Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems

Path Planning with Different Homotopy Classes in Environment with Obstacles

Master's Thesis

Matej Novosad

Prague, May 2023

Study programme: Cybernetics and Robotics
Supervisor: Ing. Robert Pěnička, Ph.D.

Author Statement

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

Prague, 26 May 2023 ....

Matej Novosad

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Novosad Matej**	Personal ID number:	**487012**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Path Planning with Different Homotopy Classes in Environment with Obstacles

Master’s thesis title in Czech:

Plánování cest s rozdílnou třídou homotopie v prostředí s překážkami

Guidelines:

1. Get familiar with sampling-based motion planning, primarily Probabilistic Roadmaps and similar approaches.
2. Study the state-of-the-art methods for path planning with different homotopy classes.
3. Propose and implement an algorithm for fast planning of multiple paths with different homotopy classes.
4. Verify functionality of the proposed algorithm in different environments.
5. Implement state-of-the-art methods and compare the performance of the proposed algorithm with performance of the state-of-the-art methods.

Bibliography / sources:

[1] R. Penicka and D. Scaramuzza, "Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments," in IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 5719-5726, 2022.
[2] B. Zhou, J. Pan, F. Gao and S. Shen, "RAPTOR: Robust and Perception-Aware Trajectory Replanning for Quadrotor Fast Flight," in IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1992-2009, 2021.
[3] E. Schmitzberger, J. L. Bouchet, M. Dufaut, D. Wolf and R. Husson, "Capture of homotopy classes with probabilistic road map," IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2317-2322, 2002.
[4] Jaillet L, Simeon T. Path Deformation Roadmaps: Compact Graphs with Useful Cycles for Motion Planning. The International Journal of Robotics Research. 2008;27(11-12):1175-1188.
[5] C. Nissoux, T. Simeon and J.-P. Laumond, "Visibility based probabilistic roadmaps," Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289), Kyongju, Korea (South), 1999, pp. 1316-1321 vol.3

Name and workplace of master’s thesis supervisor:

Ing. Robert Pěnička, Ph.D. Multi-robot Systems FEE

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

Date of master’s thesis assignment: **24.01.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

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

III. Assignment receipt

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

.	
Date of assignment receipt	Student's signature

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor Ing. Robert Pěnička, Ph.D., for his invaluable guidance, support, and mentorship throughout the course of my master's thesis. I am truly grateful for his insightful feedback, constructive criticism, and constant encouragement that have greatly contributed not only to the successful completion of this thesis, but my academic growth as well.

I would also like to express my sincere appreciation to Ing. Vojtěch Vonásek, Ph.D. for his valuable contributions throughout my master's thesis. His insights and expertise have greatly enriched this research work.

Last but not least, I would like to thank my family for their unwavering love, encouragement, and belief in my abilities.

Abstract

This thesis proposes a new method called Clustering Topological PRM (CTopPRM) for finding multiple homotopically distinct paths in 3D cluttered environments. Finding such distinct paths, e.g. going around an obstacle from a different side, is useful in many applications. Among others, using multiple distinct paths is necessary for optimization-based trajectory planners where found trajectories are restricted to only a single homotopy class of a given path. Distinct paths can also be used to guide sampling-based motion planning and thus increase the effectiveness of planning in environments with narrow passages. Graph-based representation called roadmap is a common representation for path planning and also for finding multiple distinct paths. However, challenging environments with multiple narrow passages require a densely sampled roadmap to capture the connectivity of the environment. Searching such a dense roadmap for multiple paths is computationally too expensive. Therefore, the majority of existing methods construct only a sparse roadmap which, however, struggles to find all distinct paths in challenging environments. To this end, we propose the CTopPRM which creates a sparse graph by clustering an initially sampled dense roadmap. Such a reduced roadmap allows fast identification of homotopically distinct paths captured in the dense roadmap. Comparison with the existing methods shows, that the CTopPRM improves the probability of finding all distinct paths by almost 20% in tested environments, during same run-time.

Keywords homotopy, motion planning, multi-path planning, guiding path, clustering

Abstrakt

Tato práce představuje novou metodu nazvanou Clustering Topological PRM (CTop-PRM), která umožňuje najít více homotopicky odlišných cest ve 3D prostředích s překážkami. Nalezení více takových cest, které na příklad obcházejí překážku z opačných stran, má mnoho využití. Například u optimalizačních metod na plánování trajektorií, kde nalezené trajektorie jsou omezeny pouze na homotopickou třídu původní cesty. Homotopicky odlišné cesty mohou být také využity jako naváděcí cesty pro vysoce dimenzionalní plánování pohybu a zlepšit tak účinnost plánování v prostředích s úzkými průchody. Metody pro nalezení homotopicky odlišných cest často používají grafovou reprezentaci prostředí, tzv. roadmap. Složitá prostředí s úzkými průchody vyžadují hustě vzorkovanou roadmapu, na to aby zachytila propojitelnost prostředí. Prohledávání tak husté roadmapy pro více cest je však výpočetně náročné. Navrhovaná metoda CTopPRM vytváří redukovaný graf shlukováním husté roadmapy. Tato redukovaná roadmapa umožňuje rychlou identifikaci vysokého počtu homotopicky odlišných cest. Porovnání s existujícími metodami ukazuje, že CTop-PRM zvyšuje pravděpodobnost nalezení všech cest o téměř 20 % v testovaných prostředích při zachování stejného času výpočtu.

Klíčová slova homotopie, plánování pohybu, plánování více cest, naváděcí cesta, shlukování

Abbreviations

BFS Breadth-first Search

DFS Depth-first Search

ESDF Euclidean Signed Distance Field

FIFO First In First Out

GT Ground Truth

LIFO Last In First Out

MSF Minimum-spanning Forest

MST Minimum-spanning Tree

PRM Probabilistic Roadmap

RL Reinforcement Learning

RRT Rapidly-exploring Random Trees

UAV Unmanned Aerial Vehicle

UVD Uniform Visibilty Deformation

VD Visibilty Deformation

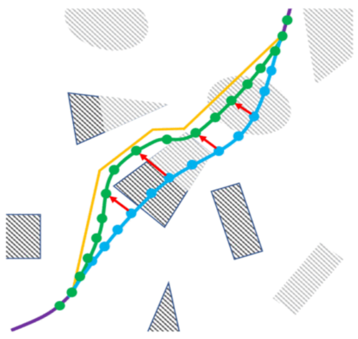
Contents

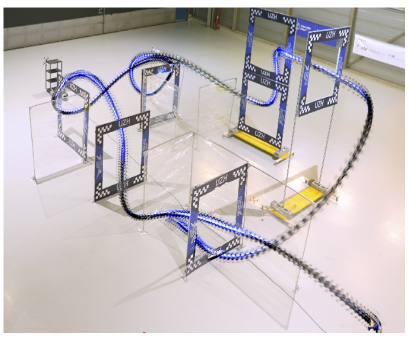
Chapter 1

Introduction

Path planning is one of the fundamental problems in robotics. It requires finding a geometrical path for a robot between given start and goal positions while avoiding collisions. However, there are several applications, depicted in Figure 1.1, that would benefit from having multiple alternative paths. To address this issue, paths with different homotopy classes should be considered. Such distinct paths cannot be transformed into each other through a continuous deformation without crossing obstacles. Therefore, homotopically distinct paths represent the topological connectivity of a cluttered environment and allow the robot to select different ways how to navigate through the environment (see Figure 1.3(d) with multiple distinct paths in building-like environment).

Finding multiple paths with distinct homotopy classes can be used, for example, in optimization-based [7] or sampling-based [5] trajectory planners (Fig. 1.1a). Trajectories are time-parametrized paths, that also have to take robot dynamics into consideration. Finding multiple paths helps to find the optimal trajectory as the trajectory planning is restricted to a single homotopy class of a given initial path. Similarly, the paths can also be used to guide Reinforcement Learning (RL) methods for agile flight [4] (Fig. 1.1b). Last but not least, finding multiple distinct paths is beneficial for guided-based planners (Fig. 1.1c) solving high-dimensional motion planning problems [2], [8], [9], [12], [17]. These planners sample the configuration space of the robot around the guiding paths and thus increase the effectiveness of planning in environments with narrow passages. However, while online trajectory planning requires fast computation, guided-based planners and reinforcement learning methods often benefit more from acquiring a higher number of guiding paths. Therefore, when searching for multiple homotopically distinct paths, a trade-off between computational time and the number of found distinct paths has to be considered. This was so far the main stumbling block of existing methods for finding homotopically distinct paths.


(a) trajectory re-planning [7]


(b) RL of agile flight [4]

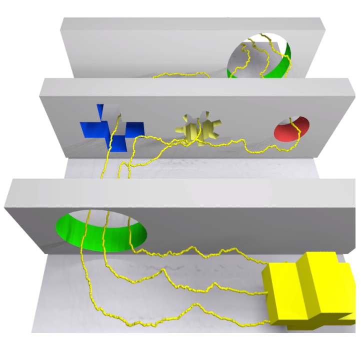

(c) guided motion planning [12]

Figure 1.1: Having multiple distinct paths increases effectiveness of trajectory re-planning (a), reinforcement learning of agile flight (b) and guided-based planning (c).

The majority of existing methods for planning multiple distinct paths use graph-based roadmap representation. To discover all distinct paths, especially in challenging 3D environments, a dense roadmap is required (see Figure 1.2b). A path can be then found in the roadmap using any standard graph searching algorithm such as Dijkstra's. However, searching for multiple paths as proposed in [26] proves to be computationally expensive, even more so in a dense roadmap. Moreover, many found paths would belong to the same homotopy class, requiring an exhaustive filtering process. Visibility-PRM [31] introduced a concept that allows the construction of a sparse roadmap, that was used by [7], [24] and [28]. Yet, Visibility-PRM's reliability, i.e., the ability to consistently capture all homotopy classes, is limited, particularly in environments with narrow passages, as shown in Figure 1.2a. Narrow passages are small regions of the configuration space and many samples are required to capture paths leading through them, which results in dense roadmaps. However, finding multiple distinct paths in the dense roadmaps is computationally very demanding. This motivates us to reduce the dense roadmaps into a sparse roadmaps that can be searched faster.

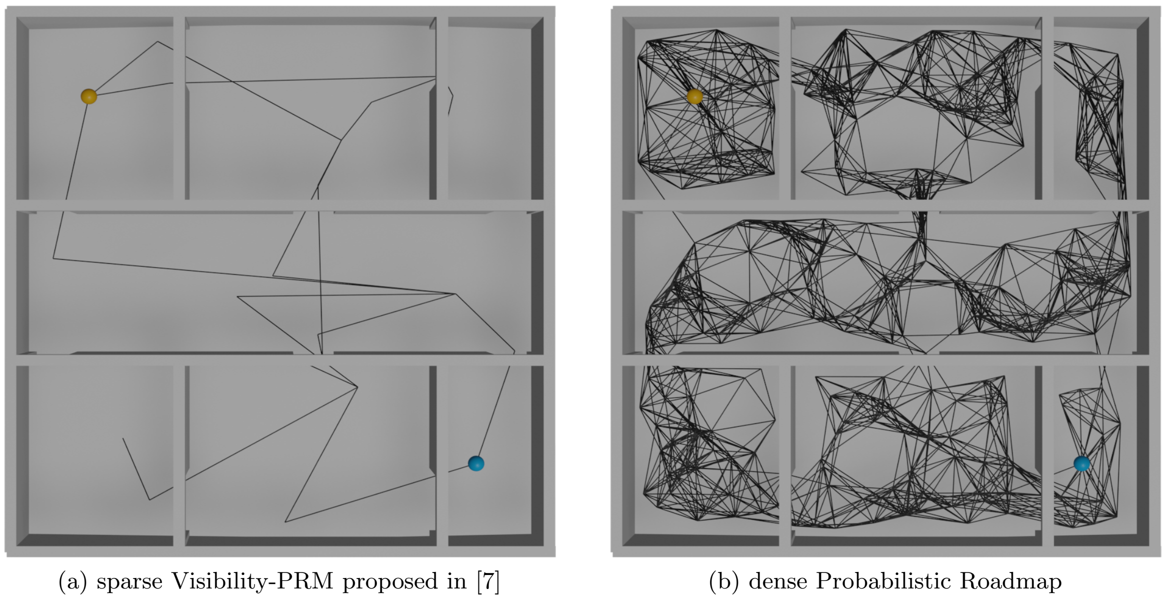

(a) sparse Visibility-PRM proposed in [7]　(b) dense Probabilistic Roadmap

Figure 1.2: Comparison of free-space captured by sparse and dense roadmaps, with the same number of sampled configurations. Even though the sparse roadmap (a) can be searched more efficiently, it fails to capture all homotopy classes that a dense Probabilistic Roadmap (PRM) (b) captures.

To this end, we propose a novel sampling-based method called Clustering Topological PRM (CTopPRM)[1], that clusters a densely sampled roadmap, to construct a sparse graph with cluster centroids as vertices, greatly reducing their number. This reduced roadmap allows fast path searching while capturing all homotopy classes that the initial dense roadmap had captured, including those that require traversal of narrow passages. Moreover, the algorithm allows adjusting trade-off between computational time and the number of paths found through a single parameter, making it suitable for both online planning within tens of milliseconds, and for offline planning with narrow passages. Main steps of the method are visualized in Figure 1.3.

[1]Video providing a step by step description and visualization of the CTopPRM algorithm is available at `https://youtu.be/azNrWBU5cAk`.

