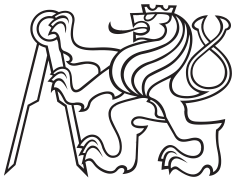
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Sampling-based Motion Planning under Constraints

Petr Zahradník

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
Field of study: Open informatics
May 2022

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 20. May 2022

Signature

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. May 2022

podpis autora práce

Abstract

Path planning is a widespread problem ranging from robotics to biochemistry. One of the recent breakthroughs in this field is a solution using random sampling. This work focuses on using random trees in the motion planning problem with constraints, where regular space sampling algorithms fail. A simple parametrization is proposed, further simplifying the problem formulation for a wide range of problems. A state-of-the-art algorithm, AtlasRRT, is thoroughly analyzed and implemented in Julia. An improvement is proposed that addresses missing details in the original paper and leads to better overall performance. The implementation is benchmarked on multiple tasks from robotics. Visualizations and comparisons with other contemporary algorithms are provided.

Keywords: path planning, planning under constraints, random trees, atlas

Supervisor: Ing. Vojtěch Vonásek, Ph.D.

Abstrakt

Plánování cesty je rozšířený problém sahající od robotiky po biochemii. Jedním z nedávných průlomů v této oblasti je řešení pomocí náhodného vzorkování. Tato práce se zabývá použitím náhodných stromů v úloze plánování pohybu s omezeními, kde běžné vzorkovací algoritmy selhávají. Součástí je návrh jednoduché parametrizace, která zjednodušuje zadání úlohy pro široké spektrum použití. Efektivní algoritmus AtlasRRT je v této práci důkladně zanalyzován a implementován v Julii. Dále navrhujeme zlepšení algoritmu řešící detaily, které v původním článku zcela chybí a přispějí k celkovému zrychlení běhu algoritmu. Implementace je dále testována na různých úlohách z oblasti robotiky. Přiloženy jsou vizualizace a porovnání s dalšími současnými algoritmy.

Klíčová slova: plánování cesty, plánování s omezeními, náhodné stromy, atlas

Překlad názvu: Vzorkovací metody plánování pohybu s omezeními

Contents

Figures

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Zahradník Petr** Personal ID number: **492286**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Sampling-Based Motion Planning under Constraints

Bachelor’s thesis title in Czech:

Vzorkovací metody plánování pohybu s omezeními

Guidelines:

1. Get familiar with robotic motion planning, particularly with sampling-based methods, e.g., RRT and PRM [1]. Get familiar with motion planning under constraints [1,2,3,4].
2. Select a suitable motion planner for planning under constraints, e.g., Atlas-RRT [2,3], and implement it.
3. Analyze the projection of states to the tangent space of the constraint manifold and design improvements (e.g., to speed it up or to increase success rate).
4. Apply methods from 2) and 3) to suitable robotic scenarios, e.g., planar manipulator, robotic manipulator, closed kinematic chain, a group of drones.
5. Experimentally verify all implemented methods and compare with suitable planners from the OMPL benchmark [4].

Bibliography / sources:

[1] LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.
[2] Kingston, Zachary, Mark Moll, and Lydia E. Kavraki. "Sampling-based methods for motion planning with constraints." Annual review of control, robotics, and autonomous systems 1 (2018): 159-185.
[3] Jaillet, Léonard and Josep M. Porta. “Path Planning with Loop Closure Constraints Using an Atlas-Based RRT.” ISRR (2011).
[4] Jaillet, Léonard, and Josep M. Porta. "Path planning under kinematic constraints by rapidly exploring manifolds." IEEE Transactions on Robotics 29.1 (2012): 105-117.
[5] Mark Moll, Ioan A. Şucan, Lydia E. Kavraki, Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization, IEEE Robotics & Automation Magazine, 22(3):96–102, September 2015. doi: 10.1109/MRA.2015.2448276.

Name and workplace of bachelor’s thesis supervisor:

Ing. Vojtěch Vonásek, Ph.D. Multi-robot Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **24.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Vojtěch Vonásek, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

List of Notations

Symbol	Meaning
$\boldsymbol{v}$	Vector
$\|\|\boldsymbol{v}\|\|$	Norm of vector $\boldsymbol{v}$
$[a; b)$	Interval $\{x \in \mathbb{R} \mid a \leq x < b\}$
$\text{atan2}(y, x)$	2-argument arctangent, defined as in FORTRAN IV library
$\mathbf{0}$	Zero vector
$\boldsymbol{I}$	Identity matrix
n	Dimension of the ambient space
k	Constraint count
$\mathcal{C}$	Configuration space
$\mathcal{S}$	State space
x_s	Start configuration
x_g	Goal configuration
$\boldsymbol{x}$	Point in the ambient space $\mathbb{R}^n$
$\mathcal{C}_i$	Chart number i
$\boldsymbol{u}^i$	Point in the tangent space $\mathbb{R}^{n-k}$ of the chart $\mathcal{C}_i$
Φ_i	Basis matrix of chart $\mathcal{C}_i$
$\phi_i(\boldsymbol{u}^i)$	Transformation of coordinates of the point $\boldsymbol{u}^i$ from the chart basis Φ_i to the ambient coordinate basis
$\psi_i(\boldsymbol{u}^i)$	Projection of a point $\boldsymbol{u}^i$ from the tangent space of $\mathcal{C}_i$ onto the manifold
$\mathcal{V}_i$	Validity area of the chart $\mathcal{C}_i$
$\mathcal{F}_i$	Set of constraints for the chart $\mathcal{C}_i$

Chapter 1

Introduction

In this work, we will be solving the path planning problem. Path planning is an essential tool in robotics and has many applications. Examples include automatic vacuum cleaner navigation [1], CNC drill control [2], drone formation guidance [3], and many more.

Path planning is the first part of motion planning. A start and a goal are given, and the algorithm must find a feasible way between them while accounting for obstacles and constraints along the way. The resulting path can be later optimized, smoothed, and used as a reference trajectory for motion control.

The methods used in this work are helpful in robotic manipulator planning, which will be our main concern. However, the application of motion planning is much broader, thanks to the straightforward task specification we will present.

Planning a path for a multi-body robot around obstacles (also known as the *generalized mover's problem*) is a well-studied topic known to be PSPACE-hard [4]. Many algorithms can solve a path planning problem, such as RRT [5], which can bee seen in the Figure 1.1.

We will be concerned with a more challenging problem of planning under constraints, where only a handful of algorithms exist. We will show the importance of constraint-based planning in manipulator planning and study one of the state-of-the-art algorithms — AtlasRRT [6]. Our contribution will consist of solving implementation details regarding the manifold covering.

The original AtlasRRT implementation relies on the transition between neighboring charts, but the corresponding paper does not mention how to do so. Naïve implementation of such an algorithm would not work due to disparities between neighboring charts. This work analyzes the problem and proposes multiple solutions to ensure no point falls into a hole while keeping a minimal overhead on the total solve time.

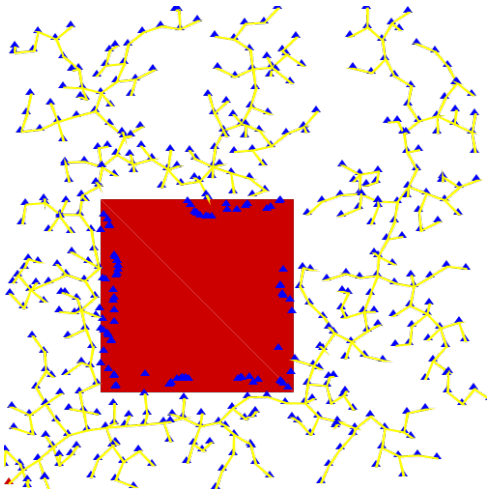
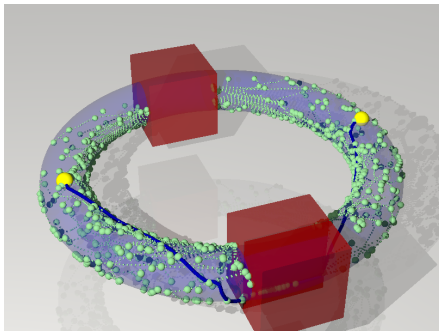


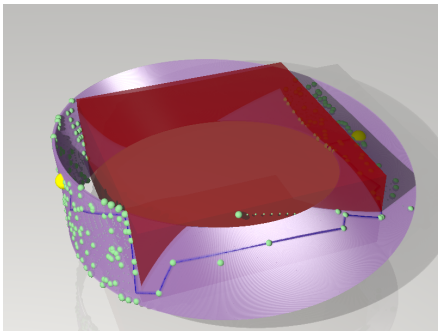
Figure 1.1: The RRT algorithm searches through the free space around the obstacle (red) and builds a tree of states (blue) starting from the bottom left.

To ensure that the implementation is correct, we tested it on a variety of problems: geometric shape traversal (Figure 1.2a), planning on algebraic surfaces (Figure 1.2b), planar robotic manipulator (Figure 1.2c), and a 3D manipulator (Figure 1.2d). Exhaustive testing in both low- and high-dimensional state spaces found no projection issues, which confirms the theoretical expectations.

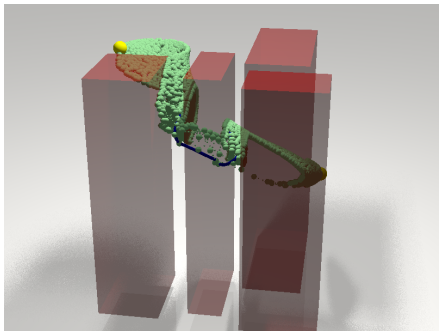
The Open Motion Planning Library (OMPL) is the de facto standard for benchmarking and comparing motion planning algorithms [7]. Since OMPL supports atlas-based planning [8] we were able to run a comparison benchmark against multiple algorithms from OMPL. We achieved similar performance to SBL [9] and BKPIECE1 [10] planners while being faster than OMPL RRT implementation over the atlas projection space.



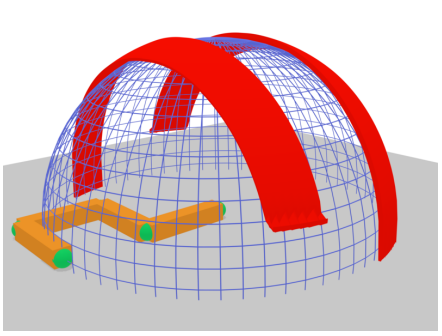
(a) : Navigating around obstacles while staying on the surface of a torus. The found path is in blue.



(b) : Path planning on a part of a cubic curve known as the Möbius strip. A path is found around the obstacle.



(c) : Cross-section of planar manipulator state space with many sampled states (green).



(d) : Path planning for a robotic manipulator (orange) with obstacles (red) and constraints (blue).

Figure 1.2: Four examples of motion planning problems we will be solving in this work.

Chapter 2

Task formulation

2.1 Configuration space

Our description of the path planning task consists of configuration parameters. These numbers reflect the current configuration of the system. For example, a robotic arm manipulator can be described using the current angle orientation of its joints. The vector of all the configuration parameters is called the *configuration*. All configurations form a *configuration space* $\mathcal{C}$. There are two notable points in the configuration space, the *start configuration* x_s, and the *goal configuration* x_g. There might be physical obstacles in the configuration space, which render some configurations infeasible.

Another possible description of the same robotic arm is using the Cartesian coordinates of its joints and the end effector (also called the *task space*). These two formulations both describe the state of the same system but with different parameters. It might sometimes be more effective to work with one or the other. The parametrizations are linked together via kinematic equations. *Forward kinematics* compute the task coordinates from the joint parameters. *Inverse kinematics* computes the joint parameters from the task coordinates.

