Deterministic Sampling-Based Algorithms for Motion Planning under Differential Constraints

Ernesto Poccia

October 10, 2017

Contents

List of Figures

List of Tables

Abstract

Probabilistic sampling-based algorithms, such as the probabilistic roadmap (PRM) and the rapidly-exploring random tree (RRT) algorithms, represent one of the most successful approaches to robotic motion planning, due to their strong theoretical properties (in terms of probabilistic completeness or even asymptotic optimality) and remarkable practical performance. Such algorithms are probabilistic in that they compute a path by connecting independently and identically distributed (i.i.d.) random points in the configuration space. Their randomization aspect, however, makes several tasks challenging, including certification for safety-critical applications and use of offline computation to improve real-time execution. Hence, an important open question is whether similar (or better) theoretical guarantees and practical performance could be obtained by considering deterministic, as opposed to random sampling sequences. It is natural to wonder whether the theoretical guarantees and practical performance of sampling-based algorithms would hold if these algorithms were to be de-randomized, i.e., run on a deterministic, as opposed to random sampling sequence. This is an important question, as de-randomized planners would significantly simplify the certification process (as needed for safety-critical applications), enable the use of offline computation (particularly important for planning under differential constraints or in high-dimensional spaces(exactly the regime for which sampling-based planners are designed), and, in the case of lattice sequences, drastically simplify a number of operations (e.g., locating nearby samples).

Results in this sense, have been recently achieved by [2] for purely geometric motion planning problems by employing low L_2-dispersion sampling sequences. However, this approach has proven to be limited for motion planning under differential constraints, probably because Euclidean distance is no longer an adequate measure of how difficult is to join two given points in the configuration space.

On the basis of these considerations, an appealing venue for extending the deterministic sampling approach to kino-dynamic motion planning appears to be the development of system-specific sampling strategies designed *ad hoc* to cover the configuration space uniformly, in terms of the control effort required to steer the systems from a given point to its nearest sample.

The author aims to develop and theoretically characterize such sampling strategies for two particular classes of dynamics system: *systems with linear affine dynamics* and *driftless control affine (DCA) systems* as well as outlining possible heuristics to extend these methodologies to more general examples of non-linear systems.

Preface

This dissertation is structured in two parts. The first part, simply titled "Motion Planning" can be considered as a brief introduction to the general subject and introduce the main concepts an terminology that will be given for granted in the second part. This exposition essentially summarizes the contents presented more extensively in [1], with a particular focus on the the so called Sampling-Base Planning Algorithms, which are the the class of algorithm of greatest interest for this work. This introductory material serves a twofolds purpose: On the one hand it makes the whole dissertation conceptually self-contained; on the other hand it provides a thematic context for the analysis presented in the second parts, that the author believes to be worth including, going the topics of this work partially beyond the University's program.

The second part, as opposed to the first, presents many original findings, being in fact a collection of the theoretical and experimental results obtained by the author during the period spent at the Autonomous System Lab of Stanford University.

The author is keen to express his gratitude towards the researchers of he ASL for their availability and technical support, with a particular reference to Prof. Marco Pavone and Dr. Edward Schmerling, and to the Italian Space Agency (ASI) for his generous financial sponsorship, in absence of which this project would have hardly taken off.

Ernesto Poccia

Part I

Motion Planning

Chapter 1

Introduction

In robotics, *motion planning* is the term used to describe the problem of determining a sequence of actions in order to steer an *autonomous system*[1] from a given initial configuration to a desired goal configuration while avoiding collision with obstacles. There are many versions of this very general problem, accordingly to the discrete or continuous nature of system's configuration space, to the presence of kino-dynamical constraints on its motion, or whether the planning task is performed trying to optimize a particular criterium like energy or time consumption.

However, all these different problems can be placed in the same conceptual frame characterized by few basic elements:

State The state of a system can be imagined as a set of parameters that completely and uniquely qualify the system at a certain time, according to the mathematical model used to describe the system itself. The *state space* is the set of all the possible state of the system or to put it another way, the state space captures all possible situations that could arise.

Time All planning problem involve a sequence of decisions that must be applied over time. Time can be explicitly modeled as in the problem of driving a vehicle through an obstacle course or, implicitly by simply reflecting the chronological sequence of actions applied to the system.

Actions also referred as *inputs* or *controls*, determine the passage of the system from the current state to the next one. Their effect is described either by a *state transition function*, as in the case of discrete motion planning, or by ordinary differential equations as in continuous-time systems.

Initial and goal states A planning problems usually involves starting in some initial state and trying to arrive at a specified goal state or any state in a set of goal states. The actions are selected in a way that tries to make this happen.

A plan a strategy adopted to reach the goal state. A plan can simply consist of sequence of actions to be taken, however if it is impossible to predict future states, a plane can specify as a function of the state, in this case it is customary to talk of *feedback plans*.

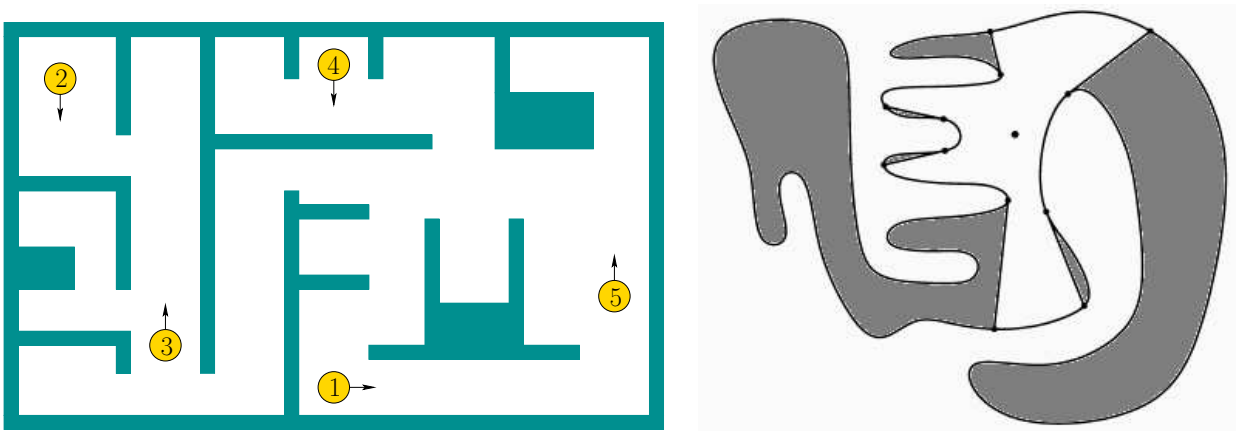
[1] By the name of autonomous system wee will refer to any robot that performs behaviors or task with a high degree of autonomy. Examples of these systems are given by spacecraft, drones and industrials robots. In the following we will designate any of those systems simply as *robots*

Finally depending of the desired outcome of the plan we make a distinction between:

Feasible motion planning problem: Find a plan that arrives at a goal state.

Optimal motion planning problem: Find a feasible plan that optimizes performances in some specified manner, in addition to arriving in a goal state.

As can be easily understood, achieving optimality is even more challenging than mere feasibility.



(a) (b)

Figure 1.1: Several mobile robots attempt to successfully navigate in an indoor environment while avoiding collisions with the walls and each other. (b) Imagine using a lantern to search a cave for missing people.

1.1 Examples of application

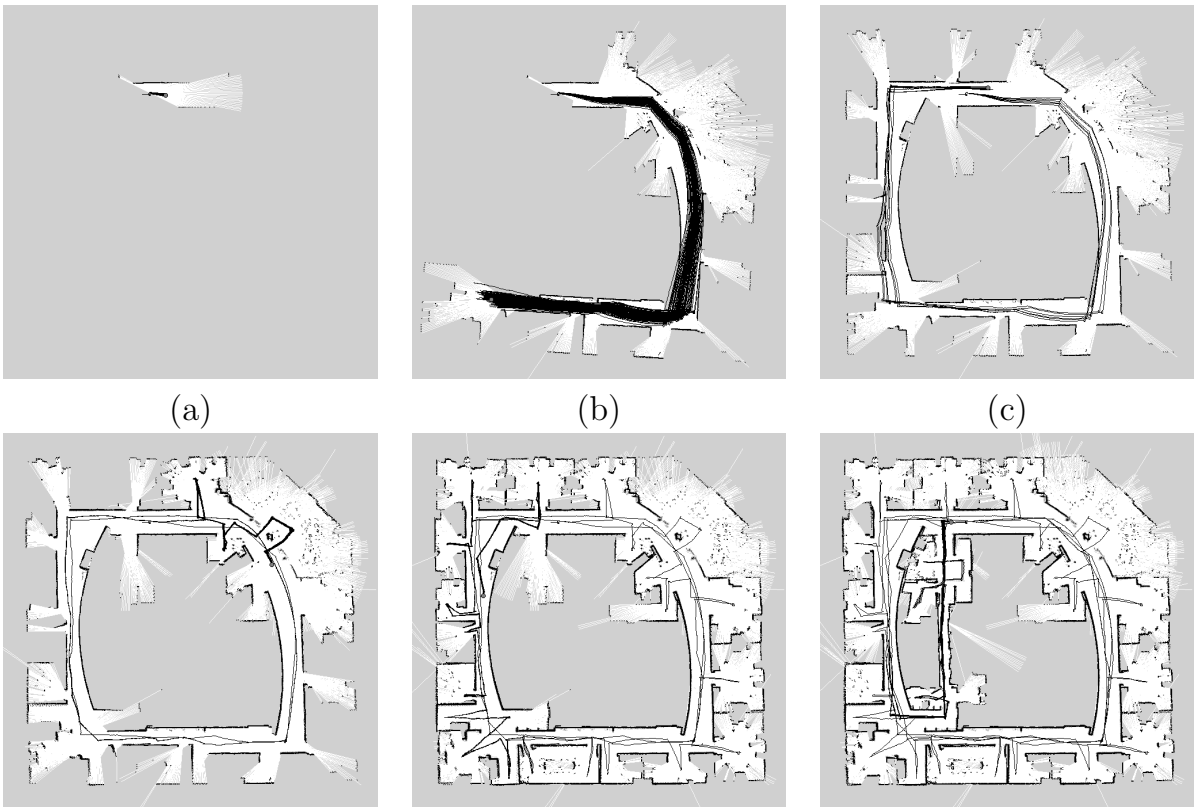
In order to fully appreciate the importance and the ubiquity of the motion planning problem, as formulated previously, in this section are provided a certain number of examples from the most different fields of application.

1.1.1 Navigate mobile robots

A common task for mobile robots is to request them to navigate in an indoor environment, as shown in Figure 1.1a A robot might be asked to perform tasks such as building a map of the environment, determining its precise location within a map, or arriving at a particular place. Most robots operate in spite of large uncertainties. At one extreme, it may appear that having many sensors is beneficial because it could allow precise estimation of the environment and the robot position and orientation. This is the premise of many existing systems, as shown for the robot system in Figure 1.2, which constructs a map of its environment. It may alternatively be preferable to develop low-cost and reliable robots that achieve specific tasks with little or no sensing.

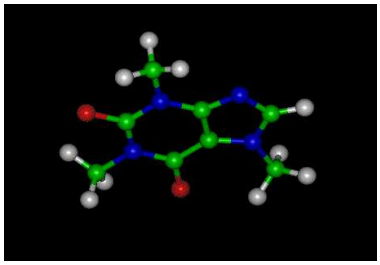
1.1.2 Flighting through the Air or in Space

Planning algorithms can help to navigate autonomous helicopters through obstacles. They can also compute thrusts for a spacecraft so that collisions are avoided around a complicated structure, such as a space station. Mission planning for interplanetary spacecraft, including solar sails, can even be performed using planning algorithms.

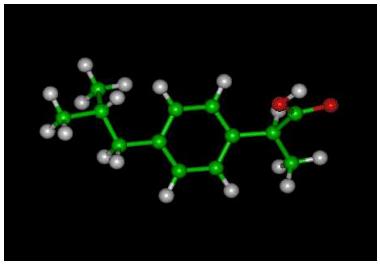


(a) (b) (c)

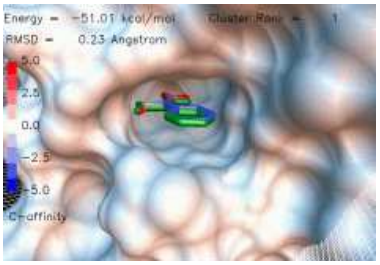
(d) (e) (f)



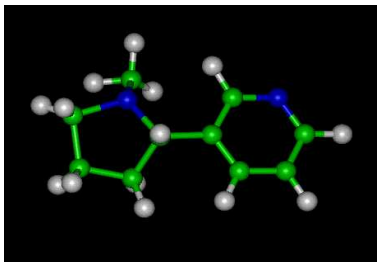
Caffeine



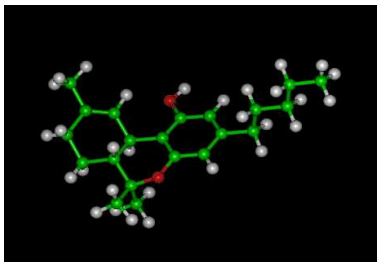
Ibuprofen



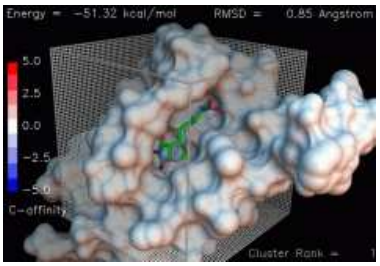
AutoDock



Nicotine



THC



AutoDock

1.1.3 Designing better drugs

Planning algorithms are even impacting fields as far away from robotics as computational biology. Two major problems are protein folding and drug design. In both cases, scientists attempt to explain behaviors in organisms by the way large organic molecules interact. Such molecules are generally flexible. Drug molecules are small (see Figure 1.3), and proteins usually have thousands of atoms. The *docking problem* involves determining whether a flexible molecule can insert itself into a protein cavity while satisfying other constraints, such as maintaining low energy. algorithms.

1.2 Organization of the Thesis

This thesis is mainly concerned with the study of algorithms for *motion planning under differential constraint*, however we will arrive to this problem gradually, beginning our analysis from the simpler, purely geometry case in which issues regarding the dynamics of the system are neglected. There are two main reasons why the author believe this approach is preferable with respect to dealing directly with the differentially constrained case. First of all, the basic problem whereby a robot does not have any constraint on its motion is well understood and solved for a large number of practical scenarios, consequently this material happens to be more suited to introduce the main aspects of a planning problem. Secondly, many of the concepts and tools developed in this context have their counterparts in the kino-dynamic formulation.

One of the first problems one has to deal with is the transition from discrete to continuous state spaces. The conceptual roadmap followed in this dissertations start from the analysis of planning algorithm for discrete systems, that can be handled as classical graph research problem, followed by the introduction of the configuration space for continuous

systems like rigid bodies and chains of rigid bodies, and finally by a possible strategy to reduce the continuous problem to a discrete one by sampling the configuration space.

Chapter 2

Discrete Planning

The planning problem considered here is the simplest to describe because the state space will be *finite*, or at most *countably infinite*. Therefore, no geometric models or differential equations will be needed to characterize the discrete planning problems. However, despite of its relative simplicity, this chapter allows us to introduce all the central concepts common to all the planning problems, regardlessly of their level of complexity. Moreover, the techniques used in the context of discrete planning represent a fundamental ingredient of the approach adopted by sampling-based motion planning algorithms, i.e. a very important class of algorithm for planning in continuous spaces, which are the main focus of this work. As mentioned before, there are two different problems to be addressed: feasibility and optimality.

2.1 Discrete Feasible Planning

The discrete feasible planning model will be defined using state-space models.The basic idea is that each distinct situation for the world is called a state, denoted by x, and the set of all possible states is called a state space, X. The world may be transformed through the application of actions that are chosen by the planner. Each action, u, when applied from the current state, x, produces a new state, x', as specified by a *state transition function*, f. It is convenient to use f to express a *state transition equation*,

$$x' = f(x, u). \tag{2.1}$$

Let $U(x)$ denote the *action space* for each state x, which represents the set of all actions that could be applied from x. For distinct $x, x' \in X$, $U(x)$ and $U(x')$ are not necessarily disjoint; the same action may be applicable in multiple states.

Therefore, it is convenient to define the set U of all possible actions over all states:

$$U = \bigcup_{x \in X} U(x). \tag{2.2}$$

As part of the planning problem, a set $X_G \subset X$ of goal states is defined. The task of a planning algorithm is to find a finite sequence of actions that when applied, transforms the initial state x_I to some state in X_G. The model is summarized as:

Formulation 1. ***(Discrete Feasible Planning)***

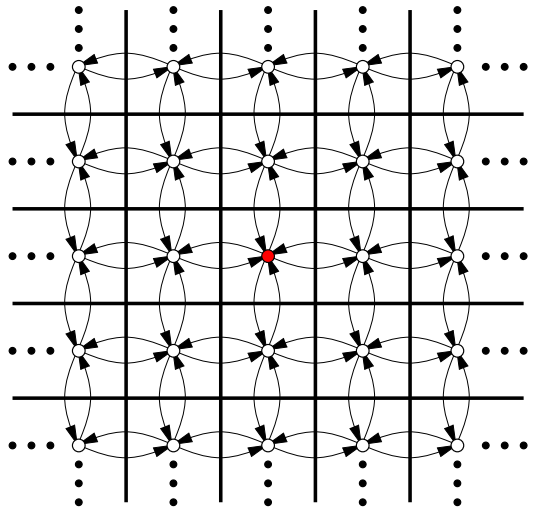


Figure 2.1: The state transition graph for an example problem that involves walk- ing around on an infinite tile floor.

1. *A nonempty state space X, which is a finite or countably infinite set of states.*
2. *For each state $x \in X$, a finite action space $U(x)$.*
3. *A state transition function f that produces a state $f(x, u) \in X$ for every $x \in X$ and $u \in U(x)$. The state transition equation is derived from f as $x' = f(x, u)$.*
4. *An initial state $x_I \in X$.*
5. *A goal set $X_G \subset X$.*

It is often convenient to express Formulation 1 as a directed *state transition graph*. The set of vertices is the state space X. A directed edge from $x \in X$ to $x' \in X$ exists in the graph if and only if there exists an action $u \in U(x)$ such that $x' = f(x, u)$. The initial state and goal set are designated as special vertices in the graph, which completes the representation of Formulation 1 in graph form.

2.2 Searching for Feasible Plans

The methods presented in this section are just graph search algorithms, but with the understanding that the state transition graph is revealed incrementally through the application of actions, instead of being fully specified in advance. The presentation in this section can therefore be considered as visiting graph search algorithms from a planning perspective. An important requirement for these or any search algorithms is to be *systematic*. If the graph is finite, this means that the algorithm will visit every reachable state, which enables it to correctly declare in finite time whether or not a solution exists. To be systematic, the algorithm should keep track of states already visited; otherwise, the search may run forever by cycling through the same states. Ensuring that no redundant exploration occurs is sufficient to make the search systematic. If the graph is infinite, then we are willing to tolerate

a weaker definition for being systematic. If a solution exists, then the search algorithm still must report it in finite time; however, if a solution does not exist, it is acceptable for the algorithm to search forever. This systematic requirement is achieved by ensuring that, in the limit, as the number of search iterations tends to infinity, every reachable vertex in the graph is explored. Since the number of vertices is assumed to be countable, this must always be possible.

2.2.1 General Forward Search

The following is a general template of search algorithms, expressed using the state-space representation.

Algorithm 1 FORWARD SEARCH

```
1: Q.Insert(X_I) and mark x_I as visited
2: while Q not empty do
3:     x ← Q.GetFirst()
4:     if x ∈ X_G then
5:         return success
6:     for all u ∈ U(x) do
7:         x' ← f(x, u)
8:         if x' not visited then
9:             Q.Insert(x')
10:        else
11:            Resolve duplicate x'
   return failure
```

At any point during the search, there will be three kinds of states:

Unvisited: States that have not been visited yet. Initially, this is every state except x_I.

Dead: States that have been visited, and for which every possible next state has also been visited. A next state of x is a state x' for which there exists a $u \in U(x)$ such that $x' = f(x, u)$. In a sense, these states are dead because there is nothing more that they can contribute to the search; there are no new leads that could help in finding a feasible plan.

Alive: States that have been encountered, but possibly have unvisited next states. These are considered alive. Initially, the only alive state is x_I.

The set of alive states is stored in a priority queue, Q, for which a priority function must be specified. The only significant difference between various search algorithms is the particular function used to sort Q. Many variations will be described later, but for the time being, it might be helpful to pick one. Therefore, assume for now that Q is a common $FIFO$ (First-In First-Out) queue; whichever state has been waiting the longest will be chosen when $Q.GetFirst()$ is called. The rest of the general search algorithm is quite simple. Initially, Q contains the initial state x_I. A **while** loop is then executed, which terminates only when Q is empty. This will only occur when the entire graph has been explored without finding any goal states, which results in a FAILURE (unless the reachable portion of X is infinite, in which case the algorithm should never terminate). In

each while iteration, the highest ranked element, x, of Q is removed. If x lies in X_G, then it reports success and terminates; otherwise, the algorithm tries applying every possible action, $u \in U(x)$. For each next state, $x' = f(x, u)$, it must determine whether x' is being encountered for the first time. If it is unvisited, then it is inserted into Q; otherwise, there is no need to consider it because it must be either dead or already in Q.

One important detail is that the existing algorithm only indicates whether a solution exists, but does not seem to produce a plan, which is a sequence of actions that achieves the goal. This can be fixed by inserting a line after line 7 that associates with x' its parent, x. If this is performed each time, one can simply trace the pointers from the final state to the initial state to recover the plan. For convenience, one might also store which action was taken, in addition to the pointer from $x'$to x.

Lines 8 and 9 are conceptually simple, but how can one tell whether x' has been visited? For some problems the state transition graph might actually be a tree, which means that there are no repeated states. Although this does not occur frequently, it is wonderful when it does because there is no need to check whether states have been visited. If the states in X all lie on a grid, one can simply make a lookup table that can be accessed in constant time to determine whether a state has been visited. In general, however, it might be quite difficult because the state x' must be compared with every other state in Q and with all of the dead states. If the representation of each state is long, as is sometimes the case, this will be very costly. A good hashing scheme or another clever data structure can greatly alleviate this cost, but in many applications the computation time will remain high. One alternative is to simply allow repeated states, but this could lead to an increase in computational cost that far outweighs the benefits. Even if the graph is very small, search algorithms could run in time exponential in the size of the state transition graph, or the search may not terminate at all, even if the graph is finite.

2.3 Particular Forward Search Method

This section presents several search algorithms, each of which constructs a search tree. Each search algorithm is a special case of algorithm 1, obtained by defining a different sorting function for Q. Most of these are just classical graph search algorithms [16].

2.3.1 Breadth First

The method given in Section 2.2.1 specifies Q as a First-In First- Out (FIFO) queue, which selects states using the first-come, first-serve principle. This causes the search frontier to grow uniformly and is therefore referred to as breadth-first search. All plans that have k steps are exhausted before plans with $k + 1$ steps are investigated. Therefore, breadth first guarantees that the first solution found will use the smallest number of steps. On detection that a state has been revisited, there is no work to do in line 12. Since the search progresses in a series of wavefronts, breadth-first search is systematic. In fact, it even remains systematic if it does not keep track of repeated states (however, it will waste time considering irrelevant cycles).

The asymptotic running time of breadth-first search is $O(|V| + |E|)$, in which $|V|$ and $|E|$ are the numbers of vertices and edges, respectively, in the state transition graph (recall, however, that the graph is usually not the input; for example, the input may be the rules

of the Rubik's cube). This assumes that all basic operations, such as determining whether a state has been visited, are performed in constant time. In practice, these operations will typically require more time and must be counted as part of the algorithm's complexity. The running time can be expressed in terms of the other representations. Recall that $|V| = |X|$ is the number of states. If the same actions U are available from every state, then $|E| = |U||X|$. If the action sets $U(x_1)$ and $U(x_2)$ are pairwise disjoint for any $x_1, x_2 \in X$, then $|E| = |U|$.

2.3.2 Depth First

By making Q a stack (Last-In, First-Out; or LIFO), aggressive exploration of the state transition graph occurs, as opposed to the uniform expansion of breadth-first search. The resulting variant is called depth-first search because the search dives quickly into the graph. The preference is toward investigating longer plans very early. Although this aggressive behavior might seem desirable, note that the particular choice of longer plans is arbitrary. Actions are applied in the **forall** loop in whatever order they happen to be defined. Once again, if a state is revisited, there is no work to do in line 12. Depth-first search is systematic for any finite X but not for an infinite X because the search could easily focus on one direction and completely miss large portions of the search space as the number of iterations tends to infinity. The running time of depth first search is also $O(|V| + |E|)$.

2.3.3 Dijkstra's Algorithm

Up to this point, there has been no reason to prefer one action over any other in the search. Section 2.3 will formalize optimal discrete planning and will present several algorithms that find optimal plans. Before go- ing into that, we present a systematic search algorithm that finds optimal plans because it is also useful for finding feasible plans. The result is the well-known Dijkstra's algorithm for finding single-source shortest paths in a graph [which is a special form of dynamic programming.

Suppose that every edge, $e \in E$, in the graph representation of a discrete planning problem has an associated nonnegative cost $l(e)$, which is the cost to apply the action. The cost $l(e)$ could be written using the state-space representation as $l(x, u)$, indicating that it costs $l(x, u)$ to apply action u from state x. The total cost of a plan is just the sum of the edge costs over the path from the initial state to a goal state.

The priority queue, Q, will be sorted according to a function $C : X \to [0, \infty]$, called the cost-to-come. For each state x, the value $C^*(x)$ is called the optimal1 cost-to-come from the initial state x_I. This optimal cost is obtained by summing edge costs,$l(e)$, over all possible paths from x_I to x and using the path that produces the least cumulative cost. If the cost is not known to be optimal, then it is written as $C(x)$. The cost-to-come is computed incrementally during the execution of the search algorithm1. Initially, $C^*(x_I) = 0$. Each time the state x' is generated, a cost is computed as $C(x') = C^*(x) + l(e)$, in which e is the edge from x to x' (equivalently, we may write $C(x') = C^*(x) + l(x, u)$). Here, $C(x')$ represents the best cost-to-come that is known so far, but we do not write C^* because it is not yet known whether x' was reached optimally. Due to this, some work is required in line 12. If x' already exists in Q, then it is possible that the newly discovered path to x' is more efficient. If so, then the cost-to-come value $C(x')$ must be lowered for x', and Q must be reordered accordingly.

When does $C(x)$ finally become $C^*(x)$ for some state x? Once x is removed from Q using $Q.GetFirst()$, the state becomes dead, and it is known that x cannot be reached with a lower cost. This can be argued by induction. For the initial state, $C^*(x_I)$ is known, and this serves as the base case. Now assume that every dead state has its optimal cost-to-come correctly determined. This means that their cost-to-come values can no longer change. For the first element, x, of Q, the value must be optimal because any path that has a lower total cost would have to travel through another state in Q, but these states already have higher costs. All paths that pass only through dead states were already considered in producing $C(x)$. Once all edges leaving x are explored, then x can be declared as dead, and the induction continues.

The running time is $O(|V| \log |V| + |E|)$, in which $|V|$ and $|E|$ are the numbers of edges and vertices, respectively, in the graph representation of the discrete planning problem. This assumes that the priority queue is implemented with a Fibonacci heap, and that all other operations, such as determining whether a state has been visited, are performed in constant time. If other data structures are used to implement the priority queue, then higher running times may be obtained.

2.3.4 Other General Search Schemes

In alternative to forward searching methods there *backward* and *bilateral* methods. The former are essentially similar to the methods seen so far, but they start from the goal state x_G and work backward until the initial state is reached. The former build two (or more) trees that are merged at certain point in order to find a possible path. One tree is grown from the initial state, and the other is grown from the goal state. The search terminates with success when the two trees meet.

2.4 Discrete Optimal Planning

This section extends Formulation 1 to allow optimal planning problems to be defined. Rather than being satisfied with any sequence of actions that leads to the goal set, suppose we would like a solution that optimizes some criterion, such as time, distance, or energy consumed. Three important extensions will be made: 1) A stage index will be used to conveniently indicate the current plan step; 2) a cost functional will be introduced, which behaves like a taxi meter by indicating how much cost accumulates during the plan execution; and 3) a termination action will be introduced, which intuitively indicates when it is time to stop the plan and fix the total cost.