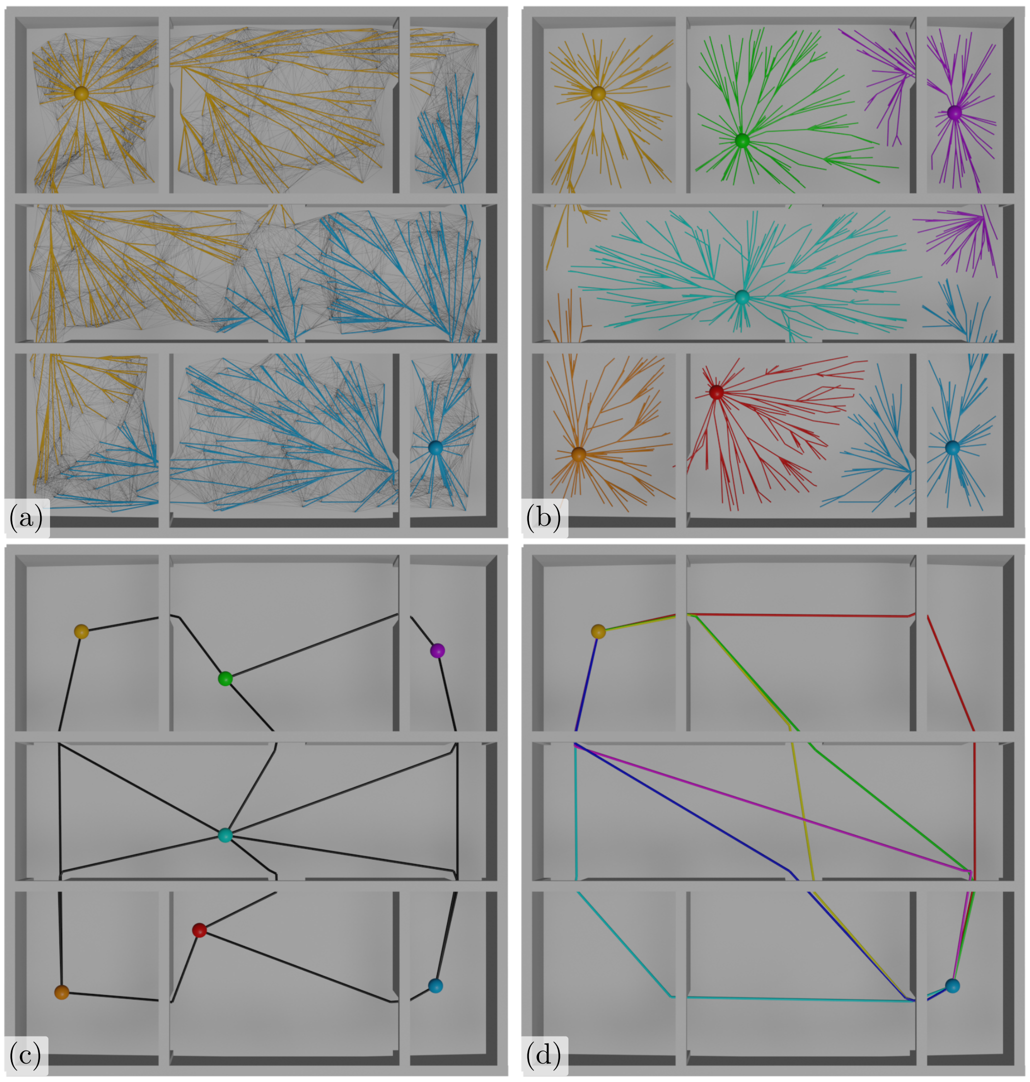


Figure 1.3: Illustration of the proposed CTopPRM method which starts by creating a dense PRM clustered around the start and goal positions (a). New cluster centroids are then iteratively added to promising places in the roadmap (b) to create a sparse graph (c), which is finally used to find distinct paths (d). Video available at `https://youtu.be/azNrWBU5cAk`.

Contributions of this thesis are as follows. We introduce an efficient method for identifying homotopically distinct topological paths, with a controllable balance between computational time and quantity of identified paths. We demonstrate that our approach, called CTopPRM, outperforms other existing methods in a variety of challenging cluttered environments. In scenarios with an easily determinable number of distinct homotopy classes, CTopPRM is shown to successfully identify 94% of all distinct paths while the other methods manage to find less than 80% of paths. In more complex environments, with a high number of distinct homotopy classes, we improved the average number of homotopy classes detected within the same run-time between 30% and 300%, depending on the scenario. A short version of this thesis was submitted to the IEEE Robotics and Automation Letters (RA-L) and the preprint is available at arXiv [1].

Chapter 2

Preliminaries

This chapter serves as the foundation for this master's thesis, providing essential background information and establishing the groundwork for the subsequent chapters. It summarizes data structures, methods and algorithms used in this thesis.

2.1 Grid-based Representation of the Planning Environment

Grid-based methods are frequently used to represent the planning environment, because they are simple to use and implement. In these approaches, the environment is divided into a grid of cells, with each cell being assigned a value to indicate its occupancy or other relevant information, e.g. distance from closest obstacle.

An often utilized grid-based representation is the occupancy grid [36]. Each cell in an occupancy grid contains a probability of it being occupied. If the probability is higher than a threshold value, that cell is considered occupied, making occupancy grids easy to implement and use. However, occupancy grids are sensitive to the grid resolution, which can impact the accuracy of collision detection and lead to the inability to capture narrow passages. They can also be computationally and memory expensive, particularly for large environments. Furthermore, since they contain only the occupancy of individual cells, occupancy grids struggle to represent continuity of space.

Euclidean Signed Distance Field (ESDF) is a grid-based representation that manages to represent the continuity of space to a certain degree. Each cell contains a value representing the distance to the nearest obstacle. Therefore, ESDF can be used to create gradient in the distance to obstacles which is valuable for planning. For this reason, we use ESDF for collision-checking and for shortening of paths, while maintaining their homotopy class, as explained in Section 5.5. Some of the popular approaches for building ESDF are Voxblox [15], FIESTA [10] and iSDF [3].

Despite their ability to effectively capture continuity of space, ESDF, just like all other grid-based representations of the planning environment, struggle with representation of high-dimensional motion planning problems. As the dimensionality of the planning environment increases, grid-based methods become exponentially more computationally and memory-intensive. Additionally, their simplicity, which is one of their main strengths, is lost for dimensions higher than three, rendering them impractical in the high-dimensional motion planning problems.

2.2 Graph-based Representation of Free-space

To efficiently represent high-dimensional continuous free-space $\mathcal{C}_{\text{free}}$ in the planning environment, graph-based representation called roadmap ($\mathcal{G} = (V, E)$, with vertices V and edges E) is commonly used. Two types of approaches are used to solve motion planning problems: combinatorial and sampling-based [25].

Combinatorial approaches find paths through the continuous environment without resorting to approximations of the object, obstacles, or collisions. All combinatorial approaches are complete, which means they will either find a solution or will correctly report that no solution exists. Visibility graphs [37] and Voronoi diagrams [35] are commonly used as roadmaps for combinatorial approaches. However, combinatorial motion planning requires occupied space $\mathcal{C}_{\text{obs}}$ to be represented explicitly (e.g. by polygonal obstacle regions). Creation of such representation is usually unreasonable in practical applications, especially in complex high-dimensional environments.

The main idea of sampling-based motion planning methods [25] is to avoid the explicit representation of $\mathcal{C}_{\text{obs}}$. Search of the planning environment is instead conducted with a sampling scheme, enabled by a collision detection module, which uses approximations to evaluate object collisions. The collision detection module takes a configuration as an input and returns whether the manipulated object in this configuration collides with the obstacles. This allows motion planning in complex environments, without the need for particular geometric models. To save memory and time space is sampled randomly and a roadmap is created from collision-free configurations, which are connected by a local planner method. Two most commonly used sampling-based planners are the Probabilistic Roadmap (PRM) [34] and Rapidly-exploring Random Trees (RRT) [33].

2.2.1 Probabilistic Roadmap

The PRM algorithm is divided into two phases. First is called construction phase, where configuration space $\mathcal{C}$ is randomly sampled for configurations, also called samples, which are added to the roadmap, if they are collision-free. After a predetermined number of collision-free samples is generated, they are connected to their neighbours, defined by neighbourhood method in Line 3 of Algorithm 1. Original PRM implementation [34] didn't allow creation of cycles, thus a modified variant sPRM [22] is more popular nowadays. Most commonly used variant is k-sPRM, in which each sample is connected to its k-nearest neighbours. Pseudo-code for construction phase of sPRM is in Algorithm 1.

Once the roadmap is constructed, query phase follows. During the query phase, start and goal configurations are connected to the roadmap and a collision-free path from the start to the goal is found by any standard graph-search algorithm, such as Dijkstra's algorithm or A*.

One of the most significant advantages of PRM is re-usability. The PRM algorithm is a multi-query method, which means that the same constructed roadmap can be used for multiple start/goal queries. This leads to usually slower roadmap construction because a roadmap that covers entire $\mathcal{C}_{\text{free}}$ is required, but makes PRM suitable for methods requiring frequent planning or re-planning.

Algorithm 1: Simplified PRM (sPRM)

```
Input: N desired number of samples, $\mathcal{C}$ configuration space
Out: $\mathcal{G}$=$(V, E)$ roadmap with vertices V and edges E
1  $V \leftarrow$ sample_free($\mathcal{C}$, N); $E \leftarrow \emptyset$
2  for each $v \in V$ do
3      $V_n \leftarrow$ neighbourhood($v$)
4      for each $u \in V_n, u \neq v$ do
5          if can_connect($u, v$) then
6              $E \leftarrow E \cup \{(u, v)\}$
7  $\mathcal{G} \leftarrow (V, E)$
```

2.2.2 Rapidly-exploring Random Trees

The RRT algorithm [33], in contrast to PRM, starts with the start configuration as the root of a tree graph. It then repeatedly generates a random configuration q_{rand} in the workspace and attempts to connect it to the existing tree. To do this, the node in the tree q_{near} nearest to the random configuration is found. New configuration q_{new} is generated in the neighbourhood of the nearest node and in the direction of the random configuration. This new configuration is then connected to the nearest node through a collision-free path, creating an edge between them and adding the new configuration to the tree. Once the tree grows close enough to the goal configuration, a collision-free path between q_{start} and g_{goal} is built. This can be done easily and efficiently due to the tree structure of the roadmap. Algorithm 2 summarizes the RRT algorithm.

Algorithm 2: Rapidly-exploring Random Tree (RRT)

```
Input: $\mathcal{C}$ configuration space, $q_{start}$ starting configuration, $g_{goal}$ goal configuration
Global params.: $I_{max}$ maximum number of iterations
Out: $\pi$ path from $q_{start}$ to $g_{goal}$
1  initialize tree $\mathcal{T}(q_{start})$
2  for $i = 1, \ldots, I_{max}$ do
3      $q_{rand} \leftarrow$ random_sample($\mathcal{C}$)
4      $q_{near} \leftarrow$ nearest_point($\mathcal{T}, q_{rand}$)
5      $q_{new} \leftarrow$ steer($q_{rand}, q_{near}$)
6      if can_connect($q_{near}, q_{new}$) then
7          $\mathcal{T}$.add_vertex($q_{new}$)
8          $\mathcal{T}$.add_edge($q_{near}, q_{new}$)
9          if is_near($q_{new}, q_{goal}$) then
10             construct path $\pi$ from $q_{start}$ to $g_{goal}$
11             return $\pi$
```

RRT is a single-query method. This means each tree is built to answer a single start/goal query. The sampling of $\mathcal{C}$ is terminated if the query can be answered, making it practically faster. However, any subsequent planning requires the entire algorithm to start over, and build a new tree, making it impractical for repeated planning.

2.2.3 Asymptotically Optimal RRT* and PRM*

Although k-sPRM and RRT algorithms excel in quick graph construction, they do not consider any optimality criteria, even though both are probabilistically complete. Path planning algorithm is probabilistically complete if probability p of finding a solution to the path planning problem $\mathcal{P}$ converges to one, as the number of samples n increases:

$$\lim_{n\to\infty} p(\text{algorithm finds a solution to } \mathcal{P}) = 1. \tag{2.1}$$

However, among the mentioned methods, only sPRM is asymptotically optimal. Asymptotic optimality refers to the property of a planning algorithm to converge towards an optimal solution, with the increasing number of samples n:

$$\lim_{n\to\infty} c(\tau) = c^*, \tag{2.2}$$

where c is a cost function that admits optimal solution with the finite cost c^* and τ is the best solution found by the planning algorithm.

