This work was submitted to the Chair of Navigation

Master's Thesis

Deterministic Sampling-Based Motion Planning

Leonard Bruns

330549

September 26, 2019

Supervisors: Dr. rer. nat. Luigi Palmieri (Robert Bosch GmbH)
M.Sc. Marius Brachvogel (RWTH Aachen)

Examiner: Univ.-Prof. Dr.-Ing. habil. Michael Meurer

I assure, that this work – with the exception of the official supervision by the chair and by my supervisor at Robert Bosch GmbH – was carried out by me without external help. The used literature is completely indicated in the bibliography.

Renningen, September 26, 2019

Leonard Bruns

(Leonard Bruns)

Abstract

A key challenge in robotics is the efficient generation of optimal robot motion with safety guarantees in cluttered environments. Recently, deterministic optimal sampling-based motion planners have been shown to achieve good performance towards this end, in particular in terms of planning efficiency, final solution cost, quality guarantees as well as non-probabilistic completeness. Yet their application is still limited to relatively simple systems (i.e., linear, holonomic, Euclidean state spaces). In this work, we extend this technique to the class of symmetric and optimal driftless systems by introducing an optimization technique for precomputing sets with known and optimized dispersion, aware of differential constraints, for sampling-based robot motion planning. We prove that the approach gives the same asymptotic dispersion as state-of-the-art low-dispersion sequences, and show through multiple experiments that it outperforms all baselines in practice. Another approach to differentially constrained, deterministic sampling-based motion planning is state lattice planning. While shown to be a computationally efficient choice in the past, the completeness guarantees that such planners give have not been formally discussed in the literature so far. We use the same reachable set-based framework to derive interpretable completeness guarantees for both PRM and state lattice planners for all symmetric and optimal systems. Additionally, based on this analysis, we propose an algorithm for generating a set of motion primitives with proven completeness guarantees.

List of Symbols

General notation

a	Scalar
$\boldsymbol{a}$	Vector
a_i	Vector element
$\boldsymbol{A}$	Matrix or tensor
$A_{i,j}$	Matrix or tensor element
$\mathcal{A}$	Set
$a_{\text{specification}}$	Variable with description (note the difference to indices)

Common variables

c	Cost
σ	Path
$\mathcal{X}$	Configuration space
$\boldsymbol{x}$	Configuration
$\mathcal{U}$	Control space
$\boldsymbol{u}$	Control
D	Dimension of the configuration space
$d, \tilde{d}$	Dispersion and modified dispersion
n	Number of samples
$\mathcal{S}$	Set of samples
$\mathcal{P}$	Set of motion primitives
$\mathcal{V}$	Set of vertices
$\mathcal{E}$	Set of edges

Contents

1 Introduction

With the technological advancement of recent years in many different fields, robots operating in close proximity of humans will become a reality in the near future. Not only must these systems move and coordinate around humans, they might also be responsible for the human operator as in the case of autonomous cars. In such safety-critical applications high reliability and verification is of major importance for every part of the system.

Examples of existing and emerging robotic products heavily depending on robust motion planning are autonomous cars (e.g., Waymo, Bosch/Daimler) and various kinds of service robots (e.g., Boston Dynamics, ANYmal). See Figure 1.1 for some examples.

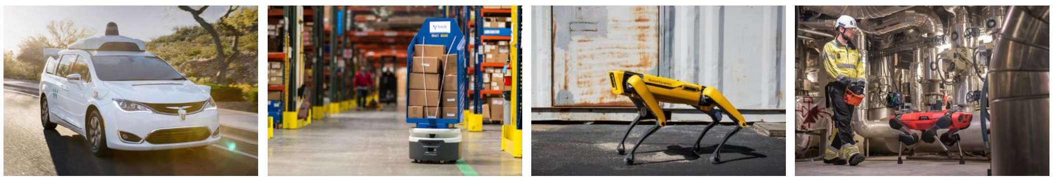
Motion planning is an integral part of any robotic system that allows to plan the motion according to the constraints and requirements of the application. Often motion planners are realized through some form of sampling-based algorithm. This branch of motion planning, while highly successful due to its scalability to complex and high-dimensional problems, classically introduces randomness into the planning process by randomly sampling possible trajectories or system configurations.

While this randomness might be a small contributing factor in comparison to the randomness that unknown and unstructured environments create, this randomness still prevents these algorithms to work fully reliable to the point of proven completeness, which is a desired property for systems potentially affecting human life.

Hence, this work focuses on different approaches to derandomize sampling-based motion planners and even use this deterministic framework to improve the performance of motion planners. The prior work in this area is mostly limited to Euclidean systems without differential constraints and the goals of this thesis are to investigate ways of improving deterministic planners and to extend the prior work to systems with differential constraints.

Contributions

In this work, we provide rigorous completeness guarantees for symmetric systems (potentially differentially constrained, nonholonomic, etc.) for which an optimal steering function is available. In addition we introduce a numeric optimization approach that minimizes the number of samples required based on the developed theory. This way, not only fewer samples are required during planning (i.e., higher



(a) Waymo[1] (b) VirtualConveyor[2] (c) Spot[3] (d) ANYmal[4]

Figure 1.1: Examples of robotic products relying on robust motion planning.

efficiency), but it can be stated that queries of certain clearance will definitely be solved by slightly modified versions of batch algorithms like probabilistic roadmap (PRM) and fast marching tree (FMT).

In addition the theory developed for these batch algorithms is extended to state lattice planning. We show that this branch focussing on highly efficient motion planners for differentially constrained systems can be seen as a special case of probabilistic roadmaps and hence, similar tools can be used to proof completeness for these planners. We use this insight to derive primitive generation schemes that give provable completeness guarantees.

Finally, as part of the work general implementations of state lattice planners as well as deterministic samplers for the Open Motion Planning Library (OMPL) are provided.

Structure

In Chapter 2 we introduce the motion planning problem we are trying to solve and the required background knowledge to understand the algorithms and terms used throughout the thesis. Chapter 3 gives an overview of the related literature in the area of deterministic motion planning. Subsequently we introduce and motivate our approach in Chapter 4. We show the experiments and results in Chapter 5, in which we compare our approach to the previously introduced baselines and discuss the results. Finally, we provide a summary and conclusion in Chapter 6. Most chapters include definitions, theorems, proofs and examples as necessary. These will be marked as such.

[1]`https://waymo.com/`, accessed: 26.09.2019
[2]`https://fetchrobotics.com/`, accessed: 26.09.2019
[3]`https://www.bostondynamics.com/spot`, accessed: 26.09.2019
[4]`https://www.anybotics.com/anymal-legged-robot/`, accessed: 26.09.2019

2 Fundamentals

This section introduces the fundamentals required for the topic of the thesis starting from the sampling-based motion planning paradigm. The other popular approach, combinatorial motion planning, will not be discussed since it is not relevant for the topic and also not applicable in many realistic scenarios that are addressed in this work. Subsequently the separate problem of differentially constrained motion planning will be introduced. Afterwards, the most important algorithms are introduced and finally the relevant parts of sampling theory are summarized. A much more thorough introduction into the topic can be found in LaValle's books on planning algorithms [1]. Before we start with sampling-based motion planning, we introduce the most important notions in the field of motion planning, which are used throughout this work.

2.1 Motion Planning

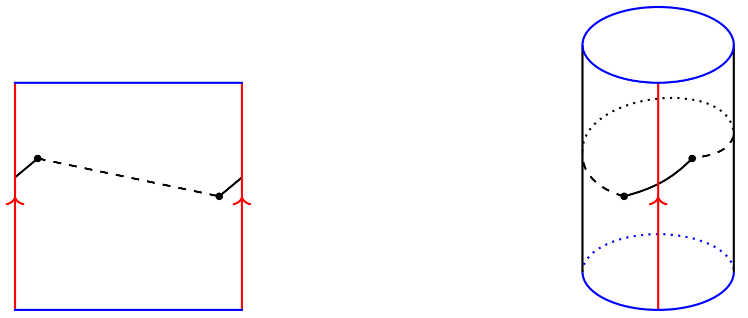
Motion planning is generally concerned with finding collision free paths between start and goal specifications in a so called configuration space.

Definition 1. Let $\mathcal{X} \subset \mathbb{R}^D$ denote the *configuration space*. A single configuration in this space will be denoted by $\boldsymbol{x} \in \mathcal{X}$. The set $\mathcal{X}_{\text{obs}} \subset \mathcal{X}$ is comprised of the states which are in collision, either due to self-collision, or collision with the environment. $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$ denotes the set of collision-free (also called valid) states.

Note that the configuration space might have so called *identification*, which means that along some dimensions there exists a wrap around, such that the distance as well as paths can cross the border to appear on the other end again. An example for an identified configuration space is 1D-orientation, often denoted by SO(2)[1]. Figure 2.1 visualizes this idea of identification. We mark identified spaces with $/ \sim$. For example the space shown in Figure 2.1 could be denoted by $[0,1] \times [0,1]/ \sim$, with $\times$ denoting the Cartesian product.

Definition 2. A *path* is defined as a function $\sigma : [0,1] \to \mathcal{X}$. A *feasible path* is defined as a function $\sigma : [0,1] \to \mathcal{X}_{\text{free}}$. We denote the set of all points along σ by

[1]SO(2) stands for special orthogonal group in dimension 2, since rotation matrices for 1D orientations have the size 2×2



(a) 2D representation (b) Identified representation

Figure 2.1: Visualization of the concept of identification. On the left a 2-dimensional space is shown for which the horizontal dimension is identified. A path between the two marked states can go across the boundary. On the right we show that this can be imagined as a cylindrical space. An example for such a space would be a cart on a circular rail (identified dimension) with an actuator that can only vary in height (non-identified dimension).

$\sigma([0,1])$, i.e.,

$$\sigma([0,1]) = \{\sigma(t) \in \mathcal{X} | t \in [0,1]\}. \tag{2.1}$$

Definition 3. Let γ denote a *query*, defined by a starting state $\boldsymbol{x}_{\text{init}} \in \mathcal{X}$ and a goal state $\boldsymbol{x}_{\text{goal}} \in \mathcal{X}$. A *feasible query* is a query for which a feasible path σ exists, such that $\sigma(0) = \boldsymbol{x}_{\text{init}}$ and $\sigma(1) = \boldsymbol{x}_{\text{goal}}$. We call such a feasible path σ solution to γ. Additionally, we define Σ_γ as the set of all possible solutions to a query γ.

Note that by this definition we limit ourselves to a single start configuration $\boldsymbol{x}_{\text{init}} \in \mathcal{X}_{\text{free}}$ and a single goal configuration $\boldsymbol{x}_{\text{goal}} \in \mathcal{X}_{\text{free}}$. While the first is quite practical in many cases, the second might not be desirable if a goal region should be considered. Such a modification would not make a big difference though and with minor adjustments most of this work should still be applicable to multiple goals or goal regions.

Based on these definitions we can now state the basic motion planning problem that we are trying to solve:

Problem 1. Given a query γ either report a solution σ or correctly report that no solution to γ exists.

Planners that solve this problem are called *complete*. Often though, especially in the context of sampling-based motion planning, completeness can not be achieved. Instead the concepts of *probabilistic completeness* and *resolution completeness* are used.

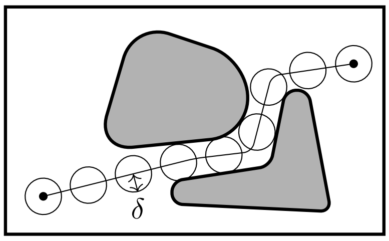


Figure 2.2: Visualization of the clearance of a query. Given a query γ, δ is the radius of the ball that can be moved from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$ while being fully contained in $\mathcal{X}_{\text{free}}$.

Definition 4. A planner is called *probabilistically complete* if it will return a solution to all feasible queries γ with a probability of 1 if the planning time $t \to \infty$ or, equivalently, the number of samples $n \to \infty$.

For resolution completeness we must first define the notion of clearance.

Definition 5. The *δ-clearance* of a path σ is defined by[1]

$$\delta(\sigma) = \sup\{r \in \mathbb{R} \mid \mathcal{B}(\boldsymbol{x}, r) \subseteq \mathcal{X}_{\text{free}} \forall \boldsymbol{x} \in \sigma([0,1])\}. \tag{2.2}$$

Based on this, the *δ-clearance* of a query γ is defined by

$$\delta(\gamma) = \sup\{\delta(\sigma) \mid \sigma \in \Sigma_\gamma\}. \tag{2.3}$$

Figure 2.2 visualizes this definition. In simple words, the clearance of a query is half the width of the widest corridor that connects $\boldsymbol{x}_{\text{init}}$ with $\boldsymbol{x}_{\text{goal}}$. Given this notion of clearance we can introduce the concept of resolution completeness.

Definition 6. A planner is called $\hat{\delta}$-complete, if it will deterministically return a solution to all queries γ with $\delta(\gamma) \geq \hat{\delta}$.

Note that a $\hat{\delta}$-complete planner might still, when faced with a query γ with clearance $\delta(\gamma) < \hat{\delta}$, return a solution. Thus, such a planner does not ensure that the solutions it finds have a clearance $\delta(\sigma) > \hat{\delta}$, only that if a solution with $\delta(\sigma) > \hat{\delta}$ exists, it will find *some* solution (potentially with lower clearance). Generally, if some clearance for safety reasons is desired, this should be handled by the collision checker overapproximating the collision model and not by the planning algorithm[2].

It is also interesting to stress the difference of planners that are $\hat{\delta}$-complete and probabilistically complete, which is a notion that can be found a lot in the literature

[1]See Definition 10 and the related discussion for the definition of $\mathcal{B}$.

[2]It would be immensely difficult (or even impossible in the general case) to optimize clearance with a planner that only builds a representation of the configuration space via sampling.

[2]. The second only states that the probability of finding an existing solution converges to 1 as long as the number of samples $n \to \infty$. Note that $\hat{\delta}$-complete implies probabilistic completeness as long as $\hat{\delta} \to 0$ for $n \to \infty$. Thus $\hat{\delta}$-completeness is a stronger property than probabilistic completeness.

Next, we introduce the notion of optimal motion planning by first introducing a cost function that measures the quality of a path.

Definition 7. The *cost* of a path σ is defined as a function $c : \sigma \to \mathbb{R}_{\geq 0}$.

Common cost functions include path length, energy, time, clearance or weighted combinations of these. In this work we only consider arc length for which c is given by

$$c = \int_0^1 ||\dot{\sigma}(t)||dt. \tag{2.4}$$

For many simple systems, e.g., ones with a constant velocity, optimizing arc length corresponds to optimizing the required time.

Using this cost we can state the optimal motion planning problem:

Problem 2. Given a query γ either correctly report that no solution to γ exists or, if γ is a feasible query, report a solution σ^* such that

$$c(\sigma^*) = \min_{\sigma \in \Sigma_\gamma} c(\sigma) = c^*. \tag{2.5}$$

Note that such a solution σ^* might not exist if $\mathcal{X}_{\text{obs}}$ would be closed. In that case we would require to find a sequence of solutions $(\sigma_n)_{n \in \mathbb{N}}$ such that $\lim_{n \to \infty} c(\sigma_n) = c^*$. This idea is visualized in Figure 2.3. This detail will not be further considered throughout this work, basically assuming an open set $\mathcal{X}_{\text{obs}}$ (and thus, by definition a closed set $\mathcal{X}_{\text{free}}$).

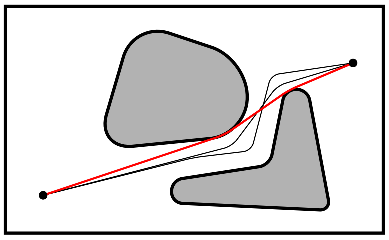
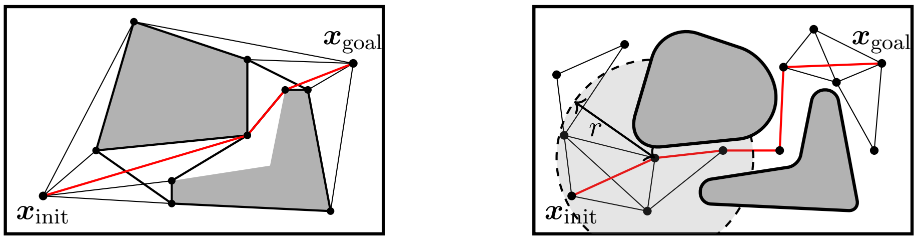


Figure 2.3: Optimal motion planning for closed $\mathcal{X}_{\text{obs}}$ requires to find a sequence $(\sigma_n)_{n \in \mathbb{N}}$, whose cost converges to the optimal cost, i.e., gets closer and closer to the boundary without touching it.

The two main directions to solve these problems are combinatorial motion planning and sampling-based motion planning. The difference is visualized in Figure

2.4. Combinatorial motion planning is based on the geometry of the problem and often uses particular environment properties such as visibility to determine connections which can contribute to optimal paths. While combinatorial motion planning normally finds optimal solutions (i.e., it often solves Problem 2) it does not scale well to more complex environments. As soon as rotations or non-polygonal obstacles are involved the analysis becomes highly complex or even impossible. The second branch, sampling-based motion planning, will be introduced next.


$\boldsymbol{x}_{\text{goal}}$
$\boldsymbol{x}_{\text{init}}$
r
$\boldsymbol{x}_{\text{goal}}$
$\boldsymbol{x}_{\text{init}}$

(a) Combinatorial (b) Sampling-based

Figure 2.4: Schematic difference between the two main branches of motion planning. Combinatorial motion planning normally finds optimal solutions, but becomes highly complex or impossible for more complex geometries and systems, while sampling-based motion planning introduces suboptimality, but scales much better to more complex problems.

2.2 Sampling-Based Motion Planning

Next, we introduce the nowadays most common choice for motion planning: sampling-based approaches. We show how sampling-based motion planners are composed of simpler components, which makes this paradigm very attractive due to its generality and extensibility.

The *collision checker* can be seen as a black box that tells us whether a configuration $\boldsymbol{x} \in \mathcal{X}$ is in collision or not. Normally such a collision checker will take as input a representation of the environment (e.g., a map) we are planning in, the geometry of the entity or entities that move in the environment and whose position are encoded in the configuration $\boldsymbol{x}$. Based on this information it returns whether the state is in collision or not. Basically, the collision checker can be seen as a way of probing the configuration space, checking whether $\boldsymbol{x}$ is part of $\mathcal{X}_{\text{free}}$ or $\mathcal{X}_{\text{obs}}$ since in many cases no explicit description of $\mathcal{X}$ is known. If such an explicit description is known, the collision checker can of course directly use that.

The *sampler* draws an $\boldsymbol{x}$ from $\mathcal{X}$. If possible it might also directly provide $\boldsymbol{x} \in \mathcal{X}_{\text{free}}$, which again normally only works if $\mathcal{X}_{\text{free}}$ is explicitly known. Otherwise the sampler can be combined with a collision checker by drawing samples until one is not in collision. Some more insights of samplers in motion planning can be found in Section 2.5.3.

The *steering function* (sometimes also called local planner) is used to connect to poses in free spaces. For example in Euclidean space simple linear interpolation gives the shortest path between two points. The situation becomes more complicated if differential constraints are introduced (see Section 2.3). For some systems there exist ways of quickly generating the optimal (regarding some criterion) paths in free space. An overview over such systems and approaches in this area is given in Section 3.3. The steering function is closely related to the *distance function* (we denote it as dist), which defines the distance between two poses in free spaces. Normally the distance function and steering function can be seen as two sides of the same coin, where the distance function returns the path length of the path returned by the steering function.

All of these components are parts of most sampling-based motion planners. The way in which these are used to find a path however differs. Let us next consider the case of graph-based motion planners.

Graph-based planning

Generally these approaches are used to abstract the motion planning problem as a graph problem. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is made up of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$ that connect the vertices. The edges can be both directed or undirected, depending on the underlying problem. In the context of motion planning the vertices normally represent poses in the configuration space $\mathcal{X}$. To build the graph often a *connection strategy* has to be specified. Popular approaches are k-nearest neighbors or connecting with all samples closer than a certain radius r using some distance function dist, that measures the distance between two poses. In addition to the vertices, the edges have to be checked for collision as well. For that either a special collision checker for paths can be used or the same collision checker is used by placing samples along the path at some sufficient resolution. For most applications such an approach is reasonable, but there might be cases in which a specific path checker could be more efficient or accurate.

Once the graph is built, a graph search is performed to find a sequence of vertices that represent the path between the start and goal vertex. Such a graph problem is also known as single-pair problem (i.e., single-source and single-destination). In fact there is no asymptotically faster algorithm to solve the single-pair problem than to solve the single-source problem [3]. Hence, the popular algorithm for finding single-source shortest paths in graphs by Dijkstra can be used [4]. While Dijkstra's algorithm solves the single-source problem, another algorithm, A*, aims specifically at solving the single-pair problem by guiding the search using a heuristic function

[5]. Note that asymptotically both algorithms still have the same complexity. Still, in practice A* is used as it gives shorter run times on average, although in the worst case it might perform worse than Dijkstra's algorithm.

Due to the similarity of the two algorithms we only explicitly state a general graph search algorithm (see Algorithm 1) and highlight how it covers both Dijkstra and A*. This forward search is based on maintaining a priority queue that keeps track of nodes to be explored and a cost to reach these nodes given the previously explored nodes. The idea is that in each step we take the node which at this point has the lowest cost to reach it. Since all other nodes that will be explored afterwards will have a higher cost there is no way we find a better path to this node later. The set $\mathcal{V}_{\text{closed}}$ keeps track of all the nodes for which the optimal cost has already been found. Note that this algorithm assumes non-negative edge costs, which is normally the case, especially in the context of motion planning.

In lines 2 and 3 the algorithm initializes the set $\mathcal{V}_{\text{closed}}$ and the priority queue $\mathcal{Q}$. The function f returns the value that will be used to sort this queue. In the initial step, only our starting node $\boldsymbol{x}_{\text{init}}$ is added. In each step we get the node with the lowest cost from the queue (line 6) and add it to the set $\mathcal{V}_{\text{closed}}$ indicating that the shortest path to this node has been found and will not change in the future. Lines 8–10 check if this node is a goal node and end the algorithm. Since we are growing a tree from the starting node $\boldsymbol{x}_{\text{init}}$ it is straightforward to return the edges which led from the initial node to the goal node, which represent the final shortest path (of course we have to store the predecessor of each node, which is not explicitly stated in this algorithm). Finally, if the node is not the goal, we continue in lines 11–13, by expanding the node (i.e., walking along each edge that originates from it). Nodes that are already closed are skipped and for others we check whether the cost via the previous node is shorter than the current estimate (line 12). The call to UpdateQueue also updates the predecessor in case of finding a shorter path than the previously stored.

The difference between Dijsktra's algorithm and A* is the function f that is used to sort the queue. In Dijsktra's algorithm it is simply the cost to reach a node from the starting node. In this case f is given by

$$f(\boldsymbol{x}_{\text{from}}, \boldsymbol{x}_{\text{to}}) = c_{\mathcal{G}}(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_{\text{from}}) + c_{\mathcal{G}}(\boldsymbol{x}_{\text{from}}, \boldsymbol{x}_{\text{to}}) \tag{2.6}$$

where $c_{\mathcal{G}}(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_{\text{from}})$ is the current cost-to-go in the graph[1] (i.e., the sum of edges to reach $\boldsymbol{x}_{\text{from}}$ from $\boldsymbol{x}_{\text{start}}$) and $c_{\mathcal{G}}(\boldsymbol{x}_{\text{from}}, \boldsymbol{x}_{\text{to}})$ returns the cost of the edge from $\boldsymbol{x}_{\text{from}}$ to $\boldsymbol{x}_{\text{to}}$. In A* we additionally use the information about the goal state to estimate

[1]Note that we use $c_{\mathcal{G}}$ to differentiate it from c as defined in Definition 7. While the latter is normally used to determine the edge cost between two neighboring vertices, the first is the cost in the graph $\mathcal{G}$ as defined by the sum of edges.

the remaining cost to the goal. f is then given by

$$f(\boldsymbol{x}_{\text{from}}, \boldsymbol{x}_{\text{to}}) = c_{\mathcal{G}}(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_{\text{from}}) + c_{\mathcal{G}}(\boldsymbol{x}_{\text{from}}, \boldsymbol{x}_{\text{to}}) + h(\boldsymbol{x}_{\text{to}}, \boldsymbol{x}_{\text{goal}}) \tag{2.7}$$

where $h(\boldsymbol{x}_{\text{to}}, \boldsymbol{x}_{\text{goal}})$ is a heuristic that tries to estimate the cost from $\boldsymbol{x}_{\text{to}}$ to $\boldsymbol{x}_{\text{goal}}$. Thus, A* basically estimates the cost of the full path and explores the space based on that cost, while Dijsktra explores the space simply expanding from the starting node using no information about the goal.

Finally, to ensure that the returned result from A* is still an optimum the heuristic has to be admissible. For that it must satisfy

$$h(\boldsymbol{x}_{\text{to}}, \boldsymbol{x}_{\text{goal}}) \leq c_{\mathcal{G}}(\boldsymbol{x}_{\text{to}}, \boldsymbol{x}_{\text{goal}}) \tag{2.8}$$

or in other words the heuristic must underestimate the true cost. It should be noted that the use of a heuristic function is especially well suited for the use in motion planning problems. Here, often a suitable heuristic function is the distance in free space whereas the cost in the graph is the cost with obstacles and thus larger or equal to the heuristic.

As hinted at above, this graph-based framework can also be used to plan under differential constraints. A few special notions and considerations are important in this regard, which we introduce next.

Algorithm 1 Generic shortest path

```
1:  procedure ShortestPath($\mathcal{V}, \mathcal{E}, \boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}$)
2:      $\mathcal{V}_{\text{closed}} \leftarrow \{\}$
3:      $\mathcal{Q} \leftarrow$ PriorityQueue()
4:      Push($\mathcal{Q}, \boldsymbol{x}_{\text{init}}, f(\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{init}})$)
5:      while $\mathcal{Q}$ not empty do
6:          $(\boldsymbol{x}_{\text{current}}, c) \leftarrow$Pop($\mathcal{Q}$)
7:          $\mathcal{V}_{\text{closed}} \leftarrow \mathcal{V}_{\text{closed}} \cup \{\boldsymbol{x}_{\text{current}}\}$
8:          if $\boldsymbol{x}_{\text{current}}$ is $\boldsymbol{x}_{\text{goal}}$ then
9:              return Edges($\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{current}}$)
10:         end if
11:         for $\boldsymbol{x}_{\text{next}}$ in Expand($\boldsymbol{x}_{\text{current}}$)$\setminus\mathcal{V}_{\text{closed}}$ do
12:             UpdateQueue($\mathcal{Q}, \boldsymbol{x}_{\text{next}}, f(\boldsymbol{x}_{\text{current}}, \boldsymbol{x}_{\text{next}})$)
13:         end for
14:     end while
15:     return NoSolution
16: end procedure
```

2.3 Planning under Differential Constraints

Next, we introduce the topic of differentially constrained motion planning. First, we introduce the mathematical fundamentals that define this topic and afterwards we show how this kind of problem can be tackled within the sampling-based motion planning framework introduced in the previous section. We also touch upon a different approach to nonholonomic sampling-based motion planning (namely sampling in control space) but this will not be the focus of this work.

Definition 8. Let $\mathcal{U} \subseteq \mathbb{R}^M$ denote the *control space*. A single control in this space will be denoted by $\boldsymbol{u} \in \mathcal{U}$.

Note that in general the control space $\mathcal{U}$ can also be a function of the current state, i.e., $\mathcal{U}(\boldsymbol{x})$. Going forward we do not state this specifically, but also not exclude it as a possibility. Next, we can use this notion of control space to define a differentially constrained system.

Definition 9. Let Σ denote a general differentially constrained system, defined by its state space $\mathcal{X}$ and its control space $\mathcal{U}$. The dynamics of the system are defined for a given $\boldsymbol{u} \in \mathcal{U}$ and $\boldsymbol{x} \in \mathcal{X}$ by the *state transition function*

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}). \tag{2.9}$$

In the context of differentially constrained motion planning, we extend the notion of *feasible path* such that it additionally has to satisfy the state transition function of the system under consideration. It is also straightforward to formulate the differentially constrained feasible and differentially constrained optimal motion planning problem given these definitions, which we skip for brevity's sake. Next, we introduce a number of terms that characterize these systems.

