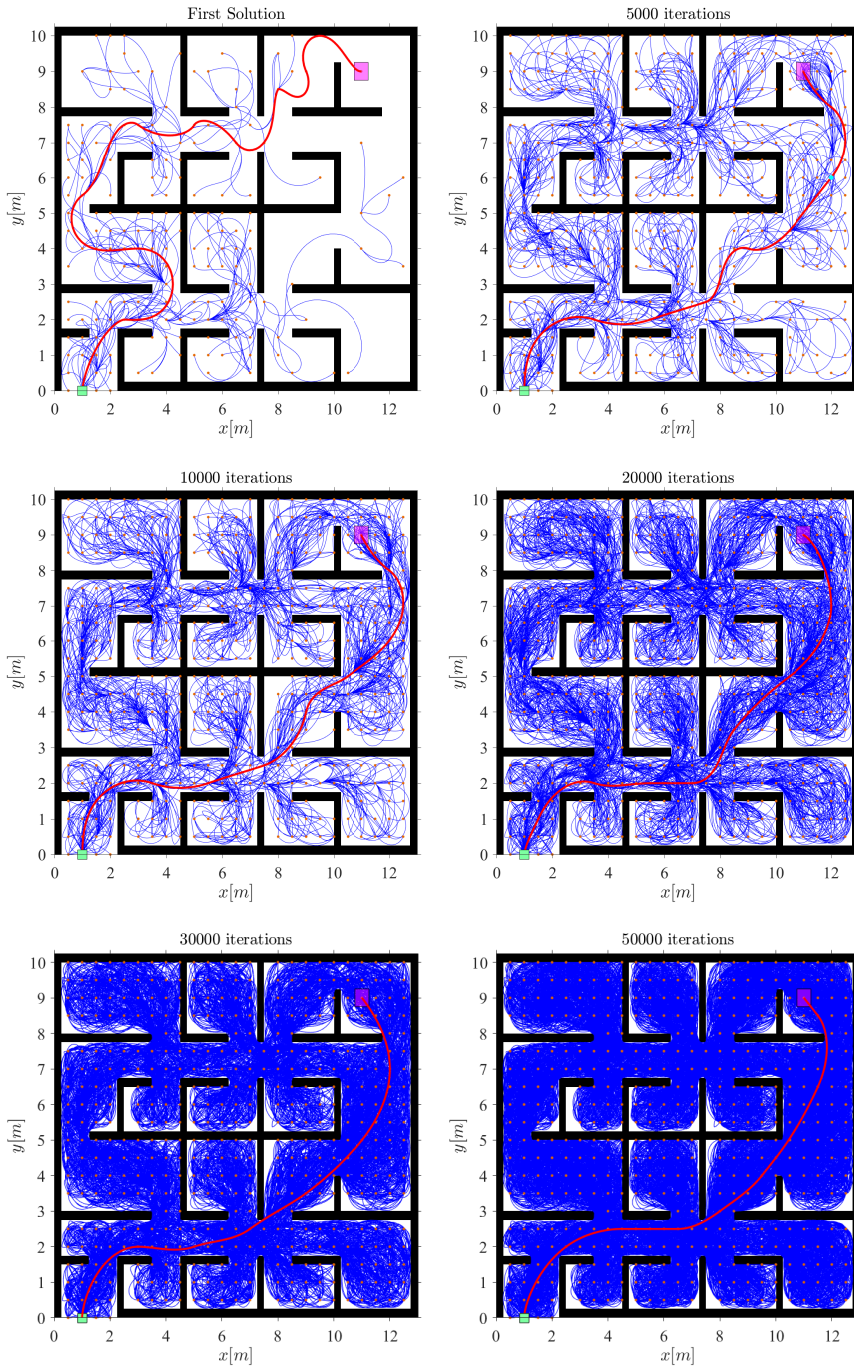


Figure 3.12: Simulation results using an indoor map and square gridding with fine resolution for various number of iterations.

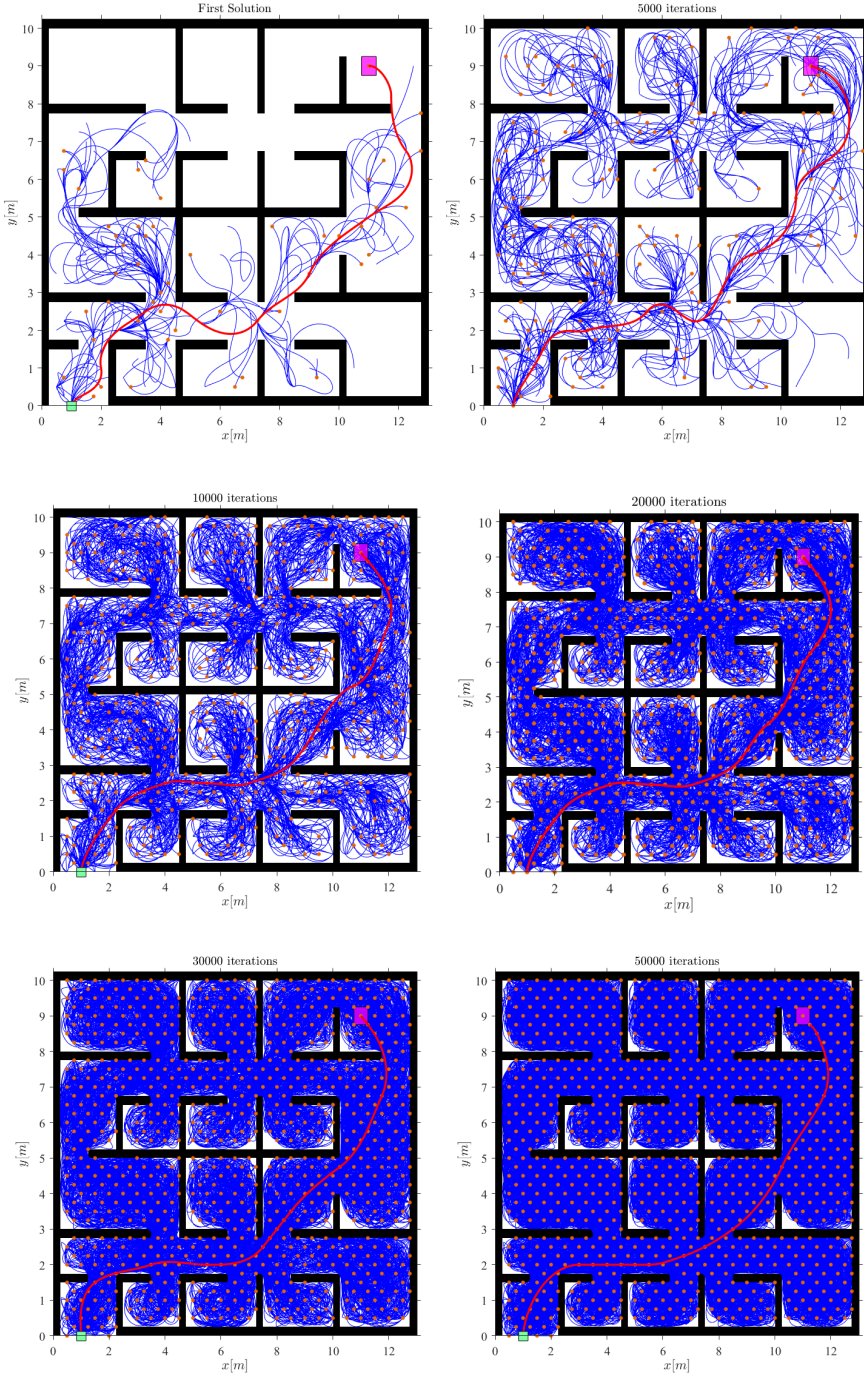


Figure 3.13: Simulation results using an indoor map and diamond gridding with fine resolution for various number of iterations.

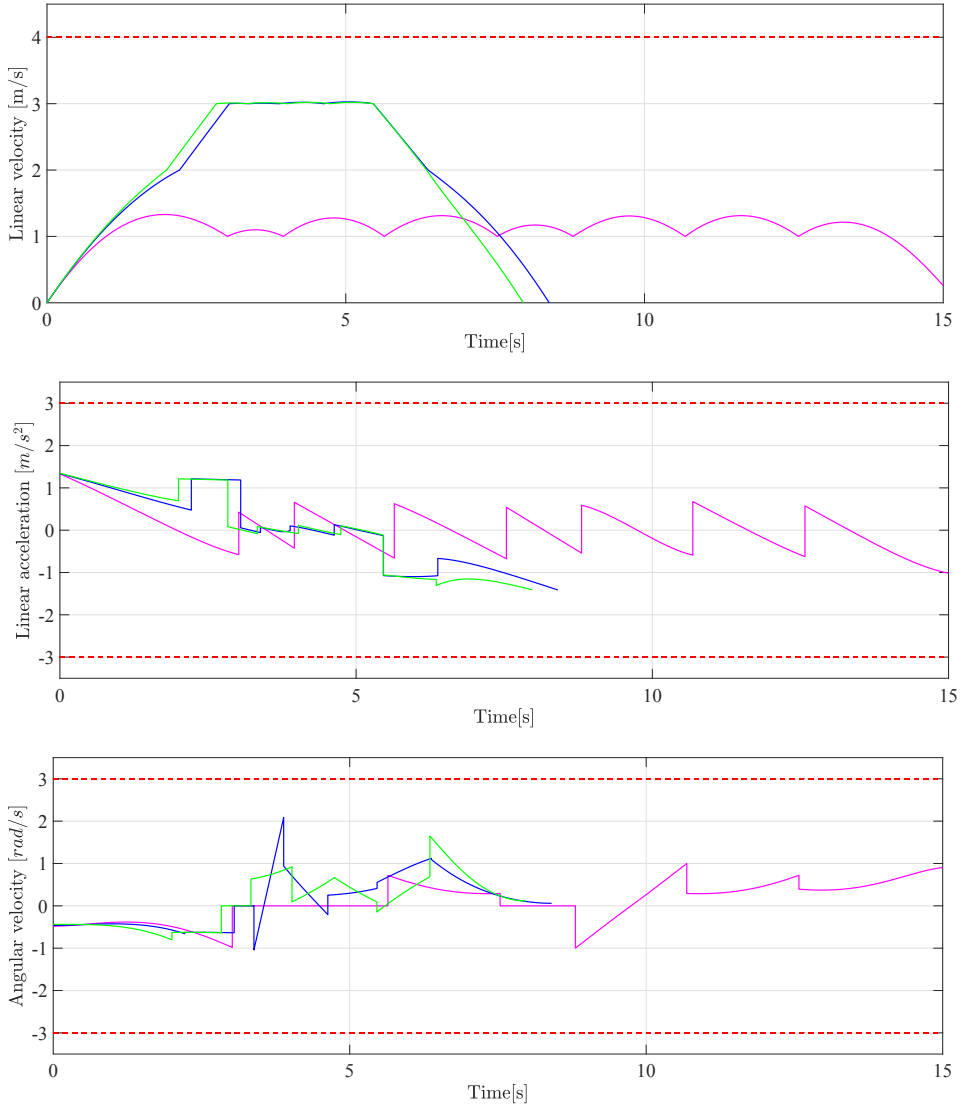


Figure 3.14: Linear velocity and actuation profiles for the optimal trajectories in Figure 3.11(d)(pink line) and Figures 3.12-3.13(f)(blue and green lines respectively). Red lines show the velocity and actuation limits.

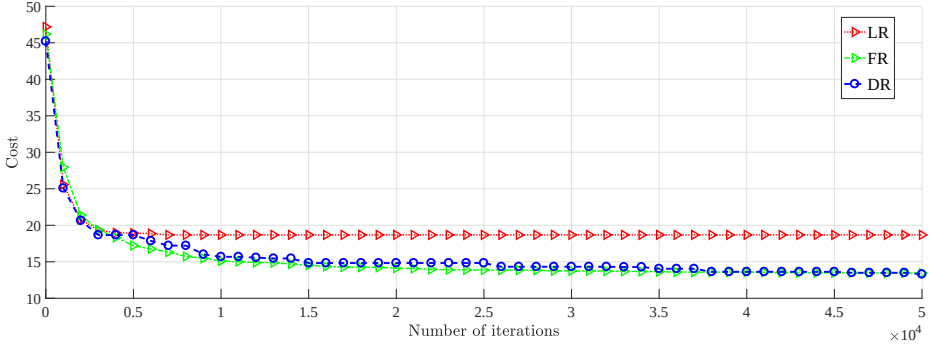


Figure 3.15: Average costs of 10 random simulations run for each different search space discretization, low resolution square grid(LR), fine resolution square grid(FR) and fine resolution diamond gridding(DR).

ing that the choice of the discretization step is strictly related to the specific problem. Finally, it is worth mentioning that as the resolution of the grid increases, the cardinality of $\mathcal{S}_{free}^{\Delta}$ increases as well, slowing down the convergence to the resolution optimal Δ -cost. Considering this aspect, a combined approach can be used that starts exploring a lower resolution gridding which is a subset of the actual one and then refines the solution with a fine resolution database.

Figure 3.16 shows the cost evolution for different discretization properties and with different information propagation. For both types of gridding the cost of the trajectory returned by the algorithm reduces over the course of planning. However, as it is clearly visible from the low resolution discretization there exists a resolution optimal trajectory that can be reached in a finite number of iterations, therefore once this limit is reached the tree will not be updated. Due to the lower number of states in the sampling set, low resolution discretization converges to the resolution optimal trajectory faster than using finer gridding. As expected, the cost of this trajectory is higher than the one returned when a finer resolution is used.

As highlighted before in (3.10) for the same discrete state space, the rewiring option allows the cost to reduce faster since every change in the tree is communicated to a set of neighbors. In other words the cost of the trajectory returned by the algorithm after the same number of iterations is lower when the Rewire procedure is used, hence the optimal cost is reached in a lower number of iterations. However, as the Figure 3.17 illustrates higher number of iterations do not express the elapsed time to reach a particular iteration since RRT*_MP with rewiring requires more time

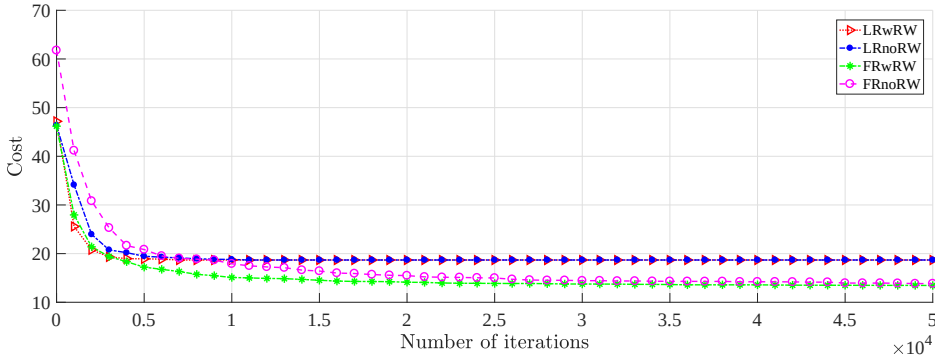


Figure 3.16: Average cost of the best trajectory as a function of iterations for different discretizations and with/without the use of rewiring procedure.

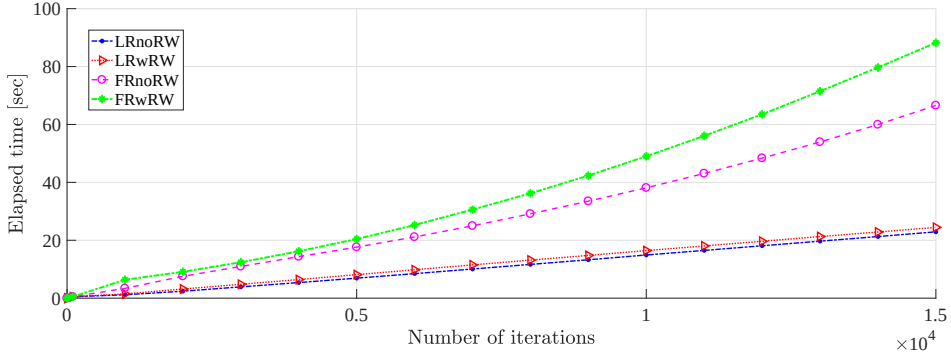


Figure 3.17: Average time taken to arrive a particular iteration for different discretization approaches with/without the use of rewiring procedure.

per iteration. The computation time increases logarithmically with the tree cardinality, the reason behind is the number nearby nodes to be checked. Clearly, a low resolution state space would result in lower number of neighbors to check connections.

Figure 3.18 shows the tree cardinality as a function of iterations. Since the algorithm does not generate new nodes, once the maximum number of nodes for the low resolution square gridding is reached, the cardinality does not increase. This is also the reason for the linear computation time profiles that correspond to a low resolution square gridding, since the number of neighbors do not change after all the nodes are added to the tree, the computation time increases linear as a function of iterations.