To this end, new planners PRM* and RRT* [22], with proved asymptotic optimality, were introduced. PRM* is an improved variable-radius variant of sPRM, were neighbourhood (see Line 3 of Algorithm 1) of a sample is defined by an "optimal" radius, dependent on number of sampled configurations n:

$$r(n) = \gamma_{PRM} \left(\frac{\log n}{n}\right)^{\frac{1}{d}}, \tag{2.3}$$

where

$$\gamma_{PRM} > \gamma^*_{PRM} = 2\left(1 + \frac{1}{d}\right)^{\frac{1}{d}} \left(\frac{\mu(\mathcal{C}_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}, \tag{2.4}$$

d is dimensionality of $\mathcal{C}$, $\mu(\mathcal{C}_{\text{free}})$ is the volume of $\mathcal{C}_{\text{free}}$ and ζ_d is the volume of the unit ball in the d-dimensional Euclidean space. Another variant, k-nearest PRM*, based on previously mentioned k-sPRM, can also be considered. Here, k is not a constant but rather a function dependent on n:

$$k(n) = k_{PRM} \log n, \tag{2.5}$$

where

$$k_{PRM} > k^*_{PRM} = e\left(1 + \frac{1}{d}\right). \tag{2.6}$$

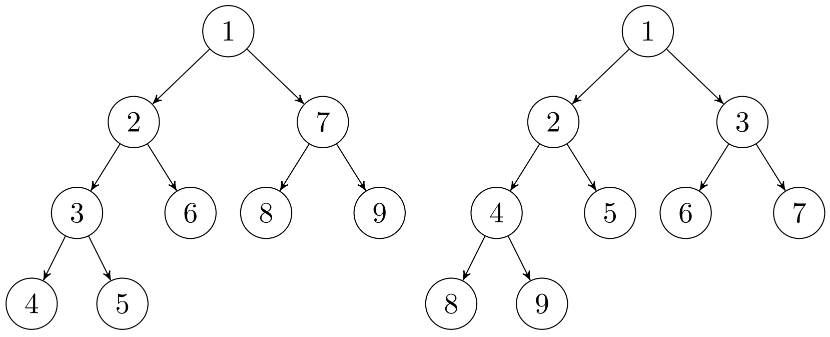
RRT* is an extension of RRT that addresses the optimality issue by introducing a new cost function to the algorithm. The key innovation of RRT* is a method called "rewiring". In traditional RRT, once a node is added to the tree, its connection to the existing tree is fixed. However, in RRT*, nodes are allowed to rewire the tree by checking if there are other nodes that can be reached with a lower cost through the new node. If there is such a node, the tree is rewired to use the new, cheaper path instead. The rewiring step is what allows RRT* to gradually converge to an optimal solution over time, as it enables the algorithm to constantly update the tree with new, shorter paths as they are discovered.

PRM* and RRT* offer notable enhancements in path quality and optimality compared to their initial implementations, PRM and RRT. However, achieving these improvements comes at the cost of increased computational and memory resources. Furthermore, PRM* and RRT* solely optimize a single path, regardless of its homotopy class. As a result, in tasks where optimal solution is not strictly required or when the objective is to find one path from each homotopy class, as is in this thesis, RRT and k-sPRM prove to be more efficient alternatives. Moreover, fast post-processing techniques can be deployed to enhance path quality. For these reasons, proposed method uses Informed-PRM [6], which uses biased sampling of the environment through and ellipsoid subset defined by start and goal configurations, to construct a roadmap representing the planning environment.

2.3 Graph-search Algorithms for Path Planning

Graph search is a problem that involves finding a path between two vertices in a graph. It is often used in problems such as finding a way out of the maze or determining the shortest path between two cities. Two popular graph search algorithms are Depth-first Search (DFS) and Breadth-first Search (BFS).

DFS works on a Last In First Out (LIFO) concept. It explores a graph by visiting the deepest unexplored node first and backtracking when it reaches a dead end. On the other hand, BFS, based on First In First Out (FIFO) concept, explores a graph by visiting all nodes at a given depth level before moving on to nodes at the next depth level. The difference between DFS and BFS is illustrated in Figure 2.1. Since the objective of this thesis is to find multiple homotopically distinct paths, DFS algorithm with augmented visited list is deployed to search for all paths in a roadmap, as described in Section 5.4.



(a) DFS (b) BFS

Figure 2.1: The order in which nodes are expanded.

Time complexity of both DFS and BFS is $\mathcal{O}(V + E)$, where V is number of vertices and E is number of edges. However, the choice between DFS and BFS depends on the specific problem being solved. DFS is faster and requires less memory, but found path is not guaranteed to be the shortest path. BFS will find the shortest path, but only in graphs with uniform edge cost. In weighted graphs, a generalization of BFS called Dijkstra's algorithm [40] is used.

Dijkstra's algorithm works by iteratively expanding the node with the smallest distance from source and updating the distances of its neighbours. This process continues until the shortest path to all nodes has been found. At the end of the algorithm, the distances to all nodes from the source node are known, and the shortest path can be reconstructed by tracing back from the target node to the source node using the recorded predecessors. Dijkstra's algorithm is guaranteed to find the shortest path in a graph with non-negative edge weights, and has a time complexity of $\mathcal{O}(E \log V)$, when implemented using a binary heap, as summarized in Algorithm 3.

Algorithm 3: Dijkstra's Algorithm

Input: $\mathcal{G} = (V, E)$ graph with vertices V and edges E, s source

Out: V set of vertices with known parent and shortest distance from source

```
1  for each $v \in V$ do
2      $v$.value $\leftarrow \infty$
3      $v$.parent $\leftarrow$ None
4  $s$.value $\leftarrow 0$
5  heap $\leftarrow \{V\}$
6  while heap $\neq \emptyset$ do
7      $v \leftarrow$ heap.pop()
8      for each neighbour $n$ of node $v$ do
9          value$_{new} \leftarrow v$.value + cost($v$, $n$)
10         if value$_{new} < n$.value then
11             $n$.value $\leftarrow$ value$_{new}$
12             $n$.parent $\leftarrow v$
```

A* [39] is a graph-search algorithm closely related to Dijkstra's algorithm, but with a added heuristic component to efficiently search for the shortest path between a starting node and a goal node in a weighted graph. As shown in Figure 2.2, A* explores significantly less nodes than Dijkstra's algorithm, making it particularly suitable for large graphs where exploring all nodes is not feasible. However, developing a feasible heuristic function can be challenging, particularly for complex problems.

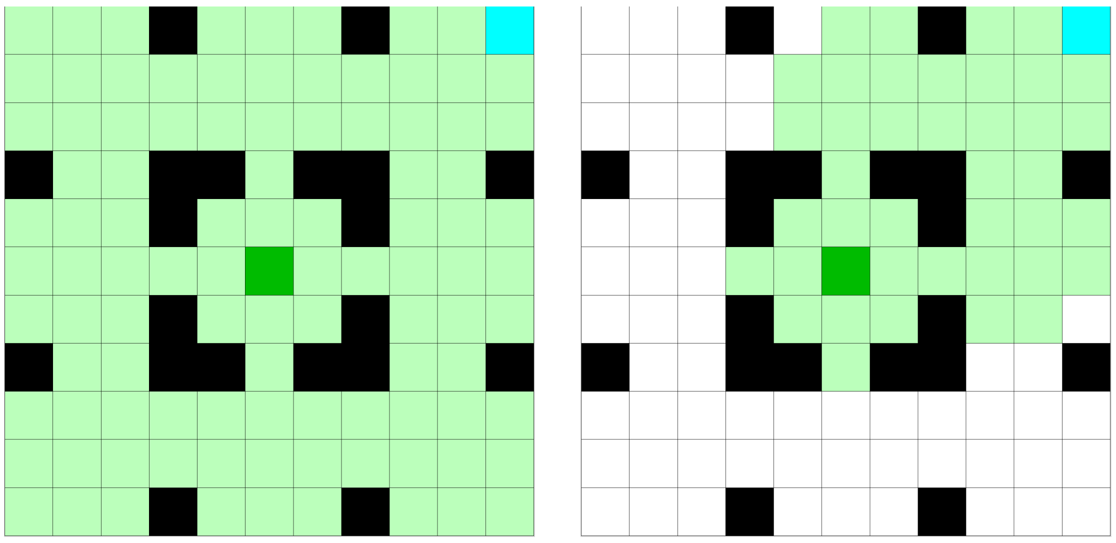

(a) Dijkstra's (b) A*

Figure 2.2: Visualization of search history of Dijkstra's and A* Algorithm. Dark green is the starting node, blue cell is the goal and light green are cells explored by a given algorithm.

In this thesis, graph-search algorithm is used after roadmap construction to ensure existence of a solution for start/goals queries, or determine more samples are required for the query to be solvable. Because multiple waypoint planning can be taken into consideration, Dijkstra's algorithm is used over A*.

2.4 Spanning Tree

In graph theory, a spanning tree $\mathcal{T}$ of an undirected graph $\mathcal{G} = (V, E)$ is a sub-graph that is a tree which includes all of the vertices V of $\mathcal{G}$. Proposed method, in order to cluster a roadmap, divides the roadmap into multiple spanning trees, also called spanning forest, where each cluster is a distinct spanning tree.

2.4.1 Minimum-spanning Tree

A Minimum-spanning Tree (MST) is a spanning tree that connects all the vertices of a weighted undirected graph with the minimum total weight. In graph theory, well-known algorithms for constructing Minimum-spanning Trees are Borůvka's algorithm [29], [44], [45], Prim's algorithm [42], both with time complexities of $\mathcal{O}(E \log V)$, and Kruskal's algorithm [43] with time complexity $\mathcal{O}(E \log E)$, where E is the number of edges and V is the number of vertices in the graph. In the case of disconnected graphs, the mentioned algorithms construct a Minimum-spanning Forest (MSF), which is a collection of disjoint trees, with each tree containing a subset of the graph's vertices. It is worth noting that these algorithms do not require the definition of a root for each tree. However, in this thesis, pre-defined roots are utilized, thereby making a shortest-path tree the appropriate choice.

2.4.2 Shortest-path Tree

Shortest-path tree [40] rooted at a vertex v of a connected, undirected graph $\mathcal{G} = (V, E)$ is a spanning tree $\mathcal{T}$ of $\mathcal{G}$, such that the path distance from root v to any other vertex $u \in V$ in $\mathcal{T}$ is the shortest path distance from v to u in $\mathcal{G}$. This definition also allows creation of a shortest-path forest [38] even in fully connected graphs, by defining a set of root nodes. In shortest-path forest, each vertex v is belongs to a shortest-path tree, that minimizes the length of path from v to the root node. Figure 2.3 visualises the transformation of weighted undirected graph with defined set of roots into a shortest-path forest.

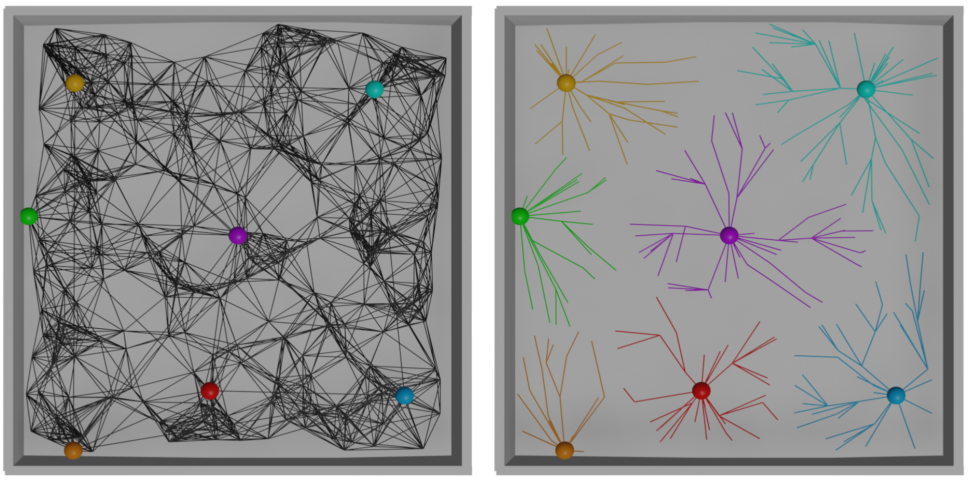

(a) weighted undirected roadmap (b) shortest-path forest

Figure 2.3: Illustration of transformation of weighted undirected graph into shortest-path forest.

The construction of shortest-path tree (or forest) can be implemented using algorithms such as Bellman-Ford algorithm [41] or Dijkstra's. Algorithm 3 essentially constructs a shortest-path tree, by assigning a parent node to each vertex. To generate a shortest-path forest, the algorithm can be adapted by specifying a set of multiple source nodes, rather than just one. In Section 5.2, we present the algorithm employed to partition a roadmap into a shortest-path forest, which subsequently aids in identifying suitable candidates for new centroids (roots).

Chapter 3

Related Work

A complete solution to the problem of finding all homotopically distinct paths in cluttered environments relies on combinatorial motion planning approaches. However, these methods [16], [20] use representations (e.g., Voronoi diagram [35]) that require an explicit representation of occupied space. An optimization-based approach described in [14] proposes to use Gaussian processes to construct a factor graph representing a distribution of multiple trajectories, which are then optimized and filtered. However, the functionality of this method was verified only in 2D. Additionally, some of the trajectories found belong to the same homotopy class. Therefore, the resulting paths must be pruned by identifying homotopically equivalent paths. The method [30] uses a concept of elastic strip, proposed in [32], for trajectory re-planning. Yet, it is only able to represent a single homotopy class and requires an initial feasible trajectory to begin with. Authors in [21] introduce homotopy relation in the form of h-signatures, applicable in both 2D and 3D, but only with time and memory-consuming space discretization. Moreover, the discretized space often fails to capture narrow passages.

To approximate the continuous configuration space, a graph-based representation called roadmap, e.g. PRM [34], is commonly used. Many existing methods for finding distinct homotopy classes [5], [7], [13], [19], [24], [28] take one of the PRM variants as a starting point. The Probabilistic Roadmap algorithm is a widely used sampling-based method for motion planning that consists of two main phases. In the construction phase, the PRM algorithm generates a set of random feasible configurations, also known as samples. These samples are then connected to each other using a local planner.

The original PRM implementation in [34] did not allow cycles in the roadmap, which limited its connectivity, completeness and the ability to capture more than one homotopy class. To address this, [22] introduced a version called sPRM that allows cycles in the graph, and is more widely used nowadays. The author of Informed-PRM [6] proposes to only sample an ellipsoid space between start and goal configurations. Method in [18] aims to generate PRM that guarantees to capture all homotopy classes in an environment, by using an obstacle biased sampler, but relies on explicit representation of occupied space, which is unreasonable for 3D environments.