Linear *Linear* systems are a special class of systems that can be defined by

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}. \tag{2.10}$$

Such systems are well studied in linear control theory [6]. The matrix $\boldsymbol{A}$ describes the influence of the current state on the next state and the matrix $\boldsymbol{B}$ describes the relation between the control input $\boldsymbol{u}$ and $\boldsymbol{x}$. Finally, the term $\boldsymbol{c}$ describes the *drift* of the system, which is a concept introduced below.

Kinodynamic *Kinodynamic* systems are systems that involve differential constraints of the second order. The term was introduced by Donald [7]. Note that while the form (2.9) does not show a second order derivative, kinodynamic systems can still be represented by this formulation. This is common in control theory, in which the derivative of one state is for example simply given by another state. Also note that kinodynamic and linear are two separate system properties.

Nonholonomic A system is called *nonholonomic* if it is not completely integrable. The mathematical concept was first introduced by Ferrers [8] and was introduced in the context of robot motion planning by Laumond [9]. A more recent introduction into the concept is provided by LaValle [1]. One easy way of identifying nonholonomic systems is given by Sussmann [10]. Systems which have less controls then state dimensions are called nonholonomic, if even though less control directions exist, it can be moved in all directions by combining multiple actions. The prime example for that are car-like systems, for which some form of parallel parking maneuver is required to move it sideways.

Control-affine A common nonlinear class of systems are *control-affine* systems which can be described by

$$\dot{\boldsymbol{x}} = \boldsymbol{g}_0(x) + \sum_{i=1}^{M} \boldsymbol{g}_i(\boldsymbol{x})u_i \tag{2.11}$$

where the term $\boldsymbol{g}_0(x)$ describes the drift (see below).

Driftless A *driftless* system is one, for which there always exists some input $\boldsymbol{u}$ such that $\dot{\boldsymbol{x}} = 0$. For example as long as $\boldsymbol{u} = 0 \in \mathcal{U}$, $\boldsymbol{c} = 0$ and $\boldsymbol{g}_0 = 0$ imply that a linear system and control-affine system are driftless, respectively. Conversely a system that is not driftless, inhibits *drift*, which intuitively means that the systems state will change no matter the provided input $\boldsymbol{u}$.

As we see later in Chapter 4, an important concept related to planning under differential constraints is the reachable set.

Definition 10. Let $\mathcal{R}(\boldsymbol{x}, \delta)$ defined by

$$\mathcal{R}_{\text{dist}}(\boldsymbol{x}, \delta) = \{\boldsymbol{y} \in \mathcal{X} \mid \text{dist}(\boldsymbol{x}, \boldsymbol{y}) \leq \delta\} \tag{2.12}$$

be the *δ-limited reachable set* of a system Σ.

In all the cases that we consider, the steering function dist that defines this set, can be considered as both the time as well as the path length (which are the same for systems moving with unit velocity). In these cases, the reachable set contains all the points that can be reached in time not greater than δ or with paths not longer than δ.

Note that for Euclidean systems with linear interpolation the δ-limited reachable sets will simply be the Euclidean balls of some radius. Since time-limited reachable sets and distance-limited balls are equivalent in the context of the thesis we use $\mathcal{B}_{\text{dist}}(\boldsymbol{x}, \delta)$ and $\mathcal{R}_{\text{dist}}(\boldsymbol{x}, \delta)$ interchangeably throughout this work[1]. If we talk explicitly about some form of balls or reachable sets we replace dist with an identifier for the system under consideration. For example in the Euclidean case we use $\mathcal{B}_2$.

[1]We use $\mathcal{R}$, if we want to stress the applicability to systems under differential constraints, and $\mathcal{B}$ if we talk about Euclidean balls.

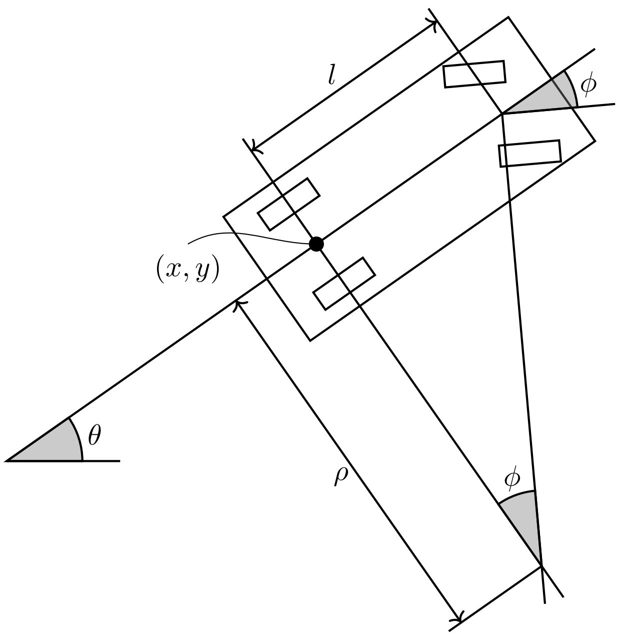


Figure 2.5: Car-like kinematics. The position of the car is defined by x and y, while the heading is defined by θ. The input $u_\phi = \phi$ defines the steering angle (i.e., the angle of the wheels) and the input u_s the forward speed of the car (not seen in the image). Based on the axis length l of the car and steering angle ϕ, a turning radius $\rho = l/\tan\phi$ can be calculated. (Image reproduced from LaValle [1])

2.3.1 Car-like kinematics

We now describe car-like kinematics, which is a nonholonomic system that is both of practical use and well analyzed. Due to these properties we use it as an exemplary system for the experiments of Section 5. In general the kinematic constraints of the system can be formulated as

$$\begin{aligned} \dot{x} &= u_\text{s} \cos\theta \\ \dot{y} &= u_\text{s} \sin\theta \\ \dot{\theta} &= \frac{u_\text{s}}{\rho} u_\phi. \end{aligned} \tag{2.13}$$

The input space $\mathcal{U}$ is two dimensional with u_s describing the speed and u_ϕ describing the steering angle. Figure 2.5 visualizes the system.

Many special cases can be defined for this kind of system by limiting the control set $\mathcal{U}$ [1]. The Dubin's car is obtained by limiting u_s to $\{0, 1\}$ and u_ϕ to $[-1, 1]$. Thus we can only go forward or stand still. The system is named after Lester Dubin's, who derived the optimal paths for this system [11]. A closely-related and relevant system for this work will be the Reeds-Shepp car [12], which is obtained by limiting u_{s} to $\{-1, 0, 1\}$ and u_ϕ to $[-1, 1]$. Basically, it is an extension of Dubin's car to also allow backwards driving.

Note that the system can be reformulated as a control-affine system by defining a new control $\tilde{u}_\phi = u_\phi u_{\mathrm{s}}$, with the same range as u_ϕ. The whole system is then given by

$$\begin{aligned} \dot{x} &= u_{\mathrm{s}} \cos\theta \\ \dot{y} &= u_{\mathrm{s}} \sin\theta \\ \dot{\theta} &= \frac{\tilde{u}_\phi}{\rho} \end{aligned} \tag{2.14}$$

and thus we can specify the control vectors $\boldsymbol{g}$ as

$$\boldsymbol{g}_{\mathrm{s}} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} \quad \boldsymbol{g}_\phi = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\rho} \end{bmatrix}. \tag{2.15}$$

For the Reeds-Shepp system exists an optimal steering function that given two poses $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ gives the control, path and path distance between the two poses. Figure 2.6 shows a few paths for the Reeds-Shepp car. The original work that proved the optimality of the paths was done by Reeds and Shepp [12]. A worked out example, that further decreases the number of required paths is provided by [10]. Furthermore another in-depth description of the problem (as well as related problems) is provided by Soueres [13].

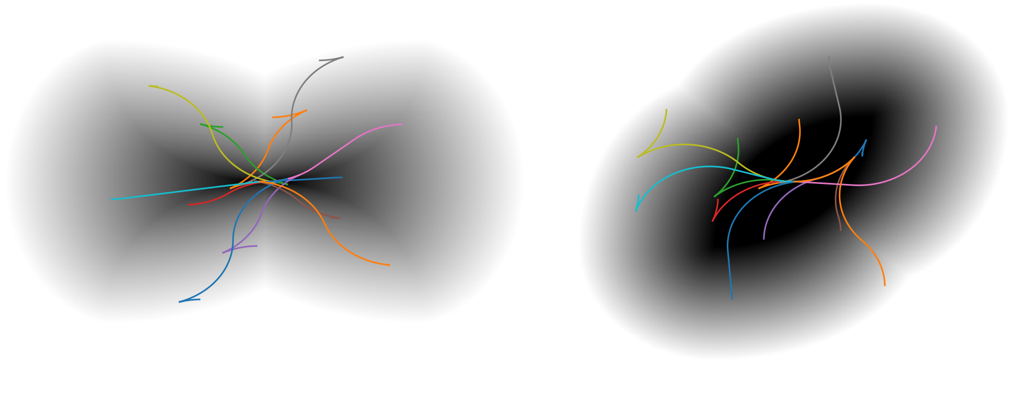

(a) $\theta_{\text{goal}} = 0$ (b) $\theta_{\text{goal}} = \pi/2$

Figure 2.6: Distance function and paths for the Reeds-Shepp case. The initial pose $\boldsymbol{x}_1 = [0, 0, 0]$ for all paths, while the end pose is constrained to $\boldsymbol{x}_2 = [x_2, y_2, 0]$ in (a) and $\boldsymbol{x}_2 = [x_2, y_2, \pi/2]$ in (b). It can be seen how the steering function in front of the car gives a small distance, while parallel to the car, parallel parking maneuvers are required and thus the distance is greater.

2.4 Sampling-Based Planning Algorithms

This section describes the utilized algorithms for the subsequent experiments. Different variations to each algorithm exist, hence, we highlight the nuances which are important to the deterministic behavior and the subsequent analysis. The theoretical extension of these algorithms to differentially constrained motion planning is provided by Schmerling [14, 15].

2.4.1 Probabilistic roadmap

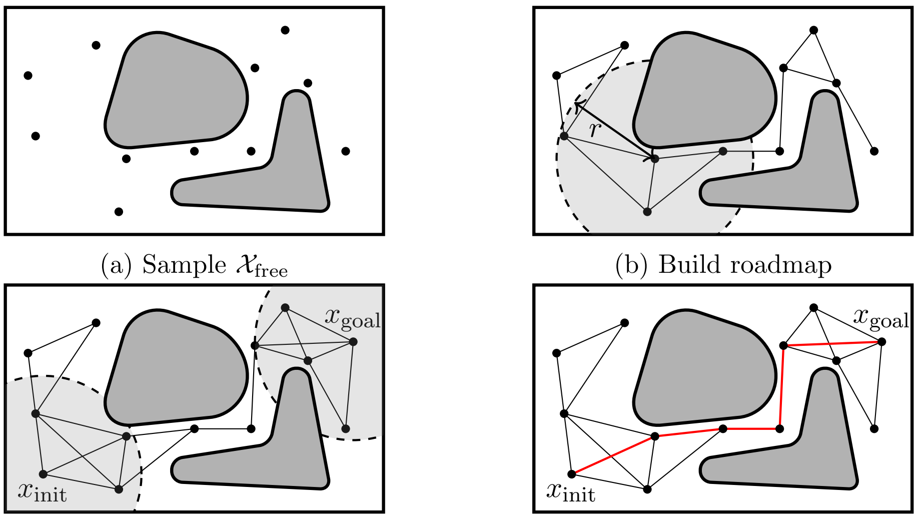
The *probabilistic roadmap* planner (PRM) is the prime example of sampling-based motion planning. It was introduced by Kavraki in 1996 [16]. The algorithm itself is listed in Algorithm 2. In general it is a straightforward realization of the principles introduced in Section 2.2.

The algorithm shown here is a simplified version of the originally proposed algorithm. The original version also includes steps to add additional samples in difficult spots, which has been omitted for easier analysis and to conform to the deterministic framework. Similar sampling-biasing techniques are proposed a lot in the literature [16, 17, 18, 19]. Similar simplified versions of PRM have been utilized by Janson [20] and Schmerling [15]. In addition similar to the original paper, we split up the algorithm into a learning phase and query phase. The idea here is use the planner in a multi-query way: first build a roadmap and then reuse it for multiple queries. In general it is straightforward to use the algorithm in a single query context by simply combining the two steps.

Let us next walk through the algorithm once. A visualization of the algorithm is shown in Figure 2.7. The algorithm first builds the set of vertices of the roadmap according to some termination criterion (Figure 2.7(a), line 2). The first loop in lines 4–10 creates the roadmap. For each vertex $\boldsymbol{x}$ we find the connection candidates via the call to Near($\boldsymbol{x}$). This function defines the aforementioned connection strategy. Once the close neighbors have been found, we check whether the connection from $\boldsymbol{x}$ to its neighbor candidate $\boldsymbol{y}$ is collision-free. If it is, we add the edge to the roadmap (line 7). The final roadmap after adding all the edges is shown in Figure 2.7(b). After the full roadmap is built, it should be stored in a multi-query scenario. From line 11 begins the query phase. First, the start $\boldsymbol{x}_{\text{init}}$ and goal $\boldsymbol{x}_{\text{goal}}$ are added to the set of vertices after which the same connection loop is used as before, but only for the newly added vertices (Figure 2.7(c), lines 12–18). Finally, the graph is fully defined and we can use the shortest path algorithm that we have previously introduced in Section 2.2 (Figure 2.7(d), line 19).

Algorithm 2 Probabilistic Roadmap

```
 1: procedure ProbabilisticRoadmap
 2:     $\mathcal{V} \leftarrow$SampleFree                                  }
 3:     $\mathcal{E} \leftarrow \emptyset$                                  }
 4:     for $\boldsymbol{x} \in \mathcal{V}$ do                             }
 5:         for $\boldsymbol{y} \in$ Near($\boldsymbol{x}$) do              }
 6:             if CollisionFree($\boldsymbol{x}, \boldsymbol{y}$) then     }  Learning phase
 7:                 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\boldsymbol{x}, \boldsymbol{y})\}$   }
 8:             end if                                                  }
 9:         end for                                                     }
10:     end for                                                         }
11:     $\mathcal{V} \leftarrow \mathcal{V} \cup \{\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}\}$   }
12:     for $\boldsymbol{x} \in \{\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}\}$ do   }
13:         for $\boldsymbol{y} \in$ Near($\boldsymbol{x}$) do              }
14:             if CollisionFree($\boldsymbol{x}, \boldsymbol{y}$) then     }
15:                 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\boldsymbol{x}, \boldsymbol{y})\}$   }  Query phase
16:             end if                                                  }
17:         end for                                                     }
18:     end for                                                         }
19:     return ShortestPath($\mathcal{V}, \mathcal{E}, \boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}$)   }
20: end procedure
```



(a) Sample $\mathcal{X}_{\text{free}}$ (b) Build roadmap (c) Add start and goal (d) Solve graph problem

Figure 2.7: Visualization of the probabilistic roadmap algorithm solving a query. In (a) the space is sampled (in this case the termination criterion is $n_{\text{valid}} = 11$), in (b) the edges are computed (in this case the connection strategy is a fixed radius r), in (c) the start $\boldsymbol{x}_{\text{init}}$ and goal $\boldsymbol{x}_{\text{goal}}$ are added using the same connection strategy (different would be possible too, e.g., larger radius). Finally, a shortest path algorithm like A* is employed to find the shortest path in the roadmap as shown in (d).

For the comparison to other algorithms it is important to note the possible termination conditions that PRM allows:

- fixed number of sample attempts n_{all}
- fixed number of valid samples n_{valid}
- fixed time to build the roadmap

The first two are straightforward, first we sample enough vertices and then the roadmap is built and the graph search is performed. For the last case the situation is more difficult, since we do not know a priori how long each state will take. OMPL [21] provides a multithreaded solution though in which vertices and edges are consistently added to the graph in one thread, while a second thread consistently runs the shortest path search to find a solution. This way both threads are stopped once the termination time is reached and the best solution found so far is returned.

The algorithm was originally proposed as a multi-query algorithm since one could build the roadmap once and reuse it for multiple queries as the collision checking has already been performed for the whole space, thus being able to solve further queries quickly without the costly use of the collision checker.

A number of different variants have been introduced for this algorithm. LazyPRM [22] tries to minimize the planning time by skipping collision checks while building the roadmap and aims at single-query application. PRM* [2] is a version of PRM that provably converges to the optimal cost. The main difference between PRM and PRM* are conditions on the connection strategy and steering function that ensure asymptotic optimality. The steering function must be optimal and for the connection strategy a connection radius

$$r(n) \geq \gamma_{\mathrm{PRM}} \left(\log(n)/n\right)^{1/D} \tag{2.16}$$

with $\gamma_{\mathrm{PRM}} > 2(1+1/D)^{1/d} \left(\mu(\mathcal{X}_{\mathrm{free}})/\zeta_d\right)^{1/D}$ is required, where D is the dimension of the space, μ denotes the volume of a set and ζ_D is the volume of the D-dimensional ball in Euclidean space. Equivalently in the case of a k-nearest neighbors connection strategy the condition on k can be formulated as

$$k(n) \geq k_{\mathrm{PRM}} \log(n) \tag{2.17}$$

with $k_{\mathrm{PRM}} = e(1 + 1/D)$. Note that strictly speaking these conditions only hold for the Euclidean case. The intuition behind these formulas is that the connection radius must not shrink faster than the dispersion of the space since that would cause a disconnected roadmap. Extensions to differentially constrained systems are provided by Schmerling [14, 15].

2.4.2 Fast marching tree

Another planning algorithm called *fast marching tree* (FMT*) that is based on dynamic programming was introduced by Janson [23]. The algorithm tries to minimize unnecessary calls to the collision checker and is thus faster for single queries when compared to probabilistic roadmaps. The algorithm can also be seen as a version of the shortest path algorithm that is more tightly coupled with the motion planner.

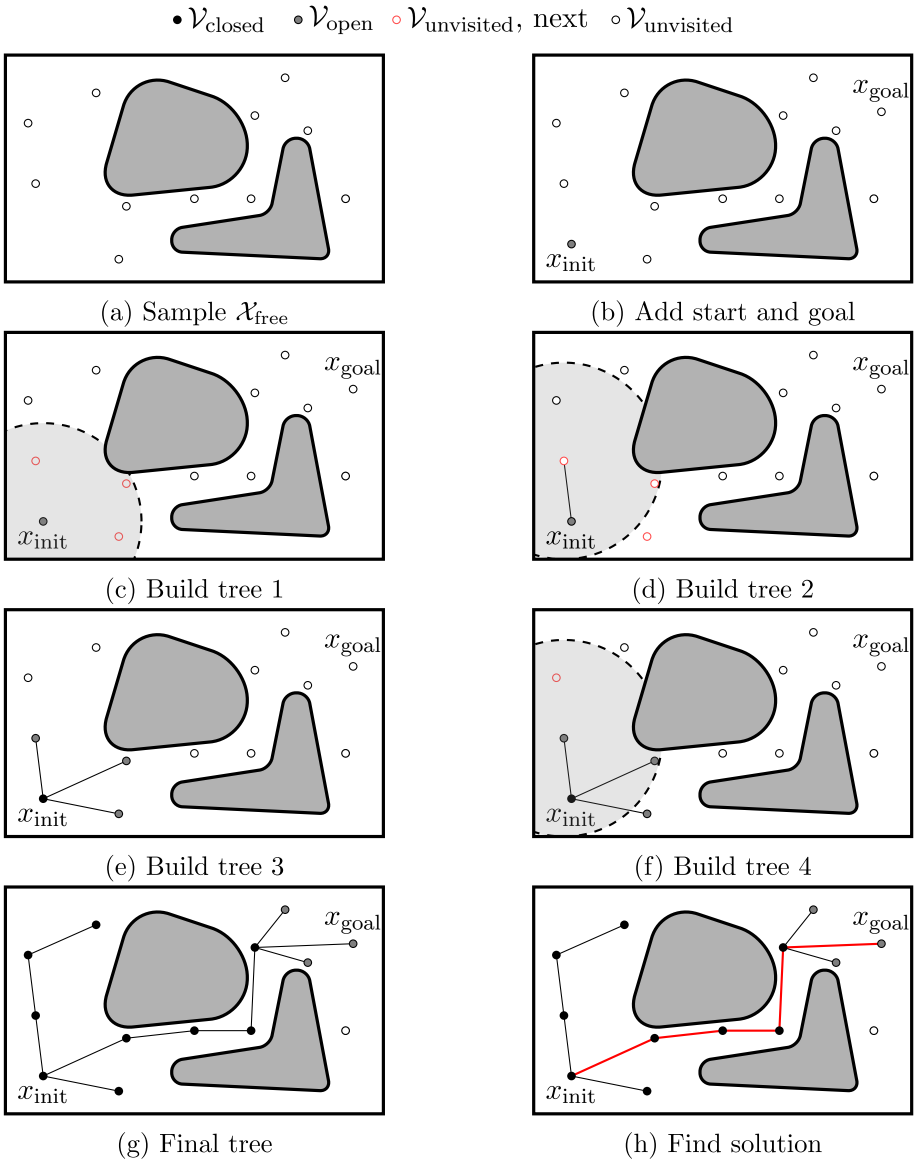
FMT* is shown in Algorithm 3 and Figure 2.8 illustrates the most important steps. The algorithm is initialized by sampling a number of valid samples (Figure 2.8(a), line 2), furthermore the start and goal vertices are added to the set of vertices $\mathcal{V}$. The algorithm maintains three sets: $\mathcal{V}_{\mathrm{open}}$, for which we know the shortest path from $\boldsymbol{x}_{\mathrm{init}}$, $\mathcal{V}_{\mathrm{unvisited}}$, which contains all vertices to which we do not know the fastest path yet, and $\mathcal{V}_{\mathrm{closed}}$ for which we have checked all possible successors. Initially only $\boldsymbol{x}_{\mathrm{init}}$ is put into $\mathcal{V}_{\mathrm{open}}$ and all other vertices are put into $\mathcal{V}_{\mathrm{unvisited}}$ (Figure 2.8(b), lines 4–6). Next, we keep picking the vertex in $\mathcal{V}_{\mathrm{open}}$ that has the lowest cost right now (Figures 2.8(c) and 2.8(f), line 7 and 19). This is the equivalent of line 6 in Algorithm 1. We refer to this vertex as $\boldsymbol{z}$. In the next step we find all the vertices that are, according to the connection strategy, close to $\boldsymbol{z}$ (see circle and red vertices in Figure 2.8(c) and 2.8(f), line 9). For each of these vertices $\boldsymbol{y}$ we check, which is the closest vertex to it in $\mathcal{V}_{\text{open}}$ (Figure 2.8(d), line 10) and connect to it, if the path between them is collision-free (lines 11–14). Note that while we got to vertex $\boldsymbol{y}$ via $\boldsymbol{z}$, we are not necessarily connecting $\boldsymbol{y}$ to $\boldsymbol{z}$ (it naturally often happens, but nothing in the algorithm requires it).

One special thing to note about FMT* is that in lines 10–15, we only check for the closest vertex. If there is a collision in-between, and the actual best connection would have been $\boldsymbol{z}$, we will never come back to this vertex to attempt this connection. Strictly speaking, because of this, the algorithm can never provide non-probabilistic completeness due to this connection strategy [1]. Asymptotically though the algorithm retains completeness as noted by the authors [20]. The exact conditions that need to come together such that a suboptimal connection occurs are also provided by the authors.

Algorithm 3 Fast Marching Tree

```
 1: procedure FastMarchingTree
 2:     $\mathcal{V} \leftarrow \{\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}\} \cup \textsc{SampleFree}$
 3:     $\mathcal{E} \leftarrow \emptyset$
 4:     $\mathcal{V}_{\text{unvisited}} \leftarrow \mathcal{V} \setminus \boldsymbol{x}_{\text{init}}$
 5:     $\mathcal{V}_{\text{open}} \leftarrow \boldsymbol{x}_{\text{init}}$
 6:     $\mathcal{V}_{\text{closed}} \leftarrow \emptyset$
 7:     $\boldsymbol{z} \leftarrow \boldsymbol{x}_{\text{init}}$
 8:     while $\boldsymbol{z}$ is not $\boldsymbol{x}_{\text{goal}}$ do
 9:         for $\boldsymbol{x} \in \textsc{Near}(\boldsymbol{z}) \cap \mathcal{V}_{\text{unvisited}}$ do
10:             $\boldsymbol{y}_{\min} \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{V}_{\text{open}} \cap \textsc{Near}(\boldsymbol{y})} \{c_{\mathcal{G}}(\boldsymbol{x}_{\text{init}}, \boldsymbol{y}) + c(\boldsymbol{y}, \boldsymbol{x})\}$
11:             if CollisionFree($\boldsymbol{y}_{\min}, \boldsymbol{x}$) then
12:                 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\boldsymbol{y}_{\min}, \boldsymbol{x})\}$
13:                 $\mathcal{V}_{\text{open,new}} \leftarrow \mathcal{V}_{\text{open,new}} \cup \{\boldsymbol{x}\}$
14:                 $\mathcal{V}_{\text{unvisited}} \leftarrow \mathcal{V}_{\text{unvisited}} \setminus \{\boldsymbol{x}\}$
15:             end if
16:         end for
17:         $\mathcal{V}_{\text{open}} \leftarrow (\mathcal{V}_{\text{open}} \cup \mathcal{V}_{\text{open,new}}) \setminus \{\boldsymbol{z}\}$
18:         $\mathcal{V}_{\text{closed}} \leftarrow \mathcal{V}_{\text{closed}} \cup \{\boldsymbol{z}\}$
19:         $\boldsymbol{z} \leftarrow \arg\min_{\boldsymbol{y} \in \mathcal{V}_{\text{open}}} \{c_{\mathcal{G}}(\boldsymbol{x}_{\text{init}}, \boldsymbol{y})\}$
20:     end while
21:     return Path($\mathcal{E}, \boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}$)
22: end procedure
```

[1]This could possibly be fixed by checking all nodes in increasing distance, but this option was not explored by the original authors possibly for efficiency reasons.



● $\mathcal{V}_{\text{closed}}$ ● $\mathcal{V}_{\text{open}}$ ○ $\mathcal{V}_{\text{unvisited}}$, next ○ $\mathcal{V}_{\text{unvisited}}$
x_{goal}
x_{init}
(a) Sample $\mathcal{X}_{\text{free}}$
(b) Add start and goal
(c) Build tree 1
(d) Build tree 2
(e) Build tree 3
(f) Build tree 4
(g) Final tree
(h) Find solution

Figure 2.8: Visualization of the fast marching tree algorithm solving a query. See text for description.

Compared to PRM* the algorithm can only terminate based on a fixed number of samples (either n_{all} or n_{valid}). No straightforward way exists so far to add more samples to an already solved query. Thus for time-based comparisons one would have to find the number of samples that approximately takes a certain amount of time. Also, the same connection strategies as introduced before can be used for

FMT*.

2.4.3 State lattice-based approaches

A slightly different approach to sampling-based motion planning that does not follow the random sampling paradigm, are so called state lattice-based approaches. In general these approaches use a set of motion primitives $\mathcal{P}$ and concatenates these to find a solution. By carefully choosing these motion primitives, planners can have many desirable properties. Since such motion primitives can be used inside of planners in quite a few different ways we provide no algorithm here, but rather just explain the different design considerations that arise for such planners. We provide a specific algorithm in Section 4.3 in the context of showing resolution completeness.

Two main directions exist for these planners. Tree-based lattice planners and graph-based lattice planners. The first will grow a tree of motion primitives starting from $\boldsymbol{x}_{\text{init}}$ and tries to connect the tree to the goal $\boldsymbol{x}_{\text{goal}}$. The latter builds a graph in which, similar to PRM*, a graph search algorithm can be used to find a solution.

Both of these approaches have to deal with the problem of connecting the tree or graph to the goal state. In case of a goal region, no approximation is required and we can simply check whether vertices are inside of it. For goal states, two options are possible. First, we can simply find the vertices close to the goal and approximate it like this (which implicitly is the same as a goal region) or we can use a steering function (if available) to try to find exact connections from the goal state to the tree or roadmap.