The presentation involves three phases. First, the problem of finding optimal paths of a fixed length is covered in Section 2.3.1. The approach, called value iteration, involves iteratively computing optimal cost-to-go functions over the state space. Although this case is not very useful by itself, it is much easier to understand than the general case of variable-length plans. Once the concepts from this section are understood, their extension to variable-length plans will be much clearer and is covered in Section 2.3.2. Finally, Section 2.3.3 explains the close relationship between value iteration and Dijkstra?s algorithm, which was covered in Section 2.2.1.

The fixed-length optimal planning formulation will be given shortly, but first we introduce some new notation. Let π_K denote a K-step plan, which is a sequence $(u_1, u_2, ..., u_K)$

of K actions. If π_K and x_I are given, then a sequence of states, $(x_1, x_2, ..., x - K + 1)$, can be derived using the state transition function, f. Initially, $x_1 = x_I$, and each subsequent state is obtained by $x_{k+1} = f(x_k, u_k)$.

The model is now given; the most important addition with respect to Formulation 1 is L, the cost functional.

Formulation 2. ***(Discrete Fixed-Length Optimal Planning)***

1. *All of the components from Formulation 1 are inherited directly: X, $U(x)$, f, x_I, and X_G, except here it is assumed that X is finite (some algorithms may easily extend to the case in which X is countably infinite, but this will not be considered here).*

2. *A number, K, of stages, which is the exact length of a plan (measured as the number of actions, $u_1, u_2, ..., u_K$). States may also obtain a stage index. For example, x_{k+1} denotes the state obtained after u_k is applied.*

3. *Let L denote a stage-additive cost (or loss) functional, which is applied to a K-step plan, π_K . This means that the sequence $(u_1, ..., u_K)$ of actions and the sequence $(x_1, ..., x_{K+1})$ of states may appear in an expression of L. For convenience, let F denote the final stage, $F = K + 1$ (the application of u_K advances the stage to $K + 1$). The* cost functional *is*

$$L(\pi_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_F). \tag{2.3}$$

   The cost term *$l(x_k, u_k)$ yields a real value for every $x_k \in X$ and $u_k \in U(x_k)$. The final term $l_F(x_F)$ is outside of the sum and is defined as $l_F(x_F) = 0$ if $x_F \in X_G$, and $l_F(x_F) = \infty$ otherwise.*

Now the task is to find a plan that minimizes L.

2.4.1 Optimal Fixed-Length Plans

Consider computing an optimal plan under Formulation 2. One could naively generate all length-K sequences of actions and select the sequence that produces the best cost, but this would require $O(|U|^K)$ running time (imagine K nested loops, one for each stage), which is clearly prohibitive. Luckily, the *dynamic programming principle* helps. The main observation is that portions of optimal plans are themselves optimal. It would be absurd to be able to replace a portion of an optimal plan with a portion that produces lower total cost; this contradicts the optimality of the original plan.

The principle of optimality leads directly to an iterative algorithm, called *value iteration*, that can solve a vast collection of optimal planning problems. The idea is to iteratively compute optimal cost-to-go (or cost-to-come) functions over the state space. In some cases, the approach can be reduced to Dijkstra's algorithm; however, this only occurs under some special conditions.

Backward value iteration

As for the search methods, there are both forward and backward versions of the approach. We will cover backward case, the forward case is analogous.

The key to deriving long optimal plans from shorter ones lies in the construction of optimal cost-to-go functions over X. For k from 1 to F, let G_k^* denote the cost that accumulates from stage k to F under the execution of the optimal plan:

$$G_k^*(x_k) = \min_{u_k,\ldots,u_K} \Big\{ \sum_{i=k}^{K} l(x_i, u_i) + l_F(x_F) \Big\}. \tag{2.4}$$

The optimal cost-to-go for the boundary condition of $k = F$ reduces to

$$G_F^*(x_F) = l_F(x_F). \tag{2.5}$$

Since there are no stages in which an action can be applied, the final stage cost is immediately received.

Now consider an algorithm that makes K passes over X, each time computing G_k^* from G_{k+1}^*, as k ranges from F down to 1. In the first iteration, G_F^* is copied from l_F without significant effort. In the second iteration, G_K^* is computed for each $x_K \in X$ as

$$G_K^*(x_K) = \min_{u_K} \big\{ l(x_K, u_K) + l_F(x_F) \big\}. \tag{2.6}$$

Since $l_F = G_F^*$ and $x_F = f(x_K, u_K)$,substitutions can be made into (2.7) to obtain

$$G_K^*(x_K) = \min_{u_K} \big\{ l(x_K, u_K) + G_F^*(f(x_K, u_K)) \big\}. \tag{2.7}$$

which is straightforward to compute for each $x_K \in X$. This computes the costs of all optimal one-step plans from stage K to stage $F = K + 1$. It will be shown next that G^*k can be computed similarly once G_{k+1}^* is given. Note that (2.5) it can be written as

$$G_k^*(x_k) = \min_{u_k,} \Big\{ \min_{u_{k+1},\ldots,u_K} \Big\{ l(x_k, u_k) + \sum_{i=k+1}^{K} l(x_i, u_i) + l_F(x_F) \Big\} \Big\}. \tag{2.8}$$

by pulling the first term out of the sum and by separating the minimization over uk from the rest, which range from u_{k+1} to u_K. The second min does not affect the $l(x_k, u_k)$ term; thus, $l(x_k, u_k)$ can be pulled outside to obtain

$$G_k^*(x_k) = \min_{u_k,} \Big\{ l(x_k, u_k) + \min_{u_{k+1},\ldots,u_K} \Big\{ \sum_{i=k+1}^{K} l(x_i, u_i) + l_F(x_F) \Big\} \Big\}. \tag{2.9}$$

The inner min is exactly the definition of the optimal cost-to-go function G_{k+1}^*. Upon substitution, this yields the recurrence

$$G_K^*(x_k) = \min_{u_k} \big\{ l(x_K, u_K) + G_{k+1}^*(x_{k+1}) \big\}, \tag{2.10}$$

in which $x_{k+1} = f(x_k, u_k)$. Now that the right side of (2.11) depends only on x_k, u_k, and G_{k+1}^*, the computation of G_k^* easily proceeds in $O(|X||U|)$ time. This computation is called a value iteration. Note that in each value iteration, some states receive an infinite value only because they are not reachable; a $(K - k)$- step plan from x_k to X_G does not exist. This means that there are no actions, $u_k \in U(x_k)$, that bring x_k to a state $x_{k+1} \in X$ from which a $(K - k - 1)$-step plan exists that terminates in X_G.

Summarizing, the value iterations proceed from G_F^* to G_1^* in $O(K|X||U|)$ time. The resulting G_1^* may be applied to yield $G_1^*(x_I)$, the optimal cost to go to the goal from x_I. It also conveniently gives the optimal cost-to-go from any other initial state. This cost is infinity for states from which X_G cannot be reached in K stages.

2.4.2 Optimal Plans of Unspecified Lengths

The value-iteration method for fixed-length plans can be generalized nicely to the case in which plans of different lengths are allowed. There will be no bound on the maximal length of a plan; therefore, the current case is truly a generalization of Formulation 1 because arbitrarily long plans may be attempted in efforts to reach X_G. The model for the general case does not require the specification of K but instead introduces a special action, u_T .

Formulation 3. ***(Discrete Optimal Planning)***

1. *All of the components from Formulation 1 are inherited directly: X, $U(x)$, f, x_I, and X_G. Also, the notion of stages from Formulation 2 is used.*

2. *Let L denote a stage-additive cost functional, which may be applied to any K-step plan, π_K, to yield*

$$L(\pi_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_F). \tag{2.11}$$

   In comparison with L from Formulation 2, the present expression does not consider K as a predetermined constant. It will now vary, depending on the length of the plan. Thus, the domain of L is much larger.

3. *Each $U(x)$ contains the special* termination action, *u_T . If u_T is applied at x_k, then the action is repeatedly applied forever, the state remains unchanged, and no more cost accumulates. Thus, for alli $\geq k$, $u_i = u_T$, $x_i = x_k$, and $l(x_i, u_T) = 0$.*

The next step is to remove the dependency on K. Consider running backward value iterations indefinitely. At some point, G_1^* will be computed, but there is no reason why the process cannot be continued onward to G_0^*, G_{-1}^*, and so on.

Eventually, enough iterations will have been executed so that an optimal plan is known from every state that can reach X_G. From that stage, say k, onward, the cost-to-go values from one value iteration to the next will be stationary, meaning that for all $i \leq k$, $G_{i-1}^*(x) = G^*i(x)$ for all $x \in X$. Once the stationary condition is reached, the cost-to-go function no longer depends on a particular stage k. In this case, the stage index may be dropped, and the recurrence becomes

$$G^*(x) = \min_u \big\{ l(x, u_) + G^*(f(x, u)) \big\}. \tag{2.12}$$

Since the particular stage index is unimportant, let $k = 0$ be the index of the final stage, which is the stage at which the backward value iterations begin. Hence, G_0^* is the final stage cost, which is obtained directly from l_F . Let $-K$ denote the stage index at which the cost-to-go values all become stationary. At this stage, the optimal cost-to-go function, $G^* : X \to R \cup \{\infty\}$, is expressed by assigning $G^* = G_{-K}^*$. In other words, the particular stage index no longer matters. The value $G^*(x)$ gives the optimal cost to go from state $x \in X$ to the specific goal state x_G.

2.5 Transition to Continuous Spaces

A central theme throughout motion planning is to transform the continuous model into a discrete one. Due to this transformation, many algorithms from Chapter 2 are embedded in motion planning algorithms. There are two alternatives to achieving this transformation:

- *combinatorial motion planning*;
- sampling-based motion planning

Combinatorial motion planning builds a discrete representation that *exactly* represents the original problem. This leads to complete planning approaches, which are guaranteed to find a solution when it exists, or correctly report failure if one does not exist. Sampling-based motion planning instead refers to algorithms that use collision detection methods to sample the configuration space and conduct discrete searches that utilize these samples. In this case, completeness is sacrificed, but it is often replaced with a weaker notion, such as *resolution completeness or probabilistic completeness.*Combinatorial methods can solve virtually any motion planning problem, and in some restricted cases, very elegant solutions may be efficiently constructed in practice. However, for the majority of ?industrial-grade? motion planning problems, the running times and implementation difficulties of these algorithms make them un- appealing. Sampling-based algorithms have fulfilled much of this need in recent years by solving challenging problems in several settings, such as automobile assembly, humanoid robot planning, and conformational analysis in drug design.

This thesis will focus only on sampling-based motion planning, the reader can refer to [1] for more detail about combinatorial methods.

Chapter 3

The Configuration Space

The state space for motion planning is a set of possible transformations that could be applied to the robot. This will be referred to as the *configuration space*, or briefly *C-space*. Once the configuration space is clearly understood, many motion planning problems that appear different in terms of geometry and kinematics can be solved by the same planning algorithms. This level of abstraction is therefore very important.

3.1 Geometric Modeling

Formulating and solving motion planning problems require defining and manipulating complicated geometric models of a system of bodies in space.

The first step is to define the world $\mathcal{W}$ for which there are two possible choices: a $2D$ world, in which $\mathcal{W} = \mathbb{R}^2$, and a $3D$-world, in which $\mathcal{W} = \mathbb{R}^3$. These choices should be sufficient for most problems; however, one might also want to allow more complicated worlds, such as the surface of a sphere or even a higher dimensional space. Unless otherwise stated, the world generally contains two kinds of entities:

obstacles: Portions of the world that are permanently occupied, for example, as in the walls of a building.

robots: Bodies that are modeled geometrically and are controllable via a motion plan.

Both obstacles and robots will be considered as (closed) subsets of $\mathcal{W}$. Let $\mathcal{O}$ refer to the obstacle region, which is a subset of $\mathcal{W}$. Let $\mathcal{A}$ refer to the robot, which is a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$, matching the dimension of $\mathcal{W}$. Although $\mathcal{O}$ remains fixed in the world, $\mathcal{W}$, motion planning problems will require moving the robot, $\mathcal{A}$. The motion of a robot can be described in terms of kinematic transformation applied to the "original" configuration of the robot. Even if this framework allows modeling of very complicate systems such as chains of bodies or flexible organic molecules, in the following the focus will be only on rigid bodies. The reason behind this choice lies in part in its simplicity and familiarity, and in part in the fact that many robots can be considered as assembly of many parts singularly modeled as rigid.

3.2 Rigid-Body Transformation

Suppose that a rigid robot, $\mathcal{A}$, is defined as a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. A rigid-body transformation is a function, $h : \mathcal{A} \to \mathcal{W}$, that maps every point of $\mathcal{A}$ into $\mathcal{W}$ with two requirements: 1) The distance between any pair of points of $\mathcal{A}$ must be preserved, and 2) the orientation of $\mathcal{A}$ must be preserved.

Using standard function notation, $h(a)$ for some $a \in \mathcal{A}$ refers to the point in $\mathcal{W}$ that is occupied by a. Let

$$h(\mathcal{A}) = \{h(a) \in \mathcal{W} | a \in \mathcal{A}\} \tag{3.1}$$

which is the image of h and indicates all points in $\mathcal{W}$ occupied by the transformed robot.

It will become important to study families of transformations, in which some parameters are used to select the particular transformation. Therefore, it makes sense to generalize h to accept two variables: a parameter vector, $q \in \mathbb{R}^n$, along with $a \in \mathcal{A}$. The resulting transformed point a is denoted by $h(q, a)$, and the entire robot is transformed to $h(q, A) \in \mathcal{W}$.

It was assumed so far that $\mathcal{A}$ is defined in $\mathbb{R}^2$ or $\mathbb{R}^3$, but before it is transformed, it is not considered to be a subset of $\mathcal{W}$. The transformation h places the robot in $\mathcal{W}$. In the coming material, it will be convenient to indicate this distinction using coordinate frames. The origin and coordinate basis vectors of $\mathcal{W}$ will be referred to as the *world frame*.Thus, any point $w \in \mathcal{W}$ is expressed in terms of the world frame.

The coordinates used to define $\mathcal{A}$ are initially expressed in the *body frame*, which represents the origin and coordinate basis vectors of $\mathbb{R}^2$ or $\mathbb{R}^3$. In the case of $\mathcal{A} \subset \mathbb{R}^2$, it can be imagined that the body frame is painted on the robot. Transforming the robot is equivalent to converting its model from the body frame to the world frame. This has the effect of placing $\mathcal{A}$ into $\mathcal{W}$ at some position and orientation.

3.2.1 2D Transformations

Translation

A rigid robot $\mathcal{A} \subset \mathbb{R}^2$ is translated by using two parameters, $x_t, y_t \in \mathbb{R}$. Taking $q = (x_t, y_t)$ as the transformation parameter, h is defined as

$$h(x, y) = (x + x_t, y + y_t). \tag{3.2}$$

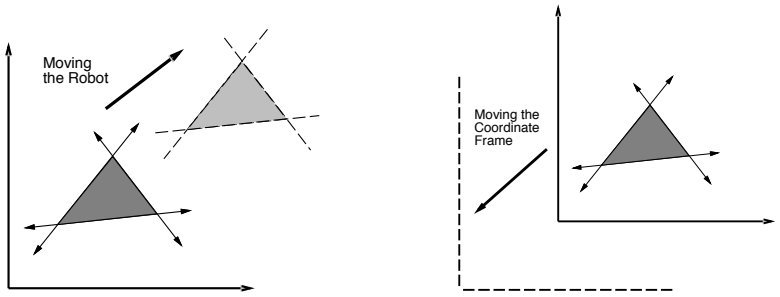
Now consider a solid representation of $\mathcal{A}$, defined in terms of primitives. Each primitive of the form

$$H_i = \{(x, y) \in \mathcal{R}^2 | f(x, y) \leq 0\} \tag{3.3}$$

is transformed to

$$h(H_i) = \{(x, y) \in \mathcal{W} | f(x - x_t, y - y_t) \leq 0\} \tag{3.4}$$

The translated robot is denoted as $\mathcal{A}(x_t, y_t)$. Translation by $(0, 0)$ is the identity transformation, which results in $\mathcal{A}(0, 0) = \mathcal{A}$, if it is assumed that $\mathcal{A} \subset \mathcal{W}$ (recall that $\mathcal{A}$ does not necessarily have to be initially embedded in $\mathcal{W}$). It will be convenient to use the term degrees of freedom to refer to the maximum number of independent parameters that are needed to completely characterize the transformation applied to the robot. If the set of allowable values for x_t and y_t forms a two-dimensional subset of $\mathbb{R}^2$, then the degrees of freedom is two.



(a) Translation of the robot (b) Translation of the frame

Figure 3.1: Every transformation has two interpretations.

Rotation

The robot, $\mathcal{A}$, can be rotated counterclockwise by some angle $\theta \in [0, 2\pi)$ by mapping every $(x, y) \in \mathcal{A}$ as

$$(x, y) \mapsto (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta) \tag{3.5}$$

Using a 2×2 rotation matrix,

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \tag{3.6}$$

the transformation can be written as

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}. \tag{3.7}$$

The column vectors of $R(\theta)$ are unit vectors, and their inner product (or dot product) is zero, indicating that they are orthogonal. Note that the rotation is performed about the origin. Thus, when defining the model of $\mathcal{A}$, the origin should be placed at the intended axis of rotation. Using the semi-algebraic model, the entire robot model can be rotated by transforming each primitive, yielding $\mathcal{A}(\theta)$. The inverse rotation, $R(?\theta)$, must be applied to each primitive.

Combining translation and rotation

Suppose a rotation by θ is performed, followed by a translation byx_t, y_t. This can be used to place the robot in any desired position and orientation. If the operations are applied successively, each $(x, y) \in \mathcal{A}$ is transformed to

$$\begin{pmatrix} x\cos\theta - y\sin\theta + x_t \\ x\sin\theta + y\cos\theta + y_t \end{pmatrix}. \tag{3.8}$$

The following matrix multiplication yields the same result for the first two vector components:

$$\begin{pmatrix} \cos\theta & -\sin\theta & x_t \\ \sin\theta & \cos\theta & y_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta + x_t \\ x\sin\theta + y\cos\theta + y_t \\ 1 \end{pmatrix}. \tag{3.9}$$

This implies that the 3×3 matrix,

$$\begin{pmatrix} \cos\theta & -\sin\theta & x_t \\ \sin\theta & y\cos\theta & y_t \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.10}$$

represents a rotation followed by a translation. The matrix T will be referred to as a *homogeneous transformation matrix.*

The transformed robot is denoted by $\mathcal{A}(x_t, y_t, \theta)$, and in this case there are three degrees of freedom.

3.2.2 3D Transformation

Rigid-body transformations for the 3D case are conceptually similar to the 2D case; however, the 3D case appears more difficult because rotations are significantly more complicated.

3D translation

The robot, $\mathcal{A}$, is translated by some $x_t, y_t, z_t \in \mathbb{R}$ using

$$(x, y, z) \mapsto (x + x_t, y + y_y, z + z_t). \tag{3.11}$$

A primitive of the form

$$H_i = \{(x, y) \in \mathcal{R}^2 | f(x, y, z) \leq 0\} \tag{3.12}$$

is transformed to

$$h(H_i) = \{(x, y) \in \mathcal{W} | f(x - x_t, y - y_t, z - z - t) \leq 0\} \tag{3.13}$$

The translated robot is denoted as $A(x_t, y_t, z_t)$.

Yaw, Pitch and Roll rotations

A 3D body can be rotated about three orthogonal axes, as shown in Figure 3.2. Borrowing aviation terminology, these rotations will be referred to as yaw, pitch, and roll:

Denoting with α, β, γ the *Euler's angles*, the transformed robot can be indicated as $\mathcal{A}(\alpha, \beta, \gamma)$.

The homogeneous transformation matrix for 3D bodies is perfectly analogous to its 2D counterpart. A 3D rigid body that is capable of translation and rotation therefore has six degrees of freedom.

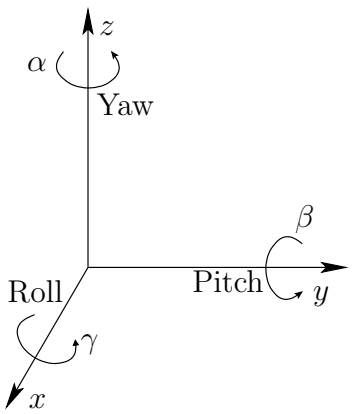


Figure 3.2: Any three-dimensional rotation can be described as a sequence of yaw, pitch, and roll rotations.

3.3 Defining the Configuration Space

The configuration space of a robot can be thought as a mainfold of dimension n where n is the the number of degrees of freedom of the robot. To solve a motion planning problem, algorithms must conduct a search in the C-space. The C-space provides a powerful abstraction that converts the complicated models that describe the mechanics of the particular system under examination into the general problem of computing a path that traverses a manifold. By developing algorithms directly for this purpose, they apply to a wide variety of different kinds of robots and transformations. In order to make this discussion more concrete, in the following we provides some examples of configuration space for two simple but import example of robots: the 2-dimensional and 3-dimensional rigid body.

3.3.1 2D Rigid Bodies: $SE(2)$

Section 3.2.1 expressed how to transform a rigid body in $\mathbb{R}^2$ by a homogeneous transformation matrix, T. The task in this chapter is to characterize the set of all possible rigid-body transformations.

Since any $x_t, y_t \in \mathbb{R}$ can be selected for translation, this alone yields a manifold $M_1 = \mathbb{R}^2$. Independently, any rotation, $\theta \in [0, 2\pi)$, can be applied. Since 2π yields the same rotation as 0, they can be identified, which makes the set of 2D rotations into a manifold, $M_2 = \mathbb{S}^1$. To obtain the manifold that corresponds to all rigid-body motions, simply take $\mathcal{C} = M_1 \times M_2 = \mathbb{R}^2 \times \mathbb{S}^1$. The answer to the question is that the C-space is a kind of cylinder.

It is important to consider the topological implications of $\mathcal{C}$. Since $\mathbb{S}^1$ is multiply connected, $\mathbb{R}^2 \times \mathbb{S}^1$ is multiply connected. It is difficult to visualize $\mathcal{C}$ because it is a 3D manifold; however, there is a nice interpretation using identification. Start with the open unit cube, $(0, 1)^3 \subset \mathbb{R}^3$. Include the boundary points of the form $(x, y, 0)$ and $(x, y, 1)$, and make the identification $(x, y, 0) \sim (x, y, 1)$ for all $x, y \in (0, 1)$. This means that when traveling in the x and y directions, there is a ?frontier? to the C-space; however, traveling in the z direction causes a wraparound.

It is very important for a motion planning algorithm to understand that this wraparound

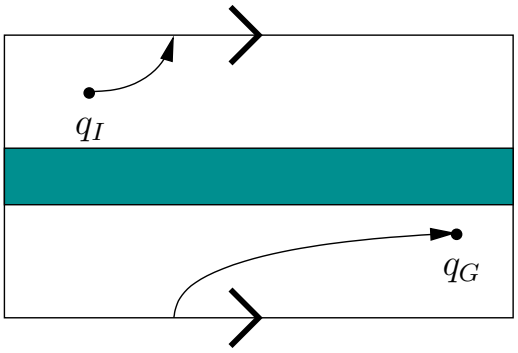


Figure 3.3: A planning algorithm may have to cross the identification boundary to find a solution path.

exists. For example, consider $\mathbb{R}\times\mathbb{S}^1$ because it is easier to visualize. Imagine a path planning problem for which $\mathcal{C} = \mathbb{R} \times \mathbb{S}^1$, as depicted in Figure 3.3. Suppose the top and bottom are identified to make a cylinder, and there is an obstacle across the middle. Suppose the task is to find a path from q_I to q_G. If the top and bottom were not identified, then it would not be possible to connect q_I to q_G; however, if the algorithm realizes it was given a cylinder, the task is straightforward. In general, it is very important to understand the topology of $\mathcal{C}$; otherwise, potential solutions will be lost.

3.3.2 3D Rigid Bodies: $SE(3)$

The C-space for a 3D rigid body is the six-dimensional mainfold $\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3$, where $\mathbb{R}*^n$ represents the *n-dimensional real projective space*. The standard definition of of $\mathbb{RP}^n$ is the set of all lines in $\mathcal{R}^{n+1}$ that pass through the origin. Each line is considered as a point in $\mathbb{RP}^n$. Defining the n-dimensional hypersphere $\mathbb{S}^n$ as

$$\mathbb{S}^n := \{x \in \mathbb{R}^{n+1} | \quad ||x|| \leq 1\}, \tag{3.14}$$

each line of $\mathbb{R}^{n+1}$ intersects $\mathbb{S}^n$ in exactly two places. These intersection points are called *antipodal*, which means that they are as far from each other as possible on $\mathbb{S}^n$. The pair is also unique for each line. If we identify all pairs of antipodal points of $\mathbb{S}^n$, a homeomorphism can be defined between each line through the origin of$\mathbb{R}^{n+1}$ and each antipodal pair on the sphere. This means that the resulting manifold, $\mathbb{S}^n/\sim$, is homeomorphic to $\mathbb{RP}^n$.

The main problem in this section is to determine the topology of $SO(3)$. Is it possible to parametrize the set of rotation using the Euler's angles, however there are some cases in which nonzero angles yield the identity rotation matrix, which is equivalent to $\alpha = \beta = \gamma = 0$. There are also cases in which a continuum of values for yaw, pitch, and roll angles yield the same rotation matrix. These problems destroy the topology, which causes both theoretical and practical difficulties in motion planning. For this reason, *unit quaternions* will be used to represent 3D rotations. Let $\mathbb{H}$ represent the set of quaternions, in which each quaternion, $h \in \mathbb{H}$, is represented as $h = a + bi + cj + dk$, and $a, b, c, d \in \mathbb{R}$. A quaternion can be considered as a four-dimensional vector. The symbols i, j, and k are used to denote three imaginary components of the quaternion. The following relationships are defined: $i^2 = j^2 = k^2 = ijk =?1$,from which it follows that $ij = k, jk = i$, and$ki = j$. Using these, multiplication of two quaternions, $h_1 = a_1 + b_1i + c_1j + d_1k$ and $h_2 = a_2 + b_2i + c_2j + d_2k$,

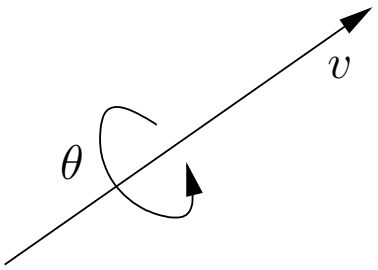


Figure 3.4: Any 3D rotation can be considered as a rotation by an angle θ about the axis given by the unit direction vector $v = [v_1 v_2 v_3]$.

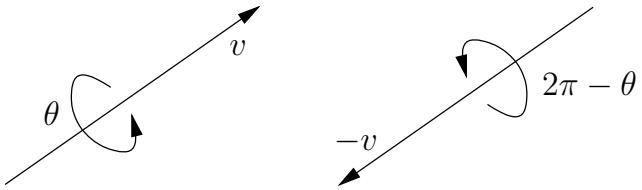


Figure 3.5: There are two ways to encode the same rotation.

can be derived to obtain $h_1 \cdot h_2 = a_3 + b_3 i + c_3 j + d_3 k$, in which

$$\begin{cases} a_3 = a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 \\ b_3 = a_1 b_2 + a_2 b_1 + c_1 d_2 - c_2 d_1 \\ c_3 = a_1 c_2 + a_2 c_1 + b_2 d_1 - b_1 d_2 \\ d_3 = a_1 d_2 + a_2 d_1 + b_1 c_2 - b_2 c_1 \end{cases} \tag{3.15}$$

Using this operation, it can be shown that $\mathbb{H}$ is a group with respect to quaternion multiplication. Note, however, that the multiplication is not commutative. For convenience, quaternion multiplication can be expressed in terms of vector multiplications, a dot product, and a cross product. Let $v = [b \quad c \quad d]$ be a three-dimensional vector that represents the final three quaternion components. The first component of $h_1 \cdot h_2$ is $a_1 a_2 - v_1 \cdot v_2$. The final three components are given by the three-dimensional vector $a_1 v_2 + a_2 v_1 + v_1 \times v_2$. Unit quaternions are quaternions for which $a^2 + b^2 + c^2 + d^2 = 1$. Note that this forms a subgroup because the multi- plication of unit quaternions yields a unit quaternion, and the other group axioms hold. The next step is to describe a mapping from unit quaternions to $SO(3)$. It may be proved that the quaternion

$$h = \cos\frac{\theta}{2} + \big(v_1 \sin\frac{\theta}{2}\big)i + \big(v_2 \sin\frac{\theta}{2}\big)j + \big(v_3 \sin\frac{\theta}{2}\big)k \tag{3.16}$$

maps to the rotation shown in Figure 3.4.