Finally, it is worth mentioning that the resulting computation time is

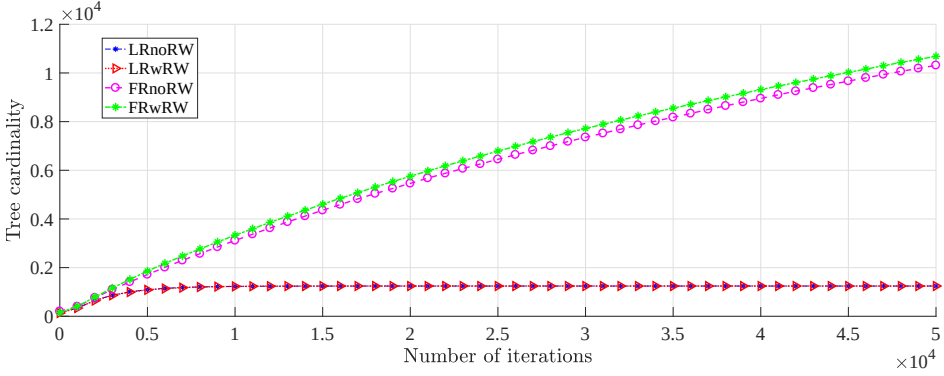


Figure 3.18: Average tree cardinality as a function of number of iterations for different discretization approaches with/without the use of rewiring procedure..

promising, even for online replanning in the case of dynamic and partially known environments. Code optimization and a C/C++ implementation can be considered for a further speed up.

3.7 Conclusion

This section discussed the randomized sampling based approaches to the motion planning problem with kinodynamic constraints and elaborated on the approach based on motion primitives to contribute to the framework of optimal sampling based algorithms. The key elements that affect the performance of a sampling based algorithms are identified as the selection of a good distance metric and a computationally efficient steering function. Proposed method simplifies the solution of TPBVPs into a LUT improving the performance of the algorithm by introducing an efficient steering function and the possibility to use the exact metric for selecting the neighbors. The algorithm is applicable for virtually any differentially constrained dynamic system with bounds on actuation. It is also worth to mention that, proposed approach can also function using a set of experimentally driven motion primitives as long as some assumptions on the systems dynamics can be made, removing the dependency on having the exact model of the system.

CHAPTER 4

Planning with Homotopy Class Constraints

Planning with topology constraints has gained an increasing attention in the recent years. Other than obtaining a collision free trajectory that minimizes a selected objective, one might be interested in how the resulting trajectory avoids the obstacles. This interest a number of fields in the robotics and autonomous vehicle applications, ranging from avoiding dangerous zones, coordinating multiple agents for better exploration and following rules on how to avoid obstacles. Furthermore, other than imposing certain rules about how to avoid the obstacles, it is also possible to use the topology information for obtaining a number of distinct good trajectories or for decomposing the otherwise complex trajectory generation problem.

Motivated with these examples, this chapter explores the planning approaches that take into account also the specific topology of the environment. In particular, it explains the basic concepts of homotopy in the context of motion planning formalizing the correspondence between a set of trajectories and the topology of the environment. Consequently, the systematic generation of homotopy class constraints and a selection of distinct promising classes of trajectories are discussed. Finally, an exact optimal motion planner that is based on optimal control and homotopy class constraints is proposed. Moreover, the sampling-based method proposed in the

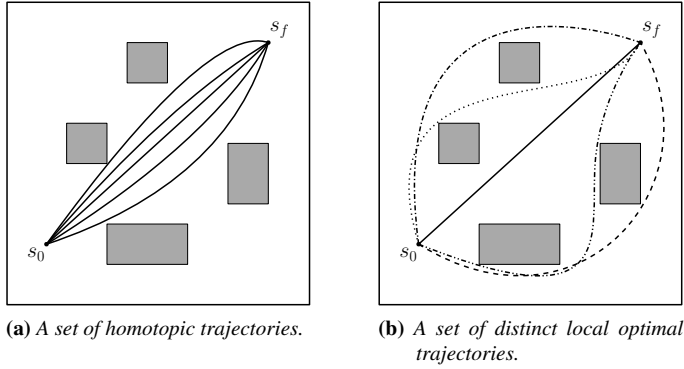


Figure 4.1: Topology information on motion planning.

previous chapter is enhanced with homotopy awareness.

4.1 Homotopy in Motion Planning

The trajectories that avoid the obstacles in a similar manner are called homotopic trajectories, in other words, the homotopic trajectories can be deformed into each other without intersecting with the obstacles (Figure 4.1a).

Let $\mathcal{P}$ denote the Cartesian space that the vehicle operates, such that $\mathcal{P} \subset \mathbb{R}^{(2,3)}$. Let $\tau(\lambda) : [0, 1] \rightarrow \mathcal{P}$ where $\forall \lambda \in [0, 1]$ be the geometric path that corresponds to the trajectory $\sigma(t)$ with a final time T , i.e., $\forall t \in [0, T]$, then the following definition explains the homotopic trajectories.

Definition 4.1 (Homotopic trajectories). *Two trajectories σ and σ' having the same initial and final positions, π_0 and π_f respectively, i.e., $\tau(0) = \tau'(0) = \pi_0$ and $\tau(1) = \tau'(1) = \pi_f$ are homotopic if there exists a continuous mapping $\psi : [0, 1] \times [0, 1] \rightarrow \mathcal{P}$ that can deform τ to τ' .*

The set of all trajectories that share the same initial and final positions define a class of trajectories that avoid the obstacles in the same way and they are called a *homotopy class* of trajectories.

Definition 4.2 (Homotopy class). *A homotopy class is a set of trajectories, $\Sigma_{\mathcal{H}}$, that have the same initial and final positions, and the corresponding geometric paths can be deformed to each other using a continuous mapping function.*

Finally, it is possible to say that two trajectories belong to distinct homotopy classes if the mapping from one to another intersects with obstacles (Figure 4.1b) or informally when they avoid the obstacles differently.

The role of homotopy has been studied in motion planning for a long time, early works particularly focusing on finding the shortest path in a given homotopy class [18, 26]. However, these algorithms consider only the case when a desired homotopy class is preliminarily known. Without having the topology information, even though the states that result in collisions are avoided the planners do not provide any information on how they are avoided. Furthermore, if one is interested in obtaining a set of good trajectories, the algorithm would return rather similar solutions. Therefore, some research has been done on exploring the proper topology of the environment and in particular returning a set of distinct good solutions that avoid the obstacles in a different manner (Figure 4.1). Most work falling into this class augment the path search with homotopy awareness and follow a coupled approach. In other words, the distinct homotopy classes are identified over the course of planning. The main idea is to represent the environment in an abstract form and obtain a graph representation, after which the graph is traversed to obtain a number of distinct paths in line with a homotopy class invariant. Some approaches such as Shiller et.al.'s [85] focus on generating the shortest path and initiating a new search with a topological constraint such that the consecutive shortest paths belong to different homotopy classes. Others, use Voronoi cells [5, 52] and apply a graph search to find distinct ways of reaching from an initial cell to a final one. However, these algorithms require all possible paths to be explored before returning the k-best solutions. Bhattacharya et.al. [9] proposed a method for capturing the topology information in search based planning using an arbitrary graph representation. In particular they applied A* search [73] while computing a class invariant signature based on Complex Analysis, named $\mathcal{H}$ -signature. The algorithm returns k-best paths that belong to distinct homotopy classes, which also means that the algorithm has to compute a number of short paths until a k-number of homotopically distinct ones are found. Consequently, adopting some theorems from electromagnetism in [10], they have enhanced this class invariant to apply also high dimensional configuration spaces.

In order to decouple the aforementioned problem, and to reduce the number of homotopy classes to be explored, some recent approaches focus on identifying the homotopy classes first, and then computing the best path in the identified classes. Hernandez et. al. [37] have improved the reference frame method [44] and captured the homotopy information by selecting a reference point and partitioning the search space by extending rays from this point to the centers of the obstacles. While proposing a systematic way of obtaining a set of homotopy class constraints, they have also augmented

a number of well-known path planning algorithms, namely H-RRT, H-Bug and H-A*, to plan within the given set of topological restrictions, which would correspond to k -distinct good paths at the end.

All the motion planners that are identified previously as topology-aware have considered the path planning problem or included the dynamics as interpolating the path points with polynomials (e.g. [52]). Gu et. al. [36] propose a sampling-based planner for autonomous navigation that can reason about the topology of the environment while generating dynamically feasible trajectories. In particular, they have used a similar method to [10] and defined a pseudo-homotopy that does not require trajectories to be co-terminal. The planner is then applied to discover different maneuver possibilities for an autonomous vehicle. In that sense, their work is more associated with the coupled approach of simultaneous planning and topology identification.

As it has also been highlighted in the previous chapters, most motion planners in literature are based on sampling-based approaches, owing to their performance in high dimensional spaces and an efficiency in handling obstacles. However, for state spaces that can be represented in a reduced number of states in order to capture collisions, exact approaches can also provide practical ways of solving the problem [71]. Motion planning for mobile robots and autonomous vehicles set a good example for this type, since the projection of the state space onto a 2D plane is enough to capture the collision information. There has been some research on computing the optimal trajectory based on a coarse initial guess, such as CHOMP [78], which is a trajectory optimization algorithm, robust to an initial guess. Nonetheless, since it is gradient based, it can only converge to the optimal trajectory in the homotopy class that it is initiated in due to the discontinuities introduced by the obstacles.

Following the decoupled approach of identifying the homotopy classes first and then solving for the best path, some recent work have also focused on kinodynamic planning with homotopy class constraints. These aim to enhance trajectory generation with homotopy belonging and to explicitly impose the constraints introduced by the obstacles in terms of spatial constraints. For an intelligent co-pilot, Anderson et.al. in [2], apply triangulation and obtain the topology information in graph form. The resulting problem is solved using Quadratic Programming (QP) in Model Predictive Control (MPC) framework. Park et. al. [71] achieve the homotopy classes by means of cell sequences and use *Mixed Integer Quadratic Programming* (MIQP) to compute trajectories that follow the cell sequence and avoid the obstacles. The homotopic motion planning approach presented in this chap-

ter is in line with these methods. In particular, the idea is to select a set of promising homotopy classes that can be imposed during the trajectory generation phase as a set of spatial constraints.

One of the contributions of this thesis to the existing algorithms that are in line with the approaches proposed here is the selection of homotopy classes, which allows to avoid the problems that can lead to higher cost trajectories by preliminarily selecting a number of promising ones. Furthermore, the proposed multiple phase Optimal Control Problem (OCP) formalization of the homotopic trajectory generation problem enables us to address a larger variety of dynamic systems without introducing any integer constraints, since the latter case corresponds to a more complex problem.

Finally, the algorithm proposed in the previous section is augmented with homotopy awareness. This strategy is different than the approaches that use a sampling-based planner (e.g. [37]) within the homotopy context, since it preliminarily selects a set of homotopy classes to guide the tree expansion other than exploring the whole search space and identifying different classes over the course of planning. This enables to eliminate the exploration of unpromising or dangerous zones, to parallelize the search by dividing the problem into distinct homotopy classes and to use an informed sampling based on the particular topology of the homotopy class to improve convergence.

4.2 Homotopy Aware Motion Planning Problem

Due to scope of thesis focusing on development of planning algorithms for mobile robots and autonomous vehicles, the environment that the vehicle operates in is assumed to be planar. Therefore, the following assumption is made about the dimensions of the workspace that allows us to generate a set of geometric constraints used over the course of trajectory generation.

Assumption 4.1. *Obstacles can be represented in 2D space and any information about collision and obstacle avoidance can be retrieved using the projection of the vehicle state space onto this 2D plane.*

Let $\mathcal{P} \subset \mathbb{R}^2$ define the 2D plane that the vehicle operates and $\mathcal{P}_{obs} \subset \mathbb{R}^2$ is the obstacle region. Consequently, the obstacle-free region is defined as $\mathcal{P}_{free} := \mathcal{P} \setminus \mathcal{P}_{obs}$. Let $\mu : S \rightarrow \mathbb{R}^2$ be a transformation that maps a state to its position in this 2D plane. Finally, the free state space, S_{free} , is defined as the set of states that can be mapped to $\mathcal{P}_{free}$, i.e., $S_{free} := \{s \in S \mid \mu(s) \in \mathcal{P}_{free}\}$.

The first part of the problem is dedicated to the systematic generation of distinct homotopy classes. For this purpose, the environment is partitioned and the topology information is obtained in the form of a graph. This information is then used to compute a class invariant such that, based on the geometry of the corresponding path, it should allow us to determine the homotopy class it belongs. Furthermore, such a class invariant can be imposed in the second part of the problem, while generating trajectories, to restrict the obstacle avoidance part of the trajectory.

Recalling the definitions of homotopic trajectories and homotopy class, it is possible to formalize the homotopy-aware motion planning problem. The following definition, assumes that the homotopy class the planner should return solutions in is preliminarily known. In other words, homotopy classes and their class invariant have been identified in the first part and a set of them is selected for trajectory generation. Therefore, the main goal in this part is to constrain the trajectories to avoid the obstacles in a desired manner.

Definition 4.3 (Homotopic kinodynamic planning). *The optimal kinodynamic motion planning problem with homotopy class constraints is then formalized as finding an optimal duration, $[0, T^*]$ and a feasible trajectory, $\sigma^*(t) : [0, T^*] \rightarrow S$ from a set of states and controls of a dynamic system over $[0, T^*]$, i.e., $s^*(t) : [0, T^*] \rightarrow S$ and $u^*(t) : [0, T^*] \rightarrow U$, such that the trajectory*

- *starts from an initial state $\sigma(0) = s_0$;*
- *avoids collisions with obstacles, i.e., $\sigma(t) \in S_{free} \forall t \leq T^*$;*
- *fulfills the actuation bounds, i.e., $u(t) \in U_{free} \forall t \leq T^*$;*
- *reaches the goal region, i.e., $\sigma(T^*) \in S_{goal}$, where S_{goal} is the goal region;*
- *minimizes an objective function defined as in (2.2);*
- *belongs to the desired homotopy class, i.e., $\sigma^* \in \Sigma_{\mathcal{H}}$, where $\Sigma_{\mathcal{H}}$ denote the set of all trajectories that belong to the desired homotopy class.*

Finally, the overall homotopy aware kinodynamic planning problem addressed here implies three distinct steps;

- first, given the representation of the obstacles in the 2D plane, identify distinct homotopy classes and select a number of promising ones,

- second, solve the homotopic kinodynamic planning problem given in Definition 4.3 for each one of the homotopy classes,
- finally, exploring a set of homotopy classes find the trajectory with the lowest cost.

The following sections explain the decomposition of the space and computing the corresponding homotopy class invariant. Consequently, different approaches are taken to solve the homotopy aware planning problem.

4.3 Homotopy Class Identification

Identifying distinct homotopy classes is the first step of planning with homotopy class constraints. In order to ensure that the resulting trajectory carries the class invariant characteristics of the desired homotopy class, it is necessary to determine these characteristics first. Hence, it is necessary to have the topology information and compute a class invariant that can then be used to impose specific topological constraints to the generated trajectory.

The topology of an environment can easily be defined as a union of partitions and the information of how they are connected. Thus, the first part of homotopy class identification refers to partitioning of the free space and obtaining the connectivity information. Consequently, the second part is dedicated to the finding of a specific set of partitions and how they are connected. They can serve as a class invariant to trajectories which avoid the obstacles in the same manner.

4.3.1 Partitioning of the Search Space

The basis for identifying distinct homotopy classes lies in partitioning the environment. Most methods identified before capturing the topology of the environment, such as *reference frame methods* or *Voronoi cells*, do not provide any information about the exact geometric constraints imposed by the obstacles. On the other hand, assuming polygonal obstacles or their polygonal approximations, *convex cell decomposition* gives out the exact geometric constraints.

One of the aims of this chapter is to generate a trajectory optimization technique that solves the homotopic kinodynamic motion planning problem, which corresponds to the class of exact planners in the motion planning terminology. Therefore, it is essential to have a set of spatial constraints that represent the geometry of the environment. In order to decouple the homotopy class identification and trajectory generation problems,

and to obtain the exact geometric constraints, the partitioning technique adopted in this thesis is based on *convex cell decomposition*.

Definition 4.4 (Convex cell decomposition [56]). *A convex polygonal decomposition is a finite collection of convex polygons, $c_i \in \mathcal{C}$, where each convex polygon c_i is called a **cell**, such that the union of the cells is equal to the closed interior of the obstacle free region, i.e., $\bigcup \{c_i \in \mathcal{C}\} = cl(\mathcal{P}_{free})$.*

Once the space is decomposed into convex cells, it is crucial to reason about the connectivity of this decomposition. Therefore, the following definition establishes the relation between these cells in terms of an adjacency graph.