Motion primitive sets in general lead to *dense* sets of points. The problem with this is that the motion primitives do not efficiently explore the space. Thus, to enable quick exploration of the space, the tree should be pruned if a vertex in some proximity already exists. Figure 2.9 compares the resulting trees when pruning at different resolutions. Clearly by pruning the tree, the same number of vertices lead to a larger covered area. In general such dense motion primitive sets lead to trees and not graphs since the motion primitives do not lead on top of other end points. Hence, such primitive sets are mostly suitable for tree-based planners. Otherwise approximating the edge connections is a valid alternative at some resolution. Such an approach leads to discontinuous paths, which may still be viable in practice, depending on the whole motion planning stack[1].

An alternative to motion primitives that lead to dense sets are motion primitives that all end on some predefined grid. Such a grid can be both uniform and nonuniform and by designing the motion primitives to start and end on the grid no pruning scheme is required. The grid defines the density of the roadmap. For this approach connecting both start and goal is not straightforward and require either a

[1]Generally a low-level controller is normally used to track trajectories and can deal to some extent with discontinuities.

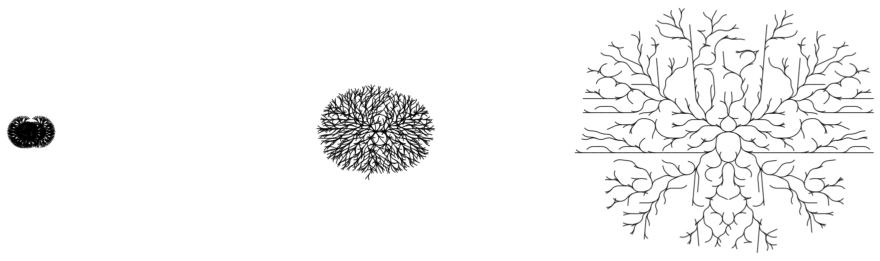

(a) No pruning (b) $s_{\text{grid}} = 2\,\text{m}$ (c) $s_{\text{grid}} = 5\,\text{m}$

Figure 2.9: Comparison of resulting trees when pruning it at different resolutions. In each case 1000 vertices are generated, while the tree on the left is not pruned at all, the others only allow a single vertex in a grid cell of size s_{grid} (regardless of heading). Clearly a huge difference in the covered area can be observed (same scale for all three trees).

steering function or approximation. The advantage of this approach is that since by concatenating the primitives the resulting endpoints will always be part of the grid and thus a dense graph can be built without any approximations. Therefore, such motion primitives are well suited for graph-based planners.

Figure 2.10 compares these two paradigms of motion primitive sets.

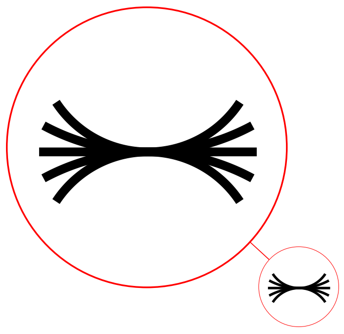
2.4.4 Other algorithms

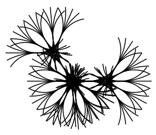
Finally, we want to give a short overview over other planning algorithms, which are not as relevant to this work, but are very popular in practice. We will not give explicit algorithms for these approaches, but only summarize the key idea behind the approach.

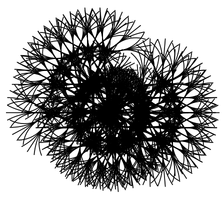
The first algorithm that we want to shortly introduce is called *rapidly-exploring random tree* (RRT) [24]. As the name suggests, this planner is based on building a tree starting from the current state and is well suited for differentially constrained motion planning. The algorithm works by randomly sampling the space and then use a steering function that ignores obstacles to steer from the closest state in the tree towards the sampled state. But instead of fully connecting to it, only a short step towards the right direction is made. This is either done from both the start and goal pose and attempts are made to connect the two trees, or a single tree is grown until it reaches a goal set. Many variants of this planner have since been proposed. Famous examples are the bidirectional version in Euclidean space, called RRTConnect [25] and the asymptotically optimal version RRT* [2] which rewires the tree if shorter paths are found.

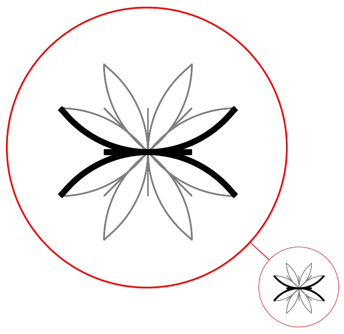
RRT* is a famous example for an *anytime* algorithm, which refers to the fact that the algorithm finds better solutions the longer it runs. Multiple planners extend RRT* to faster converge to the optimum solution, by constraining the sampling once an initial solution is found, like Informed RRT* [18] and *batch informed trees* (BIT*) [26].

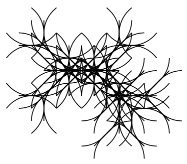
An interesting classification can be made between RRT*, BIT* and FMT*. Whereas RRT* is an anytime algorithm, that works by adding samples one at a time, BIT* is an informed batch variant that adds and processes multiple samples at a time. FMT* on the other hand is not an anytime algorithm as it only samples once and builds the whole search tree based on these samples. Hence, a classification of planners can be made depending on the way they sample the space.


(a) Dense primitives


(b) Dense 250 vertices


(c) Dense 2500 vertices


(d) Grid primitives


(e) Grid 250 vertices

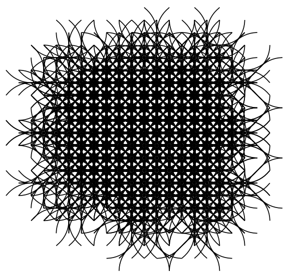

(f) Grid 2500 vertices

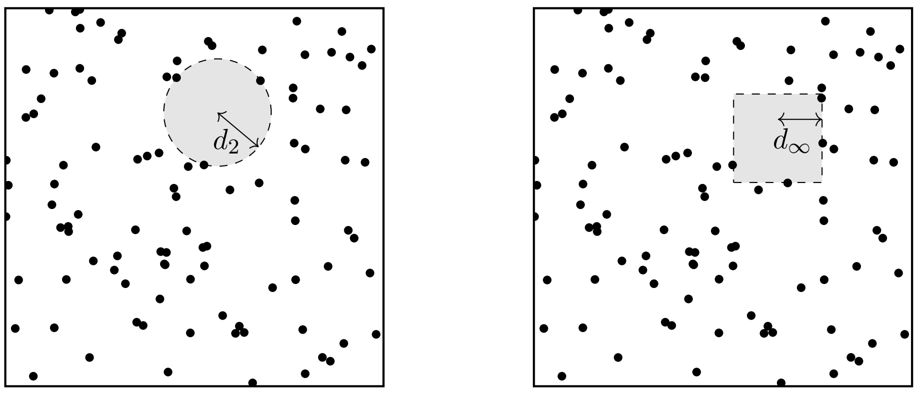
Figure 2.10: Comparison of dense and grid-based motion primitive sets. The left column shows the primitives, the middle columns shows the graph after 250 vertices (including the end vertices) and the right column after 2500 vertices. Clearly the grid-based primitives naturally lead to a well-connected graph with all vertices lying on the grid, while with the dense set, concatenating the motion primitives does not lead on top of previous end points and thus a dense tree is formed. All six figures are drawn at the same scale, which shows that the grid-based primitives naturally cover a larger area, with the same number of vertices. Another difference is that spatially close vertices in the tree might only be connected via the origin of the tree, while in the graph all vertices are well connected.

2.5 Sampling Theory

Typically samplers for independent, identically distributed (i.i.d.) variables are designed to satisfy statistical tests. In the motion planning context another metric called dispersion is much more important as it directly relates to how well a set of samples covers a space.

In this section first the dispersion metric and discrepancy metric are defined in general terms and subsequently sampling sequences that optimize this metric asymptotically are introduced. Finally, the general idea of why these metrics are important in the context of sampling-based motion planning are introduced. A more thorough, theoretical introduction to these ideas will be given in Section 3.1. This section mostly follows LaValle's introduction to sampling theory [1].

2.5.1 Dispersion



(a) d_2 dispersion (b) d_∞ dispersion

Figure 2.11: Visualization of the dispersion metric for different distance functions. (a) shows the l_2 norm, while (b) shows the l_∞ norm. The actual dispersion of the point sets are denoted by d_2 and d_∞ respectively.

The original definition of *dispersion* can be found in Definition 6.2 by Niederreiter [27]

$$d_{\text{dist}} = \sup\{r > 0 | \exists \boldsymbol{x} \in \mathcal{X} \text{ with } \mathcal{B}_{\text{dist}}(\boldsymbol{x}, r) \cap \mathcal{S} = \emptyset\}. \tag{2.18}$$

Intuitively the dispersion of a set of samples $\mathcal{S}$ is the radius of the largest ball that does not contain a single sample. Dispersion is defined based on a distance function dist, which measures the distance between two points in the space. Figure 2.11 visualizes the dispersion metric for the Euclidean norm and infinity norm. In practice d_∞ is often easier to handle analytically. Clearly though, it is possible to establish a relationship between different dispersion metrics. The idea here is, that if the dispersion of one distance function is known, we can upper and lower-bound the dispersion of a different distance function by outer and inner-bounding the ball respectively. I.e., given a set $\mathcal{S}$ with known dist1-dispersion d_{dist1}, $\mathcal{B}_{\text{dist2}}(\boldsymbol{x}, f(d_{\text{dist1}})) \subseteq \mathcal{B}_{\text{dist1}}(\boldsymbol{x}, d_{\text{dist1}}) \subseteq \mathcal{B}_{\text{dist2}}(\boldsymbol{x}, g(d_{\text{dist1}}))\, \forall\, \boldsymbol{x} \in \mathcal{X} \Rightarrow f(d_{\text{dist1}}) \leq d_{\text{dist2}} \leq g(d_{\text{dist1}})$. For example in the case of d_2 and d_∞ dispersion this gives

$$d_\infty \leq d_2 \leq \sqrt{D} d_\infty. \tag{2.19}$$

One can imagine that while random samples have a decent dispersion (i.e., better than highly non-uniformly distributed samples), it is worse than the dispersion of sets that are placed in a deterministic uniform way. Figure 2.12 shows a range of sets, which visualizes this idea.

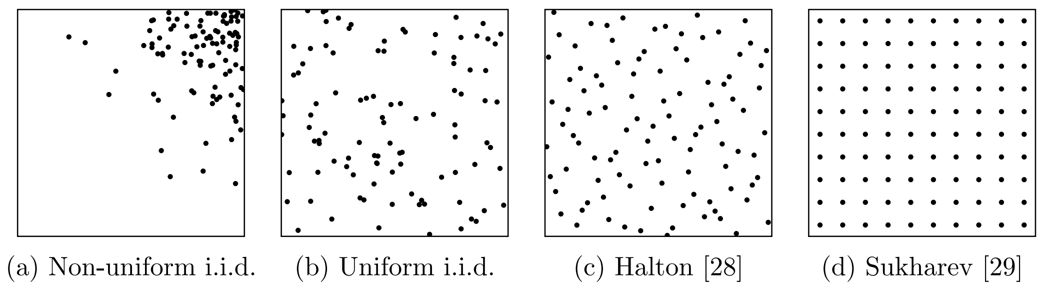


Figure 2.12: Qualitative comparison of the dispersion for different kinds of sampling sets. The last two sets are deterministic.

2.5.2 Discrepancy

Another important metric for a sample set $\mathcal{S}$ is discrepancy. It is defined by

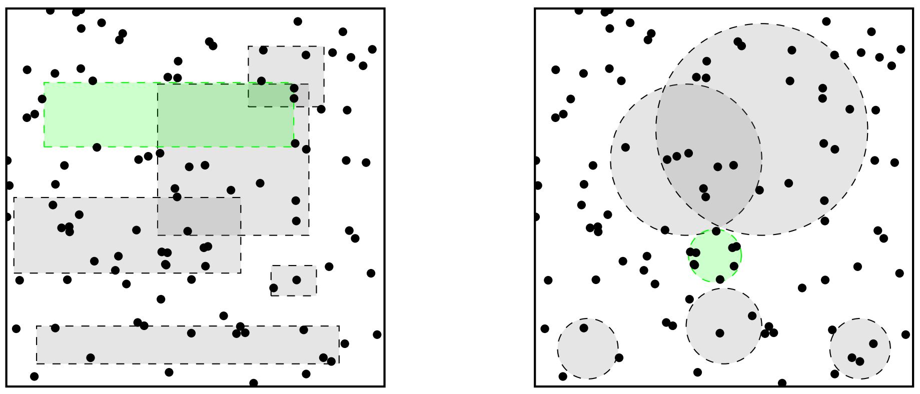
$$D_{\mathcal{R}} = \sup_{R \in \mathcal{R}} \left(\left| \frac{|\mathcal{S} \cap R|}{|\mathcal{S}|} - \frac{\mu(R)}{\mu(\mathcal{X})} \right| \right). \tag{2.20}$$

Similar to how dispersion is defined based on a distance function, discrepancy is defined based on a *range space* $\mathcal{R}$. A range space is a set of subsets of $\mathcal{X}$. Normally such a range space would be defined by some characteristic of the subsets, but also an explicit range space of some subsets of $\mathcal{X}$ would be admissible. Typical examples of range spaces would be axis-aligned rectangles

$$\mathcal{R}_{\text{aar}} = \{\mathcal{A} \,|\, \mathcal{A} = [a_1, b_1] \times \cdots \times [a_D, b_D], \mathcal{A} \subseteq \mathcal{X}\} \tag{2.21}$$

and Euclidean balls

$$\mathcal{R}_{\text{euc,balls}} = \{\mathcal{A} \,|\, \mathcal{A} = \mathcal{B}_2(\boldsymbol{x}, r) \text{ with } \boldsymbol{x} \in \mathcal{X}, r \in \mathbb{R}_{>0} \text{ s.t. } \mathcal{A} \subseteq \mathcal{X}\}. \tag{2.22}$$



(a) $\mathcal{R}_{\text{aar}}$ discrepancy (b) $\mathcal{R}_{\text{euc,balls}}$ discrepancy

Figure 2.13: Visualization of different range spaces. The gray shapes are samples from the ranges space, while the green sets visualize a possible candidate that might define the discrepancy $D_{\mathcal{R}}$. Note how these are either big with few samples (a) or small with many samples (b).

Finally, $\mu(\cdot)$ denotes the volume of a set. In the literature discrepancy often uses axis-aligned rectangular subsets, since, similar to d_∞ dispersion, this metric is suited well for analytic analysis.

Intuitively the discrepancy can be seen as how even the data is distributed with regards to the range space. That is, the right fraction in Equation (2.20) is the estimated ratio of samples that, based on the volume, should be inside the subset, while the left fraction is the actual ratio of samples. The discrepancy is defined by the biggest absolute difference that can be found based on the range space. Basically, there are two ways of maximizing the discrepancy: a big area with particularly few samples, or a small area with particularly many samples.

Figure 2.13 visualizes the discrepancy for different range spaces. Something to note is, that even if our set of samples might have a low discrepancy for one range space, the samples might still look ordered. The choice of range thus has a big influence.

For axis-aligned rectangle discrepancy D_{aar} and infinity dispersion d_∞ we can upper bound the dispersion using the discrepancy by [1]

$$d_\infty < D_{\text{aar}}^{1/D}. \tag{2.23}$$

This automatically allows to use low-discrepancy sequences as low-dispersion sequences, although it should be noted that better low-dispersion sequences might be possible when ignoring discrepancy.

2.5.3 Sampling sequences

We next introduce a few sampling sequences from the literature that are mostly designed to be low-discrepancy. Note that due to the link given between dispersion and discrepancy, low-discrepancy automatically implies low-dispersion. In this work we mostly use the Halton sequence as a representative general low-dispersion sequence. In general all of these sequences have the same asymptotic dispersion of $\mathcal{O}\left(n^{-1/D}\right)$[1]. But the sequences differ in the constant hidden in the $\mathcal{O}$-notation. It is not possible to find the constants for Halton though, but at least in lower-dimensions it looks well-distributed and the implementation is straightforward, which is why we have decided to use it for the experiments.

Van der Corput sequence

The *van der Corput* sequence [30] is a one-dimensional low-discrepancy sequence over $[0, 1]$. Mathematically the n-th number is given by first finding the representation of the number n in the base b

$$n = \sum_{k=0}^{L-1} d_k(n) b^k \tag{2.24}$$

where $d_k(n)$ denotes the k-th digit of the number. The n-th van der Corput number s_n is then given by simply taking the same digits as the decimal places, i.e.,

$$s_n = \sum_{k=0}^{L-1} d_k(n) b^{-k-1}. \tag{2.25}$$

While the sequence was originally motivated via binary operations, for general bases such generation schemes do not work anymore. Algorithm 4[2] shows a possible way to generate these numbers arithmetically.

Halton sequence

A generalization of the van der Corput sequence to higher dimensions D is the *Halton* sequence [28]. Generally it simply uses multiple van der Corput sequences with different bases $b_i, i = 1, ..., D$ to create a low-discrepancy sequence. Note that these bases should be mutually prime, since otherwise undesired correlations will occur[3]. Algorithm 5 shows the generation of the Halton sequence based on the

[1]We write $f(n) \in \mathcal{O}(g(n))$ if there $\exists\, n_0 \in \mathbb{N}, k \in \mathbb{R}$ such that $|f(n)| \leq k|g(n)| \,\forall\, n \geq n_0$.

[2]`https://en.wikipedia.org/wiki/Halton_sequence`, accessed: August 20, 2019

[3]Imagine using the same base to generate a 2D Halton sequence, all points would be pairwise equal and instead of covering $[0, 1] \times [0, 1]$ the numbers would all lie on the line $x = y$.

Algorithm 4 Van der Corput sequence

```
1: procedure VanDerCorputSample($(n, b)$)
2:     $f \leftarrow 1$
3:     $r \leftarrow 0$
4:     while $n > 0$ do
5:         $f \leftarrow \frac{1}{b}$
6:         $r \leftarrow r + f(1 \mod b)$
7:         $i \leftarrow \lfloor \frac{i}{b} \rfloor$
8:     end while
9:     return $r$
10: end procedure
```

previous sample. An alternative implementation could of course also use the van der Corput generation from Algorithm 4, but that would result in a slower computation the more samples are generated, but might be faster initially. A more detailed implementation, that also handles numerical issues is also provided by Halton [31]. An alternative that also deals with the same numeric issues and is based on integer arithmetic is proposed Berblinger [32]. For the experiments shown in Chapter 5, Algorithm 4 was implemented and generalized for multiple dimensions.

Algorithm 5 Halton sequence

```
1: procedure HaltonSamples($(\boldsymbol{s}_{n-1} \in \mathbb{R}^D, \boldsymbol{b} \in \mathbb{R}^D)$)
2:     $\boldsymbol{s} \leftarrow \boldsymbol{0} \in \mathbb{R}^D$
3:     for $i = 1, ..., D$ do
4:         $y \leftarrow \frac{1}{b_i}$
5:         $x \leftarrow 1 - s_{n-1,i}$
6:         while $x \leq y$ do
7:             $y \leftarrow \frac{y}{b_i}$
8:         end while
9:         $s_i \leftarrow (b_i + 1)y - x$
10:     end for
11:     return $\boldsymbol{s}$
12: end procedure
```

2.5.4 Sampling in motion planning

Different sampling schemes fulfill different criteria. For example low-discrepancy sampling is helpful for numerical integration methods. In this section we look into desirable properties of the sampler for motion planning systems. We see that, depending on the use case, different qualities in the sampler are desirable. This way we can also easily relate the topic of this thesis, i.e., deterministic sampling-based motion planning, to this issue of choosing the sampler.

Desirable properties for motion planners are (resolution) completeness, speed and its ability to find high quality solutions as well as achieving a high success rate for difficult queries.

Recently, learning-based approaches surfaced which learn latent representations for the map and learn where to sample the space to find a solution with less samples [33, 34]. Such an approach clearly works in favor of the speed requirement. Less samples are required by placing them in places which will most likely be required. One could also imagine that it might help with difficult queries if the characteristics of difficult parts are learned well.

Note that such approaches often learn a sample distribution, but the drawn samples are still probabilistic. Thus only probabilistic completeness can be achieved and even that might be lost due to the way the samples are picked heuristically [2].

A different branch in this area is replacing the random i.i.d. sampler with a deterministic sequence. Such sequences, as introduced in the previous section, give non-probabilistic bounds on the dispersion, which, as has been shown originally by LaValle [35] can be used to derive non-probabilistic completeness for motion planning problems. In general when comparing to i.i.d. samples reportedly this approach also achieves higher success rates, lower cost and requires less samples (i.e., is faster) to find equally good solutions. Due to its relevance for this work an in depth summary of this approach is given in Section 3.1.

In general it seems to be likely that machine learning-based approaches will *on average* require less samples and thus will outperform both i.i.d. and deterministic approaches in many runs.

Right now there does not seem to exist a single approach that can combine both non-probabilistic completeness and the advantages given by machine learning. In general of course it might be possible to simply plan in parallel, with two planners, with the deterministic planner working as a safety net for cases in which the learning-based planner fails to find a solution.

3 Related Work

In this chapter we go over the related work in the area of deterministic motion planning. The two main directions will be introduced. The first is the use of deterministic samplers, in which the same (or only slightly modified) versions of the sampling-based algorithms are used. In this context we also repeat the most important theoretic results, as they will directly relate to our extensions in Chapter 4. The theorems have been slightly rewritten to conform to our notation.

The second direction are state lattice-based approaches. While Section 2.4.3 introduced the basic idea of planning with motion primitives, in this section we summarize the literature in this area, including ways of generating and optimizing motion primitives and application-specific publications.

In general, all approaches in this chapter fit into the range of deterministic sampling-based motion planning as introduced by LaValle [35]. Figure 3.1 visualizes this range of planners. On the right-hand side is the probabilistic roadmap planner that uses i.i.d. samples to build the roadmap. Replacing the sampler with a quasi-random sequence (i.e., the Halton sequence) leads to quasi-random roadmaps. A more regular way of picking the samples is obtained with lattice-based samples. Such roadmaps are for example constructed in the state lattice-based approach, but also for samples in the Euclidean space there are ways of generating a regular lattice. Finally, when using an orthogonal grid we come to subsampled grid-search. The only difference to classical grid-search, on the left-hand side is that collision checking is still performed along the edges, while in classical grid search, collision checking and planning is done at the same resolution.

While the approach introduced in Section 3.1 fits into the category of quasi-random roadmap, the lattice-based approach introduced in Section 3.2 fits into the lattice-based grid-search category.

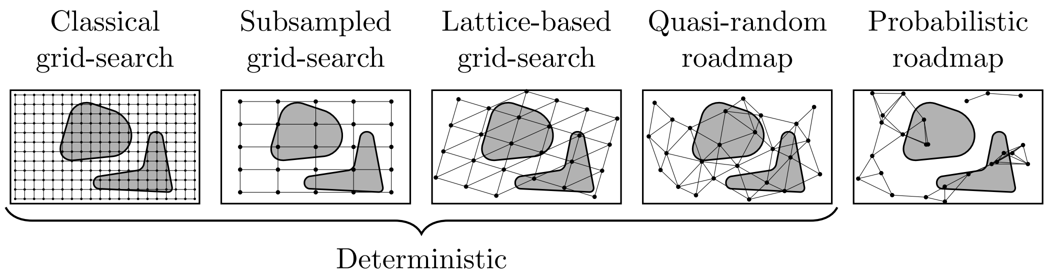


Figure 3.1: Visualization of the range of planners. See text for description.

3.1 Deterministic Sampling for Motion Planning

The link between dispersion, discrepancy and motion planning has originally been discovered by LaValle [35]. The key result that LaValle proved can be summarized by the following theorem:

Theorem 1. Given a query γ, deterministic roadmap planners will, after n low-dispersion samples have been picked, either report a solution path or correctly declare that one of the following is true: there is no solution path, or $\delta(\gamma) < 2\beta n^{-1/D} = \hat{\delta}$.

This is equivalent to stating that deterministic roadmap planners are $\hat{\delta}$-complete. Note that LaValle did not consider a specific planner or connection radius. For the planner he introduces the notion of deterministic roadmap planners, which includes the four deterministic planners shown in Figure 3.1 when using a low-dispersion sampler. For the theorem it is assumed that the dispersion after n samples is bounded by

$$d_2 \leq \beta n^{-1/D} \in \mathcal{O}\left(n^{-1/D}\right) \tag{3.1}$$

where β is the constant hidden in the $\mathcal{O}$-notation and generally depends on the dimension D. For the connection strategy a sufficiently large connection radius is assumed. Also, he only considers Euclidean systems, i.e., systems for which the distance and path between two poses is given by the Euclidean distance and linear interpolation respectively.

Following from this result LaValle derives multiple asymptotic qualities, two of which we shortly present here [35]. All of them stem from the fact that i.i.d. samples have an asymptotic dispersion of

$$d_2 \in \mathcal{O}\left((\log(n)/n)^{1/D}\right), \tag{3.2}$$

while low-dispersion sequences achieve

$$d_2 \in \mathcal{O}\left(n^{-1/D}\right). \tag{3.3}$$

Theorem 2. The number of samples required by deterministic roadmap planners to be $\hat{\delta}$-complete is asymptotically optimal.

This result follows from the fact that there is no sequence achieving lower asymptotic dispersion than $\mathcal{O}\left(n^{-1/D}\right)$.

Theorem 3. For a fixed dimension D, Algorithm 2 with i.i.d. samples requires $\mathcal{O}\left((\log n)^{\frac{1}{D}}\right)$ times as many samples as DRM planners to achieve the same d_∞ dispersion.

This result simply highlights the asymptotic difference between i.i.d. and low-dispersion sequences. Relating this to Theorem 3.1 we can see that by using low-dispersion sampling, less samples are required to get the same guarantees. In addition, in comparison to i.i.d. samples the guarantees hold deterministically, not only with a probability of 1 if $n \to \infty$.

Janson [20] extended the dispersion analysis to optimal motion planning. Next, we shortly present the two most important theorems from this work.

Theorem 4. Given a sampler with asymptotic dispersion $d_2 \in \mathcal{O}\left(n^{-1/D}\right)$ and a connection radius[1]

$$r_n \in \omega\left(n^{-1/D}\right) \tag{3.4}$$

then

$$\lim_{n\to\infty} c_n = c^*. \tag{3.5}$$

This is the deterministic version of the result from Karaman's [2] condition on the PRM connection radius. Similar to the results from LaValle, the difference to the original PRM* connection radius is the missing factor $\log(n)^{1/D}$.

The following theorem bounds the cost, based on the cost of a path with some clearance, again the idea here is that because of the dispersion, we can be sure to generally find a path of certain clearance, which can then be used to bound the cost of the actually found solution.

Theorem 5. Given a set of samples $\mathcal{S}$ with dispersion d_2 and a connection radius $r > 2d_2$ we can bound the solution cost c, based on the optimal cost $c^{(\delta)}$ with clearance $\delta > r + d_2$ by

$$c \leq \left(1 + \frac{2d_2}{r - 2d_2}\right) c^{(\delta)}. \tag{3.6}$$

Poccia looked into finding deterministic sequences, specific to differentially constrained systems [36]. In this work we reuse some of his ideas. Specifically he introduced optimized samples for two systems:

[1]We write $f(n) \in \omega\left(g(n)\right)$ if $f(n)/g(n) \to \infty$ for $n \to \infty$.

- linear systems (see Equation (2.10)) with quadratic cost and fixed-time optimal steering function, and
- Reeds-Shepp car (see Section 2.3.1).

In the first case he shows that by taking a general low-dispersion sequence one can apply a transformation to fit them to the system specific case. In the Reeds-Shepp case he proposes a tailored solution that optimizes the positions of the samples based on the weights for the ball-box theorem. Note that the author also provides the general idea for driftless control-affine systems and uses it to derive the Reeds-Shepp case, but this derivation is not general and will not work in the general driftless control-affine case.

As we use these optimized samples for the Reeds-Shepp system as a baseline to compare our approach against, we next give a description of the generation procedure proposed by Poccia. The general approach is based on first finding the description of the ball-box theorem (for more on sub-Riemannian geometry see [37, 38, 39]). These are the weight vector $\boldsymbol{w} \in \mathbb{N}^D$ as well as the privileged coordinate vectors $\boldsymbol{L}_i(\boldsymbol{x})$ with $i = 1, ..., D$. Together these describe the orientation and size ratios of boxes that can inner and outer bound the time-limited reachable sets of a system in each point $\boldsymbol{x}$. Finding samples which are asymptotically low-dispersed considering these boxes thus will automatically also be asymptotically low-dispersed considering the reachable sets of the system. Note that by doing so, we are approximating the system dynamics crudely and the practical quality of the samples will vary a lot depending on other parameters (e.g. number of samples, system constants, environment size).