Unfortunately, this representation is not unique. As illustrated in Figure 3.5 the quaternions h and $-h$ represent the same rotation because a rotation of θ about the direction v is equivalent to a rotation of $2\pi - \theta$ about the direction $?v$.

Note that the set of unit quaternions is homeomorphic to $\mathbb{S}^3$ because of the constraint $a^2 + b^2 + c^2 + d^2 = 1$. . Using identification, declare $h \sim -h$ for all unit quaternions. This means that the antipodal points of $\mathbb{S}^3$ are identified. Recall that when antipodal points are identified, $\mathbb{RP}^n \cong \mathbb{S}^n / \sim$. Hence, $SO(3) \cong \mathbb{RP}^3$. Now that the complicated part of representing $SO(3)$ has been handled, the representation of $SE(3)$ is straightforward. The general form of a matrix T in $SE(3)$ is determined by a rotation matrix $R \in SO(3)$ and and a vector $v \in \mathbb{R}^3$. Since $SO(3) \cong \mathbb{RP}^3$, and translations can be chosen independently, the resulting C-space for a rigid body that rotates and translates in $\mathbb{R}^3$ is

$$\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3. \tag{3.17}$$

As expected, the dimension of $\mathcal{C}$ is exactly the number of degrees of freedom of a free-floating body in space.

3.4 Definition of the Basic Motion Planning Problem

3.4.1 The Obstacle Region

Suppose that the world, $W = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, contains an obstacle region,$\mathcal{O} \subset \mathcal{W}$. Assume here that a robot, $\mathcal{A}W$, is defined. Let $q \subset \mathcal{C}$ denote the configuration of $\mathcal{A}$.

The *obstacle region*, $\mathcal{C}_{\text{obs}} \subset \mathcal{C}$, is defined as

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O} \neq \}, \tag{3.18}$$

which is the set of all configurations, q, at which $A(q)$, the transformed robot, intersects the obstacle region,$\mathcal{O}$. Since $\mathcal{O}$ and $\mathcal{A}(q)$ are closed sets in $\mathcal{W}$, the obstacle region is a closed set in $\mathcal{C}$.

The leftover configurations are called the *free space*, which is defined and denoted as $\mathcal{C}_{\text{free}} = \mathcal{C}$
$\mathcal{C}_{\text{obs}}$. Since $\mathcal{C}$ is a topological space and $\mathcal{C}_{\text{obs}}$ is closed, $\mathcal{C}_{\text{free}}$ must be an open set. This implies that the robot can come arbitrarily close to the obstacles while remaining in $\mathcal{C}_{\text{free}}$.

The idea of getting arbitrarily close may be nonsense in practical robotics, but it makes a clean formulation of the motion planning problem. Since $\mathcal{C}_{\text{free}}$ is open, it becomes impossible to formulate some optimization problems, such as finding the shortest path. In this case, the closure, $\text{cl}(\mathcal{C}_{\text{free}})$, should instead be used.

Finally, enough tools have been introduced to precisely define the motion planning problem. The main difficulty is that it is neither straightforward nor efficient to construct an explicit boundary or solid representation of either $\mathcal{C}_{\text{free}}$ or $\mathcal{C}_{\text{obs}}$. The components are as follows:

Formulation 4. ***The Piano Mover's Problem***

1. *A* world $\mathcal{W}$ *that can be either* $\mathcal{W} = \mathbb{R}^2$ *or* $\mathcal{W} = \mathbb{R}^3$.

2. *An obstacle region* $\mathcal{O} \in \mathcal{W}$ *in the world.*

3. *A robot defined in* $\mathcal{W}$. *It may be for example a rigid robot* $\mathcal{A}$ *or a collection of* m *links,*$\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_m$.

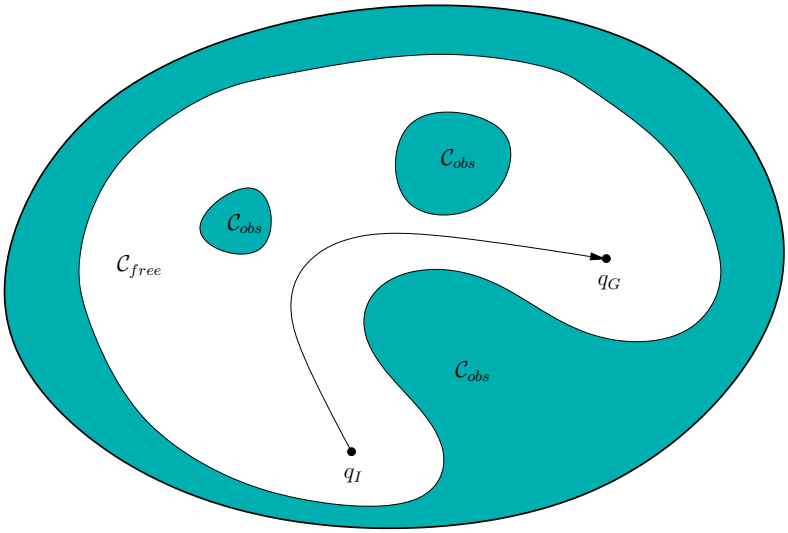


Figure 3.6: Graphic depiction of the basic motion planning problem

4. *The configuration space $\mathcal{C}$ determined by specifying the set of all possible transformations that may be applied to the robot. From this, $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are derived.*

5. *A configuration, $q_I \in \mathcal{C}_{free}$ designated as the* initial configuration.

6. *A configuration $q_G \in \mathcal{C}_{free}$ designated as the* goal configuration. *The initial and goal configurations together are often called a* query pair *(or query) and designated as (q_I, q_G).*

7. *A complete algorithm must compute a (continuous) path, $\tau : [0, T] \to \mathcal{C}_{free}$, such that $\tau(0) = q_I$ and $\tau(T) = q_G$, or correctly report that such a path does not exist.*

It was shown that this problem is PSPACE-hard, which implies NP-hard. The main problem is that the dimension of C is unbounded.

Chapter 4

Sampling-Based Motion Planning

This Chapter presents, *sampling-based motion planning*, one of the two main philosophies for addressing motion planning problems. The central idea is to avoid the explicit construction of $\mathcal{C}_{\text{obs}}$ and instead conduct a search that probes the C-space with a sampling scheme. This probing is enabled by a collision detection module, which the motion planning algorithm considers as a "black box". This enables the development of planning algorithms that are independent of the particular geometric models. This general philosophy has been very successful in recent years for solving problems from robotics, manufacturing, and biological applications that involve thousands and even millions of geometric primitives. Such problems would be practically impossible to solve using techniques that explicitly represent $\mathcal{C}_{\text{obs}}$.

It is useful to define several notions of completeness for sampling-based algorithms. These algorithms have the drawback that they result in weaker guarantees that the problem will be solved. An algorithm is considered *complete* if for any input it correctly reports whether there is a solution in a finite amount of time. If a solution exists, it must return one in finite time. Unfortunately, such completeness is not achieved with sampling-based planning. Instead, weaker notions of completeness are tolerated. The notion of denseness becomes important, which means that the samples come arbitrarily close to any configuration as the number of iterations tends to infinity. A deterministic approach that samples densely will be called *resolution complete*. This means that if a solution exists, the algorithm will find it in finite time; however, if a solution does not exist, the algorithm may run forever. Many sampling-based approaches are based on random sampling, which is dense with probability one. This leads to algorithms that are *probabilistically complete*, which

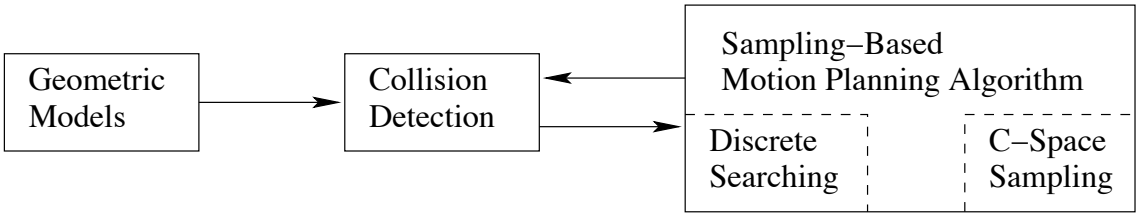


Figure 4.1: The sampling-based planning philosophy uses collision detection as a "black box" that separates the motion planning from the particular geometric and kinematic models. C-space sampling and discrete planning (i.e., searching) are performed.

means that with enough points, the probability that it finds an existing solution converges to one. The most relevant information, however, is the rate of convergence, which is usually very difficult to establish.

4.1 Sampling Theory

The state space for motion planning, $\mathcal{C}$, is uncountably infinite, yet a sampling- based planning algorithm can consider at most a countable number of samples. If the algorithm runs forever, this may be countably infinite, but in practice we expect it to terminate early after only considering a finite number of samples. This mismatch between the cardinality of $\mathcal{C}$ and the set that can be probed by an algorithm motivates careful consideration of sampling techniques. Once the sampling component has been defined, discrete planning methods from Chapter 2 may be adapted to the current setting. Their performance, however, hinges on the way the C-space is sampled. Since sampling-based planning algorithms are often terminated early, the particular order in which samples are chosen becomes critical. Therefore, a distinction is made between a sample set and a sample sequence. A unique sample set can always be constructed from a sample sequence, but many alternative sequences can be constructed from one sample set.

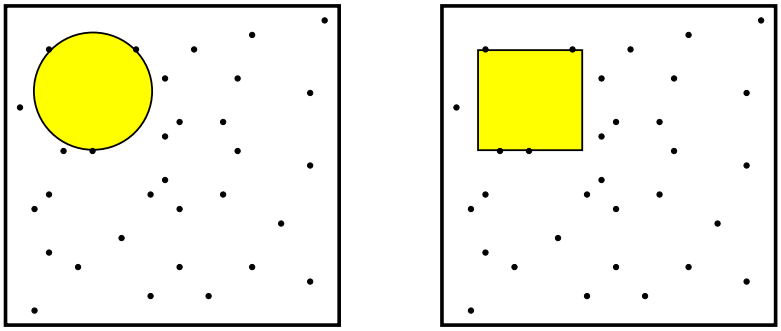
4.1.1 Random Sampling

Consider constructing an infinite sample sequence over $\mathcal{C}$. Ideally, the sequence eventually should reach every point in $\mathcal{C}$, but this is impossible because $\mathcal{C}$ is uncountably infinite. However, it is still possible for a sequence to get arbitrarily close to every element of $\mathcal{C}$ (assuming $\mathcal{C} \subset \mathcal{R}^m$). In topology, this is the notion of *denseness*. Let U and V be any subsets of a topological space. The set U is said to be dense in V if $\text{cl}(U) = V$. This means adding the boundary points to U produces V. A simple example is that $(0, 1) \subset \mathbb{R}$ is dense in $[0, 1] \subset \mathbb{R}$. A more interesting example is that the set $\mathbb{Q}$ of rational numbers is both countable and dense in $\mathbb{R}$. For any real number, such as $\pi \in \mathbb{R}$, there exists a sequence of fractions that converges to it. This sequence of fractions must be a subset of $\mathbb{Q}$. A sequence (as opposed to a set) is called dense if its underlying set is dense. The bare minimum for sampling methods is that they produce a dense sequence.

Interestingly, a random sequence is probably dense. Suppose that $\mathcal{C} = [0, 1]$. One of the simplest ways conceptually to obtain a dense sequence is to pick points at random. Suppose $I \subset [0, 1]$ is an interval of length e. If k samples are chosen independently at random, the probability that none of them falls into I is $(1 - e)^k$. As k approaches infinity, this probability converges to zero. This means that the probability that any nonzero-length interval in $[0, 1]$ contains no points converges to zero. However, the infinite sequence of independently, randomly chosen points is only dense with probability one, which is not the same as being certainly dense. The probability is just the Lebesgue measure, which is zero for a set of measure zero.

Random sampling is the easiest of all sampling methods to apply to C-spaces. One of the main reasons is that C-spaces are formed from Cartesian products, and independent random samples extend easily across these products. If $X = X_1 \times X_2$, and uniform random samples x_1 and x_2 are taken from X_1 and X_2, respectively, then (x_1, x_2) is a uniform random sample for X. This is very convenient in implementations. For example, suppose the motion planning problem involves 15 robots that each translate for any

$(x_t, y_t) \in [0,1]^2$; this yields $\mathcal{C} = [0,1]^{30}$. In this case, 30 points can be chosen uniformly at random from $[0,1]$ and combined into a 30-dimensional vector. Samples generated this way are uniformly randomly distributed over $\mathcal{C}$. Combining samples over Cartesian products is much more difficult for nonrandom (deterministic) methods. Although there are advantages to uni- form random sampling, there are also several disadvantages. This motivates the consideration of deterministic alternatives. Since there are trade-offs, it is important to understand how to use both kinds of sampling in motion planning. One of the first issues is that computer-generated numbers are not random. A *pseudo-random number generator* is usually employed, which is a deterministic method that simulates the behavior of randomness. Since the samples are not truly random, the advantage of extending the samples over Cartesian products does not necessarily hold. Sometimes problems are caused by unforeseen deterministic dependencies.



(a) L_2 dispersion (b) L_∞ dispersion

Figure 4.2: Graphic depiction of the dispersion of a set of samples

4.1.2 Low-Dispersion Sampling

This section describes an alternative to random sampling. Here, the goal is to optimize a criterion called *dispersion*. Intuitively, the idea is to place samples in a way that makes the largest uncovered area be as small as possible. This generalizes of the idea of *grid resolution*. For a grid, the resolution may be selected by defining the step size for each axis. As the step size is decreased, the resolution increases. If a grid-based motion planning algorithm can increase the resolution arbitrarily, it becomes resolution complete. Using the concepts in this section, it may instead reduce its dispersion arbitrarily to obtain a resolution complete algorithm. Thus, dispersion can be considered as a powerful generalization of the notion of "resolution".

Definition 1. *The* dispersion *of a finite set P of samples in a metric space (X, ρ) is*

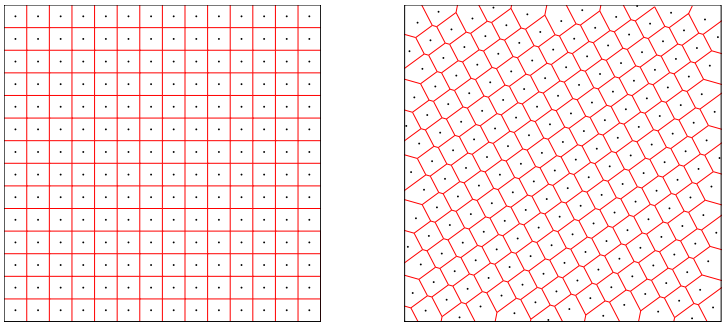
$$\delta(P) := \sup_{x \in X} \{ \min_{p \in P} \{ \rho(x, p) \} \}. \tag{4.1}$$

The figure gives an interpretation of the definition for two different metrics. An alternative way to consider dispersion is as the radius of the largest empty ball (for the L_∞ metric, the balls are actually cubes). Note that at the boundary of X (if it exists), the empty ball becomes truncated because it cannot exceed the boundary.

Optimizing dispersion forces the points to be distributed more uniformly over $\mathcal{C}$. This causes them to fail statistical tests, but the point distribution is often better for motion planning purposes. Consider the best way to reduce dispersion if ρ is the L_∞ metric and $X = [0, 1]^n$. Suppose that the number of samples, k, is given. Optimal dispersion is obtained by partitioning $[0, 1]$ into a grid of cubes and placing a point at the center of each cube, as shown for $n = 2$ and $k = 196$ in Figure. The number of cubes per axis must be $\lfloor k^{1/n} \rfloor$, in which $\lfloor \cdot \rfloor$ denotes the *floor function*. If $k^{1/n}$ is not an integer, then there are leftover points that may be placed anywhere without affecting the dispersion. Notice that $k^{1/n}$ just gives the number of points per axis for a grid of k points in n dimensions. The resulting grid will be referred to as a *Sukharev grid*.

The dispersion obtained by the Sukharev grid is the best possible. Therefore, a useful lower bound can be given for any set P of k samples:

$$\delta(P) \geq \frac{1}{2\lfloor k^{1/n} \rfloor}. \tag{4.2}$$



(a) 196-point Sukharev grid (b) 196 lattice points

Figure 4.3: The Sukharev grid and a nongrid lattice.

This implies that keeping the dispersion fixed requires exponentially many points in the dimension, n.

The reason why we decided to optimize the L_∞ dispersion instead of the more natural L_2 is that the latter is extremely difficult to optimize (except in $\mathbb{R}^2$, where a tiling of equilateral triangles can be made, with a point in the center of each one). Even the simple problem of determining the best way to distribute a fixed number of points in $[0,1]^3$ is unsolved for most values of k.

Suppose now that other topologies are considered instead of $[0,1]^n$. Let $X = [0,1]/\sim$, in which the identification produces a torus. The situation is quite different because X no longer has a boundary. The Sukharev grid still produces optimal dispersion, but it can also be shifted without increasing the dispersion. In this case, a standard grid may also be used, which has the same number of points as the Sukharev grid but is translated to the origin. Thus, the first grid point is $(0,0)$, which is actually the same as $2^n - 1$ other points by identification. If X represents a cylinder and the number of points, k, is given, then it is best to just use the Sukharev grid. It is possible, however, to shift each coordinate that behaves like $\mathbb{S}^1$. If X is rectangular but not a square, a good grid can still be made by tiling the space with cubes. In some cases this will produce optimal dispersion. For complicated spaces such as $SO(3)$, no grid exists in the sense defined so far. It is possible, however, to generate grids on the faces of an inscribed Platonic solid and lift the samples to $\mathbb{S}^n$ with relatively little distortion.

4.2 Incremental Sampling and Searching

The algorithms in this section follow the *single-query model*, which means (q_I, q_G) is given only once per robot and obstacle set. This means that there are no advantages to pre-computation, and the sampling-based motion planning problem can be considered as a kind of search. The multiple-query model, which favors pre-computation, is covered in Section 5.6.

The sampling-based planning algorithms presented in the present section are conceptually similar to the family of search algorithms summarized in Chapter 2. The main difference lies in step 3 below, in which applying an action, u, is replaced by generating a path segment, τ_s. Another difference is that the search graph, $\mathcal{G}$, is undirected, with edges

that represent paths, as opposed to a directed graph in which edges represent actions. Most single-query, sampling-based planning algorithms follow this template:

Initialization: Let $\mathcal{G}(V, E)$ represent an undirected search graph, for which V contains at least one vertex and E contains no edges. Typically, V contains q_I, q_G, or both. In general, other points in $\mathcal{C}_{\text{free}}$ may be included.

Vertex Selection Method (VSM): Choose a vertex $q_{\text{cur}} \in V$ for expansion.

Local Planning Method (LPM): For some $q_{\text{new}} \in \mathcal{C}_{\text{free}}$ that may or may not be represented by a vertex in V , attempt to construct a path $\tau_s : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$ such that $\tau(0) = q_{\text{cur}}$ and $\tau(1) = q_{\text{new}}$. τ_s must be checked with a collision detection method to ensure that it does not cause a collision. If this step fails to produce a collision-free path segment, then go to step 2.

Insert an Edge in the Graph: Insert τ_s into E, as an edge from q_{cur} to q_{new}. If q_{new} is not already in V, then it is inserted.

Check for a Solution: Determine whether $\mathcal{G}$ encodes a solution path. As in the discrete case, if there is a single search tree, then this is trivial; otherwise, it can become complicated and expensive.

Return to step 2: Iterate unless a solution has been found or some termination condition is satisfied, in which case the algorithm reports failure.

A large family of sampling-based algorithms can be described by varying the implementations of steps 2 and 3. Implementations of the other steps may also vary, but this is less important and will be described where appropriate. For convenience, step 2 will be called the vertex selection method (VSM) and step 3 will be called the *local planning method* (LPM). The role of the VSM is similar to that of the priority queue, Q, in Section 2.2.1. The role of the LPM is to compute a collision-free path segment that can be added to the graph. It is called local because the path segment is usually simple (e.g., the shortest path) and travels a short distance. It is not *global* in the sense that the LPM does not try to solve the entire planning problem; it is expected that the LPM may often fail to construct path segments. As in the case of discrete search algorithms, there are several classes of algorithms based on the number of search trees: unidirectional, bidirectional and multidirectional methods.

4.2.1 Adapting Discrete Search Algorithms

One of the most convenient and straightforward ways to make sampling-based planning algorithms is to define a grid over $\mathcal{C}$ and conduct a discrete search using the algorithms of Chapter 2.

Discretization: Assume that $\mathcal{C}$ is discretized by using the resolutions $k_1, k_2, ..., and k_n$, in which each k_i is a positive integer. This allows the resolution to be different for each C-space coordinate. Either a standard grid or a Sukharev grid can be used. Let

$$\Delta q_i = [0 \cdots 1/k_i 0 \cdots 0...0], \tag{4.3}$$

in which the first $i-1$ components and the last$n-i$ components are 0. A *grid point* is a configuration $q \in \mathcal{C}$ that can be expressed in the form

$$\sum_{i=1}^{n} j_i \Delta q_i, \tag{4.4}$$

in which each $j_i \in \{0, 1, ..., k_i\}$. The integers $j_1, ..., j_n$ can be imagined as array indices for the grid. Let the term *boundary grid point*refer to a grid point for which $j_i = 0$ or $j_i = k_i$ for some i.

Neighborhoods: For each grid point q we need to define the set of nearby grid points for which an edge may be constructed. Special care must be given to defining the neighborhood of a boundary grid point to ensure that identifications and the C-space boundary (if it exists) are respected. If q is not a boundary grid point, then the 1-*neighborhood* is defined as

$$N_1(q) = \{q \pm \Delta q_1, ..., q \pm \Delta q_n\}. \tag{4.5}$$

For an n-dimensional C-space there at most $2n$ 1-neighbors. In two dimensions, this yields at most four 1-neighbors, which can be thought of as up, down, left, and right. There are at most four because some directions may be blocked by the obstacle region. A 2-*neighborhood* is defined as

$$N_1(q) = \{q \pm \Delta q_i \pm \Delta q_j | 1 \leq i, j \leq n, i \neq j\} \cup N_1(q). \tag{4.6}$$

Similarly, a k-*neighborhood* can be defined for any positive integer $k \leq n$. For an n-neighborhood, there are at most 3^{n-1} neighbors; there may be fewer due to boundaries or collisions. The definitions can be easily extended to handle the boundary points.

Obtaining a discrete planning problem: Once the grid and neighborhoods have been defined, a discrete planning problem is obtained. Figure 4.4 depicts the process for a problem in which there are nine Sukharev grid points in $[0, 1]^2$. Using 1-neighborhoods, the potential edges in the search graph, $\mathcal{G}(V, E)$, appear in Figure 4.4a. If q_I and q_G do not coincide with grid points, they need to be connected to some nearby grid points, as shown in Figure 4.4b. Usually, all of the vertices and edges shown in Figure 4.4b do not appear in $\mathcal{G}$ because some intersect with $\mathcal{C}_{\text{obs}}$. Figure 4.4c shows a more typical situation, in which some of the potential vertices and edges are removed because of collisions. In this section, it is assumed that $\mathcal{G}$ is revealed during the search. This is the same situation that occurs for the discrete planning methods from Chapter 2. In the current setting, the potential edges of $\mathcal{G}$ are validated during the search. The candidate edges to evaluate are given by the definition of the k-neighborhoods. During the search, any edge or vertex that has been checked for collision explicitly appears in a data structure so that it does not need to be checked again. At the end of the search, a path is found, as depicted in Figure 4.4d.

4.3 Rapidly Exploring Dense Trees

This section introduces an incremental sampling and searching approach that yields good performance in practice without any parameter tuning. The idea is to incre-

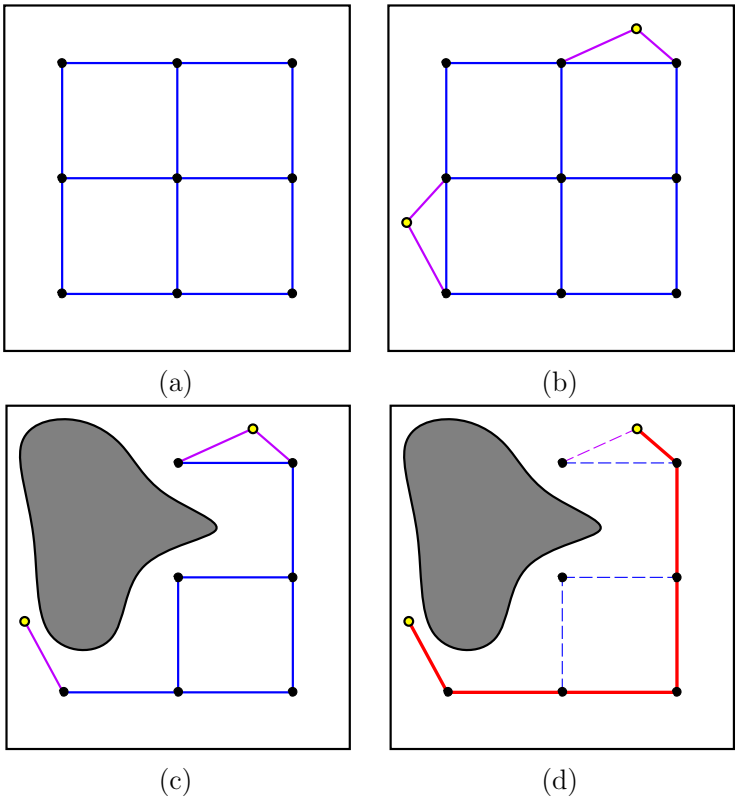


Figure 4.4: A topological graph can be constructed during the search and can successfully solve a motion planning problem using very few samples.

mentally construct a search tree that gradually improves the resolution but does not need to explicitly set any resolution parameters. In the limit, the tree densely covers the space. A dense sequence of samples is used as a guide in the incremental construction of the tree. If this sequence is random, the resulting tree is called a *rapidly exploring random tree (RRT)*. In general, this family of trees, whether the sequence is random or deterministic, will be referred to as *rapidly exploring dense trees (RDTs)* to indicate that a dense covering of the space is obtained.

4.3.1 The Exploration Algorithm

Before explaining how to use these trees to solve a planning query, imagine that the goal is to get as close as possible to every configuration, starting from an initial configuration. The method works for any dense sequence. Once again, let α denote an infinite, dense sequence of samples in $\mathcal{C}$. The ith sample is denoted by $\alpha(i)$. This may possibly include a uniform, random sequence, which is only dense with probability one. Random sequences that induce a nonuniform bias are also acceptable, as long as they are dense with probability one. An RDT is a topological graph, $\mathcal{G}(V, E)$. Let $S \subset \mathcal{C}_{\text{free}}$ indicate the set of all points reached by $\mathcal{G}$. Since each $e \in E$ is a path, this can be expressed as the swath, S, of the graph, which is defined as

$$S = \bigcup_{e \in E} e([0,1]), \tag{4.7}$$

where $e([0,1]) \subset \mathcal{C}_{\text{free}}$ is the image of the path e.

Algorithm 2 RDT

```
1 G.init(q_0)
2 for 1 ≤ i ≤ k do
3     G.addvertex(α(i))
4     q_n ← NEAREST(S(G), α(i))
5     G.addedge(q_n, α(i))
```

The exploration algorithm is first explained in Algorithm 2 without any obstacles or boundary obstructions. It is assumed that $\mathcal{C}$ is a metric space. Initially, a vertex is made at q_0. For k iterations, a tree is iteratively grown by connecting $\alpha(i)$ to its nearest point in the swath, S. The connection is usually made along the shortest possible path. In every iteration, $\alpha(i)$ becomes a vertex. Therefore, the resulting tree is dense.

Figures 4.5-4.6 illustrate an iteration graphically. Suppose the tree has three edges and four vertices, as shown in Figure 4.5a. If the nearest point, $q_n \in S$, to $\alpha(i)$ is a vertex, as shown in Figure 4.5b, then an edge is made from q_n to $\alpha(i)$. However, if the nearest point lies in the interior of an edge, as shown in Figure 4.6, then the existing edge is split so that q_n appears as a new vertex, and an edge is made from $q_n$to $\alpha(i)$. The edge splitting, if required, is assumed to be handled in line 4 by the method that adds edges. Note that the total number of edges may increase by one or two in each iteration.