After PRM is constructed, query phase follows where a path between two samples is found using any standard graph searching algorithm such as Dijkstra's or A*. However, these algorithms only find the shortest path in the roadmap. Method [26] proposes an approach that uses Dijkstra's algorithm to find all paths between start and goal node by finding a path to each node from start and from goal, resulting in total number of paths equal to number of nodes. Method then proceeds to prune any redundant paths, which is an exhaustive process, especially in dense graphs with a high number of nodes, which are necessary in challenging environments that contain multiple narrow passages. Depth-first search algorithm, followed by pruning of redundant paths according to equivalency relation introduced in [23], proposed in [19] is also affected by this issue. For graph search to be efficient, a sparser roadmap, with reduced number of nodes has to be constructed. Authors in [13] propose a method to delete certain edges from a dense roadmap to construct a sparse near-optimal graph. However, even though created graph is sparser, it still contains the same amount of nodes, thus still resulting in high number of redundant paths being found.

Visibility-PRM [31] is a variant of PRM that constructs a sparse roadmap, while discarding some of the nodes as well, resulting in a roadmap more efficient and compact compared to traditional PRM. It does so by introducing a concept of visibility domains. Each visibility domain is defined by a "guard" and covers a space "visible" to the guard. No guards are allowed to be visible to each other, therefore, they are connected through additional samples called "connectors". The method in [28] extends the original Visibility-PRM by allowing creation of cycles but keeping the roadmap simply connected, making the method suitable for distinct path searching. However, it may not capture all homotopy classes, especially in challenging environments containing multiple narrow passages. In this scenario a connector node has to be sampled exactly inside a narrow passage, but only after two guard nodes have already been created in specific locations.

Authors in [24], modify original Visibility-PRM by iteratively adding a limited number of useful cycles. This is done by connecting visible sub-roadmap components, which are subgraphs visible from a new configuration. If there is more than one component, two of them are connected through this new configuration, creating a useful cycle, and therefore a new distinct path. Unfortunately, determining a visible sub-roadmap and its separate components gets progressively more computationally expensive in complex environments.

Algorithm in [7] was designed for fast trajectory re-planning, but includes the subtask of finding homotopically distinct paths. It modifies Visibility-PRM algorithm to make it computationally efficient. This is done by discarding many generated samples, unless there are no guard nodes visible from it, in which case it becomes a new guard, or there are exactly two guard visible, becoming a connector. Additionally, if there already exists a connection between a given pair of guards with the same homotopy class, the longer one is removed. This method achieves great results in scenarios with good visibility, where long edges can be created. However is very susceptible to initial placement of new guard nodes in scenarios where visibility is limited, which affects both computational speed and functionality.

Method in [5] tackles minimum-time trajectory planning problem, but contains a unique solution for finding multiple paths with distinct homotopy classes. Algorithm starts by constructing Informed-PRM [6] which is then iteratively searched for the shortest path using Dijkstra's algorithm. For each path, the algorithm identifies a node with the smallest clearance from obstacles, and removes region around it from the roadmap. This process is repeated until no new path can be found. To address cases where multiple distinct paths pass through a deleted region, algorithm is recursively called from the start to the deleted regions and from the deleted regions to the goal. Limitation of this method is lack of information required to optimally select a region to remove, failing to find some of the paths as a result. Additionally, recursion may lead to combinatorial explosion, drastically increasing run-time.

The main limitation of existing methods is their inconsistency across different environments, leading to significant variations in their performance, especially in challenging environments that contain multiple narrow passages. CTopPRM is capable of efficiently reducing a dense roadmap, required to accurately represent such environment, by dividing it into clusters. This significantly reduces both number of edges and nodes, while maintaining the same connectivity of free space, as the initial roadmap. This allows it to both effectively and consistently identify a large number of homotopically distinct paths, even in challenging environments.

Chapter 4

Problem statement

The goal of this thesis is to tackle the problem of efficiently finding multiple homotopically distinct paths, e.g. paths between same endpoints going around an obstacle from different sides. By identifying multiple such paths, the robot has greater flexibility when planning its movements.

We study the path planning problem [25], in which we are given an object and search for a way for it to move through an environment without colliding with obstacles. Manipulated object, also called agent or robot, is denoted by $\mathcal{A}$. It operates in a space referred to as the world $\mathcal{W}$, usually a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. The obstacles within the planning environment are denoted by $\mathcal{O} \subseteq \mathcal{W}$.

The robots's position in the world is determined by configuration q. Configuration is a vector that represents a specific arrangement of the robot's degrees of freedom $q = (q_1, q_2, \dots, q_n)$. The set of all possible robot configurations is called configuration space $\mathcal{C}$, which in this thesis, we consider to be $\mathbb{R}^3$ with elements $q = (x, y, z)$. We also define two configurations, start position q_{start} and goal position q_{goal}. The set of all points occupied by the robot $\mathcal{A}$ in configuration q is denoted by $\mathcal{A}(q)$ and the set of all configurations q where $\mathcal{A}(q)$ intersects with the obstacles $\mathcal{O}$ is called occupied space $\mathcal{C}_{\text{obs}} \subseteq \mathcal{C}$

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}. \tag{4.1}$$

Conversely, the free space $\mathcal{C}_{\text{free}} = \mathcal{C} \backslash \mathcal{C}_{\text{obs}}$ is the set containing all non-colliding configurations.

A continuous curve that connects two configurations q_{start} and q_{goal} is called a path $\pi(s) \in \mathcal{C}, s \in [0, 1]$. A collision-free path is a path that is entirely contained within the free space $\mathcal{C}_{\text{free}}$

$$\pi : s \in [0, 1] \to \pi(s) \in \mathcal{C}_{\text{free}} \forall s \in [0, 1], \tag{4.2}$$

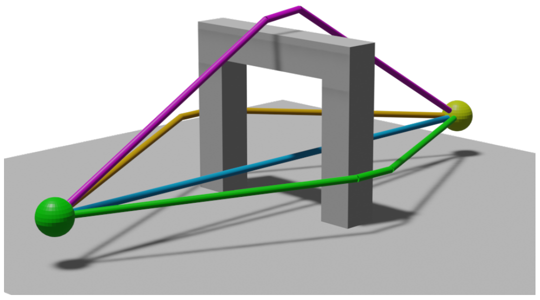
where $\pi(0) = q_{start}$ and $\pi(1) = q_{goal}$. A trajectory is a time-parametrized path that describes the motion of a moving object over a period of time, taking into account the robot's dynamics. Specifically, given a path $\pi(s)$ connecting the start configuration q_{start} and the goal configuration q_{goal}, a trajectory is a function

$$\tau : s \in [0, 1] \times t \in [0, T] \to \mathcal{C}, \tag{4.3}$$

that maps time to a point in the configuration space $\mathcal{C}$, where T is the time horizon. The trajectory $\tau(t)$ must satisfy constraints such as collision avoidance, feasibility as well as constrained posed by robot dynamics, and must follow the path $\pi(s)$ such that $\tau(0) = q_{start}$ and $\tau(T) = q_{goal}$.

To mutually differentiate between topological paths or trajectories, a topology equivalence relation has to be defined.

4.1 Topology Equivalence Relation


(a) As per the definition of homotopy, green, pink, and orange paths are considered to be equivalent. However, it is important to acknowledge that they embody considerably different motions.

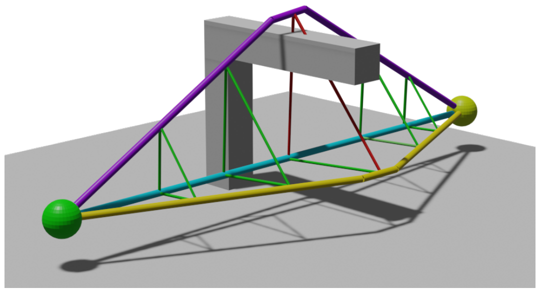

(b) Illustration of UVD equivalency check. Blue and yellow paths are determined to be equivalent, while the purple path belongs to a distinct UVD class. By the classical definition of homotopy, all three paths are equivalent

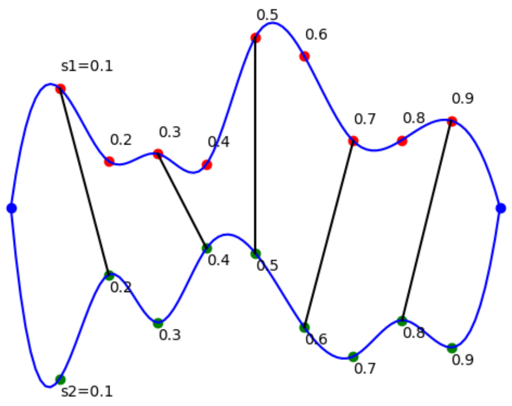
Figure 4.1: Topology equivalence relation.

A homotopy class is commonly used to define a set of topologically equivalent paths. In [27], homotopy is defined as a concept in topology where two continuous functions from one topological space to another are considered homotopic if one can be smoothly deformed into the other. This definition, applied on paths, was summarized in [24]. Homotopy of two paths $\pi(s)$ and $\pi'(s)$ in $\mathcal{C}$ is said to exist if there is a continuous map $h : [0,1] \times [0,1] \rightarrow \mathcal{C}_{\text{free}}$ such that $h(s,0) = \pi(s)$, $h(s,1) = \pi'(s)$ for all $s \in [0,1]$, and $h(0,t) = h(0,0)$ and $h(1,t) = h(1,0)$ for all $t \in [0,1]$.

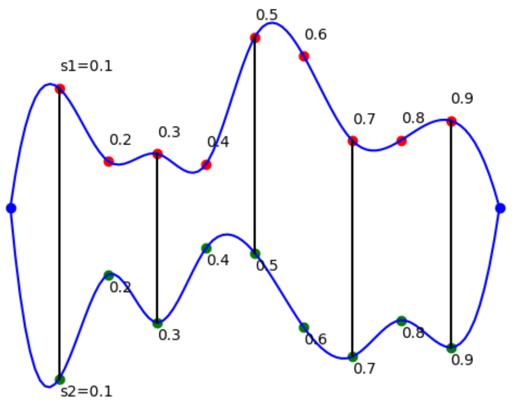
While homotopy is a widely used concept, it has been found to be inadequate for capturing a sufficient number of useful paths in $\mathbb{R}^3$ space, as shown in Figure 4.1a. To address this limitation, authors in [24] introduced the concept of Visibilty Deformation (VD) which captures more useful paths. Unlike homotopy, which allows for complex, high-dimensional transformations to be applied during path deformation, visibility deformation focuses on preserving certain visibility-related properties of the path, such as the ability to evade obstacles, effectively reducing dimensionality of deformation between the paths. However, the approach is still computationally expensive. Therefore, [7] proposes an extension to VD called Uniform Visibilty Deformation (UVD), which is more efficient.

Definition 1 *Two paths $\pi(s)$, $\pi'(s)$ parameterized by $s \in [0,1]$ and satisfying $\pi(0) = \pi'(0)$, $\pi(1) = \pi'(1)$, belong to the same uniform visibility deformation class, if for all s, line-segment from $\pi(s)$ to $\pi'(s)$ is collision-free.*

Three paths belonging to two distinct UVD classes, and an illustration of UVD equivalency check is depicted in Figure 4.1b, while Figure 4.2 shows the difference between VD and UVD. Both define a continuous map between two paths $\pi(s)$ and $\pi'(s)$ in which a point $\pi(s_1)$ is transformed to a point $\pi'(s_2)$ through a straight line-segment, where $s_1, s_2 \in [0,1]$. However, while in UVD $s_1 = s_2$, this is not necessarily true for VD. Because of this, UVD is less general and captures more UVD classes in practice. Additionally, it performs better when the paths $\pi(s)$ and $\pi'(s)$ are of similar length, and its performance deteriorates as the difference in length increases. On the other hand, when it comes to equivalency checking, UVD exhibits significantly higher computational efficiency compared to the VD.



(a) visibility deformation



(b) uniform visibility deformation

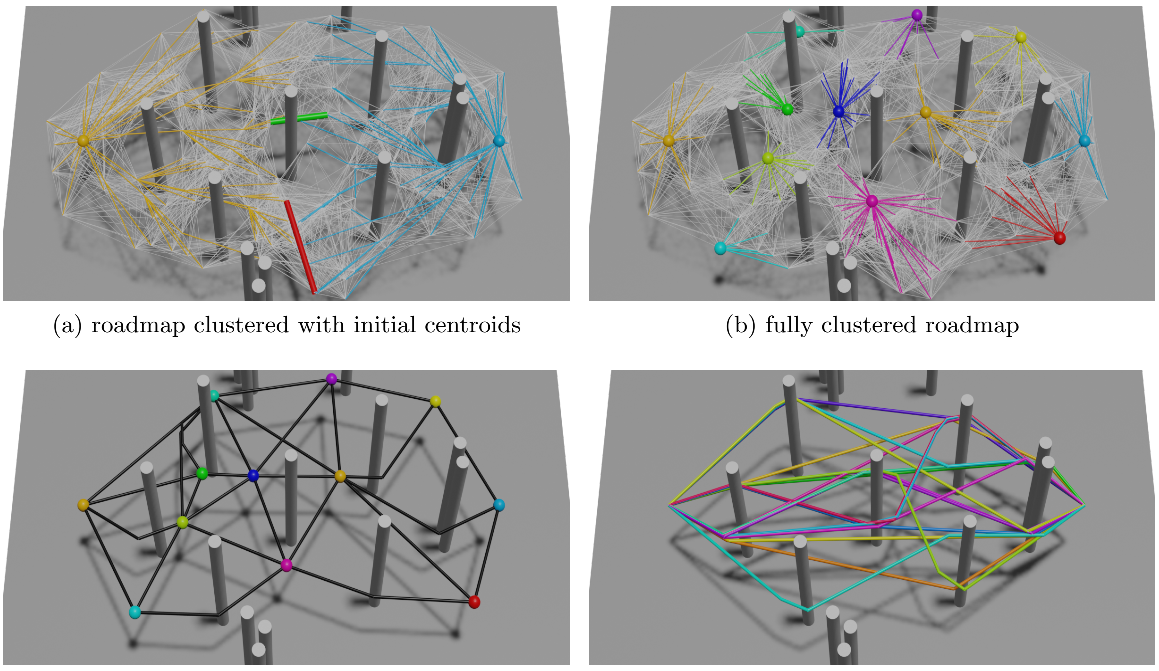
Figure 4.2: Difference between VD (a) and UVD (b). Each red point on one path is transformed to a green point on the other path through a straight-line. Each pair of points correspond to the same parameter s in UVD, but not necessarily in VD, where corresponding pair of points is found through a more time-consuming process described in [24].