Definition 4.5 (Adjacency graph [56]). *Two cells c and c' are connected if $c \cap c'$ is a line segment of non-zero length. The associated **adjacency graph** is formed as a non-directed graph $\mathcal{G}$ such that;*

- *Each cell corresponds to a node in $\mathcal{G}$*
- *Two nodes are connected by an edge if the corresponding cells are adjacent.*

An optimal convex decomposition of a polygonal space is identified in [65] and found to be NP-hard to compute due to the holes introduced by the obstacles. Therefore, a non-optimal -yet very efficient- method, named *trapezoidal decomposition* [19], is proposed. Nevertheless, having an optimal decomposition is not important for our case.

The idea of trapezoidal decomposition is to partition the $\mathcal{P}_{free}$ into a set of convex cells $\{c_i\}$, by extending rays from each of the vertex that define the polygonal obstacle region. Let $V_{\mathcal{P}}$ denote the set of vertices that correspond to the vertices of $\mathcal{P}_{obs}$. Then, the decomposition is accomplished by extending rays from each $v_p \in V_{\mathcal{P}}$ upward and downward until they hit $\mathcal{P}_{obs}$. The resulting cells are either trapezoids or triangles.

Once the convex decomposition is obtained, the adjacency information is stored in a graph structure (Figure 4.2). The undirected graph, $\mathcal{G} = (V_C, E_C)$ is computed as in the Definition 4.5, where V_C refers to the nodes that correspond to the cells and E_C represent the edges. Let $\mathcal{B}_{i,j}$ denote the line segment that represent the boundary between two cells, i.e., $\mathcal{B}_{i,j} = c_i \cap c_j$. Then, two nodes c_i and c_j are connected with an edge, $e = \{c_i, c_j\}$ if the corresponding boundary, $\mathcal{B}_{i,j}$, has non-zero length.

Assuming that the initial and final positions, π_0 and π_f respectively, lie strictly inside the cells but not on the boundaries, a cell for each can be

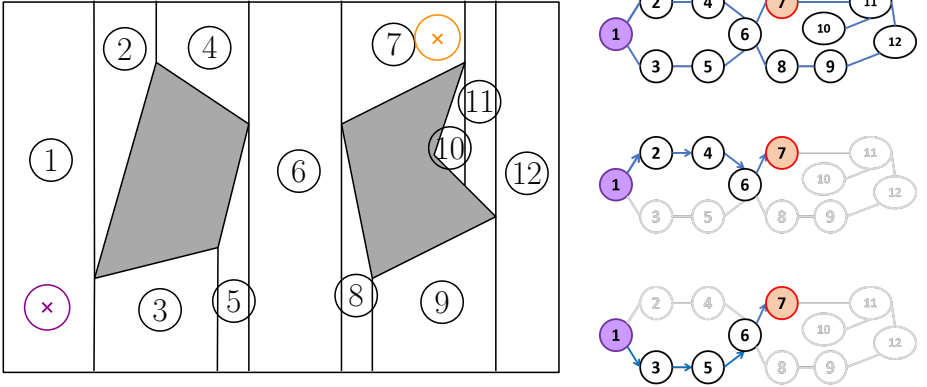


Figure 4.2: *Left: Trapezoidal decomposition and resulting convex cells, purple and orange marks are the initial and the goal positions of the robot. Right: (Top) The adjacency graph that carries the topological information about the environment, root and the end nodes are marked purple and orange respectively corresponding to the cells that contain the initial and the goal positions of the robot. (Bottom) Two example homotopy classes as cell sequences connecting the initial configuration to the goal.*

assigned as the initial and final nodes in the graph, c_0 and c_f , respectively. Once the initial and final nodes are determined, the topology information of the corresponding homotopy classes can be computed in terms of cell sequences, i.e. $\mathcal{CS} = \{c_0 \rightarrow c_1 \rightarrow \dots \rightarrow c_f\}$, using a graph search on the adjacency graph. However, searching an undirected graph results in infinite number of cell sequences from the initial node to end. Therefore, it is essential to establish a rule to limit the selected cell sequences to a certain number of promising ones.

4.3.2 Cell Sequence Selection

Once the adjacency graph, $\mathcal{G} = (V_C, E_C)$, is constructed, it should be traversed to obtain a set of cell sequences from an initial node to a final one, such that these cell-sequences can be used to solve the homotopic kinodynamic planning problem. However, before explaining the specific method applied for selecting cell sequences to generate a sequence of geometric constraints, the relation between the cell sequence and the corresponding homotopy class of trajectories should be established.

Recall that $\tau(\lambda)$ is the geometric path which corresponds to the trajectory $\sigma(t)$, furthermore denote the set of independent variables λ as $\Lambda = \{\lambda_i\}$ where $0 \leq \lambda_i \leq 1$. Then, there exist a convex cell, c_i , such that for $\lambda \in (\lambda_{i-1}, \lambda_i)$ the path $\tau(\lambda)$ is confined in c_i . Note that for $\lambda = \lambda_{i-1}$ and

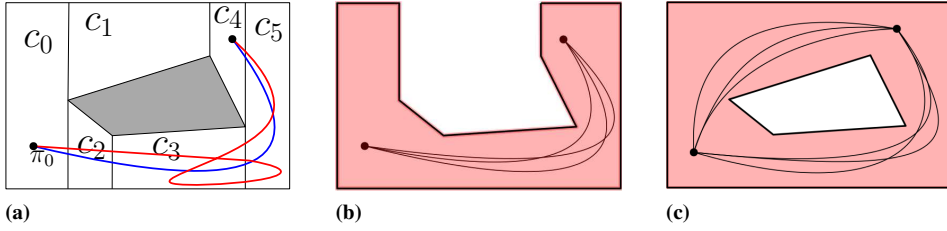


Figure 4.3: (a) Two trajectories having different cell sequences that can be reduced into the same canonical representation. (b) The polygon that corresponds to the cell sequence $\{c_0 \rightarrow c_2 \rightarrow c_3 \rightarrow c_5 \rightarrow c_4\}$. (c) The polygon that corresponds to the union of two distinct loopless cell sequences $\{c_0 \rightarrow c_2 \rightarrow c_3 \rightarrow c_5 \rightarrow c_4\}$ and $\{c_0 \rightarrow c_1 \rightarrow c_4\}$ has a hole.

$\lambda = \lambda_i$ the trajectory intersects the boundaries shared with c_{i-1} and c_{i+1} , i.e. $\tau(\lambda_{i-1}) \in \mathcal{B}_{(i-1,i)}$ and $\tau(\lambda_i) \in \mathcal{B}_{(i,i+1)}$. Therefore, similar to that, it is possible to define a sequence of cells $\mathcal{CS} = \{c_0 \rightarrow c_1 \cdots \rightarrow c_f\}$ that fully contains $\tau(\lambda)$.

Remark 4.1. Using a trapezoidal cell decomposition, each cell has two sets of boundaries, the first two boundaries are shared with the obstacles and the other two (or one if the corresponding cell is a triangle) are the boundaries with the adjacent cells. Note that if the environment is bounded, the bounding frame correspond to an obstacle. If not, one can assume that the obstacle boundaries in that case are formed at the infinity. In other words, there are no shared boundaries between cells other than the ones generated by the rays.

The canonical representation of a cell sequence is its basic representation after eliminating the repetitive subsequences. For example, for the two trajectories shown in Figure 4.3a with their corresponding cell sequences, $\mathcal{CS}_1 = \{c_0 \rightarrow c_2 \rightarrow c_3 \rightarrow c_5 \rightarrow c_4\}$ (blue) and $\mathcal{CS}_2 = \{c_0 \rightarrow c_2 \rightarrow c_3 \rightarrow c_5 \rightarrow c_3 \rightarrow c_5 \rightarrow c_4\}$ (red), the canonical representations can be computed by eliminating the repetitions. Since $\mathcal{CS}_1$ does not have any repetitive subsequences it is already in its canonical representation, while $\mathcal{CS}_2$ can be reduced to its canonical representation such that; $\mathcal{CS}_2^{can} := \{c_0 \rightarrow c_2 \rightarrow \cancel{c_3} \rightarrow \cancel{c_5} \rightarrow c_3 \rightarrow c_5 \rightarrow c_4\}$. Finally, both trajectories share the same canonical representation.

The analysis on one-to-one correspondence between loopless cell sequences and homotopy classes are also provided in [71]. Here we extend that correspondence to define a polygonal region composed of convex cells that can be used to impose the specific characteristics of a homotopy class,

rather than a specific cell sequence. In other words, we enlarge the definition to include more trajectories belonging in the same homotopy class. Assuming that the environment is decomposed using trapezoidal decomposition or any other method that generates at least two boundaries with obstacles for each convex cell, then the following proposition establishes the relationship between the polygonal region formed by the loopless cell sequence and the corresponding homotopy class.

Proposition 4.1. *The set of all trajectories with the same initial and final positions and that share the same canonical representation which corresponds to a loopless cell sequence are homotopic.*

Proof. Let $\mathcal{P}'$ denote the region defined as the union of the cells in a loopless cell sequence $\mathcal{CS}$, i.e. $\mathcal{P}' = \bigcup \{c_i \in \mathcal{CS}\}$. Denote $\mathcal{Poly}$ as the polygon obtained as the union of convex cells. Since the cell sequence does not have any loops it does not encircle any obstacles, then the interior of $\mathcal{Poly}$ does not have any holes (Figure 4.3b). This means that the space confined in the outer boundary of $\mathcal{Poly}$, i.e., the interior, is equal to $\mathcal{P}'$. By definition, it is true that $\mathcal{P}' \subset \mathcal{P}_{free}$. Therefore, the set of all the trajectories Σ' that can be projected to a set of geometric paths T' with the same initial and final positions, such that $\forall \tau' \in T'$ and $\tau'(\lambda) \in \mathcal{P}'$ for $\lambda \in [0, 1]$, then there exist a continuous mapping $\psi : [0, 1] \times [0, 1] \rightarrow \mathcal{P}'$ that can deform any $\tau'_i \in T'$ to any other $\tau'_j \in T'$. $\square$

Note that, the above proposition refers to a region that can be used as a reference for a set of trajectories belonging to the same homotopy class, but it is not restricted to the one-to-one correspondence between a trajectory and its corresponding cell sequence. This means that the set of all trajectories that can be mapped to the same polygonal area defined as the union of cells in a loopless cell sequence - in other words, the set of trajectories whose geometric path correspond to a loopless cell sequence in its canonical form - are homotopic if they share the same initial and final position.

Proposition 4.2. *Two different sets of trajectories that correspond to two different loopless cell sequences as their canonical representation represent two distinct homotopy classes.*

Proof. Similar to the proof of Proposition 4.1 let $\mathcal{P}_1$ and $\mathcal{P}_2$ denote the free space obtained as the union of convex cells in loopless cell sequences $\mathcal{CS}_1$ and $\mathcal{CS}_2$. Since two cell sequences are loopless and different, there exists at least one cell in $\mathcal{CS}_1$ that is not shared in $\mathcal{CS}_2$. Since each convex cell has

Algorithm 13: FindCellSequences

```

1 Input:  $\mathcal{G}, c_0, c_{goal}, len$ 
2  $\mathcal{BG}_{OPEN} \leftarrow [[c_0]]$ 
3  $\mathcal{BG} \leftarrow []$ 
4 while  $\mathcal{BG}_{OPEN} \neq \text{empty}$  do
5    $CS = \mathcal{BG}_{OPEN}.pop()$ 
6    $neighbor\_list = \text{explore\_neighbors}(CS[end])$ 
7   for  $\forall c \in neighbor\_list$  do
8     if  $c = c_{goal}$  then
9        $\mathcal{BG} \leftarrow \mathcal{BG}.push([CS, c])$ 
10    else
11      if  $c \notin CS$  and  $CS.length() < len$  then
12         $\mathcal{BG}_{OPEN} \leftarrow \mathcal{BG}_{OPEN}.push([CS, c])$ 
13 return  $\mathcal{BG}$ 

```

two boundaries with obstacles, the polygon $\mathcal{Poly}$ obtained as the union of convex cells in $\mathcal{CS}_1$ and $\mathcal{CS}_2$ has holes and the space contained inside the outer boundary is not continuous and is not equal to $\mathcal{P}_1 \cup \mathcal{P}_2$ (Figure 4.3c). Therefore, there does not exist a continuous mapping that can deform a trajectory with a geometric path $\tau_1 \in \mathcal{P}_1$ to another whose geometric path is $\tau_2 \in \mathcal{P}_2$. This means that two trajectories having different loopless cell sequences as their canonical representation belong to two distinct homotopy classes. $\square$

Now that the relationship between the topology of the environment and the homotopy classes are explained, it is possible to describe the method employed to select a set of promising cell sequences, such that once used for trajectory generation they result in homotopically distinct trajectories.

The graph traversal algorithms such as *Breadth First Search* (BFS) or *Depth First Search* (DFS) can be applied to traverse $\mathcal{G}$. However, virtually all graph traversal algorithms stop when they explore all the nodes of the graph or reach a designated end node. Since the goal of traversing the graph to discover homotopy classes is not just to reach the end node but to explore different ways of achieving this, it is necessary to use a modified algorithm.

The cell sequences are obtained by applying a graph traversal algorithm, FindCellSequences, as explained by the pseudocode in Algorithm 13. It adopts a DFS strategy to extract the branches of $\mathcal{G}$ (cell sequences) from an initial node to the final one, denoted as $\mathcal{BG}$, and imposes some constraints on their characteristics. In particular, the algorithm keeps a list of branches, $\mathcal{BG}_{OPEN}$, where all of them originate from the initial node, c_0 . At every

iteration the last branch in the list is selected and the neighbors of its last node are explored (line 6), this creates new branches. If a branch contains the goal cell, c_{goal} , it is moved to the list $\mathcal{BG}$ (line 9) which, at the end, corresponds to the extracted cell sequences.

In order to obtain a tractable number of cell sequences the following two restrictions are applied during the course of the graph traversal that stop the search while exploring a specific branch;

- if the resulting sequence has a duplicate of the same cell,
- if the length of the cell sequence is longer than a threshold, len .

The sequence that satisfies any of the two restrictions is avoided (line 11) and the algorithm switches exploring the neighbors of the next node in line. The first restriction corresponds to generating loopless cell sequences, since obtaining loopless cell sequences ensure that any two trajectory computed using distinct cell sequences would be as well homotopically distinct.

Selecting loopless cell sequences avoid encircling the obstacles, which is often undesirable for robotics applications. However, it does not necessarily guarantee desirable trajectories. Even though being desirable for a trajectory is problem dependent, a general rule can be applied, allowing only the cell sequences that are monotonic along the direction of motion, i.e. along the straight line connecting the initial position to the goal. This is based on the assumption that any trajectory that is projected without overlaps onto the line connecting the initial and final positions is more likely to contain the global optimal trajectory. Having overlaps on its projection over this line can be interpreted that the vehicle goes back and forth along the direction to reach the destination.

In order to achieve monotonic trajectories, the space is decomposed to obtain cell boundaries that are perpendicular to the direction of motion (Figure 4.4a-b). A similar strategy has also been adopted by Park et. al. [71]: the decomposition is carried out along a reference sweeping direction. However, using this particular decomposition does not necessarily ensure that selected loopless cell sequences correspond to a monotonic motion along the direction of motion. An example is given in Figure 4.4b-c, even though the corresponding cell sequence does not have any repetitions the corresponding trajectory is not entirely desirable and not monotonic. Therefore, we propose to keep a directed graph at the decomposition phase, so that the cell boundaries can be passed only at a particular direction, along the direction of motion. This approach would restrict selecting cell sequences that generate trajectories like in the Figure 4.4b, since the given directed graph (Figure 4.4d) would not allow selecting that particular

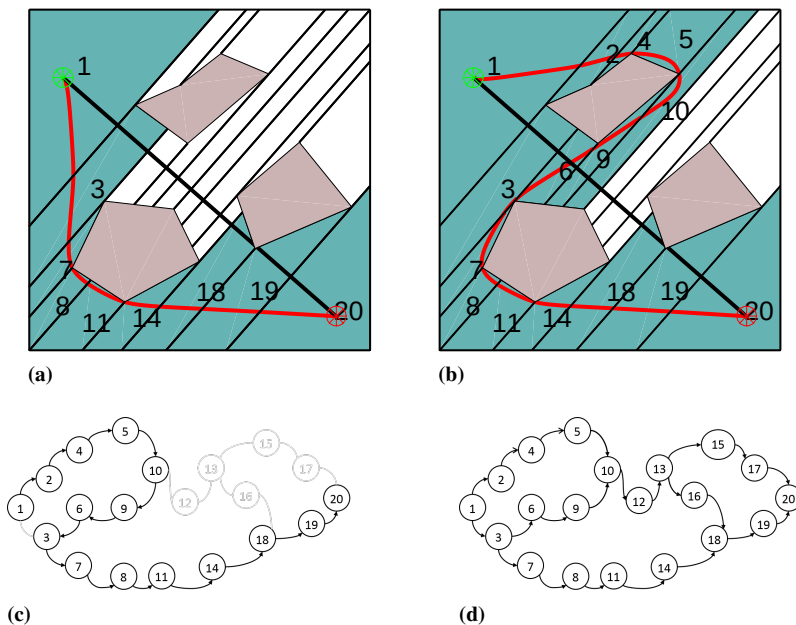


Figure 4.4: Direction of motion is from green mark to red (a) Optimal trajectory (red line) computed using a loopless cell sequence is monotonic along the direction of motion. (b) Although the cell sequence does not contain any repetitions the optimal trajectory is not monotonic along the direction of motion. (c) Cell sequence resulting in trajectory on the right when an undirected graph is used. (d) Obtaining a directed graph along the direction of motion would restrict selection of such trajectories since cell number 9 is no longer reachable from cell number 10.