For the Reeds-Shepp case the privileged coordinate vectors in the Reeds-Shepp space are given by

$$\boldsymbol{L}_1(\boldsymbol{x}) = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix}, \boldsymbol{L}_2(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\rho} \end{bmatrix}, \boldsymbol{L}_3(\boldsymbol{x}) = \begin{bmatrix} -\frac{1}{\rho}\sin\theta \\ \frac{1}{\rho}\cos\theta \\ 0 \end{bmatrix} \tag{3.7}$$

and the weight vector by $\boldsymbol{w} = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix}$. $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ are the same as $\boldsymbol{g}_\mathrm{s}$ and $\boldsymbol{g}_\phi$ from Equation (2.15), respectively. $\boldsymbol{L}_3$ is computed as the Lie bracket of of $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, i.e.,

$$[\boldsymbol{L}_1, \boldsymbol{L}_2] = \boldsymbol{J}_{\boldsymbol{L}_2}\boldsymbol{L}_1 - \boldsymbol{J}_{\boldsymbol{L}_1}\boldsymbol{L}_2. \tag{3.8}$$

where $\boldsymbol{J}_{\boldsymbol{x}}$ is the Jacobian matrix of vector $\boldsymbol{x}$.

The special characteristic of the Reeds-Shepp case that makes it possible to generate a set based on that, is that the vectors only depend on θ. Given a number of samples n that we want to generate we can compute

$$\varepsilon = \left(\frac{2\pi\rho^2}{n}\right)^{\frac{1}{4}}. \tag{3.9}$$

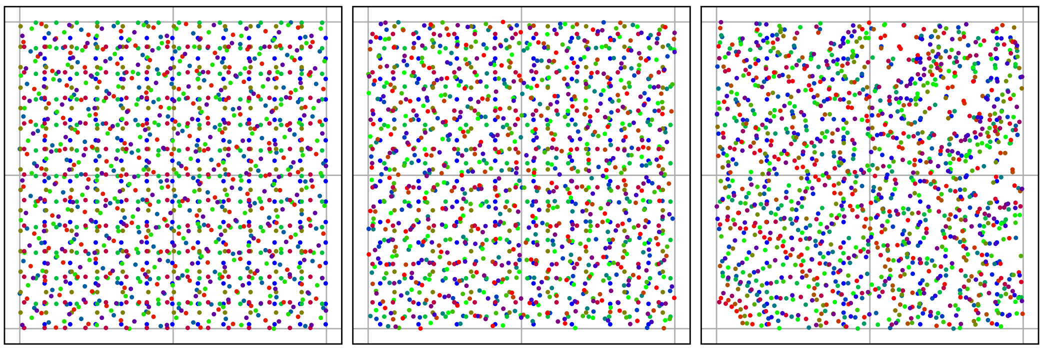

(a) $\rho = 0.1, n = 1500$ (b) $\rho = 0.25, n = 1500$ (c) $\rho = 1.0, n = 1500$

Figure 3.2: Resulting samples when using Poccia's generation procedure for different parameter combinations. The color indicates the orientation of the sample.

Based on ε we can then compute the number of different θ coordinates by $n_\theta = \lfloor \frac{2\pi\rho}{\varepsilon} + 0.5 \rfloor$ (+0.5 to round to the next integer). From that the θ coordinates of the samples are given by $\theta = k\varepsilon^{w_1} - \pi, k = 0, ..., n_\theta - 1$. For each of these values of θ we add a grid with its axes given by $\boldsymbol{L}_1$ and $\boldsymbol{L}_3$, i.e., an orthogonal grid, rotated by θ. Additionally the step size in this grid is given by ε^{w_1} and ε^{w_3} along $\boldsymbol{L}_1$ and $\boldsymbol{L}_3$, respectively. Note that due to the rotation the final set might have slightly more or less samples than the originally desired number of samples n. Figure 3.2 shows the 2D projection of some sets for $n = 1500$ and different turning radii ρ.

It should be noted that Poccia's method only provides a set, not a specific order at which the samples should be taken or any incremental way of picking the samples. Relating that to the termination criteria defined in Chapter 2 the best way to use the samples is with termination after a fixed number of sample attempts n_{all} irrespective of whether they are in collision or not. Time-based and valid samples-based termination have the risk of either running out of samples (which can be prevented by precomputing a large enough set), but also require some sorting at which the samples should be picked. Note, also that precomputing a large enough set is basically prevented by the fact of not having a good order of picking the samples. This is because by, for example, generating a dense set and randomly picking samples from it, we lose the whole beneficial structure of the set when only a subset is actually used. As described in Section 4.2, our approach generates a sorted set, which has the same downside of potentially running out of samples, but since it has a fixed order, precomputing enough samples works much better than in the case of Poccia's method.

Finally, Poccia also provides the deterministic versions for Schmerling's theorems on optimal sampling-based motion planning under differential constraints for both the driftless control-affine case [15] as well as the linear case with drift [14]. Note the interesting relation between all of these papers. Similar to how Janson [20] provides the deterministic version of Karaman's work [2], Poccia [36] provides the deterministic version of Schmerling's work [14, 15] which in itself is an extension of Karaman's work [2] to systems with differential constraints. There is also another related paper by Karaman that covers optimal nonholonomic motion planning using RRT* [40].

The influence of discrepancy was also discussed by LaValle [35]. He argues that discrepancy can be of merit in case of axis aligned problems, i.e., cases in which features of the map align with the space boundaries. Clearly such axis alignments seem arbitrary and can easily be solved by applying random rotations to either the map or the sampled points. This can also be seen by the fact that the metric itself as specified in (2.20) depends a lot on the chosen range space. For the typical axis-aligned rectangles case, obviously simply rotating the sample set would already decrease the discrepancy, without actually changing the regular characteristic of a set. Also, choosing a different range space will yield completely different numbers. Due to these reasons discrepancy will not be further considered, but the issue of axis alignment should be kept in mind to prevent rare worst-case scenarios.

Prior to the more theoretically focussed work by LaValle, Branicky already showed empirically the advantage of using Hammersley or Halton samples instead of i.i.d. samples [41]. Similar to our optimization approach there have been a few more attempts to find ways of generating optimized samples for different spaces. Lindemann provides sequences for the most common state spaces in robotics [42]. Specifically for $SO(3)$ Yershova published two different approaches [43, 44]. There has also been some work in derandomizing the RRT-based motion planners [45, 46] by using Voronoi regions. A more recent publication by Khaksar [47] is quite similar to our approach, but still uses a probabilistic optimization scheme, basically skipping samples that are in proximity to other samples. In addition it does not consider differential constraints.

3.2 State Lattice-Based Approaches

The general idea of state lattice-based planning was already introduced in Section 2.4.3. Here we give an overview over the literature that can be found in this area for both the general idea and more specifically for the generation of motion primitives.

A lot of work in this area was done by Pivtoraiko [48, 49, 50, 51, 52]. In his early work of 2005 [49] he coined the terms forward and inverse primitive generation. The forward approach is based on sampling in control space to generate primitives, while the inverse approach is based on choosing the goal pose and solve an optimal control problem to connect the initial state with it.

As noted before in Section 2.4.3, generally the forward approach will lead to non-lattice primitives. Bicchi looked into how the quantization of the control space relates to whether the reachable sets become discrete or dense (see our discussion in Section 2.4.3) [53]. This idea was applied to the N-trailer vehicle model by Pancanti [54].

Frazzoli introduced the idea of using motion primitives [55], calling the primitives *maneuvers* instead. He advocates for the use of the inverse approach to generate these maneuvers. The most similar framework of state lattice-based motion planning to the one we use was introduced by Pivtoraiko [52]. This original work mostly focussed on the general idea of generating motion primitives via an inverse approach such that a convenient discretization of the configuration space is obtained. He extended the work, introducing the idea of a heuristic look-up table and a first Manhattan distance-based primitive generation algorithm [49]. A more in-depth description of the heuristic look-up table is given by Knepper [56] and Pivtoraiko [57].

An example for the approximation-based paradigm that we have introduced in Section 2.4.3 is the work by McNaughton [58]. A theoretical framework that considers such approximation is provided by Chen [59].

Many extensions to this framework have been proposed. As noted above, Knepper introduced a heuristic look-up table to state lattice-based motion planning [56]. The idea behind this approach is to precompute the optimal paths in free space given the primitives and use this information to get a more accurate heuristic for the A* search. Pivtoraiko [60] introduced the notion of graduated fidelity, that allows to have different resolutions of primitives depending on the proximity to the robot. Another extension to allow for anytime planning was proposed by Gonzalez [61] and multiple ways of handling dynamic environments have been proposed by Kushleyev [62], Andersson [63] and Ziegler [64]. An extension to multi-robot systems was proposed by Cirillo [65]. Furthermore Butzke [66] proposes an interesting approach that adds controller-based motion primitives (i.e., follow wall or follow street). Such a paradigm would for example allow seamless planning between navigation on streets (based on controllers) and a parking lot (based on primitives).

Sakcak introduced an RRT-like approach that includes motion primitives resulting in a planner similar to state lattice planners [67].

Additionally to the previously mentioned work there is a wide-range of publications aiming at specific applications: planetary rovers [68, 69, 70], micro-UAVs (unmanned aerial vehicles) [71], 2-trailer systems [72], autonomous driving [73], autonomous trucks [74], quadcopters [63] and airplanes [75].

The surveyed literature shows the high practicability of this approach due to its low computational cost and because it enables many different optimization options such as bidirectional planning [76], offline precomputation [56], and path repairing [71].

A few papers aim at specifically optimizing the set of motion primitives. Green introduces the notion of dispersion for paths [77], which measures the similarity of two primitives by integrating the area between the two primitives and attempting to maximize this metric. The intuition behind this is that primitives which go through a similar area will have increased probability to both be in collision, while by maximizing the difference between the primitives we increase the probability of one primitive to be collision-free. Recently, Botros introduced the notion of t-spanning [78]. This approach assumes a known grid and tries to find a set of primitives, such that all grid cells are connected with primitives, in a way that the resulting path length between any two vertices is not more than t-times bigger than the optimal path length. Finally, Pivtoraiko proposed multiple algorithms to generating motion primitive sets: a simple one, based on the Manhattan distance [52] and a more complex one based on D* decomposition [76]. A recent trend for primitive generation are learning-based approaches [79].

3.3 Steering Functions

Finally, we want to give an overview of available steering functions that are necessary for the optimization method introduced in the next section. In addition such steering methods can be used in an inverse motion primitive generator.

The first optimal steering function was published by Dubin for cars that can only go forward [11], the Reeds-Shepp car that we introduced in Section 2.3.1 was published by Reeds and Shepp [12] with more details being provided by Sussmann [10] and Soueres [13]. The Dubin's car was also generalized to the 3D case by Chitsaz [80]. For the popular differential drive there are two kinds of optimal steering functions. The first by Balkcom [81] considers time-optimal trajectories with velocity-bounded wheels and the second by Chitsaz [82] finds the trajectories with the minimum wheel rotation, which in many cases will be equivalent to the most energy efficient trajectory. Finally, for a special omnidirectional robot Balkcom also published an optimal steering function [83]. An attempt to generalize finding optimal paths for car-like systems was published by Furtuna, but it only characterizes the type of paths without explicitly finding the trajectories [84]. Finally, for all other systems in which no analytical optimal steering function exists, numerical optimization can be used [85, 86].

4 Method

In this chapter we introduce the main contribution of this work. First, in the context of nonholonomic deterministic motion planning we show how dispersion links to resolution completeness via time-limited reachable sets. Based on this, we then introduce an optimization approach which can be used to precompute samples that optimize the dispersion for systems with a known optimal and symmetric steering function.

Subsequently, we describe a general framework for state lattice planners, which have already been introduced in Section 2.4.3 and 3.2. The state lattice framework naturally connects to the roadmap completeness proof, which can be applied again to proof completeness for the state lattice planner. Additionally we introduce an alternative way of proving completeness, by introducing δ-completeness, a property for the set of motion primitives that implies resolution completeness.

Both, the dispersion based-proof and δ-completeness can be used to motivate primitive generation schemes, which we detail in Section 4.3.2.

Interfaces for all approaches introduced in this section were implemented for the *Open Motion Planning Library* (OMPL) [21]. The optimization scheme that is described in Section 4.2 was implemented in Python with the resulting precomputed set being usable inside OMPL.

4.1 Differentially Constrained Deterministic Motion Planning

In general the most valuable property of deterministic motion planning is ensuring both asymptotic optimality as well as resolution completeness (see Theorem 3.1 for the Euclidean case). Having both of these, we can be sure that paths going through corridors with certain clearance will be found and that the cost is bounded by these solution paths with certain clearance (see Theorem 5 for the Euclidean case).

To extend the Euclidean framework from the literature to differentially constrained motion planning, we need to introduce a new notion of clearance, that can then be used to define resolution completeness for differential motion planning. Similar ideas have also been used by Schmerling [14, 15] and Poccia [36]. This is a straightforward extension of Definition 5 to reachable sets, defined by a distance function dist.

Definition 11. The δ_{dist}-*clearance* of a path σ is defined by[1]

$$\delta_{\text{dist}}(\sigma) = \sup\{r \in \mathbb{R} \,|\, \mathcal{R}_{\text{dist}}(\boldsymbol{x}, r) \subseteq \mathcal{X}_{\text{free}} \,\forall\, \boldsymbol{x} \in \sigma([0,1])\}. \tag{4.1}$$

Based on this, the δ_{dist}-*clearance* of a query γ is defined by

$$\delta_{\text{dist}}(\gamma) = \sup\{\delta_{\text{dist}}(\sigma) \,|\, \sigma \in \Sigma_\gamma\}. \tag{4.2}$$

We can use this definition of clearance to prove resolution completeness for general systems (potentially differentially constrained, nonholonomic, etc.) as long as they are symmetric and an optimal steering function is available.

Theorem 6. Given a set of samples $\mathcal{S}$ with known dispersion $\tilde{d}_{\text{dist}}$ and considering symmetric systems with optimal steering function, Algorithm 2 with a connection radius $r > 2\tilde{d}_{\text{dist}}$ solves all planning queries γ with clearance

$$\delta_{\text{dist}}(\gamma) > 2\tilde{d}_{\text{dist}}. \tag{4.3}$$

Proof. To see this, first note that $\mathcal{R}(\boldsymbol{x}, r)$ for optimal steering functions, is equivalent to time-limited reachable sets of the system. Hence, trajectories from $\boldsymbol{x}$ to any other point in $\mathcal{R}(\boldsymbol{x}, r)$ will also be fully contained in $\mathcal{R}(\boldsymbol{x}, r)$. Given a query γ with clearance $\delta(\gamma) \geq 2\tilde{d}_{\text{dist}}$, there exists a solution σ with $\mathcal{R}(\boldsymbol{s}_i, 2\tilde{d}_{\text{dist}}) \in \mathcal{X}_{\text{free}}$, $\forall \boldsymbol{s}_i \in \sigma([0,1])$. First, note that due to the dispersion definition, there must be a sample of $\mathcal{S}$ in both $\mathcal{R}(\boldsymbol{x}_{\text{init}}, \tilde{d}_{\text{dist}})$ and $\mathcal{R}(\boldsymbol{x}_{\text{goal}}, \tilde{d}_{\text{dist}})$. Thus it is possible to connect the start and goal configuration to the roadmap. It remains to show that a dist-clearance of $\delta_{\text{dist}}(\gamma) \geq 2\tilde{d}_{\text{dist}}$ is sufficient to find a path from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$. Let $\boldsymbol{q}_0$ and $\boldsymbol{q}_N$ denote the samples that $\boldsymbol{x}_{\text{init}}$ and $\boldsymbol{x}_{\text{goal}}$ are connected to, respectively. By taking $\boldsymbol{s}_1$ along the path σ and such that $\boldsymbol{q}_0$ lies on the border of $\mathcal{R}(\boldsymbol{s}_1, \tilde{d}_{\text{dist}})$ we see, due to the dispersion definition in Equation (2.18), there must be another sample inside this reachable set, denoted by $\boldsymbol{q}_1$. At this point we only know that the path from $\boldsymbol{q}_0$ over $\boldsymbol{s}_1$ to $\boldsymbol{q}_1$ must be collision-free. Since only $\boldsymbol{q}_0$ and $\boldsymbol{q}_1$ are known, we require a factor of 2 in the clearance (i.e., $2\tilde{d}_{\text{dist}}$ in Equation (4.3)), which ensures that the path from $\boldsymbol{q}_0$ to $\boldsymbol{q}_1$ is collision-free. To see this, note that due to the system symmetry both $\mathcal{R}(\boldsymbol{q}_0, \tilde{d}_{\text{dist}})$ and $\mathcal{R}(\boldsymbol{q}_1, \tilde{d}_{\text{dist}})$ must be contained in $\mathcal{R}(\boldsymbol{s}_1, 2\tilde{d}_{\text{dist}}) \subset \mathcal{X}_{\text{free}}$. We also know that the intersection $\mathcal{R}(\boldsymbol{q}_0, \tilde{d}_{\text{dist}}) \cap \mathcal{R}(\boldsymbol{q}_1, \tilde{d}_{\text{dist}})$ contains $\boldsymbol{s}_1$ and is thus nonempty. The trajectory from $\boldsymbol{q}_0$ to $\boldsymbol{q}_1$ must pass through this intersection and is hence collision-free. The same idea can now be repeated until the path to $\boldsymbol{q}_N$ is found. The idea of the proof is visualized in Figure 4.1. ■

[1]See Definition 10 and the related discussion for the definition of $\mathcal{R}_{\text{dist}}$.

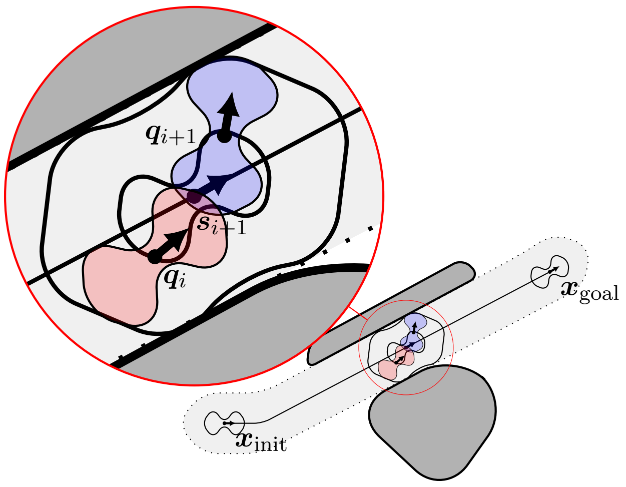


Figure 4.1: Visualization of the proof of completeness. $\boldsymbol{s}_{i+1}$ is placed along the (unknown) path of maximum clearance σ such that $\boldsymbol{q}_i$ lies on the border of $\mathcal{R}(\boldsymbol{s}_{i+1}, \tilde{d}_{\text{dist}})$. Since $\mathcal{R}(\boldsymbol{q}_i, \tilde{d}_{\text{dist}})$ and $\mathcal{R}(\boldsymbol{q}_{i+1}, \tilde{d}_{\text{dist}})$ overlap and are fully contained in $\mathcal{R}(\boldsymbol{s}_{i+1}, 2\tilde{d}_{\text{dist}})$ the path from $\boldsymbol{q}_i$ to $\boldsymbol{q}_{i+1}$ is collision-free.

4.2 Optimized Sampling Sequence

In the last section we showed that it is possible to define a distance metric that extends the dispersion-based analysis in Euclidean space to differentially constrained systems for which a distance metric can be defined. Next, we introduce a procedure that numerically optimizes this dispersion metric. It should be noted that this approach actually gives a (limited-length) sequence not a set of fixed size. In general it is possible to find improved sets when fixing the number of samples a priori, but those sets will then have worse dispersion if only a subset of them are taken.

Greedy dispersion optimization

The procedure outlined in the following can be used to precompute a sequence of a fixed number of samples. It requires the resolution of the grid, the distance metric, and the size of the environment. Since the accuracy of the algorithm is based on the resolution, the number of grid cells should at least be an order of magnitude larger than the desired number of samples. Depending on the system it might be feasible to set a different resolution for different dimensions.

Algorithm 6 shows the general flow of the algorithm in its most basic version. The tensor $\boldsymbol{D}$ is a D-dimensional (D is the dimension of the space) tensor that has one entry per grid cell. In the initial step (line 2) the matrix is initialized with ∞. Next, we pick samples until the desired number of samples n_{all} have been found (lines 3–7). This is done by greedily picking the sample at the grid cell with the largest distance at that iteration. After picking the sample, the distance tensor is updated to take into account the newly picked sample. Hence, by construction the distance tensor keeps track of the current distance of the center of each cell to the closest sample. Algorithm 7 shows the general idea behind UPDATEDISTANCETENSOR. Basically, we iterate over all grid cells and adjust the cell value in case the new sample $\boldsymbol{x}$ has a smaller distance than the currently stored value (line 4).

Algorithm 6 Greedy dispersion optimization

```
1: procedure DispersionOptimization
2:     $\boldsymbol{D} \leftarrow \infty$
3:     while $|\mathcal{S}| < n_{\text{all}}$ do
4:         $\boldsymbol{x} \leftarrow \arg\max_{\boldsymbol{c}} D_{\boldsymbol{c}}$
5:         UpdateDistanceTensor($\boldsymbol{D}, \boldsymbol{x}$)
6:         $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{x}\}$
7:     end while
8: end procedure
```

Algorithm 7 Distance tensor update

```
1: function UpdateDistanceTensor($\boldsymbol{D}, \boldsymbol{x}$)
2:     for $\boldsymbol{c}$ in $\boldsymbol{D}$ do
3:         $d \leftarrow \text{dist}(\boldsymbol{c}, \boldsymbol{x})$
4:         $D_{\boldsymbol{c}} \leftarrow \min(d, D_{\boldsymbol{c}})$
5:     end for
6: end function
```

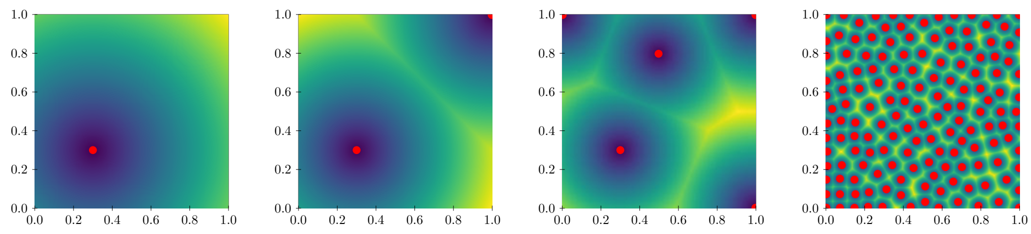
Example 4.1. A simple case to test the algorithm is the Euclidean case in which the distance function is given by the Euclidean distance as

$$\text{dist}(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||_2. \tag{4.4}$$

Figure 4.2 illustrates the progression of the algorithm for $\mathcal{X} = [0,1] \times [0,1]$. Note that the first sample is placed randomly, since the whole distance tensor is initialized with ∞. Note that many samples are placed right next to the border.

Modified dispersion optimization

One problem with the algorithm in its current form is that there is a high possibility that samples will be placed on, or more specifically up to the grid resolution close,



(a) $n = 1$ (b) $n = 2$ (c) $n = 5$ (d) $n = 150$

Figure 4.2: Progression of the distance matrix after the update step. The red dots show the samples so far and the underlying color shows the distance (note that this color is scaled differently for each plot).

the border of the space. This can also be seen in Figure 4.2(d). While this is generally coherent with the classical dispersion definition from Niederreiter [27] as given by (2.18), it poses a problem for motion planning as the boundary of the space is typically considered as an obstacle and thus all samples on the border will be discarded in the planning if the robot is not a point robot[1]. One way to prevent this, is to adjust the definition of dispersion to also require the reachable set to be fully contained in the space, i.e., $\mathcal{R}_{\text{dist}}(\boldsymbol{x}, r) \in \mathcal{X}$, instead of just $\boldsymbol{x} \in \mathcal{X}$, which yields the modified dispersion metric

$$\tilde{d}_{\text{dist}} = \sup\{r > 0 | \exists \boldsymbol{x} \in \mathcal{X} \text{ with } \mathcal{R}_{\text{dist}}(\boldsymbol{x}, r) \cap \mathcal{S} = \emptyset \wedge \mathcal{R}_{\text{dist}}(\boldsymbol{x}, r) \subseteq \mathcal{X}\}. \tag{4.5}$$

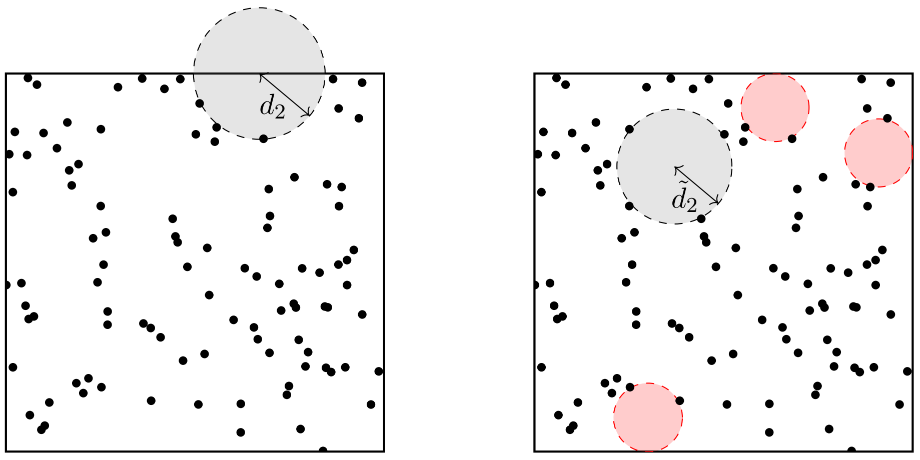
Indeed it is mentioned by LaValle [35] in the proof of completeness that the balls should be fully contained inside the set, without adjusting the dispersion metric specifically. Figure 4.3 visualizes the difference between the two metrics. Note that the completeness proof from Theorem 6 holds regardless of whether the dispersion from Equation (2.18) or the modified dispersion from Equation (4.5) is used.

One way of adapting Algorithm 6 to optimize Equation (4.5) instead, is to initialize $\boldsymbol{D}$ with the distance of the cell to the closest border. By doing so $\boldsymbol{D}$ not only tracks the distance to the closest sample, but either the distance to the closest sample or the distance to the closest border, whichever is smaller. If it is easy to compute the distance to the border, such an initialization is optimal. On the other hand, this requires an additional function DistanceToBorder, which might not be available for more complicated systems[2].

An alternative, more involved approach, is given by Algorithm 8, which is based solely on having a distance function dist. The algorithm works by picking a random sample (line 2) and prior to adding it to the set of samples $\mathcal{S}$, we first check if a border

[1]Note that this depends on the topology and collision model of the robot.

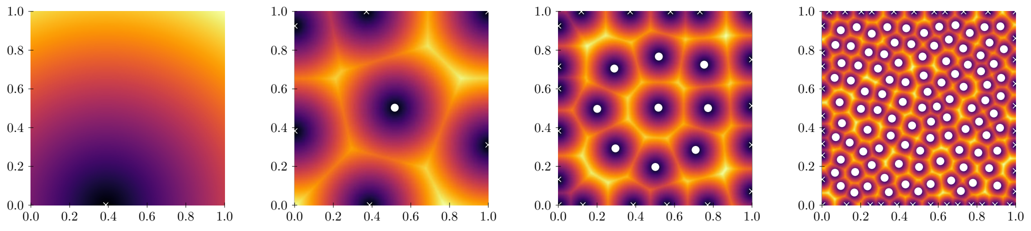
[2]For example, in the Euclidean case it is easy to compute the distance, while for the Reeds-Shepp car it is not possible.