The aim of this thesis it to propose a method which would allow fast and efficient search for a set Π of topological paths each representing a distinct UVD class. To accomplish this, a combination of topological and motion planning methods, along with graph-based search algorithms, is utilized to systematically transform a densely sampled Probabilistic Roadmap (PRM) into a sparser, low-order graph. Smaller graph should enable a more efficient search for multiple paths, and reduce the number of redundant paths found, i.e. paths belonging to the same UVD class. By reducing the size of the graph, the search for multiple paths becomes more efficient and minimizes the number of redundant paths discovered, i.e. paths belonging to the same UVD class.

Chapter 5

Clustering Topological PRM

The proposed method we named Clustering Topological PRM (CTopPRM) finds distinct topological paths using a hierarchical approach that starts by constructing a dense roadmap using Informed-PRM [6]. The nodes in the roadmap are divided into two initial clusters (that are defined by q_{start} and q_{goal}), and more clusters are iteratively identified. In each iteration, new cluster centroid is created between two neighbouring clusters. Cluster centroids are then used as vertices of new sparse roadmap, which is then searched for paths with diverse uniform visibility deformation classes. Finally, found paths are shortened and filtered. The algorithm uses Euclidean Signed Distance Field (ESDF) for collision checking. The method is summarized in Algorithm 4 and its visualization is shown in Figure 5.1. Each step of the CTopPRM algorithm is explained in more detail in the following sections.



(a) roadmap clustered with initial centroids

(b) fully clustered roadmap

(c) connections between clusters

(d) shortened distinct paths

Figure 5.1: Visualization of individual stages of the algorithm. Generated PRM is first divided into two clusters with q_{start} and q_{goal} as centroids (a). Minimum and maximum connections, marked by green and red line, are compared and new centroids are iteratively created at one of the maximum connections until roadmap is fully clustered (b). Connections between cluster centroids are found and a low-order, sparse graph (c) is constructed and then searched for distinct paths (d).

Algorithm 4 starts with **Informed-PRM** constructing a roadmap $\mathcal{G}$=(V, E) with vertices V and edges E, which is then divided into clusters by **clusterGraph** method, transforming the roadmap into a shortest-path forest (V, E_C). Method **addCentroids** iteratively adds new centroids and ultimately constructs a low-order graph (C_V, C_E). This graph is then searched, by **findDistinctPaths** method, for a set of paths Π^d, which are then filtered by **filterPaths** method, resulting in the set Π, where each path $\pi \in \Pi$ represents a distinct UVD class.

Algorithm 4: CTopPRM

```
  Input: $q_{start}$, $q_{goal}$
  Global params.: M max clusters, $\kappa_p$ DFS termination condition, $\kappa_s$ pruning
                  parameter
  Out: $\Pi = (\pi_1, \pi_2, ..., \pi_n)$ found topological paths
1 $(V, E), l \leftarrow$ Informed-PRM$(q_{start}, q_{goal})$ // [6]
2 $C_V \leftarrow \{q_{start}, q_{goal}\}$
3 $(V, E_C) \leftarrow$ clusterGraph$((V, E), C_V)$ // Alg. 5
4 $(C_V, C_E) \leftarrow$ addCentroids$((V, E_C), C_V)$ // Alg. 6
5 $\Pi^d \leftarrow$ findDistinctPaths$(C_V, C_E, \kappa_p \cdot l)$
6 $\Pi \leftarrow$ filterPaths$(\Pi^d)$
```

5.1 Dense Probabilistic Roadmap Construction

The goal of this step of the CTopPRM algorithm is to densely represent free-space $\mathcal{C}_{\text{free}}$ which is realized using **Informed-PRM** [6]. Each vertex is connected to its 14 neighbours using a straight-line, if possible. The shortest path in the constructed roadmap is then found using Dijkstra; let l denote its length. Created dense roadmap accurately represents connectivity of $\mathcal{C}_{\text{free}}$, but is inefficient for multi-path searching, due to a high number of both vertices and edges. As a result, search in the dense roadmap would not only require large computational resources, but also return a high number of redundant paths, belonging to the same UVD class. Therefore, we aim to greatly reduce number of nodes and edges, while keeping the accurate representation of free-space connectivity.

5.2 Graph Clustering

The **clusterGraph** method described in Algorithm 5 divides roadmap into clusters with shortest-path tree [40] structure, with each cluster centroid being root of its shortest-path tree. All clusters together form a shortest-path forest [38].

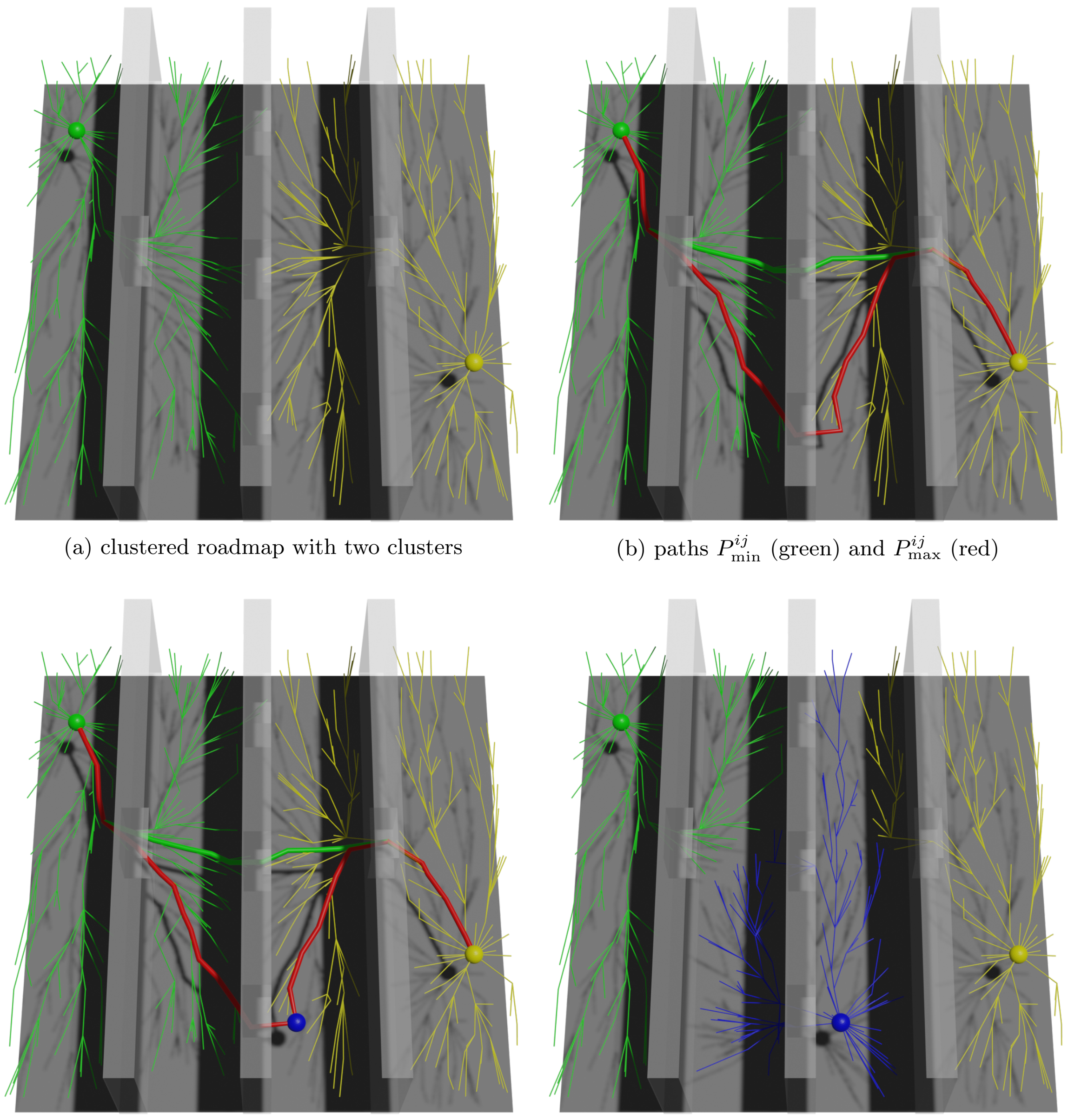
The division of the roadmap into clusters is implemented using a min-heap, making the time complexity $\mathcal{O}(E \log V)$. Initial cluster centroids are q_{start} and q_{goal}. Each shortest-path tree (cluster) is expanded from its centroid, creating connections to minimize the total cost of a path from each vertex to the nearest cluster centroid. This result is achieved using Lines 12-16 of Algorithm 5. If two neighbouring vertices belong to different clusters, total cost of path connecting two cluster centroids over these two vertices is calculated, as shown in Line 18. For each pair of neighbouring clusters i and j, the method maintains paths $P^{ij}_{\min}$ and $P^{ij}_{\max}$ that represent the shortest and the longest paths connecting the two clusters, respectively, with a prospect they might represent different UVD classes. Edges that do not belong to either cluster, but are a part of these paths are called minimum and maximum cluster connection, and they are crucial for selecting new cluster centroids in the steps to follow. An example of these connections is shown in Figure 5.1a, where green line represents minimum and red line represents maximum cluster connection.

Algorithm 5: clusterGraph

```
Input: ($V$, $E$) roadmap, $C_V$ set of cluster centroids
Out: ($V$, $E_C$) clustered roadmap
1  for each $v \in V$ do
2      v.value ← $\infty$
3      v.cluster ← -1
4  cluster_id ← 0
5  for each $c \in C_V$ do
6      c.value ← 0
7      c.cluster ← cluster_id, cluster_id ← cluster_id + 1
8  heap ← $V \cup C_V$
9  while heap $\neq \emptyset$ do
10     v ← heap.pop()
11     for each neighbour n of node v do
12         value$_{new}$ ← v.value + cost(v, n)
13         if value$_{new}$ < n.value then
14             n.value ← value$_{new}$
15             n.cluster ← v.cluster
16             n.parent ← v
17         else if v.cluster $\neq$ n.cluster AND n.cluster $\neq$ 1 then
18             cost$_{new}$ ← v.value + cost(v, n) + n.value
19             update $P^{ij}_{\min}$ and $P^{ij}_{\max}$ if necessary
```

5.3 Adding New Centroids

The motivation behind division of PRM into multiple clusters is to create an easily searchable graph with cluster centroids as vertices which will have significantly lower order than the inital densely sampled roadmap. To capture all UVD classes, while minimizing order of the graph, vertices have to be placed methodically. CTopPRM's approach to create new cluster centroids is depicted in Algorithm 6 and visualized in Figure 5.2.



(a) clustered roadmap with two clusters

(b) paths P^{ij}_{min} (green) and P^{ij}_{max} (red)

(c) new centroid

(d) re-clustered roadmap with three clusters

Figure 5.2: Visualization of new centroid selection process. In (a) are the clusters at the start of an iteration. Paths P^{ij}_{min} (green) and P^{ij}_{max} (red) are found for each pair of neighbouring clusters (b) and new centroid is created at a non-deformable connection with highest ratio of lengths of paths P^{ij}_{min} and P^{ij}_{max} (c). Roadmap is then re-clustered (d).

It starts by comparing connections $P^{ij}_{\min}$ and $P^{ij}_{\max}$ for each pair of connected clusters (i, j). In Figure 5.2b, $P^{ij}_{\min}$ is colored green and $P^{ij}_{\max}$ is red. The method **Deformable** in Line 6 of Algorithm 6 then checks if these two paths belong to the same UVD class. Each path is discretized to $n = \lceil P^{ij}_{\max}.length/\Delta d \rceil$ steps. Each line-segment between the points $P^{ij}_{\max}[k]$ and $P^{ij}_{\min}[k]$, $k = 0, \ldots, n$, is tested for collisions with the resolution Δd. If they are not deformable, creating a new centroid at the border of these two clusters is beneficial in capturing more UVD classes, as there are two distinct paths connecting two existing cluster centroids. The ratio of their lengths is then calculated and saved. The connection with the highest ratio is selected and one of the two neighbouring nodes belonging to its maximum connection, is determined as a new centroid (colored blue in Figure 5.2c), and roadmap is then clustered again, using the method described in Section 5.2. The clustering ends if $P^{ij}_{\min}$ and $P^{ij}_{\max}$ are deformable into each other for all neighbouring clusters, or a predefined maximum number of clusters M is reached. The fully clustered roadmap is shown in Figure 5.1b.