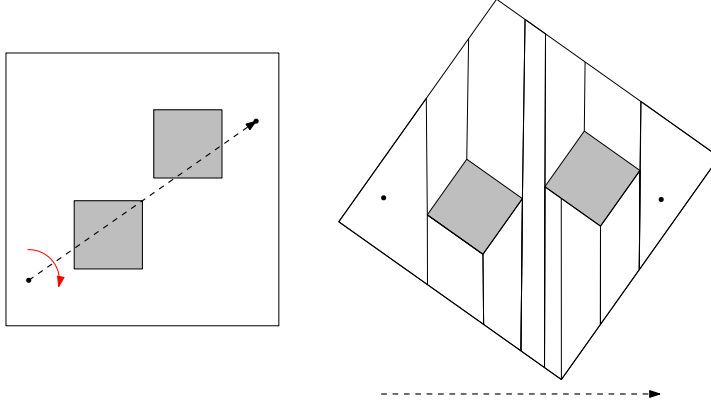


Figure 4.5: *In order to have vertical phase boundaries the workspace is rotated to align with the direction of motion (dashed line with the arrow).*

sequence. Note that, if the cell containing the final position is not accessible with the directed graph approach, e.g. behind the wall or in a closed environment, the graph builds edges to the adjacent cells until it is connected to the graph. Finally, if the preferred homotopy class is not represented in this way, one can always resort back to keeping an undirected graph.

4.4 Trajectory Generation

Once the topology information is obtained, it is necessary to develop a trajectory generation step that not only solves the motion planning problem but also takes into account the specific topological constraints. Since the first step of homotopy aware planning determines a set of cell sequences providing a set of homotopy class constraints, the second step is to develop a planner that can use these constraints to guide the trajectory to belong to a specific homotopy class.

To this end, at the first part of this section, homotopic trajectory generation is formalized as an OCP, with different phases where different kinematic constraints are active. Consequently, the topology information obtained as a union of convex cells is used to augment the sampling based planner RRT^*_{MP} with homotopy awareness.

4.4.1 Exact Optimal Planner with Homotopy Class Constraints

Benefiting from the geometric constraints arising from the cell decomposition, the homotopy aware planning can also be posed as an OCP in a multiple phase fashion. The idea is to define each convex cell as a different

phase of the problem so that the shared boundary between two cells trigger phase transition and the boundaries between obstacles can be written as inequality constraints on state variables. Therefore, a homotopy class of N_c consecutive cells can be formulated as an OCP of N_c phases.

In order to simplify the problem, in particular to simplify the phase transitions, the decomposition is applied ensuring that the rays extended from the vertices are perpendicular to the x-axis. For that purpose and for the sake of obtaining a monotonic decomposition along the direction of motion, the environment is rotated about the initial position π_0 to obtain a direction of motion parallel to the x-axis (Figure 4.5). Furthermore, in order to avoid having self-loops along the trajectory, the phases are designed to have a one to one correspondence with the selected loopless cell sequence.

Finally, the trajectory optimization problem with homotopy class constraints is formalized as follows. The independent variable, the time t , is defined in the region $t_0^{(p)} \leq t \leq t_f^{(p)}$ for phase p , and the phases are sequential with respect to t such that $t_f^{(p)} = t_0^{(p+1)}$. Each phase is subject to differential constraints (2.1) describing the dynamics of the system

$$\dot{\mathbf{s}}^{(p)} = f\left(\mathbf{s}(t)^{(p)}, \mathbf{u}(t)^{(p)}\right) \quad (4.1)$$

There are bounds on the state variables

$$\mathbf{s}_{min}^{(p)} \leq \mathbf{s}(t)^{(p)} \leq \mathbf{s}_{max}^{(p)} \quad (4.2)$$

and on control variables

$$\mathbf{u}_{min}^{(p)} \leq \mathbf{u}(t)^{(p)} \leq \mathbf{u}_{max}^{(p)} \quad (4.3)$$

The initial state of the first phase is completely fixed and the final state of the final phase can be either partially or fully specified depending on the definition of S_{goal} . Therefore,

$$\mathbf{s}(t_0)^{(1)} = \mathbf{s}_0, \mathbf{s}(t_f)^{(N_c)} = \mathbf{s}_f \quad (4.4)$$

where $\mathbf{s}(t_0)^{(1)}$ represents the state variables at the beginning of the first phase and $\mathbf{s}(t_f)^{(N)}$ are at the end of the final phase ($p = N$). Furthermore, in order to obtain a continuous trajectory through phases, the following constraint is enforced at the phase boundaries

$$\mathbf{s}(t_f)^{(p)} = \mathbf{s}(t_0)^{(p+1)} \quad (4.5)$$

Partial terminal conditions for each phase, apart from the final one, are defined as

$$\pi(t_f)^{(p)} \in \mathcal{B}_{p,p+1} \quad (4.6)$$

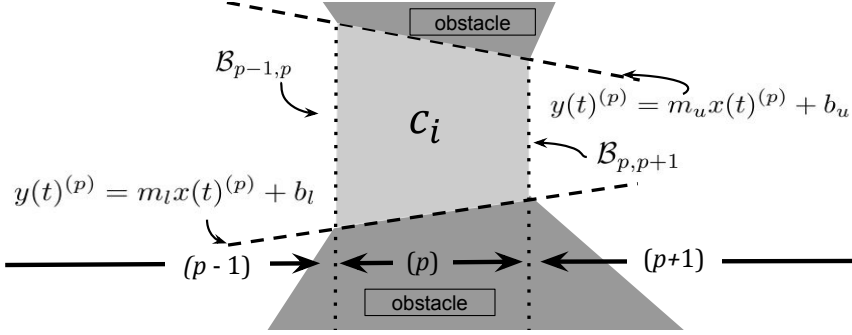


Figure 4.6: Since each cell c_i is a convex polygon the collision avoidance can be obtained as a union of half-planes perpendicular to the obstacle edge that is active during the corresponding phase.

ensuring that the state variables corresponding to the position, $\mathbf{s} = [\pi, \dots]$ and $\pi(t_f)^{(p)} = [x, y]^T$, at the end of each phase lie strictly on the line segment $\mathcal{B}_{p,p+1}$ representing the shared boundary between two consecutive phases. Constraints (4.5) and (4.6) together ensure that cell sequence defining the homotopy class is respected.

The trapezoidal decomposition provides trapezoidal cells with two sets of boundaries, one set includes the boundaries $\{\mathcal{B}_{(p,p+1)}, \mathcal{B}_{(p,p-1)}\}$ for phase transition and the other set the boundaries shared with the obstacles. These boundaries can be expressed as linear inequalities (Figure 4.6) for collision avoidance as follows

$$y(t)^{(p)} \leq m_u^{(p)} x(t)^{(p)} + b_u \quad (4.7)$$

$$-y(t)^{(p)} \leq -m_l^{(p)} x(t)^{(p)} - b_l \quad (4.8)$$

where m_u , m_l and b_u , b_l represent the slope-intercept parameters of the lines created by the upper and lower boundaries between the cell and the obstacles correspondingly. These two inequality constraints combined with (4.2), in particular the bound on the x component of the position, confine the trajectory inside the corresponding cell along the cell sequence.

Aforementioned OCP can be transformed into a Nonlinear Programming Problem (NLP) by means of transcription methods. The idea is to discretize the otherwise continuous time problem and obtain a set of decision variables. The differential constraints (4.1) are transformed into equality constraints.

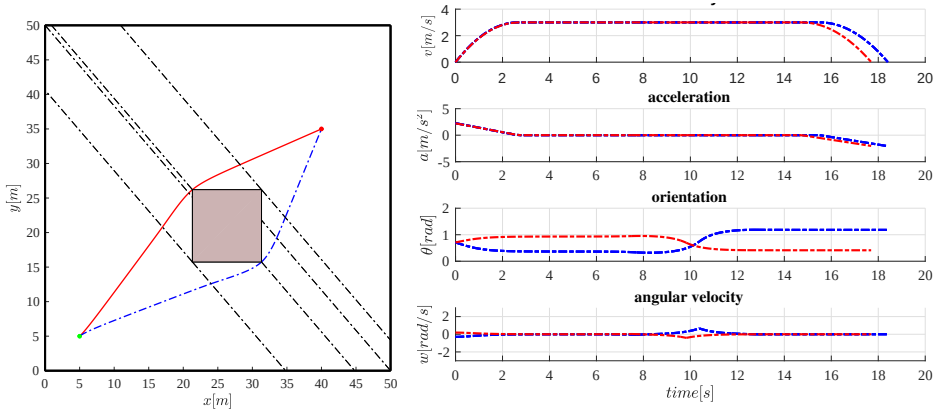


Figure 4.7: Left: Distinct local optimal trajectories from the initial position (green) to a final (red), best cost trajectory is shown in red. Right: Corresponding velocity, orientation and actuation profiles.

4.4.2 Numerical Examples using the Exact Approach

The algorithm is tested by a MATLAB implementation using the equations of motion of a unicycle like robot 3.23. For this numerical example a minimum-time, minimum-energy strategy is used to compute a cost metric for optimization and, the optimal trajectory is obtained by minimizing the following objective function over all the phases

$$J(u) = \int_{T_0}^{T_f} [1 + u(t)^T R u(t)] dt \quad (4.9)$$

where T_0 and T_f are the initial and the final times for the overall problem. The control variables a and w are equally penalized using $R = 0.5I_2$ and bounded as $a \in [-5, 5](m/s^2)$, $w \in [-3, 3](rad/s)$. Finally there is a limit on the velocity, such that $v \in [0, 3](m/s)$.

Continuous time optimal control problem with multiple phases is transformed into a NLP, which is then solved using the MATLAB toolbox GPOPS [72] which uses a *Legendre-Gauss-Radau orthogonal collocation method* to describe the equality constraints imposed by the system dynamics.

Figure 4.7a shows a simple example for a situation that two different avoidance strategies exist for a single obstacle. Homotopy classes are identified by a decomposition of the space along the direction of motion, i.e. straight line connecting initial and final positions, and two loopless se-

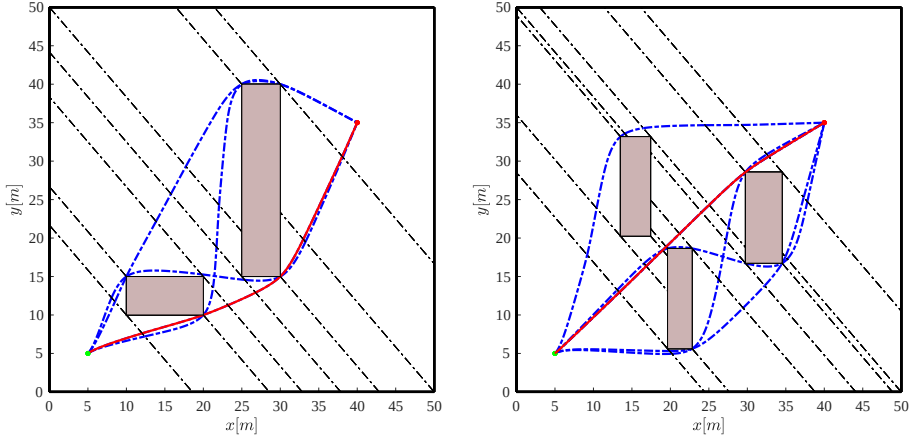


Figure 4.8: Trajectories that belong to distinct homotopy classes for different number of obstacles, lowest cost is represented in red.

quences are selected. Figure 4.7b reports the actuation and velocity profiles for both cases, which clearly demonstrates that the limits are never exceeded neither for control variables nor for velocity.

Table 4.1: Resulting loopless cell sequences and the costs of the corresponding trajectories for the scenario given in Figure 4.9

Graph type	Cell sequence	Cost
Directed	{0, 2, 3, 5, 7, 9, 12, 15, 17, 19, 20, 22}	38.05
	{0, 2, 3, 5, 7, 9, 12, 15, 17, 18, 21, 22}	57.50
	{0, 2, 3, 5, 7, 8, 14, 16, 17, 19, 20, 22}	21.88
	{0, 2, 3, 5, 7, 8, 14, 16, 17, 18, 21, 22}	29.83
	{0, 1, 6, 10, 11, 13, 14, 16, 17, 19, 20, 22}	37.14
	{0, 1, 6, 10, 11, 13, 14, 16, 17, 18, 21, 22}	37.22
	{0, 2, 3, 4, 10, 11, 13, 14, 16, 17, 19, 20, 22}	26.01
	{0, 2, 3, 4, 10, 11, 13, 14, 16, 17, 18, 21, 22}	28.71
unDirected	{0, 1, 6, 10, 11, 13, 14, 8, 7, 9, 12, 15, 17, 19, 20, 22}	86.89
	{0, 2, 3, 4, 10, 11, 13, 14, 8, 7, 9, 12, 15, 17, 19, 20, 22}	106.2
	{0, 2, 3, 4, 10, 11, 13, 14, 8, 7, 9, 12, 15, 17, 18, 21, 22}	95.24

Trajectories generated using loopless cell sequences for different number of obstacles are shown in Figure 4.8. Clearly, all of them belong to distinct classes; therefore, they represent the optimal solution to distinct sub-problems and they are monotonous along the direction of motion. Trajectories marked with red correspond to the lowest cost trajectory for the selected objective function. However, depending on the scope of the prob-

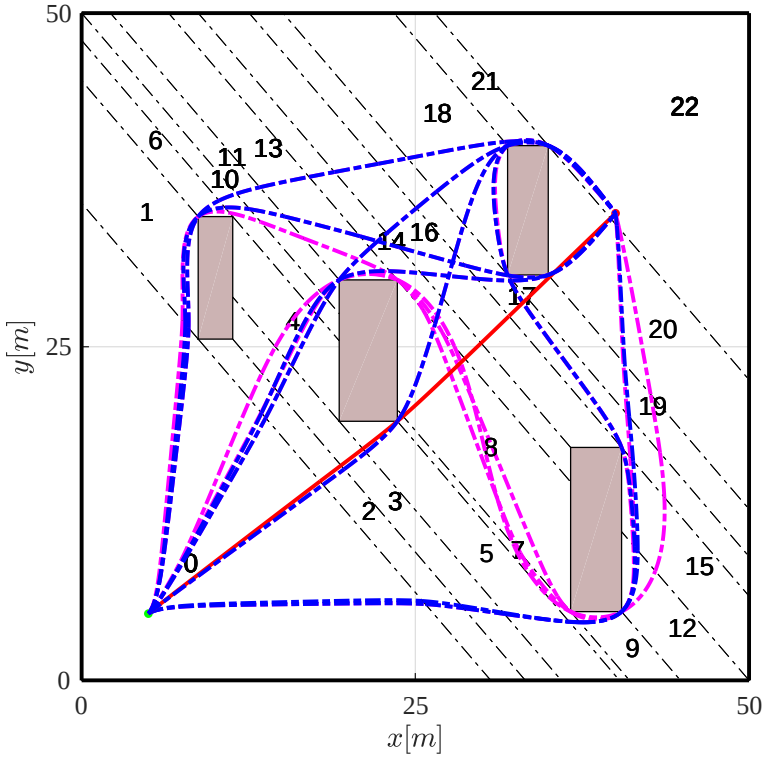


Figure 4.9: Trajectories computed using the directed and undirected graphs, magenta trajectories correspond to the undirected graph that are not solved during the computation, and the lowest cost trajectory is represented in red.

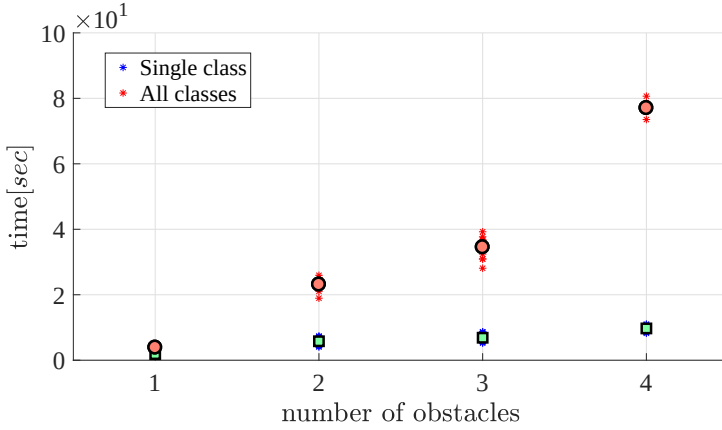


Figure 4.10: For different number of obstacles: Computation time of generating a trajectory for a sub-problem (green square corresponds to the average computation time, while blue stars are the computation times for different simulation environments.)

lem, different navigation decisions, that may or may not correspond to the best cost trajectory, can be made. Figure 4.9 shows the local optimal trajectories in terms of the identified homotopy classes using the undirected and directed graph along the direction of motion. The trajectories belonging to loopless cell sequences have considerably higher costs with respect to keeping a directed graph (Table 4.1). Additionally, preliminarily eliminated trajectories have unnecessary detours before reaching the goal, which is often not desirable in commonly used motion planning objectives. Finally, the global optimal trajectory is contained in the homotopy classes identified by the directed graph approach.