(a) Classic dispersion (b) Modified dispersion

Figure 4.3: Comparison between classic dispersion (2.18) and the modified dispersion metric (4.5). The red circles indicate that the balls close to the boundary, now must be fully included in the bounded set and thus $\tilde{d}_2 < d_2$.

point would be affected via the call to AffectedBorderPoint (see Algorithm 9), which checks if UpdateDistanceMatrix would change any border point and if it does, return the affected border point, without affecting $\boldsymbol{D}$ in any way. If no border point is affected, we proceed as usual, by updating the distance matrix and adding the sample to the set (lines 7–9). If it is, we perform an update step on the border point, but we are not adding the border point to the set of samples (lines 10–11). Similar to before we continue by picking the sample with the largest distance in the distance tensor $\boldsymbol{D}$ until we have found the desired number of samples n_{all}.



(a) $n = 0$ (b) $n = 1$ (c) $n = 9$ (d) $n = 100$

Figure 4.4: Progression of the distance matrix after the update step for the modified algorithm. The white dots show the samples so far while the white crosses show the processed border samples.

Algorithm 8 Modified dispersion optimization algorithm

```
1: procedure ModifiedDispersionOptimization
2:     $\boldsymbol{D} \leftarrow \infty$
3:     $\boldsymbol{x} = \boldsymbol{\mathcal{U}}(\mathcal{X})$
4:     $\mathcal{S} \leftarrow \{\}$
5:     while $|\mathcal{S}| < n_{\text{all}}$ do
6:         $b \leftarrow$ AffectedBorderPoint($\boldsymbol{D}, \boldsymbol{x}$)
7:         if b=None then
8:             UpdateDistanceTensor($\boldsymbol{D}, \boldsymbol{x}$)
9:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{x}\}$
10:        else
11:            UpdateDistanceTensor($\boldsymbol{D}, \boldsymbol{b}$)
12:        end if
13:        $\boldsymbol{x} \leftarrow \arg\min_{\boldsymbol{c}} D_{\boldsymbol{c}}$
14:    end while
15: end procedure
```

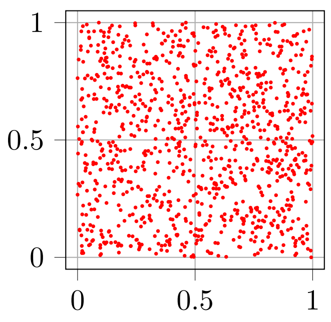
Algorithm 9 Affected border check

```
1: function AffectedBorderPoint($\boldsymbol{D}, \boldsymbol{x}$)
2:     for $\boldsymbol{c}$ in $\boldsymbol{D}$ do
3:         if $\text{dist}(\boldsymbol{c}, \boldsymbol{x}) < D_{\boldsymbol{c}} \wedge$ IsBorderPoint($\boldsymbol{c}$) then
4:             return $\boldsymbol{c}$
5:         end if
6:     end for
7:     return None
8: end function
```

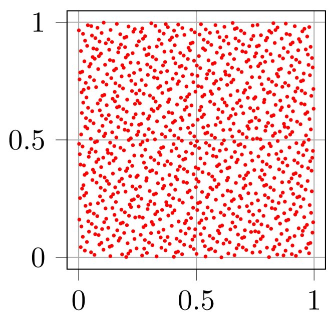
Algorithm experiments

We will next present two experiments, which confirm numerically the efficacy of the algorithm in generating samples with lower dispersion than other approaches (i.e., i.i.d. and Halton). We show these experiments here and not in Section 5, since the two following experiments are not performed in any motion planning context and only aim to empirically show the potential in optimizing the dispersion.

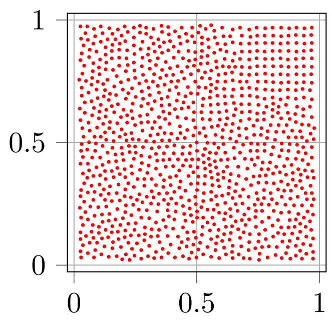
Experiment 1. (Euclidean) To ensure that the general idea of the algorithm works, sequences of samples from different samplers have been generated and numerically the progress of the dispersion has been calculated. First, we have a look at the Euclidean case, for which Figure 4.5 shows the sets as well as the dispersion progression (i.e., the dispersion after n samples). Clearly the optimized sequence outperforms both i.i.d. and Halton samples.



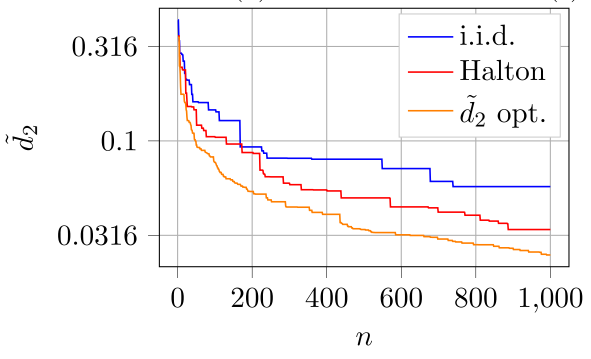
(a) i.i.d.



(b) Halton



(c) $\tilde{d}_2$ optimized

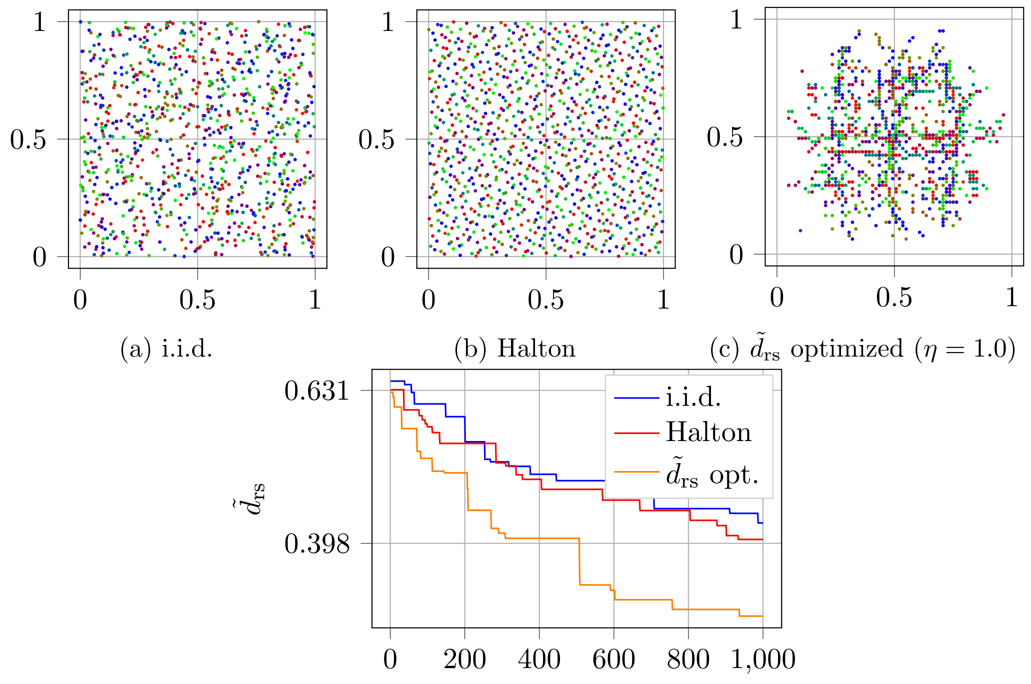


(d) $\tilde{d}_2$ progress

Figure 4.5: Comparison of different sets of samples for $\mathcal{X} = [0,1] \times [0,1]$. Each set in (a)–(c) contains $n = 1000$ samples. (d) shows the modified dispersion $\tilde{d}_2$ after a certain number of samples n. It can be seen that the optimized sequence consistently yields a lower dispersion than the low-dispersion Halton sequence or the i.i.d. samples.

Experiment 2. (Reeds-Shepp) To show the viability of the algorithm for nonlinear distance metrics the optimized sampling sets for the Reeds-Shepp system are shown in Figure 4.6. Again we see a clear advantage of the optimized sequence against both i.i.d. as well as Halton samples.

An important metric for this case is the ratio between the environment size and the turning radius. If the environment is much larger than the turning radius the set will look similar to the Euclidean set. If the environment has a similar size as the turning radius the sets structure will be mostly determined by the differential constraints of the system. This difference is because for the highly constrained case, the distances between the samples is always in the same ballpark as the minimum turning radius and thus multiple maneuvers are required to connect them. Conversely if the environment is large, the turning maneuver can become so small in comparison to the full path, that the optimal path between two points basically looks like a straight path (as long as the points are not close to each other). A comparison for different ratios $\eta = r_{\min}/w_{\text{env}}$, where $r_{\min}$ is the minimum turning radius and w_{env} is the edge length of the environment, is shown in Figure 4.7.



(a) i.i.d. (b) Halton (c) $\tilde{d}_{\text{rs}}$ optimized ($\eta = 1.0$)

(d) $\tilde{d}_{\text{rs}}$ progress

Figure 4.6: Similar analysis as shown in Figure 4.5 for the Reeds-Shepp case. The color indicates the orientation of the sample. Again the optimized sequence, clearly outperforms Halton and i.i.d. samples. Comparing Figure 4.2(c) with (c) the strong influence of the distance function on the optimized set can be seen.

Efficiency considerations

Since the sample optimization is computationally quite expensive, that is, it has a computational complexity of $\mathcal{O}(n_{\text{all}} \cdot N)$, where n_{all} is the number of samples and N is the number of grid cells. Since both the number of grid cells and number of samples in practice are often exponential in the dimension the runtime is practically exponential in the dimensionality D of the space.

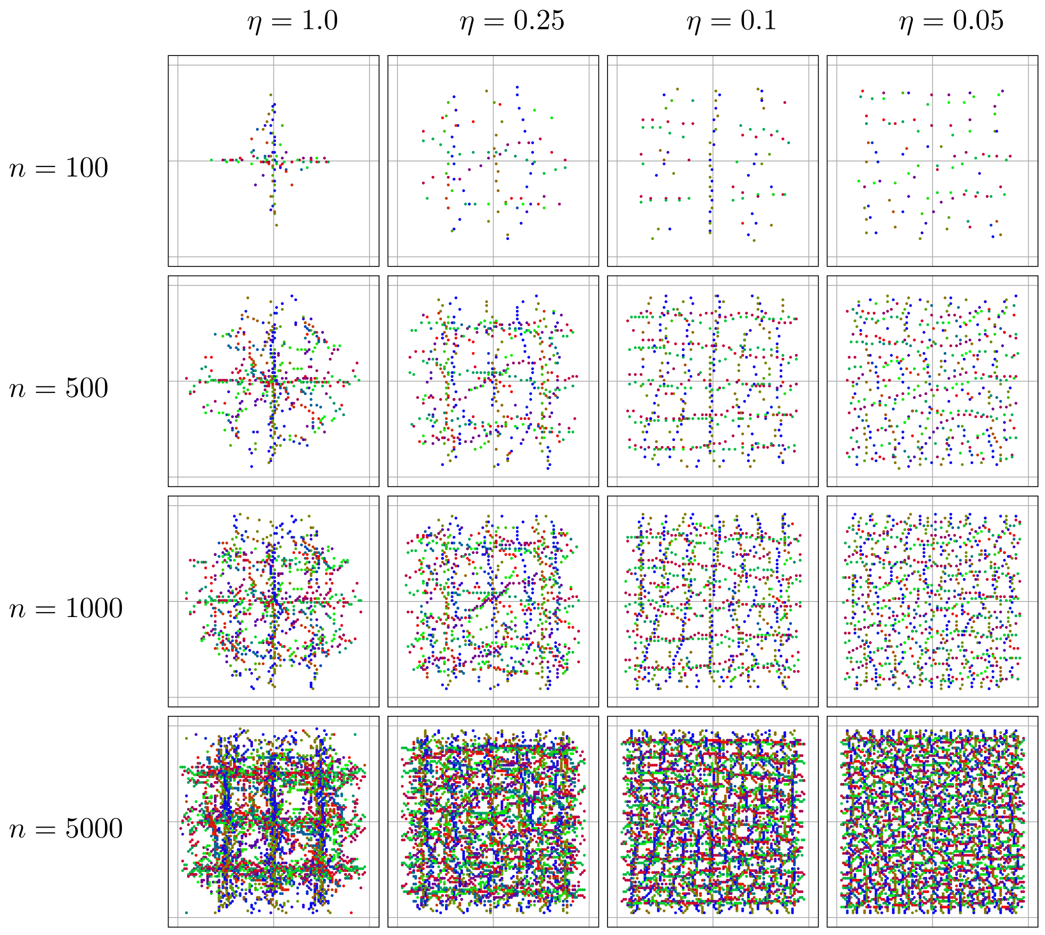


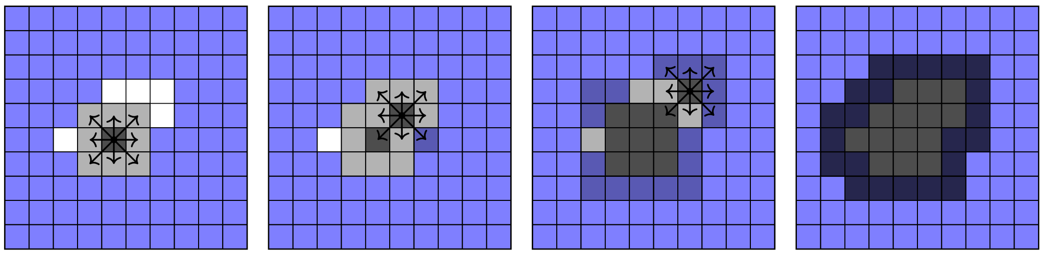
Figure 4.7: Resulting optimized sets for different ratios $\eta = r_{\min}/w_{\text{env}}$ and number of samples n. Clearly the samples adjust to the radius and converge more towards a Euclidean sets for small ratios.

To increase the number of dimensions that are practically computable we will propose a number of efficiency improvements, that make the computation feasible for a number of practically relevant systems.

The first is to apply a flood-fill algorithm to traverse the grid and only extend nodes if the distance is smaller than the previous distance. Figure 4.8 illustrates this approach. In addition to this classical flood-fill approach another modification based on that is a precomputed flood-fill, which precomputes the traversed cells in layer-like fashion as shown in Figure 4.9. Generally such flood-fill approaches are reasonable for cases in which the dynamics are not highly nonlinear or discontinuous, since this approach basically works due to limited reachable sets being connected.

For discontinuous or highly complex reachable sets the gridded approximation in conjunction with the flood-fill might cause errors in the optimization approach.

Another possibility is to only compute the distance function once and keep reusing this result afterwards. This approach was not implemented, but we want to state the general idea as a possible extension. Especially for nonlinear, more complex systems like the Reeds-Shepp distance lots of redundant computation can be avoided. In addition one could also use such an approach when no analytic steering function is available by numerically optimizing the trajectory in free-space. It should be noted though, that such an approach is mostly feasible if some invariance characteristics of the space can be exploited (e.g., translational invariance), to limit the number of optimization problems to solve.



(a) 1 check (b) 2 checks (c) 10 checks (d) 36 checks

Figure 4.8: Illustration of the flood-fill algorithm that can be used to speed up the optimization. An eight-neighborhood is used to expand the vertices. Note that by doing so, we also get back to previously checked cells or cells already in the queue. While this is only a small cost in lower dimensions, in higher dimensions this can have a big influence on the run time of the algorithm. The precomputed flood fill that is illustrated in Figure 4.9 prevents this. The blue cells are cells for which no expansion is performed (which is only realized, once we reach it), the dark gray cells are fully handled, the light gray cells are queued to be processed but have not been expanded yet and the white cells are untouched so far. The arrows visualize the expansion procedure.

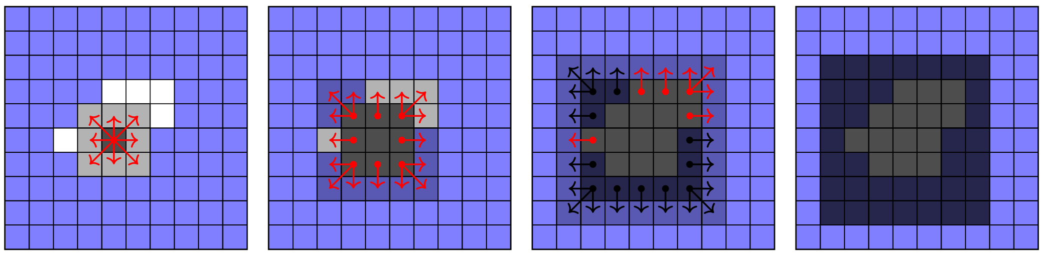

(a) 1 check (b) 9 checks (c) 25 checks (d) 49 checks

Figure 4.9: Illustration of a flood-fill variation the uses a precomputed layered neighbor list, to prevent the repeated checking of already traversed cells. Especially for high-dimensional spaces such an approach can speed up the optimization immensely. Only once all the cells in a layer have lower cost the algorithm stops.

4.2.1 Asymptotic optimality proof

Next, we show that the optimization algorithm reaches the same asymptotic dispersion for all driftless control-affine systems as Poccia's set reaches only for the Reeds-Shepp system. For the special Euclidean case, we show that the algorithm reaches the same asymptotically optimal dispersion as for example the Halton sequence. The same analysis that we perform here can be applied for many other systems, which shows theoretically that the optimization approach gives good results for a wide range of different spaces.

Throughout the discussion we assume that the distance function dist is symmetric and optimal. Also note that $\mathcal{R}(\boldsymbol{x}, r)$ denotes the *open* reachable set of cost (i.e., time or distance) r at $\boldsymbol{x}$.

Theorem 7. Under the assumption that the discretization of the space does not influence the placement of the samples, the dispersion of the set of samples $\mathcal{S}$ after n samples have been picked by Algorithm 6 can be bounded by

$$nV(\mathcal{R}(\boldsymbol{x}, d_n/2)) \leq V(\mathcal{X}) \tag{4.6}$$

where d_n denotes the dispersion defined for a distance function dist as in Equation (2.18). This yields for the D-dimensional Euclidean case an asymptotic behavior of

$$d_n \in \mathcal{O}\left(n^{-1/D}\right) \tag{4.7}$$

and for the driftless control-affine case

$$d_n \in \mathcal{O}\left(n^{-1/\tilde{D}}\right) \tag{4.8}$$

with $\tilde{D} = \sum_{i=1}^{D} w_i$, where w_i are the weights of the boxes approximating the reachable sets for driftless control-affine systems (see ball-box theorem [37, 87]).

Proof. To proof the asymptotic behavior of the algorithm, consider the case in which the discretization of the space has no effect on the placement of samples. Further we introduce some additional notation. Let d_n denote the dispersion when n samples have been picked, i.e., $|\mathcal{S}| = n$. For brevity's sake, we remove the explicit dist from the dispersion, but it is implied to be the distance function used in the algorithm, except if explicitly noted.

The key argument to analyze the asymptotic behavior of the algorithm is to realize that the n-th sample is, by construction, placed such that its distance to the closest neighbor is d_{n-1}. Due to that, after n samples have been picked, we can note that

$$d_{n-1} \leq \min_{\boldsymbol{y} \in \mathcal{S} \setminus \boldsymbol{x}} \operatorname{dist}(\boldsymbol{x}, \boldsymbol{y}) \leq 2d_{n-1} \quad \forall \boldsymbol{x} \in \mathcal{S}, \tag{4.9}$$

where the second inequality follows from the symmetry and optimality assumption of dist.

From the first inequality it follows that (note that the ball is open)

$$\mathcal{R}(\boldsymbol{x}, d_{n-1}) \cap \mathcal{S} = \emptyset \quad \forall \boldsymbol{x} \in \mathcal{S}. \tag{4.10}$$

In addition, again because of symmetry and optimality, the intersection of all balls of radius $d_{n-1}/2$ must be empty, i.e.,

$$\bigcap_{\boldsymbol{x} \in \mathcal{S}} \mathcal{R}(\boldsymbol{x}, d_{n-1}/2) = \emptyset. \tag{4.11}$$

Let $V(\cdot)$ denote the volume of a set. Note that $d_n \leq d_{n-1}$ and with n samples being in $\mathcal{S}$ we can state that

$$nV\big(\mathcal{R}(\boldsymbol{x}, d_n/2)\big) \leq nV\left(\bigcup_{\boldsymbol{x} \in \mathcal{S}} \mathcal{R}(\boldsymbol{x}, d_{n-1}/2)\right) = nV\big(\mathcal{R}(\boldsymbol{x}, d_{n-1}/2)\big) \leq V(\mathcal{X}) \tag{4.12}$$

must hold. To upper bound the dispersion for a number of samples n we would optimally use an explicit term for the volume $V\big(\mathcal{R}(\boldsymbol{x}, d_n/2)\big)$, but if no such term exists (as for general sub-Riemannian balls), we need to use a lower bound, i.e., by using the ball-box theorem [37].

We first consider the case of a D-dimensional space $\mathcal{X}$. In that case we get

$$ncd_n^D \leq V(\mathcal{X}) \tag{4.13}$$

and thus

$$d_n \leq \frac{V(\mathcal{X})^{1/D}}{c^{1/D} n^{1/D}} \in \mathcal{O}\left(n^{-1/D}\right), \tag{4.14}$$

which shows that in the Euclidean case, the asymptotic dispersion is optimal.

For the driftless control-affine case we can use the same argument as Poccia [36]. Under the assumption that the system is sufficiently regular we can find a parameter $A_{\max}$ such that

$$\mathrm{Box}^w\left(\boldsymbol{x}, \frac{d_n}{2A_{\max}}\right) \subseteq \mathcal{R}(\boldsymbol{x}, d_{n-1}/2) \tag{4.15}$$

and according to Lemma II.2 by Schmerling [15] the volume is given by

$$V\left(\mathrm{Box}^w\left(\boldsymbol{x}, \frac{d_n}{2A_{\max}}\right)\right) = \left(\frac{d_n}{2A_{\max}}\right)^{\tilde{D}} \tag{4.16}$$

with $\tilde{D} = \sum_{i=1}^{D} w_i$. Again we can rewrite (4.12) as

$$n\left(\frac{d_n}{2A_{\max}}\right)^{\tilde{D}} \leq V(\mathcal{X}) \tag{4.17}$$

and thus

$$d_n \leq \frac{V(\mathcal{X})^{1/\tilde{D}} 2A_{\max}}{n^{1/\tilde{D}}} \in \mathcal{O}\left(n^{-1/\tilde{D}}\right). \tag{4.18}$$

■

4.3 State Lattice Planner

We now describe a general state lattice planner methodology that has been implemented in OMPL to allow comparison between optimized sampling-based batch-algorithms like PRM* and FMT*, and the alternative approach to deterministic motion planning, state lattice planning. Note that this planner is no novel work, but that this rather represents a general framework that allows many different approaches to state lattice planning with only minor adjustments. In addition we use this framework to provide conditions on the primitives such that the framework provides resolution completeness.

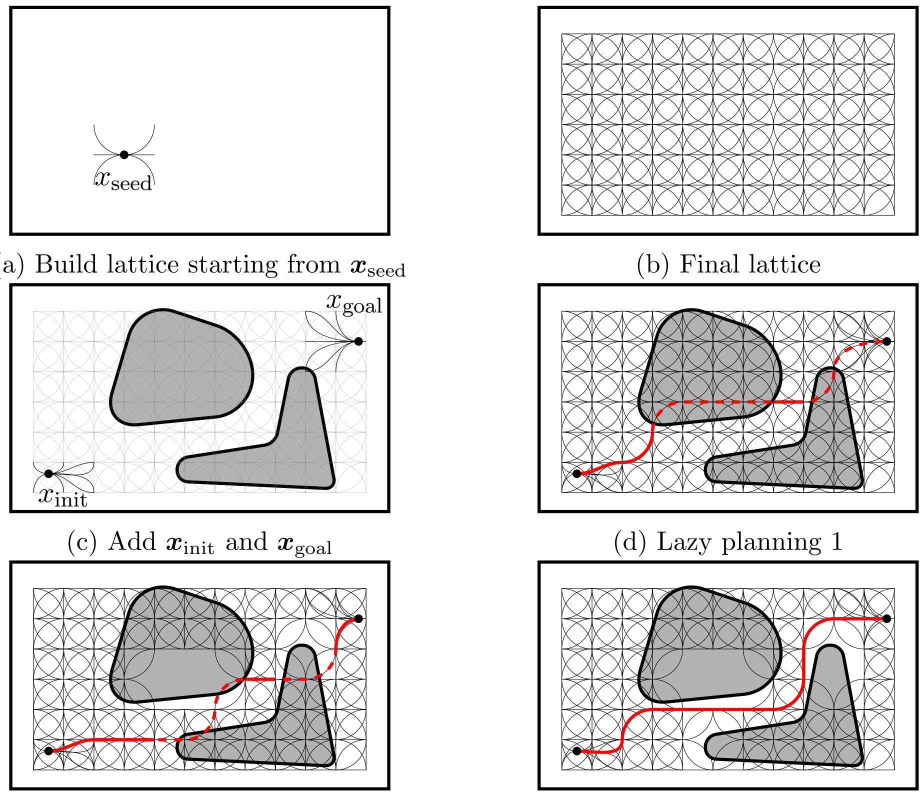
Graph-based state lattice planner

Algorithm 10 shows our generic approach. Figure 4.10 visualizes the general idea of the planner. Similarly to PRM, the algorithm can be split up into a learning phase and query phase. During the learning phase (lines 2–21) the algorithm builds the roadmap, by starting with a single vertex $\boldsymbol{x}_{\text{seed}}$ (Figure 4.10(a), lines 2–5) and expanding the vertex. The extension procedure can be seen in lines 6–21. For each motion primitive, first we check whether this motion primitive can originate from this state via PRIMITIVECHECK, if it can, we have to transform the motion primitive and get the resulting end vertex when following the primitive σ, starting from $\boldsymbol{x}_{\text{start}}$ (lines 9–10). The resulting state can additionally be adjusted via ADJUSTSTATE, which allows to use approximate lattices (i.e., discontinuous at some resolution). If this state $\tilde{\boldsymbol{x}}$ is inside the boundaries of the space and has not been added to $\mathcal{V}$ yet, we add it to the set of vertices and add it to the queue $\mathcal{Q}$ of states, which still have to be extended (lines 12–18). This procedure generates a full roadmap inside the environment bounds, see Figure 4.10(b).

The query phase (lines 22–37) is similar to LazyPRM [22]. First, we use a steering function to lazily (i.e., without collision checking) connect $\boldsymbol{x}_{\text{init}}$ and $\boldsymbol{x}_{\text{goal}}$ to the roadmap (Figure 4.10(c), lines 22–27). Afterwards we repeatedly find the shortest path and subsequently check for collisions along it (lines 28–29). If a collision is found we remove all the vertices and edges in collision (Figures 4.10(d) and 4.10(e), lines 31–34). If there is no collision along the path, we return it and end the algorithm (Figure 4.10(f), lines 29–30). Note that depending on the complexity of the collision checker it might be faster to first test all vertices (and perhaps edges) prior to starting the lazy planning. The optimal decision here depends on the size of the graph (which determines the speed of A*), the collision checker and runtime requirements.