Figure 4.7 shows an execution of Algorithm 2 for the case in which $\mathcal{C} = [0,1]^2$ and $q_0 = (1/2, 1/2)$. It exhibits a kind of *fractal behavior*. Several main branches are first

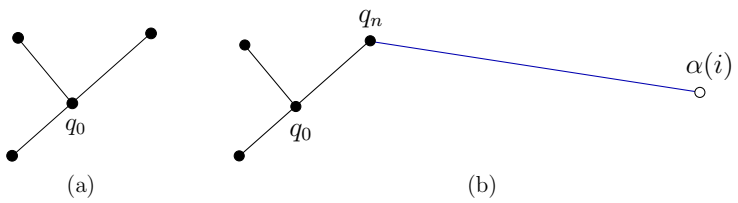


Figure 4.5: (a) Suppose inductively that this tree has been constructed so far using algorithm 2. (b) A new edge is added that connects from the sample $\alpha(i)$ to the nearest point in S, which is the vertex q_n.

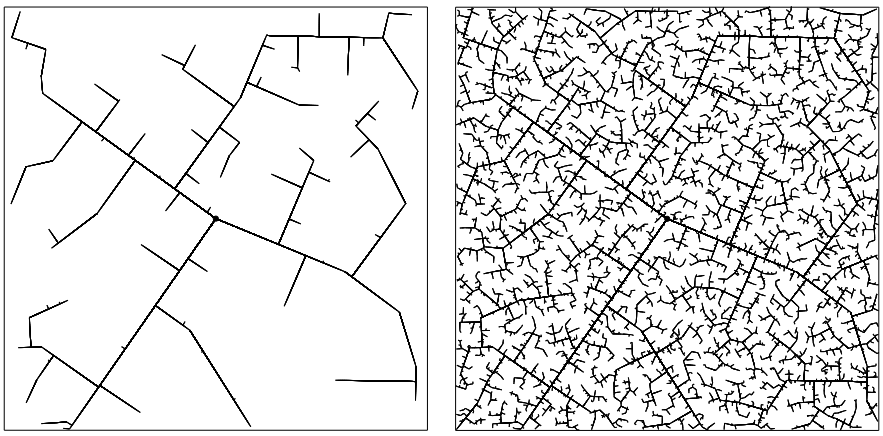


Figure 4.6: In the early iterations, the RRT quickly reaches the unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.

constructed as it rapidly reaches the far corners of the space. Gradually, more and more area is filled in by smaller branches. From the pictures, it is clear that in the limit, the tree densely fills the space. Thus, it can be seen that the tree gradually improves the resolution (or dispersion) as the iterations continue. This behavior turns out to be ideal for sampling-based motion planning.

Recall that in sampling-based motion planning, the obstacle region $\mathcal{C}_{\text{obs}}$ is not explicitly represented. Therefore, it must be taken into account in the construction of the tree. Below is indicated how to modify Algorithm 2 so that collision checking is taken into account. The modified algorithm appears in Figure 5.8.

The procedure stopping-configuration yields the nearest configuration possible to the boundary of $\mathcal{C}_{\text{free}}$, along the direction toward $\alpha(i)$. The nearest point $q_n \in S$ is defined to be same (obstacles are ignored); however, the new edge might not reach to $\alpha(i)$. In this case, an edge is made from q_n to q_s, the last point possible before hitting the obstacle. The minimum distance allowed from the obstacle depends on the implementation.

Algorithm 3 RDT

$\mathcal{G}.init(q_0)$
for $1 \leq i \leq k$ **do**
  $q_n \leftarrow NEAREST(S, \alpha(i))$
  $q_s \leftarrow STOPPING-CONFIGURATION(q_n, \alpha(i))$
  if $q_s \neq q_n$ **then**
    $\mathcal{G}.addvertex(q_s)$
    $\mathcal{G}.addedge(q_n, q_s)$

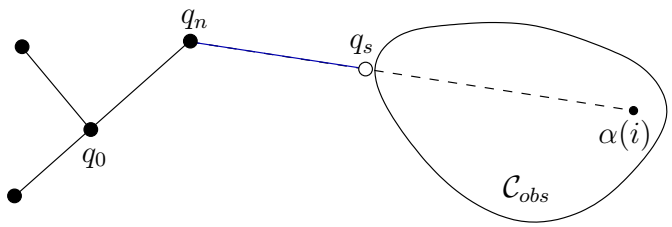


Figure 4.7: (If there is an obstacle, the edge travels up to the obstacle boundary, as far as allowed by the collision detection algorithm.

4.4 Roadmap Methods for Multiple Queries

Previously, it was assumed that a single initial-goal pair was given to the planning algorithm. Suppose now that numerous initial-goal queries will be given to the algorithm, while keeping the robot model and obstacles fixed. This leads to a *multiple-query* version of the motion planning problem. In this case, it makes sense to invest substantial time to preprocess the models so that future queries can be answered efficiently. The goal is to construct a topological graph called a roadmap, which efficiently solves multiple initial-goal queries. Intuitively, the paths on the roadmap should be easy to reach from each of q_I and q_G, and the graph can be quickly searched for a solution. The general framework presented here was initially introduced under the name *probabilistic roadmaps (PRMs)*. The probabilistic aspect, however, is not important to the method. Therefore, we refer to this family of methods as *sampling-based roadmaps*, with distinction from the *combinatorial roadmaps*.

4.4.1 The Basic Method

Let $\mathcal{G}(V, E)$ represent a topological graph in which V is a set of vertices and E is the set of paths that map into $\mathcal{C}_{\text{free}}$. Under the multiple-query philosophy, motion planning is divided into two phases of computation:

Preprocessing Phase: During the preprocessing phase, substantial effort is invested to build $\mathcal{G}$ in a way that is useful for quickly answering future queries. For this reason, it is called a *roadmap*, which in some sense should be accessible from every part of $\mathcal{C}_{\text{free}}$.

Query Phase: During the query phase, a pair, q_I and q_G, is given. Each configuration must be connected easily to $\mathcal{G}$ using a local planner. Following this, a discrete search

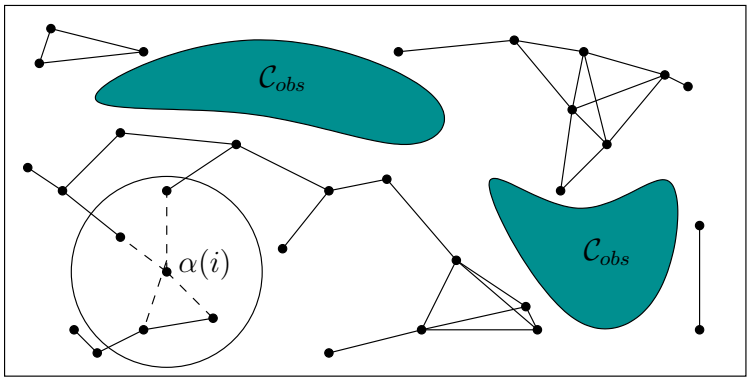


Figure 4.8: The sampling-based roadmap is constructed incrementally by attempting to connect each new sample, $\alpha(i)$, to nearby vertices in the roadmap.

is performed using any of the algorithms in Chapter 2 to obtain a sequence of edges that forms a path from q_I to q_G.

Generic preprocessing phase

Algorithm 3 presents an outline of the basic preprocessing phase, and Figure4.7 illustrates the algorithm. The algorithm utilizes a uniform, dense sequence α. In each iteration, the algorithm must check whether $\alpha(i) \in \mathcal{C}_{\text{free}}$. If $\alpha(i) \in \mathcal{C}_{\text{obs}}$, then it must continue to iterate until a collision-free sample is obtained. Once $\alpha(i) \in \mathcal{C}_{\text{free}}$, then in line 4 it is inserted as a vertex of $\mathcal{G}$. The next step is to try to connect $\alpha(i)$ to some nearby vertices, q, of $\mathcal{G}$. Each connection is attempted by the connect function, which is a typical LPM (local planning method). In most implementations, this simply tests the shortest path between $\alpha(i)$ and q.

Algorithm 4 BUILD ROADMAP

```
G.init(); i ← 0
while i < N do
    if α(i) ∈ C_free then
        G.addvertex(α(i)); i ← i + 1
        for all q ∈ NEIGHBORHOOD(α(i), G) do
            if (( not G.samecomponent(α(i), q)) and CONNECT(α(i), q)) then
                G.addedge(α(i), q)
```

The same component condition in line 6 checks to make sure $\alpha(i)$ and q are in different components of $\mathcal{G}$ before wasting time on collision checking. This ensures that every time a connection is made, the number of connected components of $\mathcal{G}$ is decreased.

Selecting neighboring samples

Several possible implementations of line 5 can be made. In all of these, it seems best to sort the vertices that will be considered for connection in order of increasing distance from $\alpha(i)$. This makes sense because shorter paths are usually less costly to check for collision, and

they also have a higher likelihood of being collision-free. If a connection is made, this avoids costly collision checking of longer paths to configurations that would eventually belong to the same connected component. Several useful implementations of NEIGHBORHOOD are

Nearest K: The K closest points to $\alpha(i)$ are considered. This requires setting the parameter K (a typical value is 15).

Component K: Try to obtain up to K nearest samples from each connected component of $\mathcal{G}$. A reasonable value is $K = 1$; otherwise, too many connections would be tried.

Radius: Take all points within a ball of radius r centered at $\alpha(i)$. An upper limit, K, may be set to prevent too many connections from being attempted.

Query phase

In the query phase, it is assumed that $\mathcal{G}$ is sufficiently complete to answer many queries, each of which gives an initial configuration, q_I , and a goal configuration, q_G. First, the query phase pretends as if q_I and q_G were chosen from α for connection to $\mathcal{G}$. This requires running two more iterations of the algorithm in algorithm 3. If q_I and q_G are successfully connected to other vertices in $\mathcal{G}$, then a search is performed for a path that connects the vertex q_I to the vertex q_G. The path in the graph corresponds directly to a path in $\mathcal{C}_{\text{free}}$, which is a solution to the query. Unfortunately, if this method fails, it cannot be determined conclusively whether a solution exists. If the dispersion is known for a sample sequence, α, then it is at least possible to conclude that no solution exists for the resolution of the planner. In other words, if a solution does exist, it would require the path to travel through a corridor no wider than the radius of the largest empty ball.

Chapter 5

Sampling-Based Planning Under Differential Constraints

In the models and methods presented in Chapter 4, it was assumed that a path can be easily determined between any two configurations in the absence of obstacles. For example, the sampling-based roadmap approach assumed that two nearby configurations could be connected by a straight line in the configuration space. The constraints on the path are *global* in the sense that the restrictions are on the set of allowable configurations.

In this chapter *differential constraints* are introduced, which restrict the allowable velocities at each point. These can be considered as *local constraints*, in contrast to the *global constraints* that arise due to obstacles. In robotics, most problems involve differential constraints that arise from the kinematics and dynamics of a robot.

A possible approach in two steps is to ignore differential constraints in the planning process and then smoothing the resulting path until it satisfies the constraints . If it is practical, a better approach is to consider differential constraints in the planning process. This yields plans that directly comply with the natural motions of a mechanical system. Differential models are generally expressed as $\dot{x} = f(x, u)$, which is the continuous-time counterpart of the state transition equation, $x_{k+1} = f(x_k, u_k)$.

The main topic of this Chapter is extending the incremental sampling and searching framework of Chapter 5 to kino-dynamic motion planning to develop resolution-complete algorithms.This is complicated by the discretization of three spaces (state space, action space, and time), whereas in Chapter 5 resolution completeness only involved discretization of the C-space. The focus is limited to sampling-based approaches because very little can be done with combinatorial methods if differential constraints exist.

5.1 Differential Models

This section provides a continuous-time counterpart to the state transition equation, $x_{k+1} = f(x_k, u_k)$, presented in Chapter 2. On a continuous state space, X (assumed to be a smooth manifold), it will be defined as $\dot{x} = f(x, u)$, which intentionally looks similar to the discrete version. It will still be referred to as a state transition equation. It will also be called a system (short for control system), which is a term used in control theory. In continuous time, the state transition function $f(x, u)$ yields a velocity as opposed to the next state. Since the transitions are no longer discrete, it does not make sense to talk about a "next"

state. Future states that satisfy the differential constraints are obtained by integration of the velocity. Therefore, it is natural to specify only velocities. This relies on the notions of *tangent spaces of a vector fields*.

5.1.1 Velocity Constraints on the Configuration Space

There are two general ways to represent differential constraints: parametric and implicit. The intuitive difference is that implicit representations express velocities that are prohibited, whereas parametric representations directly express the velocities that are allowed.

Implicit representation

Assume that the C-space $\mathcal{C}$ is a smooth manifold. Now consider placing velocity constraints on $\mathcal{C}$. In general, constraints expressed in the form shown in (5.1) are called implicit.

$$g(q, \dot{q}) \bowtie 0, \tag{5.1}$$

where $\bowtie$ could be ant one of $=, \ , >, \leq, \geq$. Generally, it can be very complicated to obtain a parametric representation of the solutions of implicit equations.

Parametric constraint

The parametric way of expressing velocity constraints gives a different interpretation to $U(q)$. Rather than directly corresponding to a velocity, each $u \in U(q)$ is interpreted as an abstract action vector. The set of allowable velocities is then obtained through a function that maps an action vector into $\mathcal{T}_q(\mathcal{C})$. This yields the configuration transition equation (or system)

$$\dot{q} = f(q, u), \tag{5.2}$$

in which f is a continuous-time version of the state transition function that was developed in Chapter 2. There are two interesting ways to interpret (5.2):

Subspace of the tangent space: If q is fixed, then f maps from U into $\mathcal{T}_q(\mathcal{C})$. This parameterizes the set of allowable velocities at q because a velocity vector, $f(q, u)$, is obtained for every $u \in U(q)$.

Vector field: If u is fixed, then f can be considered as a function that maps each $q \in \mathcal{C}$ into $\mathcal{T}_q(\mathcal{C})$. This means that f defines a vector field over $\mathcal{C}$ for every fixed $u \in U$.

5.1.2 A example of velocity constraint: an airplane

A simple aircraft flight model that may be obtained as follows. Suppose that the aircraft is flying with a constant speed v. A configuration is represented as $q = (x, y, z, \psi)$ where x, y, z are the coordinate of the center of mass of the aircraft and ψ define the flight's direction. Let u_z denote an action that directly causes a change in the altitude: $\dot{z} = u_z$. The steering action u_ψ control ψ. The configuration transition equation is

$$\begin{cases} \dot{x} = v \cos \psi \\ \dot{y} = v \sin \psi \\ \dot{z} = u_z \\ \dot{\psi} = u_\phi \end{cases} \tag{5.3}$$

5.2 Phase Space Representation of Dynamical Systems

The differential constraints defined in 5.1 are often called *kinematic* because they can be expressed in terms of velocities on the C-space. This formulation is useful for many problems, such as modeling the possible directions of motions for a wheeled mobile robot. It does not, however, enable dynamics to be expressed. To account for momentum and other aspects of dynamics, higher order differential equations are needed. There are usually constraints on acceleration $\ddot{q}$, which is defined as $d\dot{q}/dt$. The models for dynamics therefore involve acceleration $\ddot{q}$ in addition to velocity $\dot{q}$ and configuration q. Once again, both implicit and parametric models exist. For an implicit model, the constraints are expressed as

$$g_i(\ddot{q}.\dot{q}, q) = 0. \tag{5.4}$$

For a parametric model, they are expressed as

$$\ddot{q} = f(\dot{q}, q, u). \tag{5.5}$$

To deal with higher order derivatives it is useful the introduction of a *phase space*, which has more dimensions than the original C-space. Suppose that q represents a configuration, expressed using a coordinate neighborhood on a smooth n-dimensional manifold $\mathcal{C}$. Second-order constraints of the form can be expressed as first-order constraints in a $2n$-dimensional state space. Let x denote the $2n$-dimensional phase vector. The new state x is defined as follows: for each integer i such that $1 \leq i \leq n$ $x_i = q_i$, while for each i such that $n+1 \leq i \leq 2n$, $x_i = \dot{q}_{i-n}$. Suppose that a set of n differential equations is expressed in parametric form as $\ddot{q} = h(q, \dot{q}, u)$. In the phase space, there are $2n$ differential equations. The first n correspond to the phase space definition $\dot{x}_i = x_{n+i}$, for each i such that $1 \leq i \leq n$. These hold because $x_{n+i} = \dot{q}_i$ and x_i is the time derivative of $\dot{q}_i$ for $i \leq n$. The remaining n components of $\dot{x} = f(x, u)$ follow directly from h by substituting the first n components of x in the place of q and the remaining n in the place of $\dot{q}$ in the expression $h(q, \dot{q}, u)$. The result can be denoted as $h(x, u)$ (obtained directly from $h(q, \dot{q}, u)$). This yields the final n equations as $\dot{x}_i = h_{i-n}(x, u)$, for each i such that $n+1 \leq i \leq 2n$. These $2n$ equations define a phase (or state) transition equation of the form $\dot{x} = f(x, u)$. Now it is clear that constraints on acceleration can be manipulated into velocity constraints on the phase space. Constraints on higher order derivatives can be handled in a similar way.

Now that we have introduced differential constraint and phase spaces, we can in incorporate these concepts in motion planning, however it is important to point out that the physical details regarding how the dynamic systems have been modeled can be neglected, considering for any systems its state transition equation $\dot{x} = f(x, u)$ as an input of the problem.

5.3 Problem Formulation

Motion planning under differential constraints can be considered as a variant of classical *two-point boundary value problems (BVPs)*. In that setting, initial and goal states are given, and the task is to compute a path through a state space that connects initial and goal states while satisfying differential constraints. Motion planning involves the additional complication of avoiding obstacles in the state space. Techniques for solving BVPs are unfortunately not well-suited for motion planning because they are not designed for handling

obstacle regions. For some methods, adaptation may be possible; however, the obstacle constraints usually cause these classical methods to become inefficient or incomplete.

It is assumed that the differential constraints are expressed in a state transition equation, $\dot{x} = f(x, u)$, on a smooth manifold X, called the state space, which may be a C-space $\mathcal{C}$ or a phase space of a C-space. A solution path will not be directly expressed as in Chapter 2 and 4 but is instead derived from an action trajectory via integration of the state transition equation.

Let the action space U be a bounded subset of $\mathbb{R}^m$. A planning algorithm computes an action trajectory $\tilde{u}$, which is a function of the form $\tilde{u} : [0, \infty) \rightarrow U$. The action at a particular time t is expressed as $u(t)$. If the action space is state-dependent, then $u(t)$ must additionally satisfy $u(t) \in U(x(t)) \subset U$. It will also be assumed that a termination action u_T is used, which makes it possible to specify all action trajectories over $[0, \infty)$ with the understanding that at some time t_F , the termination action is applied.

The connection between the action and state trajectories needs to be formulated. Starting from some initial state $x(0)$ at time $t = 0$, a state trajectory is derived from an action trajectory $\tilde{u}$ as

$$x(t) = x(0) + \int_0^t f(x(t'), u(t'))dt'. \tag{5.6}$$

The problem of motion planning under differential constraints can be formulated as an extension of the Piano Mover's Problem in Formulation 4. The main differences in this extension are 1) the introduction of time, 2) the state or phase space, and 3) the state transition equation. The resulting formulation follows.

Formulation 5. ***(Motion Planning under Differential Constraints)***

1. *A world $\mathcal{W}$, a robot $\mathcal{A}$, an obstacle region $\mathcal{O}$, and a configuration space $\mathcal{C}$, which are defined the same as in Formulation 4.*

2. *An unbounded time interval $T = [0, \infty)$.*

3. *A smooth manifold X, called the state space, which may be $X = \mathcal{C}$ or it may be a phase space derived from $\mathcal{C}$ if dynamics is considered. Let $\kappa : X \rightarrow \mathcal{C}$ denote a function that returns the configuration $q \in \mathcal{C}$ associated with $x \in X$. Hence, $q = \kappa(x)$.*

4. *An obstacle region X_{obs} is defined for the state space. If $X = \mathcal{C}$, then $X_{obs} = \mathcal{C}_{obs}$. For general phase spaces, X_{obs} is described in detail later. The notation $X_free = X \setminus X_{obs}$ indicates the states that avoid collision and satisfy any additional global constraints.*

5. *For each state $x \in X$, a bounded action space $U(x) \subset \mathbb{R}^m \cup u_T$, which includes a termination action u_T and m is some fixed integer called the* number of action variables. *Let U denote the union of $U(x)$ over all $x \in X$.*

6. *A system is specified using a state transition equation $\dot{x} = f(x, u)$, defined for every $x \in X$ and $u \in U(x)$. If the termination action is applied, it is assumed that $f(x, u_T) = 0$ (and no cost accumulates, if a cost functional is used).*

7. *A state $x_I \in X_{free}$ is designated as the initial state.*

8. *A set $X_G \subset X_{free}$ is designated as the goal region.*

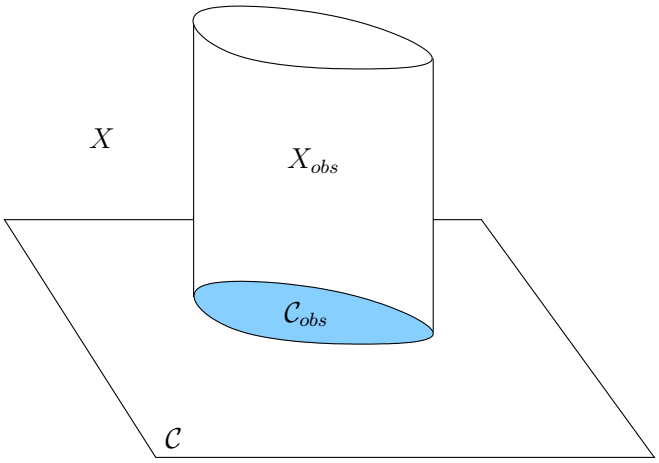


Figure 5.1: An obstacle region $\mathcal{C}_{\text{obs}} \subset \mathcal{C}$ generates a cylindrical obstacle region $X_{\text{obs}} \subset X$ with respect to the phase variables.

9. *A complete algorithm must compute an action trajectory $\tilde{u} : T \to U$, for which the state trajectory $\tilde{x}$, resulting from integration, satisfies: 1)$x(0) = x_I$, and 2) there exists some $t > 0$ for which $u(t) = u_T$ and $x(t) \in X_G$.*

Additional constraints may be placed on $\tilde{u}$, such as continuity or smoothness over time. At the very least, $\tilde{u}$ must be chosen so that the integrand of (5.5) is integrable over time. Let $\mathcal{U}$ denote the set of all permissible action trajectories over $T = [0, \infty)$. By default, $\mathcal{U}$ is assumed to include any integrable action trajectory.

5.3.1 Obstacles in the Phase Space

In Formulation 5, the specification of the obstacle region in Item 4 was intentionally left ambiguous. Now it will be specified in more detail. Any state for which its associated configuration lies in $\mathcal{C}_{\text{obs}}$ must also be a member of X_{obs}. The velocity is irrelevant if a collision occurs in the world $\mathcal{W}$. In most cases that involve a phase space, the obstacle region X_{obs} is therefore defined as

$$X_{\text{obs}} = \{x \in X | \kappa(x) \in \mathcal{C}_{\text{obs}}\}, \tag{5.7}$$

in which $\kappa(x)$ is the configuration associated with the state $x \in X$. If the first n variables of X are configuration parameters, then X_{obs} has the cylindrical structure shown in Figure 14.1 with respect to the other variables. If κ is a complicated mapping, as opposed to simply selecting the configuration coordinates, then the structure might not appear cylindrical. In these cases, (5.6) still indicates the correct obstacle region in X.

5.3.2 The Region of Inevitable Collision

One of the most challenging aspects of planning can be visualized in terms of the region of inevitable collision, denoted by X_{ric}. This is the set of states from which entry into X_{obs} will eventually occur, regardless of any actions that are applied. As a simple example, imagine that a robotic vehicle is traveling 100 km/hr toward a large wall and is only 2 meters away.

Clearly the robot is doomed. Due to momentum, collision will occur regardless of any efforts to stop or turn the vehicle. At low enough speeds, X_{ric} and X_{obs} are approximately the same; however, X_{ric} grows dramatically as the speed increases.

Let $\mathcal{U}_\infty$ denote the set of all trajectories $\tilde{u} : [0, \infty) \rightarrow U$ for which the termination action u_T is never applied (we do not want inevitable collision to be avoided by simply applying u_T). The region of inevitable collision is defined as

$$X_{\text{ric}} = \{x(0) \in X | \forall \tilde{u} \in \mathcal{U}_\infty, \quad \exists t > 0 \text{ such that } x(t) \in X_{\text{obs}}\}, \tag{5.8}$$

in which $x(t)$ is the state at time t obtained by applying (5.5) from $x(0)$.

In sampling-based planning under differential constraints, X_{ric} is not computed because it is too complicated. It is not even known how to make a "collision detector" for X_{ric}. By working instead with X_{obs}, challenges arise due to momentum. There may be large parts of the state space that are never worth exploring because they lie in X_{ric}. Unfortunately, there is no practical way at present to accurately determine whether states lie in X_{ric}. As the momentum and amount of clutter increase, this becomes increasingly problematic.

5.4 Reachability and Completeness

In Chapter 4, sampling over $\mathcal{C}$ was of fundamental importance. The most important consideration was that a sequence of samples should be dense so that samples get arbitrarily close to any point in $\mathcal{C}_{\text{free}}$. Planning under differential constraints is complicated by the specification of solutions by an action trajectory instead of a path through X_{free}. For sampling-based algorithms to be resolution complete, sampling and searching performed on the space of action trajectories must somehow lead to a dense set in X_{free}.

5.4.1 Reachable Sets

For the algorithms in Chapter 4, resolution completeness and probabilistic completeness rely on having a sampling sequence that is dense on $\mathcal{C}$. In the present setting, this would require dense sampling on X. Differential constraints, however, substantially complicate the sampling process. It is generally not reasonable to prescribe precise samples in X that must be reached because reaching them may be impossible or require solving a BVP. Since paths in X are obtained indirectly via action trajectories, completeness analysis begins with considering which points can be reached by integrating action trajectories. Assume temporarily that there are no obstacles: $X_{\text{free}} = X$. Let $\mathcal{U}$ be the set of all permissible action trajectories on the time interval $[0, \infty)$. From each $\tilde{u} \in \mathcal{U}$, a state trajectory $\tilde{x}(x_0, \tilde{u})$ is defined using (5.5). Which states in X are visited by these trajectories? It may be possible that all of X is visited, but in general some states may not be reachable due to differential constraints. Let $R(x_0, \mathcal{U}) \subset X$ denote the reachable set from x_0, which is the set of all states that are visited by any trajectories that start at x_0 and are obtained from some $\tilde{u} \in \mathcal{U}$ by integration. This can be expressed formally as

$$R(x_0, \mathcal{U}) = \{x_1 \in X | \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t \in [0, \infty) \text{ such that } x(t) = x_1\} \tag{5.9}$$

in which $x(t)$ is given by (5.5) and requires that $x(0) = x_0$.

So far the obstacle region has not been considered. Let $\mathcal{U}_{\text{free}} \subset U$ denote the set of all action trajectories that produce state trajectories that map into X_{free}. In other words,

$\mathcal{U}_{\text{free}}$ is obtained by removing from $\mathcal{U}$ all action trajectories that cause entry into X_{obs} for some $t > 0$. The reachable set that takes the obstacle region into account is denoted $R(x_0, \mathcal{U}_{\text{free}})$, which replaces $\mathcal{U}$ by $\mathcal{U}_{\text{free}}$ in (5.7). This assumes that for the trajectories in $\mathcal{U}_{\text{free}}$, the termination action can be applied to avoid inevitable collisions due to momentum. A smaller reachable set could have been defined that eliminates trajectories for which collision inevitably occurs without applying u_T . The completeness of an algorithm can be expressed in terms of reachable sets. For any given pair $x_I, x_G \in X_{\text{free}}$, a complete algorithm must report a solution action trajectory if $x_G \in R(x_I, \mathcal{U}_{\text{free}})$, or report failure otherwise. Completeness is too difficult to achieve, except for very limited cases; therefore, sampling- based notions of completeness are more valuable.