The description with task parameters is simple and practical; one can use it to reconstruct the model of the system at any given time, it is helpful for modeling physical obstacles which might obstruct the movement and can be used to steer the algorithm towards the goal configuration [11]. The configuration parameters, on the other hand, provide a straightforward way to calculate a motion plan of all the actuators and may be used for adding movement constraints such as joint angle limits.

2.2 Obstacles and constraints

When looking at the configuration space, we see that not all such points form a valid system state. For example, a robotic manipulator cannot rotate in so that the arm segments cross each other, the vacuuming robot cannot find itself inside a wall, and members of drone formation must keep a constant distance. We have to therefore limit the space by using obstacles and constraints.

An *obstacle* is an area inside $\mathcal{C}$, which consists of invalid configurations. All obstacles form an obstacle space $\mathcal{C}_{obstacle} \subseteq \mathcal{C}$; the rest of the configuration space is $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obstacle}$. For simplicity, we will be only concerned with box obstacles; n-dimensional box consists of $2n$ inequalities, two in each dimension. A configuration is located inside $\mathcal{C}_{obstacle}$ iff at least one inequality for such a box does not hold. An example of such box obstacle is in Figure 2.1a.

A *constraint* is specified in the form of a function $f\colon \mathcal{C} \to \mathbb{R}$. A configuration x is valid iff $f(x) = 0$. We will be considering only continuous and continuously differentiable functions because we will need to calculate the Jacobian of f. If there are multiple constraints, let $F\colon \mathcal{C} \to \mathbb{R}^n$ denote a vector function of them. The constraints are all satisfied at a configuration x iff $F(x) = \mathbf{0}$. For an example of a constraint see Figure 2.1b.

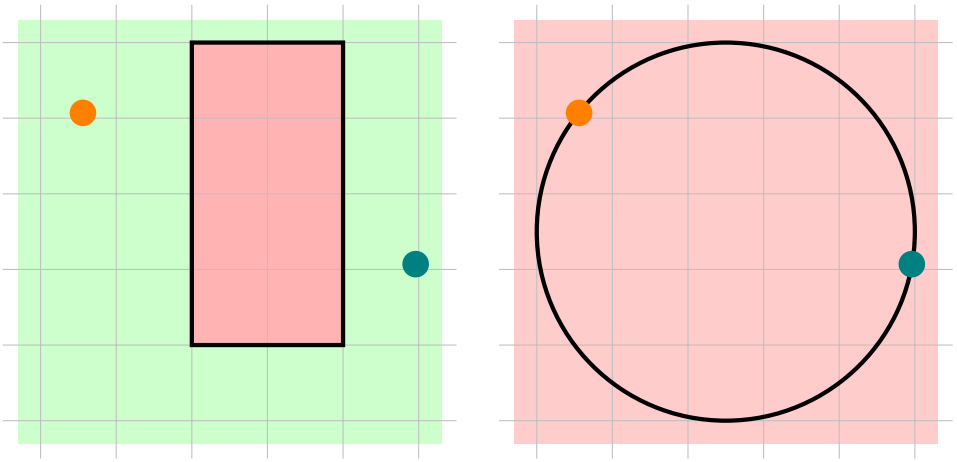
We have defined our constraint to be in the form of equality and to contain a function on the left side. This kind of constraint is called *holonomic constraint*[1]. There are other kinds of constraints that are able to, for example, limit the forces and momentum of the system by using differential equations. Furthermore, we will be concerned only with functions in spatial coordinates, not considering time as an additional dimension.

2.3 State space

Sometimes we would like to use the configuration parametrization and obstacles, and sometimes it is more beneficial to use the configuration space and joint constraints. It might be practical to calculate either forward kinematics or the inverse kinematics. For example, there is no general closed-form solution for a manipulator with 6 degrees of freedom [13] despite that forward kinematics are very simple to calculate. However, there is however a way to include all of this at once in a so-called *state space* $\mathcal{S}$.

A state space combines both the task space and the configuration space. State parameters are simply a vector of the task space parameters followed

[1] More formally, a holonomic constraint can be fully integrated to the form of $f(x, t) = 0$ [12]. The nomenclature comes from the analytical mechanics.



(a) : The red area inside the box shows invalid points $\mathcal{C}_{obstacle}$, all the other configurations shown in green are valid $\mathcal{C}_{free}$.

(b) : Circle constraint, the only valid configurations $f(x) = 0$ lie on the circle, all the other configurations are invalid. This constraint can be drawn in the Cartesian space because it is holonomic.

Figure 2.1: A visualization of obstacles and constraints. Note the difference in the validity area. While an obstacle removes a bounded area, a constraint limits the valid configurations into a subspace of a lower dimension.

by the joint parameters. State obstacles can contain obstacles from either space or both of them at the same time. The space occupied by the obstacles is analogously called $\mathcal{S}_{obstacle}$, the free space is called $\mathcal{S}_{free} = \mathcal{S} \setminus \mathcal{S}_{obstacle}$. State constraints contain constraints from either space with the addition of kinematic constraints, which link the two original spaces together.

The state space can be thought of as a cross-product of the coordinate and joint configurations spaces constrained by the kinematic constraints. We can easily see that this formulation is equivalent to the other two since we kept all the parameters and obstacles.

The kinematic constraint allows us to simplify the constraint formulation. The simple formulation gives us an advantage in path and motion planning since we can immediately access both the joint and task parameters without explicitly evaluating the kinematics. We can simply visualize the robot trajectory and use the joint parameters to plan the motion step by step.

This trick is not novel [14, 15], but we think that it is not stressed enough in the relevant literature. In the following chapters, we will be working with several different models where state space parametrization and kinematic constraints replace the need to calculate inverse (or forward) kinematics analytically.

Example 1: Two segment manipulator formulation

Two dimensional robotic manipulator with segments of fixed lengths lengths l_1 and l_2 is given as seen in Figure 2.2.

Joint parameters:

- θ_1: angle of first joint
- θ_2: angle of second joint

Coordinate parameters:

- x_1: X coordinate of second joint
- y_1: Y coordinate of second joint
- x_2: X coordinate of end effector
- y_2: Y coordinate of end effector

Forward kinematics (inputs θ_1, θ_2):

$$\begin{aligned} x_1 &= l_1 \cos\theta_1 \\ y_1 &= l_1 \sin\theta_1 \\ x_2 &= x_1 + l_2 \cos(\theta_1 + \theta_2) \\ y_2 &= y_1 + l_2 \sin(\theta_1 + \theta_2) \end{aligned} \tag{2.1}$$

Inverse kinematics (inputs x_1, y_1, x_2, y_2):

$$\begin{aligned} \theta_1 &= \text{atan2}(y_1, x_1) \\ \theta_2 &= \text{atan2}(y_2 - y_1, x_2 - x_1) - \theta_1 \end{aligned} \tag{2.2}$$

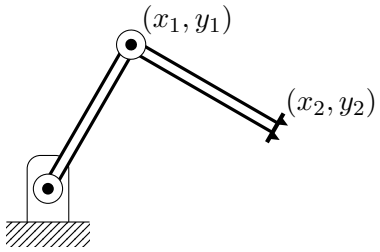
In this example, both forward and inverse kinematics have a closed form solution, so we can choose either one to form our constraints.

Example 2: Two segment manipulator state space

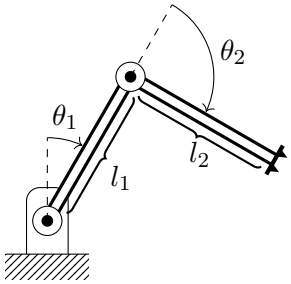
The same manipulator as in Example 1 is given, but now parameterized using the state space.

State parameters:

- x_1: X coordinate of second joint
- y_1: Y coordinate of second joint
- x_2: X coordinate of end effector



(a) : Configuration parametrization uses joint and end-effector coordinates. Points are constrained with segment lengths.



(b) : Joint parametrization uses segment length and joint angles. Segments length are usually fixed and the joint rotate freely.

Figure 2.2: Two different parametrizations of the same robotic manipulator. Note that both parametrizations consist of four parameters and two constraints, leaving two degrees of freedom.

- y_2: Y coordinate of end effector
- θ_1: angle of first joint
- θ_2: angle of second joint

Constraints:

$$\begin{aligned} x_1 &= l_1 \cos\theta_1 \\ y_1 &= l_1 \sin\theta_1 \\ x_2 &= x_1 + l_2 \cos(\theta_1 + \theta_2) \\ y_2 &= y_1 + l_2 \sin(\theta_1 + \theta_2) \end{aligned} \tag{2.3}$$

We see that even though the formulation has six state parameters, there are four constraints which limit the feasible space to two dimensions. This is expected, since our robot has only two degrees of freedom.

2.4 Dimensionality reduction

As we have seen in the previous example, the state space formulation is simple and effective. In the next chapter we will using random trees to solve the path planning. This means we need to sample random points in the space effectively. This is problematic, since the state space has dimension 6, but the feasible points all lie on a 2-dimensional manifold. Another example of planning under constraints is in the Figure 2.3. There is no straightforward way to directly generate valid points. The purpose of this work is to show specialized algorithms to overcome this problem.

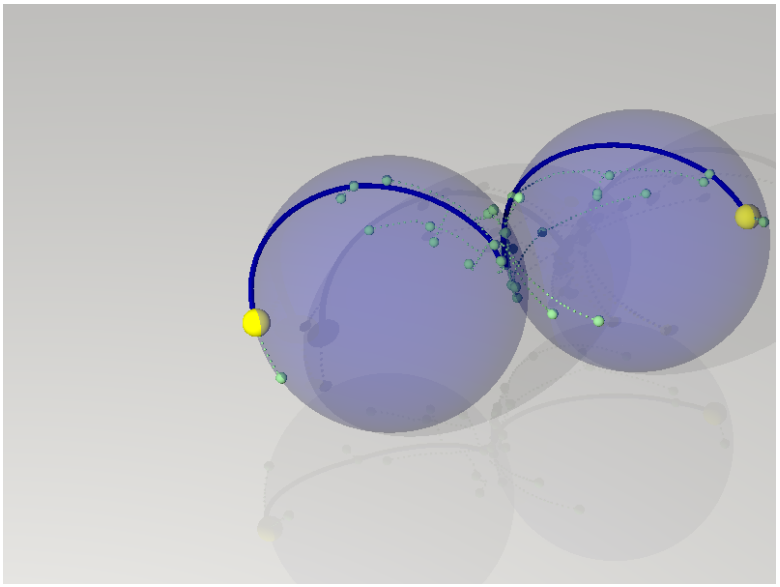


Figure 2.3: Example of a constraint. A feasible path between start and goal states (yellow) must lie on the surface of either of the two blue spheres. Note that despite the ambient space being 3D, the constrained state space is a 2D surface.

Chapter 3

Path planning

Having the task description with the state space parametrization, we need an algorithm to find a path between the start and goal state, such that each point on the path is feasible.

3.1 Feasible path

Definition 1: Path
Let $\mathcal{S} \subseteq \mathbb{R}^n$ be state space, $\mathcal{S}_{obstacle} \subset \mathcal{S}$ be obstacle space, $x_s, x_g \in \mathcal{S}_{free}$ be start and goal states, and $F\colon \mathcal{S} \to \mathbb{R}^k$ be a set of $k < n$ constraints. Path is a continuous function $p\colon [0;1] \to \mathcal{S}$, such that $p(0) = x_s$ and $p(1) = x_g$. Path p is feasible iff $p(x) \in \mathcal{S}_{free}$ and $F(p(x)) = \mathbf{0}$ for all $x \in [0;1]$.

