The Raymond and Beverly Sackler
Faculty of Exact Sciences
The Blavatnik School of Computer Science

Motion Planning via Manifold Samples

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in the School of Computer Science, Tel-Aviv University

by

Oren Salzman

This work has been carried out under the supervision of
Prof. Dan Halperin

September 2011

Acknowledgments

This thesis would not have been possible without the support of many people. I wish to express my genuine gratitude to my advisor, Prof. Dan Halperin, who patiently allowed my research topic to breathe freely, all the while providing assistance, support, insight and guidance. His vision and creativity enabled new research opportunities and collaborations.

I would like to deeply thank Michael Hemmer and Barak Raveh from the Tel-Aviv University for their ongoing and fruitful collaboration. I would also like to thank all other members of the Applied Computational Geometry group at the Tel-Aviv University's School of Computer Science who offered their support as fellow researchers and as friends.

Finally, I would like to express my love and gratitude to my beloved families; to my father for nurturing me into the academic community, to my mother for not doing so, to my dear wife Kim for her endless love and support, to my dogs Bamba and Sifaka for accompanying me on long sleepless nights and to my two unborn twins for giving me a new exciting world to explore.

Work on the thesis has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

Abstract

We present a general and modular algorithmic framework for path planning of robots. Our framework combines geometric methods for exact and complete analysis of low-dimensional configuration spaces, together with sampling-based approaches that are appropriate for higher dimensions. We suggest taking samples that are *entire low-dimensional manifolds of the configuration space*. These samples capture the connectivity of the configuration space much better than isolated point samples. Geometric algorithms then provide powerful primitive operations for complete analysis of the low-dimensional manifolds. We have implemented our framework for the concrete case of a polygonal robot translating and rotating amidst polygonal obstacles. To this end, we have developed a primitive operation for the analysis of an appropriate set of manifolds using arrangements of curves of rational functions. This modular integration of several carefully engineered components has led to a significant speedup over a careful implementation of the PRM (Probabilistic Roadmap Planner) algorithm, which represents the sampling-based approach that is prevalent in practice. For example, in a tight passage scenario we demonstrate a 27-fold speedup in running time. We prove that a specific instance of the framework is probabilistically complete. We believe that this proof is an intermediate stage towards a general proof of the entire framework. As part of the implementation, we construct a set of critical curves to decompose manifolds representing the motion of a robot free to rotate while translating along a fixed segment. The curves, which are rational functions, are passed on to the CGAL (the Computational Geometry Algorithms Library) arrangement package in order to decompose the space into free and forbidden cells; thus the implementation requires the arrangement package to handle such curves. We therefore developed a C++ *traits class* to facilitate the implementation. We include experimental results and comparisons with similar traits classes which demonstrate the efficiency of our traits class and note that it is integrated into CGAL 3.9.

Contents

1 Introduction **1**

1.1 Motion Planning 1

1.2 Motion Planning Algorithms 2

1.3 Contribution and Thesis Organization 3

2 Preliminaries **7**

2.1 Arrangements 7

2.2 Motion planning 8

3 MMS Framework **17**

3.1 Manifolds Families and Primitives 18

3.2 Constraint Generation and Applying Primitives 19

3.3 Comparison With PRM - Discussion 20

4 The Case of Rigid Polygonal Motion **21**

4.1 Manifold Families and Manifold Decomposition 21

4.2 Constraint Generation and Applying Primitives 22

4.3 Path planning 23

4.4 Experimental Results 25

5 Probabilistic Completeness of MMS for a Polygonal Robot **29**

5.1 Notation 29

5.2 Probabilistic Completeness Proof 30

6 Applied Computational Geometry Background **37**

6.1 Exact Geometric Computation 37

6.2 Mathematical Background 38

7 The New Arrangement Traits Class **41**

7.1 Background 41

7.2 Existing Traits Classes 41

7.3 General Overview Of The New Traits Class 43

7.4 Traits Class Design 44

7.5 Experimental results 47

8 Conclusions and Future Work **49**

A Critical Curves of a Rotating Robot Along a Translation Segment **51**
A.1 Background 51
A.2 Critical Curves 52
A.3 Critical Endpoints 54

1 Introduction

1.1 Motion Planning

Motion planning, in the simplest of forms, refers to the computational process of moving from one place to another in the presence of obstacles. We defer the formal definition to Chapter 2 and informally consider a rigid or articulated robot R in a two-dimensional or three-dimensional workspace moving amidst static obstacles. The robot is required to move from a free initial position to a desired free target position in a *collision free path* avoiding the obstacles. We motivate the subject by presenting several applications in diverse domains. For a general overview of the subject and its application, we refer the reader to [15, 43, 45].

Robotics Numerous robots use motion planning algorithms for path planning. One example is ATHLETE, a six-legged lunar vehicle developed by the Jet Propulsion Laboratory. The vehicle rolls on wheels when possible, but can use the wheels as feet to walk when necessary, allowing for progress in rough and steep terrain. Hauser et al. present in [29] a planner to compute these motions generating a sequence of candidate footfalls and continuous motions to reach them.

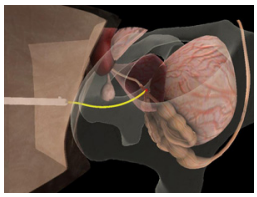
Medical Surgery Many medical procedures nowadays use robots in general and motion planning in particular to enable previously impossible operations. One such application is planning the motion of steerable needles [67]: Such needles, used in many diagnostic and therapeutic medical procedures, require planning capabilities that are often beyond the human intuition due to the needles' complex kinematics. Automatic planning algorithms are developed to enable physicians to harness the full potential of steerable needles.

Computational Biology Many problems arise in computational biology where it is necessary to reason about the motion of molecules. One such problem is drug design: A drug is most commonly a small organic molecule or a ligand that may achieve a therapeutic benefit to the patient by binding with a protein [34]. Most computational models of protein-ligand interactions consider only the energetics of the final bound state of the complex and do not examine the dynamics of the ligand as it enters the binding site. In contrast, using motion-planning techniques, Singh et al. [63] verify the existence of a low-energy path for a given ligand to a binding pocket of a protein.

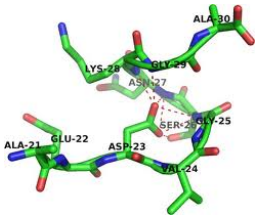
Virtual Prototyping Complex systems are designed using Computer Aided Design (CAD) applications. It is desirable to provide feedback to designers regarding the different attributes of a design such as analyzing manufacturability and serviceability. Virtual prototyping motivates the development of planners able to compute complex paths efficiently and reliably in complex geometric environments.

The next section surveys briefly the field of motion planning. Although formal definitions are deferred to Chapter 2, the section attempts to provide insights on the various approaches used to design motion planning algorithms. It starts by informally introducing the problem and continues to discuss the two principal

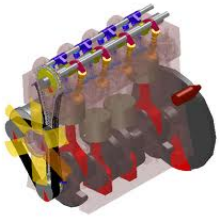
(a) ATHLETE lunar robot



(b) Steerable needle



(c) Linkage representation of a molecule



(d) Virtual prototyping of an engine

Figure 1.1: Motion planning applications

approaches in the field, the combinatorial approach and the sampling-based approach. Some methods, including the one presented in this thesis, attempt to combine the two approaches. These approaches are surveyed in an attempt to present the foundations that led to this work.

1.2 Motion Planning Algorithms

In motion planning, we consider a robot R moving in the plane or in space. There may be obstacles that the robot should not intersect while moving and restrictions on the set of valid movements that the robot may perform. The valid movements of the robot may include translations (movement in the space without changing the orientation), rotations (rotation around a specific point or axis or the rotation of joints), a combination of the above and more. The spatial pose of a robot R may be described by several parameters depending on the possible movements of the robot. If R can only translate then the spatial position of a specific point in R, its *reference point*, is used to specify its location. If it can also rotate around a specific point or if it has rotatable joints, the amount of counterclockwise rotation relative to an initial orientation is another parameter that needs to be considered.

We refer to the set of parameters defining the spatial pose of a robot R as the robot's *configuration*. The minimal number of parameters needed to uniquely define a configuration is termed the number of degrees of freedom (*dof*s) of R. The set of all robot configurations $\mathcal{C}$ is termed the *configuration space* of the robot, and decomposes into the disjoint sets of free and forbidden configurations, namely $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$, respectively.

Analytic solutions to the general motion planning problem. The motion-planning problem is computationally hard with respect to the number of *dofs* [56], yet much research has been devoted to solving both the general problem and its various instances using a combination of geometric, algebraic and combinatorial tools. The configuration-space formalism was introduced by Lozano-Perez [49]. Then, in the seminal "piano movers" series of articles [58, 59], Schwartz and Sharir introduced the notion of configuration-space *critical curves* or *critical (hyper-)surfaces*, which identify potential transitions between robot configurations in $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$ in closed-form, decomposing the configuration space into a finite set of cells that are fully contained in either $\mathcal{C}_{\text{free}}$ or $\mathcal{C}_{\text{forb}}$. Using this approach, Schwartz and Sharir proposed the first general algorithm for solving the motion planning problem, with running time that is doubly-exponential in the number of *dof*s [59]. This was later improved to exponential running time using a decomposition method by Chazelle et al. [14]. Another exponential time algorithm for the general motion-planning problem was proposed by Canny [11] and improved by Basu et al. [5]. The latter approach is generally considered too complicated to be implemented in practice.

Solutions to low-dimensional instances of the problem. Although the motion-planning problem cannot be solved analytically in an efficient manner for the general case, more efficient algorithms have been proposed for various low-dimensional instances [43, 60]. In [48] Lozano-Perez presents Minkowski sums (discussed in detail in Chapter 2) as a method to compute the configuration space of translating convex polygons and polyhedra and suggests an algorithm to compute the Minkowski sum of two convex sets. In addition, algorithms for approximating the configuration space of rotating polyhedra are presented in [48]. The bounds for computing the Minkowski sum of two convex polyhedral robots in three-dimensional space were improved by Aronov and Sharir in [1]. Improved bounds for the case of translation with rotation of a

polygonal robot in the plane are presented in [4, 28].

Implementing algorithms in the field of computational geometry revealed an interesting gap between theory and practice leading to the development of a subdiscipline of Applied Computational Geometry. This gap will be discussed in Chapter 6 yet it is worth noting that many fundamental computational-geometry algorithms were only recently implemented robustly and exactly. This is due to advances in applied computational geometry in recent years that have led to a set of implemented tools that are of interest in this context. CGAL, the Computational Geometry Algorithms Library incorporates many such tools [65]. For instance, Minkowski sums, have a fast, robust and exact planar and three-dimensional implementations [21, 26, 69]. Likewise, CGAL offers implementations of planar arrangements[1] of curves [65, C.30] (to which the Tel-Aviv University group is a major contributor), and for cylindrical algebraic decomposition of algebraic surfaces in $\mathbb{R}^3$ [9]. These are essential components in the original theoretical algorithm by Schwartz and Sharir from 1983 [59]. For a general survey of related approaches see [60].

Sampling-based approaches to motion planning. In recent years, the sampling-based approach to motion-planning has extended the applicability of motion planning algorithms beyond the restricted subset of problems that can be efficiently solved by exact algorithms [15, 45]. Sampling-based motion planning algorithms, such as Probabilistic Roadmaps (PRM) [36], Expansive Space Trees (EST) [32] and Rapidly-exploring Random Trees (RRT) [44] as well as their many variants, aim to capture the connectivity of $\mathcal{C}_{\text{free}}$ in a graph data structure via random sampling of robot configurations. This can be done either in a multi-query setting, to efficiently answer multiple queries for the same scenario, as in the PRM algorithm, or in a single-query setting, as in the RRT and EST algorithms. For a general survey on these approaches see [15, C.7]. Importantly, the PRM and RRT algorithms were both shown to be probabilistically complete [32, 35, 39, 40], that is, they are guaranteed to find a valid solution, if one exists, given an infinite amount of time. However, the required running time for finding a valid solution cannot be computed for new queries at run-time in practice, and the proper usage of sampling-based approaches may still be considered somewhat of an art. Moreover, sampling-based methods are also considered sensitive to tight passages in the configuration space, due to the high probability of missing such passages.

Hybrid methods for motion-planning: The analytic solutions, while complete, are complex and almost impractical for high-dimensional problems. The probabilistic methods, on the other hand, are simple and work well in very high dimensions when there exists a path with high clearance[2]. Few hybrid methods attempt to combine deterministic and probabilistic planning strategies. Hirsch and Halperin [31] study two-disc motion planning by exactly decomposing the configuration space of each robot, then combining the two solutions to a set of free, forbidden and mixed cells, and using PRM to construct the final connectivity graph. Zhang et al. [74] use PRM in conjunction with approximate cell decomposition, which also divides space into free, forbidden and mixed cells. Other studies suggest to connect a dense set of near-by configuration space "slices". Each slice is decomposed into free and forbidden cells, but adjacent slices are connected in an inexact manner, by e.g., identifying overlaps between adjacent slices [17, pp. 283-287], or by heuristic interpolation and local-planning [47]. In [71] a 6 *dof* RRT planner is presented with a 3 *dof* local planner hybridizing probabilistic, heuristic and deterministic methods.

1.3 Contribution and Thesis Organization

In this thesis, we describe a novel general scheme for motion planing via manifold samples (MMS), which extends sampling-based techniques like PRM. The scheme has been presented recently in the European symposium on Algorithms (ESA) 2011 [57] and is based on joint work with Michael Hemmer, Barak Raveh and Dan Halperin.

Instead of sampling isolated robot configurations, we sample *entire low-dimensional manifolds*, which can be analyzed by complete and exact methods for decomposing space. This yields an explicit representation of maximal connected patches of free configurations on each manifold, and provides a much better coverage

[1] A subdivision of the plane into zero-dimensional, one-dimensional and two-dimensional cells, called vertices, edges and faces, respectively induced by the curves. See also Section 2.1.

[2] The clearance at point p along a path is the minimal distance from p to an obstacle. The clearance of a path is the maximal clearance of all points along the path.

of the configuration space compared to isolated point samples. At the same time, the manifold samples are deliberately chosen such that they are likely to intersect each other, which allows to establish connections among different manifolds.

We present an application of MMS to the concrete case of a polygonal robot translating and rotating in the plane amidst polygonal obstacles. We present in detail appropriate families of manifolds as well as filtering schemes that should also be of interest for other scenarios. We present an exact analytic solution and efficient implementation to a motion planning problem instance, which to the best of our knowledge has not been analytically studied before. Namely, that of moving a polygonal robot in the plane with rotation and translation *along an arbitrary segment*, which involves advanced algebraic and extension of state-of-the-art applied-geometry tools.

To ease the reading of the thesis, it is divided into two parts: The first, oriented to readers familiar with motion planning, presents the new motion-planning scheme, its analysis, a specific implemented instance including technical details and experimental results. The second part, assumes some familiarity with recent developments on the applied side of Computational Geometry, discusses in detail a C++ package. The package, which was integrated into CGAL, was developed to facilitate the implementation of the motion planner discussed in the first part of the thesis. Appendix A provides details on the analytic solution for the case of a rotating polygonal robot while translating along an arbitrary line segment.

Part I

Motion Planning

We present a general and modular algorithmic framework for path planning of robots. Our framework combines geometric methods for exact and complete analysis of low-dimensional configuration spaces, together with sampling-based approaches that are appropriate for higher dimensions. We suggest taking samples that are *entire low-dimensional manifolds of the configuration space*. These samples capture the connectivity of the configuration space much better than isolated point samples. Geometric algorithms then provide powerful primitive operations for complete analysis of the low-dimensional manifolds. We have implemented our framework for the concrete case of a polygonal robot translating and rotating amidst polygonal obstacles. This modular integration of several carefully engineered components has led to a significant speedup when compared to a common implementation of the sampling-based PRM algorithm, which represents an approach that is prevalent in practice. For example, in a tight passage scenario we demonstrate a 27-fold speedup in running time. We conclude with a proof of probabilistic completeness for the framework for a polygonal robot translating and rotating in the plane.

2
Preliminaries

This chapter provides background material and definitions required for the first part of the thesis. Section 2.1 provides an introduction to Arrangements, which play a fundamental role in computational geometry. In the context of this work, arrangements are used to explicitly represent parts of the configuration space intersected by manifolds. Section 2.2 provides a formal definition of the motion planning problem. It outlines the Probabilistic Motion Planner (PRM) algorithm, essential for understanding the thesis, as this work stems from the PRM algorithm.

2.1 Arrangements

Arrangements serve as a fundamental building block in computational geometry in general and in motion planning algorithms in particular. They allow a discretization of a continuous space in an efficient manner without loosing any information regarding the space. Examples of such algorithms will be presented in Section 2.2 and will be used throughout this work. For a full survey on the subject of arrangement see [61].

Given a finite set of hyper-surfaces $\mathcal{C}$ in $\mathbb{R}^d$, the arrangement $\mathcal{A}(\mathcal{C})$ is a subdivision of $\mathbb{R}^d$ into maximal open connected cells of dimension $0, 1, \ldots, d$ induced by $\mathcal{C}$. In the sequel we limit ourselves to planar arrangements unless stated otherwise. To demonstrate a planar arrangement we refer the reader to Figure 2.1. One can see four faces (snowman's body, head, nose and unbounded face), six curves (two circles creating the snowman's body and face, two segments creating the snowman's hands and two segments creating the snowman's nose), two isolated vertices (creating the snowman's eyes) and five intersection points.

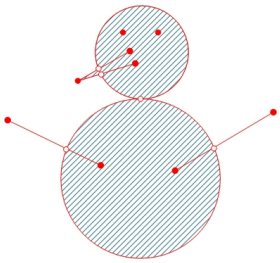


Figure 2.1: Snowman arrangement

If we denote by n the number of curves in the arrangement and assume that they are all algebraic curves of maximum constant degree, then the combinatorial complexity of a planar arrangement (the complexity of the vertices, edges and faces) is at most $O(n^2)$. This bound is tight in the worst case — for example a grid of $n/2$ horizontal segments intersecting $n/2$ vertical segments. The combinatorial complexity of a single face in an arrangement of n low-degree algebraic curves in the plane is only $O(\lambda_s(n))$ for some constant parameter s that depends on the degree of the curves; here $\lambda_s(n)$ is the nearly-linear maximum length of (n, s) Davenport-Schinzel sequences [25].

As each face in an arrangement may be complex, many computational geometry algorithms require further refinement of a given arrangement. A common method is vertical decomposition. Vertical decomposition is a subdivision of each cell of the arrangement into trapezoids by extending a vertical segment upwards and downward from each vertex until the extension touches a feature of the arrangement (see Figure 2.2 for an example of a vertical decomposition of an arrangement of segments). For a thorough description of vertical decompositions and common applications see [17]. As opposed to the original arrangement, in the decomposed arrangement, each face is bounded by up to four curves (hence having constant combinatorial complexity): The bottom of the face (*floor* curve), the top of the face (*ceiling* curve) and possibly two vertical segments at the sides (referred to as the *walls* of the face). Moreover, if the curves are segments, each cell in the decomposition is convex, which allows for queries on the cells to be answered quickly. Section 2.2 presents an application of vertical decomposition in motion planning.