Time-limited reachable set

Consider the set of all states that can be reached up to some fixed time limit. Let the time-limited reachable set$R(x_0, \mathcal{U}, t)$ be the subset of $R(x_0, \mathcal{U})$ that is reached up to and including time t. Formally, this is

$$R(x_0, \mathcal{U}, t) = \{x_1 \in X | \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t' \in [0, t] \text{ such that } x(t') = x_1\}. \tag{5.10}$$

Backward reachable sets

The reachability definitions have a nice symmetry with respect to time. Rather than describing all points reachable from some $x \in X$, it is just as easy to describe all points from which some $x \in X$ can be reached.

Let the *backward reachable* set be defined as

$$B(x_f, \mathcal{U}) = \{x_0 \in X | \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t \in [0, \infty) \text{ such that } x(t) = x_f\}, \tag{5.11}$$

in which $x(t)$ is given by (5.5) and requires that $x(0) = x_0$. The *time-limited backward reachable set* is defined as

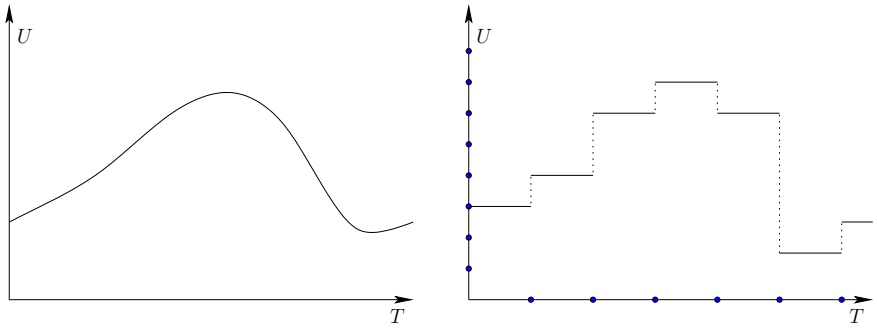
$$B(x_f, \mathcal{U}, t) = \{x_0 \in X | \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t \in [0, t] \text{ such that } x(t') = x_f\}, \tag{5.12}$$

which once again requires that $x(0) = x_0$ in (5.5). At this point, there appear to be close parallels between forward, backward, and bidirectional searches from Chapter 2. The same possibilities exist in sampling- based planning under differential constraints. The forward and backward reachable sets indicate the possible states that can be reached under such schemes. The algorithms explore subsets of these reachable sets.

5.5 The Discrete-Time Model

Under differential constraints, sampling-based motion planning algorithms all work by sampling the space of action trajectories. This results in a reduced set of possible action trajectories. To ensure some form of completeness, a motion planning algorithm should carefully construct and refine the sample set. As in Chapter 4, the qualities of a sample set can be expressed in terms of dispersion and denseness. The main difference in the current setting is that the algorithms here work with a sample sequence over $\mathcal{U}$, as opposed to over $\mathcal{C}$ as in Chapter 4. This is required because solution paths can no longer be expressed directly on $\mathcal{C}$ (or X).

The *discrete-time model* is depicted in Figure 5.3 and is characterized by three aspects:



A trajectory in $\mathcal{U}$ A trajectory in $\mathcal{U}_d$

Figure 5.2: The discrete-time model results in $\mathcal{U}_d \subset \mathcal{U}$, which is obtained by partitioning time into regular intervals and applying a constant action over each interval. The action is chosen from a finite subset U_d of U.

1. Time T is partitioned into intervals of length Δt. This enables stages to be assigned, in which stage k indicates that $(k-1)\Delta t$ units of time have elapsed.
2. A finite subset U_d of the action space U is chosen. If U is already finite, then this selection may be $U_d = U$.
3. The action $\mathrm{u}(t) \in U_d$ must remain constant over each time interval.

The first two discretize time and the action spaces. The third condition is needed to relate the time discretization to the space of action trajectories. Let $\mathcal{U}_d$ denote the set of all action trajectories allowed under a given time discretization. Note that $\mathcal{U}_d$ completely specifies the discrete-time model.

Any action trajectory in $\mathcal{U}_\lceil$ can be conveniently expressed as an action sequence $(u_1, u_2, ..., u_k)$, in which each $u_i \in U_d$ gives the action to apply from time $(i-1)\Delta t$ to time $i\Delta t$. After stage k, it is assumed that the termination action is applied.

5.5.1 Reachability Graph

After time discretization has been performed, the reachable set can be adapted to $\mathcal{U}_d$ to obtain $R(x_0, \mathcal{U}_d)$. An interesting question is: What is the effect of sampling on the reachable set? In other words, how do $R(x_0, \mathcal{U})$ and $R(x_0, \mathcal{U}_d)$ differ? This can be addressed by defining a reachability graph, which will be revealed incrementally by a planning algorithm.

Let $T_r(x_0, \mathcal{U}_d)$ denote a *reachability tree*, which encodes the set of all trajectories from x_0 that can be obtained by applying trajectories in $\mathcal{U}_\lceil$. Each vertex of $T_r(x_0, \mathcal{U}_d)$ is a reachable state, $x \in R(x_0, \mathcal{U}_d)$. Each edge of $T_r(x_0, \mathcal{U}_d)$ is directed; its source represents a starting state, and its destination represents the state obtained by applying a constant action $u \in \mathcal{U}_d$ over time Δt. Each edge e represents an action trajectory segment, $e : [0, \Delta t] \rightarrow U$. This can be transformed into a state trajectory, $\tilde{x}_e$, via integration, from 0 to Δt of $f(x, u)$ from the source state of e.

Thus, in terms of $\tilde{x}_e$, T_r can be considered as a topological graph in X (T_r will be used as an abbreviation of $T_r(x_0, \mathcal{U}_d)$). The *swath* $S(T_r)$ of T_r is

$$S(T_r) = \bigcup_{e \in E} \bigcup_{t \in [0, \Delta t]} x_e(t), \tag{5.13}$$

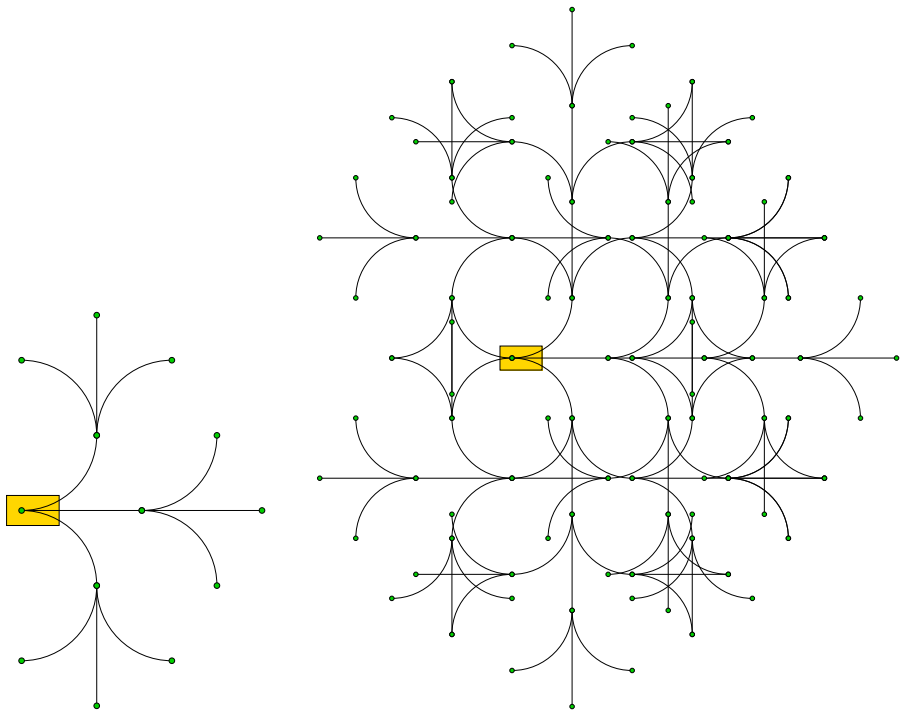


Figure 5.3: A reachability tree for the *Dubins car* with three actions. The kth stage produces 3^k new vertices.

in which $x_e(t)$ denotes the state obtained at time t from edge e.

5.5.2 Resolution Completeness

Beyond the trivial case of $\dot{x} = u$, the reachability graph is usually not a simple grid. Even if X is bounded, the reachability graph may have an infinite number of vertices, even though Δt is fixed and $\mathcal{U}_d$ is finite.

Suppose that $\dot{x} = f(x, u)$ and the discrete-time model is used. To ensure convergence of the discrete-time approximation, f must be well-behaved. This can be established by requiring that all of the derivatives of f with respect to u and x are bounded above and below by a constant. More generally, f is assumed to be Lipschitz, which is an equivalent condition for cases in which the derivatives exist, but it also applies at points that are not differentiable. If U is finite, then the Lipschitz condition is that there exists some $L \in (0, \infty)$ such that

$$||f(x, u) - f'(x', u)|| \leq L||x - x'||, \tag{5.14}$$

for all $x, x' \in X$,for all $u \in U$,and $|| \cdot ||$ denotes a norm on X. If U is infinite, then the condition is that there must exist some $L \in\in (0, \infty)$ such that

$$||f(x, u) - f'(x', u)|| \leq L(||x - x'|| + ||u - u'||) \tag{5.15}$$

for all $x, x' \in X$,and for all $u, u' \in U$. Intuitively, the Lipschitz condition indicates that if x and u are approximated by $\dot{x}$ and $\dot{u}$, then the error when substituted into f will

be manageable. If convergence to optimal trajectories with respect to a cost functional is important, then Lipschitz conditions are also needed for $l(x, u)$. Under such mild assumptions, if Δt and the dispersion of samples of U_d is driven down to zero, then the trajectories obtained from integrating discrete action sequences come arbitrarily close to solution trajectories. In other words, action sequences provide arbitrarily close approximations to any$\tilde{u} \in \mathcal{U}$. If f is Lipschitz, then the integration of (5.6) yields approximately the same result for $\tilde{u}$ as the approximating action sequence.

In the limit as Δt and the dispersion of U_d approach zero, the reachability graph becomes dense in the reachable set $R(x_I, \mathcal{U})$. Ensuring a systematic search for the case of a grid was not difficult because there is only a finite number of vertices at each resolution. Unfortunately, the reachability graph may generally have a countably infinite number of vertices for some fixed discrete-time model, even if X is bounded.

5.6 Sampling-Based Motion Planning Revisited

Now that the preliminary concepts have been defined for motion planning under differential constraints, the focus shifts to extending the sampling-based planning methods of Chapter 4. This primarily involves extending the incremental sampling and searching framework from Section 4.2 to incorporate differential constraints. If an efficient BVP solver is available, then it may also be possible to extend sampling-based roadmaps of Section 4.4 to handle differential constraints.

5.6.1 Basic Components

Sampling theory

There are at least two continuous spaces: X, and the time interval T. In most cases, the action space U is also continuous. Each continuous space must be sampled in some way. In the limit, it is important that any sample sequence is dense in the space on which sampling occurs. This was required for the resolution completeness concepts of Section 5.5.

Sampling of T and U can be performed by directly using the random or deterministic methods of Chapter 4. Time is just an interval of $\mathbb{R}$, and U is typically expressed as a convex m-dimensional subset of $\mathbb{R}^m$.

Some planning methods may require sampling on X. The definitions of discrepancy and dispersion from Chaptern 4 can be easily adapted to any measure space and metric space, respectively. Even though it may be straightforward to define a good criterion, generating samples that optimize the criterion may be difficult or impossible.

Collision detection

As in Chapter 4, efficient collision detection algorithms are a key enabler of sampling-based planning. If $X = \mathcal{C}$, then the methods of Chapter 4 directly apply. If X includes phase constraints, then additional tests must be performed.

In sintesys, determining whether $x \in X_{\text{free}}$ involves

1. Using a collision detection algorithm to ensure that $\kappa(x) \in \mathcal{C}_{\text{free}}$.

2. Checking x to ensure that other constraints of the form $h_i(x) \leq 0$ have been satisfied.

System simulator

new component is needed for sampling-based planning under differential constraints because of (5.6). Motions are now expressed in terms of an action trajectory, but collision detection and constraint satisfaction tests must be performed in X. Therefore, the system, $\dot{x} = f(x, u)$ needs to be integrated frequently during the planning process. Similar to the modeling of collision detection as a "black box", the integration process is modeled as a module called the system simulator.

Integration can be considered as a module that implements (5.6) by computing the state trajectory resulting from a given initial state $x(0)$, an action trajectory $\tilde{u}(t)$, and time t. The incremental simulator encapsulates the details of integrating the state transition equation so that they do not need to be addressed in the design of planners. However, that information from the particular state transition equation may still be important in the design of the planning algorithm.

5.6.2 Local Planning

The methods of Chapter 4 were based on the existence of a local planning method (LPM) that is simple and efficient. This represented an important part of both the incremental sampling and searching framework of Section 4.2 and the sampling- based roadmap framework of Section 4.4 In the absence of obstacles and differential constraints, it is trivial to define an LPM that connects two configurations. They can, for example, be connected using the shortest path (geodesic) in $\mathcal{C}$. The sampling-based roadmap approach from Section 4.4 relies on this simple LPM. n the presence of differential constraints, the problem of constructing an LPM that connects two configurations or states is considerably more challenging. Recall that this is the classical BVP, which is difficult to solve for most systems. There are two main alternatives to handle this difficulty in a sampling- based planning algorithm:

1. Design the sampling scheme, which may include careful selection of motion primitives, so that the BVP can be trivially solved.

2. Design the planning algorithm so that as few as possible BVPs need to be solved. The LPM in this case does not specify precise goal states that must be reached.

If the BVP is efficiently solved, then virtually any sampling-based planning algorithm from Chapter 4 can be adapted to the case of differential constraints. This is achieved by using the module in Figure 5.2 as the LPM. For example, a sampling-based roadmap can use the computed solution in the place of the shortest path through $\mathcal{C}$. If the BVP solver is not efficient enough, then this approach becomes impractical because it must typically be used thousands of times to build a roadmap. Under the second alternative, it is assumed that solving the BVP is very costly. The planning method in this case should avoid solving BVPs whenever possible. Some planning algorithms may only require an LPM that approximately reaches intermediate goal states, which is simpler for some systems. Other planning algorithms may not require the LPM to make any kind of connection. The LPM may return a motion primitive that appears to make some progress in the search but is not designed to connect to a prescribed state.

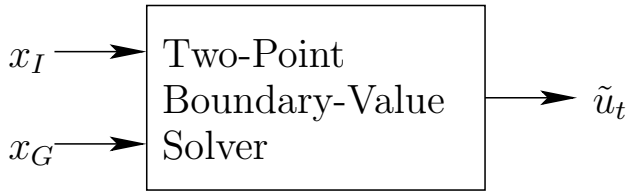


Figure 5.4: The BVP is treated as a black box that gives a control sequence as an output for any couple of start and end points.

5.6.3 General Framework Under Differential Constraints

Initialization: Let $\mathcal{G}(V, E)$ represent an undirected search graph, for which the vertex set V contains a vertex for x_I and possibly other states in X_{free}, and the edge set E is empty. The graph can be interpreted as a topological graph with a swath $\mathcal{S}(\mathcal{G})$.

Swath-point Selection Method (SSM): Choose a vertex $x_{\text{cur}} \in \mathcal{S}(\mathcal{G})$ for expansion.

Local Planning Method (LPM): Generate a motion primitive $\tilde{u}^p : [0, t_F] \rightarrow X_{\text{free}}$ such that $u(0) = x_{\text{cur}}$ and $u(t_F) = x_r$ for some $x_r \in X_{\text{free}}$, which may or may not be a vertex in $\mathcal{G}$. Using the system simulator, a collision detection algorithm, and by testing the phase constraints, $\tilde{u}^p$ must be verified to be violation-free. If this step fails, then go to Step 2.

Insert an Edge in the Graph: Insert $\tilde{u}^p$ into E. Upon integration, $\tilde{u}^p$ yields a state trajectory from x_{cur} to x_r. If x_r is not already in V , it is added. If x_{cur} lies in the interior of an edge trajectory for some $e \in E$, then e is split by the introduction of a new vertex at x_{cur}.

Check for a Solution: Determine whether $\mathcal{G}$ encodes a solution path. In some applications, a small gap in the state trajectory may be tolerated.

Return to Step 2: Iterate unless a solution has been found or some termination condition is satisfied. In the latter case, the algorithm reports failure.

The main new complication is due to BVPs. See Figure 5.3. Recall that for most systems it is important to reduce the number of BVPs that must be solved during planning as much as possible. Assume that connecting precisely to a prescribed state is difficult. Figure 5.3a shows the best situation, in which forward, unidirectional search is used to enter a large goal region. In this case, no BVPs need to be solved. As the goal region is reduced, the problem becomes more challenging. Figure 5.3b shows the limiting case in which X_G is a point $\{x_G\}$. This requires the planning algorithm to solve at least one BVP.

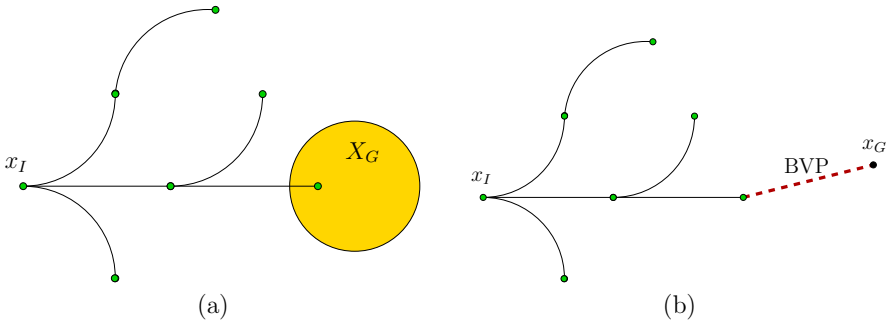


Figure 5.5: (a) Forward, unidirectional search for which the BVP is avoided. (b) Reaching the goal precisely causes a BVP.

5.7 RDT-Based Methods

RDTs were originally developed for handling differential constraints, even though most of their practical application has been to the Piano Mover's Problem. This section extends the ideas of Section 4.3 from $\mathcal{C}$ to X and incorporates differential constraints. Let α denote an infinite, dense sequence of samples in X. Let $\rho : X \times X \rightarrow [0, \infty]$ denote a distance function on X, which may or may not be a proper metric. The distance function may not be symmetric, in which case $\rho(x_1, x_2)$ represents the directed distance from x_1 to x_2.

The RDT is a search graph as considered so far in this section and can hence be interpreted as a subgraph of the reachability graph under some discretization model. For simplicity, first assume that the discrete-time model of Section 5.5 is used, which leads to a finite action set U_d and a fixed time interval Δt. The set $\mathcal{U}^p$ of motion primitives is all action trajectories for which some $u \in U_d$ is held constant from time 0 to Δt. The more general case will be handled at the end of this section. Paralleling Section 4.3, the RDT will first be defined in the absence of obstacles. Hence, let $X_{\text{free}} = X$. The construction algorithm is defined in Algorithm 5.

Algorithm 5 SIMPLE RDT WITH DIFFERENTIAL CONSTRAINTS

$\mathcal{G}.init(x_0)$
for $1 \leq i \leq k$ **do**
  $x_n \leftarrow NEAREST(S(\mathcal{G}), \alpha(i))$
  $(\tilde{u}^p, x_r) \leftarrow LOCALPLANNER(x_n, \alpha(i))$
  $\mathcal{G}.addvertex(x_r)$
  $\mathcal{G}.addedge(\tilde{u}^p)$

The RDT, denoted by $\mathcal{G}$, is initialized with a single vertex at some $x_0 \in X$. In each iteration, a new edge and vertex are added to $\mathcal{G}$. Line 3 uses ρ to choose x_n, which is the nearest point to $\alpha(i)$ in the swath of $\mathcal{G}$. In the RDT algorithm of Section 4.4, each sample of α becomes a vertex. Due to the BVP and the particular motion primitives in $\mathcal{U}^p$, it may be difficult or impossible to precisely reach $\alpha(i)$. Therefore, line 4 calls an LPM to determine a primitive $\tilde{u}^p \in \mathcal{U}^p$ that produces a new state x_r upon integration from x_n. The result is depicted in Figure 5.6. For the default case in which $\mathcal{U}^p$ represents the discrete-time model, the action is chosen by applying all $u \in U$ over time Δt and selecting the one that minimizes

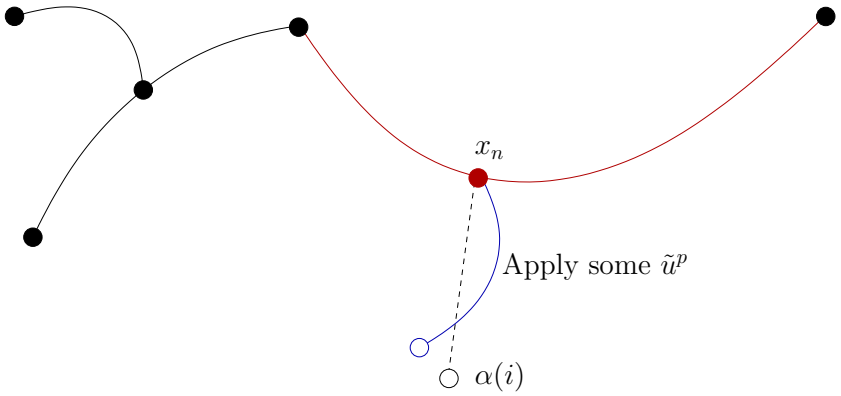


Figure 5.6: If the nearest point S lies in the state trajectory segment associated to an edge, then the edge is split into two, and a new vertex is inserted into $\mathcal{G}$.

$\rho(x_r, \alpha(i))$. One additional constraint is that if x_n has been chosen in a previous iteration, then $\tilde{u}_p$ must be a motion primitive that has not been previously tried from x_n; otherwise, duplicate edges would result in $\mathcal{G}$ or time would be wasted performing collision checking for reachability graph edges that are already known to be in collision. The remaining steps add the new vertex and edge from x_n. If x_n is contained in the trajectory produced by an edge e, then e is split as described in Section 4.3.

Part II

Deterministic Sampling-based Motion Planning

In part 1 we gradually introduced the main steps involved in any sampling based motion planning problem. By way of contrast, the focus of part 2 is on a particular components i.e the *sampling*. As anticipated, many algorithms used in practice are based on i.i.d uniform random sampling schemes. Although such algorithms have proven to yield good outcomes in many practical scenarios and in certain cases to converge to optimal solutions, in a probabilistic sense ([4],[5]), it is natural to wonder better performance or stronger theoretical guarantees can be granted by using *deterministic* sampling schemes specifically designed to realize a "good" coverage of the robot's state space.

This venue was originally explored by [2], whose results are reported in Chapter 6, in order to introduce the topic. However, Janson's analysis is limited to planning problem without differential constraints.

The *original contribution* of this work is the attempt to extend the methodologies and the theoretical results of [2] to two particular classes of dynamic systems. This is done in Chapter 7 and 8 by developing two novel (the best of our knowledge) sampling schemes explicitly designed to minimize a dispersion-like parameter related to the particular systems dynamics.

Chapter 6

Low-Dispersion Deterministic Sampling

This chapter presents some of the results obtained by [2] in the context of motion planning without differential constraint.

We show that the PRM algorithm is asymptotically optimal when run on deterministic sampling sequences in d dimensions whose L_2-dispersion is upper-bounded by $\gamma N^{-1/n}$, for some $\gamma \in \mathbb{R} > 0$ (we refer to such sequences as deterministic low-dispersion sequences), and with a connection radius $r_N \in \omega(N^{-1/n})$. In other words, the cost of the solution computed over N samples converges deterministically to the optimum as $N \to \infty$. As a comparison, the analogue result for the case of i.i.d. random sampling holds almost surely or in probability [Karaman and Frazzoli, 2011, Janson et al., 2015] (as opposed to deterministically) and requires a connection radius $\Omega(\log(N)/N)^{1/n}$, i.e., bigger.

The approach adopted in this chapter will be adapted later to obtain similar results for the more challenging case of kino-dynamic motion planning, which is the main focus of this thesis.

6.1 Background material

A key characteristic of any set of points on a finite domain is its L_2-dispersion. This concept will be particularly useful in elucidating the advantages of deterministic sampling over i.i.d. sampling. As such, in this section we review some relevant properties and results on the L_2-dispersion.

Definition 2. *For a finite, nonempty set $\mathcal{S}$ of points contained in a n-dimensional compact Euclidean subspace $\mathcal{X}$ with positive Lebesgue measure, its L_2-dispersion $D(\mathcal{S})$ is defined as*

$$D(\mathcal{S}) := \sup_{\mathbf{x}\in\mathcal{X}} \max_{\mathbf{s}\in\mathcal{S}} \{r > 0 : \exists \mathbf{x} \in \mathcal{X} \text{ with } B(\mathbf{x}, r) \cap \mathcal{S} = \emptyset\} \tag{6.1}$$

where $B(\mathbf{x}, r)$ is the open ball of radius r centered at $\mathbf{x}$.

Intuitively, the L_2-dispersion quantifies how well a space is covered by a set of points $\mathcal{S}$ in terms of the largest open Euclidean ball that touches none of the points. The quantity $D(\mathcal{S})$ is important in the analysis of path optimality as an optimal path may pass through

an empty ball of radius $D(\mathcal{S})$. Hence, $D(\mathcal{S})$ bounds how closely any path tracing through points in $\mathcal{S}$ can possibly approximate that optimal path.

The L_2-dispersion of a set of deterministic or random points is often hard to compute, but fortunately it can be bounded by the more-analytically tractable L_∞-dispersion. The L_∞-dispersion is defined by simply replacing the L_2-norm in equation (1) by the L_∞-norm, or max-norm. The L_∞-dispersion of a set $\mathcal{S}$, which we will denote by $D_\infty(\mathcal{S})$, is related to the L_2-dispersion in n dimensions by

$$D_\infty(\mathcal{S}) \leq D_2(\mathcal{S}) \leq \sqrt{n} D_\infty(\mathcal{S}), \tag{6.2}$$

which allows us to bound $D(\mathcal{S})$ when $D_\infty(\mathcal{S})$ is easier to compute. In particular, an important result due to is that the L_∞-dispersion of N independent uniformly sampled points on $[0,1]^n$ is $O((\log(N)/N)^{1/n})$ with probability 1. Corollary to this is that the L_2-dispersion is also $O((\log(N)/N)^{1/n})$ with probability 1.

Remarkably, there are deterministic sequences with L_2-dispersions of order $O(N^{-1/n})$, an improvement by a factor $\log(N)^{1/n}$. For instance, the Sukharev sequence, whereby $[0,1]^n$ is gridded into $N = k^n$ hypercubes and their centers are taken as the sampled points, can easily be shown to have L_2-dispersion of $(\sqrt{n}/2)N^{-1/n}$ for $N = k^n$ points. As we will see, the use of sample sequences with lower L_2-dispersions confers on PRM a number of beneficial properties, thus justifying the use of certain deterministic sequences instead of i.i.d. ones. In the remainder of the Chapter we will refer to sequences with L_2-dispersion of order $O(N^{-1/n})$ as low-dispersion sequences. A natural question to ask is whether we can use a sequence that minimizes the L_2-dispersion. Unfortunately, such an optimal sequence is only known for $n = 2$, in which case it is represented by the centers of the equilateral triangle tiling.

6.2 Problem Definition

Let $\mathcal{X} = [0,1]^n$ be the configuration space, where $n \in \mathbb{N}$. Let $\mathcal{X}_{\text{obs}}$ be a closed set representing the obstacles, and let $\mathcal{X}_{\text{free}} = \text{cl}(\mathcal{X}$
$\mathcal{X}_{\text{obs}})$ be the obstacle-free space, where $\text{cl}(\cdot)$ denotes the closure of a set. The initial condition is $x_{\text{init}} \in \mathcal{X}_{\text{free}}$, and the goal region is $\mathcal{X}_{\text{goal}} \subset \mathcal{X}_{\text{free}}$. A specific path planning problem is characterized by a triplet $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathcal{X}_{\text{goal}})$. A function $\sigma : [0,1] \to \mathbb{R}^n$ is a path if it is continuous and has bounded variation. If $\sigma(\tau) \in \mathcal{X}_{\text{free}}$ for all $\tau \in [0,1]$, is said to be collision-free. Finally, if σ is collision-free, $\tau(0) = \mathbf{x}_{\text{init}}$, and $\tau(1) \in \text{cl}(\mathcal{X}_{\text{goal}})$, then is said to be a feasible path for the planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathcal{X}_{\text{goal}})$.