There might be a single unique feasible path for a given problem, infinitely many feasible paths, or none at all. Our goal will be to find any of those paths if at least one exists. Furthermore, we are only interested in non-optimal path planning.

3.2 Constraint relaxation

First of all, let us focus on a relaxed version of the problem with no constraints, only obstacles. All the feasible states lie in the vector space R^n and outside the obstacles. Many algorithms solve the relaxed problem; here are some examples:

- A* discretizes the space into a grid and iteratively searches neighboring nodes using the Breadth-first search (BFS) while discounting nodes that

are near the goal [16], see Figure 3.1a. The performance of A* depends on choosing a suitable heuristic which might be difficult in a space with many obstacles.

- The potential field method also discretizes the space into a grid and moves to the neighboring cell with the lowest potential [17], see Figure 3.1b. The potential should be high near the obstacles and low near the goal. In a space with many obstacles, the potential function creates many local minima and is difficult to traverse.

- The visibility graph algorithm constructs a graph from obstacle vertices that are "visible" from existing nodes [18, 19], see Figure 3.1c. The number of visibility edges scales quadratically with respect to the number of obstacle vertices. Computing such a graph is therefore inefficient for spherical obstacles.

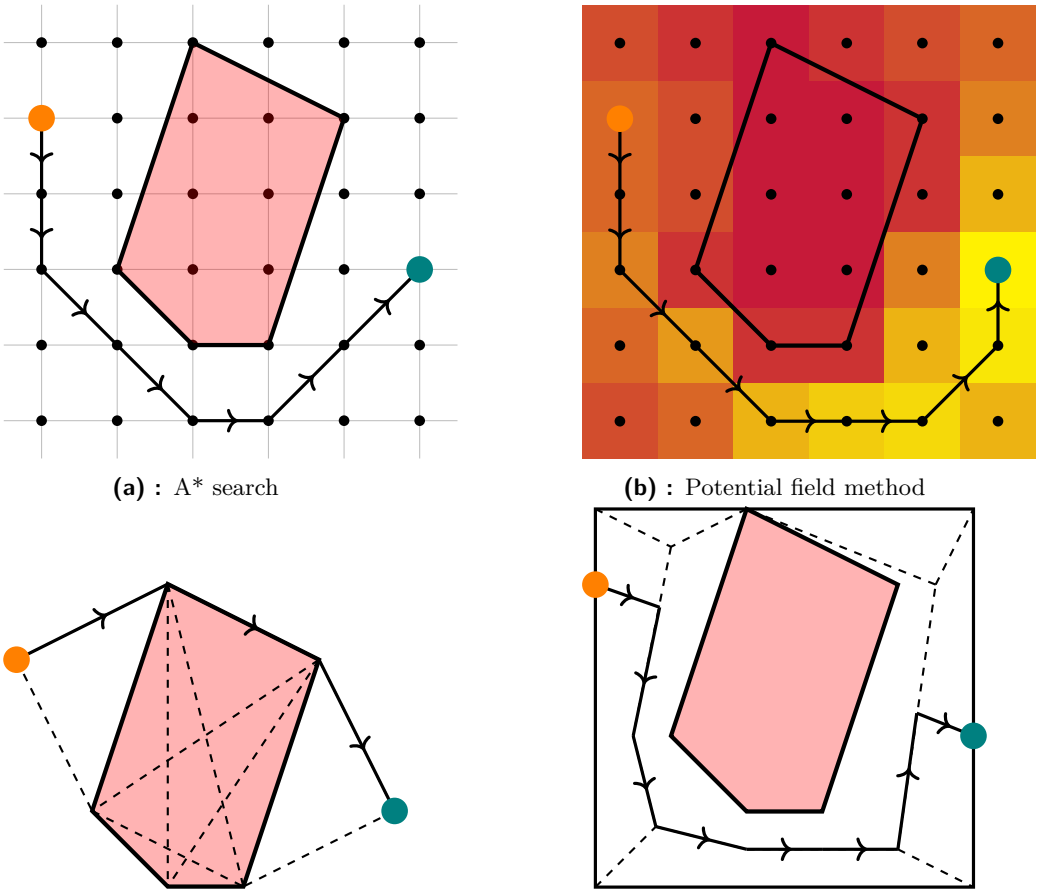
- Voronoi road map planning traverses Voronoi cell boundaries constructed from obstacles [20, 21], see Figure 3.1d. The Voronoi diagram around n points can be constructed in $\mathcal{O}(n \log n)$ [22], Voronoi diagram for obstacles with n vertices can be constructed in $\mathcal{O}(n \log^2 n)$ in 2D [23], $\mathcal{O}(n^2)$ in higher dimensions [24].

Note that all the previously mentioned algorithms can only be used to navigate in the free space around obstacles, but cannot solve the path planning problem with constraints as this would require a discretization of the constraint surface and effective calculation of geodesics.

3.3 Randomized search

All previously mentioned algorithms need to discretize the space that might be computationally- or memory-expensive and give suboptimal results. It was not until 1990 when Glavina published an algorithm that generated random subgoals and tried to build a single roadmap incrementally[25]. In 1994 Horsch et al. succeeded in generating many random points and connecting them with straight lines forming a random graph [26].

A similar method was developed independently by Kavraki et al. in 1996 called the Probabilistic roadmap [27]. Random points in state space are sampled and connected by straight lines to previously sampled points where possible. In areas with tight corridors and many obstacles, more points are sampled to increase the chance of connecting the graph components. Finally, the graph is traversed to find a feasible path as seen in Figure 3.2.



(a) : A* search

(b) : Potential field method

(c) : Visiblity graph algorithm

(d) : Voronoi road map

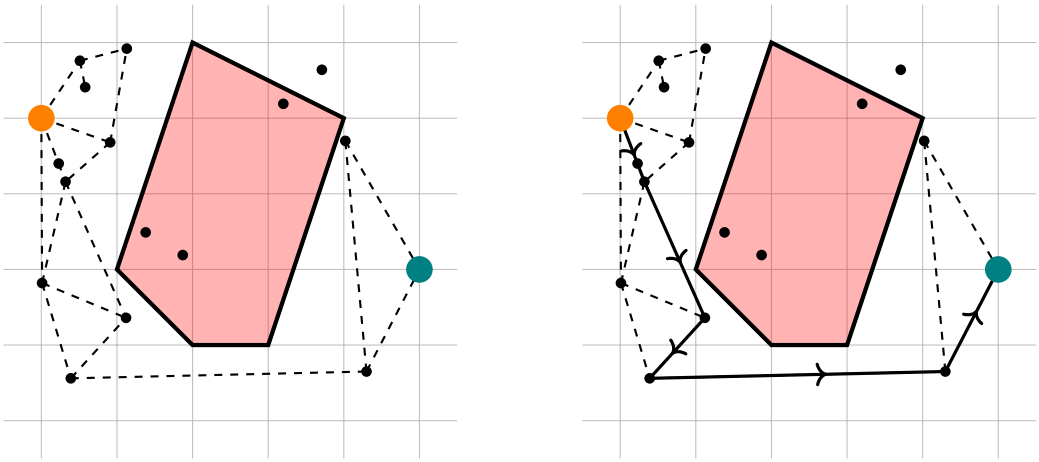
Figure 3.1: Comparison of various discrete path planning algorithms.

3.4 RRT

Our primary concern will be the Rapidly-exploring Random Tree (RRT) algorithm published by LaValle in 1998 [5]. This algorithm repeatedly chooses a random point and connects it to the nearest already explored point. This way, the tree is incrementally built and can later be traversed to find a path.

At the beginning, a tree T is initialized with the starting point x_s. In each of the K steps, a random point x_{rand} is uniformly sampled from the state space, and the nearest neighbor x_{near} is chosen among the states already inside the tree. Then, an input u is selected from x_{rand} in the direction of x_{near} and a step of length Δt is performed, creating a new state x_{new}. This state is added to the tree together with an edge from x_{near}. In our case, the input u is simply a straight motion in the direction of x_{rand}. The pseudocode as presented in [5] can be seen in algorithm Algorithm 3.1.

The RRT algorithm has many desirable properties [5]:



(a) : Fully built probabilistic roadmap in a space with obstacles.

(b) : Probabilistic roadmap with a found path, shown in solid black.

Figure 3.2: Path planning with a probabilistic roadmap.

Algorithm 3.1: Rapidly-exploring random tree

Require: $x_{init} \in \mathcal{S}_{free}$, $K > 0$, $\Delta t > 0$

procedure RRT($x_{init}, K, \Delta t$)
    T.init(x_{init})
    for $k = 1$ to K **do**
        $x_{rand} \leftarrow$ RandomState()
        $x_{near} \leftarrow$ NearestNeighbor(x_{rand}, T)
        $u \leftarrow$ SelectInput(x_{rand}, x_{near})
        $x_{new} \leftarrow$ NewState($x_{near}, u, \Delta t$)
        T.addVertex(x_{new})
        T.addEdge(x_{near}, x_{new}, u)
    return T

- It is simple to understand and fast to run.
- It does not explicitly need a goal state; it simply explores the space.
- It is probabilistically complete. That means that if a solution exists, the probability of finding it approaches one as the number of iterations goes to infinity.
- The distribution of vertices approaches the sampling distribution. This is a critical point as it allows us to search the space uniformly as long as we have a uniform sampling method, which we will discuss later.
- Expansion is biased towards unexplored areas [28]. That is known as the Voronoi bias.

3.4.1 Voronoi bias

Definition 2: Voronoi cell

Let $p_1, ..., p_n$ be points in a metric space with a distance function d. A Voronoi cell R_i for point p_i is set of points which are closer to p_i than to any other point $R_i = \{x \mid \forall j \neq i : d(x, p_i) \leq d(x, p_j)\}$.

Definition 3: Voronoi bias

When uniformly sampling a point, its probability being inside a certain Voronoi cell is proportional to its size. Suppose we encounter a situation where our randomly sampled points are all in a small area. In that case, these points will have small Voronoi cells and the probability is high that the next point will not fall into the same area again see Figure 3.3. The point sampling seems to be biased toward unexplored regions; this observation is called the *Voronoi bias*.

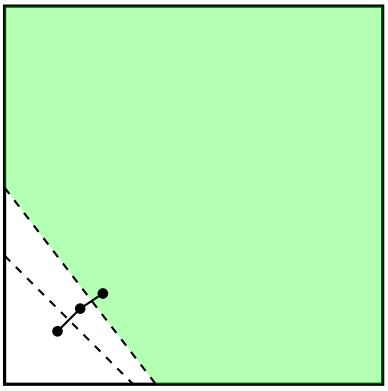


Figure 3.3: Voronoi bias predicts that the top right point has the highest chance to be expanded from since it has the largest Voronoi cell.

The same principle can also be used to observe that points with large Voronoi cells are more likely to be extended by RRT; thus the tree will grow towards unexplored regions [28].

3.4.2 Optimality

Even though RRT is probabilistic complete, it does not find an optimal path. For optimality, a modified version called RRT* was developed by Karaman et al. and proven correct [29]. After adding a new node, RRT* does not connect it to the nearest neighbor but instead finds the shortest path from the start, possibly rerouting some edges. This way, all paths in thee are optimal, and an optimal solution to the goal is asymptotically found.

Chapter 4

Planning with constraints

Even though RRT has many desirable properties, it is only as good as its random generator. RRT node distribution is proven to approach the underlying sampling distribution [28]. If we want to explore the space uniformly, we need to sample it uniformly.