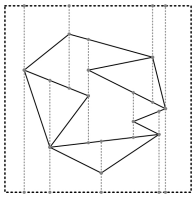


Figure 2.2: Vertical decomposition

2.2 Motion planning

2.2.1 Background

We refer to the *workspace* W as a subset of the two- or three-dimensional physical space: $W \subset \mathbb{R}^k$ *s.t* $k \in \{2, 3\}$; thus the workspace is the physical space in which the robot is moving. It is usually convenient (and reasonable) to assume that W is bounded by a two- or three-dimensional box. The *robot* R is a collection of rigid objects. The workspace also consists of *obstacles* that are a collection of rigid objects that R should not intersect.

A *configuration* c is a set of parameters specifying the position and orientation of all the bodies composing R. The *Configuration Space* C is the set of all configurations and the number of *Degrees of Freedom (dof)* of R is the minimal number of independent parameters needed to uniquely define each configuration. Thus a (complex) robot in the two- or three-dimensional space can be represented as a point in a (possibly) higher-dimensional parameterized space. We term a configuration as *collision free* if it represents a placement of the robot in W such that R does not intersect any obstacle, and does not violate the mechanical limitations of the robot. A configuration is said to be *semi-free* if it represents a placement of the robot in W such that R "grazes" an obstacle but does not intersect any other obstacle. The configuration space C is thus divided into the two disjoint sets: The *free space* $\mathcal{C}_{\text{free}}$ which is the open set of configurations that are *collision free* and the *forbidden space* $\mathcal{C}_{\text{forb}} = C \setminus \mathcal{C}_{\text{free}}$.

We demonstrate the notion of configuration space by considering two simple scenarios in a two-dimensional workspace. Let R be a polygonal robot such that the movements of R are restricted to translations only. Let p_r be a point in R (any point in W would suffice), every point in R is at constant distance from p_r wherever the R is placed in W (the orientation with respect to p_r does not change either). Thus the placement of R is uniquely defined by specifying the placement of p_r. In this case the configuration space is two-dimensional and a configuration is represented by two parameters x and y: the location of the reference point p_r in W. If we allow the robot to rotate as well as translate, to uniquely identify the placement of R, the amount of counterclock rotation with regards to an initial orientation should be specified. In this case the configuration space is three-dimensional. A configuration is represented by two parameters x and y for the location of the reference point in W and a third parameter θ for the amount of counterclock rotation with regards to an initial orientation.

A *path* γ is a continuous mapping $\gamma : [0, 1] \to C$ and a *free path* is a path γ such that every configuration in the path is in $\mathcal{C}_{\text{free}}$. This allows for a formal definition of the basic path planning problem—finding a free path for a robot amidst obstacles in a workspace from a given free source configuration to a desired free target configuration. There are many variants and extensions to the definitions above, a robot, for example may not be rigid and the obstacles may move with time.

Several families of algorithms have been proposed to address the path planning problem, these families include Potential Fields, Sampling-Based Motion Planning, Visibility Graphs and more. We survey several families, for a thorough introduction to the field see, e.g., [43].

2.2.2 Combinatorial Motion Planning

Combinatorial motion planning approaches find paths through the continuous configuration space without resorting to approximations. Some of the methods are impractical for high-dimensional configuration spaces but for one of the simplest, most fundamental of motion-planning problems, namely a robot translating amidst obstacles, an exact and efficient combinatorial motion planning algorithm exists that uses *Minkowski Sums*.

Minkowski Sums

Given two sets $P, Q \in R^d$, their Minkowski sum, denoted as $P \oplus Q$, is their point-wise vector sum, namely the set: $P \oplus Q = \{p + q | p \in P, q \in Q\}$. Figure 2.3 demonstrates the Minkowski sum of a rectangle and a triangle in the plane. Minkowski sums are used in many applications, such as motion planning, computer-aided design and manufacturing. The complexity of the Minkowski sum of two planar polygons P, Q with m and n vertices respectively is $\Theta(m+n)$ if both P and Q are convex, $\Theta(mn)$ in the case one of the two is convex [37] and $\Theta(m^2n^2)$ in the general case.

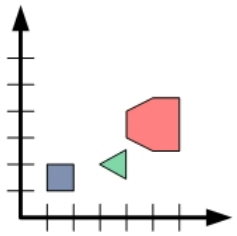


Figure 2.3: Minkowski sum of a rectangle and a triangle

Minkowski sums can be used to detect collision between two sets and compute their relative placement. These basic operations play an important role in solving motion-planning problems as will be demonstrated below. Let P and Q be two point sets in the plane having a non-empty intersection, let $p \in P \cap Q$ and let $-Q$ denote the reflection of Q about the origin. Note that since $p \in Q$, then $-p \in -Q$ and thus $P \oplus (-Q)$ contains the origin as $(0,0) = p + (-p)$. The reverse direction is also true: if $(0,0) \in P \oplus (-Q)$, then $P \cap Q \neq \emptyset$. Thus, the sum $P \oplus Q$ is the configuration-space obstacle induced by the obstacle P with respect to the robot Q.

These properties of Minkowski sums lead to immediate applications in combinatorial motion planning for a translating polygonal robot:

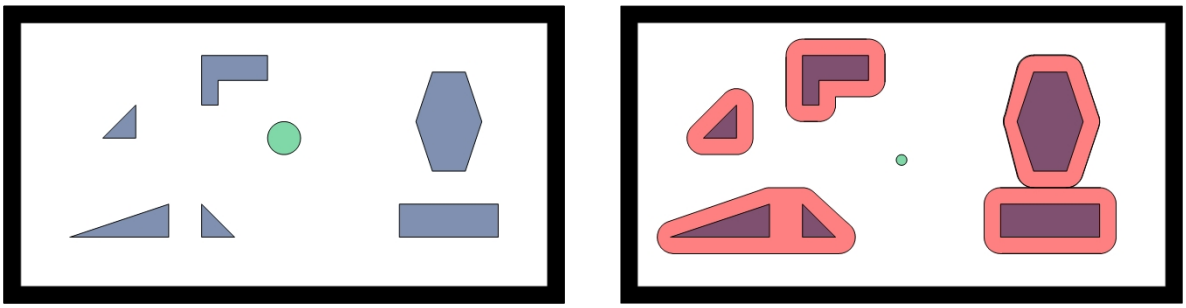
- **Collision detection**: Detecting collision between a robot R and an obstacle O can be done by testing if the origin lies inside the Minkowski sum of the reflected robot and the obstacle $M = -R \oplus O$.
- **Computing $\mathcal{C}_{\text{forb}}$**: Let $P_1, \ldots P_m$ be the set of polygonal obstacles and let $r_0 \in R$ be a reference point on the robot R. Translating R such that r_0 coincides with the origin, and reflecting the robot about the origin leads us to $-R$. Thus $\mathcal{C}_{\text{forb}} = \bigcup_m^{i=1} O_i \oplus -R$.
- **Determining if a collision-free path exists**: given a free start location s and a free target location t, we wish to determine whether there exists a collision-free path connecting s and t. We simply compute $\mathcal{C}_{\text{forb}}$, and consider its complement, $\mathcal{C}_{\text{free}} = \mathbb{R}^2 \setminus \mathcal{C}_{\text{forb}}$. A path exists between s and t iff these two points lie in the same connected component of $\mathcal{C}_{\text{free}}$.

Figure 2.4a demonstrates a workspace consisting of polygonal obstacles (blue) and a disk robot (green), while Figure 2.4b illustrates the corresponding configuration space computed using the Minkowski sum of the reflected robot and the obstacles. The configuration space representation reduces the motion-planning problem to a point in the plane translating amidst the inflated obstacles.

Once the configuration space is computed, determining if a collision-free path exists is straightforward as demonstrated. Computing the path may be slightly more complicated. Several approaches exist, all represent the continuous motion planning problem discretely without losing any of the original connectivity information needed to solve it.

Planar Roadmap Generation

In this section we review several techniques for computing a path for a point amidst polygonal obstacles. The techniques discretize a continuous space by generating a *roadmap*—a graph-like representation of the space at hand. The motivation for examining this simple case may come either from the workspace (applications



(a) Workspace (b) Corresponding configuration space

Figure 2.4: Workspace and corresponding configuration space of a disk

considering a point robot) or the configuration space (a configuration is a point representation of a complex robot as demonstrated above). In particular, Minkowski Sums provide us with a reduction from polygon motion to point robot motion. We will describe three methods for planning a path for a point:

- Cell Decomposition Roadmap
- Shortest Path Roadmap
- Maximum Clearance Roadmap

The Cell Decomposition Roadmaps The basic idea behind this method is that a path between the initial configuration and the goal configuration can be determined by subdividing the free space into smaller regions called cells. After this decomposition, a connectivity graph is constructed according to the adjacency relationships between the cells, where the nodes represent the cells in the free space, and the edges between the nodes show that there is a free crossing between the corresponding cells. From this connectivity graph, a continuous path can be determined by simply following adjacent free cells from the source point to the target point. Planning a path inside a decomposed cell is usually trivial due to the low complexity of the decomposed cell. A vertical decomposition as described in Section 2.1 is a common choice for cell decomposition. Figure 2.5 demonstrates a decomposition of the free space computed using Minkowski sums into cells and a path planned in the decomposed free space.

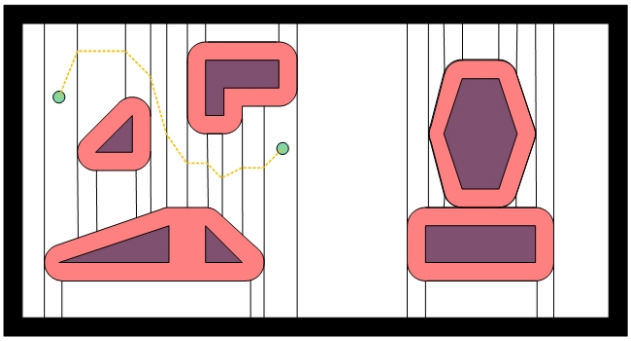
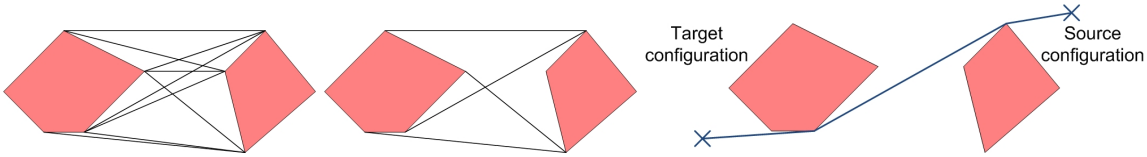


Figure 2.5: Path in the decomposed free space

Shortest-Path Roadmaps A simple and powerful algorithm exists for computing the shortest path between two points as introduced first in [51]. To consider this algorithm one must expand the notion of the free space $\mathcal{C}_{\text{free}}$ which was defined as an open set to include its boundaries. We must consider the problem of determining shortest paths in the closure of $\mathcal{C}_{\text{free}}$. This means that the robot is allowed to "touch" or "graze" the obstacles, but it is not allowed to penetrate them. A *semi-free* path is defined as a path that is in the closure of $\mathcal{C}_{\text{free}}$.



(a) Visibility graph (b) Reduced visibility graph (c) Shortest path

Figure 2.6: Visibility graph and shortest path

Let $P = P_1, \ldots, P_m$ be a set of simple pairwise interior-disjoint polygons having n vertices in total. The *visibility graph* of P is an undirected graph defined on the set of polygon vertices, whose set of edges consists of those pairs of vertices that are mutually visible. Two vertices are *mutually visible* if the straight line segment connecting them does not intersect the interior of any of the polygons in P. In this case, we call this segment a *visibility edge*. The visibility graph (consisting of the vertices and all the visibility edges) can be used to compute shortest paths amidst polygonal obstacles, where the polygons are considered as open sets. Each edge is given a weight equal to the Euclidean distance between its two end-vertices. To find a shortest path between a source and a target configuration, one simply needs to connect them to the visibility graph and execute Dijkstra's algorithm [18] starting from the vertex representing the source configuration. In fact, it is sufficient to consider only the edges that are bitangent to the polygons they connect, namely edges that can be infinitesimally extended in both directions without penetrating any polygon. Such bitangent edges are called *reduced* visibility edges (see Figure 2.6 for an illustration).

The path created using the visibility graph is the shortest path between the source and the target configurations. As such, the robot's *clearance* (namely the minimal distance from the obstacles) is zero in the general case. This may be undesirable in some applications.

Maximum-Clearance Roadmaps A *maximum clearance roadmap* tries to keep the robot as far from the obstacles as possible. For a point robot this means moving along the bisectors of the *Voronoi Diagram*: Let $O = \{o_1, \ldots, o_n\}$ be a set of n objects referred to as *Voronoi sites*. The *Voronoi diagram* Vor(O) is the partition of space into maximally connected cells where each cell consists of the points that are closest to one particular site (or a set of sites). These cells are referred to as *Voronoi cells*. The *bisector* $B(o_i, o_j)$ of two Voronoi sites o_i and o_j is the locus of points that have an equal distance to both sites. That is $B(o_i, o_j) = \{p | dist(o_i, p) = dist(o_j, p)\}$. An edge in the Voronoi diagram is thus part of a bisector of two sites such that there does not exist a third site closer to any point on the edge.

The Voronoi diagram can be used to compute paths of maximal clearance from the obstacles. The source and target configurations are added to the roadmap by connecting them to the closest bisector. It can be shown that the total complexity of a planar Voronoi diagram is $O(n)$ where n is the total number of polygon vertices and that it can be constructed in $O(n \log n)$ time (see e.g. [2, 72]).

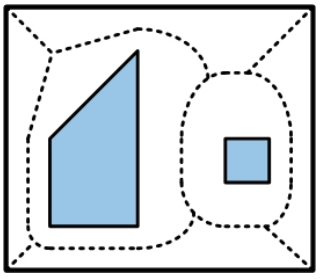
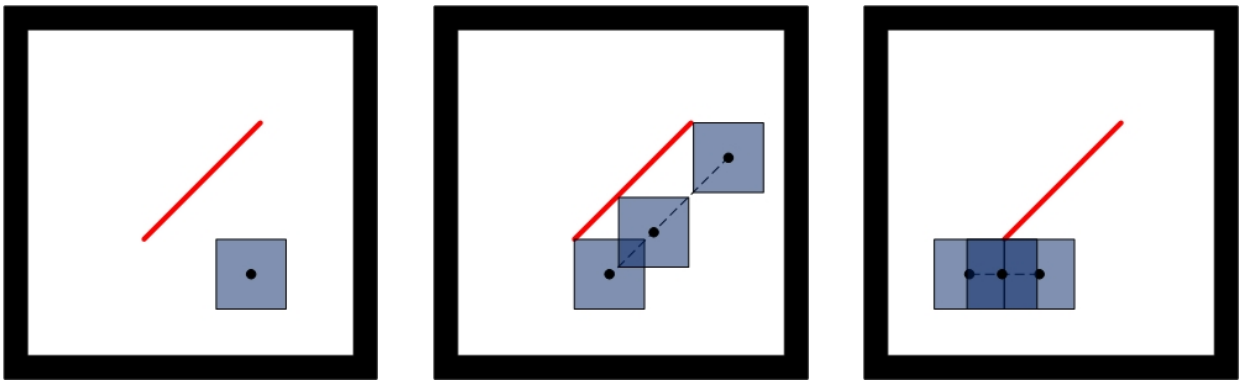


Figure 2.7: Maximum-clearance roadmap of two obstacles



(a) Workspace

(b) Robot's vertex - obstacles's edge critical curve

(c) Robot's edge - obstacles's vertex critical curve

Figure 2.8: Critical curve (in a dashed line) example

Explicit General Representation of the Configuration Space

An explicit representation of the configuration space for the general case may be extremely complicated as it may involve curves of high algebraic complexity, yet for some cases an explicit representation of the configuration space (or part of it) is needed. Computing the configuration space explicitly relies on the notion of *critical curves* initially presented in [59] and arrangements (see Section 2.1). A simple observation on the structure of the configuration space is that the changes between $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$ occur when the robot is tangent to an obstacle.

In a two-dimensional configuration space, the changes between $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$ occur when a robot's vertex coincides with an obstacle's edge or when a robot's edge coincides with an obstacle's vertex. These two cases will be referred to as vertex-edge and edge-vertex, respectively. These incidences are represented by curves in the configuration space. If all these critical curves are inserted in to an arrangement, each of the arrangement cells is completely in $\mathcal{C}_{\text{free}}$ or in $\mathcal{C}_{\text{forb}}$. This gives a discretization of the continuous configuration space without loosing any data.

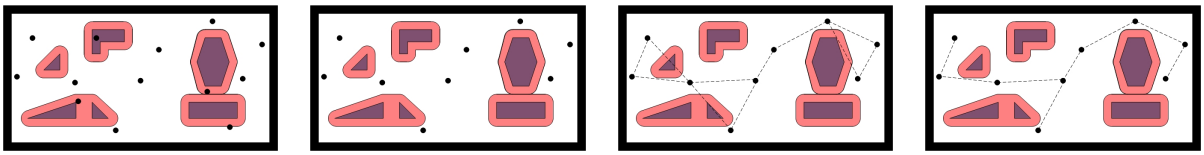
This is best demonstrated by a simple example, in Figure 2.8, on the left-hand side, we see a square robot R and a line-segment obstacle that R should avoid. The middle figure demonstrates a critical curve formed by translating R while one vertex is tangent to the obstacle's edge. The right-hand figure demonstrates a critical curve formed by translating R while one edge is tangent to the obstacle's vertex.

Although the method holds for any configuration space of degree d, it is impractical for large d as the induced critical surfaces may be of dimension $d-1$ and the arrangement's complexity is exponential in d in the worst case [56].

2.2.3 Sampling-Based Motion Planning

Sampling-based motion planning algorithms attempt to capture the connectivity of the free space by sampling configurations, hence avoiding an explicit representation of the configuration space. These algorithms construct roadmaps similar to those discussed in the previous section. Typically they are faster than analytic methods yet they lack the complete information regrading the structure of the configuration space. Sampling-based approaches can be divided into two families: The first, attempts to construct a roadmap of the entire free space (hence suitable for multi-query problems) while the second, attempts to incrementally explore the free space by attempting to connect a source configuration and a target configuration (hence more suitable for single-query problems).

We will describe an algorithm from the first family: the Probabilistic Roadmap Planner (PRM). Algorithms from the second family share basic common features applied in a different fashion. For a recent survey on the subject of sampling-based motion planning see [66]. A PRM-type algorithm consists of two stages: A preprocessing stage and a query stage.



(a) PRM sampled milestones (b) PRM valid milestones (c) PRM local paths (d) PRM roadmap

Figure 2.9: PRM example

Probabilistic Roadmap Planner The preprocessing stage constructs a data structure, referred to as a *Roadmap*, which is a graph where the vertices are the free configurations and the edges represent valid paths between pairs of free configurations. Algorithm 1 outlines the sampling stage: the configuration space is sampled to produce free configurations or *milestones* (line 4). The algorithm typically uses a *collision detector* to test if the sampled configuration is collision free or not (line 5); configurations which are not collision free are discarded. Collision-free configurations are added to the roadmap as outlined in Algorithm 2: neighboring configurations (usually k nearest neighbors, where k is a parameter of the algorithm) are located (line 2). For each such neighbor, a *local planner* attempts to connect the two configurations (line 4) and if a path exists an edge is added to the graph. The *local planner* typically uses dense sampling between the configurations in order to determine if a collision free path exists.