The goal region$\mathcal{X}_{\text{goal}}$ is said to be regular if there exists $\xi > 0$ such that $\forall y \in \partial\mathcal{X}_{\text{goal}}$, there exists $z \in \mathcal{X}_{\text{goal}}$ with $B(z;\xi) \subset \mathcal{X}_{\text{goal}}$ and $y \in \partial B(z;\xi)$. Intuitively, a regular goal region is a smooth set with a boundary that has bounded curvature. Regularity is a technical condition we will use in our results, but is in fact quite weak, as nearly any goal region can be well-approximated by a regular goal region. Furthermore, we will say $\mathcal{X}_{\text{goal}}$ is ξ-regular if $\mathcal{X}_{\text{goal}}$ is regular for the parameter ξ. Denote the set of all paths by Σ. A cost function for the planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathcal{X}_{\text{goal}})$ is a function $c : \Sigma \to \mathbb{R}_0$; in this paper we will focus on the arc length function. The problem is then defined as follows:

Problem 1. *Given a path planning problem $(\mathcal{X}_{free}, \mathbf{x}_{init}, \mathcal{X}_{goal})$ with an arc length cost function $c : \Sigma \to \mathbb{R}_0$, find a feasible path σ^* such that $c(\sigma^*) = \min\{c(\sigma) : \sigma \text{ is feasible}\}$. If no such path exists, report failure.*

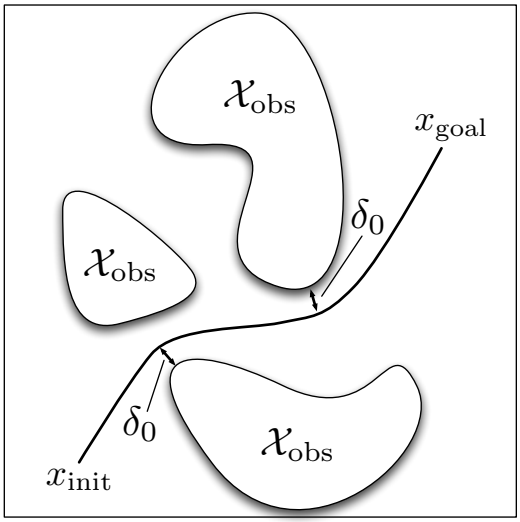


Figure 6.1: An example of a planning problem with a feasible δ_0-clear path.

A path planning problem can be arbitrarily difficult if the solution traces through a narrow corridor, which motivates the standard notion of *path clearance*.

For a given $\delta > 0$, define the δ-interior of $\mathcal{X}_{\text{free}}$ as the set of all configurations that are at least a distance δ from $\mathcal{X}_{\text{obs}}$. Then a path is said to have strong δ-clearance if it lies entirely inside the δ-interior of $\mathcal{X}_{\text{free}}$. Further, a path planning problem with optimal path cost c^* is called δ-robustly feasible if there exists a strictly positive sequence $\delta_n \to 0$, and a sequence $\{\sigma_n\}_{i=1}^n$ of feasible paths such that $\lim_{n\to\infty} c(\sigma_n) = c^*$ and for all $n \in \mathbb{N}$, σ_n has strong δ_n-clearance, $\sigma_n(1) \in \partial\mathcal{X}_{\text{goal}}$, and $\sigma_n(\tau) \notin \mathcal{X}_{\text{goal}}$ for all $\tau \in (0, 1)$.

Lastly, in this paper we will be considering a generic form of the PRM algorithm. That is, denote by gPRM (for generic PRM) the algorithm given by Algorithm 6. The function `SampleFree`(N) is a function that returns a set of $N \in \mathbb{N}$ points in $\mathcal{X}_{\text{free}}$. Given a set of samples V , a sample $\mathbf{v} \in V$, and a positive number r, `Near`$(V, \mathbf{v}, r)$ is a function that returns the set of samples $\{\mathbf{u} \in V : ||\mathbf{u} - \mathbf{v}||_2 < r\}$. Given two samples $\mathbf{u}, \mathbf{v} \in \mathcal{V}$, `CollisionFree`$(\mathbf{u}, \mathbf{v})$ denotes the boolean function which is true if and only if the line joining u and v does not intersect an obstacle. Given a graph $\mathcal{G}(V, E)$, where the node set V contains $\mathbf{x}_{\text{init}}$ and E is the edge set, `ShortestPath`$(\mathbf{x}_{\text{init}}, V, E)$ is a function returning a shortest path from $\mathbf{x}_{\text{init}}$ to $\mathcal{X}_{\text{goal}}$ in the graph $\mathcal{G}(V, E)$ (if one exists, otherwise it reports failure). Deliberately, we do not specify the definition of SampleFree and have left r_N in line 3 of Algorithm 6 unspecified, thus allowing for any sequence of points?deterministic or random?to be used, with any connection radius. We want to clarify that we are in no way proposing a new algorithm, but just defining an umbrella term for the PRM class of algorithms which includes, for instance, $sPRM$ and PRM^* as defined in [Karaman and Frazzoli, 2011].

Algorithm 6 gPRM

$V \leftarrow \{\mathbf{x}_{\text{init}}\} \cup \texttt{SampleFree}(N)$; $E \leftarrow \emptyset$
for all $\mathbf{v} \in V$ **do**
  $X_{\text{near}} \leftarrow \texttt{Near}(V/\{\mathbf{v}\}, \mathbf{v}, r_N)$
  for $\mathbf{x} \in X_{\text{near}}$ **do**
    if $\texttt{CollisionFree}(\mathbf{v}, \mathbf{x})$ **then**
      $E \leftarrow E \cup \{(\mathbf{v}, \mathbf{x})\} \cup \{(\mathbf{x}, \mathbf{v})\}$
return $\texttt{ShortestPath}(\mathbf{x}_{\text{init}}, V, E)$

6.3 Theoretical Results

In this section the main theoretical results are presented. We begin by proving that gPRM on low-dispersion sequences is asymptotically optimal, in the deterministic sense, for connection radius $r_N \in \omega(N^{-1/n})$. Previous works has required r_N to be at least $\Omega((\log(N)/N)^{1/n})$ for asymptotic optimality.

Theorem 1. *Let $(\mathcal{X}_{free}, \mathbf{x}_{init}, \mathcal{X}_{goal})$ be a δ-robustly feasible path planning problem in n dimensions, with $\mathcal{X}_{goal}$ ξ-regular. Let c^* denote the arc length of an optimal path σ^*, and let c_N denote the arc length of the path returned by gPRM (or ∞ if gPRM returns failure) with n vertices whose L_2-dispersion is $D(V)$ using a radius r_N. Then if $D(V) \leq \gamma N^{-1/n}$ for some $\gamma \in \mathbb{R}$ and*

$$N^{1/n} r_N \to \infty, \tag{6.3}$$

then $\lim_{n\to\infty} c_N = c^*$.

Proof. Fix $\varepsilon > 0$. By the δ-robust feasibility of the problem, there exists a σ_ϵ such that $c(\sigma_\varepsilon) \leq (1 + \varepsilon/3)c^*$ and σ_ϵ has strong δ_ε-clearance for some $\delta_\epsilon > 0$. Let R_N be a sequence such that $R_N \leq r_N$, $N^{1/n} R_N \to \infty$, and $R_N \to 0$, guaranteeing that there exists a $N_0 \in \mathbb{N}$ such that for all $N \geq N_0$,

$$(4 + 6/\varepsilon)\gamma N^{-1/n} \leq R_N \leq \min\{\delta_\epsilon, \xi, c^*\epsilon/6\}. \tag{6.4}$$

For any $N \geq N_0$, construct the closed balls $B_{N,m}$ such that $B_{N,i}$ has radius $\gamma N^{-1/n}$ and has center given by tracing a distance $(R_N - 2\gamma N^{-1/n})i$ from $\mathbf{x}_0$ along σ_ϵ (this distance is positive until $R_n N - 2\gamma N^{-1/n})i > c(\sigma_\varepsilon)$). This will generate $M_N = \lfloor c(\sigma_\varepsilon)/(R_n - 2\gamma N^{-1/n}) \rfloor$ balls. Define B_{N,M_N+1} to also have radius $\gamma N^{-1/n}$ but center given by the point where σ_ε meets $\mathcal{X}_{\text{goal}}$. Finally, define B_{N,M_N+2} to have radius $\gamma N^{-1/n}$ and center defined by extending the center of B_{N,M_N+1} into $\mathcal{X}_{\text{goal}}$ by a distance $R_N - 2\gamma N^{-1/n}$ in the direction perpendicular to $\partial\mathcal{X}_{\text{goal}}$. Note that $B_{N,M_N+2} \subset \mathcal{X}_{\text{goal}}$.

Since the dispersion matches the radii of all the $B_{N,m}$, each B_N, m has at least one sampled point within it. Label these points $\mathbf{x}_1, ..., \mathbf{x}_{M_N+2}$, with the subscripts matching their respective balls of containment. For notational convenience, define $\mathbf{x}_0 := \mathbf{x}_{\text{init}}$. Note that by construction of the balls, for $i \in \{0, ..., M_N + 1\}$, each pair of consecutively indexed points $(\mathbf{x}_i, \mathbf{x}_{i+1})$ is separated by no more than $R_N \leq r_N$. Furthermore, since $R_N \leq \delta_\epsilon$ there cannot be an obstacle between any such pair, and thus each pair constitutes an edge in the gPRM graph. Thus, we can upper-bound the cost c_N of the gPRM solution by the sum of the lengths of the edges $(\mathbf{x}_0, \mathbf{x}_1), ..., (\mathbf{x}_{M_N+1}, \mathbf{x}_{M_N+2})$:

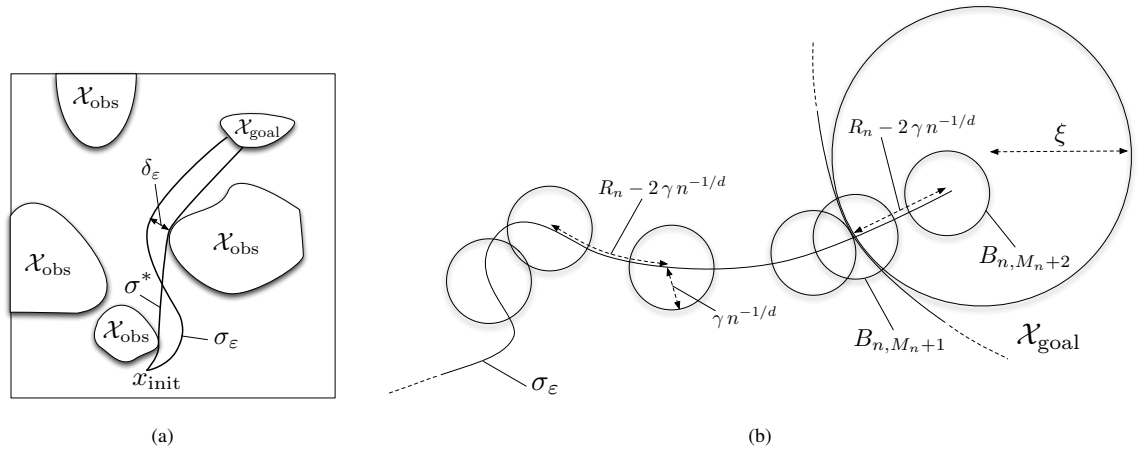


Figure 6.2: (a): Illustration in 2D of σ_ε as the shortest strongly δ_ϵ-robust feasible path, as compared to the optimal path σ^*, as used in the proof of the theorem. (b): Illustration in 2D of the construction of $B_1, ..., B_{M_N+2}$ in the proof of the theorem.

$$
\begin{aligned}
c_N &\leq \sum_{i=0}^{M_N+1} ||\mathbf{x}_{i+1} - \mathbf{x}_i|| \leq (M_N+2)R_N \leq \frac{c(\sigma_\varepsilon)}{R_N - 2\gamma N^{-1/n}} R_N + 2R_N \\
&\leq c(\sigma_\varepsilon + \frac{2\gamma N^{-1/n}}{R_N - 2\gamma N^{-1/n}} c(\sigma_\varepsilon) + 2R_N = c(\sigma_\varepsilon) + \frac{1}{\frac{R_N}{2\gamma N^{-1/n}} - 1} c(\sigma_\varepsilon) + 2R_N \qquad (6.5) \\
&\leq \big(1+\varepsilon/3\big)c^* + \frac{1}{3/\varepsilon + 1}\big(1+\varepsilon/3\big)c^* + \frac{\varepsilon}{3}c^* = (1+\varepsilon)c^*
\end{aligned}
$$

The second inequality follows from the fact that the distance between $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ is upper-bounded by the distance between the centers of $B_{N,i}$ and $B_{N,i+1}$ (which is at most $R_N - 2\gamma N^{-1/n}$) plus the sum of their radii (which is $2\gamma N^{-1/n}$). The last inequality follows from the facts that $c(\sigma_\varepsilon) \leq (1+\epsilon/3)c^*$ and equation (6.4). □

Note that if gPRM using $r_N > 2D(V)$ reports failure, then there are two possibilities: (i) a solution does not exist, or (ii) all solution paths go through corridors whose widths are smaller than $2D(V)$. Such a result can be quite useful in practice, as solutions going through narrow corridors could be undesirable anyways

6.4 Extension to Kinodynamic Planning

It is interesting to understand if the sampling strategy described in this chapter can be extended to motion planning with differential constraint In particular, we consider here the extension to systems with linear affine dynamics of the form: $\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}$, where A, B, and $\mathbf{c}$ are constants. The extension of the L_2-dispersion-based analysis of this paper to that case poses some challenges. The key roadblock is that the L_2-dispersion is no longer a particularly accurate measure of how suitable a set of points is to track an optimal differentially- constrained path. Essentially, Euclidean balls must be replaced

by *perturbation balls* [4], which are high-dimensional ellipses. To be clear, by a high-dimensional ellipse we mean a volume defined by

$$\{\mathbf{x} : \mathbf{x}^T Q \mathbf{x} < r\} \tag{6.6}$$

for some positive-definite matrix Q and scalar r. Although such ellipses may be inner-bounded by a Euclidean ball, this (poor) approximation adds an exponential factor of the controllability index of the pair (A, B) to the analysis. (Assuming the pair (A, B) is controllable, the controllability indices ν_i give a fundamental notion of how difficult a linear system is to control in the various directions. The number of controllability indices is equal to the number of control inputs, that is to the number of columns of B. The maximum, that is $\nu = \max \nu_i$, is referred to as the controllability index of the pair (A, B).)

The following theorem (whose proof is largely based on the analysis framework devised in [4]) summarizes the optimality result. Here gDPRM is just Algorithm 1 except that `Near` uses the cost in [4], equation (2)], instead of arc-length.

Theorem 2. *Under the assumptions of [4], Theorem VI.1, gDPRM with deterministic low-dispersion sampling is asymptotically-optimal for*

$$r_N = C_1 N^{-1/(\nu n)}, \tag{6.7}$$

for some constant C_1, while gDPRM with iid uniform sampling is asymptotically optimal for

$$r_N = C_2 \Big(\frac{\log N}{N} \Big)^{1/\tilde{D}}, \tag{6.8}$$

for some constant C_2, where $\tilde{D} = (n + \sum \nu_i^2)/2$.

If $\nu = 1$ (i.e., all directions are equally difficult to control), deterministic sampling and our analysis show all the same benefits as in the case of the path planning (non-kinodynamic) problem by getting rid of the $\log N$ term required by i.i.d. sampling without changing the exponent (as in this case, $\nu n = \mathrm{d}$ and $\tilde{D} = n$). Note that a special case where $\nu = 1$ is represented by the single-integrator model $\dot{\mathbf{x}}[t] = \mathbf{u}[t]$, which effectively reduces the kinodynamic planning problem to the path planning problem stated in chapter 4.

However, in general, the exponent for the case of deterministic low-dispersion sampling (i.e., the exponent in equation (6.7)) may be worse. For instance, for the double-integrator model in three dimensions, namely $\ddot{\mathbf{x}}[t] = \mathbf{u}[t]$ and $n = 6$, the three controllability indices are $\nu_1 = \nu_2 = \nu_3 = 2$. As a consequence, one obtains $\nu n = 12$ and $\tilde{D} = 9$, and the radius in equation (6.7) (i.e., for the deterministic case) is larger that the radius for equation (6.8) (i.e., for the case with i.i.d. uniform sampling).

This is not to say that deterministic sampling is necessarily inappropriate or not advantageous for differentially- constrained problems, but just that the analysis used here is inadequate (most critically, we crudely inner-bound ellipses via Euclidean balls). Our analysis does, however, suggest possible ways forward. One could consider a measure of dispersion which applies more specifically to ellipses, and possibly tailor a deterministic sequence to be low-dispersion in this sense. To our knowledge, no assessment of sample sequences in terms of this type of dispersion has been performed previously, and this represents a theoretically and practically important direction for future research (together with studying tailored notions of sampling sequences and dispersion for other classes of dynamical systems, e.g., driftless systems).

These problems will be the topics of Chapters 7 and 8.

Chapter 7

Systems with Linear Affine Dynamics

The content of this chapter is focused on a particular class of dynamic systems, i.e. systems with linear affine dynamics. Although their relative mathematical simplicity could seem excessively restrictive, the study of these systems is still of interest since more realistic non-linear models can be locally approximated via linearization around a reference point.

The chapter is organized as follow:

After a formal definition of the problem and an introduction to some well known mathematical concepts required for the subsequent analysis, in section 7.3 we describe how to build our novel sampling set for a given linear affine control system, together with a characterization of its properties (Lemma 3, Theorem 3). The two successive section characterize the asymptotic behavior of algorithms based on our sampling scheme. In section 7.4 we give the proof of the main result of our analysis (Theorem 5). This new theorem can be considered as a deterministic version of Theorem IV.6 of [4]. In section 7.5 Theorem 6 is introduced as corollary of Theorem 5, guaranteeing the deterministic convergence of our algorithm toward an optimal solution i a deterministic sense. Finally section 7.6 makes a comparison between the performance of the deterministic algorithm with its probabilistic counterpart.

7.1 Problem Definition

Let $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{U} = \mathbb{R}^m$ be the state space and control input space, respectively, of the robot, and let the dynamics of the robot be defined by the following linear system, which we require to be formally controllable:

$$\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}, \tag{7.1}$$

where $\mathbf{x}[t] \in \mathcal{X}$ is the state of the robot, $\mathbf{u}[t] \in \mathcal{U}$ is the control input of the robot, and $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}$, and $\mathbf{c} \in \mathbb{R}^n$ are constant and given.

A *trajectory* of the robot is defined by a tuple $\pi = (\mathbf{x}[], \mathbf{u}[], \tau)$, where τ is the arrival time or duration of the trajectory,$\mathbf{u} : [0, \tau] \to \mathcal{U}$ defines the control input along the trajectory, and $\mathbf{x} : [0, \tau] \to \mathcal{X}$ are the corresponding states along the trajectory given $\mathbf{x}[0]$ with $\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}$.

The cost $c[\pi]$ of a trajectory π is defined by the function:

$$c[\tau] = \int_0^\tau (1 + \mathbf{u}[t]^T R \mathbf{u}[t]) dt. \tag{7.2}$$

The matrix R determines the relative costs of the control inputs, as well as their costs relative to the duration of the trajectory. We denote this linear affine dynamical system with cost by $\Sigma = (A, B, \mathbf{c}, R)$.

Let $\mathcal{X}_{\text{free}} \subset \mathcal{X}$ define the free state space of the robot, which consists of those states that are within user-defined bounds and are collision-free with respect to obstacles in the environment. Let $\mathcal{X}_{\text{goal}} \subset \mathcal{X}$ be the goal region of the motion planning problem. A *dynamically feasible* trajectory $\pi = (\mathbf{x}[], \mathbf{u}[], T)$ is *collision-free* if $\mathbf{x}[t] \in \mathcal{X}_{\text{free}}$ for all $t \in [0, T]$. A trajectory π is said to be *feasible* for the trajectory planning problem if it is dynamically feasible, collision-free, $\mathbf{x}[0] = \mathbf{x}_{\text{init}}$, and $\mathbf{x}[T] \in \mathcal{X}_{\text{goal}}$. The objective is to find the feasible path with minimum associated cost.

7.2 Background Material

A critical component of our approach is to be able to compute the optimal trajectory $\pi^*[\mathbf{x}_0, \mathbf{x}_1]$ (and its cost $c^*[\mathbf{x}_0, \mathbf{x}_1]$) between any two states $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{X}$, in absence of obstacles and without restrictions on the control input.

Given a fixed arrival time τ and two states $\mathbf{x_0}$ and $\mathbf{x_1}$, we want to find a trajectory $(\mathbf{x}[], \mathbf{u}[], \tau)$ such that $\mathbf{x}[0] = \mathbf{x_0}$, $\mathbf{x}[\tau] = \mathbf{x_1}$, and $\dot{\mathbf{x}}[t] = A\mathbf{x}[t] + B\mathbf{u}[t] + \mathbf{c}$ for all $0 \leq t \leq \tau$, minimizing the cost function:

$$c[\tau] = \int_0^\tau (1 + \mathbf{u}^T R \mathbf{u}) dt.$$

This is the so called fixed state, fixed time optimal control problem, the reader may refer to [**?**, Lewis]or more details.

Let $G[t]$ be the *weighted controllability Gramian* given by:

$$G[t] = \int_0^t \exp[At'] B R^{-1} B^T \exp[A^T t'] dt'. \tag{7.3}$$

We note that $G[t]$ is a positive-definite matrix for $t > 0$ if the dynamics of the system is controllable.

Further, let $\bar{\mathbf{x}}[t]$ describe what the state would be at time t, starting at $\mathbf{x}_0$ at time 0 , if no control input were applied:

$$\bar{\mathbf{x}}[t] = \exp[At]\mathbf{x_0} + \int_0^t \exp[A(t-s)]\mathbf{c} ds, \tag{7.4}$$

Then, the optimal control policy for the fixed final state, fixed final time problem is given by:

$$\mathbf{u}[t] = R^{-1} B^T \exp[A^T(\tau - t)] G[\tau]^{-1} (\mathbf{x_1} - \bar{\mathbf{x}}[\tau]), \tag{7.5}$$

which is an open-loop control policy.

By filling in the this control policy in the cost function , we find a closed-form expression for the cost optimal trajectory between $\mathbf{x_0}$ and $\mathbf{x_1}$ for a given arrival time τ:

$$c[\tau] = \tau + ||\mathbf{x_1} - \bar{\mathbf{x}}[\tau]||^2_{G[\tau]^{-1}}, \tag{7.6}$$

where we defined the norm:

$$||\mathbf{x}||_{G[t]^{-1}} = \sqrt{\mathbf{x}^T G[t]^{-1}\mathbf{x}}. \tag{7.7}$$

The optimal connection time τ^* may be computed by minimizing $c[\tau]$ over τ. Let $\pi^*[\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_K]$ denote the concatenation of the trajectories $\pi^*[\mathbf{x}_k, \mathbf{x}_{k+1}]$ between successive states $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_K \in \mathcal{X}$.

We can now introduce a result characterizing the effect of perturbations of the endpoints of path on its cost. The reader can refer to [4] for the more details.

Lemma 1. *Let $\mathbf{x_0}, \mathbf{x_1} \in \mathcal{X}$, $\mathbf{x_0} \neq \mathbf{x_1}$, $\pi = \pi^*[\mathbf{x_0}, \mathbf{x_1}] = (\mathbf{x}[], \mathbf{u}[], \tau^*)$, and denote $c = c[\pi]$. Consider bounded start and end state perturbations $\delta\mathbf{x_0}, \delta\mathbf{x_1} \in \mathbb{R}^n$ such that*

$$\max\{||\delta\mathbf{x_0}||_{G[\tau^*]^{-1}}, ||\delta\mathbf{x_1}||_{G[\tau^*]^{-1}} \leq \eta\sqrt{c}\}.$$

Let $\sigma = \pi^*[\mathbf{x_0} + \delta\mathbf{x_0}, \mathbf{x_1} + \delta\mathbf{x_1}] = (\mathbf{y}[], \mathbf{v}[], \tilde{\tau}^*)$ be the optimal trajectory between the the perturbed endpoints. Then for $\mathbf{x_0}, \mathbf{x_1}$ such that c is sufficiently small, we have the cost bound

$$c[\sigma] \leq c[\pi](1 + 4\eta + O[\eta^2 + \eta c]).$$

Additionally we may bound the geometric extent of σ:

$$||\mathbf{y}[t] - \mathbf{x_0}|| = O[c(||\mathbf{x_0}|| + \eta + 1)]$$

for $t \in [0, \tilde{\tau}^*]$.

The previous lemma motives the following:

Definition 3. *Given a state $\mathbf{x} \in \mathcal{X}$, a fixed time τ and $r > 0$, the* fixed-time perturbation ball *centered in $\mathbf{x}$ of radius r is defined as*

$$\Delta[\mathbf{x}, \tau, r] := \{\mathbf{z} : \|\mathbf{x} - \mathbf{z}\|_{G^{-1}[\tau]} \leq r\}. \tag{7.8}$$

This set represents perturbations of $\mathbf{x}$ with limited effects on both incoming and outgoing trajectories (depending on whether a point is viewed or a start state perturbation respectively).

We also briefly reviewing the concept of *controllability indices* for a controllable system (A, B). Let $\mathbf{b}_k$ denote the kth column of B. Consider searching the columns of the controllability matrix $\mathcal{C}[A, B] = [B \quad AB \quad \cdots \quad A^{n-1}B]$ from left to right for a set of n linearly independent vectors. This process is well-defined for a controllable pair (A, B) since $\text{rank}[\mathcal{C}[A, B]] = n$. The resulting set $S = \{\mathbf{b}_1, A\mathbf{b}_1, ..., A^{\nu_1-1}\mathbf{b}_1, \mathbf{b}_2, ..., A^{\nu_2-1}\mathbf{b}_2, ..., A^{\nu_m-1}\mathbf{b}_m\}$ defines the controllability indices $\{\nu_1, ..., \nu_m\}$ where $\sum_{k=1}^m \nu_k = n$ and $\nu = \max_k \nu_k$ is called the *controllability index* of (A, B). The ν_k give a fundamental notion of how difficult a system is to control in various directions; indeed these indices are a property of the system invariant with respect to similarity transformation, e.g. permuting the columns of B. We can now state a technical lemma giving a lower bound for the determinant of $G[t]$ that will be used in the next section. The reader can refer to [4] for the proof.

Lemma 2. *Suppose that the pair (A, B) has controllability indices $\{\nu_1, ..., \nu_m\}$, then*

$$\det[G[t]] = \Theta[t^D]$$

as $t \to 0$, where $D = \sum_{k=1}^m \nu_k^2$.

7.3 Low $G[\tau]^{-1}$-Dispersion Sampling Set

In order to extend the sampling strategy adopted in [2] for purely kinematic motion planning to differentially constrained systems we need to develop a novel concept of dispersion, reflecting the the notion of distance between two different points of the state space $\mathcal{X}$ as the cost required to steer the system from the starting point to the arrival, respecting the constraints imposed by the dynamics of the system.

Keeping in mind that the L_2-dispersion of a set of points $\mathcal{S} \subset \mathcal{X}$ can be interpreted as the radius of the biggest Euclidean ball in $\mathcal{X}$ that does not contain any point of $\mathcal{S}$, a reasonable way to adapt the traditional definition of dispersion to the case under consideration appears to be:

the radius of the smallest fixed-time perturbation ball that does not contain any point of $\mathcal{S}$.

Note that, according to this definition, for the same set $\mathcal{S}$, at different fixed times τ correspond different values of dispersion. We will refer to this dispersion measure briefly as $G[\tau]^{-1}$-dispersion.