Definition 4: Rejection sampling

Rejection sampling is a simple method to sample space with obstacles. First, a random point in the space is sampled. If it lies inside an obstacle, it is rejected, and the process is repeated until we sample a point outside all obstacles [30]. It is sometimes more efficient to compute multiple iterations of rejection sampling than to sample from an explicit distribution function [31].

RRT does not necessarily need to sample $\mathcal{S}_{free}$. Nevertheless, if desired, the sampling outside the obstacles can be easily overcome by using rejection sampling. Sampling while satisfying constraints however, is much more difficult.

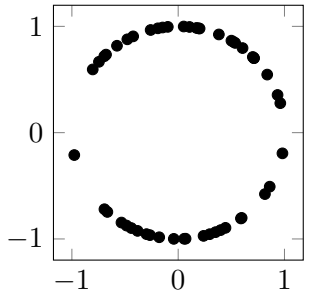
Example 3: Sampling on a circle

Let our constraint be a circle $x^2+y^2=1$. We want to find a random point on this circle. Rejection sampling will not work because the probability of a random point in $[0;1]^2$ has zero probability of satisfying $x^2+y^2=1$ since $[0;1]^2$ is a two-dimensional square, but the circle is a one-dimensional curve.

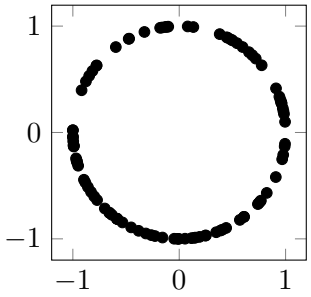
One idea is to sample in one dimension first and then try to satisfy a constraint. First x is uniformly sampled from $[0;1]$ and then y is chosen from $\{\sqrt{1-x^2}; -\sqrt{1-x^2}\}$. This is a valid distribution but not a uniform one, since points with $x \approx \pm 1$ will be sampled with less probability then point with $x \approx 0$ as seen in the Figure 4.1a.

This is not what we wanted. We need to sample the point uniformly along

the curve, that is to approach the probability density function $f(\theta) = \frac{1}{2\pi}$ where $\theta \in [0; 2\pi)$ is polar angle of a sampled point. Since the circle is a simple curve, we can use its polar parametrization $x = \cos\theta, y = \sin\theta$ and sample $\theta \in [0; 2\pi)$ uniformly. This distribution looks much more natural as seen in the Figure 4.1b.



(a) : Sampling with a uniform x coordinate.



(b) : Sampling with a uniform polar angle θ.

Figure 4.1: 50 random points on a circle sampled with two different distributions.

4.1 Manifolds

In this section, we will explain several essential definitions from topology.

Definition 5: Euclidean space

The Euclidean space of dimension n is a subset of the vector space $\mathbb{R}^n$. The elements of the Euclidean space are n-tuples of real numbers and are called points. The Euclidean space can be seen as subset of the vector space over $\mathbb{R}$ with the dot product as an inner product. The induced distance function is then $d(x, y) = ||x - y|| = \langle x - y; x - y\rangle = (x - y) \cdot (x - y)$. Unit ball centered at the point c in Euclidean space is $\{x \in \mathbb{R}^n \mid ||d(x, c)|| < 1\}$.

Definition 6: Homeomorphism

Bijection f between two topological space is homeomorphism iff f and f^{-1} are both continuous. Homeomorphisms allow us to compare two topological spaces for equality up to continuous deformation. If such a function exists, the two spaces are said to be homeomorphic.

Example 4: Homeomorphism

Let $f(\theta) = (x; y) = (\cos\theta; \sin\theta)$ for $\theta \in [0; 2\pi)$, $x^2 + y^2 = 1$. Then $f^{-1}(x, y) = \text{atan2}(y, x)$. We can easily see that both functions are continuous and we have found a bijection between $[0; 2\pi)$ and a unit circle. They are therefore hemeomorfic, see Figure 4.2.

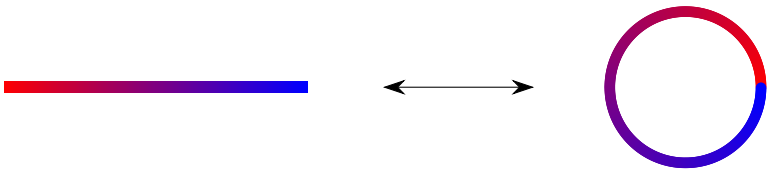


Figure 4.2: Homeomorphism between an interval and a circle.

Definition 7: Manifold

A manifold of dimension n is a topological space that locally resembles Euclidean space. That means that a neighborhood exists around every point that is hemeomorfic to Euclidean unit ball of the dimension n.

Definition 8: Chart

A chart on a manifold M at a point x is a homeomorphism from the neighborhood of x to a Euclidean unit ball, see Figure 4.3. We will be using the term *chart* for both the homeomorphism and the corresponding Euclidean unit ball.

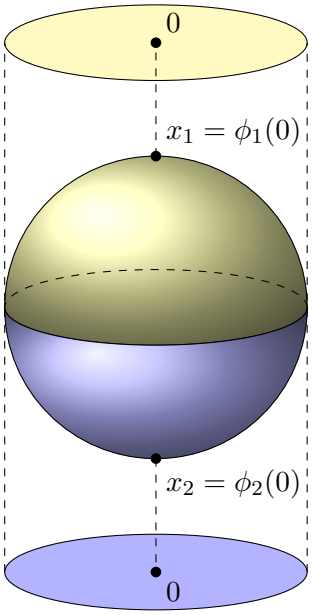


Figure 4.3: Example of approximating a sphere with two circle charts.

Definition 9: Atlas

An atlas of a manifold M is a set of charts centered at some points on M, see Figure 4.4.

Since a manifold is homeomorphic to the Euclidean space at every point, it can be locally approximated by an atlas. This is an important feature we will use later.

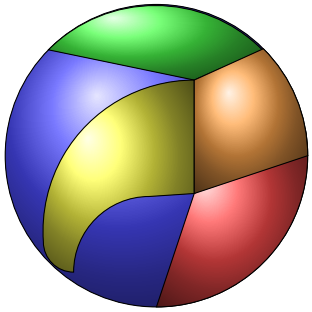


Figure 4.4: Example of an atlas covering the surface of a sphere. Each color represents a different chart.

Definition 10: Smooth manifold

A manifold is smooth if a differential calculus is defined on every chart of the manifold, see Figure 4.5.

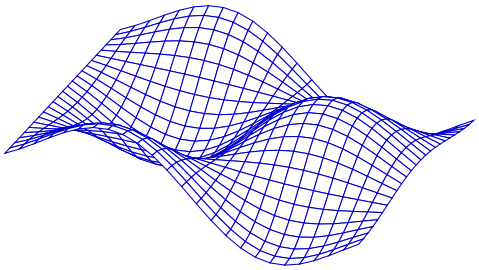


Figure 4.5: Example of a smooth manifold, where standard the 3D calculus works.

Definition 11: Tangent space

Suppose that M is an n-dimensional smooth manifold and $x \in M$ is a point on the manifold. Let $\phi\colon (-1;1) \to \mathbb{R}^n$ be a differentiable curve such that $\phi(0) = x$. Then the vector $\phi'(0)$ is said to be tangent to the manifold at the point x. Set of all such functions ϕ defines a vector space of tangent vectors at point x, denoted T_xM, see Figure 4.6.

Definition 12: Riemannian manifold

The Riemannian manifold is a smooth manifold with a smooth inner product defined on its tangent space. That means that an inner product is defined in each tangent space and this inner product varies smoothly from point to point along the manifold.

We will be concerned with the Riemannian manifolds since they naturally allow us to work with the tangent spaces. Many of the usual manifolds are Riemannian, such as circles, spheres, ellipsoids, parabolas, and other polynomial surfaces.

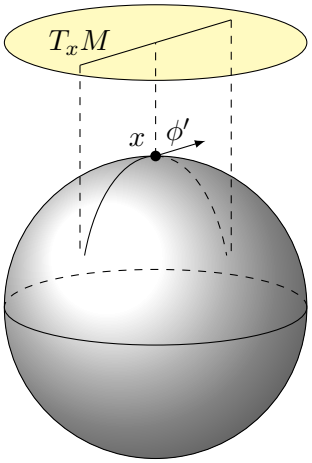
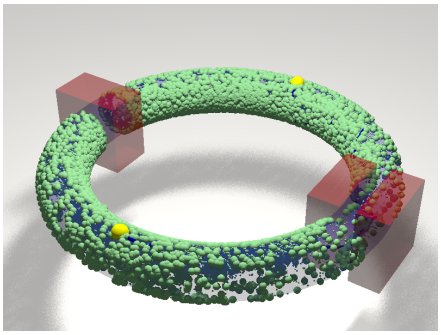


Figure 4.6: Sphere with curve ϕ passing through its pole and calculated derivative vector. Drawing more curves would yield more derivatives creating the whole tangent vector space T_xM (yellow).

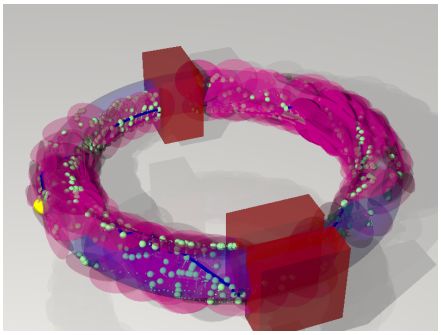
Our definition of the state space $\mathcal{S}$ will be the manifold satisfying $F(x) = \mathbf{0}$ for some smooth differentiable function $F : X \to \mathbb{R}^k$. The space tangent to the manifold at x is then the nullspace of its Jacobian $J_x = \mathbf{0}$. In case of k independent constraints in $\mathbb{R}^n$, the dimension of the constrained state space will be $\mathbb{R}^{n-k}$ and the dimension of the tangent space $\mathbb{R}^{n-k}$.

Example 5: Torus constraint

Let us consider a constraint in shape of a torus surface. The ambient space has dimension $n = 3$, but the $k = 1$ constraints limit the space to the surface of a torus, which is a subset of $\mathbb{R}^{n-k} = \mathbb{R}^2$, see Figure 4.7a. Charts tangent to the space are balls (circles) of the dimension $n - k = 2$, see Figure 4.7b.



(a) : Many green points are sampled on the surface of the torus satisfying the constraint.



(b) : Purple circles are the charts forming an atlas approximating the torus surface.

Figure 4.7: Torus constraint in a 3D space.

4.2 Sampling on manifold

As we have seen previously, sampling a random point uniformly on the manifold might not be easy. Since an atlas of charts can be locally approximate the manifold, we can sample on the atlas instead. This idea inspired many algorithms [32]; we will be mainly concerned with AtlasRRT by Jaillet et. al.

Paper titled *Path Planning with Loop Closure Constraints using an Atlas-based RRT* [6] contains a simple description of the AtlasRRT algorithm. This algorithm takes the start and goal states and finds a path that is feasible under the set of constraints $F(x) = \mathbf{0}$, which form a manifold.

Pseudocodes taken from the original paper are listed below.

The algorithm begins with initializing an RRT tree and an atlas with charts at x_s and x_g. AtlasRRT keeps the simple loop of RRT as seen in the Algorithm 4.1. A random point x_r is sampled on the atlas, and the tree is extended towards this point. A bidirectional search was used in the paper to speed up computation. The first tree is extended toward the random point x_r by a step of size δ, the second tree is extended towards the newly added point, and then the two trees are swapped, which ensures that both trees expand by the same speed. Eventually, the two trees meet closer than δ and form one graph containing an approximate path from x_s to x_g.