Here, 40 different environments in total have been created to contain different number of randomly generated rectangular obstacles in various sizes. Simulations are carried out on an IntelCore i7 @2.40 GHz personal computer with 8GB RAM. The shortest path in the polygonal environment corresponding to the proper homotopy class is used as an initial guess. Figure 4.10 shows the computation times of trajectory optimization problem for different number of obstacles including the decomposition. The length of time needed to solve a single problem (generating a trajectory in the designated homotopy class) is also reported, since the trajectory optimization problem for each homotopy class is independent and, hence, parallelizable.

Finally, Table 4.2 shows the average number of phases for each sub-problem for different number of obstacles. As expected, higher number

Table 4.2: For different number of obstacles: Number of phases (length of the corresponding loopless cell sequence on the undirected graph) that correspond to different homotopy classes and number of homotopy classes.

Number of obstacles	Number of phases		Number of homotopy classes
1	5		2
2	8		4
	min	max	
3	8	10	5
4	12	13	8

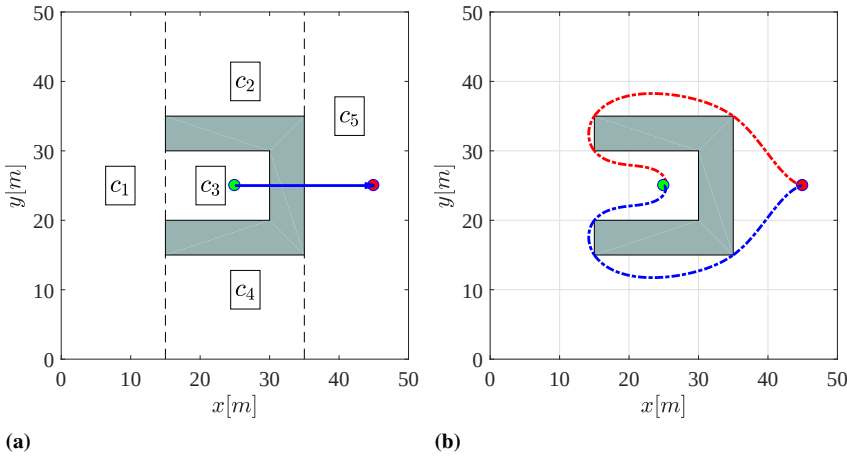


Figure 4.11: (a) Decomposition of the workspace and the direction of motion from the start to the goal. (b) Solution trajectories that belong to different homotopy classes.

of obstacles increase the partitioning of the space, as a result, increasing the number of phases for each sub-problem. This, in return, increases the length of time needed to solve each sub-problem. Furthermore, as the number of obstacles increases so does the number of homotopy classes, i.e., the number of sub-problems to be solved. However, once a trajectory is computed from an initial state to the goal it can be used as an initial guess for further replanning queries, which reduces the computational time needed to generate a trajectory. Notably, the method explained here is designed mainly for structured environments and in response to emerging obstacles that can be approximated by polygons, consequently, this approach is not scalable for complex environments that result in high levels of partitioning.

The choice of directed graph is a heuristic for focusing first on the

promising problems, but not a restriction on the overall problem. Indeed, if the problem does not admit a monotonic trajectory from an initial state to the goal, the graph representation of the topology will not include a path to the goal cell if a directed graph is used. For example, when the vehicle needs to get out of a U-shaped environment the search using a directed graph along the direction of motion will contain only the initial cell, c_3 , but not a cell sequence to the goal (Figure 4.11a). In that case, the directed graph is augmented by progressively adding edges to the goal cell from the adjacent ones until the goal cell is connected to the initial one. In this case, the sequences $\mathcal{CS}_1 = \{c_3 \rightarrow c_2 \rightarrow c_1 \rightarrow c_3\}$ and $\mathcal{CS}_2 = \{c_3 \rightarrow c_4 \rightarrow c_1 \rightarrow c_3\}$ can be discovered and the resulting trajectories can be generated (Figure 4.11b). Note that, as long as there exists a trajectory connecting the initial and final states, the corresponding cell sequence will be represented in the decomposition, and therefore will be discovered by the graph search.

4.4.3 Sampling Based Planner with Homotopy Class Constraints

Cell decomposition provides a systematic generation geometric constraints that can be used to constrain the evolution of a trajectory topologically. As a result, these constraints can also be embedded into the framework of RRT^*_{MP} , providing a homotopy aware motion planning such that any trajectory returned by the algorithm will avoid the obstacles in the desired manner. Other than constraining the search within a selected homotopy class or obtaining distinct local (resolution) optimal trajectories, homotopy awareness can also be used to parallelize the overall search where each problem focuses on different homotopy class.

Homotopic RRT^* with Motion primitives

Homotopic RRT^*_{MP} is a constrained version of the algorithm proposed in Chapter 3. Its function is similar to the original version but permits a growth only in the specific homotopy class. Using an exact decomposition, any homotopy class constraint given as a cell sequence corresponds to a subset of the free space, i.e. $\mathcal{P}' \subset \mathcal{P}_{\text{free}}$, where $\mathcal{P}' = \bigcup \{c_i \in \mathcal{CS}\}$.

Remark 4.2. Following from the Proposition 4.1, it can be concluded that any two trajectories having the same initial and final positions, that can be projected onto the same polygonal environment created as a union of loop-less sequence of convex cells are homotopic, even though their respective geometric paths correspond to cell sequences with loops.

Algorithm 14: `Sample()` for goal biased sampling

```

1 Input:  $S_{goal}^\Delta, S'_{free}^\Delta, r_g$ 
2  $r \leftarrow \text{Rand}(0, 1)$ 
3 if  $r < r_g$  then
4    $\lfloor \text{Sample}(S_{goal}^\Delta)$ 
5 else
6    $\lfloor \text{Sample}(S'_{free}^\Delta)$ 
7 return  $s_{rand}$ 

```

This allows us to define the homotopy class constraints as a feasible region, without explicitly tracking the corresponding sequence of cells passed for each node to be added to the tree. Any trajectory that can be projected onto this free space will be homotopic to the others as long as they share the same initial and final positions since they share the same canonical cell sequence. Therefore, the RRT*_MP is modified to ensure that the tree built by the algorithm lies within this feasible region. Given a loopless cell sequence $\mathcal{CS} = \{c_0 \rightarrow c_1 \rightarrow \dots \rightarrow c_f\}$, and the corresponding polygonal region, i.e. $\mathcal{P}' = \bigcup \{c_i \in \mathcal{CS}\}$, the definition of the free state space is defined for each cell sequence as;

$$\mathcal{S}'_{free} = \{s \in \mathcal{S} : \mu(s) \in \mathcal{P}'\}.$$

Recall that, μ is a function that maps a state to its position in this 2D plane. To ensure that the edges used to build the tree can contribute to obtaining a trajectory in the desired homotopy class, the `isCollisionFree` procedure checks if a tentative trajectory lies in this new definition of the free state space. Consequently, the discrete free state space is defined as

$$\mathcal{S}'_{free}^\Delta = \{s \in \mathcal{S}^\Delta \wedge s \in \mathcal{S}'_{free}\}$$

where $\mathcal{S}^\Delta$ is the discrete state space obtained by gridding the $\mathcal{S}$.

The sampling procedure is updated to sample from this feasible region, i.e., $\text{Sample}(\mathcal{S}'_{free}^\Delta)$. Since the homotopy between trajectories require that the end points of their corresponding geometric paths are the same, the goal region should explicitly refer to a specific position. However, it is important to note that the homotopy between two trajectories does not interest other states, in other words they can take any value within the problem bounds. Hence, the definition of a goal region still stands. In order to improve the convergence of RRT*_MP in this case, it is augmented with a goal-biased sampling strategy similar to [58] where the corresponding `Sample()` procedure is given in Algorithm 14. The idea is to switch between drawing

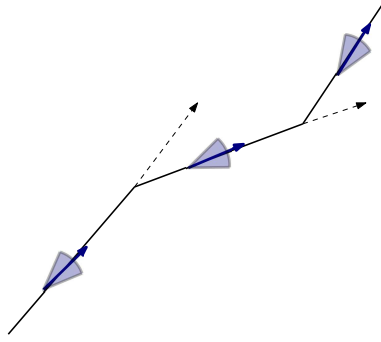


Figure 4.12: Flow of the shortest geometric path. Solid black line represents the shortest path found on the visibility graph, and the blue solid angles are the sampling range allowed for orientation.

samples from the discrete free state space corresponding to the particular homotopy class and from the goal region (if a random variable, r , is below a threshold, r_g , as given in Line 3).

Shortest geometric path for informed sampling

The exact decomposition of the free space is also useful for making informed decisions during the sampling phase. In particular, the shortest geometric path can be used to sample in a restricted promising region. In order to avoid confusions, in this section, the word shortest geometric path is reserved to the shortest path connecting the initial position to the final one in the collision-free space, while ignoring the system dynamics. Therefore, `RRT*_MP` is used to impose the full dynamics with a biased sampling strategy.

The idea of using the topology information for enhancing the sampling is based on the observations that are made in the previous section, such that the local optimal trajectories computed by solving the OCP problem are in the vicinity of this shortest geometric path. Considering the general objectives that a motion planning algorithm accounts for, such as minimum time, minimum distance or minimum energy expenditure, most of them correspond to generating smooth paths¹ as short as possible as the system dynamics permits. Hence, it could be reasonable to assume that the (local) optimal trajectory would lie in the vicinity of this shortest geometric path.

In order to use an informed sampling approach based on the shortest path in a polygon, the shortest path that connects the initial position to a

¹In order to reason about the shortest path in a polygon, the word path is selected on purpose, that refers to the geometric path corresponding to the computed trajectory.

final one inside the polygonal region, $\mathcal{P}'$, which corresponds to the desired cell sequence $\mathcal{CS}$, is computed. To this end, the visibility graph containing the line segments that are confined inside the designated polygonal zone is traversed using the A* algorithm [73].

Once the shortest path is found, it is possible to define a sampling set that restricts the states to be sampled to the ones that are in the vicinity of this path. Note that, no other restrictions are considered for the remaining states, e.g. on the velocity components or on the higher order derivatives. This results in a more focused set, that contains promising samples which can contribute to finding trajectories that solve the motion planning problem. In particular, a sampleable zone is designed such that the distance between the position component of any drawn sample and the shortest geometric path is not larger than a threshold², and the orientation of a sample is bounded by the limits determined by the *flow* of this shortest path (Figure 4.12). For the samples that are in the vicinity of a line segment the orientation bounds are determined such that, $\theta \in (\theta_i^i - \theta_{th}, \theta_i^i + \theta_{th})$, where θ is the orientation component of the state to be sampled, θ_i^i is the orientation of the i^{th} line segment which is a part of the shortest geometric path and finally θ_{th} is a threshold to establish a cone of orientations. Furthermore, for the circular zone around the points that correspond to the intersection of two consecutive line segments, characterized by the radius equal to the distance threshold, the bounds are computed using the minimum and the maximum among two limits determined by the two consecutive lines. The same is also performed for the initial and the final positions, using the initial and final states and the respective line segments.

Using the information generated by computing the shortest geometric path in order to improve sampling is also recently proposed by Palmieri et. al. [69], named Theta*-RRT. It first computes the shortest geometric path in a 2D environment using the Theta* algorithm [67], consequently, uses this path to bias the sampling applied by RRT while searching the full state space for kinodynamic motion planning. The concept of *flow* explained here is similar to the path-biasing proposed in [69], however they apply a more advanced approach for computing the angle bias weighted by a function of its projection on the line segment. On the other hand, here, the concept of *flow* is used only to determine upper and lower bounds on possible orientations. The main difference of our approach compared to Theta*-RRT is exploring all of the promising homotopy classes. In other words, instead of finding the shortest collision-free geometric path, we compute homotopically distinct shortest paths that can further be used to generate

²The distance is computed using Euclidean norms.

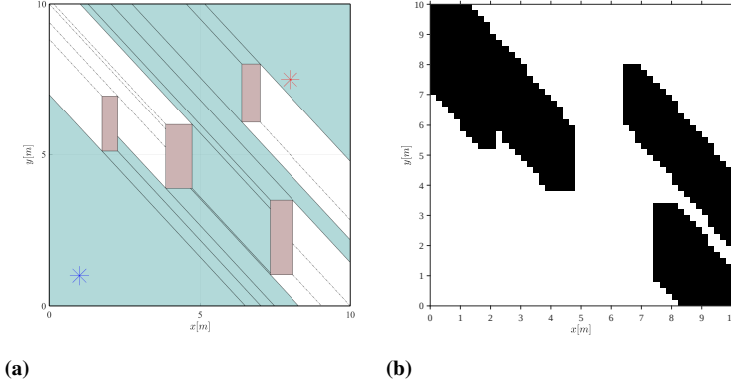


Figure 4.13: Feasible region for homotopy aware RRT*_MP. (a) Selected homotopy class, highlighted with blue, that provides a sequence of cells from the initial cell (containing blue star) to a final one (containing red star). (b) Corresponding feasible region that correspond to the cell sequence, everywhere else is assigned as an obstacle.

distinct trajectories. Therefore, if a lower cost trajectory exists in a class which does not correspond to the shortest geometric path, this lower cost trajectory can still be generated. On the other hand, the trajectory generated by Theta*-RRT is destined to be in the same homotopy class of the path used for biasing the sampling.

Once the shortest path and the corresponding flow is determined, the procedure $\text{Sample}(\tilde{S}^\Delta)$ for informed sampling returns samples from a subset of the free state space that corresponds to the desired homotopy class, i.e. $\tilde{S}^\Delta \subset \mathcal{S}_{free}^\Delta$ that implies the restrictions imposed by the shortest path.

4.4.4 Numerical Example using Sampling Based Approach

Numerical examples are carried out using the same motion primitives, in particular using the fine resolution square gridding, specified in Section 3.6.2 for a $10 \times 10[m^2]$ environment populated with 4 obstacles. The algorithm is constrained to return a solution that follows a selected homotopy class. Once the desired homotopy class is given in terms of a cell sequence from an initial cell to a final one, the algorithm starts computing the free region (Figure 4.13a-b)

Figure 4.14 shows the trajectories computed after 10^4 number of iterations for the homotopy class given in Figure 4.13a, using different sampling techniques. Clearly, the homotopy class restrictions are respected since the corresponding occupancy grid for collision-detection assigns every cell

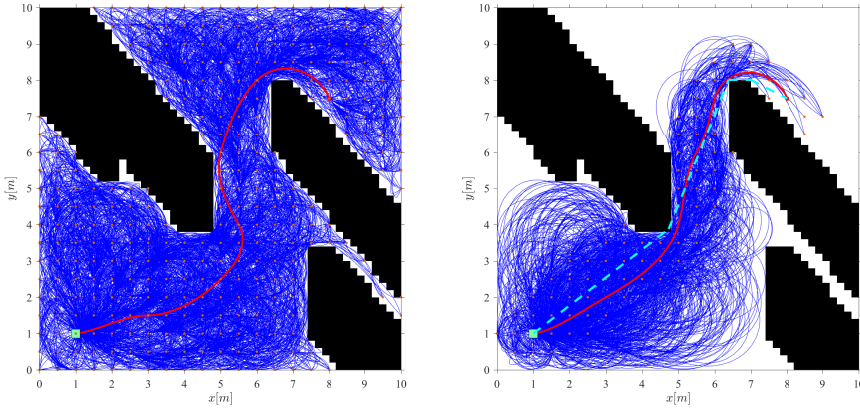


Figure 4.14: Different sampling strategies for the same homotopy class for the same number of iterations (10^4). Orange dots represent the nodes (states) explored and added to the tree. Left: Sampling from the free state space from the corresponding homotopy class. Right: Using the shortest homotopic path (cyan dashed line) for informed sampling.

other than those corresponding to the desired cell sequence as an obstacle, it ensures that there is only one way to avoid the obstacles. Furthermore, even though there are no restrictions on how the cells are passed by the trajectory as long as they are contained in the corresponding canonical sequence, the trajectory returned by the algorithms correspond to a loopless sequence.

Using the shortest path in the polygon representing the corresponding homotopy class allows us to explore a more restricted area focusing on the samples that can contribute to finding a trajectory that solves the homotopic trajectory planning problem. Figure 4.15 demonstrates the cost evolution as a function of iterations for two different approaches. In addition, the cost of the optimal trajectory for the same homotopy class is also computed using the exact method explained in Section 4.4.1 and it is provided for comparison. It is clear that, using an informed sampling that accounts for the *flow* of the shortest path provides better solutions faster compared to sampling the complete free region, which is proven to be resolution optimal.