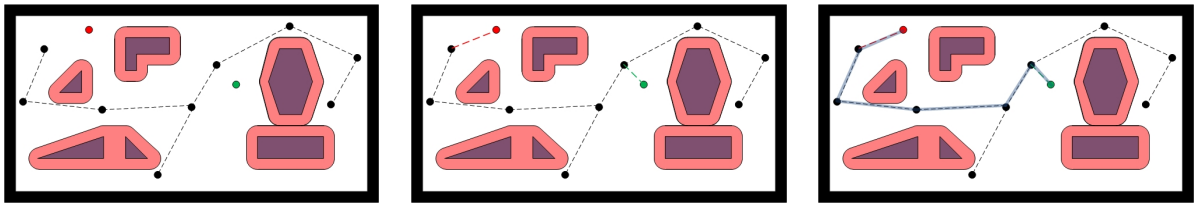
Figure 2.9 demonstrates the preprocessing stage[1] of the PRM algorithm applied to the configuration space illustrated in Figure 2.4b. Figure 2.9a illustrates a set of configurations (both valid and invalid) sampled by the PRM algorithm . Figure 2.9b illustrates the discarding of invalid samples by the algorithm leaving only a set of collision-free samples. Figure 2.9c illustrates the attempts to connect close-by configurations by a local planning including valid and invalid local paths while Figure 2.9d illustrates the final roadmap after the invalid local paths are discarded.

In the query stage (Algorithm 3) the source and target configurations are added to the graph and connected to nearby configurations using the local planner. Then, the graph is searched for a path between the nodes to yield a free path in the configuration space. This is illustrated in Figure 2.10 where Figure 2.10a illustrates the query, Figure 2.10b illustrates the additional edges added to the roadmap, and Figure 2.10c illustrates the solution.

The algorithm's variants include different sampling strategies and different connection strategies. The reader is referred to [45, C. 5] for an extensive discussion on the variants and their properties.

As discussed in [33] the PRM uses only two *probes*: the collision detector and the local planner. Hence the roadmap is built by considering a restricted portion of the configuration space at each step. This gives sampling-based planners the ability to create a roadmap representing the connectivity of $\mathcal{C}_{\text{free}}$ without explicitly considering the complete structure of $\mathcal{C}_{\text{free}}$ as opposed to combinatorial planners. Yet this also reveals the main drawback of sampling-based planners. In Figure 2.11 [33], we see a very simple example for which the PRM algorithm performs poorly. The narrow corridor in the workspace is a narrow passage in the configuration space. The probability of the PRM algorithm to sample configurations within the narrow

[1]The figure does not follow Algorithms 1 and 2 exactly. Algorithms 1 and 2 consider configurations one at a time while the figure considers sets of configurations for ease of visualization.



(a) PRM sampled milestones (b) PRM valid milestones (c) PRM roadmap

Figure 2.10: PRM example

Algorithm 1 build_roadmap $(G = (V, E), k)$

```
1: V ← {}
2: E ← {}
3: loop
4:    c ←sample_configuration()
5:    if collision_free (c) then
6:       add_configuration(G, c, k)
7:    end if
8: end loop
```

Algorithm 2 add_configuration $(G = (V, E), c, k)$

```
1: V ← V ∪ {c}
2: N_c ←nearest_neighbors(V, c, k)
3: for all n ∈ N_c do
4:    if local_planner (c, n) ≠ ∅ then
5:       E ← E ∪ {(c, n)}
6:    end if
7: end for
```

Algorithm 3 query $(G = (V, E), s, t)$

```
1: if s ∉ V then
2:    add_configuration(G, s, k)
3: end if
4: if t ∉ V then
5:    add_configuration(G, t, k)
6: end if
7: return find_path(G, s, t)
```

passage is low due to the measure of the passage. Moreover, the probability to connect two configurations that lie in the narrow passage is low due to the shape of the passage.

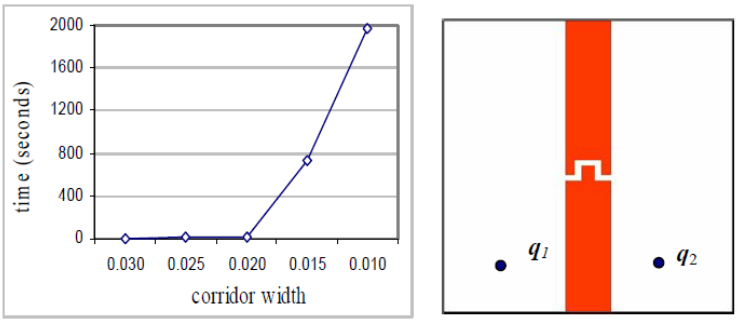


Figure 2.11: The plot shows the average running time for a PRM algorithm to connect configurations q_1 and q_2 as the width of the corridor decreases. Figure taken from [33].

Probabilistic Completeness Although the PRM algorithm may perform poorly for some practical cases, it has the appealing attribute of *probabilistic completeness*: The algorithm is guaranteed to find a valid solution, if one exists, given an infinite amount of time.

The following theorem is proved in [15, C.7] for a PRM operating in Euclidean $\mathbb{R}^d$, where Q_{free} denotes the free subset of $[0,1]^d$, *clr*(γ) is the clearance of a path and the measure μ is the volume of space:

Theorem 2.2.1. *Let $a, b \in Q_{free}$ such that there exists a path γ between a and b. Then the probability of PRM correctly answering the query (a, b) after generating n configurations is given by*

$$Pr[(a,b)\text{Success}] = 1 - \Pr[(a,b)\text{Failure}] \geq 1 - \left\lceil \frac{2L}{\rho} \right\rceil e^{-\sigma^d n}.$$

Where L is the length of the path γ, $\rho = clr(\gamma)$, $B_1(\cdot)$ is the unit ball in $\mathbb{R}^d$ and $\sigma = \frac{\mu(B_1(\cdot))}{2^d \mu(Q_{free})}$

3
MMS Framework

This chapter describes a general framework for solving motion-planning problems: Motion Planning via Manifold Samples or MMS for short. We propose a multi-query planner for path planning problems of a possibly high-dimensional configuration space. The framework combines analytic methods used to exactly represent sub-parts of the configuration space with PRM-like probabilistic methods used to represent the configuration space and search in it.

Preprocessing—Constructing the connectivity graph The preprocessing stage constructs a data structure capturing the connectivity of $\mathcal{C}$ using manifolds as samples. The manifolds are decomposed into cells in $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}}$ in a complete and exact manner; we call a cell of the decomposed manifold that lies in $\mathcal{C}_{\text{free}}$ a *free space cell* (FSC) and refer to the data structure $\mathcal{G}$ as the connectivity graph. The FSCs serve as nodes in $\mathcal{G}$ while two nodes in $\mathcal{G}$ are connected by an edge if their corresponding FSCs intersect. See Figure 3.1 for an illustration.

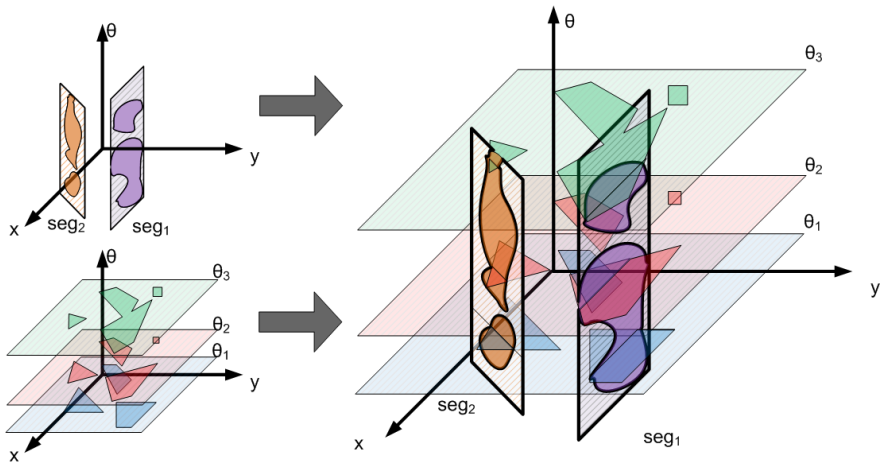


Figure 3.1: Three-dimensional configuration space: The left side illustrates two families of manifolds where the decomposed cells are darkly shaded. The right side illustrates their intersection that induces the connectivity graph $\mathcal{G}$.

We formalize the preprocessing stage by considering manifolds induced by a family of constraints Ψ, such that $\psi \in \Psi$ defines a manifold m_ψ of the configuration space and FSC_{m_ψ} is the set of FSCs of m_ψ.

The construction of a manifold m_ψ and its decomposition into FSCs are carried out via an appropriate Ψ-primitive, denoted P_Ψ, applied to an element $\psi \in \Psi$. By a slight abuse of notation we refer to an FSC both as a cell and as a node in the graph. For example, a family of constraints is obtained by fixing all but one of the *dofs* of R, inducing a set of 1-manifolds in $\mathcal{C}$. The primitive in this case will decompose the 1-manifolds into intervals completely in $\mathcal{C}_{\text{free}}$ or completely in $\mathcal{C}_{\text{forb}}$.

Using this notation, Algorithm 4 summarizes the construction of $\mathcal{G}$. In lines 3-4, a new manifold constraint is generated and added to the collection X of manifold constraints. In lines 5-7, the manifold induced by the new constraint is decomposed by the appropriate primitive and its free cells are added to the graph.

Query Once the connectivity graph $\mathcal{G}$ has been constructed it can be queried for paths between two free configurations q_s and q_t in the following manner: A manifold that contains q_s (respectively q_t) in one of its FSCs is generated and decomposed. Its FSCs and their appropriate edges are added to $\mathcal{G}$. $\mathcal{G}$ is traversed from the FSC containing q_s to the FSC containing q_t. The result of this traversal is a list of FSCs $L_{FSC} = \text{fsc}_1 \ldots \text{fsc}_n$ and *not* a path in the configuration space. The framework guarantees that each such node fsc_i is completely in $\mathcal{C}_{\text{free}}$ and that every two consecutive nodes intersect.

An actual path is then computed by locally computing a path γ_1 from $q_s \in \text{fsc}_1$ to a configuration $c_2 \in \text{fsc}_1 \cap \text{fsc}_2$. Local paths γ_i are then computed in fsc_i from $c_i \in \text{fsc}_{i-1} \cap \text{fsc}_i$ to $c_{i+1} \in \text{fsc}_i \cap \text{fsc}_{i+1}$ for every $i \in \{2 \ldots (n-1)\}$. Finally a local path is computed from c_n to q_t in fsc_n. Concatenating all the local paths results in the desired continuous path in $\mathcal{C}_{\text{free}}$. The methods presented in Chapter 2 may be of use when computing the local path or any other method depending on the problem at hand.

The rest of the chapter presents the required details to complete the MMS framework. We discuss desirable properties of manifolds in Section 3.1 and discuss the strategies that may be used to generate manifold samples in Section 3.2. The chapter concludes with a brief discussion comparing the MMS framework to the standard PRM algorithm.

3.1 Manifolds Families and Primitives

Choosing the specific set of manifold families depends on the concrete problem at hand, as detailed in the next chapter. However, it seems desirable to retain some general properties: (i) Each manifold should be sufficiently *simple* such that it is possible to decompose it into free and forbidden cells in a computationally efficient manner. (ii) The manifold families should *cover* the configuration space in a dense fashion. (iii) The manifold families allow for local connections between close-by configurations, a property which we term *spanning*. We anticipate that these simple and intuitive properties (perhaps subject to some further tuning) may lead to a proof of probabilistic completeness of the framework. This conjecture is motivated by the work presented in Chapter 5 which details a proof of probabilistic completeness of the framework for the specific instance that we shall discuss in Chapter 4.

Algorithm 4 construct_connectivity_graph

1: $V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $X \leftarrow \emptyset$
2: **repeat**
3: $\quad \psi \leftarrow$ generate_constraint(V,E,X)
4: $\quad X \leftarrow X \cup \{\psi\}$
5: $\quad \text{FSC}_{m_\psi} \leftarrow P_\Psi(m_\psi)$
6: $\quad V \leftarrow V \cup \{\text{fsc} |\ \text{fsc} \in \text{FSC}_{m_\psi}\}$
7: $\quad E \leftarrow E \cup \{(\text{fsc}_1, \text{fsc}_2) \mid \text{fsc}_1 \in V, \text{fsc}_2 \in \text{FSC}_{m_\psi}, \text{fsc}_1 \cap \text{fsc}_2 \neq \emptyset \ \&\ \text{fsc}_1 \neq \text{fsc}_2\}$
8: **until** stopping_condition
9: **return** $G(V, E)$

3.2 Constraint Generation and Applying Primitives

Regardless of the specific choice of manifold families, a naïve way to generate constraints that induce manifolds is sampling a constraint uniformally from all possible constraints. As primitives may be computationally complex and should thus be applied sparingly, we suggest a general exploration/connection scheme and additional optimization heuristics to be used in concrete implementations of the proposed general scheme. We describe strategies in general terms, providing conceptual guidelines for concrete implementations, as demonstrated in Chapter 4.

3.2.1 Exploration and Connection Phases

We propose that constraints should be generated in two phases: *exploration* and *connection*. In the exploration phase constraints are generated such that primitives will produce FSCs that introduce new connected components in $\mathcal{C}_{\text{free}}$. The aim of the exploration phase is to increase the coverage of the free configuration space as efficiently as possible. In contrast, in the connection phase constraints are generated such that primitives will produce FSCs that connect existing connected components in $\mathcal{G}$. Once a constraint is generated, $\mathcal{G}$ is updated as described above. Finally, we note that we can alternate between exploration and connection, namely we can decide to further explore after some connection work has been performed.

3.2.2 Region of Interest - RoI

Decomposing an entire manifold m_ψ by a primitive P_Ψ may be unnecessary. Patches of m_ψ may intersect $\mathcal{C}_{\text{free}}$ in highly explored parts or connect already well-connected parts of $\mathcal{G}$ while others may intersect $\mathcal{C}_{\text{free}}$ in sparsely explored areas or less well connected parts of $\mathcal{G}$. Identifying the regions where the manifold is of good use (depending on the phase) and constructing m_ψ only in those regions increases the effectiveness of P_Ψ while desirably biasing the samples. We refer to a manifold patch that is relevant in a specific phase as the *Region of Interest* - RoI of the manifold.

3.2.3 Constraint Filtering

Let $\psi \in \Psi$ be a constraint such that applying P_Ψ to ψ yields the set of FSCs on m_ψ. If we are in the connection phase, inserting the associated nodes into $\mathcal{G}$ and intersecting them with the existing FSCs should connect existing connected components of $\mathcal{G}$. Otherwise, the primitive's contribution is poor. We suggest applying a filtering predicate immediately after generating a constraint ψ to check if $P_\Psi(\psi)$ may connect existing connected components of $\mathcal{G}$. If not, the primitive should not be constructed and ψ should be discarded.

Algorithm 5, which is the implementation of line 3 in Algorithm 4 describes generating constraints and filtering. The algorithm is simply a call to two different constraint generation algorithms (lines 2,5) depending on the current phase (line 1). One should note that the constraint generating algorithm for the exploration phase takes into account the set of manifolds already sampled (line 2). On the other hand, the constraint generating algorithm for the connection phase takes into account the current connectivity graph (line 5).

Algorithm 5 generate_constraint V, E, X

```
1: if exploration_phase() then
2:    ψ ← generate_exploration_constraint(X)
3: else
4:    repeat
5:       ψ ← generate_connection_constraint(V, E)
6:    until constraint_filtering (ψ ≠ filter_out)
7: end if
8: return ψ
```

3.3 Comparison With PRM - Discussion

The fundamental difference between the PRM algorithm's approach and ours is the sample types used. While the PRM uses points as samples, our framework suggests using manifolds as samples. This gives rise to the question "Which manifolds should be used?" which is an interesting question and further research should be applied to it. A sampled point in the PRM algorithm is either in $\mathcal{C}_{\text{free}}$ or in $\mathcal{C}_{\text{forb}}$. It is safe to assume that this can be checked in an efficient manner using a collision detector. In our framework, on the other hand, the manifolds need to be decomposed into cells in $\mathcal{C}_{\text{free}}$ or $\mathcal{C}_{\text{forb}}$ which can be a costly and difficult-to-implement operation. As opposed to probing a single point in the configuration space, as is done in the PRM algorithm, using manifolds as samples captures the connectivity of entire sub-portions of $\mathcal{C}$ at once.

Once the collision detector reports that a PRM sample is in $\mathcal{C}_{\text{free}}$ it is added to the roadmap. Thus the points play a dual role of samples and nodes in the roadmap. In our framework on the other hand, the nodes of the roadmap are cells in the decomposed manifolds. Valid local paths in the PRM algorithm induce edges in the roadmap while in our framework intersecting FSCs induce edges. Finally, the data structure constructed by the PRM algorithm is a roadmap graph while the MMS algorithm constructs a *geometric intersection graph*[1] of FSCs. The differences mentioned above between the suggested scheme and the PRM algorithm are summarized in Table 3.1.

We point to another subtle difference between PRM and MMS, which makes it harder to prove probabilistic completeness for MMS. Although a similar property holds for most variants of PRM, we describe it here for the variant that we have outlined in Section 2.2.3. For a milestone q let r be the distance between q and its kth nearest neighbor. Then if we know that the clearance of the milestone q is at least ρ for $\rho \leq r$ then we are guaranteed that q is connected in the roadmap to any other milestone in the ball of radius ρ around q. A similar property does not hold for MMS. It is much more difficult to guarantee such a property for MMS since there is an infinity of free points in the FSCs. It is not impossible to guarantee such a property for MMS as well, but it will make the algorithm considerably more complicated.

	PRM	**MMS**
Sample type	Point	Manifold
Decomposition	Collision detector	Analytic primitive
Node type	Point	FSC
Node connection method	Local planner	Intersecting FSCs
Data structure	Roadmap graph	Geometric intersection graph of FSCs

Table 3.1: Comparison between our scheme and PRM algorithm

[1] A geometric graph has vertices for geometric objects, and edges connect pairs of vertices whose objects fulfil some property; here the property is intersection.

4

The Case of Rigid Polygonal Motion

We demonstrate the scheme suggested in Chapter 3 by considering a (not necessarily convex) polygonal robot R translating and rotating in the plane amidst polygonal obstacles. A configuration of R describes the position of the (arbitrarily chosen) reference point of R and the orientation of R. As we consider full rotations, the configuration space $\mathcal{C}$ is the three-dimensional space $\mathbb{R}^2 \times S^1$.

Section 4.1 discusses the manifolds used for this instance of MMS, Section 4.2 discusses the means to generate the constraints by suggesting several heuristics that follow the exploration/connection paradigm presented in Chapter 3. Section 4.3 elaborates on the local planning applied within the cells in the query stage. We conclude this chapter with experimental results of an implementation of the presented algorithm. We remark that the engineering work invested in optimizing MMS yielded an algorithm comparable and even surpassing a motion planner that is in prevalent use. For example, in a tight passage scenario we demonstrate a 27-fold speedup in running time.

4.1 Manifold Families and Manifold Decomposition

As described in Chapter 3, we consider manifolds defined by *constraints* and construct and decompose them using *primitives*. We suggest the following constraints restricting the motions of the robot R, and describe their associated primitives:

- The **Angle constraint** fixes the orientation of R while it is still free to translate anywhere within the workspace.
- The **Segment constraint** restricts the position of the reference point to a segment in the workspace while R is free to rotate.
- The **Point constraint** fixes the location of R to a point p while it is still free to rotate around p.