Algorithm 4.1: AtlasRRT

Require: $x_s, x_g \in \mathcal{S}_{free}$, $F : \mathcal{S} \to \mathbb{R}^n$, $F(x_s) = F(x_g) = \mathbf{0}$
Ensure: valid path between x_s and x_g is returned

procedure AtlasRRT(x_s, x_g, F)
    $T_s \leftarrow$ InitRRT(x_s)
    $T_g \leftarrow$ InitRRT(x_g)
    $\mathcal{A} \leftarrow$ InitAtlas(F, x_s, x_g) ▷ *Prepare initial charts*
    Done $\leftarrow$ False
    while not Done **do**
        $x_r \leftarrow$ SampleOnAtlas($\mathcal{A}$)
        $n_r \leftarrow$ NearestNode(T_s, x_r)
        $x_l \leftarrow$ ExtendTree($T_s, \mathcal{A}, n_r, x_r$)
        $n_l \leftarrow$ NearestNode(T_g, x_l)
        $x'_l \leftarrow$ ExtendTree($T_g, \mathcal{A}, n_l, x_l$)
        if $||x_l - x'_l|| < \delta$ **then** ▷ *Until the trees meet*
            Done $\leftarrow$ True
        else
            Swap(T_s, T_g)
    return Path(T_s, x_l, T_g, x'_l)

The hardest part of the original problem is the random sampling on the manifold. As seen in the Algorithm 4.2 the sampling is simple on the atlas. A random chart is chosen from the atlas, and a random point in a radius of R from chart origin is sampled and converted to the ambient coordinates.

Algorithm 4.2: Sampling on an atlas

Require: atlas $\mathcal{A}$, constraints $\mathcal{F}$, $R > 0$
Ensure: random point on the explored part of atlas is returned

```
1: procedure SampleOnAtlas(A)
2:     repeat
3:         r ← RandomChartIndex(A)
4:         u_r ← RandomOnBall(R)
5:     until u_r ∈ F_r
6:     return φ_r(u_r)
```

In the last part, the tree must be extended towards the random point x_r. This was straightforward in basic RRT, but the random point is no longer feasible, because we sampled on a chart, not on the manifold itself. Since the chart locally approximates the manifold, the sampled point is expected to lie near the manifold. It can be projected onto the manifold using $\psi_i \colon \mathbb{R}^n \to \mathbb{R}^n$ as seen in the Algorithm 4.3. To speed up the computation, the extension in the same direction is repeated as long as the new points remain feasible and their projections are not far away from each other (which sets an implicit bound on the chart deviation, more on this in the next section).

The most crucial part of AtlasRRT is the atlas building. After the tree is extended by one step, the validity of the new point is tested. If the point falls outside the chart validity area $\mathcal{V}_c$, a new chart is created at the new point and added to the atlas. This way, the atlas is slowly growing towards unexplored areas. Each time a new chart is added, an inequality bound should be added between overlapping charts, further decreasing the validity area to $\mathcal{F}_c$. If the new point is outside $\mathcal{F}_c$, it is moved to the neighboring chart. We will define both of these terms properly in the following section. A visualization of an atlas can be seen in the Figure 4.8.

There are multiple extensions of RRT other than AtlasRRT, which can also be used on a manifold, such as

- CBiRRT uses RRT in the ambient space and projects onto the manifold in every step [33]
- HC-planner incrementally builds chart at each point to simplify sampling [34]
- TB-RRT builds the tree fully on the atlas and resizes charts according to the local curvature [35]

Algorithm 4.3: Adding a branch to the AtlasRRT

Require: tree T, atlas $\mathcal{A}$, index of the nearest node in tree n, random sample x_r

Ensure: x_n is a valid node on the manifold if an extension was performed

procedure ExtendTree($T_s, \mathcal{A}, n, x_r$)
  $c \leftarrow$ ChartIndex(n) ▷ *Find a chart containing a node n*
  $u_r \leftarrow \psi_c^{-1}(x_r)$
  $x_r \leftarrow \phi_c(u_r)$
  Blocked $\leftarrow$ False
  while not Blocked **and** $||u_n - u_r|| > \delta$ **do** ▷ *Extend until blocked*
    $u_j \leftarrow (u_r - u_n)\delta/||u_r - u_n||$ ▷ *Calculate a step*
    $x_j \leftarrow \psi_c(u_j)$
    if $||x_j - x_n|| > 2\delta$ **or** Collision(x_j) **then** ▷ *Check curvature*
      Blocked $\leftarrow$ True
    else
      New $\leftarrow$ False
      if $u_j \notin \mathcal{V}_c$ **then**
        $c \leftarrow$ NewChart($\mathcal{A}, x_n$)
        New $\leftarrow$ True
      else
        if $u_j \notin \mathcal{F}_c$ **then** ▷ *Is there a chart nearby*
          $c \leftarrow$ MoveToChart(x_n, u_j)
          New $\leftarrow$ True
      if New **then**
        $u_j \leftarrow \psi_c^{-1}(x_j)$
        $u_r \leftarrow \psi_c^{-1}(x_r)$
        $x_r \leftarrow \phi_c(u_r)$
      $n \leftarrow$ AddNode($T, \mathcal{A}, c, x_j$)
  return x_n

AtlasRRT sits somewhere in between these algorithms. It uses atlas for sampling, but projects the points onto the manifold in each step. The main goal is to minimize the number of charts and maximize their coverage along the manifold.

4.3 Implementation details

4.3.1 Projection

The implementation of AtlasRRT relies heavily on the ability to project points from a tangent chart onto the manifold itself. Thus we need to find the mapping $\psi_i(\boldsymbol{x}_j^i) = \boldsymbol{u}_j$, which maps a point $\boldsymbol{x}_j^i$ from the chart $\mathcal{C}_i$ to the point $\boldsymbol{u}_j$ on the manifold. Such mapping must map the origin to the chart-

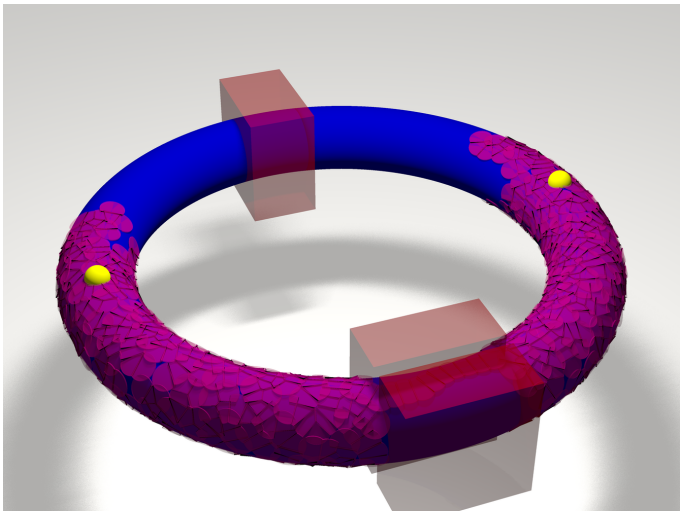


Figure 4.8: Atlas of charts (purple) built by our implementation of AtlasRRT in the Torus benchmark. The feasible states are on the surface of the blue torus. Narrow passage can be seen in the red constraint which the path planner must find in order to connect the start and goal states (yellow).

manifold tangency point $\psi_j(\mathbf{0}) = \boldsymbol{x}_i$. Orthogonal matrix Φ_i is a basis of the tangent space if it is orthogonal to the Jacobian of the manifold J evaluated at the desired point of tangency x_i

$$\begin{bmatrix} J_{\boldsymbol{x}_i} \\ \Phi_i^\top \end{bmatrix} \Phi_i = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{I} \end{bmatrix}. \tag{4.1}$$

Given the basis Φ_i, points in the chart $\mathcal{C}_i$ can be easily sampled since $\mathcal{C}_i$ is nothing more than a Euclidean ball. The point $\boldsymbol{u}_j^i$ can be transformed into the ambient space

$$\boldsymbol{x}_j^i = \phi_i(\boldsymbol{u}_j^i) = \boldsymbol{x_i} + \Phi_i \boldsymbol{u}_j^i \tag{4.2}$$

and then orthogonally projected onto the manifold to obtain $\boldsymbol{x_j}$. This is equivalent to solving the system of equations

$$\begin{bmatrix} F(\boldsymbol{x}_j) \\ \Phi_i^\top (\boldsymbol{x}_j - \boldsymbol{x}_j^i) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{4.3}$$

which the paper suggests to solve iteratively by Newton's method.

$$\begin{bmatrix} J_{\boldsymbol{x}_j} \\ \Phi_i^\top \end{bmatrix} \Delta \boldsymbol{x}_j = - \begin{bmatrix} F(\boldsymbol{x}_j) \\ \Phi_i^\top (\boldsymbol{x} - \boldsymbol{x}_j^i) \end{bmatrix} \tag{4.4}$$

In our implementation, a step of Newton's method requires calculating the QR decomposition of the left-hand side matrix in the Equation 4.4 and internally uses LAPACK [36]. We have found experimentally that for high-dimensional problems, the projection dominates the time complexity of the whole problem.

4.3.2 Chart validity

The paper defines validity area $\mathcal{V}_i$ as the area from chart $\mathcal{C}_i$ as the set of points $\boldsymbol{u}_j^i$ such that

$$\begin{aligned} ||\boldsymbol{u}_j^i|| &\leq \beta R, \\ ||\psi_i(\boldsymbol{u}_j^i) - \phi_i(\boldsymbol{u}_j^i)|| &\leq \varepsilon \end{aligned} \tag{4.5}$$

hold for some $0 < \beta < 1$ and $\varepsilon \ll R$. The first equation is necessary to make the validity area smaller than the sampled radius and potentially explore new areas. The second limits the distance of the chart from the manifold.

Furthermore, if during the branch extension a point falls outside $\mathcal{V}_i$, a new chart is constructed. An additional constraint is introduced to prevent the construction of a new chart in the same area repeatedly. The list of constraints $\mathcal{F}_i$ for chart $\mathcal{C}_i$ contains the equation

$$2\boldsymbol{u}^\top \psi_i^{-1}(\boldsymbol{x}_j) \leq ||\psi_i^{-1}(\boldsymbol{x}_j)||^2 \tag{4.6}$$

for every neighboring chart $\mathcal{C}_j$ with the center $\boldsymbol{x}_j$. This inequality orthogonally bisects the projection of the neighboring center to the chart. Our ultimate goal should be to construct an approximate bijection between the manifold and the atlas. This is however, not possible using the Equation 4.6, because neighboring charts' borders do not form a pair of complementary half-spaces.

The original paper [6] states the following:

> "When a given chart $\mathcal{C}_i$ is fully surrounded by neighboring charts, the intersection of the half-spaces defined by the inequalities in $\mathcal{F}_i$ conform a polytope that conservatively bounds the actual validity area for the chart, taking into account the presence of neighboring charts."

This quote is generally not true, as seen in the Figure 4.9. When imposing a constraint between the neighboring charts, holes are formed, and it is possible that a sampled point cannot be projected onto the manifold. The

greater the angle between the charts the smaller the valid area. This must be accounted for in the algorithm, otherwise an infinite loop is created in the MoveToChart procedure. There are several approaches to the problem.