Since the informed sampling strategy uses a smaller sampling set, the generated tree has a lower cardinality than the counterpart (Figure 4.16). However, this does not necessarily mean that the generated tree is sparse. Instead, for the early steps of the algorithm, using an informed sampling generates a denser tree since the search is focused on a specific area. This

can be seen also from Figure 4.17, comparing the computation times for both approaches. Using an informed sampling takes more time at the beginning due to the higher number of neighbors to check for connection. However, this search eventually leads to a lower cost trajectory in a shorter time, meaning that, if left to run for the same length of time, an informed sampling would return a trajectory with a lower cost other than sampling the complete free space.

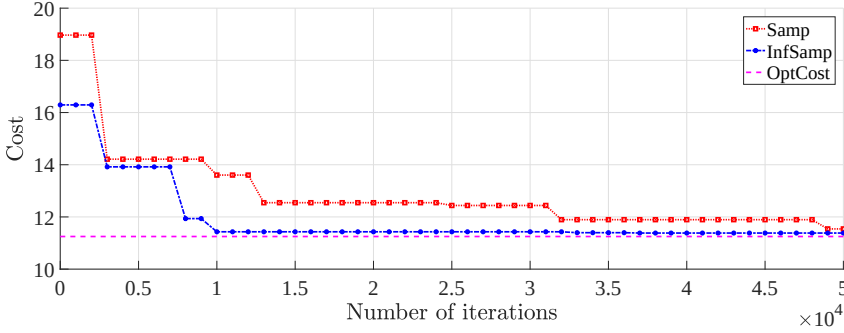


Figure 4.15: Average cost of the best cost trajectory as a function of number of iterations for different sampling strategies with reference to the cost of the optimal trajectory computed using the exact approach (pink line).

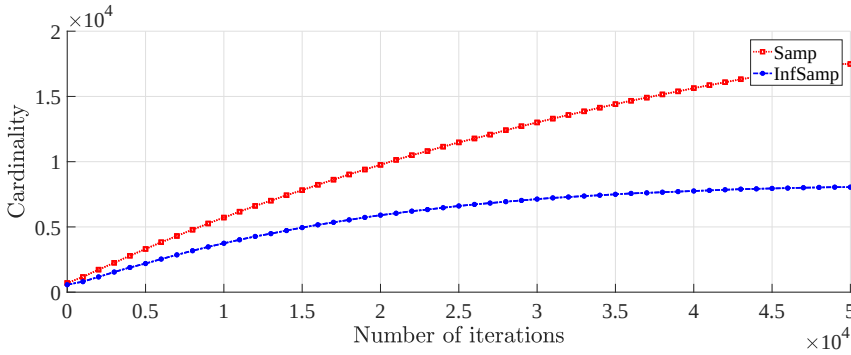


Figure 4.16: Average tree cardinality as a function of number of iterations for different sampling strategies.

4.5 Conclusion

This chapter discusses the effects of topological constraints on motion planning. In particular, it elaborates on the contributions of this thesis over the framework of a decoupled approach of homotopy class identification and

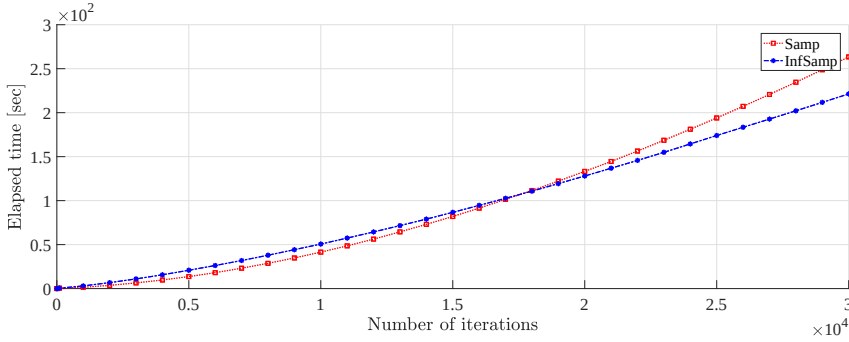


Figure 4.17: Computation time as a function of number of iterations for different sampling strategies.

trajectory generation. In the first part, the relation between the polygonal areas formed by a loopless sequence of convex cells and the homotopy classes was established. Consequently, a heuristic was proposed to select a number of promising homotopy class constraints which lead to selecting a number of sub-problems to solve. The second part is focused on trajectory generation with homotopy class constraints. In particular, an exact optimal motion planner based on homotopy class constraints was proposed that generates trajectories solving the multiple phase OCP. Lastly, RRT*_MP was augmented with homotopy awareness and a sampling strategy was designed to improve the convergence to the resolution optimal solution within the given homotopy class.

CHAPTER 5

Landmark-Aware Motion Planning

Another factor to be considered for a planner comes with the constraints imposed by the sensors. Especially for robots operating in GPS denied environments, such as indoors or underwater, it is essential to maintain connection with a landmark in a known position, be it a signal emitting beacon, an artificial or a natural visual cue. This allows the vehicle to localize itself over the course of the navigation. However, as expected, these landmarks can only be perceived by autonomous vehicles from a limited range. Therefore, it is planner's duty to ensure that the resulting trajectory passes within the perception range of these landmarks.

As well as the localization based on landmarks, one can also consider these range constraints as signal emitting bases. For a number of common tasks accomplished by autonomous vehicles and mobile robots, such as surveillance or search and rescue, it is important to maintain connection with a signal emitting base in order to transmit acquired data or to receive tactical information.

These broad and interesting application areas have motivated the interest to extend the framework of the planners described in previous chapters to consider the range-based connectivity constraints imposed by the landmarks. This section, after providing a definition of landmarks, explains the

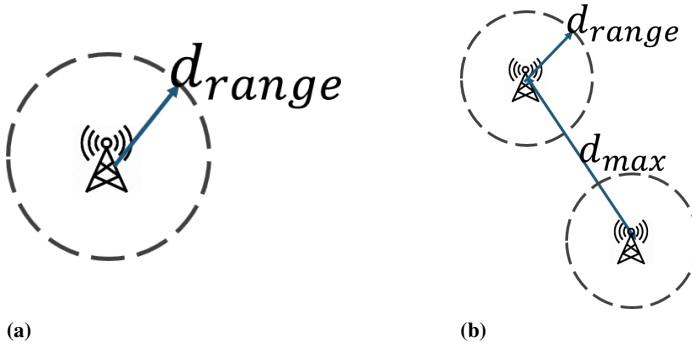


Figure 5.1: (a) A landmark with an emitting radius of d_{range} . (b) The maximum allowed distance, d_{max} , between two consecutive landmarks.

proposed extensions to the motion planning approaches in order to address the landmark-aware motion planning problem.

5.1 Landmark-Aware Motion Planning Problem

In this thesis, the landmarks considered are points that have a certain radius of perception (Figure 5.1a). In other words, if the robot is within a certain radius from the center of the landmark, it is considered to be connected to that particular landmark. These type of constraints are introduced by signal emitting beacons, robots equipped with Software Defined Radios (SDR) or signal emitting bases with an emission range. Therefore, it is worth reminding that the problem addressed here does not consider any visibility constraints introduced by cameras with limited Field of View (FOV). However, it can be generalized to account for visual cues; for example, using 360° cameras or using limited FOV cameras with their camera axis perpendicular to the ceiling or to the floor.

Landmark-aware motion planning have been explored in this manner in a few works. To illustrate, Babel [4] obtained a landmark-aware planner for an Unmanned Aerial Vehicle for GPS denied environments based on natural visual cues. The idea is to assign a cost to each landmark, depending on the orientation that they are passed, where the cost is calculated depending on the prospective landmarks towards the heading. This problem is then solved by traversing the graph and finding a set of landmarks from the start to the goal, minimizing a cost. For a home service robot, Zhang et al. [94] use Quick Response (QR) codes deployed on the floor, where the global path planner ensures that the robot passes from as many

landmarks as possible to reduce the motor encoder error. For both of the approaches, the landmarks are selected as waypoints and the second step of the algorithm adapts a dynamically feasible and smooth path interpolating these landmarks by using polynomials or splines.

On the other hand, the approaches proposed in this chapter, that extend the planners introduced in the previous chapters, consider the landmarks as range constraints rather than explicit waypoints. Hence, the problem is posed as kinodynamic motion planning problem with connectivity constraints. In other words, the algorithm is free to select any set of states, that solves the problem ensuring that the connectivity constraint is satisfied at each state .

Let the set $\mathcal{L} = \{L_i\}$ represent the centers of N_l number of landmarks available in the environment¹, where $i \in \mathbb{N}$ and $0 \leq i \leq N_l$. Then the robot is connected to a particular landmark, L_i , at time $t = T$ if the following condition is true for the position component of the state

$$\|\pi(T) - L_i\| \leq d_{range} \quad (5.1)$$

where $\pi(t)$ represents the position of the robot and d_{range} is the radius of the landmark. It is assumed that the landmarks are identical in terms of their perception/emission range. Note that, the robot can be connected to more than one landmark at a time instant.

The following definition formalizes the kinodynamic motion planning problem with landmark constraints, i.e. connectivity constraints.

Definition 5.1 (Landmark aware motion planning). *The optimal kinodynamic motion planning problem with connectivity constraints is formalized as finding a feasible trajectory $\sigma^*(t) : [0, T^*] \rightarrow \mathcal{S}$ from a set of states and controls of a dynamic system and a duration $[0, T^*]$, i.e., $s^*(t) : [0, T^*] \rightarrow \mathcal{S}$ and $u^*(t) : [0, T^*] \rightarrow \mathcal{U}$, such that the trajectory*

- *starts from an initial state $\sigma(0) = s_0$;*
- *avoids collisions with obstacles, i.e., $\sigma(t) \in \mathcal{S}_{free} \forall t \leq T^*$;*
- *fulfills the actuation bounds, i.e., $u(t) \in \mathcal{U}_{free} \forall t \leq T^*$;*
- *reaches the goal region, i.e., $\sigma(T^*) \in \mathcal{S}_{goal}$;*
- *minimizes an objective function defined as in (2.2);*

and any state along the trajectory

¹Note that, the Assumption 4.1 about the dimensions of the workspace, such that the robot operates on a 2D plane, is also made for this chapter.

- is connected with at least one landmark in the set $\mathcal{L}$, $\forall t \leq T^*$, i.e.,

$$\min_{L_i \in \mathcal{L}} \{ \|\pi(t) - L_i\| \} \leq d_{range} \quad (5.2)$$

where $\pi(t)$ corresponds to the position component of a state along the trajectory.

A second type of problem occurs when the vehicle can move without the coverage of a landmark as long as a maximum allowed travel distance between two landmarks is not exceeded. In other words, a vehicle can travel without being connected if it will end up in the connectivity radius of a consecutive landmark, and if the traveled distance between these two is not longer than a threshold.

Definition 5.2 (Landmark sequence). *For a given set of landmarks, $\mathcal{L}$, that populate the environment, let $\mathcal{LS} = \{L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{N_{conn}}\}$, denote a sequence of landmarks that the trajectory, $\sigma(t)$, passes from while connecting an initial state to a final one. Finally, $\mathcal{LS}$ is defined as a sequence of landmarks such that for each $L_i \in \mathcal{LS}$, there exist a duration of time, $[T_{0,i}, T_{1,i}]$ where (5.1) is satisfied $\forall t \in [T_{0,i}, T_{1,i}]$ and for two consecutive landmarks in the sequence, L_i and L_{i+1} , it is true that $T_{1,i} \leq T_{0,(i+1)}$.*

Once the definition of a landmark sequence is established, it is possible to redefine the landmark-aware motion planning problem to consider the particular case when the robot can travel without coverage.

Definition 5.3 (Landmark aware motion planning with maximum allowed distance). *The landmark aware optimal kinodynamic planning problem with maximum allowed distance is formalized as finding a feasible trajectory $\sigma(t) : [0, T^*] \rightarrow \mathcal{S}$ from a set of states and controls of a dynamic system and a duration $[0, T^*]$, i.e., $s^*(t) : [0, T^*] \rightarrow \mathcal{S}$ and $u^*(t) : [0, T^*] \rightarrow \mathcal{U}$, such that the trajectory*

- starts from an initial state $\sigma(0) = s_0$;
- avoids collisions with obstacles, i.e., $\sigma(t) \in \mathcal{S}_{free} \forall t \leq T^*$;
- fulfills the actuation bounds, i.e., $u(t) \in \mathcal{U}_{free} \forall t \leq T^*$;
- reaches the goal region, i.e., $\sigma(T^*) \in \mathcal{S}_{goal}$;
- minimizes an objective function defined as in (2.2);

and finding a sequence of landmarks, $\mathcal{LS}$, defined as in Definition 5.2 such that the following is true $\forall L_i \in \mathcal{LS}$,

$$\|L_i - L_{i+1}\| \leq d_{max} \quad (5.3)$$

where d_{max} is the maximum allowed distance between two consecutive landmarks that the robot passes from.

5.2 Sampling Based Landmark-Aware Planner

Landmark-aware motion planning problem or planning with connectivity constraints can also be addressed using the framework of RRT*_MP. However, in order to augment the framework with this objective, some modifications need to be done. In particular, based on the scope of the problem, the free space - where the landmarks are concerned - has to be defined explicitly so that the planner can select samples from a proper region or can check the feasibility of the trajectory.

This section explains the implementation of the two problems given in Definitions 5.1-5.3 in the RRT*_MP framework. The algorithm starts with forming a graph structure connecting the landmarks that allow a desired motion, depending on the problem. Consequently, the feasible region is defined and a compound cost is proposed to penalize the trajectory to account also for the distance from the network of landmarks. Finally, some numerical examples are provided to illustrate the effectiveness of the approach for both landmark-aware motion planning problems.

Connectivity Graph

The landmark-aware motion planning problem restricts the vehicle to move strictly inside a zone defined by the landmarks. Considering the starting point and the ultimate goal region of the vehicle, a network could be formed preliminarily to restrict this space to the union of the landmarks that allow a desired motion from start to goal. If the constraint is to be in the vicinity of at least one landmark at all times, this network is designed starting from the initial state and including only the landmarks that are separated with their emission range (Figure 5.2a-b). On the other hand, if the vehicle is allowed to travel for some distance without coverage as in the problem stated in Definition 5.3, then, a network is created which connects the landmarks that are away from each other less than a distance of d_{max} (Figure 5.3).

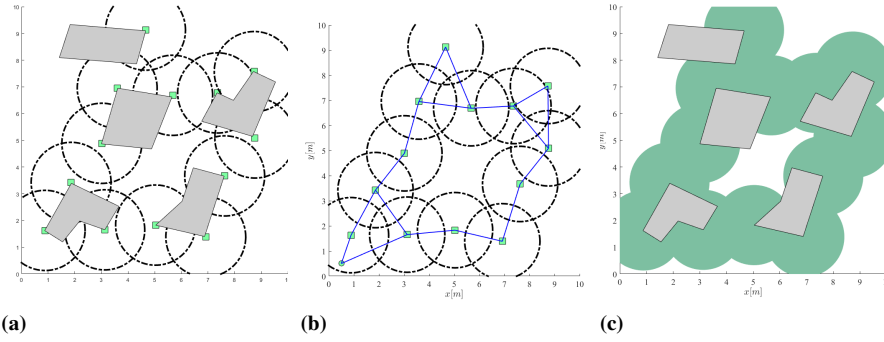


Figure 5.2: Defining the connectivity zone: (a) Signal emitting bases, with their emitting radii. (b) Connectivity graph formed starting from the initial state (green circle) connecting the landmarks that allow the desired motion. (c) Corresponding feasible region (green) in terms of connectivity with landmarks.

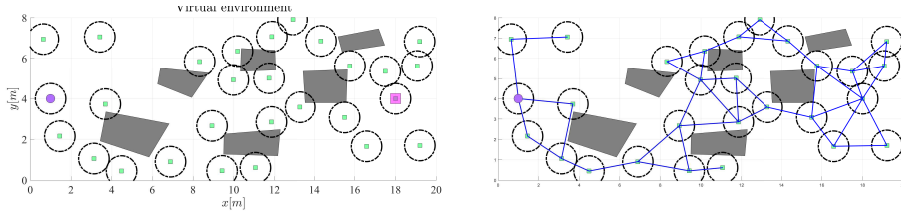


Figure 5.3: Left: A virtual environment populated with landmarks, robot initial position is marked with purple and the magenta square is the goal region. Right: Connectivity graph, when the robot is allowed to travel between two landmarks without coverage.