Algorithm 10 Generic graph-based state lattice planner

```
 1: procedure StateLatticePlanner($\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}, \boldsymbol{x}_{\text{seed}}, \mathcal{P}$)
 2:     $\mathcal{V} \leftarrow \{\tilde{\boldsymbol{x}}_{\text{seed}}\}$
 3:     $\mathcal{Q} \leftarrow$ Queue()
 4:     Push($\mathcal{Q}, \tilde{\boldsymbol{x}}_{\text{seed}}$)
 5:     $\mathcal{E} \leftarrow \emptyset$
 6:     while $\mathcal{Q}$ is not empty do
 7:         $\tilde{\boldsymbol{x}}_{\text{start}} \leftarrow$ Pop($\mathcal{Q}$)
 8:         for $\sigma \in \mathcal{P}$ do
 9:             if PrimitiveCheck($\boldsymbol{x}_{\text{start}}, \sigma$) then
10:                 $\boldsymbol{x}_{\text{end}} \leftarrow$ GetEndState($\tilde{\boldsymbol{x}}_{\text{start}}, \sigma$)
11:                 $\tilde{\boldsymbol{x}}_{\text{end}} \leftarrow$ AdjustState($\boldsymbol{x}_{\text{end}}$)
12:                 if SatisfiesBounds($\mathcal{X}$, $\tilde{\boldsymbol{x}}_{\text{end}}$) then
13:                     if $\tilde{\boldsymbol{x}}_{\text{end}}$ not in $\mathcal{V}$ then
14:                         $\mathcal{V} \leftarrow \mathcal{V} \cup \{\tilde{\boldsymbol{x}}_{\text{end}}\}$
15:                         Push($\mathcal{Q}, \tilde{\boldsymbol{x}}_{\text{end}}$)
16:                         $\mathcal{E} \leftarrow \{(\tilde{\boldsymbol{x}}_{\text{start}}, \tilde{\boldsymbol{x}}_{\text{end}}, \sigma)\}$
17:                     end if
18:                 end if
19:             end if
20:         end for
21:     end while
22:     $\mathcal{V} \leftarrow \mathcal{V} \cup \{\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}\}$
23:     for $\boldsymbol{x} \in \{\boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}\}$ do
24:         for $\boldsymbol{y} \in$ Near($\boldsymbol{x}$) do
25:             $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\boldsymbol{x}, \boldsymbol{y})\}$
26:         end for
27:     end for
28:     while $\sigma \leftarrow$ ShortestPath($\mathcal{V}, \mathcal{E}, \boldsymbol{x}_{\text{init}}, \boldsymbol{x}_{\text{goal}}$) is not NoSolution do
29:         if ($\mathcal{V}_{\text{col}}, \mathcal{E}_{\text{col}}$) $\leftarrow$ GetCollisions($\sigma$) is $(\emptyset, \emptyset)$ then
30:             return $\sigma$
31:         else
32:             $\mathcal{V} \leftarrow \mathcal{V} \setminus \mathcal{V}_{\text{col}}$
33:             $\mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{E}_{\text{col}}$
34:         end if
35:     end while
36:     return NoSolution
37: end procedure
```



(a) Build lattice starting from $\boldsymbol{x}_{\text{seed}}$

(b) Final lattice

(c) Add $\boldsymbol{x}_{\text{init}}$ and $\boldsymbol{x}_{\text{goal}}$

(d) Lazy planning 1

(e) Lazy planning 2

(f) Final solution

Figure 4.10: Visualization of the state lattice planner. See text for description.

Implementation design

To understand the design choices it is necessary to have a general overview of how OMPL modularizes the motion planning problem. Two branches of motion planning are supported by OMPL: geometric motion planning and control-based motion planning. Geometric motion planning has no notion of the control space $\mathcal{U}$, while control-based motion planning performs the whole search in the control space. Interestingly, state lattice planning is somewhere between these two branches as the search is performed in the state space, while the motion primitives include the control information. This way efficient state space-based heuristics can speed up the search, while still conforming to the differential constraints.

In Section 2.4.3 we showed that both tree-based and graph-based approaches are possible. We have decided to create a graph-based approach. The motivation for this is two-fold. First, it allows the use of sophisticated graph libraries (i.e., Boost Graph Library [3]) and second, it allows for a lazy roadmap approach in which first the graph is built ignoring the collision checks and then, once a query and collision checker (and thus map) is specified, collision checking and path finding is performed together. To implement the state lattice planner we provide two components to OMPL:

- A new state space, that wraps around another state space (reuses the distance function and state definition) and adds the motion primitives as an extension to the state space. We call this state space *lattice state space*.
- A lazy graph-based planner that uses a lattice state space to build a roadmap considering the motion primitives and plans inside of it.

By separating the design into these two components we separate the idea of motion primitives from the exact planner. This way it would be straightforward to reuse such a lattice state space for a tree-based lattice planner as future work.

As hinted at above this framework also works as an initial step towards bridging the gap between the two separate branches of control-based and geometric motion planning. While not implemented, it would be a natural extension to now add interfaces that create motion primitives based on a control state space, but then use geometric motion planning to solve the problem.

4.3.1 Completeness for state lattice planners

In this section we show how the dispersion-based completeness framework introduced in Section 4.1 can also be applied for the state lattice algorithm in the previous chapter. Additionally we introduce an alternative way to prove completeness for our state lattice planner based on a property called δ-completeness. Both approaches naturally lead to motion primitive generation schemes, which will be introduced in Section 4.3.2.

The first key insight is, that the completeness analysis from before can still be used in the context of state lattice planners. This is because the primitives basically only provide the connectivity of an orthogonal grid for which the dispersion analysis can be used. This insight leads to the following theorem:

Theorem 8. Given a set of motion primitives $\mathcal{P}$, generated in a grid with dispersion d_{dist} with a connection radius

$$r > 2d_{\text{dist}} \tag{4.19}$$

using an optimal steering function and considering symmetric systems, Algorithm 10 solves all planning queries γ with clearance

$$\delta_{\text{dist}}(\gamma) > 2d_{\text{dist}}. \tag{4.20}$$

Proof. This follows directly from Theorem 6, since the motion primitives just represent a possibility of offline computing the connectivity of the samples, by exploiting invariance properties of the systems. ■

Let us next introduce the notion of δ-complete motion primitives.

Definition 12. A set $\mathcal{P}$ of motion primitives is called δ-complete, if for all primitive origins $\boldsymbol{p} \in \{\sigma_{\mathcal{P}}(0) | \sigma_{\mathcal{P}} \in \mathcal{P}\}$ and for all points $\boldsymbol{x} \in \partial\mathcal{R}(\boldsymbol{p}, \delta)$ there exists a primitive $\sigma_{\mathcal{P}} \in \mathcal{P}$ such that $\sigma_{\mathcal{P}}(0) = \boldsymbol{p}$ and $\sigma_{\mathcal{P}}(1) \in \mathcal{R}(\boldsymbol{x}, \delta) \setminus \boldsymbol{p}$.

Here $\partial\mathcal{R}$ denotes the border of the reachable set and $\sigma_{\mathcal{P}}$ is a single motion primitive from the set $\mathcal{P}$, with $\sigma_{\mathcal{P}}(0)$ being its start point and $\sigma_{\mathcal{P}}(1)$ being its end point. Figure 4.11 visualizes this notion of δ-completeness for motion primitives. Using this definition we can now show completeness for the state lattice approach.

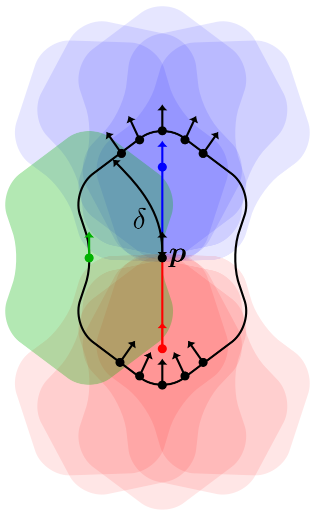


Figure 4.11: Visualization of the δ-completeness for motion primitives. Two primitives are shown (blue and red), which cover some of the reachable sets along the border (blue and red shaded areas respectively). For a set of primitives $\mathcal{P}$ to be δ-complete, all reachable sets along the border must be covered by some primitive. In this example the green set is not covered yet.

Theorem 9. Given a set of δ-complete, optimal motion primitives $\mathcal{P}$ and considering symmetric systems with optimal steering function, Algorithm 10 solves all planning queries γ with clearance

$$\delta_{\text{dist}}(\gamma) > 2\delta. \tag{4.21}$$

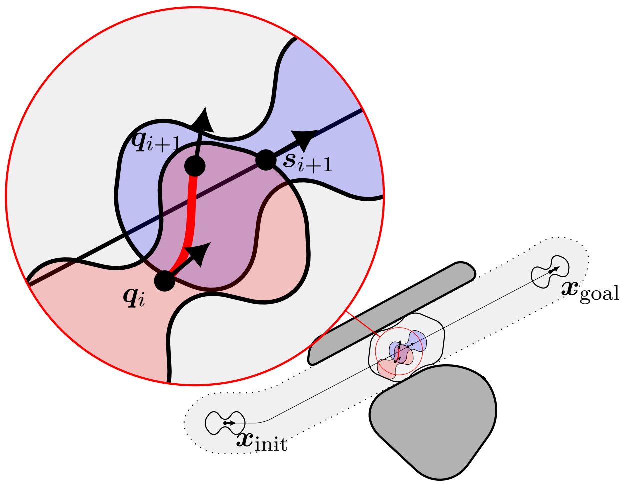


Figure 4.12: Visualization of the proof of completeness for the state lattice planner. The key insight is, that due to the δ-completeness (see Definition 12) of $\mathcal{P}$ there will always be a motion primitive leading into the reachable set around $s_i + 1$ (the blue area). In this case the red primitive leads to the next state q_{i+1} in this set.

Proof. First, δ-completeness implies that the set of vertices initially generated by the algorithm will generate a set of vertices with dispersion $d \leq \delta$. Thus, by using a connection radius of $r > \delta$ (line 24 in Algorithm 10) it is possible to connect both x_{init} and x_{goal} to the roadmap without any collision. Next, we follow a similar argument as in the dispersion-based completeness proof. Given a query γ with clearance $\delta(\gamma) > 2\delta$, there exists a solution σ with $\mathcal{R}(s_i, 2\delta) \in \mathcal{X}_{\text{free}}, \forall s_i \in \sigma([0,1])$. Let q_0 be the state that x_{init} is connected to. Let s_1 be the state along σ such that it lies on the border of $\mathcal{R}(q_0, \delta)$. Note that such a sample must exist (as long as σ is not fully contained in $\mathcal{R}(x_{\text{init}}, \delta)$), because due to symmetry at least $x_{\text{init}} \in \mathcal{R}(q_0, \delta)$ lies either on the border or inside $x_{\text{init}} \in \mathcal{R}(q_0, \delta)$. Now we can use the δ-completeness of $\mathcal{P}$, as per definition, there must exist a motion primitive that ends inside of $\mathcal{R}(s_1, \delta)$ and which is collision-free due to a clearance of 2δ around s_1 and optimality of the motion primitive (here we use the same argument as in the proof of Theorem 6, that a symmetric and optimal path between any two points of a reachable set with radius δ, will not leave the larger reachable set with radius 2δ). The same procedure can now be repeated with the end state q_1 of the primitive and the next state along the path s_2, until we reach q_N which is connected to x_{goal}. The idea of the proof is visualized in Figure 4.12. ■

4.3.2 Primitive generation

Given the previous proof of completeness, the only thing left is a description on how to generate the motion primitive set $\mathcal{P}$. To allow a fair comparison we limit ourselves to motion primitives ending on grid points, i.e., we are not doing any approximations and get a continuous solution without any approximations. We also assume, that we have a steering function available to connect start and goal to the graph correctly. Note though that the advantage of the motion primitive approach is that it does not require an analytic steering function to be viable (i.e., we could use numeric solvers to connect start and goal and still enable fast planning via motion primitives).

Both of the following approaches assume that a grid of vertices is given and we want to find a way of connecting the vertices to achieve a good trade-off between solution quality and complexity. Generally as a first step for a given grid, the dispersion of the grid given a distance function dist should be assessed, since this gives a lower bound on the achievable completeness (see Theorem 8 and 9). This dispersion d_{dist} can then be used to obtain highest efficiency in the following generation schemes.

Connection strategy-based

The first approach is based on using a connection strategy to precompute the motion primitives. Similar to the connection strategies discussed in Section 2.4.1 two main possibilities exist here: a fixed connection radius r or to use k-nearest neighbors to generate the motion primitives.

In the first case completeness of the state lattice planners is ensured by using

$$r > 2d_{\text{dist}}. \tag{4.22}$$

In the second case the exact bound for k is not known so far. A good starting point, if the number of vertices n is known (which is not the case for variable size environments, which are supported by a state lattice planner in general), is given by Karaman [2] for the Euclidean case

$$k > e(1 + 1/D)\log(n). \tag{4.23}$$

Note though, that this is clearly not a perfect strategy yet. First, n is not known a priori in the general state lattice case (the environment size and primitives define the number of vertices) and thus the formula is difficult to apply when generating the motion primitives (without upper bounding n). In addition it seems unnecessary that k even depends on n, since the density of the samples does not change in the state lattice approach. Probably the best k depends on the topology of the space and further theoretical work is required in this direction.

Given that the primitive computation is performed offline anyways, the better choice seems to be to use a fixed connection radius, since completeness for this radius has been theoretically proven (see Section 4.3.1).

Minimal δ-complete primitive set

Next, we propose an alternative connection strategy based on Theorem 8, that tries to find a minimal set of primitives that ensures δ-completeness.

Algorithm 11 shows the formal description of the approach. As input the algorithm takes the potential grid points $\mathcal{G}$ to connect to, the origin points $\mathcal{G}_0 \subseteq \mathcal{G}$ (this depends for which dimensions invariance can be used, i.e., for the Reeds-Shepp case these are all the grid points at $(0, 0)$ with all different initial heading directions), a fine grid for numerically approximating the reachable set $\mathcal{G}_{\text{fine}}$, the desired completeness δ and finally ε which is used to numerically approximate the reachable sets. $\mathcal{G}_{\text{fine}}$ and ε together define the numerical approximation of the border of the reachable set as

$$\partial\mathcal{R}(\boldsymbol{x}, \delta) \approx \mathcal{G}_{\text{fine}} \cap \{\mathcal{R}(\boldsymbol{x}, (1+\varepsilon)\delta) \setminus \mathcal{R}(\boldsymbol{x}, \delta))\}. \tag{4.24}$$

The algorithm is performed for each of the origin states of the grid $\mathcal{G}_0$. For each of these points $\boldsymbol{x}_0$ we need to find a set of δ-complete primitives.

First, we numerically find all border points of $\mathcal{R}(\boldsymbol{x}_0, \delta)$ (lines 3–8). We next check for each of the grid points of $\mathcal{G}$ which border points it covers or equivalently are inside of $\mathcal{R}(\boldsymbol{x}_{\text{grid}}, \delta)$ (condition in line 14). The whole process is comprised of lines 9–20, since some additional book keeping has to be done for the next step. The collection (i.e., set of sets) $\mathcal{T}$ is used to keep track of the sets of border points that are covered by a single grid point. The associated grid point is saved in the mapping ξ.

The problem that we are left with after this step is to find the minimum number of grid points, such that all border points are covered. This problem is equivalent to a minimum set cover problem, which is NP-complete [88], i.e., we ignore the link to grid points first, and only try to find the minimum set cover of the collection $\mathcal{T}$ and the universe $\partial\mathcal{R}$. Different solvers could be used here, but we use a simple greedy approach, that picks the set with most border points, remove all the covered border points, and proceed until all border points are covered (see Algorithm 12 and description below). While this approach does not give an optimal solution, it empirically seems to work well to compute a set of primitives with low complexity. The given algorithm is kept general by allowing any minimum set cover algorithm to be used (line 21). Note that if a δ-completeness of δ is not obtainable with some grid, some border points will not be part of $\mathcal{T}$ and thus no set cover can be found and we can terminate the algorithm.

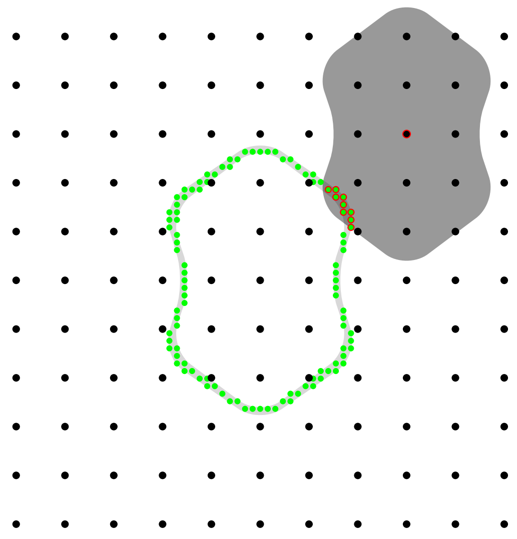


Figure 4.13: Visualization of the general idea behind the generation scheme. The green dots visualize the discrete approximation of the border of the δ-reachable set $\partial\mathcal{R}$. The black dots are the grid points $\mathcal{G}$. For each of the grid points we try to find the border points covered by it. The red circled points visualize this for one grid point and the border points covered by it. Subsequently we try to find the minimum number of grid points, such that all border points are covered, which can be formulated as a minimum set cover problem.

Finally, when we find a set cover, we use the previously stored mapping ξ to obtain the associated end points and generate the actual motion primitives and add them to the set $\mathcal{P}$ (lines 22–25).

The greedy algorithm that is used to approximate the minimum set cover is given as Algorithm 12. As input, it takes a collection $\mathcal{T}$ and a set of elements $\mathcal{U}$, called the universe. In each step the algorithm takes the largest set of $\mathcal{T}$ (i.e., largest cardinality in line 4) and adds it to the current cover (line 5). Subsequently the chosen set is removed from the collection $\mathcal{T}$. Note that in this step, we are not only removing the set $\mathcal{T}_{\text{temp}}$ itself from $\mathcal{T}$, but also each element $\boldsymbol{t} \in \mathcal{T}_{\text{temp}}$ from each set it appears in from $\mathcal{T}$. This is important, since otherwise the cardinality in subsequent steps would not correctly take into account the previously chosen sets. This procedure is repeated until the whole universe $\mathcal{U}$ is covered by the union (i.e., all the unique elements) of $\mathcal{T}_{\text{cover}}$ (see condition in line 3).

Algorithm 11 δ-complete primitive generation

```
procedure GenerateDeltaCompletePrimitives($\mathcal{G}, \mathcal{G}_0 \subseteq \mathcal{G}, \mathcal{G}_{\text{fine}}, \delta, \varepsilon$)
    for $\boldsymbol{x}_0 \in \mathcal{G}_0 \subseteq \mathcal{G}$ do
        $\partial\mathcal{R} \leftarrow \{\}$
        for $\boldsymbol{y} \in \mathcal{G}_{\text{fine}}$ do
            if $\delta < \text{dist}(\boldsymbol{x}, \boldsymbol{y}) < (1+\varepsilon)\delta$ then
                $\partial\mathcal{R} \leftarrow \partial\mathcal{R} \cup \{\boldsymbol{y}\}$
            end if
        end for
        $\mathcal{T} \leftarrow \{\}$                                ▷ Collection of sets
        $\xi : \mathcal{T} \rightarrow \mathcal{G}$                  ▷ Mapping from $\mathcal{T}$ to $\mathcal{G}$
        for $\boldsymbol{x}_{\text{grid}} \in \mathcal{G} \setminus \boldsymbol{x}_0$ do
            $\mathcal{T}_{\text{temp}} \leftarrow \{\}$
            for $\boldsymbol{x}_{\text{border}} \in \partial\mathcal{R}$ do
                if $\text{dist}(\boldsymbol{x}_{\text{grid}}, \boldsymbol{x}_{\text{border}}) < \delta$ then
                    $\mathcal{T}_{\text{temp}} \leftarrow \mathcal{T}_{\text{temp}} \cup \{\boldsymbol{x}_{\text{border}}\}$
                end if
            end for
            $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathcal{T}_{\text{temp}}\}$        ▷ Add $\mathcal{T}_{\text{temp}}$ to collection $\mathcal{T}$
            $\xi(\mathcal{T}_{\text{temp}}) \leftarrow \boldsymbol{x}_{\text{grid}}$             ▷ Link $\boldsymbol{x}_{\text{grid}}$ to $\mathcal{T}_{\text{temp}}$
        end for
        $\mathcal{T}_{\text{cover}} \leftarrow$ MinimumSetCover($\mathcal{T}, \partial\mathcal{R}$)
        $\mathcal{P} \leftarrow \{\}$
        for $\mathcal{T}_{\text{temp}} \in \mathcal{T}_{\text{cover}}$ do
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{steer}(\boldsymbol{x}_0, \xi(\mathcal{T}_{\text{temp}}))\}$
        end for
    end for
    return $\mathcal{P}$
end procedure
```

Algorithm 12 Greedy minimum set cover

```
procedure GreedyMinimumSetCover($\mathcal{T}$,$\mathcal{U}$)
    $\mathcal{T}_{\text{cover}} \leftarrow \{\}$
    while $\bigcup \mathcal{T}_{\text{cover}} \neq \mathcal{U}$ do
        $\mathcal{T}_{\text{temp}} \leftarrow \arg\max_{\mathcal{T}_i \in \mathcal{T}} |\mathcal{T}_i|$
        $\mathcal{T}_{\text{cover}} \leftarrow \mathcal{T}_{\text{cover}} \cup \mathcal{T}_{\text{temp}}$
        $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{T}_{\text{temp}}$
    end while
end procedure
```

5 Experiments and Discussion

This chapter will first introduce the metrics used to compare the approaches, which baselines are compared and other parameters relevant to the results. Subsequently, the results will be shown and discussed. Finally, we provide a discussion of the relation between probabilistic roadmaps and state lattice planners, highlighting the close relationship between the two.

5.1 Experiments

5.1.1 Environments

Reeds-Shepp car in 2D grid

The Reeds-Shepp car was already introduced in Section 2.3.1. Planning is performed in 2D grid maps with a polygonal footprint. The collision checker is visualized in Figure 5.1. The paths are checked using the same collision checker using a resolution that ensured no collisions happening in practice. The parameter defining these environments is $\eta = r_{\min}/w_{\text{env}}$ as that defines both the Poccia samples and the optimized samples.

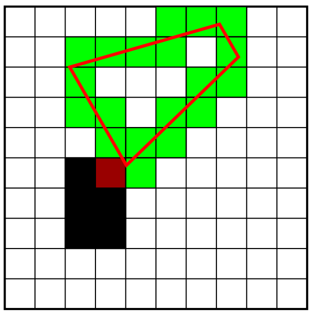
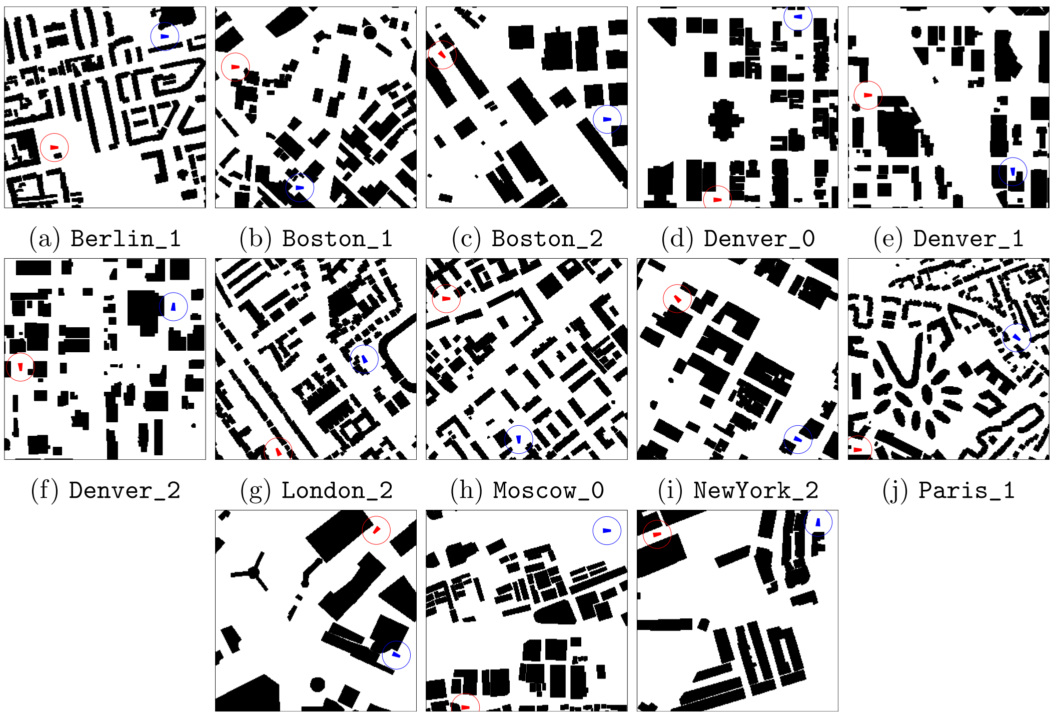


Figure 5.1: Collision checker for the 2D grid environments. The continuous description of the car will be overapproximated by tracing out the outline (red line), which is checked for any collisions. The black squares indicate the obstacles, the green squares the checked, obstacle-free cells and the red square indicates the found collision.

To prevent any bias introduced by the maps and to gain more insight into different characteristics of the algorithm three kinds of maps have been used:

- A subset of the benchmark problems created by Sturtevant [89]. The utilized maps with their respective names are shown in Figure 5.2. The queries for these maps have been both handpicked (see Figure 5.2) and randomized. The main purpose of these maps is to show the applicability in realistic cluttered environments.
- A set of custom made benchmark maps highlighting special motion planning cases (see Figure 5.3).
- Random maps 5.4 with small rectangle obstacles, placed randomly to reach a certain coverage (see Figure 5.4). The main purpose of these maps is to minimize any bias in the dataset and plan paths of different characteristic.



(a) `Berlin_1` (b) `Boston_1` (c) `Boston_2` (d) `Denver_0` (e) `Denver_1`
(f) `Denver_2` (g) `London_2` (h) `Moscow_0` (i) `NewYork_2` (j) `Paris_1`
(k) `Shanghai_0` (l) `Shanghai_1` (m) `Sydney_2`

Figure 5.2: Subset of city maps by Sturtevant [89] used throughout the experiments. The red (circled footprint) and blue poses indicate the start and goal pose for the custom scenarios, respectively.

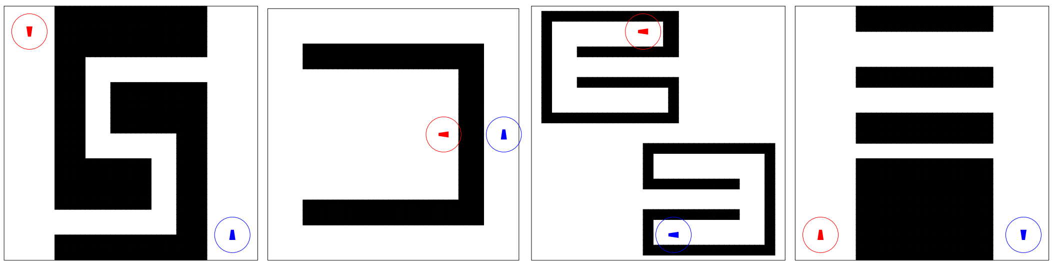


Figure 5.3: Custom environments for the Reeds-Shepp case.

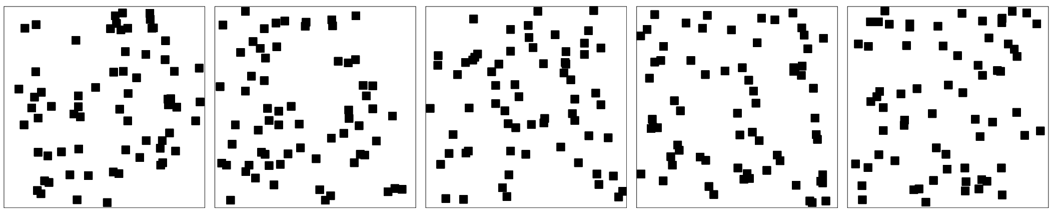


Figure 5.4: Random grid maps for the Reeds-Shepp case. The maps are generated by randomly placing small rectangular obstacles until a certain coverage is obtained. Only queries with a specified minimum cost are used for the experiments to prevent simple queries from dominating the benchmark results.

Kinematic chain in the plane

The second environment in which tests have been performed is a simple D-dimensional kinematic chain in the plane. Figure 5.5 shows the system and Figure 5.6 shows the collision checker.

Two kinds of maps have been tested. Random maps, shown in Figure 5.8, and custom-made maps, shown in Figure 5.7.

5.1.2 Parameters

For all the sampling-based experiments we use k-nearest neighbors as the connection strategy. We use

$$k = e\left(1 + \frac{1}{D}\right)\log n \tag{5.1}$$

as the default k value dependent on the number of samples n, which ensures asymptotic optimality in the general case as proven by Karaman [2].

We use either a fixed number of sample attempts (denoted by n_{all}) or a fixed number of collision-free samples (denoted by n_{valid}) as our termination criterion.