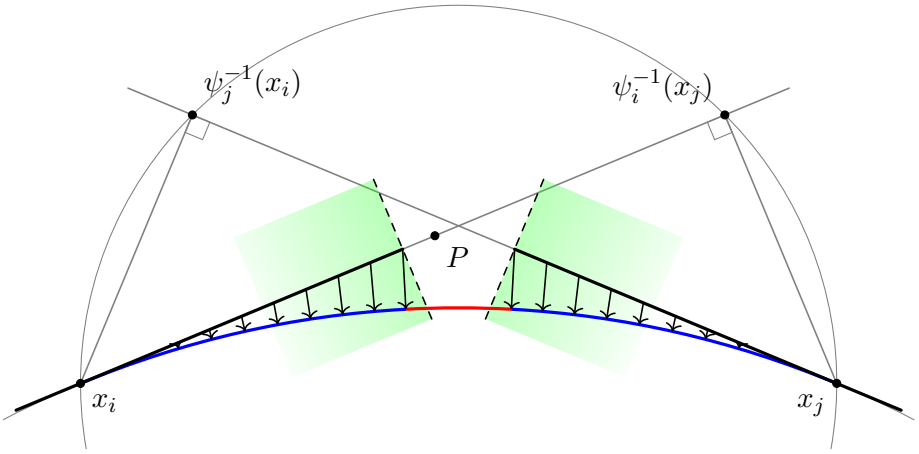


Figure 4.9: Two neighboring charts (black) approximate a circular manifold (blue). A constraint is added according to the Equation 4.6 (dashed). This however creates a hole in the approximation (red). If a point P is sampled just outside the constraint, it cannot be moved to the neighboring chart because it's outside of the neighbor's constraint as well. This happens due to the fact that constraints bisect a chord of the Thales's circle and every chord (except for parallel charts) is shorter than the diameter.

4.3.3 Chart disparity reduction

The first approach originates from Micheal Henderson [37] who states that the manifold approximation is accurate as long as

$$||\Phi_i^\top \Phi_j|| \leq 1 - \varepsilon \tag{4.7}$$

hold for any neighboring charts $\mathcal{C}_i$, $\mathcal{C}_j$, in a later paper [38] expressed as

$$||\Phi_i^\top \Phi_j|| \geq \cos \alpha. \tag{4.8}$$

The authors of the AtlasRRT paper are aware of this fact and hope that the issue will not arise at all with sufficiently low chart deviation. This, however is overlooking the issue, not solving it.

The second approach is rather simple. Move the constraint plane further from the chart center slightly. More formally it add a factor $k > 1$ to the Equation 4.6 to obtain

$$2\boldsymbol{u}^\top \psi_i^{-1}(\boldsymbol{x}_j) \leq k \cdot ||\psi_i^{-1}(\boldsymbol{x}_j)||^2. \tag{4.9}$$

In practice, $k = 1.1$ was found to be a reasonable value reducing the chance of landing in a hole to a minimum. It still does not guarantee the issue will be solved but it is simple to implement and was experimentally verified to speed up the overall solution. For a geometric demonstration see the Figure 4.10.

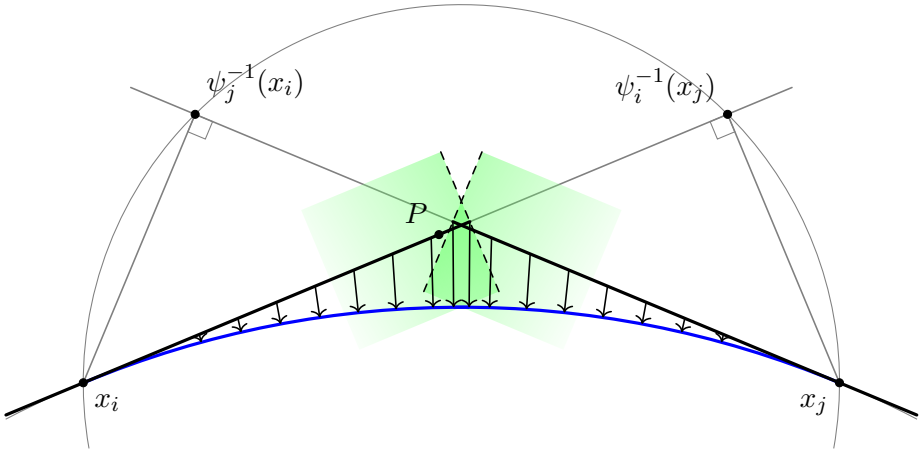


Figure 4.10: The same setting as in Figure 4.9, but now the Equation 4.9 is used to calculate chart constraints with $k = 1.2$. We see that there is no hole anymore and the point P can be projected onto the manifold.

The third approach is the only one to solve the problem completely. When MoveToChart is called, an algorithm is used to detect cycles. Once a point is moved to the same chart for the second time, the process is stopped and a new chart is formed centered at this point. This way, the point can be assigned to a chart and most of the hole is covered by the new chart. This adds complexity to MoveToChart and increases the number of charts but guarantees to solve the issue. There will be as many charts as the points sampled in the worst case. Experiments have shown that this is usually not the case.

In practice, we recommend combining all three approaches. Choose a smaller chart size to reduce deviation, move chart constraint by a factor k, and employ a cycle detection algorithm to add a new chart if necessary.

Chapter 5

Results

5.1 Chart disparity

One of the goals of this work is to improve chart creation. The solutions proposed in the previous chapter allow us to choose a larger chart radius R, effectively detect if a particular area of the manifold already has a chart in the atlas, and create new charts only when necessary, leading to a reduction of the chart count.

The chart creation was benchmarked on the Torus benchmark. The state space has dimension $n = 3$ and contains a $k = 1$ constraint

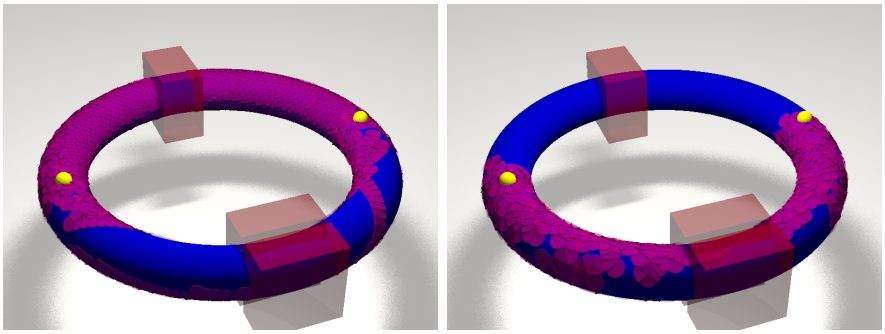
$$F(x) = (x_1^2 + x_2^2 + x_3^2 + 200^2 - 30^2)^2 - 4 * 200^2 * (x_1^2 + x_2^2) \tag{5.1}$$

The start and goal states are separated by two obstacles, one blocks the path, the other contains a narrow passage in the middle. The difference between the original and the improved version with inequality growth by $k = 1.1$ can be seen in the Figure 5.1.

Across 50 runs, the average count of charts was (2252 ± 251) for the original implementation and (839 ± 109) in our improved version. The average runtime was (41 ± 2) s compared to (20 ± 10) s In 27 test cases (54%) of the unimproved version, a point was sampled outside between two charts and could not be projected to either of them.

5.2 Comparison with OMPL

The Open Motion Planning Library (OMPL [7]) is the de facto standard for benchmarking path planning algorithms. OMPL tries to abstract various



(a) : Path found by AtlasRRT implemented according to the original paper.

(b) : Path found by our improved implementation of AtlasRRT with much less charts used.

Figure 5.1: Comparison of the original and improved AtlasRRT. The number of charts (purple) is greatly reduced, speeding up the algorithm run and finding the path much quicker.

parts of the planning problems to allow algorithm substitution. Planning under constraints is possible in OMPL, despite having no constrained planning algorithms. Instead, a regular space planner is used, and the underlying space itself is automatically projected onto the manifold using one of the projection methods [8].

For our purposes we selected a collection of non-optimizing path planners: RRT [5], RRTConnect [39], LazyRRT [5, 40], EST [41], BiEST [41], SBL [9], KPIECE1 [10], BKPIECE1 [10], LBKPIECE1 [10, 40] and STRIDE [42]. All the planners were initialized with their default configuration which adjusts sampling parameters according to the goal distance. An atlas-based projection from the OMPL suite was used [8].

Our planner program is written in Julia [43] as an implementation of the OMPL::BASE::PLANNER interface. Start and goal configurations are transferred from C++ to Julia via a native interop. Due to the nature of Julia just-in-time (JIT) compilation, a dummy preheat run on an unrelated problem was executed before each set of benchmarks.

5.2.1 Torus

Algebraic surfaces are easy to visualize and provide a good starting point to test the implementation. Here we used a toroidal constraint in a 3D ambient space, the same as in the previous section.

All of the benchmarked algorithms were able to find a feasible path in most runs under 10 seconds. Our implementation of AtlasRRT had 100% success rate. On average, our implementation was twice as fast as OMPL RRT over the atlas projection space and equally performant as EST, KPIECE1, and STRIDE. Several OMPL algorithms, such as SBL and BKPIECE1 performed

better in this benchmark at the cost of not solving all the instances in the specified timeout. See the full comparison in the Figure 5.2.

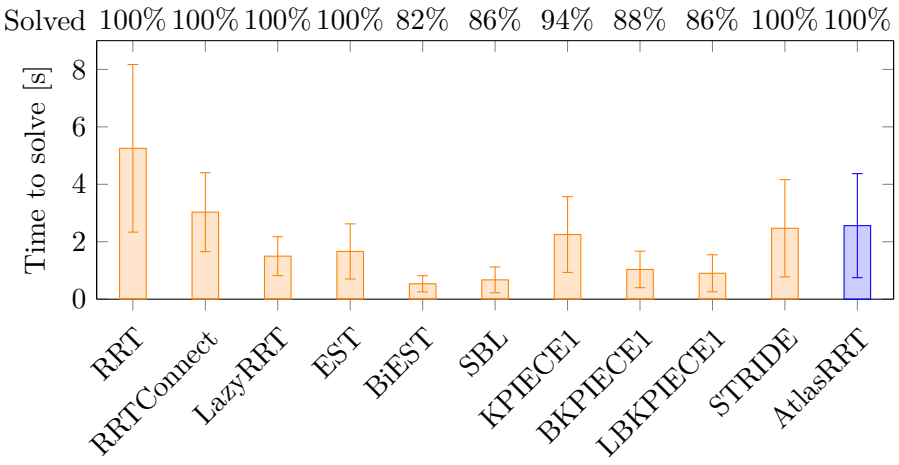


Figure 5.2: Comparison of various OMPL solvers using Atlas projection on the torus benchmark versus our implementation of AtlasRRT. Average solve time is plotted along with a standard deviation interval. A count of successfully solved instances out of 50 runs is listed above the chart. The timeout was 10 seconds.

5.2.2 Planar manipulator

A planar 2D manipulator with two revolute joints (also known as SCARA [44]) is well known in the industry. The inverse kinematics of this robot can be calculated in a closed form, but it still requires a selection from two solutions so as not to cross the links. Here we instead parametrize the configuration with two coordinates of the end effector and the angles of both joints. By solving this problem, we obtain a feasible path in both the effector and joint coordinates. The state space has dimension $n = 4$ and is restricted by $k = 2$ constraints. There are multiple obstacles in the end effector configuration space. Joint angle limits are checked.

Our implementation had a 100% success rate and was on average faster than all of the OMPL-based algorithms. Render of a found path in the state space can be seen in the Figure 5.3.