Angle-primitive The Angle-primitive, for a constraining angle θ (denoted $P_\Theta(\theta)$) is constructed by computing the Minkowski sum of $-R_\theta$ with the obstacles[1]. We refer to such a decomposed manifold as a horizontal layer or simply a *layer*. This induces a two-dimensional horizontal plane where the cells are polygons as demonstrated in Figure 4.1b.

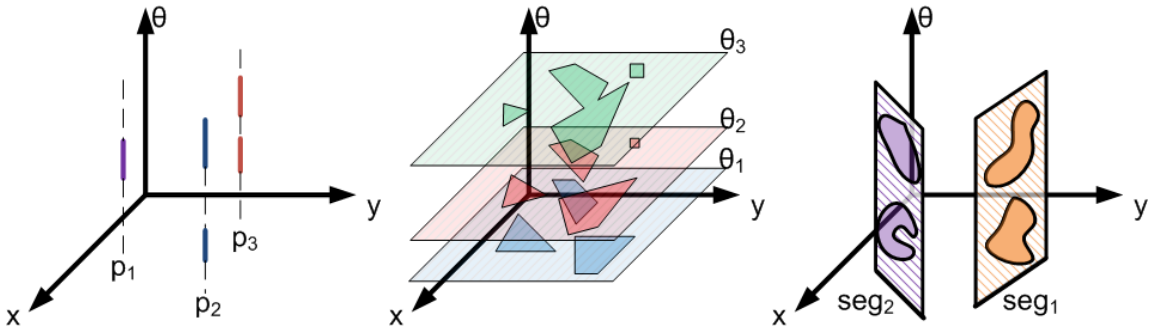
Segment-primitive Limiting the possible positions of the robot's reference point r to a given segment s, results in a two-dimensional configuration space. Each vertex (or edge) of the robot in combination with

[1] We use $-R_\theta$ to denote R rotated by θ and reflected about the origin.

each edge (or vertex) of an obstacle give rise to a critical curve in this configuration space. Namely the set of all configurations that put the two features in contact, and thus mark a potential transition between $\mathcal{C}_{\text{forb}}$ and $\mathcal{C}_{\text{free}}$. Our analysis (Appendix A) shows that these critical curves can be expressed by rational functions. Rational functions are functions that can be written as a quotient of two polynomials. In our case, both the numerator and denominator of the functions are of degree two with rational coefficients. These functions have favorable properties when compared with arbitrary algebraic curves, and consequently can be handled more efficiently. Thus, the Segment-primitive is the construction of these critical curves and their arrangement. This induces a two-dimensional vertical slab where the cells are defined by the subdivision induced by rational curves as demonstrated in Figure 4.1c.

Point-primitive The Point-primitive is a degenerate case of the Segment-primitive where the two-dimensional arrangement reduces to a vertical line and the cells are intervals on that line as demonstrated in Figure 4.1a.

We defer the implementation details on how these predicates are decomposed to Appendix A. For now, notice that the Segment-primitive is computationally far more time consuming than the Angle-primitive, since it involves arrangements of curves of higher algebraic degree.



(a) Point primitive (b) Angle primitive (c) Segment primitive

Figure 4.1: Manifolds and their decomposition

4.2 Constraint Generation and Applying Primitives

We use manifolds constructed by the Angle-primitive for the exploration phase and manifolds constructed by the Segment-primitive for the connection phase. Since the Segment-primitive is far more costly than the Angle-primitive, we focused our efforts on optimizing the former.

Generating a constraint in the exploration phase (line 2 in Algorithm 5, Chapter 3) is generating an angle $\theta \in [-\pi \dots \pi)$. Several options may be used to generate θ. Our implementation generates angles at constant intervals according to a predefined parameter. Other options may include sampling angles using a uniform distribution, or taking into account the topological criticalities in the θ-axis of the three-dimensional arrangement of the configuration space. It should be noted that generating angles at constant intervals leads to an algorithm that is resolution complete[2] and not probabilistically complete as shown in the next chapter.

Generating a constraint in the connection phase (line 5 in Algorithm 5) is equivalent to generating a segment in the workspace $\mathbb{R}^2$. As suggested in Section 3.2 we may consider the Segment-primitive in a subset of the range of angles. This results in a somewhat "weaker" yet more efficient primitive than considering the whole range. If the connectivity of a restricted region of the configuration space is desired, then using this optimization may suffice while considerably speeding up the algorithm.

[2] An algorithm is said to be resolution complete if it is guaranteed to find a path if the resolution of a parameter used by the algorithm is sufficiently fine. If the algorithm uses a grid for example, the resolution of the underlying grid is the parameter considered.

Generating segments: Let *fsc* be a random FSC. Consecutive layers (manifolds of the Angle constraint) have a similar structure unless topological criticalities occur in $\mathcal{C}$ between the layers. Once such a criticality occurs, an FSC either appears and grows, or shrinks and disappears. We thus suggest a heuristic for generating a segment in the workspace for the Segment-primitive using the size of *fsc* as a parameter where we refer to small and large cells according to pre-defined threshold constants. The RoI used will be proportional to the size of the FSC.

The segment generated will be chosen with one of the following procedures used in Algorithm 6 which is an implementation of line 5 in Algorithm 5:

- **Random procedure**: Return a random segment from the workspace.
- **Large cell procedure**: Return a random segment from the projection of *fsc* onto the xy-plane.
- **Small cell procedure**: Choose an adjacent layer to *fsc* and project its FSCs onto the xy-plane. Project *fsc* onto the xy-plane as well and intersect the two projections. Return a segment connecting a random point from the projection of *fsc* and a random point in the intersection.

Algorithm 6 generate_segment_constraint (V,E)

```
1: if random_num ([0, 1]) ≥ random_threshold then
2:    return random_segment_procedure()
3: else
4:    fsc ← random_fsc(V)
5:    α ← [size(fsc) - small_cell_size] / [large_cell_size - small_cell_size]
6:    if random_num([0, 1]) ≤ α then
7:       return small_cell_procedure(fsc,V)
8:    else
9:       return large_cell_procedure(fsc,V)
10:   end if
11: end if
```

Constraint Filtering: As suggested in Section 3.2, we avoid computing unnecessary primitives. All FSCs that will intersect a "candidate" constraint s, namely all FSCs of layers in its RoI, are tested. If they are all in the same connected component in $\mathcal{G}$, s can be discarded as demonstrated in Algorithm 7 which is the implementation of line 6 in Algorithm 5.

Algorithm 7 filter_segment (s,RoI,V,E)

```
1: cc_ids ← ∅                    //set of connected component ids
2: for all v ∈ V do
3:    fsc ← free_space_cell(v)
4:    if constraining_angle(fsc)∈ RoI then
5:       cc_ids ← cc_ids∪ connected_component_id(v)
6:    end if
7: end for
8: if |cc_ids| ≤ 1 then
9:    return filter_out
10: end if
```

4.3 Path planning

A path is planned as described in Chapter 3, namely for a query $[(x_s, y_s, \theta_s), (x_t, y_t, \theta_t)]$, a layer at angle θ_s and a layer at angle θ_t are added to $\mathcal{G}$. A list of intersecting FSCs is extracted from $\mathcal{G}$ by traversing the

graph using a breadth-first search (BFS) algorithm. Within each FSC a local planner is used to construct a path in the configuration space. As the cells of the two primitives differ, the local planners used are different.

An FSC of an Angle-primitive is a polygon. The local planner used within the polygon is the shortest-path planner discussed in Chapter 1: The visibility graph is computed within the polygon inducing a shortest-path roadmap.

An FSC of a Segment-primitive is an arrangement cell of the subdivision of a slab induced by the critical curves. In order to discuss the local planner within the arrangement cell, we briefly describe the parametrization defining the arrangement; full details are given in Appendix A.

A robot R is a simple polygon with vertices $\{v_1, \ldots, v_n\}$ where $v_i = (x_i, y_i)^T$ and edges $\{(v_1, v_2), \ldots (v_n, v_1)\}$. We assume that the reference point of R is located at the origin. The position of R in the workspace is defined by a configuration $q = (r_q, \theta_q)$ where $r_q = (x_q, y_q)^T$. Thus, q maps the position of a vertex v_i as follows:

$$v_i(q) = M(\theta_q)v_i + r_q,$$

where $M(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ is the rotation matrix.

Given a segment $seg = [s, t]$ where $s = (x_s, y_s)^T$ and $t = (x_t, y_t)^T$ we define the parametrization $(\alpha, \tau) \in [0, 1] \times \mathbb{R}$ as follows:

$$r_q = (1 - \alpha)s + \alpha t,$$

$$\theta_q = 2 \arctan \tau.$$

The parametrization fixes the robot's reference point to the supporting line of seg. The parametrized vertex is represented as:

$$v_i(\alpha, \tau) = M(\tau)v_i + (1 - \alpha)s + \alpha t,$$

where $M(\tau) = \frac{1}{1+\tau^2} \begin{bmatrix} 1 - \tau^2 & -2\tau \\ 2\tau & 1 - \tau^2 \end{bmatrix}$.

Hence the parametrized configuration space is not planar but cylindrical: $\tau = \pm\infty$ represents the same point for a fixed α due to the identification of θ at $\pm\pi$. Hence cells in the planar arrangement at $\pm\infty$ with the same α values are considered as the same cell in the configuration space. The FSC is one such cell, it is decomposed into trapezoidal cells (see Chapter 1), and a graph data structure holds the decomposed cells as nodes and adjacent cells as edges. The local planner traverses this graph thus computing a list of adjacent trapezoidal neighboring cells.

As neighboring cells always share a vertical wall between them, planning a path between two cells is always planning a path between two vertical walls of a cell. The ceiling and floor of the cell are rational functions, therefor a straight line may intersect them. The local planner attempts to compute a straight line between the two vertical walls of the cell. If this fails, it constructs a midpoint of the cell and computes a path from each vertical wall to the midpoint in a recursive fashion. Figure 4.2 demonstrates this process.

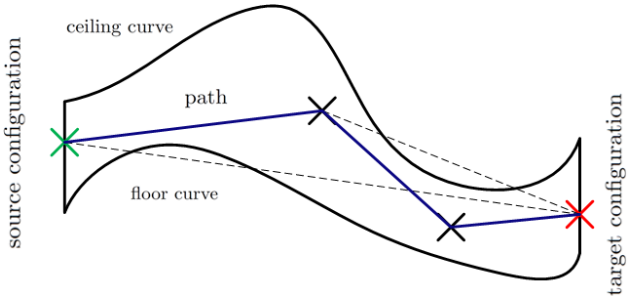
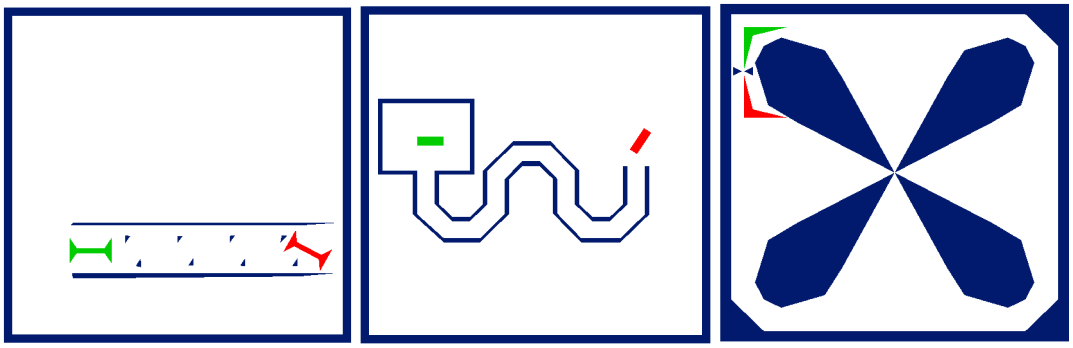


Figure 4.2: A local path planned within one trapezoidal cell of an arrangement of rational functions. The path is refined twice by the local planner.



(a) Tunnel scenario (b) Snake scenario (c) Flower scenario

Figure 4.3: Experimental scenarios

4.4 Experimental Results

The algorithm discussed in this chapter is implemented in C++. It is based on CGAL's arrangement package, which is used for the geometric primitives, and the BOOST graph library [62], which is used to represent the connectivity graph $\mathcal{G}$.

The implementation of the Angle-primitive is an application of CGAL's Minkowski sums package [65, C.24]. We remark that we ensure (using the method of Canny et al. [12]) that the angle θ is chosen such that $\sin\theta$ and $\cos\theta$ are rational. This allows for an exact rotation of the robot and an exact computation of the Minkowski sum. The implementation of the Segment-primitive attempts to represent the configuration space explicitly by construction of an arrangement of critical curves (see Chapter 2). As our analysis shows (Appendix A), the critical curves are rational functions. We provide further details on the construction of the arrangement in the second part of the thesis.

We demonstrate the performance of our planner using three different scenarios. All scenarios consist of a workspace, a robot with obstacles and one query (source and target configurations). Figure 4.3 illustrates the scenarios where the obstacles are drawn in blue and the source and target configurations are drawn in green and red, respectively. All reported tests were measured on a Dell 1440 with one 2.4GHz P8600 Intel Core 2 Duo CPU processor and 3GB of memory running with a Windows 7 32-bit OS. Preprocessing times presented (in seconds) are times that yielded at least 80% (minimum of 5 runs) success rate in solving queries.

4.4.1 Algorithm Parameters and Variants

Our planner has two parameters: the number n_θ of layers to be generated and the number n_s of segment constraints to be generated (before filtering). We chose the following values for these parameters: $n_\theta \in \{10, 20, 40, 80\}$ and $n_s \in \{2^i | i \in \mathbb{N}, i \leq 14\}$. For a set of parameters (n_θ, n_s) we report the preprocessing time t and whether a path was found (marked ✓) or not found (marked ×) once the query was issued. The results for the flower scenario are reported in Table 4.1. We show that a considerable increase in parameters has only a limited effect on the preprocessing time.

In order to test the effectiveness of our optimizations, we ran the planner with and without any heuristic for choosing segments and with and without segment filtering. We also added a test with all optimizations using the old traits class. The parameters used for each case are the ones that yielded the fastest preprocessing time for each case. The results for the flower scenario can be viewed in Table 4.2. We remark that the engineering work invested in optimizing MMS yielded an algorithm comparable and even surpassing a motion planner that is in prevalent use as shown next.

		n_θ			
		10	20	40	80
n_s	256	(6,×)	(11,×)	(12,×)	(16,×)
	512	(7,×)	(13,×)	(14,×)	(25,×)
	1024	(16,×)	(20,✓)	(23,✓)	(35,✓)
	2048	(30,×)	(35,✓)	(38,✓)	(51,✓)
	4096	(46,✓)	(53,✓)	(60,✓)	(82,✓)

Table 4.1: Parameter sensitivity

Traits	Segment Generation	Filtering	n_θ	n_s	t
New	random	not used	20	8192	1418
		used	20	8192	112
	heuristic	not used	40	512	103
		used	20	1024	20
Old	heuristic	used	20	1024	138

Table 4.2: Optimization results

4.4.2 Comparison With PRM

We used an implementation of the PRM algorithm as provided by the OOPSMP package [53]. For fair comparison, we did not use cycles in the roadmap as cycles increase the preprocessing time significantly. We manually optimized the parameters of each planner over a concrete set. As with previous tests, the parameters for MMS are n_θ and n_s. The PRM variant implemented by the OOPSMP package iterates between sampling a batch of configurations and attempting to connect the milestones to their neighbors. The parameters used for this implementation are the number of neighbors (denoted k) to which each milestone should be connected and the percentage of time used to sample new milestones (denoted % st in Table 4.3).

Furthermore, we ran the flower scenario several times, progressively increasing the robot size. This caused a "tightening" of the passages containing the desired path. Figure 4.4 demonstrates the preprocessing time as a function of the tightness of the problem for both planners. A tightness of zero denotes the base scenario (Figure 4.3c) while a tightness of one denotes the tightest problem solved in these experiments.

The results show a speedup for all scenarios when compared to the PRM implementation. Moreover, our algorithm has little sensitivity to the tightness of the problem as opposed to the PRM algorithm. In the tightest experiment, MMS runs 27 times faster than the PRM implementation.

The results shown indicate that the suggested framework, along with algorithmic engineering yields a carefully implemented algorithm for the case of a polygonal robot translating and rotating in the plane amidst polygonal obstacles. The optimizations used for this instance, namely a new traits class for rational curves, powerful segment generation techniques and a filtering scheme may be reused for other instances and thus, we believe, are interesting by themselves. We have shown that, put together, these optimizations result in a speed-up in running time that makes this algorithm comparable and even faster than the running time of a carefully implemented PRM algorithm. While this in itself is promising, what gives further support to the suggested framework is the behavior shown for narrow passages. Sampling entire manifolds and not isolated points increases the probability of sampling such a passage. Moreover, as with other computational-geometric algorithms, the number of curves and not their mutual distance governs the complexity of the algorithm (in the Real RAM model at least), thus yielding an algorithm with lower sensitivity to tight passages.

Scenario	MMS			PRM			Speedup
	n_θ	n_s	t	k	% st	t	
Tunnel	20	512	100	20	0.0125	180	1.8
Snake	40	256	22	20	0.025	140	6.3
Flower	20	1024	20	24	0.0125	40	2

Table 4.3: Comparison with PRM

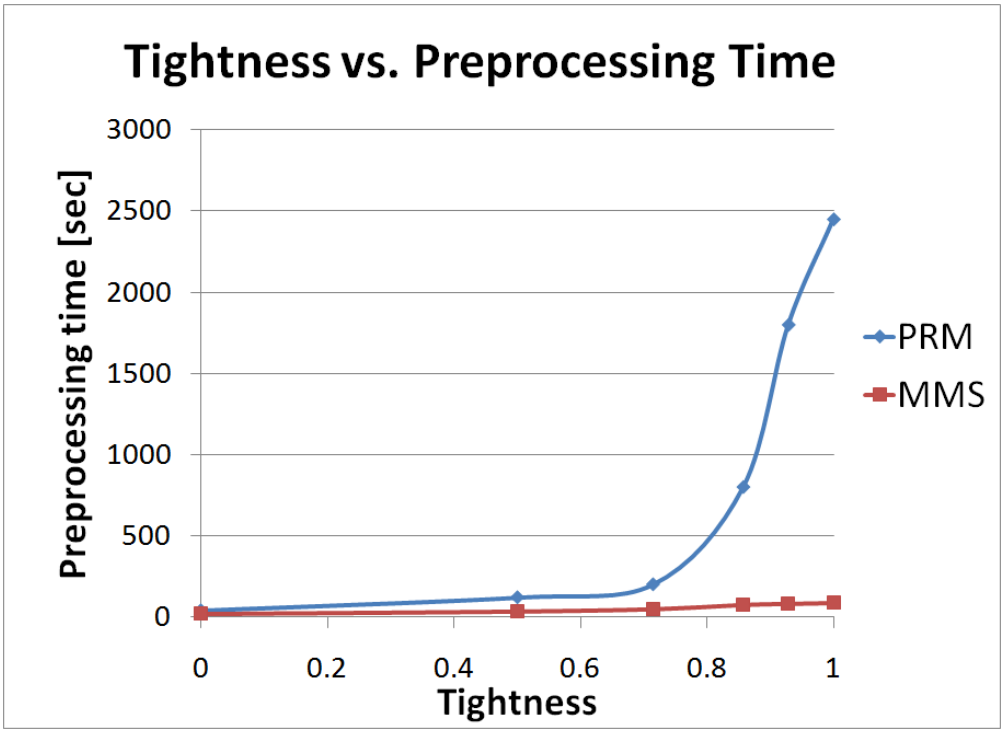


Figure 4.4: Tightness results

5

Probabilistic Completeness of MMS for a Polygonal Robot

We analyze the application of MMS to the case of a polygonal robot rotating and translating in the plane. We show that for any collision-free path $\gamma_{a,b}$ of clearance $\rho > 0$ between two configurations a and b, it is possible to construct a path from a to b with distance at most ρ from $\gamma_{a,b}$ on the union of manifolds used. Moreover the probability of finding such a path by the MMS algorithm increases exponentially with the number of samples.