Now some nomenclatures are introduced. Given a set $\mathcal{S} \subset \mathcal{X}$ of N samples, and a generic distance function $d : \mathcal{X} \times \mathcal{X} \to [0, +\infty)$, for any pair of points $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ we denote by $\mathcal{E}_{\mathbf{x}}^{(d)}[\mathbf{y}]$ the set

$$\mathcal{E}_{\mathbf{x}}^{(d)}[\mathbf{y}] := \{\mathbf{z} \in X : d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y})\}, \tag{7.9}$$

that represents the ellipsoid centered in $\mathbf{x}$ with radius $d(\mathbf{x}, \mathbf{y})$. Then for any $\mathbf{x} \in \mathcal{X}$, the set $\mathcal{Y}_{\mathcal{S}}^{(d)}[\mathbf{x}]$ is defined as:

$$\mathcal{Y}_{\mathcal{S}}^{(d)}[\mathbf{x}] := \{\mathbf{y} \in \mathcal{X} : \mathcal{S} \cap \mathcal{E}_{\mathbf{x}}^{(d)}[\mathbf{y}] = \emptyset\}. \tag{7.10}$$

Finally we define the functional $\Phi^{(d)} : \mathcal{X}^N \to \mathbb{R}^+$ as

$$\Phi^{(d)}[\mathcal{S}] = \max_{\mathbf{x} \in \mathcal{X}} \max_{\mathbf{y} \in \mathcal{Y}_{\mathcal{S}}^{(d)}[\mathbf{x}]} \{d(\mathbf{x}, \mathbf{y})\}. \tag{7.11}$$

As can be easily understood, $\Phi^{(d)}$ represents the dispersion induced by the distance d. From now on, d_1 and d_2 represent the Euclidean and the $G[\tau]^{-1}$-induced distance, respectively. For the sake of brevity, we write $\Phi^{(1)}$ in place of $\Phi^{(d_1)}$ and $\Phi^{(2)}$ in place of $\Phi^{(d_2)}$, the same applies to the other symbols previously introduced.

Further, given a bijective map $f : \mathbb{R}^n \to \mathbb{R}^n$ we define the set $f[\mathcal{S}]$ as:

$$f[\mathcal{S}] := \{\mathbf{s}' \in \mathcal{X} : \exists \mathbf{s} \in \mathcal{S} | \mathbf{s}' = f[\mathbf{s}]\}. \tag{7.12}$$

Note that, being f bijective, $f(\mathcal{S})$ has the same cardinality of $\mathcal{S}$.

In the scientific literature ([1],[2],[12]) are presented many ways to build sets characterized by a low L_2-dispersion, i.e. sets such that

$$\Phi^{(1)}[\mathcal{S}] \leq \gamma N^{-1/n}, \tag{7.13}$$

for some positive constant γ, being n the dimension of $\mathcal{X}$.

The basic idea is to apply a bijective map f to the element of a set $\mathcal{S}$ with low $\Phi^{(1)}$ to obtain a set $f[\mathcal{S}]$ with low $\Phi^{(2)}$. The quest for such a function gives rise to the following:

Problem 2. *Given two distances on $\mathcal{X}$, namely d_1 and d_2, determine if a bijective map $f : \tilde{\mathcal{X}} \to \mathcal{X}$ exists, such that:*

$$\forall \mathbf{x}, \mathbf{y} \in \tilde{\mathcal{X}} \quad d_1(\mathbf{x}, \mathbf{y}) = d_2(f[\mathbf{x}], f[\mathbf{y}]). \tag{7.14}$$

If this is indeed the case, determine f.

For the particular choice of d_1, d_2 considered in this work, the following result holds

Lemma 3. *Let $G[\tau]^{-1} = L^T[\tau]L[\tau]$ be the Cholesky decomposition of $G[\tau]^{-1}$, and $f : \mathbb{R}^n \to \mathbb{R}^n$ the linear map $f_\tau[\mathbf{x}] := L[\tau]^{-1}\mathbf{x}$, then*

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \quad d_1(\mathbf{x}, \mathbf{y}) = d_2(f_\tau[\mathbf{x}], f[\mathbf{y}]). \tag{7.15}$$

Proof. To start with, notice that L is well defined, being G^{-1} a positive definite matrix. We have that:

$$\begin{aligned} d_2(f_\tau[\mathbf{x}], f_\tau[\mathbf{y}]) &= \sqrt{(f_\tau[\mathbf{x}] - f_\tau[\mathbf{y}])^T G[\tau]^{-1}(f_\tau[\mathbf{x}] - f_\tau[\mathbf{y}])} \\ &= \sqrt{(L^{-1}\mathbf{x} - L^{-1}\mathbf{y})^T (L^T L)(L^{-1}\mathbf{x} - L^{-1}\mathbf{y})} \\ &= \sqrt{((L^{-1}\mathbf{x} - L^{-1}\mathbf{y})^T L^T)(L(L^{-1}\mathbf{x} - L^{-1}\mathbf{y}))} \\ &= \sqrt{(L(L^{-1}\mathbf{x} - L^{-1}\mathbf{y}))^T (L(L^{-1}\mathbf{x} - L^{-1}\mathbf{y}))} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})} \\ &= d_1(\mathbf{x}, \mathbf{y}) \end{aligned} \tag{7.16}$$

□

We can now prove the following:

Theorem 3. *Let $\mathcal{S} \subset \tilde{\mathcal{X}}$ be a set of N samples, then for any $\tau > 0$,*

$$\Phi^{(2)}[f_\tau(\mathcal{S})] = \Phi^{(1)}[\mathcal{S}], \tag{7.17}$$

where $\tilde{\mathcal{X}}_\tau = f_\tau^{-1}[\mathcal{X}]$.

Proof. We first need to prove the intermediate result:

$$\mathbf{y} \in \mathcal{Y}^{(1)}_{\mathcal{S}}[\mathbf{x}] \iff f_\tau(\mathbf{y}) \in \mathcal{Y}^{(2)}_{f_\tau(\mathcal{S})}[f_\tau(\mathbf{x})]. \tag{7.18}$$

Given any $\mathbf{y}' \in \mathcal{X}$ there is a unique $\mathbf{y} \in \tilde{\mathcal{X}}_\tau$ s.t. $\mathbf{y}' = f_\tau(\mathbf{y})$. By definition, the proposition: $\mathbf{y}' \notin \mathcal{Y}^{(2)}_{f_\tau(\mathcal{S})}[f_\tau(\mathbf{x})]$ holds if and only if $\exists \mathbf{s}' \in f_\tau(\mathcal{S})$ s.t. $\mathbf{s}' \in \mathcal{E}^{(2)}_{f_\tau(\mathbf{x})}[\mathbf{y}']$. Imposing $\mathbf{s}' = f_\tau(\mathbf{s})$, with $\mathbf{s} \in \mathcal{S}$, we have the following chain of equivalences:

$$\begin{aligned} \mathbf{s}' \in \mathcal{E}^{(2)}_{f_\tau(\mathbf{x})}[\mathbf{y}'] &\iff d_2(f_\tau[\mathbf{x}], \mathbf{s}') \leq d_2(f_\tau[\mathbf{x}], \mathbf{y}') \\ &\iff d_2(f_\tau(\mathbf{x}), f_\tau(\mathbf{s})) \leq d_2(f_\tau(\mathbf{x}), f_\tau(\mathbf{y})) \\ &\iff d_1(\mathbf{x}, \mathbf{s}) \leq d_1(\mathbf{x}, \mathbf{y}) \\ &\iff \mathbf{s} \in \mathcal{E}^{(1)}_{\mathbf{x}}[\mathbf{y}] \end{aligned} \tag{7.19}$$

that is equivalent to say that $\mathbf{y} \notin \mathcal{Y}_{\mathcal{S}}^{(1)}[\mathbf{x}]$. Now the initial proposition can be easily proved as follow

$$\begin{aligned}
\Phi^{(2)}[f_\tau(\mathcal{S})] &= \max_{\mathbf{x}' \in \mathcal{X}} \max_{\mathbf{y}' \in \mathcal{Y}^{(2)}_{f(\mathcal{S})}[\mathbf{x}']} \{d_2(\mathbf{x}', \mathbf{y}')\} \\
&= \max_{\mathbf{x} \in \tilde{\mathcal{X}}_\tau} \max_{\mathbf{y}' \in \mathcal{Y}^{(2)}_{f_\tau[\mathcal{S}]}[f_\tau(\mathbf{x})]} \{d_2(f_\tau[\mathbf{x}], \mathbf{y}')\} \\
&= \max_{\mathbf{x} \in \tilde{\mathcal{X}}_\tau} \max_{\mathbf{y} \in \mathcal{Y}^{(1)}_{\mathcal{S}}[\mathbf{x}]} \{d_2(f_\tau[\mathbf{x}], f_\tau[\mathbf{y}])\} \\
&= \max_{\mathbf{x} \in \tilde{\mathcal{X}}_\tau} \max_{\mathbf{y} \in \mathcal{Y}^{(1)}_{\mathcal{S}}[\mathbf{x}]} \{d_1(\mathbf{x}, \mathbf{y})\} \\
&= \Phi^{(1)}[\mathcal{S}]
\end{aligned} \tag{7.20}$$

□

It is important to note that the space $\tilde{\mathcal{X}}_\tau$ on which we are supposed to find a low L_2-dispersion sampling set, is different from the initial space $\mathcal{X}$ where we the final low $G[\tau]^{-1}$-dispersion sampling set has to be placed. For example, assuming for $\mathcal{X}$ a unit n-dimensional hyper-cube, $\tilde{\mathcal{X}}_\tau$ is a n-dimensional hyper-parallelogram with measure

$$\mu[\tilde{X}_\tau] = |\det[L[\tau]]|\mu[X] = \frac{1}{\sqrt{|\det[G[\tau]]|}}. \tag{7.21}$$

At this point we describe a possible way to build a low L_2-dispersion set $\tilde{\mathcal{S}}_\tau$ on the space $\tilde{\mathcal{X}}_\tau$, by embedding this region in a parallelepiped and building a regular grid on it. However, it is important to note that the use of this particular sampling method is inessential to our aim, in that other methods, if characterized by similar bound on the L_2-dispersion of the corresponding samples, can be used.

It is useful to remember that the L_2-dispersion of a set $\mathcal{S}$ is related to its L_∞-dispersion by the following inequality ([2], [1]):

$$D_\infty[\mathcal{S}] \leq D_2[\mathcal{S}] \leq \sqrt{n} D_\infty[\mathcal{S}]. \tag{7.22}$$

Now, suppose we want to place a to-be-determined number of points on $\tilde{\mathcal{X}}_\tau$, in such a way that their L_∞-dispersion is at worst d. In order to do this we construct a grid on $\tilde{\mathcal{X}}_\tau$ following way:

1. Consider a vertex of $\tilde{\mathcal{X}}_\tau$ and pick one of the edges departing from that vertex. Let a_1 denote the length of the selected edge.

2. For all the other edges, take their orthogonal projections on the edges already considered, whose lengths are noted by $a_2, ..., a_n$ respectively. In this way, we have that $a_1 \cdot ... \cdot a_n = \mu[\tilde{\mathcal{X}}_\tau]$.

3. Split up each one of the mutually orthogonal edges obtained so far in segments of length d and build a grid.

4. Take a point $\tilde{\mathbf{s}}$ in $\tilde{\mathcal{X}}_\tau$ for each of the cells of side d that compose the grid.

It is evident that the set of points $\tilde{\mathcal{S}}_\tau \subset \tilde{\mathcal{X}}_\tau$ built in this way satisfies the requirement imposed on the L_∞-dispersion, and it is now possible to proceed with the computation of the number of points N required to build $\tilde{\mathcal{S}}_\tau$.

$$N = \prod_{i=1}^{n} \left\lceil \frac{a_i}{d} \right\rceil < \prod_{i=1}^{n} (1 + \frac{a_i}{d}) \leq \gamma_\infty^n \frac{\mu[\tilde{X}\tau]}{d^n}, \tag{7.23}$$

for some constant γ_∞. Inverting the previous inequality we have that, given $N \in \mathbb{N}$ it's always possible to arrange a set $\mathcal{S}[N]$ of N points on $\tilde{\mathcal{X}}_\tau$, in such a way that

$$D_\infty[\mathcal{S}[N]] \leq \gamma_\infty \Big(\frac{\mu[\tilde{\mathcal{X}}_\tau]}{N} \Big)^{1/n}, \tag{7.24}$$

and consequently

$$D_2[\mathcal{S}[N]] \leq \gamma \Big(\frac{\mu[\tilde{\mathcal{X}}_\tau]}{N} \Big)^{1/n}, \tag{7.25}$$

for some constant γ.

We have now all the element to describe explicitly the family of sampling sets that we propose to use in place of the iid samples, for the linear quadratic motion planning problems, and whose theoretical properties will are discussed in the next section of this paper.

Definition 4. *For each number of samples $N \in \mathbb{N}$ and for any fixed arrival time $\tau > 0$ the set $\mathcal{S}_\tau[N] \subset \mathcal{X}$ is defined as*

$$\mathcal{S}_\tau[N] := f_\tau[\tilde{\mathcal{S}}_\tau], \tag{7.26}$$

where $\tilde{\mathcal{S}}_\tau \in \tilde{\mathcal{X}}_\tau$ has L_2-dispersion less than $\gamma \Big(\frac{\mu[\tilde{\mathcal{X}}_\tau]}{N} \Big)^{1/n}$ for some constant γ. As a consequence of Theorem 1, the same upper bound holds for the $G[\tau]^{-1}$-dispersion of $\mathcal{S}_\tau[N]$.

7.4 Deterministic Exhaustivity

In [4] was proved a property characterizing random sampling schemes for the LQDMP, namely their *probabilistic exhaustivity*: *any feasible trajectory through the configuration space $\mathcal{X}$ is "traced" arbitrarily well by connecting randomly distributed points from a sufficiently large sample set covering the configuration space.* We now define what it means for series of states to closely approximate a given trajectory.

Definition 5. *Let $\pi = (\mathbf{x}, \mathbf{u}, T_\pi)$ be a dynamically feasible trajectory. Given a set of waypoints $\{\mathbf{y}_k\}_{k=0}^K \subset X$, we associate the trajectory $\sigma = \pi^*[\mathbf{y}_0, ..., \mathbf{y}_K] = (\mathbf{y}, \mathbf{v}, T_\sigma)$. We say $\{\mathbf{y}_k\}$ (ε, r, p)-*trace *the trajectory π if: (a) $c[\sigma] \leq (1 + \varepsilon)c[\pi]$, (b) $c^*[\mathbf{y}_k, \mathbf{y}_{k+1}] \leq r$ for all k, and (c) the maximum distance from ant point of $\mathbf{y}$ to $\mathbf{x}$ is no more then p.*

The main results of that analysis is summarized in the following theorem ([4, Theorem IV.6]):

Theorem 4. *Let Σ be a controllable system and suppose $\pi = (\mathbf{x}, \mathbf{u}, T)$ is a dynamically feasible trajectory with strong δ-clearance. Let $N \in \mathbb{N}$, $\varepsilon > 0$ and consider a set of sample nodes $V = \{\mathbf{x}[0]\} \cup \texttt{SampleFree}[N]$. Define $\tilde{D} = (d + n)/2$, $C_{\Sigma, \mathcal{X}_{free}} =$*

$\big(C_\mu^{-1}\tilde{D}^{-1}6^{n+D/2}2^{n/2}[\mathcal{X}_{free}]\big)^{1/\tilde{D}}$ *and consider the event* E_N *that there exist waypoints* $\{\mathbf{y}_k\}_{k=0}^K \subset V$ *which* (τ_N, ϵ, p_N)*-trace* π*, where*

$$r_N = (1+\eta)^{1/\tilde{D}} C_{\Sigma,\mathcal{X}_{free}} \Big(\frac{\log N}{N}\Big)^{1/\tilde{D}}, \tag{7.27}$$

for a free parameter $\eta \geq 0$*, and* $p_N = C_p r_N$*, for some constant* C_p*. Then, as* $N \to \infty$*, the probability that no such waypoint set exists is asymptotically bounded as*

$$1 - \mathbb{P}[E_N] = O\big(N^{-\eta/\tilde{D}} \log^{-1/\tilde{D}} N\big). \tag{7.28}$$

However, random sampling schemes only give probabilistic guarantees of existence of a trajectory approximation, in the sense that for any finite number of samples, given a dynamically feasible trajectory, exists among them a sequence of points that trace that path, only with a certain probability. By way of contrast, with a deterministic sampling scheme like the one introduced in the previous section, it is *always* possible to pick waypoints $\{\mathbf{y}_k\}_{k=0}^K$ among the samples that trace arbitrarily well any feasible path, provided that the number of used samples N is sufficiently large. By analogy, we will refer to this property of our deterministic sampling scheme as *deterministic exhaustivity*. Moreover, the new sampling strategy allows the use of a cost threshold r_N asymptotically smaller than the one required by its random counterpart.

Similarly to what is done in [4], our approach for proving deterministic exhaustivity proceeds by tiling with a sequence of perturbation balls any given feasible trajectory in the state space, but this time the considerations about the measure of the balls [4, Lemmas IV.4, IV.5], used to prove that these sets contain samples (with a certain probability), are no longer needed. Instead, we only have to prove that the sampling set $\mathcal{S}_{\tau_N}[N]$ is "good enough", so that each small perturbation ball contains at least a sample, for N sufficiently large. We take these points as a sequence of waypoints which tightly follows the reference path with few exceptions, and never has a gap over any section of the reference path when it does deviate further.

Theorem 5. *Let* Σ *be a system satisfying the assumptions* A_Σ *and suppose* $\pi = (\mathbf{x}, \mathbf{u}, T)$ *being a dynamically feasible trajectory with strong* δ*-clearance,* $\delta > 0$*. Let* $N \in \mathbb{N}$*,* $\varepsilon > 0$*, and consider a set of nodes* $V := \{\mathbf{x}[0]\} \cup \mathcal{S}_{\tau_N}[N]$*, where* $\tau_N = r_N/6$*. Then for sufficiently large* N*, exist waypoints* $\{y_k\}_{k=0}^K \subset V$ *which* (ε, r_N, p_N)*-trace* π*, where* $r_N = \omega(N^{-1/\tilde{D}})$*, (with* $\tilde{D} = \frac{n+D}{2}$ *as in [2]), and* $p_N = C_p r_N$ *for some constant* C_p*.*

Proof. Note that in the case $c[\pi] = 0$ we may pick $\mathbf{y_0} = \mathbf{x}[0]$ to be the only waypoint and the result is trivial. Therefore assume $c[\pi] > 0$. Take $\mathbf{x}[t_k]$to be points spaced along π at cost intervals $r_N/2$; more precisely let $t_0 = 0$, and for $k = 1, 2, ...$ consider

$$t_k = \min\{t \in (t_{k-1}, T) | c^*[\mathbf{x}[t_{k-1}], \mathbf{x}[t]] \geq r_n/2\}. \tag{7.29}$$

Let K be the first k for which the set is empty; take $t_K = T$. Fixed $\beta = \min\{\varepsilon, 2\sqrt{6}\}$, for each k we define

$$S_k = \Delta[\mathbf{x}[t_k], \tau_N, (\beta/6)\sqrt{r_N/2}], \tag{7.30}$$

We know that, considering how $\mathcal{S}_{\tau_N}[N]$ was defined, any ellipsoid with radius greater than the d_{τ_N}-induced dispersion of $\mathcal{S}_{\tau_N}[N]$, contains at least a sample. Now we prove the for N

large enough the radius of the sets S_k is greater than the d_{τ_N}-induced dispersion of $\mathcal{S}_{\tau_N}[N]$, or equivalently

$$\gamma\Big(\frac{\mu[\tilde{\mathcal{X}}_\tau]}{N}\Big)^{1/n} \leq (\beta/6)\sqrt{r_N/2}. \tag{7.31}$$

We know that $\mu[\tilde{\mathcal{X}}_{\tau_N}] = 1/\sqrt{|\det[G[\tau]]|}$, but for Lemma 2:

$$\det[G[t]] = \Theta[t^D] \tag{7.32}$$

as $t \to 0$.

Since $\tau_N = r_N/6 \to 0$ as $N \to \infty$, we can pick $\bar{N} \in \mathbb{N}$ and a positive constant α in such a way that $\mu[\tilde{\mathcal{X}}_{\tau_N}] \leq \alpha r_N^{-D/2}$, for all $N \geq \bar{N}$. For this reason, provided that:

$$\gamma\left(\frac{\alpha}{N \cdot r_N^{D/2}}\right)^{1/n} \leq (\beta/6)\sqrt{r_N/2}, \tag{7.33}$$

the inequality (7.37) will automatically be satisfied. Straightforward calculations show that the last inequality holds iff $r_N \geq K(\varepsilon)N^{-1/\tilde{D}}$, where $K(\varepsilon) = \left(\frac{6\sqrt{2}\gamma\alpha^{1/n}}{\beta(\varepsilon)}\right)^{\frac{2}{(D/n+1)}}$.

Now since by hypothesis $r_N = \omega(N^{-1/\tilde{D}})$, no matter how big $K(\varepsilon)$ is, exists $N_0 \in \mathbb{N}$ such that $r_N \geq K(\varepsilon)N^{-1/\tilde{D}}$, for all $N > \max\{N_0, \bar{N}\}$.

We note that the S_k are disjoint (as long as $\varepsilon < 2\sqrt{6}$) since $c^*[\mathbf{x}[t_{k-1}], \mathbf{x}[t_k] = r_N/2$ implies $||\mathbf{x}[t_{k-1}] - \mathbf{x}[t_k]||_{G[r_N/6]^{-1}} \geq \sqrt{r_N/3}$.

Now we will show the existence of suitable waypoints $\{\mathbf{y}_k\}_{k=0}^K \subset V$. Suppose that every S_k contains a point of V, we may apply Lemma 1 to verify that these points (ε, r_N, p_N)-trace π. For $\mathbf{y}_k \in S_k$, $\mathbf{y}_{k-1} \in S_{k-1}$ we have:

$$\begin{aligned} c^*[\mathbf{y}_{k-1}, \mathbf{y}_k &\leq (1+\beta)c^*[\mathbf{x}[t_{k-1}], \mathbf{x}[t_k]] \\ &\leq (1+\varepsilon/2)(r_N/2). \end{aligned} \tag{7.34}$$

Hence the total cost of $\pi^*[\mathbf{y}_0, ..., \mathbf{y}_K] = (\mathbf{y}[], \mathbf{v}[], T_\sigma)$ is bounded above by $(1+\epsilon)c[\pi]$. We also have $c^*[\mathbf{y}_{k-1}, \mathbf{y}_k] \leq r_N$ for all k. The maximum Euclidean distance from any point of $\mathbf{y}$ (say, on the segment $\pi^*[\mathbf{y}_k, \mathbf{y}_{k+1}]$) to $\mathbf{x}$ is bounded above by its distance to $\mathbf{x}[t_k]$ which by Lemma IV.1 is $O[r_N(||\mathbf{x}_0|| + 1)] = O[r_N]$ as $N \to \infty$ since $||\mathbf{x}[t]||$ achieves some fixed maximum over $[0, T]$. □

It is interesting at this point to note that the requirement on r_N imposed by $\mathcal{S}_{\tau_N}[N]$ is weaker than the one imposed by the use of i.i.d sampling sequences, as shown in [4], being in that case

$$r_N = C\Big(\frac{\log N}{N}\Big)^{1/\tilde{D}}. \tag{7.35}$$

7.5 Deterministic Convergence to an Optimal Solution

As a corollary of the Deterministic Exhaustivity property of the low-$G[\tau]^{-1}$ dispersion samples, is it possible to prove the following result, that will be referred to as *deterministic optimality*:

Theorem 6. *Let $(\Sigma, \mathcal{X}_{free}, \mathbf{x}_{init}, \mathcal{X}_{goal})$ be a trajectory planning problem satisfying the assumptions of [4]. Then the gDPRM algorithms (Chapter 6; [2]) are asymptotically optimal for*

$$r_N = \omega(r_n^{-1/\tilde{D}}). \tag{7.36}$$

The proof is perfectly similar to that of Theorem VI.1 in [4], and will be therefore omitted. The reader would appreciate the analogy between Theorem 6 and Theorem 1 of section 6. Even in this case,the leverage of the properties of the deterministic sampling scheme adopted allows to spare a logarithmic term in the expression of the connection radius r_N. A smaller connection radius implies a smaller number of connection attempted by the planning algorithm in the process of expansion of the exploration tree, with consequent economy in time and computational effort.

7.6 Experimental Results

The theoretical results presented in the previous sections characterize the asymptotical behavior of planning algorithm based on a low-$G[\tau]^{-1}$ sampling set. These result are significative in that they give guarantees of convergence to a optimal solutions in a deterministic sense, however for practical purpose a merely asymptotic analysis is of little relevance. In order to highlight the benefits offered by deterministic sampling with respect to iid uniform random sampling a comparison is made between the two version of the algorithm for a number of samples of practical interest. To compare the performances of of the sampling strategy previously described with random iid samples, we have done simulations on a 2-dimensional and a 3-dimensional double integrator system, applying the same algorithm[1] with the same cost threshold (parameter r) in both the cases, and just changing the sampling scheme adopted.

The $DPRM^*$ algorithms were implemented in Julia and run using a Unix operating system with a 2.0 GHz processor and 8 GB of RAM.

The double integrator robot is a circular robot capable of moving in any direction by controlling its acceleration. Its state space is four-dimensional, and its linear dynamics are described by:

$$\dot{\mathbf{x}}[t] = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \mathbf{x}[t] + \begin{bmatrix} 0 \\ I \end{bmatrix} \mathbf{u}[t]. \tag{7.37}$$

Being for this system A a nilpotent matrix ($A^2 = 0$), the Controllability Gramian G can be computed explicitly as

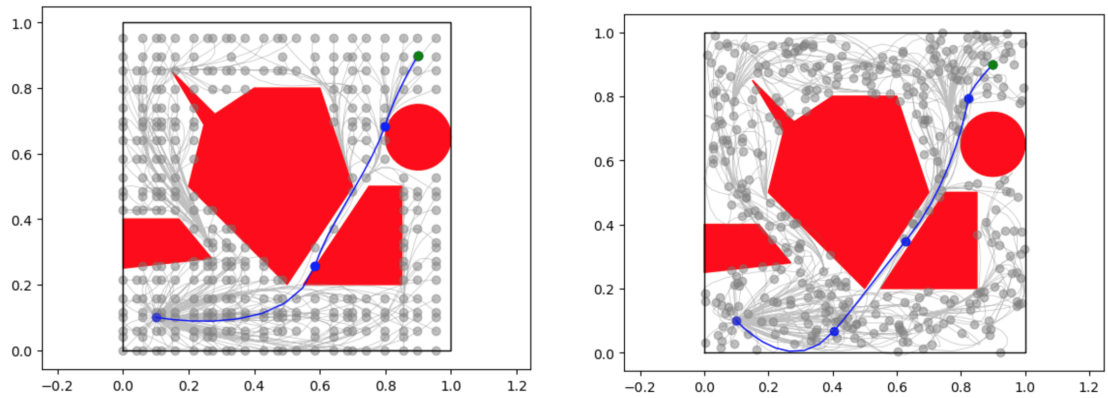
$$G[t] = \int_0^t (I + A\tau) B R^{-1} B^T (I + A^T \tau) d\tau.$$

In particular, taking $R = I$, G assumes the simple closed form expression:

$$G[t] = \begin{bmatrix} I\frac{t^3}{3} & I\frac{t^2}{2} \\ I\frac{t^2}{2} & It \end{bmatrix}. \tag{7.38}$$

The two tables compare the cost of the path returned by the deterministic and probabilistic sampling based algorithm (c_{det} and c_{prob}) varying the the number of samples N

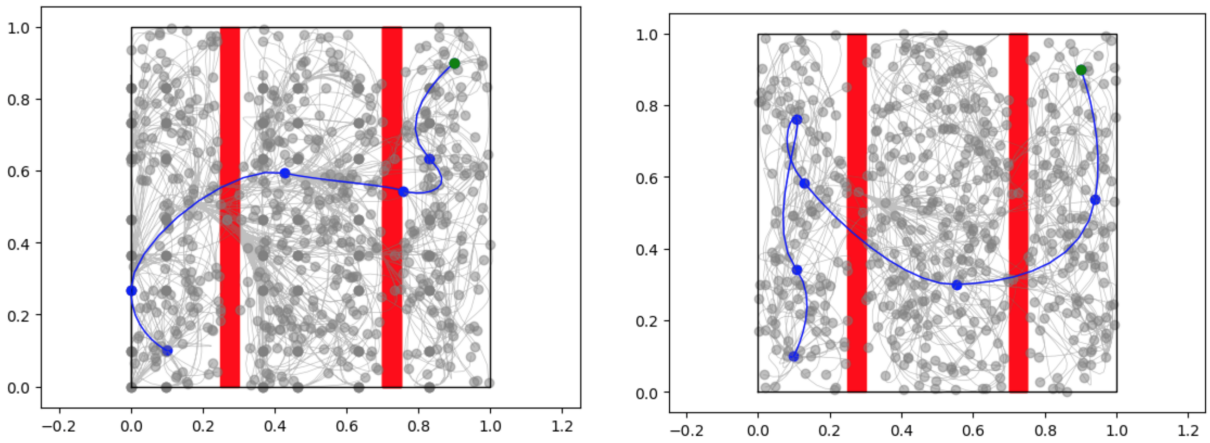
[1] The $DPRM^*$ algorithm, see [4] fore a detailed description of the algorithm and its implementation.