5.2.3 3D manipulator

Robotic manipulators operating in a 3D task space are successfully used in many industrial applications. Many such as assembling or welding robots require constructions with many degrees of freedom (DOF) [45, 46]. There is still no known closed-form solution to the inverse kinematics of a general

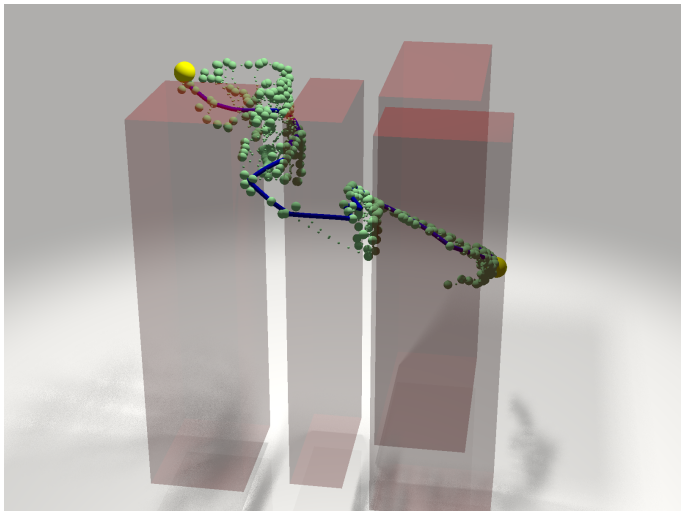


Figure 5.3: 3D cross section of the 4D state space in the planar manipulator benchmark. The horizontal axes are the spatial coordinates of the end effector, the vertical axis is the angle of the first joint.

6 DOF manipulator [13]. Unique construction designs are often used to simplify calculations.

We will use a four-link manipulator with three ball joints. The manipulator's end effector is constrained to a spherical constraint, and there are two obstacles between start and goal configurations. This construction resembles a closed kinematic chain. The configuration parameters are the coordinates of the three intermediate joints and the end effector. State space has $n = 12$ dimension and $k = 5$ constraints.

Both OMPL and our implementation instantly found a path through the 12-dimensional ambient space when ignoring obstacles. With the presence of obstacles some of the OMPL algorithms' performance dropped, and our AtlasRRT implementation was unable to progress beyond the first obstacle even after five minutes. See a visualization of the OMPL-found path in the Figure 5.5 and an overview of sampled states of our implementation in the Figure 5.6. See the algorithm comparison in the Figure 5.7.

This benchmark demonstrates that pure AtlasRRT can effectively sample a constrained state space but does not perform well in spaces with many obstacles and narrow passages.

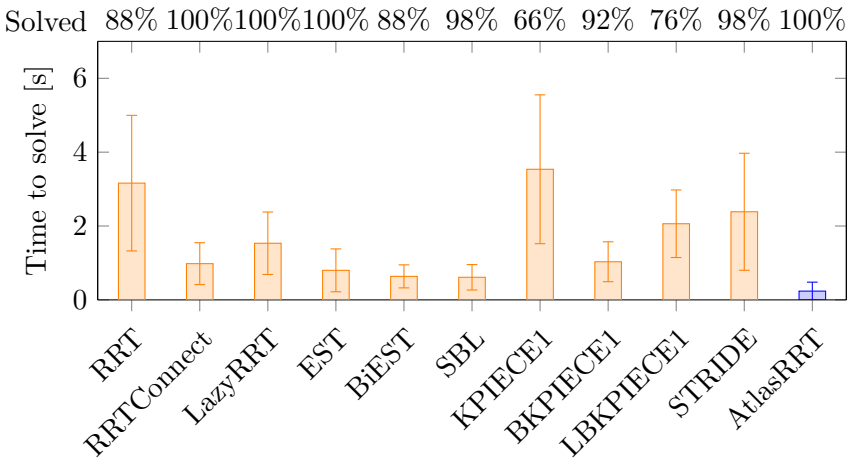
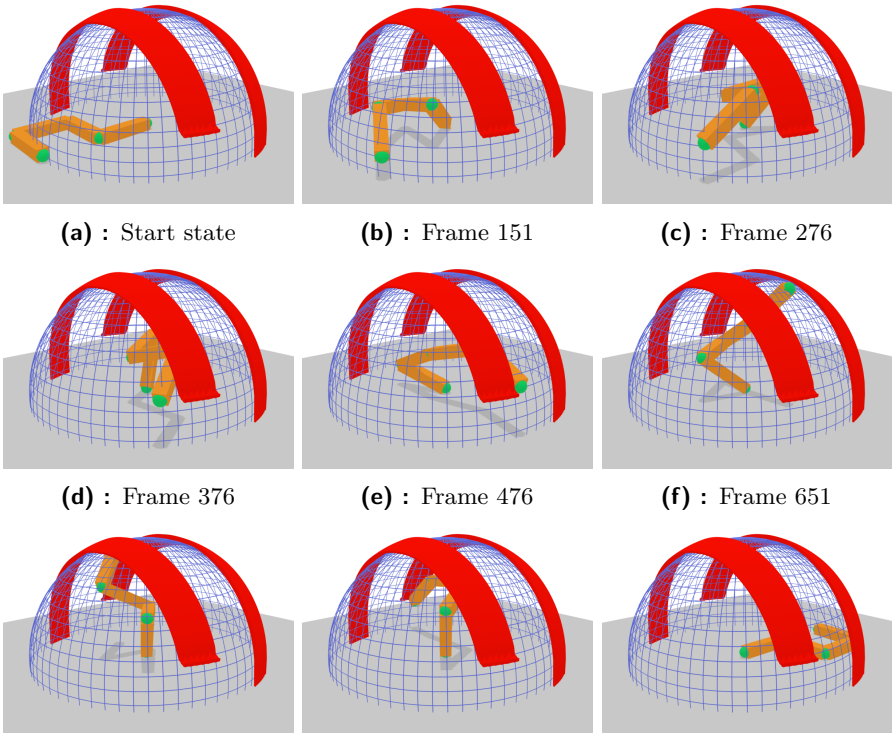


Figure 5.4: Comparison of various OMPL solvers using Atlas projection on the planar manipulator benchmark versus our implementation of AtlasRRT. The timeout was 10 seconds. Our implementation solved all test instances almost instantly.



(a) : Start state **(b) :** Frame 151 **(c) :** Frame 276

(d) : Frame 376 **(e) :** Frame 476 **(f) :** Frame 651

(g) : Frame 826 **(h) :** Frame 1126 **(i) :** Goal state

Figure 5.5: Sequence of states found by OMPL RRT planner in the 3D manipulator benchmark. Robotic arm (orange) performs motion around obstacles (red) while the end-effector (green) is constrained to keep contact with a sphere (blue). Rendered with the OMPL visualization script for blender [47].

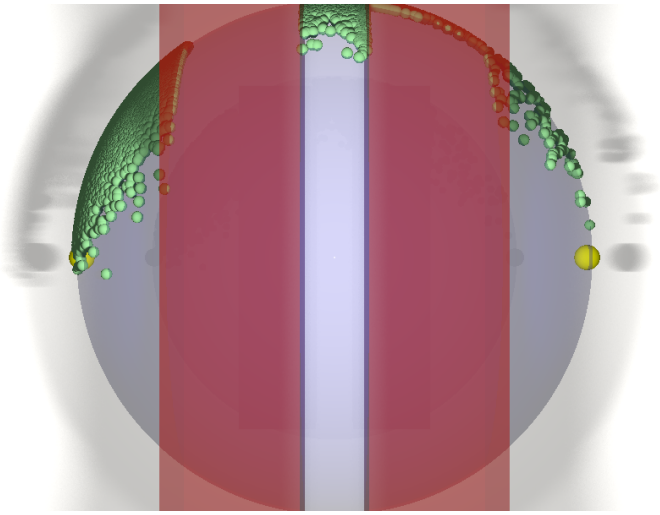


Figure 5.6: An attempt to solve the 3D manipulator benchmark. Our implementation generated thousands of feasible states (green) but still failed to navigate itself around the obstacles (red) and timed out. Top-down view.

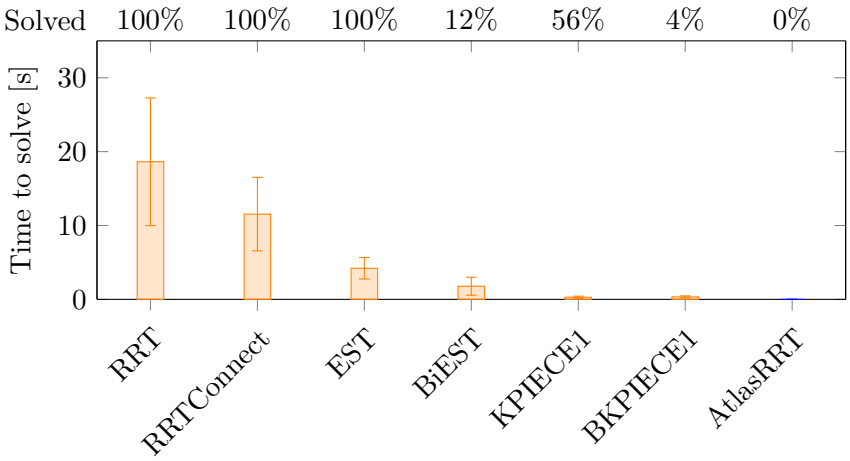


Figure 5.7: Comparison of various OMPL solvers using Atlas projection on the 3D manipulator benchmark. Our implementation did not succeed to find a path around the obstacles in the 60 second timeout in any of the 50 test runs.

Chapter 6

Conclusion

6.1 Conslusion

This work addressed the path planning problem. This task arises in many fields, including robotic manipulators, transport, and biochemistry. We formulated the path planning problem with obstacles and holonomic constraints. First, we covered solving a relaxed version of the problem using graph algorithms and a sampling-based planner. Next, we studied the AtlasRRT algorithm and successfully implemented it.

The main goal of this work was to complete and improve the AtlasRRT implementation. We proposed a novel approach to reduce the chart disparity with probabilistic and deterministic approaches. In the last chapter, we used our implementation to solve several different problems from robotics. The algorithm was found to run correctly, find a feasible path, and not crash due to the chart disparities.

Our implementation is comparable to several state-of-the-art algorithms from the OMPL test suite and is faster than a plain RRT over an atlas projection space. This makes the AtlasRRT algorithm and our proposed improvements suitable for a wide range of problems, which we demonstrated on a series of benchmarks, including planar and multi-DOF manipulators.

6.2 Future work

Despite being suitable for problems in high dimensions, the bottleneck of the AtlasRRT is the QR decomposition inside the projection routine. We experimented with problems having hundreds of parameters and constraints, and the tree extension performance dropped significantly. For those problems, TB-RRT [35] might be more suitable since it does not need projection

in every step. Using a different algorithm would potentially allow efficient planning for rigid cloths and protein folding, see Figure 6.1.

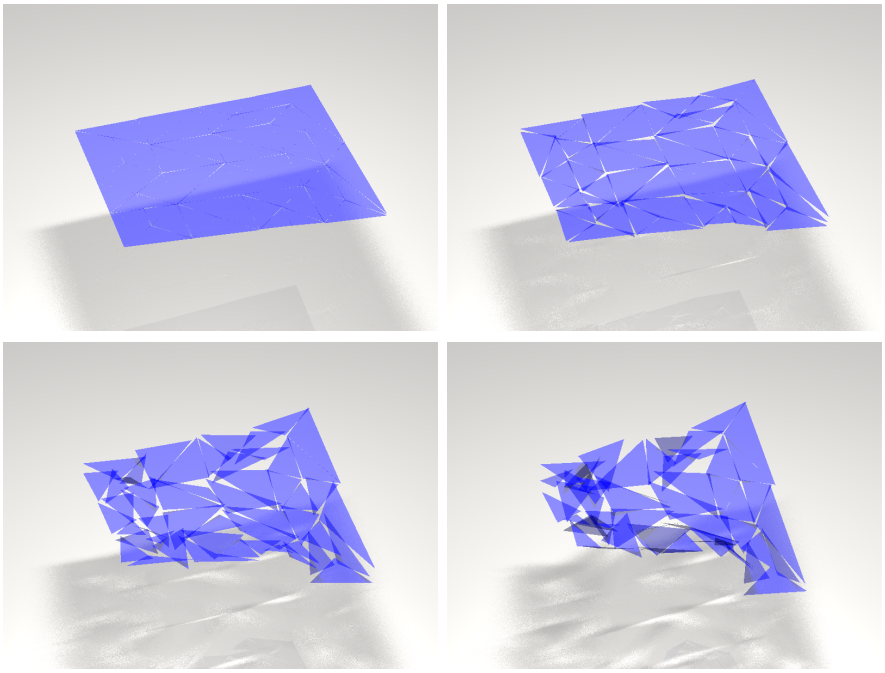


Figure 6.1: Experimental deformation of a rigid mesh. The mesh is reparametrised according to [48] to increase the number of DOF, the goal is to minimize a sum of vertices distances to the center. The runtime of this simulations is in the order of minutes. The mesh consists of 50 triangles, 150 parameters and 100 constraints.