Setting: A configuration is defined by a three-dimensional point $p = (x_p, y_p, \theta_p)$ where x_p, y_p is the planar pose of the robot R in the workspace and θ_p is the amount of counterclockwise rotation of R relative to its original placement. The configuration space is $\mathbb{R}^2 \times S^1$. We assume that the workspace is bounded by a box and by scaling both the translational and rotational parts, we consider the configuration space to be $\mathcal{C} = [0,1]^2 \times [0,1)$. As $\mathbb{R}^2 \times S^1$ induces a periodic workspace and not a cube, when we consider angle differences, it is done modulo one.

MMS Algorithm: We consider two families (types) of manifolds, *horizontal layers* and *vertical lines*[1]. The MMS algorithm samples n_p constraints for the lines, these constraints are points sampled uniformally from the xy-plane $[0,1]^2$. The MMS algorithm samples n_θ constraints for the layers, these constraints are angles sampled uniformally from the θ-axis $[0,1)$. For a query (a,b) where $a = (x_a, y_a, \theta_a)$ and $b = (x_b, y_b, \theta_b)$, we assume that the MMS algorithm adds the horizontal layers at angles θ_a, θ_b.

5.1 Notation

We introduce the following notation and definitions used in the proof:

- $\mathcal{C}_{\text{free}}$ is a subset of the three-dimensional space $\mathcal{C}$ consisting of all collision-free configurations.
- $dist(p,q)$ is the distance between two points $p = (p_x, p_y, p_\theta)$, $q = (q_x, q_y, q_\theta)$ in $[0,1]^3$ such that $dist(p,q) = min\{\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_\theta - q_\theta)^2}, \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (2 - p_\theta - q_\theta)^2}\}$. This is the periodic Eucledian distance on $\mathcal{C}$ where $\theta = 1$ and $\theta = 0$ are identical.
- $B_r(p)$ three-dimensional ball of radius r centered at the three-dimensional point p.

[1] Considering vertical lines and not vertical slabs as discussed in Chapter 4 is done for simplifying the proof. This is obviously a special case of the same family of manifolds thus the presented proof applies for the case of vertical manifolds.

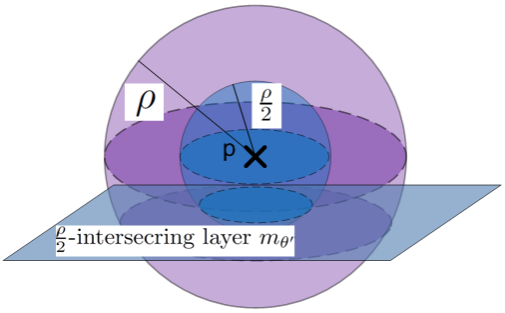


Figure 5.1: A $\frac{\rho}{2}$-intersecting layer of a point p

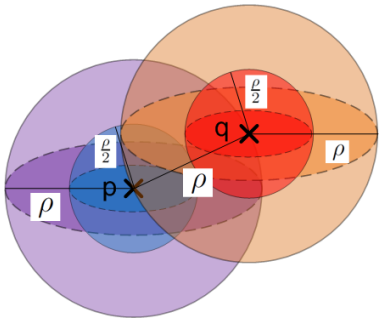


Figure 5.2: Points of distance ρ apart inscribed by balls of radii ρ and $\frac{\rho}{2}$

- $D_r(\tilde{p})$ two-dimensional disk of radius r centered at the two-dimensional point $\tilde{p}$ in the xy-plane.
- $\Gamma = \{\gamma_{p,q} \mid \gamma_{p,q}$ is a continuous mapping $[0,1] \to \mathcal{C}$ where $\gamma(0) = p$ and $\gamma(1) = q\}$.
- $\text{Im}(\gamma) = \{c \in \mathcal{C} | \ \exists \alpha \in [0,1], \gamma(\alpha) = c\}$ is the image of a path γ.
- $m_{\theta'} = \{(x, y, \theta') \in \mathcal{C} \ | x, y \in [0,1]\}$ is a horizontal layer at a fixed angle $\theta' \in [0, 1)$.
- $m_{(x',y')} = \{(x', y', \theta) \in \mathcal{C} \ | \theta \in [0,1)\}$ is a vertical line at a fixed point $(x', y') \in [0,1]^2$.
- $M_\Theta = \{m_\theta | \theta \in [0,1)\}$ is the family of horizontal layers.
- $M_P = \{m_{x,y} | (x,y) \in [0,1]^2\}$ is the family of vertical lines.
- **$\frac{\rho}{2}$-intersecting** For a point p in $\mathcal{C}$ and a bound ρ, a horizontal layer $m_{\theta'}$ is $\frac{\rho}{2}$-intersecting if $m_{\theta'} \cap B_{\frac{\rho}{2}}(p) \neq \emptyset$. See Figure 5.1 for an illustration.
- **ρ-connecting** For a point p in $\mathcal{C}$ a vertical line m_v is ρ-connecting if the projection of p onto the xy-plane is of distance not greater than $r = \frac{\sqrt{3}}{2}\rho$ from v.
- $Pr[(a,b)\text{SUCCESS}]$ Probability of MMS algorithm to return a path from a to b when such a path exists.
- $Pr[(a,b)\text{FAILURE}]$ Probability of MMS algorithm not to return a path from a to b when such a path exists.

5.2 Probabilistic Completeness Proof

Motivation and outline: In order to construct a path from a source configuration to a target configuration, we first show that for any two close-by configurations, it is possible to construct a path on the manifolds used by MMS such that the path starts from the vicinity of the source configuration and terminates at the vicinity of the target configuration. Two close-by configurations will be formally described as two points with distance no greater than ρ, while the vicinity of a point p will be formally described as $B_{\frac{\rho}{2}}(p)$, the points of distance smaller or equal than $\frac{\rho}{2}$ from p. Manifolds that may induce a path between two close-by configurations are a ($\frac{\rho}{2}$-intersecting) horizontal layer intersecting the vicinity of the first point, a ($\frac{\rho}{2}$-intersecting) horizontal layer intersecting the vicinity of the second point and a (ρ-connecting) vertical line intersecting both layers. Moreover, the properties of the ρ-connecting vertical line will ensure that the path induced by these manifolds is within distance of at most ρ of either points. These properties will be shown in Lemma 5.2.1, Lemma 5.2.2 and Lemma 5.2.3. Using these lemmas, Theorem 5.2.4 will show the probabilistic completeness of the MMS algorithm in this case.

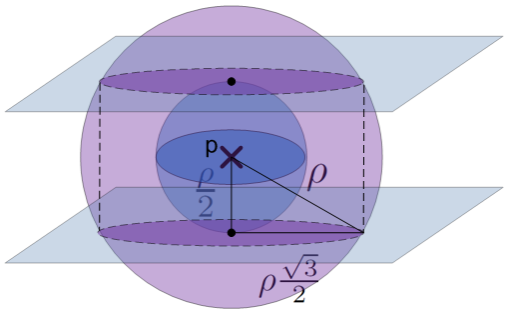


Figure 5.3: Maximal area of intersection between a $\frac{\rho}{2}$-intersecting layer and a vertical line

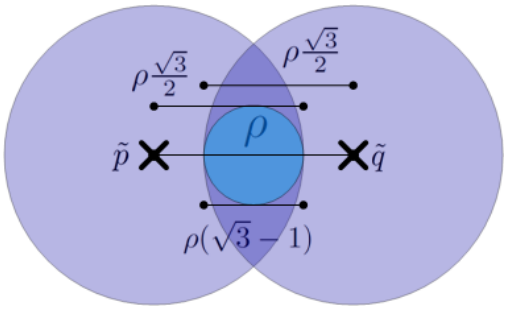


Figure 5.4: Projection $\tilde{p}, \tilde{q}$ of two points p and q of distance ρ. Center disk is $D_{\frac{\sqrt{3}-1}{2}}(\frac{\tilde{p}+\tilde{q}}{2})$

In the following we consider points p, q such that $dist(p,q) \leq \rho$. For each point we consider the balls centered at the points with radii ρ and $\frac{\rho}{2}$. See Figure 5.2 for an illustration.

Lemma 5.2.1. *Let p be a point in $[0,1]^3$, $m_{\theta'_p} \in M_\Theta$ be a $\frac{\rho}{2}$-intersecting layer of p and $m_v \in M_P$ be a ρ-connecting vertical line of p. The intersection of the two manifolds is within distance ρ of p. Namely, $m_{\theta'_p} \cap m_v \in B_\rho(p)$*

Proof. $m_{\theta'_p}$ is $\frac{\rho}{2}$-intersecting with $p = (x_p, y_p, \theta_p)$ thus the distance between p and $m_{\theta'_p}$ along the θ-axis is $|\theta'_p - \theta_p| \leq \frac{\rho}{2}$. m_v is ρ-connecting with p thus the projected distance between p and the two-dimensional point v, the constraint defining the line m_v, on the xy-plane is no more than $\frac{\sqrt{3}}{2}\rho$. By the Pythagorean Theorem, the distance of the intersection point $p'_v = m_{\theta'_p} \cap m_v$ from p is bounded by $\sqrt{(\frac{\rho}{2})^2 + (\frac{\sqrt{3}}{2}\rho)^2} = \rho$. For an illustration see Figure 5.3. □

Lemma 5.2.2. *Let p and q be two points in $\mathcal{C}$ such that $dist(p,q) \leq \rho$ and let $\tilde{p}$ and $\tilde{q}$ denote their projection onto the xy-plane. Let $m_v \in M_P$ be a vertical line induced by a two-dimensional point v on the xy-plane.*

If $v \in D_{\frac{\sqrt{3}-1}{2}}(\frac{\tilde{p}+\tilde{q}}{2})$ then m_v is ρ-connecting for both p and q.

Proof. Let v be a point on the xy-plane inside the disk of radius $\frac{\sqrt{3}}{2}$ located midway between $\tilde{p}$ and $\tilde{q}$ (Figure 5.4). By the triangle inequality

$$dist(v,\tilde{p}) \leq dist(v, \frac{\tilde{p}+\tilde{q}}{2}) + dist(\frac{\tilde{p}+\tilde{q}}{2}, \tilde{p}) \leq \frac{\sqrt{3}-1}{2}\rho + \frac{\rho}{2} = \frac{\sqrt{3}}{2}\rho.$$

Hence v is ρ-intersecting for p. Symmetrically, v is ρ-intersecting for q. □

Lemma 5.2.3. *Let p, q be two points such that $dist(p,q) \leq \rho$, $m_{\theta'_p}$, $m_{\theta'_q}$ be two $\frac{\rho}{2}$-intersecting layers of p and q respectively. Let m_v be a ρ-connecting vertical line of both p and q as defined in Lemma 5.2.2. Let p' be the projection of p onto $m_{\theta'_p}$, q' be the projection of q onto $m_{\theta'_q}$ and $\gamma_{p',q'}$ be the path between p' and q'. $Im(\gamma_{p',q'}) \subseteq (m_{\theta'_p} \cup m_v \cup m_{\theta'_q}) \cap (B_\rho(p) \cup B_\rho(q))$.*

Proof. Let $\gamma_{p',q'}$ be the concatenation of five paths γ_1, γ_2, γ_3, γ_4, γ_4 as illustrated in Figure 5.5 where:

- γ_1 is the segment connecting p' and the intersection p'_v of the two layers $m_{\theta'_p}$ and m_v,
- γ_2 is the segment connecting p'_v and the projection p_v of p onto the vertical line m_v,
- γ_3 is the segment connecting p_v and q_v, the projection of q onto the vertical line m_v,

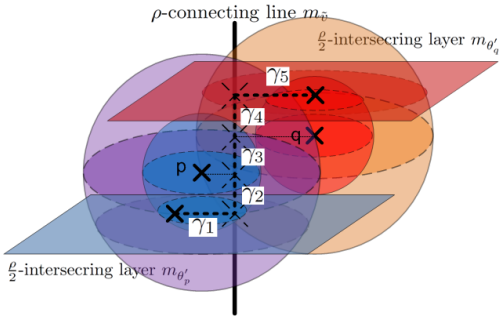


Figure 5.5: Illustration of path induced by $\frac{\rho}{2}$-intersecting layers of p and q and a ρ-connecting line of p, q.

- γ_4 is the path connecting q_v and q'_v, the intersection of the two layers m_v and $m_{\theta'_q}$,
- γ_5 is the segment connecting q'_v and q'.

We will show that each path is contained within $B_\rho(p) \cup B_\rho(q)$:
$\gamma_1 \subseteq B_\rho(p)$: $m_{\theta'_p}$ is $\frac{\rho}{2}$-intersecting thus $p' \in B_\rho(p)$. $p'_v \in B_\rho(p)$ by Lemma 5.2.1. By the convexity of $B_\rho(p)$, it holds that $\gamma_1 \subseteq B_\rho(p)$.
$\gamma_2 \subseteq B_\rho(p)$: $p'_v \in B_\rho(p)$ by Lemma 5.2.1. m_v is a ρ-connecting vertical line thus p_v, the projection of p onto m_v is of maximal distance $\frac{\sqrt{3}}{2}\rho$ from p, hence $p_v \in B_\rho(p)$. $B_\rho(p)$ is convex, thus $\gamma_2 \subseteq B_\rho(p)$.
$\gamma_3 \subseteq B_\rho(p) \cup B_\rho(q)$: Recall that $dist(p,q) \leq \rho$ thus $|\theta_p - \theta_q| \leq \rho$. A point $p_0 = (x_v, y_v, \theta_0) \in \gamma_3$ is either closer to p or closer to p, we assume w.l.o.g that it is closer to p. Thus $|\theta_0 - \theta_p| \leq \frac{\rho}{2}$. m_v is ρ-intersecting, thus the distance on the xy-plane between the projections of p_0 and p is at most $\frac{\sqrt{3}}{2}\rho$. As the segments (p, p_v) and (p_v, p_0) are perpendicular we apply the Pythagorean Theorem:

$$dist(p, p_0) = \sqrt{dist(p, p_v)^2 + dist(p_v, p_0)^2} \leq (\frac{\sqrt{3}}{2}\rho)^2 + (\frac{\rho}{2})^2 = \rho.$$

$\gamma_4 \subseteq B_\rho(q)$: Symmetric to γ_2.
$\gamma_5 \subseteq B_\rho(q)$: Symmetric to γ_1. □

Theorem 5.2.4. *Let $a, b \in \mathcal{C}_{\text{free}}$ such that there exists a collision-free path $\gamma_{a,b} \in \Gamma$ of length L and clearance ρ between a and b. Then the probability of the MMS algorithm to return a path between a and b after generating n_θ layers and n_p vertical lines is:*

$$Pr[(a,b)\text{SUCCESS}] = 1 - Pr[(a,b)\text{FAILURE}] \geq 1 - \left\lceil \frac{L}{\rho} \right\rceil \left[\left(1 - \pi r^2\right)^{n_p} + (1-\rho)^{n_\theta} \right]$$

where $r = \frac{\sqrt{3}-1}{2}\rho$.

Proof. Let $l = \left\lceil \frac{L}{\rho} \right\rceil$ and observe that there are l points on $\gamma_{a,b}$ $a = p_1, \ldots, p_l = b$ such that $B_\rho(p_i) \in \mathcal{C}_{\text{free}}$ and $dist(p_i, p_{i+1}) \leq \rho$. *MMS* adds the manifolds m_{θ_a} and m_{θ_b} to the connectivity graph. m_{θ_a} has an FSC intersecting $B_{\frac{\rho}{2}}(a)$ and m_{θ_b} has an FSC intersecting $B_{\frac{\rho}{2}}(b)$.

Let $M'_P \subseteq M_P$ be the set of n_p lines sampled at random by the MMS algorithm and $M'_\Theta \subseteq M_\Theta$ be the set of n_θ layers sampled at random by the MMS algorithm. If there is a subset of the sampled lines $\{m_{\tilde{p}_1} \ldots m_{\tilde{p}_{l-1}}\} \in M'_P$ and a subset of the sampled layers $\{m_{\theta'_2} \ldots m_{\theta'_{l-1}}\} \in M'_\Theta$ such that $(m_{\theta'_i}, m_{\tilde{p}_i}, m_{\theta'_{i+1}})$ follow the conditions of Lemma 5.2.3 for $i \in \{1 \ldots l-1\}$, then a path from $B_{\frac{\rho}{2}}(p_1)$ to $B_{\frac{\rho}{2}}(p_l)$ may be constructed. This is done by concatenating adjacent paths. Moreover, as $m_{\theta'_1} = m_{\theta_a}$ and $m_{\theta'_l} = m_{\theta_b}$, the path can start at a and terminate at b such that its distance from $\gamma_{a,b}$ is at most ρ hence it is in $\mathcal{C}_{\text{free}}$.

Let $I_1 \ldots I_l$ be a set of indicator variables such that each I_i witnesses the event that there is a ρ-connecting vertical line $m_{\tilde{p}_i}$ of p_i and p_{i+1}. Let $J_2 \ldots J_l$ be a set of indicator variables such that each J_i witnesses the

event that there is a $\frac{\rho}{2}$-intersecting layer of p_i. It follows that MMS succeeds in answering the query (a, b) if $I_i = 1$ for all $1 \leq i \leq l-1$ and $J_j = 1$ for all $2 \leq j \leq l-1$. Therefore,

$$Pr[(a,b)\text{FAILURE}] \leq Pr\left(\vee_{i=1}^{l-1}(I_i = 0)\ \vee_{i=2}^{l-1}(J_j = 0)\right) \leq \sum_{i=1}^{l-1} Pr[I_i = 0] + \sum_{j=2}^{l-1} Pr[J_j = 0].$$

The events $I_i = 0$ are independent since the vertical samples are independent. The events $J_j = 0$ are independent since the layer samples are independent. $Pr[I_i = 1] = \pi r^2$ for $r = \frac{\sqrt{3}-1}{2}\rho$. Thus the probability that none of the n_p uniform independent point samples falls in $D_r(\frac{\tilde{p}_{i+1}+\tilde{p}_i}{2})$ is $Pr[I_i = 0] = \left(1 - \pi r^2\right)^{n_p}$ for $r = \frac{\sqrt{3}-1}{2}\rho$. $Pr[J_j = 1] = \rho$. Thus the probability that none of the n_θ uniform independent angle samples falls in $[p_j - \frac{\rho}{2}, p_j + \frac{\rho}{2}]$ is $Pr[J_j = 0] = (1-\rho)^{n_\theta}$. Since the sampling is uniform and independent:

$$Pr[(a,b)\text{FAILURE}] \leq \left\lceil \frac{L}{\rho} - 1 \right\rceil \left(1 - \pi r^2\right)^{n_p} + \left\lceil \frac{L}{\rho} - 2 \right\rceil (1-\rho)^{n_\theta} \leq \left\lceil \frac{L}{\rho} \right\rceil \left[\left(1 - \pi r^2\right)^{n_p} + (1-\rho)^{n_\theta}\right].$$

□

It follows that as n_p and n_θ tend to ∞, the probability of failing to find a path under the conditions stated in Theorem 5.2.4 tend to zero.