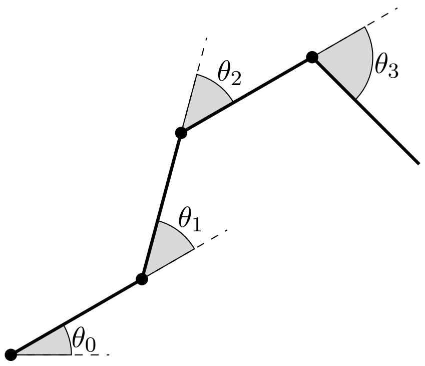
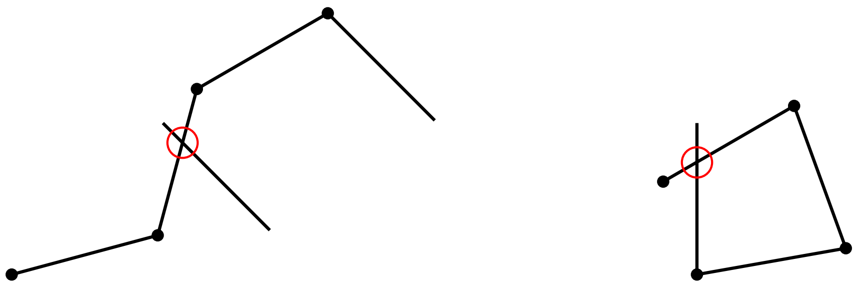


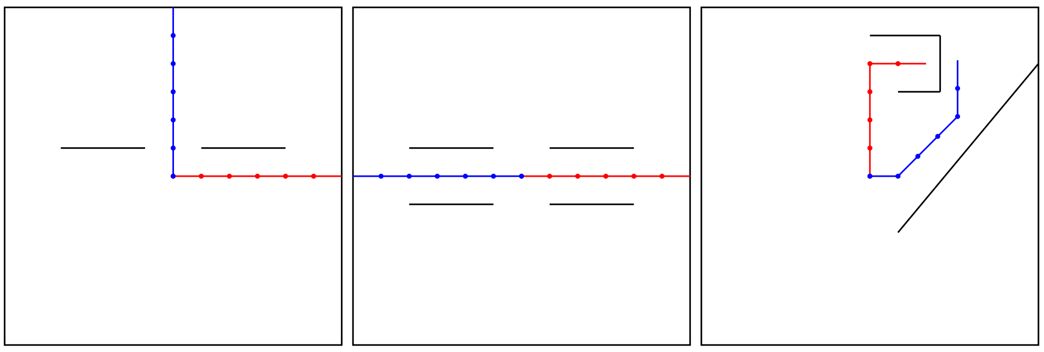
Figure 5.5: Kinematic chain in the plane. In this case a 4D chain is shown, that is defined by its four joint angles.



(a) Environment collision

(b) Self collision

Figure 5.6: Collision checker for the kinematic chain. Two possible kinds of collisions are checked. The red circle indicates the collision between the arm and the environment in (a), and a self-collision of the kinematic chain in (b).



(a) `Two_Walls`

(b) `Four_Walls`

(c) `Workspace`

Figure 5.7: Custom maps for the kinematic chain case.

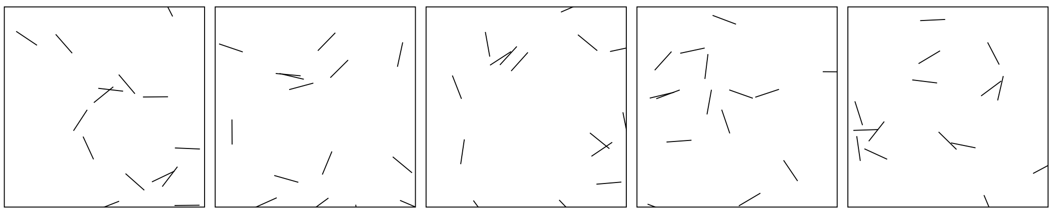


Figure 5.8: Random maps for the kinematic chain. The maps are generated by placing 15 walls of length $l = 1.5\,\mathrm{m}$ in random position and orientation inside the workspace.

We do not use time-based benchmarks, since the results in this case depend heavily on the exact implementation (i.e., how often do we reload precomputed samples from the HDD). By using samples as the deciding criterion, we focus on the quality of the samples, not the implementation, which is the focus of this work.

To assess the quality of the samplers and the motion primitive sets, we use two metrics:

- success rate after the termination criterion is fulfilled, and
- cost (i.e., path length in m) of the solution.

Additionally, for the randomized, quantitative benchmarks we introduce a new score metric that tries to capture the general quality of the solution for many random runs. The metric works by keeping track of a score between every pair of samplers. Assume two samplers A and B being run on the same n random queries. For each query the score of the better sampler will be increased by 1 and the score of the sampler with the worse score will be decreased by 1. If both samplers achieve the same result (e.g., both fail to find a solution) the score of both remains unchanged. Clearly this score will generally increase the higher n is. To facilitate the interpretation of the results, we normalize the final score by $\sqrt{n}$, which would be the standard deviation of such a score if the winning probability of both samplers for each query is 0.5. This way, a score of greater than 1, can be interpreted as a significant advantage, while a score below -1 can be interpreted as a significant disadvantage.

5.1.3 Baselines

To compare the sampling optimization approach to other approaches we use 4 different baselines:

- i.i.d. samples, which is still the default approach of many sampling-based motion planners,

- Halton samples, as a general low-dispersion sequence,
- Poccia, as an optimized set of samples for the Reeds-Shepp case, and
- state lattice approach, as an alternative approach to planning with differential constraints (see below for more details on the primitive sets $\mathcal{P}$).

For Halton samples we use the first D numbers which are mutually prime as the base numbers. For Poccia, the procedure described in Section 3.1 has been used to generate sets of approximately the desired number of samples n_{all}. Note also, that since Poccia's set does not provide a sequence by construction, we randomize the order of the set for the use in OMPL. Additionally, we omit Poccia in cases where we plan based on n_{valid}, since Poccia would always be at a disadvantage in such cases. Finally, in cases in which we compare to the state lattice approach, we also only use n_{all}, such that the potential maximum number of vertices in the lattice roadmap is the same as the number of sample attempts from the other samplers.

Since the quality of the state lattice approach depends a lot on the utilized set of primitives, we compare a number of different sets of primitives. We include the lattice approach with k-nearest neighbor primitives ($k = 20$) in Experiment 6. This conforms with the connection strategy for the other samplers and thus the complexity of the A* query should be similar. The two primitive sets for this case are shown in Figure 5.9. In both cases 8 heading angles are used and the grid was increased for the larger environment to maintain short planning times (2 m grid for an environment size of 50 m, 5 m grid for a size of 100 m).

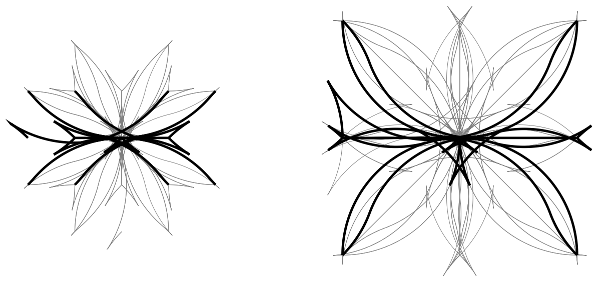
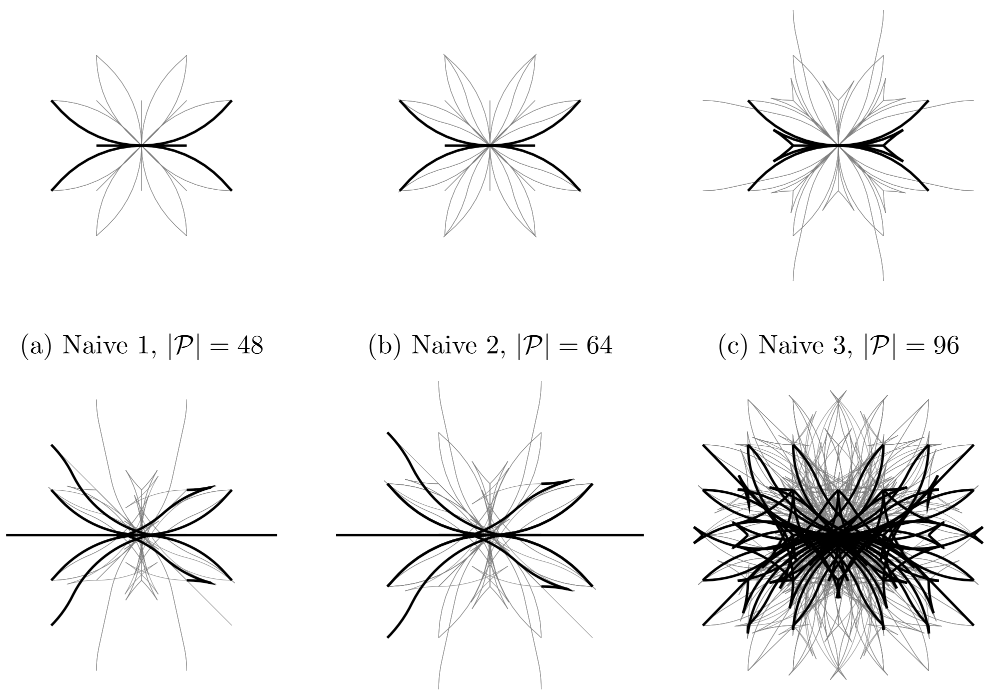

(a) 2 m grid (b) 5 m grid

Figure 5.9: Motion primitive sets used in Experiment 6 generates with the k-nearest strategy with $k = 20$.

The primitive sets used to analyze the difference between different generation schemes is shown in Figure 5.10. For the comparison we use some naively generated sets (a)–(c), which try to avoid cusps, our optimized set (d), a combined set from

(a) and (d), shown in (e), and finally a set generated by using a dispersion-based fixed connection radius (f). Generally, we would expect the biggest set to perform the best and the smallest set to perform the worst. All sets use the same grid of size 2 m with 8 uniform heading angles. For (d), $\delta = d_{\text{rs}}$ was used, while for (f) $r = 2d_{\text{rs}}$ was used. $d_{\text{rs}} \approx 4\,\text{m}$ was numerically computed for such a lattice of infinite size, by applying a similar algorithm as the one used for the dispersion optimization (i.e., computing the distance tensor without adding any samples).



(a) Naive 1, $|\mathcal{P}| = 48$ (b) Naive 2, $|\mathcal{P}| = 64$ (c) Naive 3, $|\mathcal{P}| = 96$

(d) Optimized, $|\mathcal{P}| = 90$ (e) Combined, $|\mathcal{P}| = 138$ (f) Radius, $|\mathcal{P}| = 752$

Figure 5.10: Motion primitive sets generated by different methods and used throughout the experiments. All maps use a grid of 2 m and 8 uniform heading angles. See text for further description.

5.2 Results and Discussion

5.2.1 Dispersion optimization

First, we show results comparing the dispersion optimization approach to the other sampling approaches. We start with qualitative experiments for both environments.

We show some results for the city maps for the Reeds-Shepp car and the custom scenarios for both the Reeds-Shepp car and kinematic chain. Subsequently we show quantitative results to give an idea of the general performance difference between the different samplers.

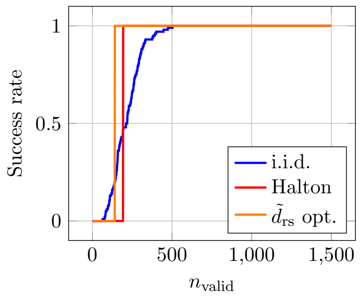
Qualitative

For the qualitative experiments of the Reeds-Shepp car we limit ourselves to the case $\eta = 0.1$, which seems like a practical case (i.e., 5 m turning radius in a 50 m environment). Results for other η values are shown subsequently for the quantitative experiments.

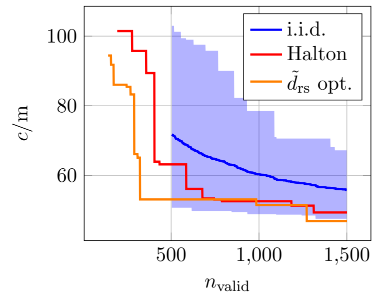
Experiment 3 (Reeds-Shepp car, city maps)**.** Figure 5.11 shows the results on the maps `Moscow_0` and `Denver_2`. It can be seen that by using the optimized samples the first solution is found earlier and that the cost is lower than when using i.i.d. or Halton samples in most cases. Of course such qualitative behavior depends a lot on the chosen query and map. Thus it is expected that there are also cases in which the results look reversed. Table 5.1 summarizes the final cost after $n_{\text{all}} = 1500$ samples for all maps, to give a more complete picture of these results. Clearly there are also cases in which Halton yields a better cost, but still, in most cases the optimized samples are best. The fact that the results are not completely uniform show that random factors like the start and end pose, as well as the obstacle placement have a large influence on the result. This motivates the more extensive randomized experiments shown later, which can be interpreted as marginalizing over obstacle configurations, only analyzing the effect of the samples.

	i.i.d.						
	Suc	Min	Avg	Max	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.
`Berlin_1`	0.00	·	·	·	∞	∞	∞
`Boston_1`	1.00	62.51	70.07	78.07	66.46	65.10	**63.50**
`Boston_2`	1.00	50.98	56.10	70.51	**53.15**	61.59	58.32
`Denver_0`	0.96	62.30	70.26	85.45	63.21	**63.17**	64.24
`Denver_1`	0.98	48.43	59.39	70.56	**53.47**	53.51	60.45
`Denver_2`	1.00	49.07	55.14	64.92	52.73	54.07	**50.94**
`London_2`	0.49	87.24	·	·	104.97	**92.80**	∞
`Moscow_0`	0.96	53.83	71.98	117.10	63.15	84.00	**53.08**
`NewYork_2`	1.00	67.59	70.99	74.99	71.54	69.82	**69.02**
`Paris_1`	0.01	167.51	·	·	∞	∞	∞
`Shanghai_1`	0.91	78.62	93.79	123.62	91.46	94.73	**88.45**
`Shanghai_0`	1.00	52.24	70.56	82.10	71.28	**59.11**	72.12
`Sydney_2`	0.86	70.82	81.90	100.73	80.81	**74.05**	77.08

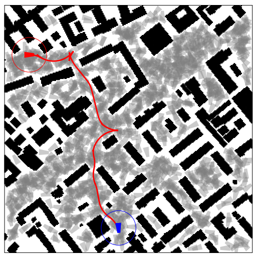
Table 5.1: Full results for all the city maps for the queries shown in Figure 5.2 and a fixed number of samples $n_{\mathrm{all}} = 1500$. The best result is highlighted in bold (ignoring the minimum i.i.d. case). It can be seen that while the optimized is the best most often, it depends a lot on the exact query, which set performs the best.



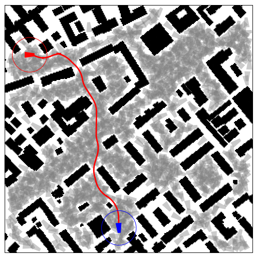
(a) Success rate



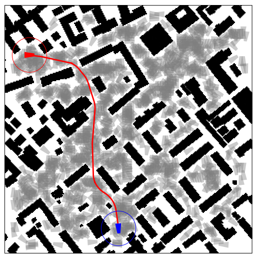
(b) Cost



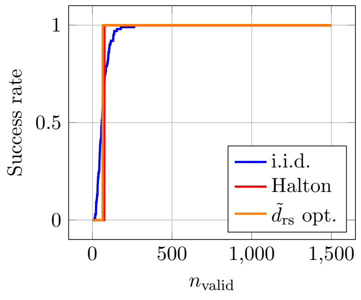
(c) i.i.d.



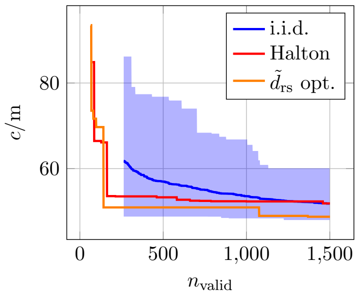
(d) Halton



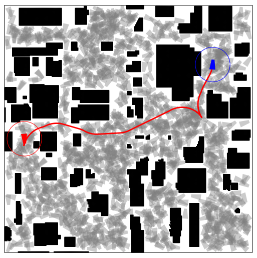
(e) $\tilde{d}_{\text{rs}}$ opt.



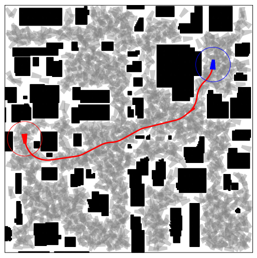
(f) Success rate



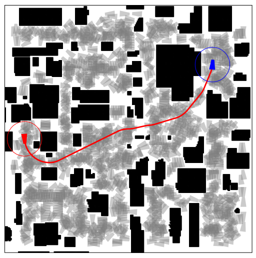
(g) Cost



(h) i.i.d.



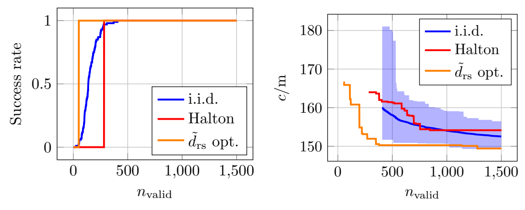
(i) Halton



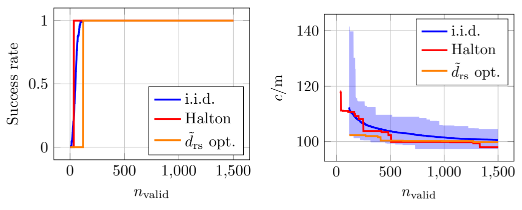
(j) $\tilde{d}_{\text{rs}}$ opt.

Figure 5.11: Qualitative comparison of different sampling approaches on two maps. (a) to (e) show the results for `Moscow_0` and (f) to (j) show the results for `Denver_2`. The resulting paths and sample sets for $n_{\text{valid}} = 1500$ are shown. The shaded area in the cost progression plots show the minimum and maximum cost out of 50 i.i.d. runs.

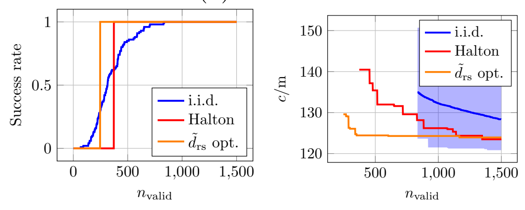
Experiment 4 (Reeds-Shepp car, custom maps)**.** Figure 5.12 shows cost and success rate progressions for the custom maps from Figure 5.3. The results show, that the optimized samples find the solution first in all cases, except the `U` environment. It can also be seen in the cost progression that it not only finds the first solution, but also the solution with lowest cost, for small number of samples. In the end Halton normally catches up and all three samplers converge to the optimum cost. These results confirm the theoretical advantage of achieving lower dispersion with the same number of samples.



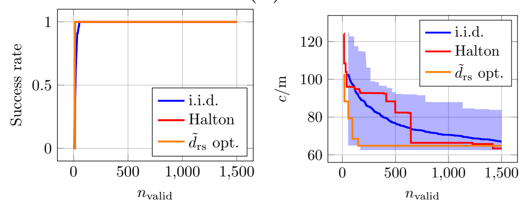
(a) `Narrow`.



(b) `U`



(c) `Double_Bug_trap`

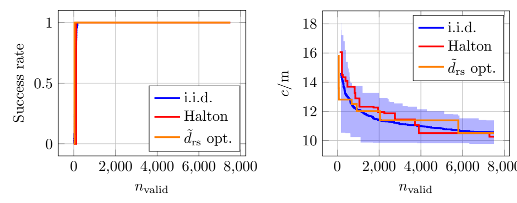


(d) `Corridors`

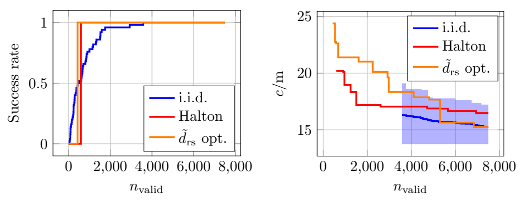
Figure 5.12: Resulting cost and success rate progression on the custom maps. It can be seen that in all cases, except `U` the optimized sequence is the first to find the solution. Additionally the cost obtained by the optimized samples is much lower in the range of 200–700 samples, showing that the narrow corridors giving the shortest path are found with fewer samples compared to the other samplers, which catch up later.

Experiment 5 (Kinematic chain, custom maps)**.** Figure 5.13 shows the results for the custom maps for the kinematic chain. The space was $\{[-\pi, \pi]/\sim\}^6$ (i.e., 6D with wrap-around) and self-collisions were allowed for these experiments. Compared with the previous results in the Reeds-Shepp case there is only a slight advantage of the optimized sequence which finds a solution with the fewest samples in all cases, but it is difficult to see any clear trend regarding the cost from these examples.

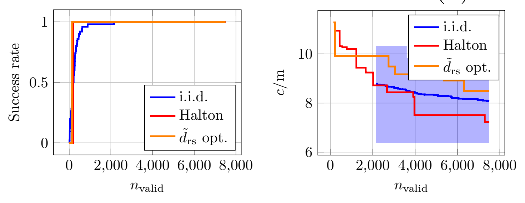
In general there is no theoretically proven advantage of the optimized sequence over Halton for this case, but even i.i.d. shows good results, although achieving a success rate of 100% clearly requires more samples except for the simple `Two_Walls` map. From that, we can observe the theoretical advantage of deterministic samples in general, but clearly further experiments are required to gain more insight. Generally this space seems to profit less from the deterministic samples and the optimization procedure. An interpretation of these results is given in the context of the quantitative results of Experiment 8.



(a) `Two_Walls`



(b) `Four_Walls`



(c) `Workspace`

Figure 5.13: Resulting cost and success rate progression on the custom maps. See text for description.

Generally all these qualitative results show promising results for deterministic sampling in general. An additional benefit from the optimization procedure that we have introduced in Section 4.2 can be observed in both Reeds-Shepp experiments. Of course, such handpicked examples do not approximate nearly the distribution of possible queries that a motion planner would generally have to solve and only give some intuitive indication of possible benefits. Hence, we next go over more thorough quantitative results to show the benefits when being faced with many different queries.

Quantitative

Experiment 6 (Reeds-Shepp car, city maps)**.** Table 5.2 shows both, the performance score as well as the success rate histograms. For these experiments we also include the state lattice planner with the two k-nearest neighbor-based motion primitive sets, which are shown in Figure 5.9. The 2 m primitives are used in the $\eta = 0.1$ case and the 5 m primitives in the $\eta = 0.05$ case. To allow a fair comparison, the number of samples of the other planners were adjusted to the same number of grid samples.

Our optimization approach outperforms all the baselines. It can also be seen that generally Halton clearly performs better than i.i.d. samples, and Poccia's approach does not seem to offer any major advantage over Halton. Also looking at the success rate, we can see that for the more constrained $\eta = 0.1$ case, the optimized approach shows a significant advantage, while for $\eta = 0.01$ all approaches, except for the state lattice planner achieve the same high success rate.

In both metrics, the state lattice approach gives worse results than the other approaches. There are two possible reasons for this: first, the connection of start and end are performed in lazy fashion for the state lattice planner. Thus, it can happen that all the connected vertices are invalid and no further connections are attempted; second, the placement of samples might be suboptimal, i.e., 8 heading angles might not be the best choice for the given turning radius and environment size. This shows a problem with the state lattice approach as arbitrary heading angles have the problem of not leading on top of grid cells, not allowing for simple, straight trajectories. Thus finding the right trade off between grid definition and complexity of the primitives is a difficult problem. Finally, it should be noted, that a fixed number of vertices, might not be a fair basis for comparison, since the main advantage of the state lattice approach is the precomputed connectivity, which will not show in this kind of comparison. Still, given the PRM framework, this experiment shows, that having freedom in the sample placement, not being constrained to an orthogonal and regular grid, gives better performance.

Experiment 7 (Reeds-Shepp car, random maps)**.** Table 5.3 shows the scores for 1000 random maps. In all cases the optimized sequence clearly gets the best score, which again confirms the theoretical advantages in practice. In general the same trend can be seen for all η values. As expected Poccia also outperforms Halton, which makes sense, since the set is tailored to the Reeds-Shepp case. Finally, the advantage of the optimized set seems to be smaller for $\eta = 1.0$, this could be because of the general low success rate for this case (this is because for such a highly constrained case, $n_{\text{all}} = 1500$ is not high enough to reliably find a result). Figure 5.14 shows the histogram that still confirms a large advantage of the optimization procedure, but since so many cases fail, the scores will generally be a bit lower since there is a high probability for a draw and thus the normalization will decrease the scores.

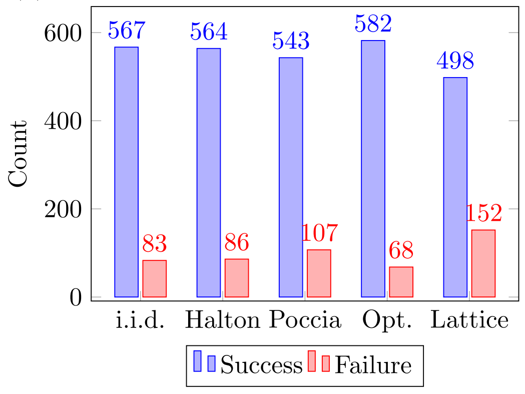
Experiment 8 (Kinematic chain, random maps)**.** Table 5.4 shows the results. Looking solely at the scores the optimized approach again gives the best results in both the identified and non-identified case. On the other hand the success rate shows another trend. The optimized sequence fails in many cases as soon as the identification is removed. The reason for this was investigated via additional experiments and by looking closer at the generated samples. Due to the modified dispersion, samples only slowly approach the border of the space. In high dimensions there remains

	i.i.d.	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.	Lattice
i.i.d.	·	-4.35	-4.20	-6.47	13.06
Halton	4.35	·	0.20	-1.37	14.51
Poccia	4.20	-0.20	·	-1.22	15.89
$\tilde{d}_{\mathrm{rs}}$ opt.	6.47	1.37	1.22	·	16.00
Lattice	-13.06	-14.51	-15.89	-16.00	·

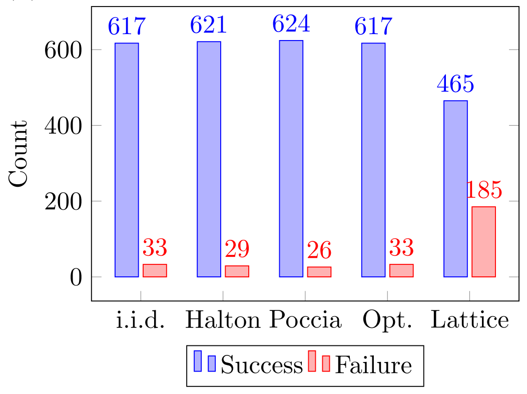
(a) Performance score, $\eta = 0.1, n_{\mathrm{all}} = 5000$

	i.i.d.	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.	Lattice
i.i.d.	·	-5.61	-7.26	-8.12	18.79
Halton	5.61	·	-1.10	-2.35	20.83
Poccia	7.26	1.10	·	-1.06	21.49
$\tilde{d}_{\mathrm{rs}}$ opt.	8.12	2.35	1.06	·	21.22
Lattice	-18.79	-20.83	-21.49	-21.22	·

(b) Performance score, $\eta = 0.05, n_{\mathrm{all}} = 3200$



(c) Success rate, $\eta = 0.1, n_{\mathrm{all}} = 5000$



(d) Success rate, $\eta = 0.05, n_{\mathrm{all}} = 3200$

Table 5.2: Performance scores and success rate histograms for 50 random queries on each of the city maps from Figure 5.2 for different η values. The smaller η the less constrained the movement of the car. A positive value indicates that the row performed better than the column. See text for description.

quite a large uncovered area around the border of the space [1]. From a completeness perspective this is fine, but for the kinematic chain another factor comes into play: samples close the border also represent the fully contracted arm, which represents a very important state to solve many queries as such a configuration is required for solving many narrow corridor problems for the kinematic chain (i.e., the arm covers the smallest area). Hence, the usefulness of the samples depends heavily on the position in the configuration space due to the geometry of the problem. A simple Euclidean distance function in the joint space does not capture this geometry and thus the optimization is not as successful as in the Reeds-Shepp case, in which all positions contribute equally without any assumptions on the queries to solve.