(a) *Deterministic samples:* $N = 625$, $r = 2.5$, $c_{det} = 4.41$.

(b) *iid random samples:* $N = 625$, $r = 2.5$, $c_{prob} = 5.08$.

Figure 7.1: Experimental results for the 2-dimensional double integrator.



(a) *Deterministic samples:* $N = 704$, $r = 2.0$, $c_{det} = 8.53$.

(b) *iid random samples:* $N = 704$, $r = 2.0$, $c_{prob} = 9.61$.

Figure 7.2: Experimental results for the 3-dimensional double integrator: Plan view of the 3-dimensional obstacles set. A narrow window is present on each one of two the red walls.

N	r	c_{det}	c_{min}	c_{max}	c_{average}	c_{median}	Failure Rate
361	2.0	6.07	5.01	7.24	6.23	6.68	0%
361	1.5	6.38	5.59	10.89	7.65	7.74	0%
361	1.2	7.79	6.64	·	·	9.6	23.5%
361	1.0	9.28	7.87	·	·	Fail	90.9%
841	1.5	4.87	4.51	7.45	5.57	5.34	0%
841	1.2	4.91	4.38	8.13	6.06	5.40	0%
841	1.0	7.09	5.27	·	·	8.19	7.1%
1089	1.5	4.61	4.28	6.89	5.36	5.26	0%
1089	1.2	4.64	4.47	6.99	5.62	5.79	0%
1089	1.0	6.43	4.89	11.08	7.71	7.61	0%

Table 7.1: Simulation results for the 2-dimensional double integrator.

N	r	c_{det}	c_{min}	c_{max}	c_{average}	c_{median}	Failure Rate
1000	2.0	6.56	6.69	9.77	8.03	8.18	0%
1000	1.8	6.56	6.13	11.65	9.14	8.96	0%
1000	1.5	7.19	10.03	·	·	14.31	20.0%
1000	1.4	6.9	12.72	·	·	Fail	66.7%
512	2.0	7.00	7.43	12.48	9.76	9.58	0%
512	1.8	7.15	9.92	14.66	12.21	12.64	0%
512	1.6	9.92	8.76	·	·	15.98	44.4%

Table 7.2: Simulation results for the 3-dimensional double integrator.

and the cost threshold r. Since the output of the probabilistic algorithm may differ at different trials, we report the average and the median value of c_{prob} (assuming that at a failure corresponds a cost of ∞).

The results presented clearly highlight the statistically superior performances of the deterministic sampling over its probabilistic counterpart in term of cost and failure rate.

Chapter 8

Driftless Control Affine Dynamical Systems

In this section we try to replicate the approach already used for systems with linear affine dynamics, i.e. we want to prove that deterministic versions of the results provided in [5] hold if an appropriate deterministic sampling scheme is considered in place of the iid random samples. Given the remarkable mathematical complexity of the general case, this work will focuses on the particular case of the Reeds-Shepp Car, with the hope of gaining some insights to deal with the general problem.

The structure of the chapter reflects that of Chapter 7:

After formally defining the motion problem and recalling some results from Sub-Riemannian Geometry (instrumental our analysis), the original contribution of the work is introduced. In section 8.2 we present a novel dispersion-like parameter Φ_N, specifically thought to describe how well the w-weighted boxes Box^w centered in each sample give an effective coverage of the mainfold defining the state space of the system.

In section 8.3 we prove the main theoretical result of the chapter i.e. Theorem 8, characterizing the asymptotic behavior of algorithms based on low-dispersion sampling schemes. This new theorem can be considered as a deterministic version of Theorem IV.5 in [5].

In section 8.4 we give an explicit description of how to build the deterministic sampling set for the Reeds-Shepp car system. There are no analogous-of this sampling method in the literature, the best of our knowledge.

Finally section 8.5 provides a survey of the experimental results obtained in a simulation environment.

8.1 Background Material

We provide now some definitions and a brief review of key results in differential geometry, on which we will rely extensively later in the paper. For further references on the geometry of non-holonomic systems the reader can refer to [6],[5],[13].

Let $\mathcal{X} \subset \mathbb{R}^n$ be the manifold defining a configuration space. Within this space let us

consider driftless control-affine (DCA) dynamical systems of the form

$$\dot{\mathbf{x}}[t] = \sum_{i=1}^{m} \mathbf{g}_i[\mathbf{x}(t)]\mathbf{u}_i[t], \quad \mathbf{x} \in \mathcal{X}, \quad \mathbf{u} \in \mathcal{U} \tag{8.1}$$

where the available motions of trajectories $\mathbf{x}[t]$ are linear combinations given by input control functions $\mathbf{u}_i[t]$ and their corresponding actions at each point in the space $\mathbf{g}_i[\mathbf{x}]$. We shall assume in this paper that $\mathbf{g}_1, ..., \mathbf{g}_m$ are smooth vector fields on $\mathcal{X}$, and that the control set $\mathcal{U} \subset \mathbb{R}^m$ is closed and bounded. We also assume $\mathcal{U}$ is symmetric about the origin so that the system is time-reversible and $\mathbf{0}$ is in the interior of the convex hull of $\mathcal{U}$. This last condition ensures that the local possibilities for motion at each point appear as a linear space spanned by the $\mathbf{g}_i$, a fact essential to the forthcoming controllability discussion.

The arc length of a path $\mathbf{x}[t]$ is given by $l(\mathbf{x}) = \int_0^T ||\dot{\mathbf{x}}[t]||dt$. The arc length function induces a sub-Riemannian distance on $\mathcal{X}$ defined by:

$$d(\mathbf{x}_1, \mathbf{x}_2) := \inf_{\mathbf{x}} l(\mathbf{x}), \tag{8.2}$$

where the infimum is taken over all the dynamically feasible trajectories starting in $\mathbf{x}_1$ and ending in $\mathbf{x}_2$.

The main hurdle in this case, with respect to systems with linear affine dynamics, is that the shape and size of the sub-Riemmanian balls:

$$B(\mathbf{x}_0, \varepsilon) := \{\mathbf{x}| \quad d(\mathbf{x}_0, \mathbf{x}) < \varepsilon\} \tag{8.3}$$

may potentially change from point to point on $\mathcal{X}$, while the shape and dimension of the perturbation balls $\Delta[\mathbf{x}_0, \tau, r]$ were fixed for any fixed time τ.

At this point a series of results regarding system controllability are presented following the discussion in [6] and [13]. As noted above, the vector fields $\mathbf{g}_1, ..., \mathbf{g}_m$ characterizing the system represent a set of possible motions for a trajectory within $\mathcal{X}$. More precisely at each point $p \in \mathcal{X}$ the vectors $\mathbf{g}_1[p], ..., \mathbf{g}_m[p]$ span a linear subspace of local directions within the tangent space $T_p\mathcal{X}$.

Let $\mathcal{L} = \mathcal{L}(\mathbf{g}_1, ..., \mathbf{g}_m)$ be the distribution, equivalently the set of local vector subspaces, generated by the vector fields $\mathbf{g}_1, ..., \mathbf{g}_m$. We define recursively $\mathcal{L}_1 = \mathcal{L}$, $\mathcal{L}_{k+1} = \mathcal{L}_k + [\mathcal{L}_1, \mathcal{L}_k]$ where $[\mathcal{L}_1, \mathcal{L}_k] = \text{Span}\{[Y, Z] : Y \in \mathcal{L}_1, Z \in \mathcal{L}_k\}$. Then $\mathcal{L}_k$ is the distribution generated by the iterated Lie brackets of the $\mathbf{g}_i$ with k terms or fewer. The Lie hull of $\mathcal{L}$ is $\text{Lie}(\mathcal{L}) := \bigcup_{k \geq 1} \mathcal{L}_k$. Let $\mathcal{L}_k(\mathbf{x})$ denote the vector space corresponding to $\mathbf{x} \in \mathcal{X}$ in $\mathcal{L}_k$.

The vector fields $\mathbf{g}_1, ..., \mathbf{g}_m$ are said to be *bracket generating* if $\text{Lie}(\mathcal{L})(\mathbf{x}) = T_{\mathbf{x}}\mathcal{X}$ for all $\mathbf{x} \in \mathcal{X}$. This requirement is also referred to as *Chow's condition* and means that arbitrary local motion may be achieved by composing motions along the control directions $\mathbf{g}_i$. In fact, provided that the $\mathbf{g}_i$ are bracket generating, any two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ may be connected by a trajectory satisfying the dynamics of the system.

Using the bracket-generating assumption we select a local orthonormal frame for $T_{x_0}\mathcal{X}$ of vector fields $Y_1, ..., Y_n$ as follows: the set $\{Y_1 = \mathbf{g}_1, ..., Y_{n-1} = \mathbf{g}_m\}$ spans $\mathcal{L}$ near $\mathbf{x}_0$; $\{Y_1, ..., Y_{n_2}\}$ spans $\mathcal{L}_2$ near $\mathbf{x}_0$; $\{Y_1, ..., Y_{n_3}\}$ spans $\mathcal{L}_3$ near $\mathbf{x}_0$; and so on.

Define the weights $w_i = k$ if $Y_i(\mathbf{x}_0) \in \mathcal{L}_k(\mathbf{x}_0)$ and $Y_i(\mathbf{x}_0) \notin \mathcal{L}_{k-1}(\mathbf{x}_0)$. Applying a procedure developed in [6], the coordinate system y_i corresponding to this local frame

may be transformed into a privileged coordinate system z_i by a polynomial change of coordinates. Given privileged coordinates z_i, define the *pseudonorm* at $\mathbf{x}_0$ as

$$||z(\mathbf{x})||_{\mathbf{x}_0} := \max_i \{|z_i|^{1/w_i}\}. \tag{8.4}$$

Using this pseudonorm we define the w-weighted box of size ε at $\mathbf{x}_0$ as the point set $\mathrm{Box}^w(\mathbf{x}_0, \varepsilon) := \{z \in \mathbb{R}^n : ||z||_{\mathbf{x}_0} \leq \varepsilon\}$. We can now state the *Ball-Box Theorem* ([6]):

Theorem 7. *Fix a point $\mathbf{x}_0 \in \mathcal{X}$ and a system of privileged coordinates $z_1, ..., z_n$ at $\mathbf{x}_0$. Then there exist positive constants $a(\mathbf{x}_0), A(\mathbf{x}_0) > 0, \sigma(\mathbf{x}_0) > 0$ such that for all $\mathbf{x}$ with $d(\mathbf{x}_0, \mathbf{x}) < \sigma(\mathbf{x}_0)$,*

$$a(\mathbf{x}_0)||z(\mathbf{x})||_{\mathbf{x}_0} \leq d(\mathbf{x}_0, \mathbf{x}) \leq A(\mathbf{x_0})||z(\mathbf{x})||_{\mathbf{x}_0}. \tag{8.5}$$

It can be shown that there exists a continuously varying system of privileged coordinates on $\mathcal{X}$ so that the inequality (3) holds at all $\mathbf{x}_0$ for continuous positive functions $a(\cdot), A(\cdot)$, and $\sigma(\cdot)$ on $\mathcal{X}$. Let us assume that the system is sufficiently regular such that there exist bounds $0 < a_{\min} \leq a(\mathbf{x}) \leq A(\mathbf{x}) \leq A_{\max} < \infty$ and $\sigma(\mathbf{x}) \geq \sigma_{\min} > 0$ for all $\mathbf{x} \in \mathcal{X}$.

8.2 The quest for possible sampling schemes

The goal of this section is to find a deterministic sampling scheme $\mathcal{S}^*[N]$ suited for the particular class of dynamic systems we are considering.

Ideally speaking, $\mathcal{S}^*[N]$ should be an analogous of the sampling sets $\mathcal{S}_{\tau_N}[N]$ described in section 2, in the sense that it should arrange the N samples in a way that minimizes the radius ε_N of the largest empty Sub-Riemannian ball.

Unfortunately, the determination of such a set is challenging for two main reasons:

1. The exact determination of the Sub-Riemannian ball of a given radius in a given point can be hard, and probably unfeasible from a computational point of view.

2. Shape, orientation and size of the balls, for a given radius, change from point to point.

However, the first problem (an a part of the second one) can be bypassed thanks to the *Ball-Box Theorem*. Since by Theorem 4:

$$\mathrm{Box}^w(\mathbf{x}_0, \varepsilon/A_{\max}) \subseteq B(\mathbf{x}_0, \varepsilon), \tag{8.6}$$

for all $\mathbf{x}_0 \in \mathcal{X}$ and for ϵ sufficiently small, we can instead define $\mathcal{S}^*[n]$ as the set that minimize the radius ε_n of the greatest empty w-weighted box $\mathrm{Box}^w(\mathbf{x}_0, \varepsilon)$. The advantage in this approach lies in the fact that, as stated in [7], the shape and orientation of these boxes can be efficiently computed for a large class of dynamical system. But again, since the orientation of the boxes change from point to point in a way that depends inherently on the characteristics of the particular systems studied, it is practically impossible to give an explicit description of $\mathcal{S}^*[N]$ for the general case.

In order to understand how to build the set $\mathcal{S}^*[N]$ for a particular DCA system, consider the functional $\Phi_N : \mathcal{X}^N \longrightarrow \mathbb{R}^+$, defined as:

$$\Phi_N[\mathcal{S}] := \max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{s}\in\mathcal{S}} \max_{1\leq i\leq d} |\mathbf{L}_i(\mathbf{s}) \cdot (\mathbf{x} - \mathbf{s})|^{1/w_i}. \tag{8.7}$$

Where the vector fields $\mathbf{L}_i[\mathbf{x}]$ represent the unit versors associated to the privileged coordinate axes in $\mathbf{x}$. Φ_N represents the the radius of the largest empty box in $\mathcal{X}$ and can be interpreted as a pseudonorm-induced dispersion measure of the set $\mathcal{S}$.

However, the determination of the optimal set (if such a set exists):

$$\mathcal{S}^*[N] = \text{argmin}_{\mathcal{S}\in X^N}\Phi_N[\mathcal{S}]. \tag{8.8}$$

is not strictly necessary for our aim. Indeed, a family of sets satisfying the relation

$$\Phi_N[\mathcal{S}_N] \leq \gamma N^{-1/D}, \tag{8.9}$$

for a fixed real constant γ, where $D = \sum_{i=1}^{d} w_i$, would be sufficient to extend the theoretical results already obtained for the system with linear affine dynamics to the new class of systems under consideration.

8.3 Deterministic Exhausticity

Provided that such a family of sets with the properties indicated in the previous section exists, it is possible to prove deterministic exhaustivity, analogously to what was done for the system with linear affine dynamics:

Theorem 8. *Let Σ be a system satisfying the assumptions A_Σ and suppose $\pi = (\mathbf{x}, \mathbf{u}, T)$ being a dynamically feasible trajectory with strong δ-clearance, $\delta > 0$. Let $n \in \mathbb{N}$, $\varepsilon > 0$, and consider a set of nodes $V := \{\mathbf{x}[0]\} \cup \mathcal{S}[N]$, where $\Phi_N[\mathcal{S}_N] \leq \gamma N^{-1/D}$. Then for sufficiently large N, exist waypoints $\{\mathbf{y}_k\}_{k=0}^{K} \subset V$ which (ε, r_N)-trace π, where $r_N = \omega(N^{-1/D})$, where $D = \sum_{i=1}^{d} w_i$.*

Proof. Neglecting the trivial case for $c[\pi] = 0$, we repeat the same construction made in [5, Theorem IV.5]. If we are able to prove that at least a sample fall in each one of the balls $B^{\beta}_{n,m}$ ($\beta = \varepsilon/2$), we have done. But, since by hypothesis, exists a constant $\gamma > 0$ such that there is at least a sample $\mathbf{s}$ of $\mathcal{S}$ in each box $\text{Box}^w(\mathbf{x}_0, \rho_n)$ with $\rho_N = \gamma N^{-1/D}$, it is sufficient to prove that, for sufficiently hight N, $\text{Box}^w(\mathbf{x}_m, \rho_n) \subseteq B^{\beta}_{n,m}$.

Remember that, for Theorem 4

$$\text{Box}^w(\mathbf{x}_0, \varepsilon/A_{\max}) \subseteq B(\mathbf{x}_0, \varepsilon),$$

for all $\mathbf{x}$ with $||\mathbf{x}_0 - \mathbf{x}|| \leq \sigma_{\min}$. Since $\rho_N \to 0$ for $N \to \infty$, exists $\bar{N} \in \mathbb{N}$ such that $\rho_N < \sigma_{\min} \quad \forall N > \bar{N}$. Hence, for $N > \bar{N}$ we have that:

$$\text{Box}^w(\mathbf{x}_m, \rho_N) \subseteq B(\mathbf{x}_m, A_{\max}\rho_N). \tag{8.10}$$

At this point we only need to prove that

$$B(\mathbf{x}_m, A_{\max}\rho_N) \subseteq B^{\beta}_{n,m} \tag{8.11}$$

or, equivalently

$$A_{\max}\rho_N \leq \ \beta r_N/4, \tag{8.12}$$

again, for N large enough.

The last inequality holds iff

$$r_N \geq K(\varepsilon) n^{-1/D}, \tag{8.13}$$

for

$$K(\varepsilon) = 8 A_{\max} \gamma / \varepsilon. \tag{8.14}$$

But, no matter how big $K(\varepsilon)$ is, (8.13) is true for N sufficiently large, by the definition of r_n. □

Again, asymptotic optimality can be easily proved starting from deterministic exhaustivity, as in Chapter 7. Notice that results similar to Theorem 8 hold for many deterministic sampling schemes but with different requirements on the cost treeshold r_N. For example, the classic low L_2-dispersion sampling sequences adopted in Chapter 6 requires

$$r_N = \omega(N^{-1/sn}), \tag{8.15}$$

where

$$s = \max_{1 \leq i \leq n} w_i \tag{8.16}$$

is the so called *non-holonomy degree* of the systems. This property however is not completely satisfactory because $sn \geq D$, where the equality holds only in the trivial case $s = 1$, i.e. when $m = n$ and

$$\text{span}(\mathbf{g}_1(\mathbf{x}), ..., \mathbf{g}_m(\mathbf{x})) = \mathcal{T}_{\mathbf{x}}\mathcal{X}. \tag{8.17}$$

In the following section we shows how to build a low dispersion set $\mathcal{S}$ for the particular case of the Reeds-Shepp Car.

8.3.1 Reeds-Shepp Car

Given any configuration $\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ of the Reeds-Shepp car, the vector fields $\mathbf{L_1}(\mathbf{x}) = [\cos\theta, \sin\theta, 0]^T$ and $\mathbf{L_2}(\mathbf{x}) = [0, 0, 1]^T$ span the distribution induced by its dynamics. The Lie bracket of these two vector fields can be calculated as

$$\mathbf{L_3}(\mathbf{x}) = [\mathbf{L_1}, \mathbf{L_2}] = [-\sin\theta, \cos\theta, 0]^T. \tag{8.18}$$

Notice that the vector fields $\{\mathbf{L_1}, \mathbf{L_2}, \mathbf{L_3}\}$ span the whole tangent space, satisfying Chow's condition. Then, Chow's theorem merely tells us that, given two configurations of the Reeds-Shepp car, there exists a dynamically-feasible trajectory, i.e., a horizontal curve, that joins the two. The weights vector for this system is $\mathbf{w} = [1, 1, 2]$ and $D = 4$.

A possible way to place N samples along the state space $\mathcal{X} = [0; 1]^2 \times [-\pi; \pi]$ (this assumption is just explicative) so that $\Phi_N[\mathcal{S}] \leq \varepsilon$, is the following:

Fixed an integer N_2 , consider N_2 equally spaced planes on the configuration space $\mathcal{X}$, $\theta = k\varepsilon^{w_2}$, $k = 0, ..N_2 - 1$. Then, for each of these planes, e.g. $\theta = \bar{\theta}$, we build a grid oriented as the local orthonormal axes $\mathbf{L_1}(\bar{\theta}), \mathbf{L_3}(\bar{\theta})$. This is possible, because these axes do not change orientation along the plane $\theta = \bar{\theta}$. The step of the grid in the direction of $\mathbf{L_1}(\bar{\theta})$ is fixed as ε^{w_1}, while the step in the direction of $\mathbf{L_3}(\bar{\theta})$ is chosen to be ε^{w_3}. Let N_1, N_3 be the numbers of rows and columns in the grid. Although N_1 and N_3 depend on the particular plane considered, we certainly have $N_i \sim 1/\varepsilon^{w_i}$. Evidently, $\mathcal{S}$ satisfies the condition $\Phi_n[S] \leq \epsilon$. Moreover, the number of points required to build S is given by:

$$N = N_1 N_2 N_3 \sim 1/\varepsilon^D, \tag{8.19}$$

or equivalently $\varepsilon \sim N^{-1/D}$, as we wanted.

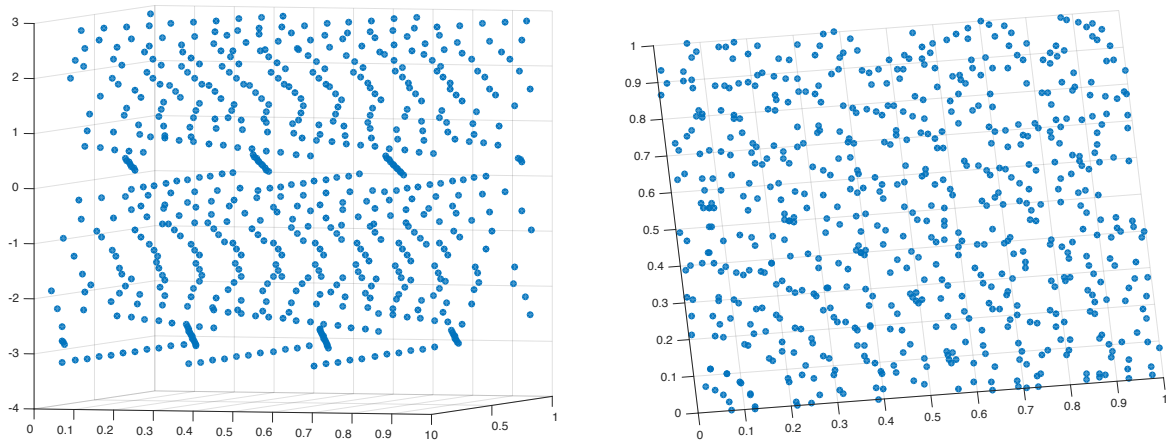


Figure 8.1: 3-dimensional representation of the Reeds-Shepp Car sampling set $\mathcal{S}$, for $N = 630$ samples, realized with Matlab.

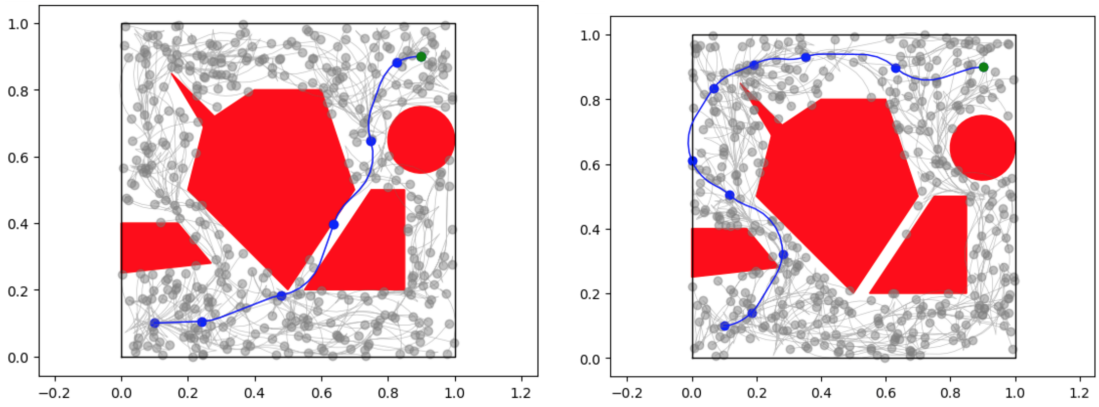
N	r	c_{det}	c_{min}	c_{max}	c_{average}	c_{median}	Failure Rate
630	0.4	1.29	1.32	1.77	1.56	1.658	0%
630	0.3	1.33	1.38	1.9	1.63	1.64	0%
630	0.2	1.51	1.85	·	·	1.85	7.69%
630	0.15	2.60	·	·	·	Fail	100%
1280	0.2	1.29	1.44	1.96	1.80	1.87	0%
1280	0.15	1.94	2.05	3.06	2.46	2.43	0%
1280	0.12	2.81	·	·	·	Fail	100%
1936	0.15	1.43	1.51	2.21	1.93	1.95	0%
1936	0.12	2.07	2.09	·	·	3.35	35.7%
1936	0.1	3, 84	·	·	·	fail	100%

Table 8.1: Simulation results for the Reeds-Shepp Car.

8.4 Experimental Results

Following the approach already adopted for the double integrator, simulations for the Reeds-Shepp Car are done twice, using the same algorithm with the same cost threshold (parameter r) in both the cases, and just changing the sampling scheme adopted.

The $DPRM^*$ algorithm ([5]) were implemented in Julia and run using a Unix operating system with a 2.0 GHz processor and 8 GB of RAM. Near neighbor sets were precomputed and cached at algorithm initialization after the sample set was selected. Note that for batch-processing (as opposed to anytime) algorithms such as $DFMT^*$ and $DPRM^*$, one can precompute both near neighbor sets and sub-Riemannian distance ahead of time, as they do not depend on the obstacle configuration. The price to pay is a moderate increase in memory requirements.



(a) *Deterministic samples:* $N = 630$, $r = 0.2$, $c_{det} = 1.33$.

(b) *iid random samples:* $N = 630$, $r = 0.2$, $c_{prob} = 1.88$.

Figure 8.2: Experimental results for the Reeds-Shepp Car.

Chapter 9

Conclusion

Bibliography

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

[2] L. Janson, B. Ichter, M. Pavone, *Deterministic Sampling-Based Motion Planning: optimality, Complexity, and Performance.*

[3] D. J. Webb, J. van der Berg, *Kinodynamics RRT*: Asymptotically Optimal Motion Planning for Robots with Linear Dynamics.* 2013 IEEE International Conference on Robotics and Automation (ICRA).

[4] E. Schmerling, L. Janson, M. Pavone, *Optimal Sampling-Based Motion Planning under Differential Constraints: the Drift Case with Linear Affine Dynamics.*

[5] E. Schmerling, L. Janson, M. Pavone, *Optimal Sampling-Based Motion Planning under Differential Constraints: the Driftless Case.*

[6] A. Bellaiche, *Sub- Riemannian Geometry.* Eds. Birkhauser, 1996.

[7] S. Karaman, E. Frazzoli, *Sampling-based Optimal Motion Planning for Non-holonomic Dynamical Systems.*

[8] F. L. Lewis and V. L. Syrmos, *Optimal control.* New York: Wiley, 1995.

[9] G. Goretkin, A. Perez, R. Platt Jr., G. Konindaris, *Optimal Sampling-Based Planning for Linear-Quadratic Kinodynamic Systems.*

[10] T. Caldwell, N. Correl, *Fast Sample-Based Planning for Dynamic Systems by Zero-Control Linearization-Based Steering.*

[11] A.Perez, R. Platt Jr., G. Konindaris, L. Kaelbling, T. Lozano-Perez, *LQR-RRT*: Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics.* 2012 IEEE International Conference on Robotics and Automation.

[12] A. Yershova, S. M. LaValle, *Deterministic Sampling Methods for Spheres and* SO(3).

[13] R. Montgomery, *A tour of subriemannian geometries, their geodesics, and applications.* Mathematical Surveys and Monographs. American Mathematical Society, 2002, vol. 91.

[14] E. Glassman, R. Tedrake, *A Quadratic Regulator-Based Heuristic for Rapidly Exploring State Space.*

[15] A. Shkolnik, M. Walter, R. Tedrake *Reachability-Guided Sampling for Planning Under Differential Constraints.* 2009 IEEE International Conference on Robotics and Automation.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms (2nd Ed.). MIT Press, Cambridge, MA, 2001.