Algorithm 6: addCentroids

```
Input: $(V, E_C)$ clustered roadmap, $C_V$ set of cluster centroids
Global params.: $\Delta d$ collision-detection resolution, M max clusters
Out: $(C_V, C_E)$ simplified roadmap
1  can_add ← true
2  while can_add AND $num\_clusters < M$ do
3      can_add ← false
4      ratio ← ∅
5      for each (i, j) ∈ connected_clusters do
6          if not deformable($P^{ij}_{\min}$, $P^{ij}_{\max}$, $\Delta d$) then
7              ratio ← ratio ∪{ $\frac{P^{ij}_{\max}.\text{length}}{P^{ij}_{\min}.\text{length}}$ }
8              can_add ← true
9      new_centroid ← getCentroid(max(ratio))
10     $C_V \leftarrow C_V \cup$ new_centroid
11     $(V_C, E_C)$ ← ClusterGraph((V, E), $C_V$)
12 $C_E \leftarrow \emptyset$
13 for each (i, j) ∈ connected_clusters do
14     $E_{new}$ ← shorten($P^{ij}_{\min}$)
15     $C_E \leftarrow C_E \cup E_{new}$
```

After clustering ends, new simple low-order graph is created with cluster centroids as vertices. Edges connecting them are found by a sequence shown in Lines 12-15. For each pair of connected clusters, an edge connecting the corresponding pair of clusters is created from saved path, $P^{ij}_{\min}$. Each of these edges are then shortened using a method similar to shortening in [7] and [5], further explained in Section 5.5. As a result, these edges together with a set of cluster centroids finalize the construction of the simple low-order graph, which can be seen in Figure 5.1c. Entire roadmap reduction process is illustrated in Figure 5.3.

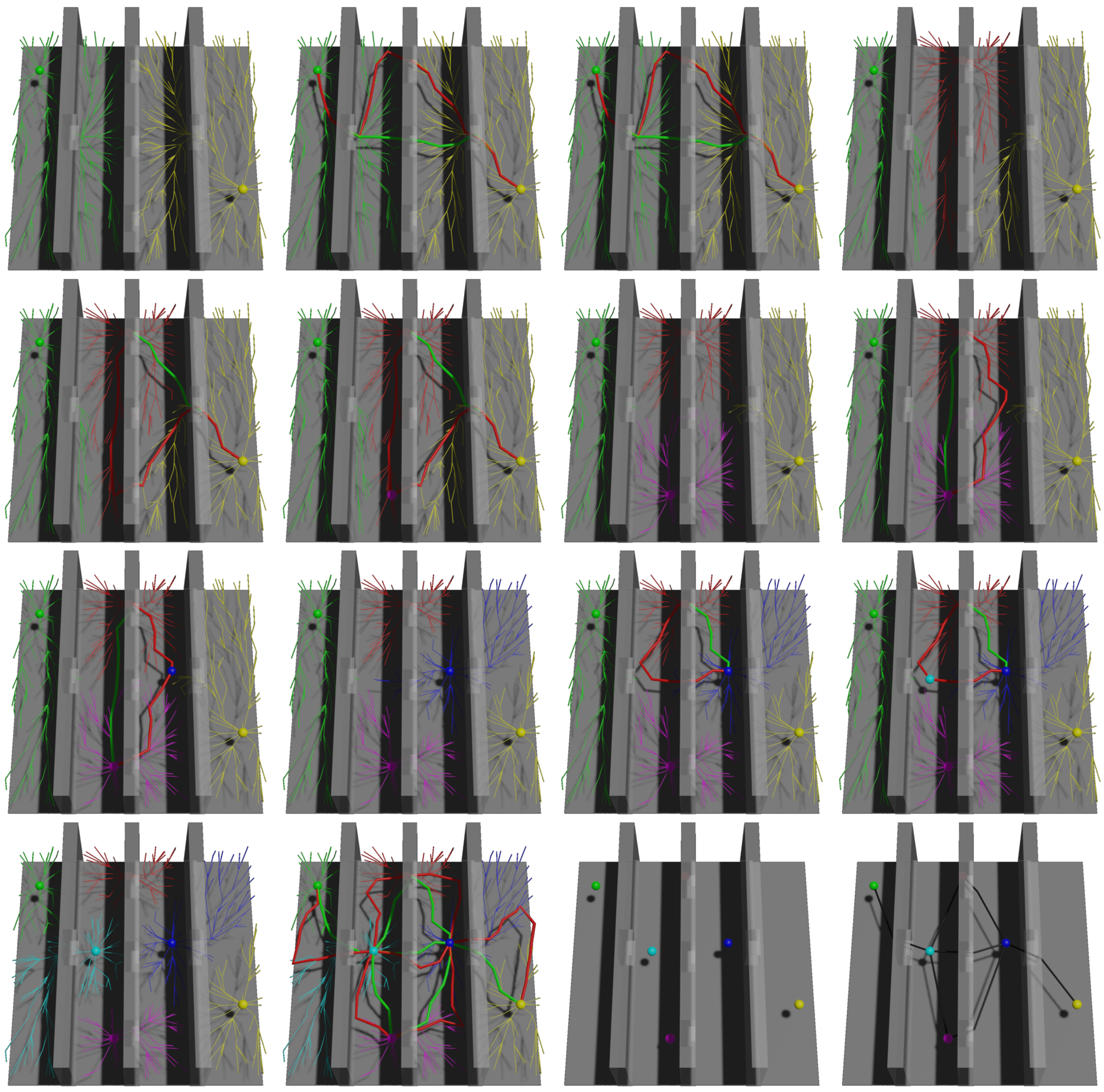


Figure 5.3: Illustration of the roadmap reduction process through iterative clustering. New centroids are created at path P^{ij}_{max} between neighbouring clusters (i, j) for which P^{ij}_{min} and P^{ij}_{max} are not deformable and have the largest length ratio, up until all connections are deformable. A reduced graph is then constructed with cluster centroids as vertices and shortened P^{ij}_{min} as edges.

5.4 Multi-path Search

The method **findDistinctPaths** in Line 5 of Algorithm 4 searches the graph using Depth-first search (DFS) algorithm with visited list, similar to [16]. In this depth-limited DFS, the expansion on the current node is terminated if the current path length is greater than κ_p times the length of best solution l. This way, parameter κ_p directly controls the trade-off between the number of paths found and computational time. By lowering κ_p, long and convoluted paths are not explored further, preventing potential combinatorial explosion, as well as reducing the run-time of both DFS and the following filtering process, due to the reduced number of paths found. Contrarily, increasing the value of κ_p may result in increased number of identified paths, but with a corresponding increase in computational time. Method **findDistinctPaths** is summarized in Algorithm 7.

Algorithm 7: findDistinctPaths

```
Input: $(C_V, C_E)$ simplified roadmap, $q_{start}$, $q_{goal}$, $l$ length of shortest path
Global params.: $\kappa_p$ DFS termination condition
Out: $\Pi^d = (\pi^d_1, \pi^d_2, ..., \pi^d_n)$ found distinct paths
1  $\Pi^d \leftarrow \emptyset$
2  $p^d \leftarrow \{q_{start}.\text{cluster}\}$
3  $visited \leftarrow \{q_{start}.\text{cluster}\}$
4  Function findPathsRecurse($p^d, visited$):
5      if $p^d$.length() $> l \cdot \kappa_p$ then
6          return
7      last_id $\leftarrow p^d$.last()
8      for each node_id $\in$ connected_clusters(last_id) do
9          if node_id $==$ $q_{goal}$.cluster then
10             $p^d$.push_back(node_id)
11             $\Pi^d \leftarrow \Pi^d \cup p^d$
12             return
13         if node_id $\notin visited$ then
14             $visited \leftarrow visited \cup$ node_id
15             $p^d$.push_back(node_id)
16             findPathsRecurse($p^d, visited$)
```

5.5 Path Filtering

To accommodate future applications, e.g. planning high-speed trajectories for Unmanned Aerial Vehicles (UAVs) along the paths, CTopPRM uses a series of methods, included in **filterPaths** function in Line 6 of Algorithm 4, summarized in Algorithm 8, to augment and filter found paths. Method similar to shortening in [7] and [5] is first used to shorten all found paths. Method, illustrated in Figure 5.4 and depicted in Lines 2-12 of Algorithm 8, begins by discretizing each path $\pi_k^d \in \Pi^d$ found in the previous step (Section 5.4), using a resolution of Δd, into a set of waypoints P^d. An UVD equivalent shortcut P^s is then initialized with the first waypoint in P^d. For the first waypoint $p^d \in P^d$ that cannot be seen from the last point $p^s \in P^s$, the algorithm searches for a voxel obstructing the view, and moves it away from obstacles in the direction orthogonal to the line segment l^d, while remaining co-planar to both l^d and the ESDF gradient at the obstructing voxel. Point pushed away from the obstacle is added to P^s and process is repeated until P^d and P^s connect the same two endpoints.

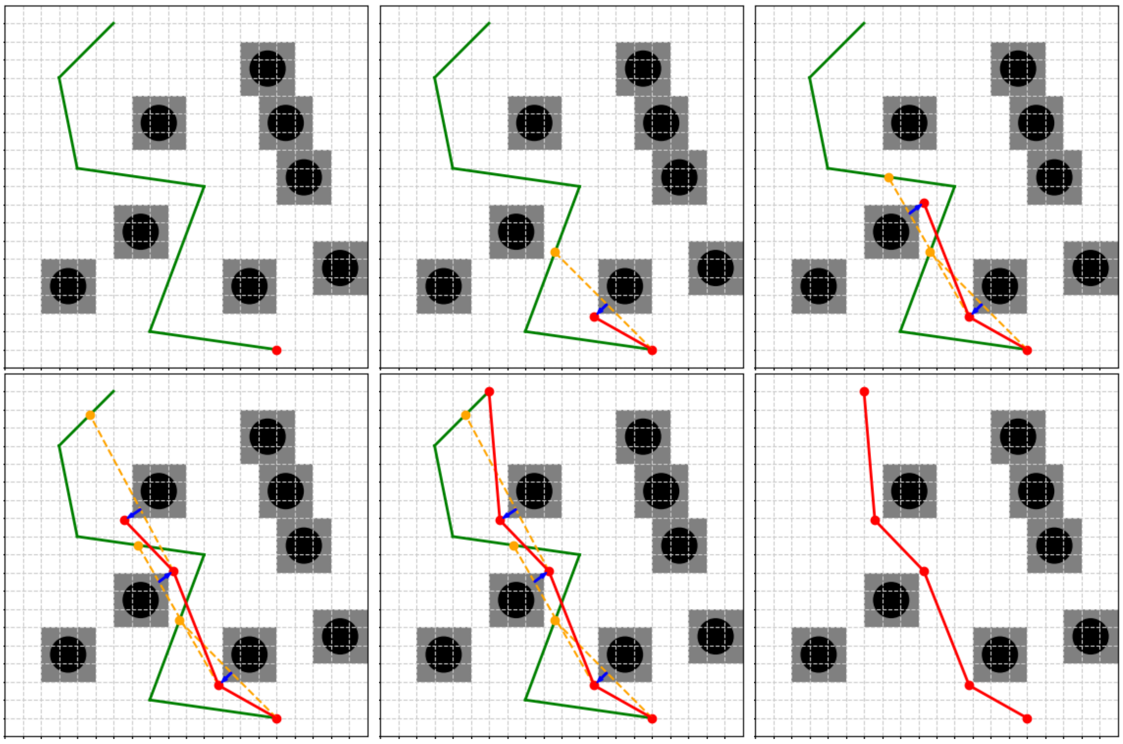


Figure 5.4: Illustration of the shortening process. Obstacles are colored black and occupied cells are grey. Detoured path (green) found by **findDistinctPaths** method is shortened to the red path. Yellow lines are not collision-free, thus the voxel obstructing the view has to be pushed away from obstacles (blue arrow), resulting in a new waypoint (red) belonging to the shortened path.

Any shortened paths longer than a threshold defined as length of the shortest found path multiplied by the parameter κ_s are then pruned away (see Lines 13-16 of Algorithm 8). Doing this filters out any paths that either take a unreasonably long detour or include suboptimal movement, e.g. looping around obstacles. Finally, one last UVD equivalency check is performed on shortened paths to filter out any paths belonging to the same UVD class, as depicted in Lines 17-20 of Algorithm 8.

Algorithm 8: filterPaths

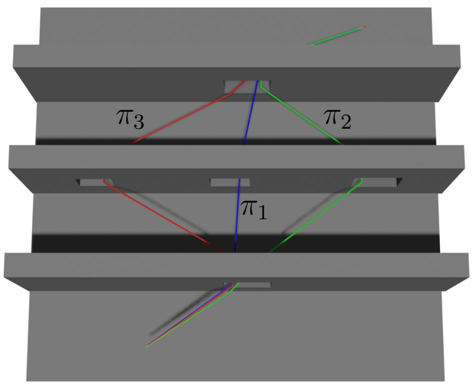
```
Input: $\Pi^d$ distinct paths found by findDistinctPaths method
Global params.: $\Delta d$ collision-detection resolution, $\kappa_s$ pruning parameter, clearance
Out: $\Pi$ set of shortened paths each representing different UVD class
1  $\Pi^s \leftarrow \emptyset$
2  for each $\pi^d \in \Pi^d$ do                                        shorten paths
3      $P^d \leftarrow$ discretize($\pi^d$, $\Delta d$)
4      $P^s \leftarrow \{P^d$.first()$\}$
5      for each $p^d \in P^d$ do
6          $l^d \leftarrow$ Line($P^s$.last(), $p^d$)
7          if not collision-free($l^d$, $\Delta d$) then
8              $p^b \leftarrow$ collisionPoint($l^d$)
9              $p^s \leftarrow$ pushFromObstacle($p^b$, clearance)
10             $P^s$.push_back($p^s$)
11     $P^s$.push_back($P^d$.last())
12     $\Pi^s \leftarrow \Pi^s \cup P^s$
13 length$_{\min} \leftarrow$ shortest($\Pi^s$)
14 for each $\pi^s \in \Pi^s$ do                                        remove too long paths
15     if $\pi^s$.length $> \kappa_s \cdot$ length$_{\min}$ then
16         $\Pi^s$.remove($\pi^s$)
17 for each $(\pi_i, \pi_j) \in \Pi^s$ do                               remove equivalent paths
18     if deformable($\pi_i, \pi_j$, $\Delta d$) then
19         $\Pi^s$.remove(longest($\pi_i, \pi_j$))
20 $\Pi \leftarrow \Pi^s$
```

The final output of **filterPaths** method, and therefore of the entire CTopPRM algorithm (Algorithm 4), is a set of paths Π, where each $\pi_i \in \Pi$ represents a different UVD class.