Part II

New Generic Arrangement Traits For Rational Functions

In the first part of the thesis, we suggested a general algorithmic framework and presented a specific implementation of the motion planner for the case of a polygonal robot translating and rotating in the plane. As part of the implementation, we construct a set of critical curves to decompose manifolds representing the motion of the robot free to rotate while translating along a fixed line segment. The curves, which are rational functions, are passed on to the CGAL arrangement package in order to decompose the space into free and forbidden cells; thus the implementation requires the arrangement package to handle such curves. This part of the thesis presents a C++ package that contributes a new *traits class* to CGAL which was developed to facilitate the implementation. We include experimental results and comparisons with similar classes which demonstrate its efficiency and note that it is integrated into CGAL 3.9.

6

Applied Computational Geometry Background

This chapter provides background material and definitions required for the second part of the thesis. Section 6.1 provides an introduction to CGAL and its `Arrangement_2` package while Section 6.2 provides general mathematical background on rational functions, their properties, and on algebraic numbers.

6.1 Exact Geometric Computation

Most computational-geometry algorithms assume the so-called *real* RAM model that allows for infinite precision arithmetic operations [54], thus discarding any errors that could arise from rounding errors. Numeric inaccuracies in the context of geometric algorithms may lead to inconsistent results in topological predicates [38]. In addition, the input is assumed to be in *general position*, namely degenerate positions are excluded (for example no three points lie on a line or no four points lie on a circle). The two assumptions mentioned create a gap between theory and practice. When implementing computational-geometry algorithms these assumptions need to be revised. Overcoming the *real* RAM model assumption may be achieved by using *exact computation* [73] while handling degeneracies needs to be taken into account when implementing robust algorithms.

The Computational Geometry Algorithms Library CGAL, is a C++ software library aiming to provide generic robust implementation of geometric algorithms and data structures. The project was launched in 1996 as a collaborative effort of several research institutes in Europe and Israel. The library provides many implemented algorithms such as Convex hull algorithms, Delaunay triangulations, Voronoi diagrams, Polygon manipulation and more. CGAL is used in various fields including computer graphics, Computer Aided Design (CAD), bioinformatics, motion planning and more.

The C++ library follows the *generic programing paradigm* as presented in [3] which allows for great flexibility in designing and implementing algorithms. It is a programming paradigm where algorithms are described in terms of abstract types. These types (referred to as *models*) have to follow predefined behaviors (referred to as *concepts*) and are provided as parameters at instantiation time of the algorithms. This provides the ability to write generic code that is not type specific. This is best demonstrated by an example: The following code illustrates a generic `swap()` function. When the generic function is compiled it is instantiated with a data type that must be constructible from another member of the data type and must have an assignment operator. A data type that fulfills these requirements is a model of the concepts commonly called *Copy Constructible* and *Assignable* [3].

```
template <typename T> void swap(T& a, T& b)
{
    T tmp  = a;
    a = b;
    b = tmp;
    return;
}
```

CGAL's structure consists of three layers, each providing a defined functionality and making use of the layer beneath it. The top-most layer, consists of the algorithms and data structures. These algorithms operate on geometric objects such as points and lines, and perform tests on them. These geometric objects and predicates are grouped into *Kernels* (the second layer). The third layer consists of fundamental utilities including extensions of the STL [64], BOOST [10], and QT [55] libraries. Another part of the third layer comprises classes representing auxiliary libraries such as special number types and operations on numbers of these types. *Kernels* make use of number types to represent the geometric objects, these number types may be machine provided number types (such as `int` or `double`) but may include exact or multi-precision number types based on packages such as LEDA [46], CORE [16] or GMP [24]. Depending on the algorithm and the input, the different number types and kernels provide a trade-off between efficiency and accuracy. In particular, a number type may make use of *Lazy exact computations*: If the floating-point evaluation of a certain predicate is ensured to give the right answer, it may be used. Otherwise the computation resorts to exact evaluation. Thus the number type is able to detect when it is sufficient to use floating point arithmetic and when exact computation is needed [52].

CGAL's `Arrangement_2` package supports various operations on planar arrangements. The package uses *traits classes* to achieve flexibility and generality. A traits class unites all data types and operations needed by a certain algorithm and is passed as an additional template parameter to the algorithm. The `Arrangement_2` package is templated with a traits class that wraps geometric predicates needed by the algorithm. This design enables the user to use the arrangement package with different types of curves.

For a complete survey on the subjects of robust geometric computation, CGAL and the `Arrangement_2` package the reader is referred to [20, 19, 22, 27, 70].

6.2 Mathematical Background

6.2.1 Definitions

A *rational function* is a function that can be written as a quotient of two *polynomials* [30]. In the context of our work we will limit ourselves to rational functions of one variable with rational coefficients. For this specific case a rational function can be written as $F(x) = \frac{P(x)}{Q(x)}$ where x is the variable and $P(x)$ and $Q(x)$ are polynomial functions of x. The *degree* of a polynomial is the highest exponent for a term with non-zero coefficient in a polynomial expressed in canonical form (i.e. as a sum or difference of terms). A *root* of a polynomial P is a value x_0 such that $P(x_0) = 0$ (thus $(x - x_0)$ is a *factor* of P). The *multiplicity* of a root x_0 is the largest integer k such that $(x - x_0)^k$ is a factor of P. An *algebraic* number is a number that is a root of a non-zero polynomial in one variable with rational (or equivalently, integer) coefficients.

6.2.2 Properties of Polynomials

Let $P(x) = a_m x^m + \ldots + a_0 x^0$ be a polynomial with rational coefficients. $P(x)$ has the following properties [42]:

- Right-hand behavior: If the leading coefficient, a_m, of the polynomial is *positive* (respectively *negative*), then the right-hand side of the graph will *rise* (respectively *fall*) towards $+\infty$ (respectively $-\infty$).
- Left-hand behavior: If the degree, m, of the polynomial is *even* (respectively *odd*), the left-hand side will do the *same* (respectively *opposite*) as the right-hand side.

- X-interception behavior: If the multiplicity of a root is *odd* (respectively *even*), the graph will *change* sides and cross the axis (respectively will *stay* on the same side and touch the axis).

6.2.3 Properties of Rational Functions

Several properties of rational functions are cited here; see [41] for an elaboration on the properties of rational functions. The properties that follow are under the assumption that $F(x) = \frac{P(x)}{Q(x)}$ where $P(x) = a_m x^m + \ldots + a_0 x^0$ and $Q(x) = b_n x^n + \ldots + b_0 x^0$ have rational coefficients and no common factors.

- X-monotonicity: Every rational function is x-monotone[1], since it is a function.
- Coefficient type: Every rational function with rational coefficients can be expressed as an equivalent rational function with integer coefficient. This may be done by multiplying each coefficient of the numerator with the LCM[2] of the coefficients of the denominator and multiplying each coefficient of the denominator with the LCM of the coefficients of the numerator.
- Vertical asymptotes: A rational function has vertical asymptotes in all the roots of the denominator.
- Horizontal asymptotes: A rational function has horizontal asymptotes at $y = 0$ if $m < n$ and at the line $y = \frac{a_m}{b_n}$ if $m = n$.
- Multiplicity and x-interception: If a rational function has a root at $F(x) = 0$ and the root's multiplicity is *odd* (respectively *even*), the graph will change sides and *cross the axis* (respectively stay on the same side and *touch* the axis).
- Multiplicity and vertical asymptotes: If a rational function has a root at the denominator and the root's multiplicity is *odd* (respectively *even*), the graph changes sides and one side of the vertical asymptote rises to $+\infty$ while the other side falls to $-\infty$ (respectively both sides of the vertical asymptote rise to $+\infty$ or both sides fall to $-\infty$).

6.2.4 Properties of Algebraic Numbers

Several properties of algebraic numbers are cited here; see [68] for an elaboration on the properties of algebraic numbers.

- Rational numbers: The set $\mathbb{Q}$ of rational numbers is a subset of the algebraic numbers.
- The field of algebraic numbers: The algebraic numbers form a field in particular the sum, difference and quotient of two algebraic numbers is an algebraic numbers.

An important conclusion derived immediately from the presented properties is:

Corollary 6.2.1. *The intersection of two rational functions $F(x), G(x)$ with integer coefficients, is a point $p = (x, y)$ where x and y are algebraic numbers.*

[1] A continuous planar curve C is x-monotone if every vertical line intersects it at most once.
[2] LCM—Least Common Multiplier.

7 The New Arrangement Traits Class

7.1 Background

The CGAL package `Arrangement_2` follows the *generic programming paradigm* [3]. It is designed to create arrangements of general curves by receiving a *traits class* as a template argument. This traits class defines the type of curves in use and the required operations on them. This gives tremendous flexibility in using the package for a variety of different motion-planning (sub)problems. Here we use it once for line segments via the Minkowski sums package written on top of the arrangement package, and once with graphs of rational functions, for which we devised a new and particularly efficient traits class. The following section gives more details about the latter.

An arrangement traits class is required to provide (and by that define) the curve type in use. For these curves it must provide a specific set of operations, such as splitting curves into x-monotone sub-curves, computing the intersections of two curve segments, comparing intersection points, or comparing the y-order of two curves at a certain x-coordinate. Current implementation of traits classes include support for segments, rays, lines, circular arcs, polylines, conic arcs and rational arcs.

As mentioned in the first part of the thesis, the critical curves defined for our motion planning problem are in the form of rational functions. They are used to create motion paths by creating an arrangement of the curves (see Figure 7.1 for an example of an arrangement of rational functions). Thus, a traits class supporting rational functions was needed. Such a traits class (`CGAL::Arr_rational_arc_traits_2`) existed at the time of writing (see [13]). A second traits class (`CGAL::Arr_algebraic_segment_traits_2`) that supports arbitrary algebraic curves was in the process of making at the time this work started. A traits class is the fundamental building block in creating an arrangement and highly affects running time. Both classes had drawbacks in supported features and time complexity and a new class was designed and implemented as part of this work to overcome the drawbacks.

7.2 Existing Traits Classes

7.2.1 `CGAL::Arr_rational_arc_traits_2`

The traits class `CGAL::Arr_rational_arc_traits_2` (deprecated as of CGAL 3.9) supports rational arcs where the arcs may be bounded over an interval $[x_{min}, x_{max}]$, unbounded, or defined over a "ray", namely, defined over $(\infty, x_{max}]$ or $[x_{min}, \infty)$. The following summary of the class is based on the class documenta-

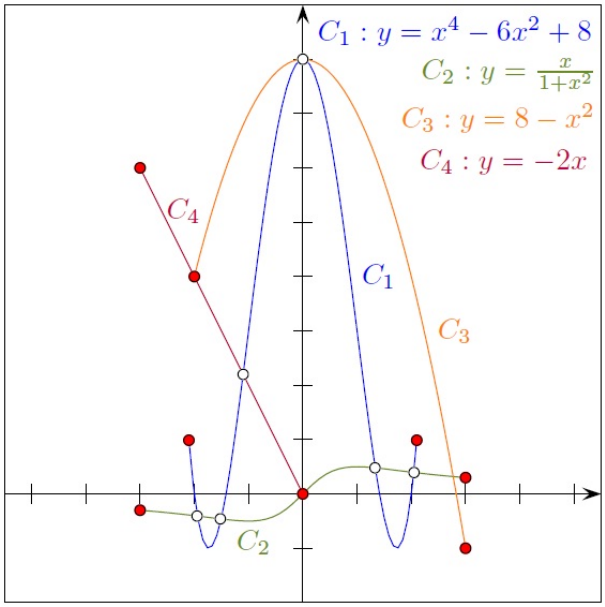


Figure 7.1: Rational function arrangement example (taken from [65, C.30]).

tion [65, C.30].

The polynomial coefficients of the numerator polynomial and the denominator polynomial are represented by integral numbers. Thus the x-coordinates of all arrangement vertices (in particular, those representing intersection points) can be represented as algebraic numbers. The y-coordinates, which can be obtained by simple arithmetic operations on the x-coordinates, are also algebraic numbers.

The class therefore requires separate representations of the curve coefficients and the point coordinates. Thus, the class is templated with two arguments: The first, `NtTraits` requires a class that defines the following nested number types `Integer`, `Rational` and `Algebraic` and supports various operations on them, yielding certified computation results (for example, it can convert rational numbers to algebraic numbers and can compute roots of polynomials with integer coefficients). The second, `AlgKernel` requires a geometric kernel templated with the `NtTraits::Algebraic` number-type. The recommended instantiation is using the `CGAL::CORE_algebraic_number_traits` class as the `NtTraits` parameter, with `CGAL::_Cartesian<NtTraits::Algebraic>` instantiating the kernel. The number types in this case are provided by the CORE [16] library, with its ability to exactly represent algebraic numbers.

The class suffers from two main drawbacks:

- Inefficiency, and
- lack of support for vertical segments.

Efficiency Drawbacks

As the class uses the `CGAL::CORE_algebraic_number_traits` class as the `NtTraits` parameter, predicates supplied by the class are extremely inefficient (timewise). Comparisons of points and intersections require evaluating algebraic expressions which is a time-consuming task. This is done although every predicate has a "topological" point of view: It does not requires the exact number or point considered but an evaluation of the relative position of two features (namely a point and a curve). This is best demonstrated via an example: In order to consider if a point p is above, below or on a curve, the traits class computes the value of the curve at the x-coordinate of the point and compares the two numbers. These computations and comparisons are extremely time consuming and may (as will be showed in 7.4) be done using an alternative approach.

Lack of Support for Vertical Segments

As vertical segments are not rational functions, the traits class does not support this type of curves. Thus a vertical decomposition of an arrangement computed with the traits cannot be computed using only the functionality of the traits class itself.

7.2.2 CGAL::Arr_algebraic_segment_traits_2

At the time of starting this work an additional traits class was underway. This class gave generic support for algebraic curves, namely the (real) zero locus of a polynomial $f(x, y)$ in two variables. Obviously rational curves are a special case of algebraic curves; thus the class supports the desired curves. Vertical segments are also considered as algebraic curves, thus the class overcomes the second drawback mentioned in 7.2.1

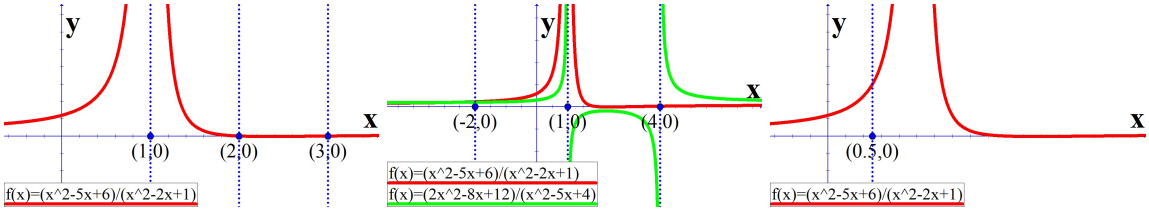
The class is designed to exploit the features of an algebraic kernel which is a model of the concept CGAL::AlgebraicKernel_d_2. The algebraic kernel is targeted to provide black-box implementations of state-of-the-art algorithms to determine, compare and approximate real roots of univariate polynomials. The package can efficiently represent an algebraic number x_0, using a polynomial P which has x_0 as its root, and an isolating interval. The representation of algebraic numbers led to a powerful and efficient traits class as described in [7, 8]. This internal representation leads to a number type that has the following properties:

- CGAL::AlgebraicKernel_d_1::Algebraic_real_1 is real embeddable , for instance, it is possible to compare two algebraic reals, to determine the sign of an algebraic real or to ask for its machine double approximation.
- The representation is output sensitive in the sense that if no two roots lie close, the isolating interval can be coarse.
- The numbers do not allow arithmetic operations (such as adding two real numbers). It is however possible to ask for the sign of a polynomial at an algebraic number.

7.3 General Overview Of The New Traits Class

The traits class CGAL::Arr_algebraic_segment_traits_2 inspired the creation of a new, dedicated traits class, CGAL::Arr_rational_function_traits_2 supporting rational functions only while using the same principles as the CGAL::Arr_algebraic_segment_traits_2 class. This new rational-function traits class was designed to achieve high efficiency surpassing the speed of the general CGAL::Arr_algebraic_segment_traits_2 class by exploiting the fact the curves considered may only be rational functions. Initially the class did not support vertical segments. A support in the form of a class wrapper was later added, suggesting a generic technique for converting a general traits class that does not support vertical segments into a class that supports vertical segments.

The new traits class CGAL::Arr_rational_function_traits_2 defines internally a class CGAL::Arr_rational_arc::Rational_function representing a rational function providing a representation of the complete topology of one rational function. This class gives the necessary foundation for two other internal classes that are the key to the efficiency of the new traits class. The first class, CGAL::Arr_rational_arc::Rational_function_pair represents the analysis of the topology of two rational functions. The second class, CGAL::Arr_rational_arc::Algebraic_point_2 represents a point internally by storing the x-coordinate in the form of a real number and a rational function. The y-coordinate is expressed implicitly as the value of the rational function at the x-coordinate. The internal classes mentioned, combined with an internal caching, constitute the infrastructure to efficiently providing all the predicates required by the traits concept.



(a) Rational function analysis (b) Rational function pair analysis (c) Point representation

Figure 7.2: Internal data structures

7.4 Traits Class Design

7.4.1 Templated Arguments

The traits class `CGAL::Arr_rational_function_traits_2`, the rational curves and almost every internal class are all templated with an *Algebraic Kernel* which is a model of the concept `AlgebraicKernel_d_1`. The Kernel is used for all the operations used on data types (integers, rational numbers, non-algebraic points, algebraic numbers etc.).

7.4.2 Internal Data Structures

Rational Function Implementation For a rational function $F = P/Q$ with $P, Q \in \mathbb{Z}[X]$ coprime, a naïve representation would be to simply store the polynomials P and Q. Considering the predicates required by the traits class reveals that a representation of rational functions should support topological queries efficiently, namely the x-coordinates where the rational function changes sign.

We therefor subdivide the x-axis into intervals such that the sign of F is invariant within each interval[1] (see Figure 7.2a). This is obtained by computing the sorted sequence of the real roots (with multiplicity) of P and Q. For the right-most interval the sign is determined by the signs of the leading coefficients of P and Q. The remaining signs are concluded from right to left using the multiplicity of the computed roots as described in Section 6.2.

Rational-Function Pair Implementation Rational functions serve as the building block used to represent both algebraic points and rational curves. The traits predicates consist of queries on the relative position of pairs of points, curves and points and pairs of curves. Thus the relative position of two rational functions is needed. Namely for a certain x-coordinate, determining whether the first rational function is above, below or intersecting with the second rational function.

Following the above observation, an internal class was created allowing for fast queries on the topological relations of two rational functions at a point. For two functions $F(x) = f_n(x)/f_d(x)$ and $G(x) = g_n(x)/g_d(x)$, we similarly subdivide the x-axis into intervals such that the y-order of F and G is invariant within each interval. Of course, the order may change at intersection points, the x-coordinates of which are given by the real roots of $r = f_n g_d - g_n f_d \in \mathbb{Z}[X]$. However, the order may also change at vertical asymptotes of f and g. Thus, the subdivision is given by the sorted sequence of the real roots of f_d, g_d and r. Again, once the order is computed for one interval, the others can be concluded via the multiplicities of the roots (see Figure 7.2b).