The key takeaway from this experiment is that the choice of distance function and topology of a problem can heavily influence the effectiveness of the optimization approach. The distance function should somehow capture how helpful a sample is,

[1]In this example, when optimizing $n = 30000$ samples for the non-identified space $[-3, 3]^6$ all resulting samples will be fully contained in $[-2, 2]^6$.

	i.i.d.	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.
i.i.d.	·	-1.08	-1.74	-4.87
Halton	1.08	·	-1.08	-4.11
Poccia	1.74	1.08	·	-3.10
$\tilde{d}_{\mathrm{rs}}$ opt.	4.87	4.11	3.10	·

(a) $\eta = 1.0$

	i.i.d.	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.
i.i.d.	·	-2.09	-6.17	-6.26
Halton	2.09	·	-4.21	-4.08
Poccia	6.17	4.21	·	-1.20
$\tilde{d}_{\mathrm{rs}}$ opt.	6.26	4.08	1.20	·

(b) $\eta = 0.25$

	i.i.d.	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.
i.i.d.	·	-4.71	-7.62	-10.69
Halton	4.71	·	-3.16	-6.70
Poccia	7.62	3.16228	·	-2.47
$\tilde{d}_{\mathrm{rs}}$ opt.	10.69	6.70	2.47	·

(c) $\eta = 0.1$

	i.i.d.	Halton	Poccia	$\tilde{d}_{\mathrm{rs}}$ opt.
i.i.d.	·	-3.48	-6.36	-11.61
Halton	3.48	·	-1.83	-7.15
Poccia	6.36	1.83	·	-4.59
$\tilde{d}_{\mathrm{rs}}$ opt.	11.61	7.15	4.59	·

(d) $\eta = 0.05$

Table 5.3: Performance score for 1000 runs on random maps for the Reeds-Shepp case and $n_{\text{all}} = 1500$ for all cases. A positive value indicates that the row performed better than the column. See text for further description.

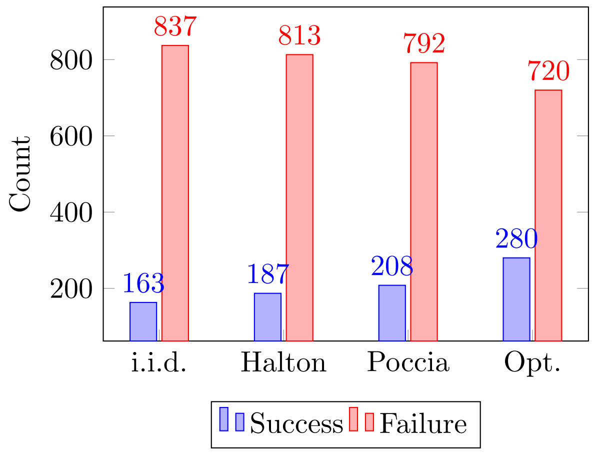


Figure 5.14: Success histogram for $\eta = 1.0$. This shows a generally low success rate for this case, but also shows the clear advantage when solving difficult queries with specialized sets. The success rate is lowest for i.i.d., then increases for Halton (general low-dispersion), increases more for Poccia (Reeds-Shepp case) and is best for our optimized sequence.

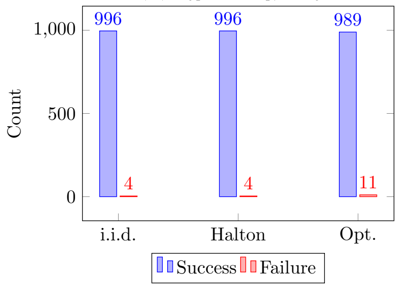
given everything that is known about the environments and queries that should be solved.

	i.i.d.	Halton	$\tilde{d}_{\text{rs}}$ opt.
i.i.d.	·	-2.69	-2.28
Halton	2.69	·	-1.01
$\tilde{d}_{\text{rs}}$ opt.	2.28	1.01	·

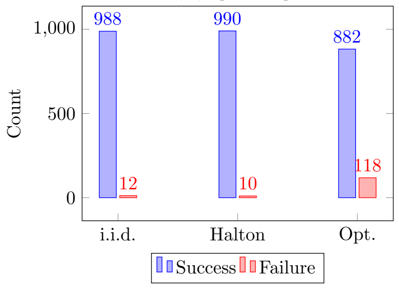
(a) $\{[-\pi,\pi]/\sim\}^6$

	i.i.d.	Halton	$\tilde{d}_{\text{rs}}$ opt.
i.i.d.	·	-1.14	-7.78
Halton	1.14	·	-7.78
$\tilde{d}_{\text{rs}}$ opt.	7.78	7.78	·

(b) $[-3,3]^6$



(c) $\{[-\pi,\pi]/\sim\}^6$



(d) $[-3,3]^6$

Table 5.4: Performance score and success rate histograms for 1000 runs on random maps for the 6D kinematic chain after $n_{\text{all}} = 30000$ samples. A positive value indicates that the row performed better than the column. While the score shows a similar trend in both cases, the success rate of the optimized sequence is worse, especially in the non-identified case. See text for interpretation of this observation.

5.2.2 Primitive generation

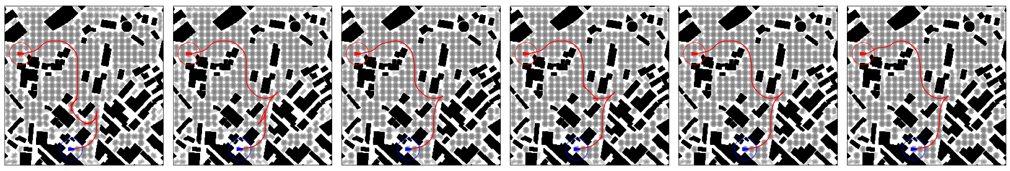
To assess the efficacy of the primitive generation approach that we propose in Section 4.3.2 we want to do a comparison between different primitive sets. The idea here is to mostly show the general efficacy. More experiments are required to compare to other generation methods like the ones summarized in Section 3.2. Again we first show some qualitative results and then results for randomized queries. We are comparing the six different sets, which are shown in Figure 5.10. A major difficulty in comparing these primitive sets, is the large difference in computational efficiency. Since the actual computation time depends heavily on the ratio of vertices to edges, total number of vertices, total number of edges as well as the difficulty of the query, it is difficult to compare motion primitive sets in a fair way. Here we decide to use the same underlying grid for all primitive sets, which generally means, that larger sets of primitives are expected to outperform smaller sets. On the other hand, if a smaller set outperforms a larger set, despite it being smaller, we can conclude that this smaller set should be better than the larger one.

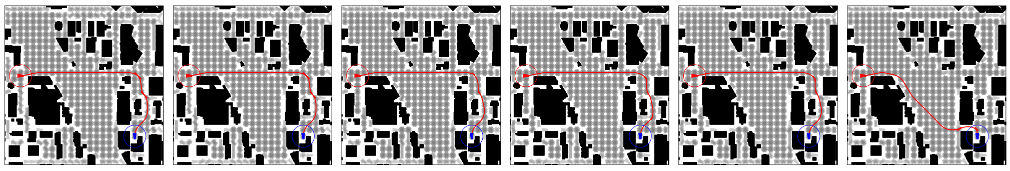
Qualitative

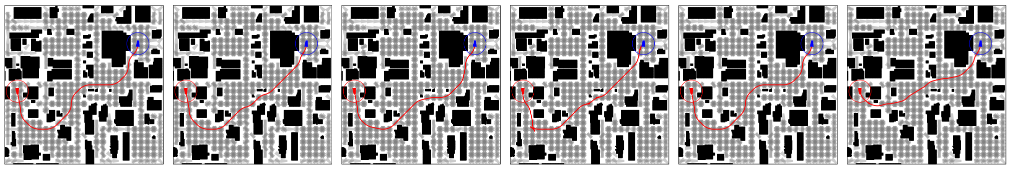
Experiment 9. First, we start with the scores on the city maps. Table 5.5 shows the results for the six different motion primitive sets. As expected the cost of the fixed radius set is always the lowest, but that is expected, since it is a superset of all the other primitive sets, so it will at least reach the same or better cost than other sets. In general the trend that more primitives gives better results is confirmed, but the optimization approach does not seem to give results. While solving more queries than Naive 1 and Naive 2, the scores for the solved cases is often worse. Finally, it can be seen that the combined set of Naive 1 and Optimized yields quite similar results to Naive 3. Figure 5.15 shows the resulting paths for four different maps. Only the the radius set clearly finds smaller corridors in comparison to the other approaches, which mostly differ in unnecessary cusps added along the path.

	Naive 1	Naive 2	Naive 3	Optimized	Combined	Radius
`Berlin_1`	87.14	77.84	**63.12**	75.50	66.57	62.68
`Boston_1`	∞	∞	∞	∞	∞	∞
`Boston_2`	∞	**80.94**	83.16	88.23	82.06	55.83
`Denver_0`	∞	∞	∞	∞	∞	118.70
`Denver_1`	∞	∞	106.73	112.55	108.88	85.54
`Denver_2`	∞	∞	∞	∞	∞	∞
`London_2`	∞	∞	**54.35**	∞	∞	53.23
`Moscow_0`	71.41	**70.67**	70.99	74.51	70.99	69.92
`NewYork_2`	72.44	**64.71**	72.23	66.21	55.83	49.65
`Paris_1`	76.59	76.59	**76.18**	102.74	76.38	69.16
`Shanghai_1`	80.94	80.94	**79.17**	84.25	69.01	65.99
`Shanghai_0`	70.04	70.04	**63.35**	67.45	65.03	52.09
`Sydney_2`	62.74	62.03	**61.28**	73.17	62.74	51.79

Table 5.5: Resulting solution cost on the city maps for the state lattice planner with six different motion primitive sets. The best result is highlighted in bold (ignoring the radius case, since it outperforms the other sets by construction). See text for description.


(a) Naive 1 (b) Naive 2 (c) Naive 3 (d) Opt. (e) Comb. (f) Radius


(g) Naive 1 (h) Naive 2 (i) Naive 3 (j) Opt. (k) Comb. (l) Radius


(m) Naive 1 (n) Naive 2 (o) Naive 3 (p) Opt. (q) Comb. (r) Radius

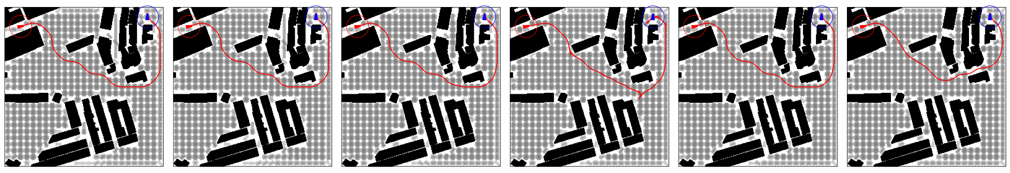

(s) Naive 1 (t) Naive 2 (u) Naive 3 (v) Opt. (w) Comb. (x) Radius

Figure 5.15: Resulting paths for the state lattice planning with different motion primitive sets on the maps `Berlin_1`, `Denver_1`, `Denver_2` and `Sydney_2`. It can be seen that the most expansive primitive set Radius is also the only one that finds the narrow corridors for the last three maps. The other paths mostly differ in slightly different turns and additional cusps.

Quantitative

Experiment 10. As before we now want to confirm the trend seen from the example maps in more extensive, randomized experiments. Table 5.6 shows the performance score for the motion primitive sets on randomized maps. It can be seen that in general the optimized set performs worse than every other set. Only by combining it with the simple set we can achieve competitive scores, with Naive 3 still outperforming it despite being the smaller set. Hence, clearly the optimization approach does not help in practice and naive approaches, even without proven completeness, perform well. The takeaway from this is that the proposed optimization that aims at proven completeness with few primitives does not provide benefits on the achievable solution cost yet. An open question is whether this is due to insufficient optimization procedure (see Chapter 6) or whether proven completeness simply does not correlate with achieving low cost in the average case. It could also be that the naive sets actually achieve the same or better completeness without it being explicitly shown.

	Naive 1	Naive 2	Naive 3	Optimized	Combined	Radius
Naive 1	·	-9.14	-26.66	20.27	-21.35	-30.80
Naive 2	9.14	·	-16.16	24.03	-11.51	-30.67
Naive 3	26.66	16.16	·	27.51	8.16	-30.48
Optimized	-20.27	-24.03	-27.51	·	-28.14	-30.86
Combined	21.35	11.51	-8.16	28.14	·	-30.64
Radius	30.80	30.67	30.48	30.86	30.64	·

Table 5.6: Performance score for 1000 runs on random maps for the Reeds-Shepp case and the different motion primitive sets shown in Figure 5.10. A positive value indicates that the row performed better than the column. See text for description.

5.2.3 From probabilistic roadmaps to state lattice planning

Finally, we want to discuss the connection between these different approaches to deterministic motion planning as they are all highly related. The goal of this section is to draw the connection between the different methods and put them into perspective.

State lattice planning, while looking quite different from probabilistic roadmaps at first sight, is actually very similar. If the motion primitives are designed in such a way, that they can be continuously connected (i.e., no approximation) in a graph-based framework (either implicitly defined by an online A* search or explicitly by precomputing the graph), the primitives will imply an orthogonal grid. For such an orthogonal grid we can compute the dispersion (i.e., with our proposed algorithm) and because the start and end have to be connected to this graph, the dispersion of the grid defines the completeness that can be achieved by a state lattice planner. Note that since we are constrained to orthogonal grids, this dispersion will in general be higher than if we optimize the dispersion taking into account the distance metric of the system.

The motion primitives finally define the connection between the vertices and is thus highly related to the connection strategy. By precomputing the motion primitives with a connection radius defined by two times the dispersion of the grid, we directly achieve resolution completeness for state lattice planners. Alternatively we can use our alternative proof of completeness for motion primitives as it allows to precompute a sparser set of motion primitives while maintaining completeness. Note that the potential of finding such a sparse set of primitives that gives resolution completeness is only enabled through the regular, orthogonal positioning of the vertices, as otherwise primitives could not be precomputed.

Comparing this with PRM using our optimized set of samples, we see that we require less vertices, but can not use the same sparse connection strategy. Note that in principle precomputing edge information would also be viable for PRM with optimized samples, although it would have much higher memory requirements, since each edge is unique in comparison to the state lattice, in which only the motion primitives have to be stored. Another advantage of state lattice planning to the optimization approach is its independency from the environment size, since the motion primitives can simply be extended until the whole space is covered.

To summarize, state lattice planning can be seen as another roadmap planner, that uses the invariance existing in many practical systems (i.e., translational invariance) to precompute connection information (i.e., motion primitives). As we show, this can be used to precompute more sparse connections, while preserving the same completeness. The cost to pay is being constrained to orthogonal grids, which means a higher number of vertices is required to achieve the same resolution completeness.

6 Conclusion and Outlook

In this work we showed the viability of using deterministic sampling-based motion planning to derandomize planners and get non-probabilistic completeness guarantees. Additionally we have introduced an approach to precompute optimized samples based on a distance function, which allows to skip unnecessary collision checks during planning and can thus help to speed up planners. We showed in extensive experiments that the optimized samples outperform all baselines in both cost and success rate, which confirms the theoretic benefits in practice. The optimization approach is general only requiring an optimal steering function and is thus applicable for many practical systems like differential drive and car-like kinematics.

In general it can be said, that there are only few reasons to use i.i.d. samples nowadays. In this work we only worked with uniformly distributed samples, as such are important for getting completeness guarantees. Generally using for example Halton samples in combination with learned distributions to generate deterministic, nonuniform samples seems like a good start to remove randomness from such approaches as well. Also to keep the completeness guarantees, safety critical systems might deploy two parallel motion planners (or use interleaved samples), that on one hand use nonuniform samples to find a solution quickly in most cases, and a fallback system based on deterministic uniform sampling that provides the required completeness guarantees.

The greedy optimization approach we propose gives clear benefits for low-dimensional systems, but is quite simple in nature and more complicated approaches would definitely be thinkable. Examples include optimizing the sample position for a horizon of samples, a graph representation-based approach instead of a grid, to allow better handling of higher dimension, and investigating the possibility of parallelizing the approach, since right now, it is a sequential algorithm by nature.

We also showed that a reachable set-based framework can be used to prove resolution completeness for symmetric and optimal systems, which includes many practical systems. We used the same framework to prove completeness both for PRM and state lattice planners, which also clearly shows the relationship between the two. For state lattice planners this insight can be used to define a sparser condition on the primitives to reach resolution completeness. While the proposed algorithm, does not perform well in practice (i.e., higher cost and failure), it requires less primitives to give the same resolution completeness as a dispersion-based connection radius. Obviously having proven resolution completeness, must not coincide with achieving low cost and high success rate. An interesting question is, whether better performing naive approaches actually give resolution completeness as we defined it and how to adapt the optimization technique to better capture the characteristics of the state lattice paradigm (i.e., concatenation of primitives). Therefore, further research is required in this area. Still, our analysis clearly shows the close relationship between roadmap and state lattice approaches, showing that the two are practically equivalent in many situations.

Outlook

Multiple extensions to this work are possible. First, for the optimization approach many extensions pose both practical as well as theoretical interesting challenges:

Non-symmetric systems We believe that most of the analysis can also be carried out by combining positive time (i.e., sets of points reachable from a given point within some time or distance) and negative time reachable sets (i.e., sets of points, which can reach a given point in some time or distance). This should allow to extend the optimization to non-symmetric systems as well.

Drift systems Drift systems have the interesting property that time-limited reachable sets might not contain themselves anymore. The question is how to define the reachable sets, steering functions and optimization objective in this case.

Environment agnostic optimization Our approach works by optimizing the samples for a given environment. In practice the size of the environment might not be known beforehand. Hence, optimizing samples for arbitrary environments without sacrificing too much dispersion and efficiency is another interesting problem.

Non-uniform sampling Integrating priors and increasing the density of samples in difficult parts of the state space is a very common approach towards efficient sampling-based motion planning. In light of our approach, the desired property would be to efficiently decrease dispersion in difficult parts of the state space. This seems quite related to the previous question of environment agnostic sampling optimization, which could be seen as the first step in this direction.

Second, for state lattice planners, the results showed that our optimization approach is not yet sufficient to outperform naive approaches in practice. We suspect the reason for this to be that we are optimizing for each primitive origin a separate set of primitives. This ignores the fact that the primitives must work well together to find paths and thus a joint optimization of all primitives should give better results. Additionally the fact that new primitives are built by concatenation should also be incorporated into the optimization.

Bibliography

[1] S. M. LaValle, *Planning algorithms.* Cambridge University Press, 2006.

[2] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[3] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual.* Pearson Education, 2001.

[4] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] C.-T. Chen, *Linear system theory and design.* Oxford University Press, Inc., 1998.

[7] B. Donald, P. Xavier, J. Canny, and J. Reif, “Kinodynamic motion planning,” *Journal of the ACM*, vol. 40, no. 5, pp. 1048–1066, 1993.

[8] N. Ferrers, “Extension of Lagrange’s equations,” *The Quarterly Journal of Pure and Applied Mathematics*, vol. 12, no. 45, pp. 1–5, 1872.

[9] J.-P. Laumond, “Feasible trajectories for mobile robots with kinematic and environment constraints,” in *Proceedings of the International Conference on Intelligent Autonomous Systems*, pp. 346–354, 1986.

[10] H. J. Sussmann and G. Tang, “Shortest paths for the Reeds-Shepp car: A worked out example of the use of geometric techniques in nonlinear optimal control,” tech. rep., Rutgers Center for Systems and Control, 1991.

[11] L. 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*, vol. 79, no. 3, pp. 497–516, 1957.

[12] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[13] P. Soueres and J.-D. Boissonnat, "Optimal trajectories for nonholonomic mobile robots," in *Robot Motion Planning and Control*, pp. 93–170, Springer, 1998.

[14] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics," in *Proceedings of the IEEE Conference on Decision and Control*, pp. 2574–2581, 2015.

[15] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the driftless case," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2368–2375, 2015.

[16] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] L. Palmieri, S. Koenig, and K. O. Arras, "RRT-based nonholonomic motion planning using any-angle path biasing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2775–2781, 2016.

[18] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, 2014.

[19] S. Thomas, M. Morales, X. Tang, and N. M. Amato, "Biasing samplers to improve motion planning performance," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1625–1630, 2007.

[20] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 46–61, 2018.

[21] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[22] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 521–528, 2000.

[23] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[24] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[25] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 995–1001, 2000.

[26] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3067–3074, IEEE, 2015.

[27] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, vol. 63. SIAM, 1992.

[28] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerische Mathematik*, vol. 2, no. 1, pp. 84–90, 1960.

[29] A. G. Sukharev, "Optimal strategies of the search for an extremum," *USSR Computational Mathematics and Mathematical Physics*, vol. 11, no. 4, pp. 119–137, 1971.

[30] J. van der Corput, "Verteilungsfunktionen i & ii," in *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen*, vol. 38, pp. 1058–1066, 1935.

[31] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Communications ACM*, vol. 7, no. 12, pp. 701–702, 1964.

[32] M. Berblinger and C. Schlier, "Monte Carlo integration with quasi-random numbers: Some experience," *Computer Physics Communications*, vol. 66, no. 2-3, pp. 157–166, 1991.

[33] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 7087–7094, 2018.

[34] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2407–2414, 2019.

[35] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 673–692, 2004.

[36] E. Poccia, "Deterministic sampling-based algorithms for motion planning under differential constraints," Master's thesis, Stanford University, 2017.

[37] R. Montgomery, *A tour of subriemannian geometries, their geodesics and applications.* American Mathematical Soc., 2002.

[38] A. Bellaïche, "The tangent space in sub-Riemannian geometry," in *Sub-Riemannian Geometry*, pp. 1–78, Springer, 1996.

[39] R. S. Strichartz, "Sub-riemannian geometry," *Journal of Differential Geometry*, vol. 24, no. 2, pp. 221–263, 1986.

[40] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 5041–5047, 2013.

[41] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang, "Quasi-randomized path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1481–1487, 2001.

[42] S. R. Lindemann, A. Yershova, and S. M. LaValle, "Incremental grid sampling strategies in robotics," in *Algorithmic Foundations of Robotics VI*, pp. 313–328, Springer, 2004.

[43] A. Yershova and S. M. LaValle, "Deterministic sampling methods for spheres and SO(3)," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3974–3980, 2004.

[44] A. Yershova, S. Jain, S. M. Lavalle, and J. C. Mitchell, "Generating uniform incremental grids on SO(3) using the hopf fibration," *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 801–812, 2010.

[45] S. R. Lindemann and S. M. LaValle, "Incrementally reducing dispersion by increasing Voronoi bias in RRTs," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3251–3257, 2004.

[46] S. R. Lindemann and S. M. LaValle, "Steps toward derandomizing RRTs," in *Robot Motion and Control*, pp. 287–300, Springer, 2006.

[47] W. Khaksar, T. S. Hong, M. Khaksar, and O. Motlagh, "A low dispersion probabilistic roadmaps (LD-PRM) algorithm for fast and efficient sampling-based motion planning," *International Journal of Advanced Robotic Systems*, vol. 10, no. 11, p. 397, 2013.

[48] M. N. Pivtoraiko, *Differentially constrained motion planning with state lattice motion primitives*. PhD thesis, Carnegie Mellon University, 2012.

[49] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, pp. 1–7, 2005.

[50] M. Pivtoraiko and A. Kelly, "Generating state lattice motion primitives for differentially constrained motion planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 101–108, 2012.

[51] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[52] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3231–3237, 2005.

[53] A. Bicchi, A. Marigo, and B. Piccoli, "On the reachability of quantized control systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 4, pp. 546–563, 2002.

[54] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi, "Motion planning through symbols and lattices," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3914–3919, 2004.

[55] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[56] R. A. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables.," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3375–3380, 2006.

[57] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Optimal, smooth, nonholonomic mobile robot motion planning in state lattices," tech. rep., Carnegie Mellon University, 2007.

[58] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, “Motion planning for autonomous driving with a conformal spatiotemporal lattice,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4889–4895, 2011.

[59] P. Cheng, *Sampling-based motion planning with differential constraints.* PhD thesis, University of Illinois at Urbana-Champaign, 2005.

[60] M. Pivtoraiko and A. Kelly, “Differentially constrained motion replanning using state lattices with graduated fidelity,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2611–2616, 2008.

[61] A. González-Sieira, M. Mucientes, and A. Bugarín, “Anytime motion replanning in state lattices for wheeled robots,” in *Proceedings of the Workshop of Physical Agents*, pp. 217–224, 2012.

[62] A. Kushleyev and M. Likhachev, “Time-bounded lattice for efficient planning in dynamic environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1662–1668, 2009.

[63] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, “Receding-horizon lattice-based motion planning with dynamic obstacle avoidance,” in *Proceedings of the IEEE Conference on Decision and Control*, pp. 4467–4474, 2018.

[64] J. Ziegler and C. Stiller, “Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1879–1884, 2009.

[65] M. Cirillo, T. Uras, and S. Koenig, “A lattice-based approach to multi-robot motion planning for non-holonomic vehicles,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 232–239, 2014.

[66] J. Butzke, K. Sapkota, K. Prasad, B. MacAllister, and M. Likhachev, “State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 258–265, 2014.

[67] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, “Sampling-based optimal kinodynamic planning with motion primitives,” *Autonomous Robots*, pp. 1–18, 2018.

[68] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[69] M. Pivtoraiko, T. M. Howard, I. Nesnas, and A. Kelly, "Field experiments in rover navigation via model-based trajectory generation and nonholonomic motion planning in state lattices," in *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, pp. 25–29, 2008.

[70] A. Rusu, S. Moreno, Y. Watanabe, M. Rognant, and M. Devy, "State lattice generation and nonholonomic path planning for a planetary exploration rover," in *Proceedings of the International Astronautical Congress*, 2014.

[71] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-UAV motion replanning for exploring unknown environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2452–2458, 2013.

[72] O. Ljungqvist, N. Evestedt, M. Cirillo, D. Axehill, and O. Holmer, "Lattice-based motion planning for a general 2-trailer system," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 819–824, 2017.

[73] S. Wang, *State lattice-based motion planning for autonomous on-road driving.* PhD thesis, Free University of Berlin, 2015.

[74] M. Cirillo, "From videogames to autonomous trucks: a new algorithm for lattice-based motion planning," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 148–153, 2017.

[75] A. A. Paranjape, K. C. Meier, X. Shi, S.-J. Chung, and S. Hutchinson, "Motion primitives and 3d path planning for fast flight through a forest," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 357–377, 2015.

[76] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2172–2179, 2011.

[77] C. Green and A. Kelly, "Toward optimal sampling in the space of paths," in *Proceedings of the International Symposium on Robotics Research*, vol. 3, p. 1, 2007.

[78] A. Botros and S. L. Smith, "Computing a minimal set of t-spanning motion primitives for lattice planners," *arXiv preprint arXiv:1903.10483*, 2019.

[79] A. Sivaramakrishnan, Z. Littlefield, and K. E. Bekris, "Towards learning efficient maneuver sets for kinodynamic motion planning," *arXiv preprint arXiv:1907.07876*, 2019.

[80] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a Dubins airplane," in *Proceedings of the IEEE Conference on Decision and Control*, pp. 2379–2384, 2007.

[81] D. J. Balkcom and M. T. Mason, "Time optimal trajectories for bounded velocity differential drive vehicles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 199–217, 2002.

[82] H. Chitsaz, S. M. LaValle, D. J. Balkcom, and M. T. Mason, "Minimum wheel-rotation paths for differential-drive mobile robots," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 66–80, 2009.

[83] D. J. Balkcom, P. A. Kavathekar, and M. T. Mason, "Time-optimal trajectories for an omni-directional vehicle," *The International Journal of Robotics Research*, vol. 25, no. 10, pp. 985–999, 2006.

[84] A. A. Furtuna, D. J. Balkcom, H. Chitsaz, and P. Kavathekar, "Generalizing the Dubins and Reeds-Shepp cars: fastest paths for bounded-velocity mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2533–2539, 2008.

[85] C. Fernandes, L. Gurvits, and Z. Li, "A variational approach to optimal non-holonomic motion planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 680–685, 1991.

[86] D. A. Anisi, J. Hamberg, and X. Hu, "Nearly time-optimal paths for a ground vehicle," *International Journal of Control Theory and Applications*, vol. 1, no. 1, pp. 2–8, 2003.

[87] W.-L. Chow, "Über Systeme von linearen partiellen Differential-gleichungen erster Ordnung," in *The Collected Papers Of Wei-Liang Chow*, pp. 47–54, World Scientific, 2002.

[88] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, pp. 85–103, Springer, 1972.

[89] N. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012.