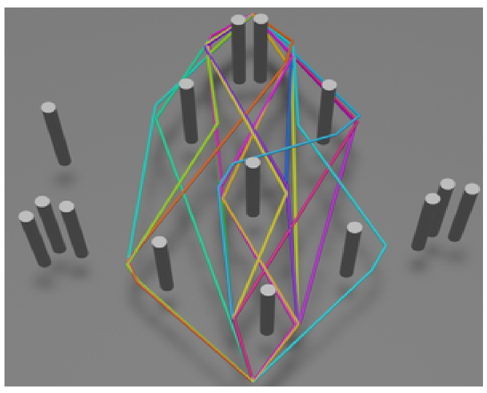
Chapter 6

Results

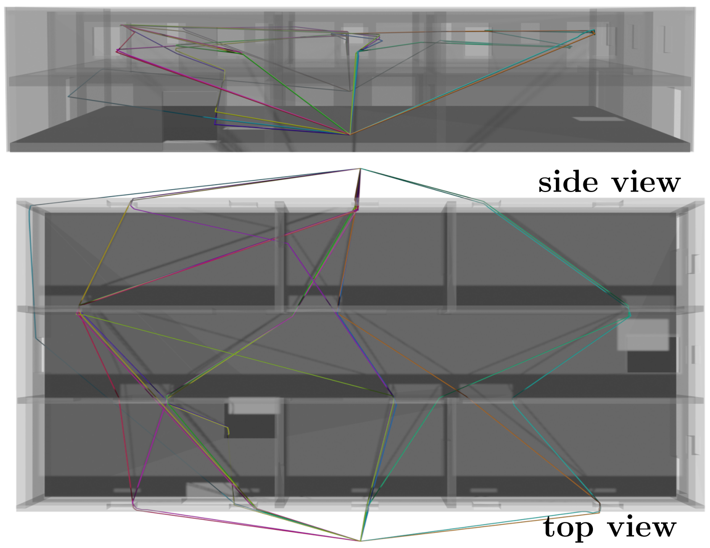
The performance of the CTopPRM is evaluated in three different environments shown in Figure 6.1. The purpose of having multiple thematically different environments is to evaluate robustness of our and related methods. The most important evaluation metric is the number of UVD classes each method is able to find in form of a path within such a UVD class. We also consider computational time and quality of found paths, represented by their respective lengths.



(a) windows in 1-3-1 scenario (proposed in [11])



(b) poles



(c) building

Figure 6.1: Maps of the environments used for evaluating CTopPRM and related algorithms with paths found by CTopPRM.

CTopPRM is implemented in C++, and experiments are run on AMD Ryzen 7 6800HS CPU. Values of parameters used in the test is shown in Table 6.1.

Table 6.1: Algorithm parameters & map size

		windows	**poles**	**building**
	map size	27x26.7x8	10x10x2.8	30x20x6.3
All methods	clearance	0.3	0.3	0.2
	Δd	0.1	0.2	0.2
	PRM samples	500	300	1000
	κ_s	1.5	1.2	1.5
CTopPRM	M	9	20	20
	κ_p	1.8	1.6	1.7

The computational time of all methods examined in this study is primarily influenced by two key parameters: Δd — the resolution of collision detection, and the number of samples used to build the dense roadmap (PRM samples). The size of the spherical robot is specified by the clearance parameter. Notably, the CTopPRM algorithm includes two exclusive parameters: M (maximum number of clusters) and κ_p (DFS termination condition). Both parameters affect the trade-off in performance of the method, reducing computational time, but at a cost of quality and quantity of found paths. Pruning parameter κ_s is used to remove any long and convoluted paths from the final output of the CTopPRM method.

We evaluate the performance of CTopPRM by comparing it with three other methods that tackle the same challenge. These include the method we call RAPTOR from [7], which solves the sub-task of identifying distinct topological paths, the Distinct Path Search algorithm proposed in [5], referred to as B. spheres, and an approach based on [24] called P-D-PRM, adapted slightly to ensure reasonable run-time.

6.1 Windows Environment

The first set of experiments is performed in environments called windows which contain a small number of narrow passages (windows) placed on one to three parallel walls, making maximum number of distinct UVD classes, Ground Truth (GT), easily determinable. Name of the maps in Tables 6.2 and 6.3 indicates number of windows on each wall. For example, scenario 1-3-1 shown in Figure 6.1a contains one window on the first wall, three on the second and one on the third. Zero indicates a given wall is missing completely and 's' for 'side' indicates that a specific window is not in the middle of the wall. These scenarios are tested in $\mathbb{R}^2$ space with a circular robot. Each algorithm is evaluated on 100 runs in every scenario. We report computational time (c.t.) and success rate of each algorithm in finding every single distinct path that exists in a given scenario.

The results of this experiment are shown in Tables 6.2 and 6.3 as well as Figures 6.2-6.9. They indicate that CTopPRM algorithm manages to find all but one path, across all testing scenarios, with highest success rate. Additionally, CTopPRM finds most paths with a success rate close to 100%, with lowest success rate being 70%, proving its effectiveness and reliability, both absolutely and relatively to other methods.

All the methods performed better in simpler scenarios where only one window is required to be passed (0-2-0 and 0-3-0 shown in Figures 6.2 and 6.6), and their performance deteriorates as the number of narrow passages increases. Visibility-based methods P-D-PRM and RAPTOR demonstrate the most significant decline in performance, with success rates dropping below 10% for multiple paths in different scenarios. Most notably in scenario 1-2-1, shown in Figure 6.4, where these methods fail to detect any of two paths in over 90% runs.

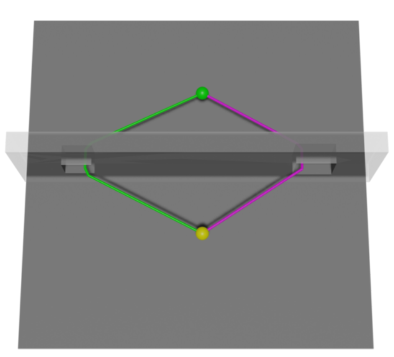
Table 6.2: Success rate of finding each distinct path $\pi_i \in \Pi_{\text{env}}$ in scenarios where **GT=2**

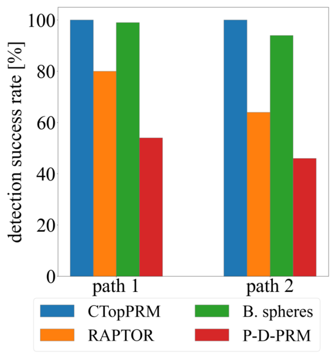
		scenario			
		0-2-0	1-2-0	1-2-1	1s-2-1s
CTopPRM	c.t.[ms]	41	54	51	48
	π_1[%]	**100**	**99**	**99**	**98**
	π_2[%]	**100**	**99**	**99**	**97**
RAPTOR [7]	c.t.[ms]	75	65	**35**	42
	π_1[%]	80	51	4	40
	π_2[%]	64	42	7	26
B. spheres [5]	c.t.[ms]	**36**	55	51	86
	π_1[%]	99	89	95	87
	π_2[%]	94	94	58	91
P-D-PRM [24]	c.t.[ms]	39	**52**	43	**40**
	π_1[%]	54	35	9	42
	π_2[%]	46	24	8	9

Table 6.3: Success rate of finding each distinct path $\pi_i \in \Pi_{\text{env}}$ in scenarios where **GT=3**

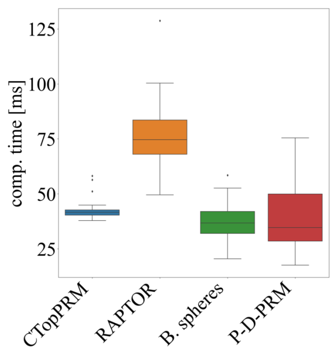
		scenario			
		0-3-0	1-3-0	1-3-1	1s-3-1s
CTopPRM	c.t.[ms]	70	51	47	54
	π_1[%]	94	**100**	**100**	**100**
	π_2[%]	**100**	**71**	**89**	**99**
	π_3[%]	**96**	**70**	**89**	**81**
RAPTOR [7]	c.t.[ms]	148	101	43	46
	π_1[%]	94	**100**	41	86
	π_2[%]	**100**	0	12	16
	π_3[%]	92	0	10	14
B. spheres [5]	c.t.[ms]	53	41	51	85
	π_1[%]	**98**	95	90	88
	π_2[%]	98	2	78	89
	π_3[%]	93	3	52	68
P-D-PRM [24]	c.t.[ms]	**49**	**38**	**41**	**21**
	π_1[%]	86	**100**	41	89
	π_2[%]	98	0	9	9
	π_3[%]	81	0	5	7

Interesting results arise from scenario 1-3-0 (Figure 6.7) where P-D-PRM and RAPTOR find the shortest path π_1 in every run, due to q_{goal} being visible from q_{start}, but fail to ever identify any of the remaining paths π_2 and π_3. Due to the layout of the map, we can assume that Visibility-PRM blocks itself off by placing a guard node in an unfavorable position. Additionally, B.spheres method, which otherwise shows more competitive results, is also able to detect these paths with less than 10% success rate. Contrarily, CTopPRM detects both paths in 70% of runs, proving its robustness in more challenging scenarios.


(a) scenario

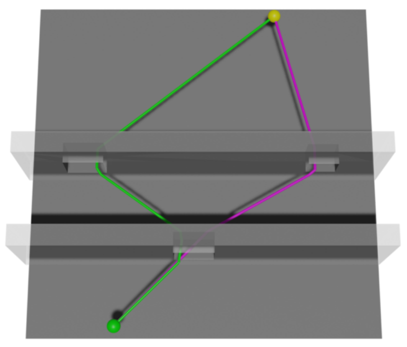


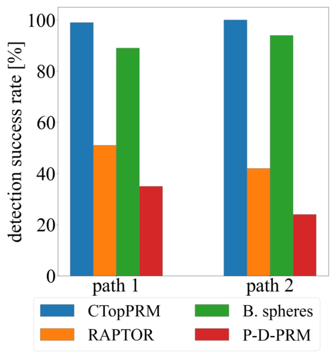
(b) success rate



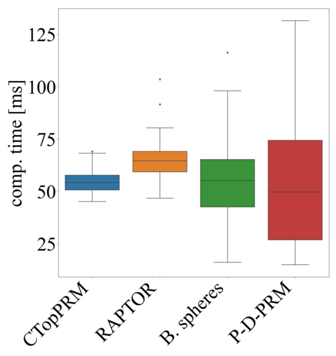
(c) computational time

Figure 6.2: Algorithm performance in windows 0-2-0 scenario.


(a) scenario

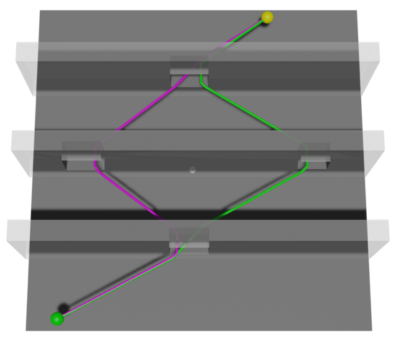


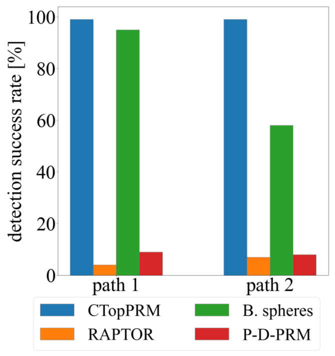
(b) success rate



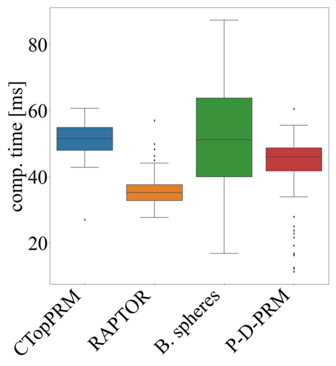
(c) computational time

Figure 6.3: Algorithm performance in windows 1-2-0 scenario.


(a) scenario

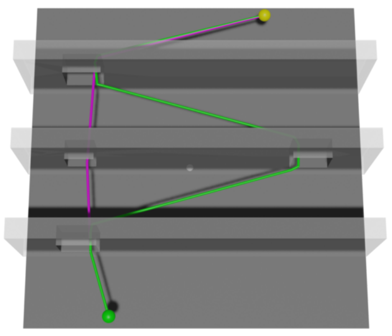


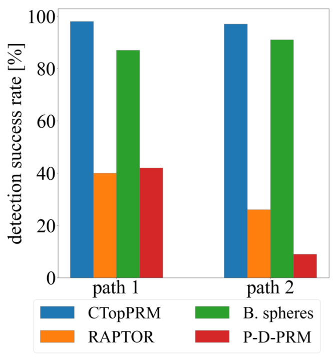
(b) success rate



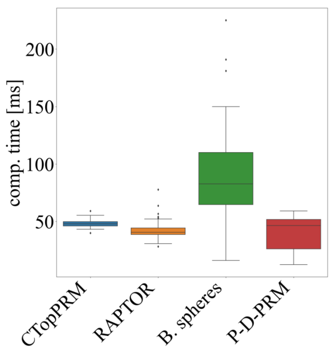
(c) computational time

Figure 6.4: Algorithm performance in windows 1-2-1 scenario.