Point Implementation A point is internally stored as an x-coordinate and a rational function (see Figure 7.2c for an example demonstrating this notion) where the y-coordinate is stored implicitly. This allows for efficiently answering predicates concerning topological interrelation of two points. Comparison is trivial by comparing the x-coordinates and the rational functions. A more interesting predicate is the `compare_xy_2` predicate. This predicate receives a second point, and returns an xy-lexicographic comparison result. This

[1] It should be noted that these intervals are not maximal, e.g., a rational function may have a vertical asymptote at a pole where both sides of the pole approach ∞ and the function's sign is equal for both intervals.

is achieved by first comparing the x-coordinates of the two points. If they are the same, a rational function pair is constructed from the point's rational functions. Using the internal analysis of the rational function pair, it is trivial to determine the interrelations of the two rational functions at the appropriate x-coordinate.

7.4.3 Optimization: Handles & Caches

In order to achieve a higher level of performance, several optimizations have been implemented. Each of the classes described above is wrapped using a handle. The handle supports reference counting [23], trivial comparison and rapid copy construction.

A slightly more sophisticated yet much more powerful optimization is the use of caches. The same rational functions and rational-function pairs may be needed several times throughout the program. An example for such a case is when dealing with a rational curve: if the curve needs to be subdivided into x-monotone curves, each of the x-monotone curves will share the same rational function; however, the boundaries of its domain of definitions will be different. Two internal caches are used to store the created rational functions and rational-function pairs. Creation of a new instance is carried out via a factory function [23] which first searches the cache for an existing instance and only upon failure will creates a new instance.

Consider two rational functions F and G, the rational-function pair constructed from F and G is exactly symmetric to the rational function pair constructed from G and F. This observation suggests a canonical representation of the rational-function pair[2]. A wrapper is used to wrap the canonicalized pair and return the necessary outcome.

7.4.4 Rational Curves and Geometric Traits

A rational curve is defined by a rational function and the x-coordinates of the boundaries (if they exist). In order to make it x-monotone, one needs to ensure that the rational function has no poles between the boundaries. Thus both the `Curve_2` and `X_monotone_2` classes share a base class consisting of a rational function, its boundaries and several predicates. These predicates are quite simple and exploit the power of the infrastructure created (namely the rational function, the rational-function pairs and the points).

7.4.5 Vertical Segment Support

As with the `CGAL::Arr_rational_arc_traits_2` class, the new generic traits class does not support vertical segments. This support is achieved as follows[3]: a vertical segment class was added to the traits' infrastructure[4] and a new construction was introduced, supplying the intersection result of a rational curve with a vertical segment.

This simple addition facilitates the creation of a new traits class wrapping the old one. The new traits class tries to represent a curve, either as a rational curve or as a vertical segment. This is implemented using `boost::variant` in a straight-forward fashion [10].

One can organize the predicates required by a traits class into the following types:

1. Predicates supporting queries regarding the traits' point type (e.g. `Compare_x_2`).
2. Predicates supporting queries regarding the traits' curve type (e.g. `Is_vertical_2`).
3. Predicates supporting queries regarding the interaction between the traits' curve type and the traits' point type (e.g. `Compare_y_at_x_2`).
4. Predicates supporting queries regarding the interaction between two instances of the traits' curve type (e.g. `Compare_y_at_x_left_2`).

[2] Canonical representation is done by always considering the first function as the one with the lower handle ID.

[3] This technique may be generalized to add vertical segment support to any traits class that does not support vertical segments.

[4] A vertical segment is simply represented by an x-coordinate, upper and lower rational functions to implicitly represent the upper and lower y-coordinates and flags indicating if the segment is bounded or unbounded from above or below.

A quick glance at the types reveals the necessary work needed to implement each predicate type for the wrapper traits class, namely the class that adds the handling of vertical segments:

1. This predicate type is not affected by the addition of vertical segments; using the appropriate predicate of the traits class suffices.

2. For non-vertical segments, a call to the appropriate predicate of the traits class suffices. For vertical segments these predicates need to be implemented[5].

3. As with the case above, for non-vertical segments, a call to the appropriate predicate of the traits class suffices. For vertical segments these predicates need to be implemented.

4. For the case where the two curve types are the same (both are vertical or both are non-vertical), the method described in section 3 suffices. For interaction between vertical segments and non-vertical segments, care should be exercised. One can implement all these predicates using other predicates already defined in the traits class; hence no new code should be introduced. The only exception is for the `Intersect_2` predicate where actual analysis of the curve is needed.

The quick analysis suggests a generic method for supporting vertical segments in traits classes. A future writer of a traits class may write a new geometric traits for a new type of curves not supporting vertical segments. By implementing only the vertical segments class and an intersection construction with the new type of curves and following the scheme suggested, a wrapper is easily produced, supporting the new curve type and vertical segments.

[5]It is worth noting that if the vertical segments are defined generically (e.g. via the traits point type only), these predicates are generic and not traits dependent.

7.5 Experimental results

We use random instances of rational functions which were also used in another study [7, 8]. Results can be found in Table 7.1. The parameters n, m describe the degree of the rational function and the number of rational functions generated respectively. The parameters V, E, F describe the number of vertices, edges and faces of the constructed arrangement respectively. The results show an average speedup of 3 with respect to `CGAL::Arr_algebraic_segment_traits_2` and a speedup of at least 10 with respect to `CGAL::Arr_rational_arc_traits_2`.

n, m	#(V,E,F)	new rational traits (this work)	algebraic traits	speedup
2,10	(98, 220, 123)	0.01	0.12	6.25
2,20	(360, 766, 407)	0.07	0.35	4.55
2,30	(900, 1862, 963)	0.2	0.75	3.85
2,40	(1836, 3766, 1931)	0.4	1.47	3.70
2,50	(2306, 4724, 2419)	0.55	1.92	3.57
2,60	(3676, 7486, 3811)	0.87	2.9	3.33
2,70	(5066, 10298, 5233)	1.21	4.05	3.45
2,80	(6110, 12394, 6285)	1.5	4.93	3.33
2,90	(7828, 15868, 8041)	1.93	6.37	3.33
2,100	(8918, 18056, 9139)	2.28	7.19	3.23
6,10	(114, 254, 141)	0.05	0.24	4.55
6,20	(566, 1188, 623)	0.28	0.97	3.57
6,30	(1222, 2526, 1305)	0.63	2.09	3.33
6,40	(2074, 4266, 2193)	1.14	3.63	3.23
6,50	(3268, 6670, 3403)	1.89	5.7	3.03
6,60	(4800, 9764, 4965)	2.8	8.24	3.03
6,70	(7020, 14238, 7219)	4.06	11.78	2.94
6,80	(8636, 17482, 8847)	5.21	14.59	2.86
6,90	(11266, 22790, 11525)	6.87	19.15	2.86
6,100	(13124, 26534, 13411)	8.2	22.85	2.86
2,20	(360, 766, 407)	0.07	0.35	4.76
3,20	(422, 890, 469)	0.11	0.48	4.17
4,20	(508, 1066, 559)	0.17	0.64	3.85
5,20	(574, 1204, 631)	0.23	0.84	3.70
6,20	(566, 1188, 623)	0.28	1.01	3.70
7,20	(594, 1242, 649)	0.34	1.16	3.45
8,20	(576, 1204, 629)	0.39	1.31	3.45
9,20	(622, 1312, 691)	0.47	1.62	3.45
10,20	(652, 1368, 717)	0.55	1.81	3.33
11,20	(616, 1292, 677)	0.59	1.94	3.33
12,20	(630, 1322, 693)	0.67	2.2	3.33
13,20	(738, 1536, 799)	0.85	2.65	3.13
14,20	(616, 1288, 673)	0.83	2.81	3.45
15,20	(676, 1418, 743)	0.99	3.14	3.23

Table 7.1: Comparison of generic traits with our new traits for CGAL revision 57740.

n, m	#(V,E,F)	new rational traits (this work)	old rational traits	speedup
2,10	(98, 220, 123)	0.01	0.27	15.4
2,20	(360, 766, 407)	0.07	0.93	12.9
2,30	(900, 1862, 963)	0.2	2.07	11.0
2,40	(1836, 3766, 1931)	0.4	4.59	12.1
2,50	(2306, 4724, 2419)	0.55	5.38	10.6
2,60	(3676, 7486, 3811)	0.87	8.08	9.9
2,70	(5066, 10298, 5233)	1.21	12.24	11.0
2,80	(6110, 12394, 6285)	1.5	13.39	9.7
2,90	(7828, 15868, 8041)	1.93	18.37	10.3
2,100	(8918, 18056, 9139)	2.28	19.74	9.4
6,10	(114, 254, 141)	0.05	2.17	44.4
6,20	(566, 1188, 623)	0.28	12.13	45.2
6,30	(1222, 2526, 1305)	0.63	23.84	38.9
6,40	(2074, 4266, 2193)	1.14	37.84	34.1
6,50	(3268, 6670, 3403)	1.89	58.52	31.5
6,60	(4800, 9764, 4965)	2.8	86.13	32.7
6,70	(7020, 14238, 7219)	4.06	132.4	34.0
6,80	(8636, 17482, 8847)	5.21	152.06	30.7
6,90	(11266, 22790, 11525)	6.87	212.23	32.4
6,100	(13124, 26534, 13411)	8.2	240.09	30.4
2,20	(360, 766, 407)	0.07	0.94	13.5
3,20	(422, 890, 469)	0.11	1.77	16.9
4,20	(508, 1066, 559)	0.17	4.61	28.3
5,20	(574, 1204, 631)	0.23	8.91	39.9
6,20	(566, 1188, 623)	0.28	12.22	46.8
7,20	(594, 1242, 649)	0.34	18.1	54.3
8,20	(576, 1204, 629)	0.39	22.79	61.9
9,20	(622, 1312, 691)	0.47	36.89	79.2
10,20	(652, 1368, 717)	0.55	52.22	97.4
11,20	(616, 1292, 677)	0.59	50.49	86.1
12,20	(630, 1322, 693)	0.67	66.07	98.6
13,20	(738, 1536, 799)	0.85	108.82	128
14,20	(616, 1288, 673)	0.83	93.1	118
15,20	(676, 1418, 743)	0.99	140.72	149

Table 7.2: Comparison of old traits with our new traits for CGAL revision 57740.

8

Conclusions and Future Work

We presented a general and modular algorithmic framework for path planning of robots. The framework combines geometric methods for exact and complete analysis of low-dimensional configuration spaces, together with sampling-based approaches that are appropriate for higher dimensions. The framework suggests sampling entire low-dimensional manifolds of the configuration space. These samples capture the connectivity of the configuration space much better than isolated point samples. In addition, the framework includes several general heuristics and optimizations that may be used for different motion-planning instances.

We have implemented our framework for one such instance, namely a polygonal robot translating and rotating in the plane amidst polygonal obstacles. We present a significant speedup of our implementation over the PRM sampling-based algorithm. The implementation included developing a new traits class for the arrangement package of CGAL which has been integrated into CGAL 3.9.

We present a proof of probabilistic completeness for our scheme for this specific instance. We believe that this is an intermediate step towards a general proof of probabilistic completeness for the entire scheme.

To conclude, we outline directions for extending and enhancing the current work. Our primary goal is to use the MMS framework to solve progressively more complicated motion-planning problems. As suggested earlier, we see the framework as a platform for convenient transfer of strong geometric primitives into motion-planning algorithms. For example, among the recently developed tools are efficient and exact solutions for computing the Minkowski sums of polytopes in $\mathbb{R}^3$ [21, 26, 69] as well as for exact update of the sum when the polytopes rotate [50]. These could be combined into an MMS for full planning of rigid motion of a polytope among polytopes, which, extrapolating from the current experiments, could outperform more simplistic solutions in existence.

Looking at more intricate problems, we anticipate some difficulty in turning constraints into manifolds that can be exactly decomposed. In these cases, we propose to have manifolds where the decomposition yields some *approximation* of the FSCs, using, for example, recent advanced meshing tools. We can endow the connectivity-graph nodes with an attribute describing their approximation quality. One can then decide to only look for paths all whose nodes are above a certain approximation quality. Alternatively, one can extract any solution path and then refine only those portions of the path that are below a certain quality.

We foresee an extension of the framework to other problems that involve high-dimensional arrangements of critical hypersurfaces such as assembly planning or loop closure in molecular modeling. It is difficult to describe the entire arrangement analytically, but there are often situations where constraint manifolds could be computed analytically. Hence, it may be possible to shed light using manifold samples on the problems at hand to analytically capture pertinent information of high-dimensional arrangements of hypersurfaces. Notice that although in Chapter 4 we used only planar manifolds, there are recently developed tools to

construct two-dimensional arrangements of curves on curved surfaces [6] which give further flexibility in choosing the manifold families.

A

Critical Curves of a Rotating Robot Along a Translation Segment

A.1 Background

We consider a polygonal robot translating along a fixed segment while rotating amidst polygonal obstacles. This means that the reference point of the robot moves along a fixed line segment, and the robot can rotate around this reference point. A critical curve is a curve representing a motion of the robot while a feature of the robot boundary is in contact with a feature of a boundary of an obstacle: Either a robot's edge is in contact with an obstacle's vertex or a robot's vertex is in contact with an obstacle's edge. These cases will be referred to as *vertex-edge* and *edge-vertex*, respectively. Figure A.1 demonstrates part of an arrangement of critical curves constructed for the Tunnel scenario (Figure 4.3a) for a segment connecting the source and target configurations. The green and red crosses mark the source and target configurations respectively and the blue poly-line is a path constructed within an FSC.

We define the critical curves by first assuming that both the segment and the edge at hand (either the robot's or the obstacle's) are full lines (containing the segment and the edge). We then introduce "critical endpoints" taking into account the fact that the robot's translation is restricted to a segment and that the edge considered is not a line. This is done by identifying the geometric locus where a vertex and an edge's endpoint are in contact. The rest of this section introduces the necessary notions to analyze the problem. We develop the general equations in Section A.2 and describe the critical endpoints in Section A.3.

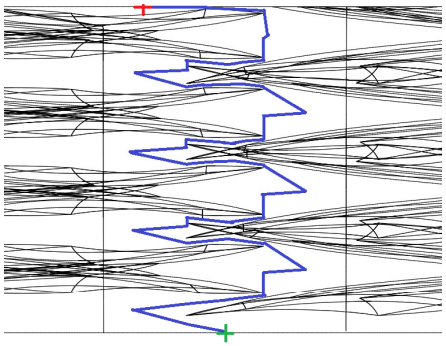


Figure A.1: Arrangement of critical curves

A robot R is a simple polygon with vertices $\{v_1, \ldots, v_n\}$ where $v_i = (x_i, y_i)^T$ and edges $\{(v_1, v_2), \ldots (v_n, v_1)\}$. We assume that the reference point of R is located at the origin. The position of R in the workspace is defined by a configuration $q = (r_q, \theta_q)$ where $r_q = (x_q, y_q)^T$.

Thus, q maps the position of a vertex v_i as follows:

$$v_i(q) = M(\theta_q)v_i + r_q,$$

where $M(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ is a rotation matrix.

Given a segment $seg = [s, t]$ where $s = (x_s, y_s)^T$ and $t = (x_t, y_t)^T$ we define the parametrization $(\alpha, \tau) \in [0, 1] \times \mathbb{R}$ in Equations A.1, A.2

$$r_q = (1 - \alpha)s + \alpha t, \tag{A.1}$$

$$\theta_q = 2 \arctan \tau. \tag{A.2}$$

The parametrization fixes the robot's reference point to the supporting line of seg. The parametrized vertex is represented in Equation A.3

$$v_i(\alpha, \tau) = M(\tau)v_i + (1 - \alpha)s + \alpha t, \tag{A.3}$$

where $M(\tau) = \frac{1}{1+\tau^2} \begin{bmatrix} 1 - \tau^2 & -2\tau \\ 2\tau & 1 - \tau^2 \end{bmatrix}$.

A.2 Critical Curves

A.2.1 Robot's Vertex - Obstacle's Edge

Let v_i be a robot's vertex and $e = (v_{o_1}, v_{o_2})$ be an obstacle's edge that is supported by the line $l : ax+by+c = 0$. The critical curve in this case is defined by adding the constraint that $v_i(q) \in l$ thus:

$$ax_i(q) + by_i(q) + c = 0. \tag{A.4}$$

Plugging Equation A.3 into Equation A.4 yields:

$a[(1 - \tau^2)x_i - 2\tau y_i] + a(1 + \tau^2)(1 - \alpha)x_s + a(1 + \tau^2)\alpha x_t +$

$b[(2\tau)x_i + (1 - \tau^2)y_i] + b(1 + \tau^2)(1 - \alpha)y_s + b(1 + \tau^2)\alpha y_t +$

$c(1 + \tau^2) = 0,$

when simplified:

$\tau^2 \cdot [-ax_i + ax_s - by_i + by_s + c] +$

$\tau \cdot [-2ay_i + 2bx_i] + 1 \cdot [ax_i + ax_s + by_i + by_s + c] +$

$\alpha T^2 \cdot [-ax_s + ax_t - by_s + by_t] + \alpha \cdot [-ax_s + ax_t - by_s + by_t] = 0.$

Hence:

$$\alpha = \frac{p_2\tau^2 + p_1\tau + p_0}{q_2\tau^2 + q_0}, \tag{A.5}$$

where

$p_2 = a(x_i - x_s) + b(y_i - y_s) - c$, $\quad q_2 = a(x_t - x_s) + b(y_t - y_s)$, $\quad a = y_{o_1} - y_{o_2}$,

$p_1 = 2(ay_i - bx_i)$, $\quad b = x_{o_2} - x_{o_1}$,

$p_0 = -a(x_i + x_s) - b(y_i + y_s) - c$, $\quad q_0 = q_2$, $\quad c = x_{o_1}y_{o_2} - x_{o_2}y_{o_1}$.

A.2.2 Robot's Edge - Obstacle's Vertex

Let $e = (v_1, v_2)$ be a robot's edge and v_0 be an obstacle's vertex. The critical curve in this case is defined by adding the constraint that v_0 lies on the line $l : ax + by + c = 0$ supporting e. To simplify the notation we will consider $l : (1+\tau^2)ax + (1+\tau^2)by + (1+\tau^2)c = 0$. This line has the following coefficients: $(1+\tau^2)a = (1+\tau^2)(y_2(q) - y_1(q))$, $(1+\tau^2)b = (1+\tau^2)(x_1(q) - x_2(q))$ and $(1+\tau^2)c = (1+\tau^2)(x_2(q)y_1(q) - x_1(q)y_2(q))$. Let us denote $\Delta_x = (x_2 - x_1)$ and $\Delta_y = (y_2 - y_1)$.