Landmarks as a Feasible Region

If the connectivity constraint limits the motion of the vehicle to be in connection with the landmarks at all times, then a restricted zone that is completely covered by the landmarks can be defined. Let $\mathcal{Q}_{conn} \subset \mathbb{R}^{n_q}$ denote the configurations that are in the vicinity of at least one landmark and recall that $\kappa : \mathcal{S} \rightarrow \mathcal{Q}$ is a function that mapping a state to its configuration. Following that, the feasible region in terms of connectivity is defined as the set of states that are connected with at least one landmark; i.e.:

$$\mathcal{S}_{conn} = \{s \in \mathcal{S} \mid \kappa(s) \in \mathcal{Q}_{conn}\}.$$

A trajectory is feasible if all the states along the trajectory lies in $\mathcal{S}_{free}$ and are connected with a landmark, i.e., $s \in \mathcal{S}_{conn}$. Therefore, the feasible

region can be defined as the intersection of these two subsets:

$$\mathcal{S}_{feas} = \{s \in \mathcal{S} \mid s \in \mathcal{S}_{free} \wedge s \in \mathcal{S}_{conn}\}.$$

In order to implement the feasible region into the framework of RRT*_MP, some slight modifications are required. Clearly, since RRT*_MP uses a discrete state space for sampling, the `Sample()` procedure is modified to draw samples from $\mathcal{S}_{feas}^\Delta$, where $\mathcal{S}_{feas}^\Delta = \{s \in \mathcal{S}^\Delta \mid s \in \mathcal{S}_{feas}\}$. Consequently, a new procedure is introduced, named `isConnected( $\sigma$ )`, that checks if the trajectory, σ , connecting two states, $s \in \mathcal{S}_{feasible}$ and $s' \in \mathcal{S}_{feasible}$ lies entirely in $\mathcal{S}_{conn}$. Finally, the procedures verifying the feasibility of a tentative trajectory, in particular `FindBestParent` and `Rewire`, are augmented with this subroutine that, collectively with `isCollisionFree` ensures that all states along the trajectory lie in the feasible region, $\mathcal{S}_{feas}$. Figure 5.2c shows an example of the feasible region with respect to the given arrangement of landmarks.

Compound Cost

Other than representing the landmarks as a feasible region, one might also be interested in minimizing a cost comprising the vicinity of the trajectory to the underlying network. In particular, for the second problem given in Definition 5.3 allowing a maximum travel distance as long as two consecutive landmarks are d_{max} apart from their centers, we propose to use a compound cost that penalizes the cost of the optimal cost trajectory connecting two states with the divergence from the network.

Finally, in order to penalize the divergence from the network, the cost function assigning a non-zero cost to each feasible trajectory is modified. Cost of the each edge includes the cost of the trajectory obtained from the specific motion primitive and is penalized with the normalized cost of being away from the network to be defined as follows

$$\text{Cost}(\sigma_{s \rightarrow s'}) = \left(1 + \frac{\bar{d}}{d_{range}}\right) J^* \quad (5.4)$$

where J^* represents the optimal cost of the specific motion primitive that is selected to build the edge $e = \{s, s'\}$ and $\bar{d}$ is the average of the distances between the points along the trajectory (in terms of position) and the nearest point on the network. The rationale behind this cost choice can be interpreted as a position varying weight on the optimal cost of the motion primitive which is computed based on the specific objective during the database generation phase.

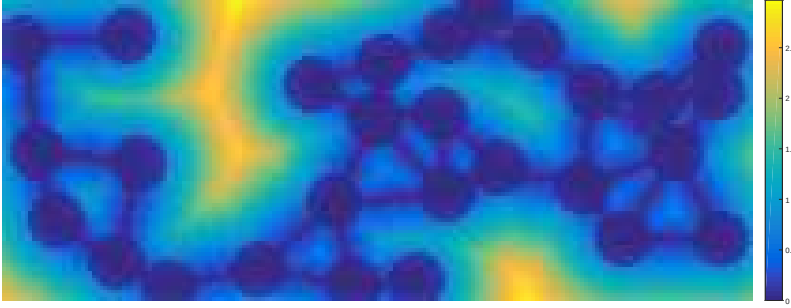


Figure 5.4: Costmap for the connectivity graph given in Figure 5.3. The blue zones correspond to lower cost areas and the cost is measured as the minimum distance from the network in meters, while the areas covered with landmarks are weighted to have a lower cost.

Costmap

Computing the distance from each point along the trajectory to the nearest point along the network is a computationally challenging process. Therefore, an efficient way to compute these distances in terms of a costmap is also proposed and it can answer multiple queries for the same environment once it is computed. The idea is based on gridding the 2D workspace² and computing the *Distance Transform* [66] of this grid, an approach adopted from image processing. Given a set of grid cells occupied by the points along the edges of the network, $u_j \in U$, the distance transform assigns each grid cell, the Euclidean distance from the nearest point along the network

$$DT(\mathbf{v}) = \min_{u_j \in U} ||\mathbf{v} - u_j||$$

where $\mathbf{v}$ represent the coordinates of the grid cell. As a result computing the distance of the vehicle to the nearest point on the network is converted into a LUT. Note that using the approach in [66], this costmap can be computed in linear time.

Other than directly assigning the distances, one might also design the costmap reducing the cost for areas that are covered by the landmarks as given in Figure 5.4. The idea is to penalize the part of the trajectory within the connectivity range of a landmark less since the vehicle would be connected in that case no matter how close it is to the landmark, as long as it is in the range. Finally, designing the costmap this way at each initiation

²For consistency the vehicle is assumed to be operating on a planar surface, but the idea is easily transferable as well to 3D.

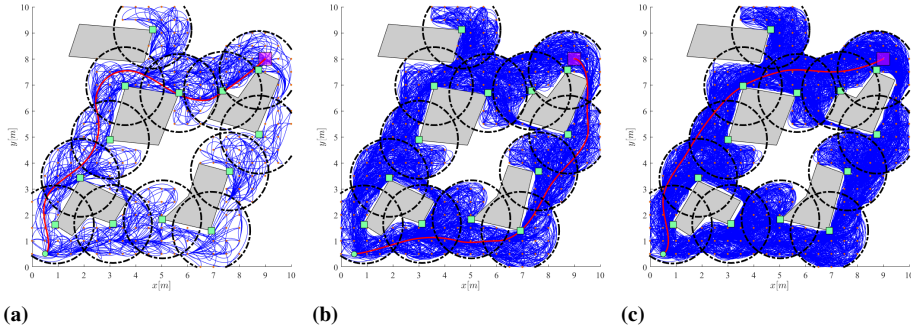


Figure 5.5: Motion planning with the restriction of being connected with a landmark at all times. Tree (blue edges, orange nodes) generated for different number of iterations, the best trajectory (red solid line) connecting the initial state (green circle) to a goal region (magenta square) and the emitting region of the landmarks (dashed black circles) are highlighted.

of the Cost (5.4), the average of the distances from the points along the trajectory, $\bar{d}$ is computed using the precomputed costmap.

Numerical Example

Both problems given in Definitions 5.1 and 5.3 are considered in the framework of RRT*_MP with the dynamic model and the database of motion primitives (fine resolution square gridding) explained in Section 3.6 using also the Rewire procedure.

For the first case, the algorithm is tested using the virtual environment given in Figure 5.2. The goal of the planner is to find the optimal trajectory with respect to an objective function inherited from the database of motion primitives without breaking the connection with a network of landmarks. Finally, once a feasible region is defined as in Figure 5.2c, the algorithm ensures that all the trajectories corresponding to the edges in the tree are strictly contained in this region. Figure 5.5 shows the resulting tree and the lowest cost trajectories for different number of iterations. Clearly, at each point along the trajectory, the vehicle is in connection with at least one landmark.

The effectiveness of the approach for the second problem (Definition 5.3) is verified by using a virtual environment given in Figure 5.3. In order to ensure that the trajectory follows the network in terms of a landmark sequence without diverging more than a distance threshold, the costmap is used to check the distance between a point along the trajectory and the closest point on the network. Moreover, a compound cost is implemented

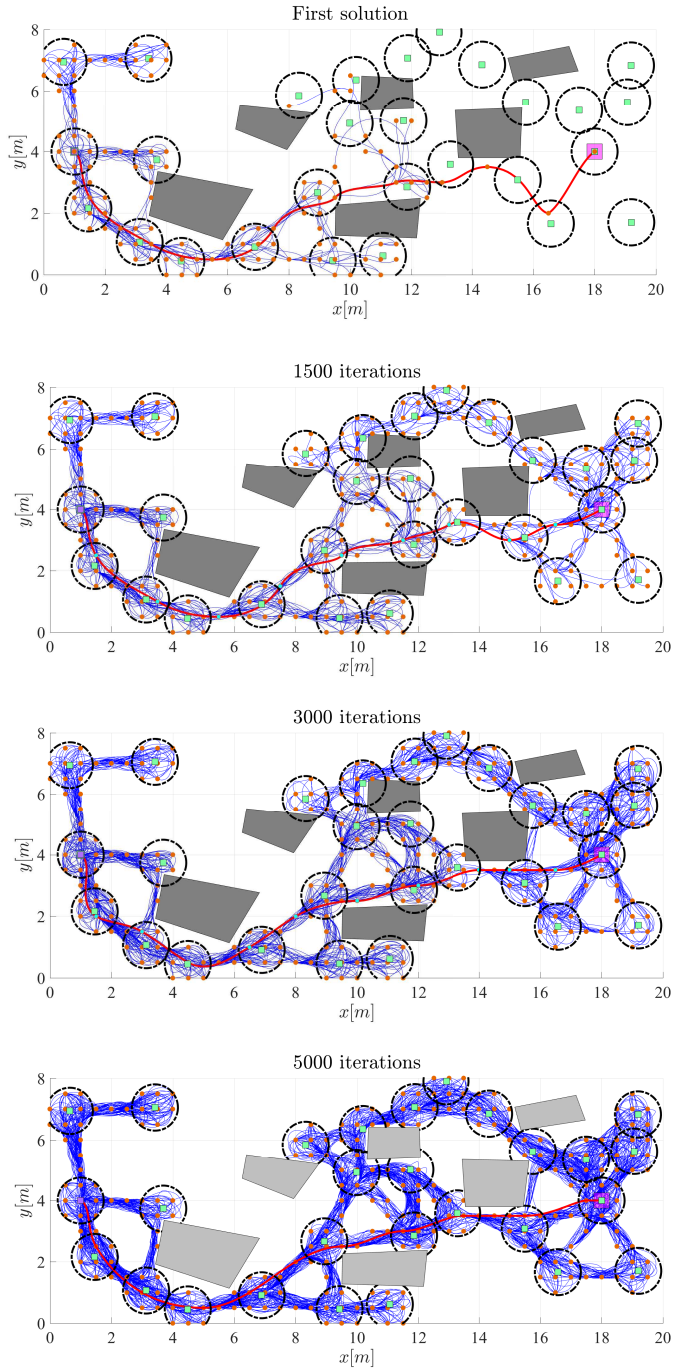


Figure 5.6: Motion planning with landmarks with maximum allowed distance. Resulting tree (blue edges, orange nodes) and the best cost trajectory (red solid line) are shown for different number of iterations.

as explained previously to modify the Cost function in order to account for this divergence. Finally, Figure 5.6 shows the resulting tree and the best cost trajectory obtained for different number of iterations. The algorithm at the end provides a dynamically feasible and resolution optimal trajectory along with a set of landmarks to be passed from without exceeding the maximum allowed distance between two consecutive landmarks and between a point along the trajectory and the closest point on the network.

For both methods, the landmark selection and trajectory generation are considered in a coupled approach as selecting landmarks while generating the optimal trajectory but not as selecting landmarks as waypoints and interpolating them. This approach puts the focus on generation of dynamically feasible trajectories that are executable by the actual system and ensures that the desired connection is established.

5.3 Exact Optimal Planner with Connectivity Constraints

The problem of being connected to at least one landmark, as given in Definition 5.1, is also addressed by using the exact optimal planner described in Chapter 4. The same framework comprising three steps is inherited: decomposing the environment and selecting cell sequences, solving for the optimal trajectory in the selected class and, finally, selecting the lowest cost trajectory. In order to account for the connectivity with landmarks cell sequence selection is slightly modified and an additional constraint on state variables, in particular on the position component of the state variables, is added.

Cell Sequence Selection

Recall that the very first step of exact approach based on homotopy class constraints is to systematically generate a set of algebraic path constraints to impose the homotopy class and to account for obstacles. In the landmark-aware motion planning, it is possible that some cells do not have coverage of a landmark. In addition, even though a cell has landmark coverage it may be unreachable from a particular initial state without breaking connection.

Considering these aspects, a cell sequence that contains a cell with either one of these conditions will not yield a feasible solution and can be eliminated before attempting to solve for a local optimal trajectory. For that purpose, before traversing the adjacency graph, some cells can be preliminarily marked as *no-go cells*, $\mathcal{C}_{nogo}$, if they do not have any landmark coverage, i.e.:

$$\mathcal{C}_{noGo} = \{c_i \in \mathcal{C} \mid \mu_L(\mathcal{P}_L \cap c_i) = 0\}$$

where $\mathcal{P}_L \subset \mathcal{P}$ corresponds to the space covered by landmarks recalling that $\mathcal{P}$ is the 2D workspace of the vehicle, and μ_L is the Lebesgue measure. Once the forbidden cells are highlighted, the graph traversal algorithm returns cell sequences with coverage.

Establishing the each cell along a particular cell sequence, $\mathcal{CS}$, does not necessarily ensure that there is a feasible trajectory within that homotopy class with all the points along the trajectory connected to at least one landmark. Therefore, for each cell sequence, a preliminary feasibility check is applied, establishing that there exists a path from an initial position to the final one that is fully contained in $\mathcal{P}_L$. However, it is worth mentioning that, while ensuring a feasible path, this preliminary check does not guarantee dynamical feasibility. Thus, even after this check, the homotopy class corresponding to the cell sequence may not yield a solution.

Connectivity Constraint

The second modification to the exact planner is the introduction of the connectivity constraint to the OCP. The problem given in Definition 5.1 requires that the robot is in the vicinity of at least one landmark as in (5.2). However, this constraint itself is not smooth and can be handled only by some particular nonlinear solvers. Therefore, in order to have a more general problem, a smooth constraint ensuring connectivity is proposed.

The connectivity constraint in this case relies on a particle filter like approach. The idea is to define a function in order to obtain the weighted average of the distances from each landmark, which then constrains the vehicle to be in the vicinity of the closest landmark for all the points along the trajectory. For each one of the landmarks, the weights are computed using a Gaussian function with the mean located at the center of the landmark:

$$\alpha_i = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(L_i)} \quad (5.5)$$

and the weighted average of the distance, $d(t)$, from the landmarks is computed using these weights and the distances from the landmarks

$$d(t) = \sum_{i=1}^{N_l} \frac{\alpha_i}{\bar{\alpha}} \|\pi(t) - L_i\| \quad (5.6)$$

where N_l represents the total number of landmarks and $\bar{\alpha}$ is a normalization factor such that

$$\bar{\alpha} = \sum_{j=1}^{N_l} \alpha_i \quad (5.7)$$

Finally, the connectivity constraint is defined as

$$d(t)^{(p)} \leq d_{range} \quad (5.8)$$

for each phase. The multiple phase OCP described in Section 4.4.1 is augmented with this additional constraint and, as in the original problem this problem can also be converted into a NLP to be solved efficiently.

Numerical Example

The effectiveness of the approach is verified by using a numerical example in a virtual environment populated with landmarks and obstacles. Figure 5.7 shows the resulting trajectories for four different homotopy classes by using the same vehicle model and the objective function with the same actuation and velocity bounds as in Section 4.4.2. Clearly, while generating distinct trajectories belonging to distinct homotopy classes, the planner also ensures that the resulting trajectory is always in connection with at least one landmark.

In this numerical example, the same approach as the one discussed in Section 4.4.1 is employed such that the cell sequences are selected as the loopless ones from an undirected graph and the best trajectory is defined as the one having the lowest cost. However, in the presence of landmarks, one might also be interested in other specific aspects. For example, the robustness of the trajectory in terms of connection could be one of these aspects. In other words, having distinct good trajectories allows us to make higher level decisions by selecting a trajectory that belongs to a class with higher number of landmarks so that if one of the landmarks along the way is compromised, there will always be an alternative to reach the goal.