Appendix A

List of attachements

- **images.zip** A collection of visualization images from this work in both SD and 4K resolution.
- **sourcecode.zip** Complete source code for all experiments presented here and visualization scripts.

Appendix B

Bibliography

[1] C. Hofner and G. Schmidt. "Path planning and guidance techniques for an autonomous mobile cleaning robot". In: *Robotics and Autonomous Systems* 14.2 (1995). Research on Autonomous Mobile Robots, pp. 199–212. ISSN: 0921-8890. DOI: 10.1016/0921-8890(94)00034-Y.

[2] A. Yuen, K. Zhang, and Y. Altintas. "Smooth trajectory generation for five-axis machine tools". In: *International Journal of Machine Tools and Manufacture* 71 (2013), pp. 11–19. ISSN: 0890-6955. DOI: 10.1016/j.ijmachtools.2013.04.002.

[3] M. Saska, Z. Kasl, and L. Přeucil. "Motion planning and control of formations of micro aerial vehicles". In: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress, pp. 1228–1233. ISSN: 1474-6670. DOI: 10.3182/20140824-6-ZA-1003.02295.

[4] J. H. Reif. "Complexity of the mover's problem and generalizations". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)* (1979), pp. 421–427.

[5] S. M. LaValle. *Rapidly-exploring random trees: a new tool for path planning*. Tech. rep. 1998. DOI: 10.1.1.35.1853.

[6] L. Jaillet and J. M. Porta. "Path Planning with Loop Closure Constraints Using an Atlas-Based RRT". In: *Robotics Research : The 15th International Symposium ISRR*. Ed. by H. I. Christensen and O. Khatib. Cham: Springer International Publishing, 2017, pp. 345–362. ISBN: 978-3-319-29363-9. DOI: 10.1007/978-3-319-29363-9_20.

[7] I. A. Şucan, M. Moll, and L. E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (2012-12), pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[8] Z. Kingston, M. Moll, and L. E. Kavraki. "Exploring Implicit Spaces for Constrained Sampling-Based Planning". In: *Intl. J. of Robotics Research* 38.10–11 (2019-09), pp. 1151–1178. DOI: 10.1177/0278364919868530.

[9] G. Sánchez and J.-C. Latombe. "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking". In: *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, 2003, pp. 403–417. DOI: 10.1007/3-540-36460-9_27.

[10] I. A. Şucan and L. E. Kavraki. "Kinodynamic Motion Planning by Interior-Exterior Cell Exploration". In: *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*. Ed. by G. S. Chirikjian et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 449–464. ISBN: 978-3-642-00312-7. DOI: 10.1007/978-3-642-00312-7_28.

[11] A. Shkolnik and R. Tedrake. "Path planning in 1000+ dimensions using a task-space Voronoi bias". In: *2009 IEEE International Conference on Robotics and Automation*. Kobe: IEEE, 2009-05. DOI: 10.1109/ROBOT.2009.5152638.

[12] J.-P. Laumond. "Feasible Trajectories for Mobile Robots with Kinematic and Environment Constraints". In: *Intelligent Autonomous Systems, An International Conference*. NLD: North-Holland Publishing Co., 1986, pp. 346–354. ISBN: 0444701680.

[13] D. Manocha and J. Canny. "Efficient inverse kinematics for general 6R manipulators". In: *IEEE Transactions on Robotics and Automation* 10.5 (1994-10), pp. 648–657. ISSN: 2374-958X. DOI: 10.1109/70.326569.

[14] S. Dalibard et al. "Whole-body task planning for a humanoid robot: a way to integrate collision avoidance". In: *2009 9th IEEE-RAS International Conference on Humanoid Robots*. Paris: IEEE, 2009-12. DOI: 10.1109/ICHR.2009.5379547.

[15] Z. Yao and K. Gupta. "Path planning with general end-effector constraints: using task space to guide configuration space search". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Edmonton, Alta., Canada: IEEE, 2005, pp. 1875–1880. DOI: 10.1109/IROS.2005.1545305.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968-07), pp. 100–107. ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.

[17] Y. Hwang and N. Ahuja. "A potential field approach to path planning". In: *IEEE Transactions on Robotics and Automation* 8.1 (1992-02), pp. 23–32. ISSN: 2374-958X. DOI: 10.1109/70.127236.

[18] *A Mobile Automaton: An Application of Artificial Intelligence Techniques*. International Joint Conference on Artificial Intelligence (Washington, D.C. May 7–9, 1969). 1969-01. DOI: 10.21236/ada459660.

[19] M. B. Ignatyev, F. M. Kulakov, and A. M. Pokrovskiy. *Robot-manipulator control algorithms*. NTIS Report JPRS 59717. Translated from Russian. Joint Publication Research Service, 1973.

[20] H. Niu et al. "Voronoi-Visibility roadmap-based path planning algorithm for unmanned surface vehicles". In: *J. Navig.* 72.04 (2019-07), pp. 850–874. DOI: 10.1017/S0373463318001005.

[21] E. Masehian and M. R. Amin-Naseri. "A Voronoi Diagram-Visibility Graph-Potential Field Compound Algorithm for Robot Path Planning". In: *J. Robot. Syst.* 21.6 (2004-06), pp. 275–300. ISSN: 0741-2223. DOI: 10.5555/1060023.1060024.

[22] S. Fortune. "A sweepline algorithm for Voronoi diagrams". In: *Algorithmica* 2.1-4 (1987-11), pp. 153–174. DOI: 10.1007/bf01840357.

[23] D. T. Lee and R. L. Drysdale III. "Generalization of Voronoi Diagrams in the Plane". In: *SIAM Journal on Computing* 10.1 (1981), pp. 73–87. DOI: 10.1137/0210006.

[24] L. De Floriani and E. Puppo. "An on-line algorithm for constrained Delaunay triangulation". In: *CVGIP: Graphical Models and Image Processing* 54.4 (1992), pp. 290–300. ISSN: 1049-9652. DOI: 10.1016/1049-9652(92)90076-A.

[25] B. Glavina. "Solving findpath by combination of goal-directed and randomized search". In: *Proceedings., IEEE International Conference on Robotics and Automation*. 1990, 1718–1723 vol.3. DOI: 10.1109/ROBOT.1990.126257.

[26] T. Horsch, F. Schwarz, and H. Tolle. "Motion planning with many degrees of freedom-random reflections at C-space obstacles". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 1994, 3318–3323 vol.4. DOI: 10.1109/ROBOT.1994.351060.

[27] L. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[28] "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Algorithmic and Computational Robotics*. A K Peters/CRC Press, 2001-04, pp. 303–307. DOI: 10.1201/9781439864135-43.

[29] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011-06), pp. 846–894. DOI: 10.1177/0278364911406761.

[30] G. Casella, C. P. Robert, and M. T. Wells. "Generalized Accept-Reject sampling schemes". In: *Institute of Mathematical Statistics Lecture Notes - Monograph Series*. Institute of Mathematical Statistics, 2004, pp. 342–347. DOI: 10.1214/lnms/1196285403.

[31] R. Meyer, B. Cai, and F. Perron. "Adaptive rejection Metropolis sampling using Lagrange interpolation polynomials of degree 2". In: *Computational Statistics & Data Analysis* 52.7 (2008), pp. 3408–3423. ISSN: 0167-9473. DOI: 10.1016/j.csda.2008.01.005.

[32] Z. Kingston, M. Moll, and L. E. Kavraki. "Sampling-Based Methods for Motion Planning with Constraints". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018-05), pp. 159–185. DOI: 10.1146/annurev-control-060117-105226.

[33] D. Berenson et al. "Manipulation planning on constraint manifolds". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009-05. DOI: 10.1109/robot.2009.5152399.

[34] J. M. Porta, L. Jaillet, and O. Bohigas. "Randomized path planning on manifolds based on higher-dimensional continuation". In: *The International Journal of Robotics Research* 31.2 (2012), pp. 201–215. DOI: 10.1177/0278364911432324.

[35] B. Kim et al. "Tangent bundle RRT: A randomized algorithm for constrained motion planning". In: *Robotica* 34.1 (2014-05), pp. 202–225. DOI: 10.1017/s0263574714001234.

[36] E. Anderson et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback).

[37] M. E. Henderson. "Mulitple parameter continuation: computing implicitly defined k-manifolds". In: *International Journal of Bifurcation and Chaos* 12.03 (2002-03), pp. 451–476. DOI: 10.1142/S0218127402004498.

[38] L. Jaillet and J. M. Porta. "Path Planning Under Kinematic Constraints by Rapidly Exploring Manifolds". In: *IEEE Transactions on Robotics* 29.1 (2013-02), pp. 105–117. DOI: 10.1109/tro.2012.2222272.

[39] J. Kuffner and S. LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. IEEE. DOI: 10.1109/robot.2000.844730.

[40] R. Bohlin and L. Kavraki. "Path planning using lazy PRM". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 521–528 vol.1. DOI: 10.1109/ROBOT.2000.844107.

[41] D. Hsu, J.-C. Latombe, and R. Motwani. "Path planning in expansive configuration spaces". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 3. 1997, 2719–2726 vol.3. DOI: 10.1109/ROBOT.1997.619371.

[42] B. Gipson, M. Moll, and L. E. Kavraki. "Resolution Independent Density Estimation for motion planning in high-dimensional spaces". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 2437–2443. DOI: 10.1109/ICRA.2013.6630908.

[43] J. Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671.

[44] M. M. Gulzar et al. "Kinematic modeling and simulation of an economical SCARA manipulator by Pro-E and verification using MATLAB/Simulink". In: *2015 International Conference on Open Source Systems Technologies (ICOSST)*. 2015-12, pp. 102–107. DOI: 10.1109/ICOSST.2015.7396410.

[45] H. Ananthanarayanan and R. Ordóñez. "Real-time Inverse Kinematics of (2n+1) DOF hyper-redundant manipulator arm via a combined numerical and analytical approach". In: *Mechanism and Machine Theory* 91 (2015), pp. 209–226. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2015.04.011.

[46] G. Chirikjian and J. Burdick. "Design and experiments with a 30 DOF robot". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. 1993, 113–119 vol.3. DOI: 10.1109/ROBOT.1993.291862.

[47] M. Moll, I. A. Şucan, and L. E. Kavraki. "Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization". In: *IEEE Robotics & Automation Magazine* 22.3 (2015-09), pp. 96–102. DOI: 10.1109/MRA.2015.2448276.

[48] J. Bender, R. Diziol, and D. Bayer. "Simulating Inextensible Cloth Using Locking-free Triangle Meshes." In: 2011-01, pp. 11–17. DOI: 10.2312/PE/vriphys/vriphys11/011-017.