Simplifying a, b, c yields:

$$
\begin{aligned}
(1+\tau^2)a &= (1+\tau^2)(y_2(q) - y_1(q)) \\
&= [(2\tau)x_2 + (1-\tau^2)y_2] + (1+\tau^2)[(1-\alpha)y_s + \alpha y_t] \\
&\quad -[(2\tau)x_1 + (1-\tau^2)y_1] - (1+\tau^2)[(1-\alpha)y_s + \alpha y_t] \\
&= [\Delta_y + (2\Delta_x)\tau - \Delta_y\tau^2],
\end{aligned}
$$

$$
\begin{aligned}
(1+\tau^2)b &= (1+\tau^2)(x_1(q) - x_2(q)) \\
&= [(1-\tau^2)x_1 - 2\tau y_1] + (1+\tau^2)[(1-\alpha)x_s + \alpha x_t] \\
&\quad -[(1-\tau^2)x_2 - 2\tau y_2] - (1+\tau^2)[(1-\alpha)x_s + \alpha x_t] \\
&= [-\Delta_x + \Delta_y 2\tau + \Delta_x\tau^2],
\end{aligned}
$$

$$
\begin{aligned}
(1+\tau^2)c &= (1+\tau^2)(x_2(q)y_1(q) - x_1(q)y_2(q)) \\
&= (1+\tau^2)[\tfrac{1}{1+\tau^2}[(1-\tau^2)x_2 - 2\tau y_2] + (1-\alpha)x_s + \alpha x_t] \\
&\quad [\tfrac{1}{1+\tau^2}[(2\tau)x_1 + (1-\tau^2)y_1] + (1-\alpha)y_s + \alpha y_t] \\
&\quad -(1+\tau^2)[\tfrac{1}{1+\tau^2}[(1-\tau^2)x_1 - 2\tau y_1] + (1-\alpha)x_s + \alpha x_t] \\
&\quad [\tfrac{1}{1+\tau^2}[(2\tau)x_2 + (1-\tau^2)y_2] + (1-\alpha)y_s + \alpha y_t] \\
&= \tfrac{1}{(1+\tau^2)}[-4(x_1y_2 - x_2y_1)\tau^2 - (x_1y_2 - x_2y_1)(1-\tau^2)^2] \\
&\quad +[(1-\tau^2)\Delta_x - 2\tau\Delta_y][y_s + (y_t - y_s)\alpha] \\
&\quad +[-2\tau\Delta_x - (1-\tau^2)\Delta_y][x_s + (x_t - x_s)\alpha] \\
&= -(1+\tau^2)(x_1y_2 - x_2y_1) \\
&\quad +(\Delta_x - 2\Delta_y\tau - \Delta_x\tau^2)(y_s + (y_t - y_s)\alpha) \\
&\quad +(-\Delta_y - 2\Delta_x\tau + \Delta_y\tau^2)(x_s + (x_t - x_s)\alpha).
\end{aligned}
$$

Denoting $k = x_1y_2 - x_2y_1$ and inserting v_0 into the line equation yields:

$$
\begin{aligned}
&[\Delta_y + (2\Delta_x)\tau - \Delta_y\tau^2]x_0 + [-\Delta_x + \Delta_y 2\tau + \Delta_x\tau^2]y_0 - k(1+\tau^2) \\
&+[\Delta_x - 2\Delta_y\tau - \Delta_x\tau^2][y_s + (y_t - y_s)\alpha] + [-\Delta_y - 2\Delta_x\tau + \Delta_y\tau^2][x_s + (x_t - x_s)\alpha] = 0.
\end{aligned}
$$

Now,

$\tau^2 \cdot [\Delta_x(y_0 - y_s) - \Delta_y(x_0 - x_s) - k] +$

$\tau \cdot [2\Delta_x(x_0 - x_s) + 2\Delta_y(y_0 - y_s)] + 1 \cdot [-\Delta_x(y_0 - y_s) + \Delta_y(x_0 - x_s) - k] +$

$\alpha\tau^2 \cdot [-\Delta_x(y_t - y_s) + \Delta_y(x_t - x_s)] +$

$\alpha\tau \cdot [-2\Delta_y(y_t - y_s) - 2\Delta_x(x_t - x_s)] + \alpha \cdot [\Delta_x(y_t - y_s) - \Delta_y(x_t - x_s)] = 0.$

Finally:

$$\alpha = \frac{m_2\tau^2 + m_1\tau + m_0}{n_2\tau^2 + n_1\tau + n_0}, \tag{A.6}$$

where

$m_2 = \Delta_y(x_0 - x_s) - \Delta_x(y_0 - y_s) + k,$ $\quad n_2 = \Delta_y(x_t - x_s) - \Delta_x(y_t - y_s),$

$m_1 = -2\Delta_x(x_0 - x_s) - 2\Delta_y(y_0 - y_s),$ $\quad n_1 = -2\Delta_x(x_t - x_s) - 2\Delta_y(y_t - y_s),$

$m_0 = -m_2 + 2k,$ $\quad n_0 = -n_2$ and

$\Delta_x = (x_2 - x_1)$, $\Delta_y = (y_2 - y_1)$, $k = x_1y_2 - x_2y_1$.

A.3 Critical Endpoints

We consider two types of critical endpoints: The first, denoted *Type 1* results from the fact the edge (either of the obstacle or of the robot) is not a line. The second, denoted *Type 2* results from the fact that the translation segment is not a line.

Type 1 endpoints

Type 1 endpoints represent the (α, τ) values such that a robot's vertex coincides with an obstacle's vertex. For the case of a robot's vertex v_i and an obstacle's edge $e = (v_{o_1}, v_{o_2})$ (vertex-edge case) the robot's vertex is v_i and the obstacle's vertex is either v_{o_1} or v_{o_2}. For the case of a robot's edge $e = (v_1, v_2)$ and an obstacle's vertex v_o the robot's vertex is either v_1 or v_2 and the obstacle's vertex is v_o. As the cases are analyzed in the same manner, we will consider a robot's vertex v_i and an obstacle's vertex v_o.

$$\begin{aligned}
x_o &= x_i(\alpha, \tau) \\
&= \tfrac{1}{1+\tau^2}[(1-\tau^2)x_i - 2\tau y_i] + (1-\alpha)x_s + \alpha x_t \\
(x_s - x_t)\alpha &= \tfrac{1}{1+\tau^2}[(1-\tau^2)x_i - 2\tau y_i] + x_s - x_o \\
(x_t - x_s)\alpha &= \tfrac{1}{1+\tau^2}[(x_i)\tau^2 + (2y_i)\tau - x_i] + (x_o - x_s) \\
\alpha &= \tfrac{(x_i)\tau^2 + (2y_i)\tau - x_i}{(1+\tau^2)(x_t - x_s)} + \tfrac{(x_o - x_s)}{(x_t - x_s)} \\
\alpha &= \tfrac{(x_i)\tau^2 + (2y_i)\tau - x_i}{(1+\tau^2)(x_t - x_s)} + \tfrac{(x_o - x_s)\tau^2 + (x_o - x_s)}{(1+\tau^2)(x_t - x_s)} \\
\alpha &= \tfrac{(x_o - x_s + x_i)\tau^2 + (2y_i)\tau + x_o - x_s - x_i}{(1+\tau^2)(x_t - x_s)}.
\end{aligned}$$

And,

$$
\begin{aligned}
y_o &= y_j(\alpha,\tau) \\
&= \tfrac{1}{1+\tau^2}[(2\tau)x_i + (1-\tau^2)y_i] + (1-\alpha)y_s + \alpha y_t \\
(y_s - y_t)\alpha &= \tfrac{1}{1+\tau^2}[(-y_i)\tau^2 + (2x_i)\tau + (y_i)] + y_s - y_o \\
(y_t - y_s)\alpha &= \tfrac{1}{1+\tau^2}[(y_i)\tau^2 - (2x_i)\tau - (y_i)] + y_o - y_s \\
\alpha &= \tfrac{1}{(1+\tau^2)(y_t-y_s)}[(y_i)\tau^2 - (2x_i)\tau - (y_i)] + \tfrac{y_o-y_s}{y_t-y_s} \\
\alpha &= \tfrac{1}{(1+\tau^2)(y_t-y_s)}[(y_i)\tau^2 - (2x_i)\tau - (y_i)] + \tfrac{(y_o-y_s)(1+\tau^2)}{(y_t-y_s)(1+\tau^2)} \\
\alpha &= \tfrac{[(y_i)\tau^2-(2x_i)\tau-(y_i)]+[(y_o-y_s)\tau^2+(y_o-y_s)]}{(1+\tau^2)(y_t-y_s)} \\
\alpha &= \tfrac{(y_o-y_s+y_i)\tau^2-(2x_i)\tau+(y_o-y_s-y_i)}{(1+\tau^2)(y_t-y_s)}.
\end{aligned}
$$

Hence,

$$
\begin{aligned}
\tfrac{(x_o-x_s+x_i)\tau^2+(2y_i)\tau+x_o-x_s-x_i}{(1+\tau^2)(x_t-x_s)} &= \tfrac{(y_o-y_s+y_i)\tau^2-(2x_i)\tau+(y_o-y_s-y_i)}{(1+\tau^2)(y_t-y_s)} \\
[(y_t-y_s)(x_o-x_s+x_i) - (x_t-x_s)(y_o-y_s+y_i)]\tau^2 & \\
+2[(y_t-y_s)(y_i) + (x_t-x_s)(x_i)]\tau & \\
+[(y_t-y_s)(x_o-x_s-x_i) - (x_t-x_s)(y_o-y_s-y_i)] &= 0
\end{aligned}
$$

Type 2 Endpoints

The robot's refernce point coincides with the translation segments's endpoints when $\alpha = 0$ or $\alpha = 1$. Thus for the *vertex-edge* case, for $\alpha = 0$ the endpoints are located at the roots of Equation A.7 and for $\alpha = 1$ the endpoints are located at the roots of Equation A.8. For the *edge-vertex* case, for $\alpha = 0$ the endpoints are located at the roots of Equation A.9 and for $\alpha = 1$ the endpoints are located at the roots of Equation A.10.

$$p_2\tau^2 + p_1\tau + p_0 = 0 \tag{A.7}$$

$$(p_2 - q_2)\tau^2 + (p_1 - q_1)\tau + (p_0 - q_0) = 0 \tag{A.8}$$

where $p_i,\ i \in \{1\ldots3\}$, $q_j,\ j \in \{1,3\}$ are defined in Equation A.5.

$$m_2\tau^2 + m_1\tau + m_0 = 0 \tag{A.9}$$

$$(m_2 - n_2)\tau^2 + (m_1 - n_1)\tau + (m_0 - n_0) = 0 \tag{A.10}$$

where $m_i,\ i \in \{1\ldots3\}$, $n_j,\ j \in \{1,3\}$ are defined in Equation A.6.

Bibliography

[1] Boris Aronov and Micha Sharir. On translational motion planning of a convex polyhedron in 3-space. *SIAM J. Comput.*, 26(6):1785–1803, 1997.

[2] Franz Aurenhammer and Rolf Klein. *Handbook of Computational Geometry*, chapter 5, pages 201–290. Elsevier Publishing House, December 1999.

[3] Matthew H. Austern. *Generic programming and the STL: using and extending the C++ Standard Template Library.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[4] Francis Avnaim, Jean-Daniel Boissonnat, and Bernard Faverjon. A practical exact motion planning algorithm for polygonal object amidst polygonal obstacles. In *Workshop on Geometry and Robotics*, pages 67–86, London, UK, 1989. Springer-Verlag.

[5] Saugata Basu, Richard Pollack, and Marie-Francoise Roy. *Algorithms in Real Algebraic Geometry.* Algorithms and Computation in Mathematics. Springer-Verlag, 2003.

[6] Eric Berberich, Efi Fogel, Dan Halperin, Kurt Mehlhorn, and Ron Wein. Arrangements on parametric surfaces I: General framework and infrastructure. *Mathematics in Computer Science*, 4(1):45–66, 2010.

[7] Eric Berberich, Michael Hemmer, and Michael Kerber. A generic algebraic kernel for non-linear geometric applications. Research Report 7274, INRIA, 2010.

[8] Eric Berberich, Michael Hemmer, and Michael Kerber. A generic algebraic kernel for non-linear geometric applications. In *Symposium on Computational Geometry*, pages 179–186, New York, NY, USA, 2011. ACM.

[9] Eric Berberich, Michael Kerber, and Michael Sagraloff. Exact geometric-topological analysis of algebraic surfaces. In *Symposium on Computational Geometry*, pages 164–173, College Park Maryland, USA, 2008.

[10] Boost — portable C++ libraries.
`http://www.boost.org`.

[11] John Canny. *Complexity of Robot Motion Planning (ACM Doctoral Dissertation Award).* The MIT Press, June 1988.

[12] John Canny, Bruce Donald, and Eugene K. Ressler. A rational rotation method for robust geometric algorithms. In *Symposium on Computational Geometry*, pages 251–260, New York, NY, USA, 1992. ACM.

[13] Cgal — computational geometry algorithms library.
`http://www.cgal.org`.

[14] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. A singly exponential stratification scheme for real semi-algebraic varieties and its applications. *Theoretical Computer Science*, 84(1):77 – 105, 1991.

[15] Howie Choset, Wolfram Burgard, Seth Hutchinson, George Kantor, Lydia E. Kavraki, Kevin Lynch, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementation.* MIT Press, Boston, MA, USA, June 2005.

[16] Core number library.
`http://cs.nyu.edu/exact/core_pages`.

[17] Mark De Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer, third edition, 2008.

[18] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[19] Eyal Flato, Dan Halperin, Iddo Hanniel, Oren Nechushtan, and Eti Ezra. The design and implementation of planar maps in CGAL. In *Abstracts 15th European Workshop Comput. Geom.*, pages 154–168. INRIA Sophia-Antipolis, 1999.

[20] Eyal Flato, Dan Halperin, Ido Hanniel, Oren Nechushtan, and Eti Ezra. The design and implementation of planar maps in CGAL. *ACM Journal of Experimental Algorithmics*, 5, 2000. Special Issue, selected papers of the Workshop on Algorithm Engineering (WAE).

[21] Efi Fogel and Dan Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. *CAD*, 39(11):929–940, 2007.

[22] Efi Fogel, Ron Wein, and Dan Halperin. Code flexibility and program efficiency by genericity: Improving cgal's arrange-

ments. In *European Symposium on Algorithms*, pages 664–676, 2004.

[23] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[24] GMP — GNU multiple precision arithmetic library. `http://gmplib.org`.

[25] Leonidas J. Guibas, Micha Sharir, and Shmuel Sifrony. On the general motion-planning problem with two degrees of freedom. *Discrete & Computational Geometry*, 4:491–521, 1989.

[26] Peter Hachenberger. Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica*, 55(2):329–345, 2009.

[27] Dan Halperin. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002.

[28] Dan Halperin and Micha Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete & Computational Geometry*, 16(2):121–134, 1996.

[29] Kris K. Hauser, Timothy Bretl, Jean-Claude Latombe, and Brian Wilcox. Motion planning for a six-legged lunar robot. In *Workshop on the Algorithmic Foundations of Robotics*, pages 301–316, 2006.

[30] Michiel Hazewinkel, editor. *Encyclopaedia of Mathematics*. Springer, 1995.

[31] Shai Hirsch and Dan Halperin. Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane. In *Workshop on the Algorithmic Foundations of Robotics*, pages 225–241, 2002.

[32] David Hsu, Jean claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. *Int. J. Comp. Geo. & App.*, 4:495–512, 1999.

[33] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. In *International Journal of Robotics Research*, volume 25, pages 627–643, Thousand Oaks, CA, USA, July 2006. Sage Publications, Inc.

[34] Lydia E. Kavraki. Geometry and the discovery of new ligands. In *Algorithms for Robotic Motion and Manipulation*, pages 435–448. A.K. Peters, 1997.

[35] Lydia E. Kavraki, Mihalis. N. Kolountzakis, and Jean-Claude Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[36] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[37] Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, April 1986.

[38] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. *Comput. Geom. Theory Appl.*, 40:61–78, May 2008.

[39] James J. Kuffner and Steven M. Lavalle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[40] Andrew Ladd and Lydia E. Kavraki. Generalizing the analysis of PRM. In *IEEE International Conference on Robotics and Automation*, pages 2120–2125, 2002.

[41] Serge Lang. *Algebra*. Springer, Berlin, 2002.

[42] Ron Larson. *College Algebra: A Graphing Approach*. Cengage Learning, 1999.

[43] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[44] Steven M Lavalle. Rapidly-exploring random trees: A new tool for path planning. *In Computer Science Dept., Iowa State University*, Tech. Rep:98–11, 1998.

[45] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[46] LEDA — library of efficient data types and algorithms. `http://www.algorithmic-solutions.com/leda/index.htm`.

[47] Jyh-ming Lien. Hybrid motion planning using Minkowski sums. In *In Proc. Robotics: Sci. Sys. (RSS)*, 2008.

[48] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *MIT AI Memo 605*, 1980.

[49] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

[50] Naama Mayer, Efi Fogel, and Dan Halperin. Fast and robust retrieval of Minkowski sums of rotating convex polyhedra in 3-space. In *Symposium on Solid and Physical Modeling*, pages 1–10, 2010.

[51] Nils J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proc. of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, Washington, DC, 1969.

[52] Sylvain Pion and Andreas Fabri. A generic lazy evaluation scheme for exact geometric computations. *Sci. Comput. Program.*, 76(4):307–323, 2011.

[53] Erion Plaku, Kostas E Bekris, and Lydia E Kavraki. OOPS for motion planning: An online open-source programming system. In *International Conference on Robotics and Automation*, pages 3711–3716. IEEE, April 2007.

[54] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[55] QT — a cross-platform application and ui framework. `http://qt.nokia.com/products/`.

[56] John H. Reif. Complexity of the mover's problem and generalizations. In *Symposium on Foundations of Computer Science*, pages 421–427, Washington, DC, USA, 1979. IEEE Computer Society.

[57] Oren Salzman, Michael Hemmer, Barak Raveh, and Dan Halperin. Motion planning via manifold samples. In *arXiv:1107.0803*, 2011.

[58] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Commun. Pure appl. Math*, 35:345 – 398, 1983.

[59] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298 – 351, 1983.

[60] Micha Sharir. *Handbook of Discrete and Computational Geometry*, chapter 47, pages 733–754. Goodman, Jacob E. and O'Rourke, Joseph, CRC Press, Inc., Boca Raton, FL, USA, 2 edition, 2004.

[61] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and their Geometric Applications*. Cambridge University Press, New York, NY, USA, 2010.

[62] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Professional, 2001.

[63] Amit P. Singh, Jean-Claude Latombe, and Douglas L. Brutlag. A motion planning approach to flexible ligand binding. In *Intelligent Systems for Molecular Biology*, pages 252–261, 1999.

[64] STL — C++ standard template library. `http://www.sgi.com/tech/stl`.

[65] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 3.7 edition, 2010. http //www.cgal.org/.

[66] Konstantinos I. Tsianos, Ioan Alexandru Sucan, and Lydia E. Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1:2–11, August 2007.

[67] Jur van der Berg, Sachin Patil, Ron Alterovitz, Pieter Abbeel, and Ken Goldberg. LQG-based planning, sensing, and control of steerable needles. In *Workshop on the Algorithmic Foundations of Robotics*, pages 373–389, 2010.

[68] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[69] Ron Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *European Symposia on Algorithms*, pages 829–840, 2006.

[70] Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin. Advanced programming techniques applied to CGAL's arrangement package. In *Computational Geometry: Theory and Applications*, page 05, 2005.

[71] Jade Yang and Elisha Sacks. RRT path planner with 3 DOF local planner. In *International Conference on Robotics and Automation*, pages 145–149, 2006.

[72] Chee-Keng Yap. An $O(n \log n)$ algorithm for the voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2:365–393, 1987.

[73] Chee-Keng Yap. Towards exact geometric computation. *Computational Geometry: Theory and Applications*, 7:3–23, January 1997.

[74] Liangjun Zhang, Young J. Kim, and Dinesh Manocha. A hybrid approach for complete motion planning. In *International Conference on Intelligent Robots and Systems*, pages 7–14, 2007.