5.4 Conclusion

This chapter explains the motion planning problem, when the vehicle has to communicate with a landmark that can be perceived only from a range. The problem is addressed using the frameworks of two different motion planners explained in the previous chapters and the effectiveness is demonstrated with the simulation results. The main difference between the approach employed here and the existing literature is it focuses on generating

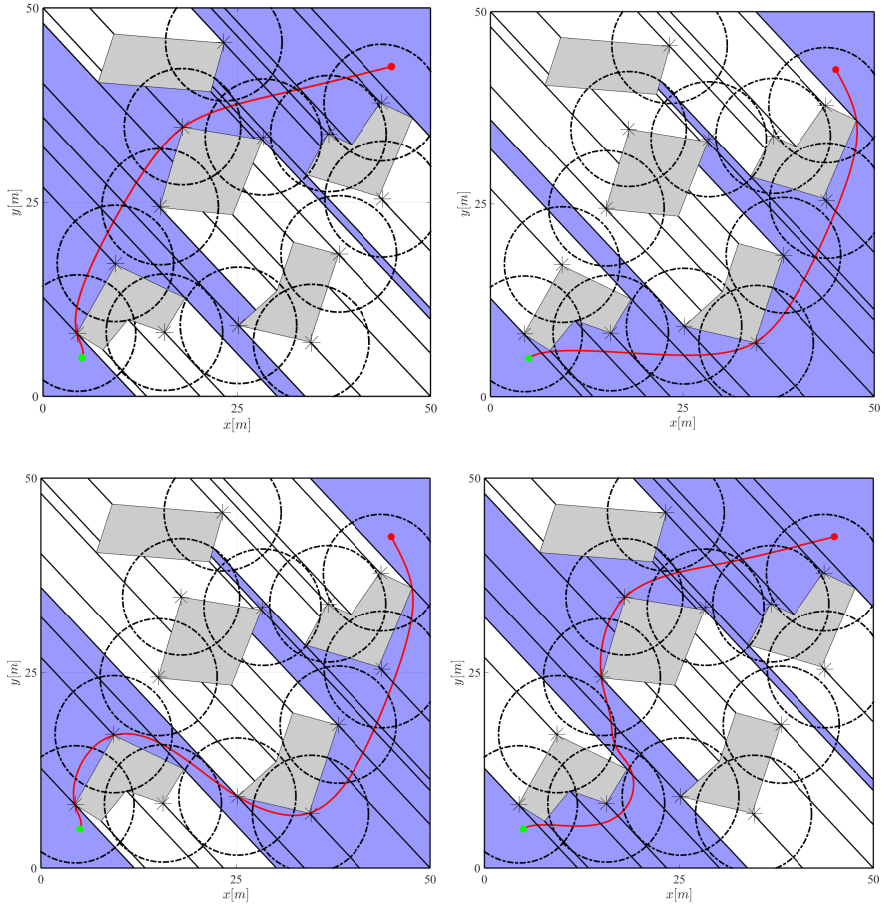


Figure 5.7: Environment with polygonal obstacles (gray areas) and landmarks (represented as black stars and with a circular range -dashed black lines-) Initial and final positions are green and red points respectively. The resulting local optimal trajectories (solid red lines) computed on the corresponding homotopy classes (blue cell sequences). The space is decomposed after rotating the environment about the initial position (green dot) to obtain a cell sequence for monotonic trajectory.

dynamically feasible trajectories while imposing the constraints aroused by the landmarks, but not on selecting a set of landmarks and adapting a feasible trajectories since this does not always necessarily respect the constraints or ensure that dynamically feasible trajectories exist.

CHAPTER 6

Conclusions and Prospective Works

This thesis strives to explain the contributions in the field of optimal motion planning algorithms. In particular, it focuses on how to develop efficient algorithms that not only solve the motion planning problem but also study the underlying exact dynamics of autonomous vehicles or mobile robots. The problem is addressed using important classes of motion planning algorithms: the sampling based approach and the exact approach. Furthermore, some detailed attention is given to planning with respect to the topology of workspace. Finally, the specific problem of planning while keeping the connection alive is considered. This chapter summarizes the thesis and draws some concluding remarks. Consequently, it highlights the possible future works and research directions to improve the work presented here.

Summary and Conclusions

In the first part of the thesis, the optimal kinodynamic planning is discussed in the framework of randomized sampling based approaches while particularly concentrating on RRT* based algorithms. The main difficulty in planning with arbitrary dynamics stems from the requirement to solve challenging TPBVPs at each step. In order to reduce the computational

load caused by this requirement, a novel approach combining search and sampling-based algorithms is proposed. The idea is to introduce a database of motion primitives and avoid solving TPBVPs using a LUT. To ensure that any two states can be connected using the motion primitives, the samples are drawn from the discrete state space (uniform grid). The algorithm is proven to be probabilistically complete, resolution optimal and asymptotically optimal as the grid resolution goes to zero, allowing infinite number of distinct states to be sampled. Other than the sampling-based motion planners that focus on generating efficient TPBVP solvers, the algorithm proposed in this thesis is not restricted to a limited set of dynamics and can handle any arbitrary dynamics problems with additional constraints and bounds on actuation.

The second part elaborates on the motion planning under topology constraints. In particular, two problems have been considered: obtaining a set of good trajectories that belong to distinct homotopy classes and obtaining a set of algebraic constraints that can be used to impose the homotopy class belonging during the phase of trajectory generation. To be able to limit the number of homotopy classes to be solved to a number of promising ones, a systematic cell sequence selection method is also proposed, which is proven to be effective in eliminating classes with higher cost trajectories for the 'minimum final time minimum energy' problem at hand. Having the exact representation of the free space, a motion planning framework based on the homotopy class constraints and multiple phase formalization of the optimal control problem is developed. Lastly, the sampling-based algorithm is also augmented with topology awareness property.

Finally, both of the aforementioned algorithms are applied to a peculiar problem of landmark aware kinodynamic planning. Considering a range constraint imposed by landmarks, the idea is to ensure that the vehicle keeps the connection alive with a set of landmarks that allow a motion from an initial state to a goal. Unlike the approaches attempting to solve similar problems by focusing on landmark selection while aiming to adapt trajectories that would pass through them, we have addressed the problem from the opposite angle. Specifically, the problem is formalized to find the optimal trajectory that, at the end, respects the connectivity constraints. This allows us to reason about the optimality of the trajectory without bringing restrictions to force it to pass from waypoints.

Prospective Improvements

Despite the research provided here concentrating on efficient kinodynamic motion planning with certain objectives, there are still some questions that are left unanswered. From a theoretical point of view, even though the completeness and optimality analysis have been carried out for RRT*_MP, the explicit description of how these properties change for the heuristic proposed in the homotopic counterpart should also be discussed.

From the application point of view, the imminent prospective work is to test the algorithms developed in this thesis in a real environment. In order to apply the planner to a real robot, RRT*_MP is being implemented in C++ language, using the common sampling based motion planning library OMPL [89]. Furthermore, improving the framework of RRT*_MP, in particular diverging from the RRT* framework and including an effective pruning without sacrificing optimality, hence eliminating the need of uniform gridding will be considered. Finally, some effort will be put on improving the database and in particular the description of the motion primitives to remove the restriction on the boundary points such that a large class of trajectories can be stored in one primitive that can be deformed to others during the course of the planning.

Considering the planning with homotopy class constraints, it has been demonstrated that increasing the number of obstacles also increases the computation time of the exact method for a single problem since the number of phases defining each class is increased. The algorithm performs well, generating distinct obstacle avoidance strategies while the planner considers a small number of obstacles at each time. However, it is not scalable for solving a complex problem involving numerous obstacles. Therefore, a prospective improvement in this area is to compute a more conservative sequence of obstacle avoidance constraints that include only the salient obstacle boundaries.

Finally, for landmark-aware planning, the problem will be extended to include more specific aspects such as motion planning for multiple agents while keeping the connection among them alive and taking into account the time needed for successful data transmission/reception.

Bibliography

- [1] Sterling J Anderson, Sisir B Karumanchi, and Karl Iagnemma. Constraint-based planning and control for safe, semi-autonomous operation of vehicles. In *Intelligent Vehicles Symposium (IV)*, 2012 IEEE, pages 383–388. IEEE, 2012.
- [2] Sterling J Anderson, Sisir B Karumanchi, Karl Iagnemma, and James M Walker. The intelligent copilot: A constraint-based approach to shared-adaptive control of ground vehicles. *IEEE Intelligent Transportation Systems Magazine*, 5(2):45–54, 2013.
- [3] Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA)*, 2013 IEEE International Conference on, pages 2421–2428. IEEE, 2013.
- [4] Luitpold Babel. Flight path planning for unmanned aerial vehicles with landmark-based visual navigation. *Robotics and Autonomous Systems*, 62(2):142–150, 2014.
- [5] Bonny Banerjee and B Chandrasekaran. A framework for planning multiple paths in free space. In *Proceedings of 25th Army Science Conference, Orlando, FL*, 2006.
- [6] Maximilian Beinhofer, Jörg Müller, and Wolfram Burgard. Near-optimal landmark selection for mobile robot navigation. In *Robotics and Automation (ICRA)*, 2011 IEEE International Conference on, pages 4744–4749. IEEE, 2011.
- [7] Dimitri P Bertsekas and John N Tsitsiklis. *Introduction to probability*, volume 1. Athena Scientific Belmont, MA, 2002.
- [8] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance control and dynamics*, 21(2):193–207, 1998.
- [9] Subhrajit Bhattacharya, Vijay Kumar, and Maxim Likhachev. Search-based path planning with homotopy class constraints. In *Third Annual Symposium on Combinatorial Search*, 2010.
- [10] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, 2012.
- [11] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli. Efficient collision checking in sampling-based motion planning via safety certificates. *The International Journal of Robotics Research*, 35(7):767–796, 2016.

Bibliography

- [12] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.
- [13] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.
- [14] Jonathan Bohren, Tully Foote, Jim Keller, Alex Kushleyev, Daniel Lee, Alex Stewart, Paul Vernaza, Jason Derenick, John Spletzer, and Brian 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.
- [15] Béla Bollobás and Oliver Riordan. *Percolation*. Cambridge University Press, 2006.
- [16] Frederic Bourgault, Alexei A Makarenko, Stefan B Williams, Ben Grocholsky, and Hugh F Durrant-Whyte. Information based adaptive robotic exploration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 540–545. IEEE, 2002.
- [17] Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. Completeness of randomized kinodynamic planners with state-based steering. *Robotics and Autonomous Systems*, 89:85–94, 2017.
- [18] Bernard Chazelle. A theorem on polygon cutting with applications. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 339–349. IEEE, 1982.
- [19] Bernard Chazelle. Approximation and decomposition of shapes. *Algorithmic and Geometric Aspects of Robotics*, 1:145–185, 1985.
- [20] Peng Cheng and Steven M LaValle. Reducing metric sensitivity in randomized trajectory design. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 43–48. IEEE, 2001.
- [21] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.
- [22] Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Aaai*, volume 6, pages 942–947, 2006.
- [23] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.
- [24] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.
- [25] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.
- [26] Alon Efrat, Stephen G Kobourov, and Anna Lubiw. Computing homotopic shortest paths efficiently. *Computational Geometry*, 35(3):162–172, 2006.
- [27] Rida T Farouki. Pythagorean-hodograph curves. *Handbook of computer aided geometric design*, pages 405–427, 2002.
- [28] Dave Ferguson, Michael Darms, Chris Urmson, and Sascha Kolski. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1149–1154. IEEE, 2008.
- [29] Dave Ferguson, Thomas M Howard, and Maxim Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11-12):939–960, 2008.

- [30] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. In *American Control Conference, 2001. Proceedings of the 2001*, volume 1, pages 43–49. IEEE, 2001.
- [31] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. Technical report, Tech. Report TR-2014-JDG006, ASRL, University of Toronto, 2014.
- [32] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2997–3004. IEEE, 2014.
- [33] David González, Joshue Pérez, Ray Lattarulo, Vicente Milanés, and Fawzi Nashashibi. Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 1430–1435. IEEE, 2014.
- [34] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2429–2436. IEEE, 2013.
- [35] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Obbtrees: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM, 1996.
- [36] Tianyu Gu, John M Dolan, and Jin-Woo Lee. Automated tactical maneuver discovery, reasoning and trajectory planning for autonomous driving. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5474–5480. IEEE, 2016.
- [37] Emili Hernandez, Marc Carreras, and Pere Ridao. A comparison of homotopic path planning algorithms for robotic applications. *Robotics and Autonomous Systems*, 64:44–58, 2015.
- [38] B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.
- [39] Thomas M Howard, Colin J Green, Alonzo Kelly, and Dave Ferguson. State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. *Journal of Field Robotics*, 25(6-7):325–345, 2008.
- [40] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.
- [41] Jeong hwan Jeon, Raghvendra V Cowlagi, Steven C Peters, Sertac Karaman, Emilio Frazzoli, Panagiotis Tsiotras, and Karl Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *American Control Conference (ACC), 2013*, pages 188–193. IEEE, 2013.
- [42] Jeong hwan Jeon, Sertac Karaman, and Emilio Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the rrt. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 3276–3282. IEEE, 2011.
- [43] Joo Young Hwang, Jun Song Kim, Sang Seok Lim, and Kyu Ho Park. A fast path planning by path graph optimization. *IEEE Transactions on systems, man, and cybernetics-part a: systems and humans*, 33(1):121–129, 2003.

- [44] Kevin Dean Jenkins. *The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes*. PhD thesis, Monterey, California. Naval Postgraduate School, 1991.
- [45] Sertac Karaman. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104.
- [46] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7681–7687. IEEE, 2010.
- [47] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.
- [48] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5041–5047. IEEE, 2013.
- [49] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Any-time motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.
- [50] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.
- [51] Hassan K Khalil. *Nonlinear Systems*, volume 2. Prentice-Hall, New Jersey, 1996.
- [52] Markus Kuderer, Christoph Sprunk, Henrik Kretschmar, and Wolfram Burgard. Online generation of homotopically distinct navigation paths. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6462–6467. IEEE, 2014.
- [53] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.
- [54] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.
- [55] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.
- [56] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.
- [57] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.
- [58] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.
- [59] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.
- [60] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Sparse methods for efficient asymptotically optimal kinodynamic planning. In *Algorithmic foundations of robotics XI*, pages 263–282. Springer, 2015.
- [61] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.
- [62] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

- [63] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.
- [64] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.
- [65] Andrzej Lingas. The power of non-rectilinear holes. In *International Colloquium on Automata, Languages, and Programming*, pages 369–383. Springer, 1982.
- [66] Calvin R Maurer, Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, 2003.
- [67] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. In *AAAI*, pages 1177–1183, 2007.
- [68] Chia Ching Ooi and Christian Schindelhauer. Minimal energy path planning for wireless robots. *Mobile Networks and Applications*, 14(3):309–321, 2009.
- [69] Luigi Palmieri, Sven Koenig, and Kai O Arras. Rrt-based nonholonomic motion planning using any-angle path biasing. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2775–2781. IEEE, 2016.
- [70] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3859–3866. IEEE, 2012.
- [71] Junghee Park, Sisir Karumanchi, and Karl Iagnemma. Homotopy-based divide-and-conquer strategy for optimal trajectory planning via mixed-integer programming. *IEEE Transactions on Robotics*, 31(5):1101–1115, 2015.
- [72] Michael A Patterson and Anil V Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1, 2014.
- [73] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.
- [74] Mathew Penrose. *Random geometric graphs*. Number 5. Oxford University Press, 2003.
- [75] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2537–2542. IEEE, 2012.
- [76] Mihail Pivtoraiko and Alonzo Kelly. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7, 2005.
- [77] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.
- [78] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*, pages 489–494. IEEE, 2009.
- [79] John H Reif. Complexity of the mover’s problem and generalizations. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 421–427. IEEE, 1979.
- [80] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

- [81] Gildardo Sánchez and Jean-Claude Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.
- [82] Mac Schwager, Philip Dames, Vijay Kumar, and Daniela Rus. Multi-robot mapping and exploration of environments with hazards. In *RSS 2011 workshop on 3D exploration, mapping, and surveillance with aerial robots*, 2011.
- [83] Ulrich Schwesinger, Roland Siegwart, and Paul Furgale. Fast collision detection through bounding volume hierarchies in workspace-time space for sampling-based motion planners. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 63–68. IEEE, 2015.
- [84] Alessandro Settimi, Danilo Caporale, Przemyslaw Kryczka, Mirko Ferrati, and Lucia Pallotino. Motion primitive based random planning for loco-manipulation tasks. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 1059–1066. IEEE, 2016.
- [85] Zvi Shiller, Yusuke Fujita, Dan Ophir, and Yoshihiko Nakamura. Computing a set of local optimal paths through cluttered environments and over open terrain. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4759–4764. IEEE, 2004.
- [86] Christoph Sprunk, Boris Lau, Patrick Pfaffz, and Wolfram Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 72–77. IEEE, 2011.
- [87] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.
- [88] Samantha Stoneman and Roberto Lampariello. Embedding nonlinear optimization in RRT* for optimal kinodynamic planning. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 3737–3744. IEEE, 2014.
- [89] Ioan A. Șucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. <http://ompl.kavrakilab.org>.
- [90] Chris Urmson and Reid Simmons. Approaches for heuristically biasing rrt growth. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1178–1183. IEEE, 2003.
- [91] Dustin J Webb and Jur van den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5054–5061. IEEE, 2013.
- [92] Christopher Xie, Jur van den Berg, Sachin Patil, and Pieter Abbeel. Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4187–4194. IEEE, 2015.
- [93] Wenda Xu, Junqing Wei, John M Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2061–2067. IEEE, 2012.
- [94] Sen Zhang, Lei Sun, Zhongliang Chen, Xiang Lu, and Jingtai Liu. Smooth path planning for a home service robot using η 3-splines. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 1910–1915. IEEE, 2014.