(a) scenario

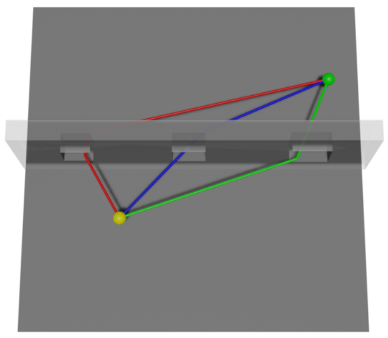


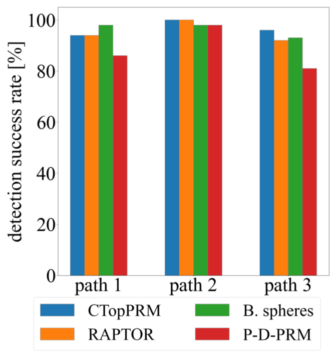
(b) success rate



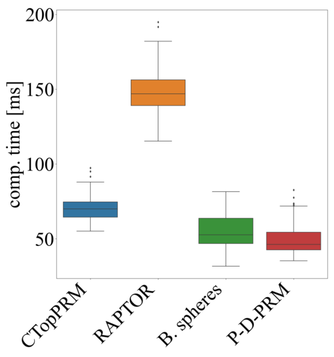
(c) computational time

Figure 6.5: Algorithm performance in windows 1s-2-1s scenario.


(a) scenario

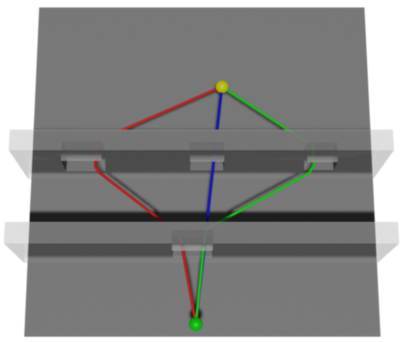


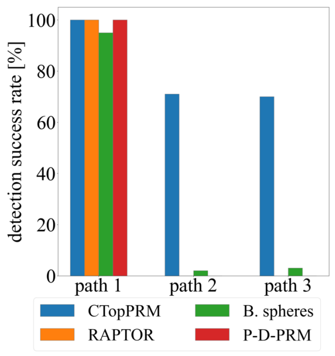
(b) success rate



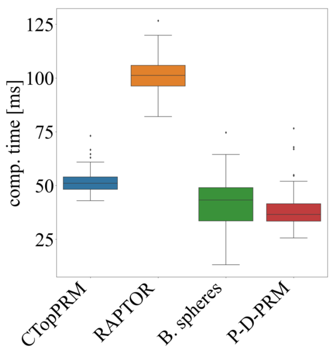
(c) computational time

Figure 6.6: Algorithm performance in windows 0-3-0 scenario.


(a) scenario

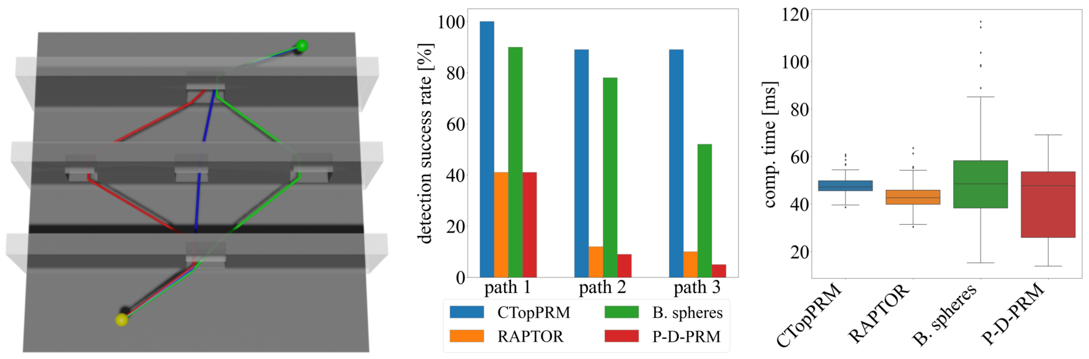


(b) success rate



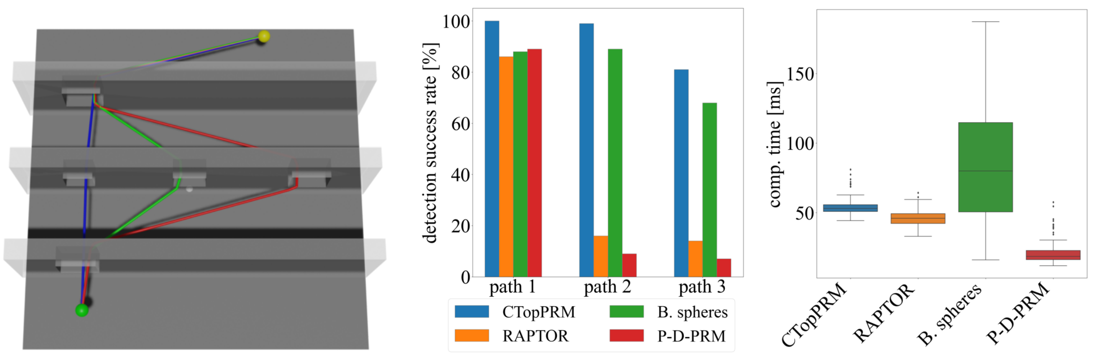
(c) computational time

Figure 6.7: Algorithm performance in windows 1-3-0 scenario.



(a) scenario (b) success rate (c) computational time

Figure 6.8: Algorithm performance in windows 1-3-1 scenario.



(a) scenario (b) success rate (c) computational time

Figure 6.9: Algorithm performance in windows 1s-3-1s scenario.

The run-time of all methods depends not only on the size of the input roadmap, but also on number of paths detected, as most of computational time is taken by filtering process described in Section 5.5. This is why scenario 0-3-0 is interesting to us, since all tested methods identify similar amount of paths. As already mentioned, the performance of the methods depends on the number of random samples used to create the initial roadmap. We show the performance with the increasing number of random samples in Figure 6.10.

They indicate that with the lower number of samples, the methods B. spheres and CTopPRM manage to find more paths, but are clearly slower than both P-D-PRM and RAPTOR. Additionally, it is important to note that P-D-PRM finds significantly fewer paths than the other methods, because it consists of two phases, which have to share the total amount of samples. With a growing number of samples, the performance of all algorithms in terms of the number of paths found converges towards three, which is the ground truth in this scenario. However, unlike B. spheres and CTopPRM, which record just a minor increase in computational time, both P-D-PRM and RAPTOR become significantly slower.

Overall, CTopPRM shows computational speed competitive with other related methods, while clearly outclassing them in terms of path detection success rate. Therefore, CTopPRM

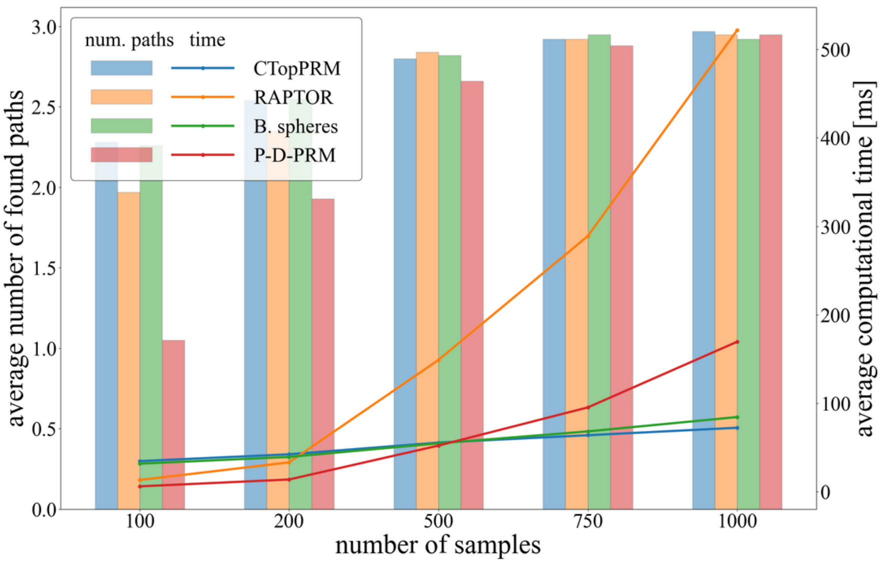


Figure 6.10: Effect of increasing number of samples on performance.

proves to be the most efficient and effective in scenarios with smaller number of distinct UVD classes.

6.2 Complex Environments

The second set of experiments was conducted in complex environments, containing a high number of distinct UVD classes. The first environment, called "poles" and depicted in Figure 6.1b, resembles a small forest-like area, while the second environment, shown in Figure 6.1c and called "building", requires a robot to traverse a closed, multi-level building area through doors and windows. For each of these environments, we tested the performance of the methods in three different scenarios with different start and goal configurations. Each method was tested in 100 runs in each scenario. The poles scenarios are tested in $\mathbb{R}^2$ space with a circular robot and building scenarios are tested in $\mathbb{R}^3$ space with a spherical robot.

Table 6.4 summarizes performance of the methods in terms of computational time, quantity of found paths represented by highest number of paths found in a single run (best) and average number of found paths across all 100 runs. Quality of paths is evaluated as an average length of n-shortest paths found over all 100 runs, where n is indicated in the table.

$$\begin{aligned} &\text{n-short.} = \frac{\sum_{i=0}^{n} \pi_i.\text{length}}{n}, \\ &\pi_i.\text{length} < \pi_j.\text{length}\ \forall \pi_i, \pi_j \in \Pi,\ i < j. \end{aligned} \tag{6.1}$$

This metric is supposed to show if a method is able to consistently find k shortest paths in each scenario.

The results show that CTopPRM performs the best in terms of quantity of paths found, identifying the most paths in a single run in every scenario, as well as significantly outscoring other methods in average number of paths found in all scenarios. Additionally, CTopPRM also outperforms other methods in terms of quality of found paths in all but one scenario in poles environment, where it records a score just 1% worse than RAPTOR. CTopPRM is also the only method to find at least n paths across 100 runs in every single scenario. Failure to do so is denoted by N/A in Table 6.4.

Table 6.4: Quantity and quality of paths found in poles and building environments

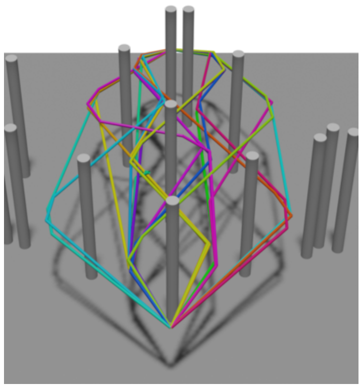
		poles, n=400			**building**, n=300		
		scenario 1	scenario 2	scenario 3	scenario 1	scenario 2	scenario 3
CTopPRM	c.t.[ms]	19	16	13	142	151	124
	best	**19**	**6**	**11**	**25**	**36**	**11**
	avg.	**15.16**±3.76	**4.80**±0.96	**7.85**±1.71	**7.78**±4.43	**8.74**±5.79	**4.77**±1.74
	n-short.	7.83	**7.79**	**7.52**	**39.49**	**35.93**	**33.16**
RAPTOR [7]	c.t.[ms]	14	20	**7**	29	21	29
	best	**19**	5	9	2	1	1
	avg.	13.71±3.03	3.61±0.94	7.28±1.44	0.41±0.58	0.01±0.10	0.06±0.24
	n-short.	**7.75**	N/A	7.54	N/A	N/A	N/A
B. spheres [5]	c.t.[ms]	**10**	**14**	13	93	114	122
	best	11	**6**	8	7	10	7
	avg.	7.47±1.48	3.42±1.05	4.06±1.08	3.24±1.18	2.85±2.02	2.26±1.17
	n-short.	7.87	N/A	7.60	46.95	N/A	N/A
P-D-PRM [24]	c.t.[ms]	55	135	27	39	35	42
	best	**19**	**6**	8	1	0	1
	avg.	14.11±1.82	4.42±1.08	7.18±0.84	0.16±0.37	0.00±0.00	0.02±0.14
	n-short.	7.80	**7.79**	**7.52**	N/A	N/A	N/A

Interestingly, P-D-PRM algorithm, which had the worst score, but shortest run-time in windows environment, achieves results most competitive with CTopPRM in terms of quantity and quality of found paths, but has the longest run-time in the poles environment. Both P-D-PRM and RAPTOR achieve results comparable to CTopPRM in poles environment, while B. spheres method, which was most competitive in windows environment, performs significantly worse than other related methods. This is caused by the increased visibility in poles environment, allowing long connections for Visibility-PRM based methods. On the other hand, P-D-PRM and RAPTOR fail to identify a single path in majority of runs in building environment. P-D-PRM is even unable to detect a single path across all 100 runs for a whole scenario. Building environment consists of multiple narrow passages, which was shown in Section 6.1, to be unfavourable for Visibility-PRM based methods. CTopPRM outperforms other methods in the complex building environment by a significant margin, showcasing its ability to deliver consistent results in all environments, verifying its robustness.

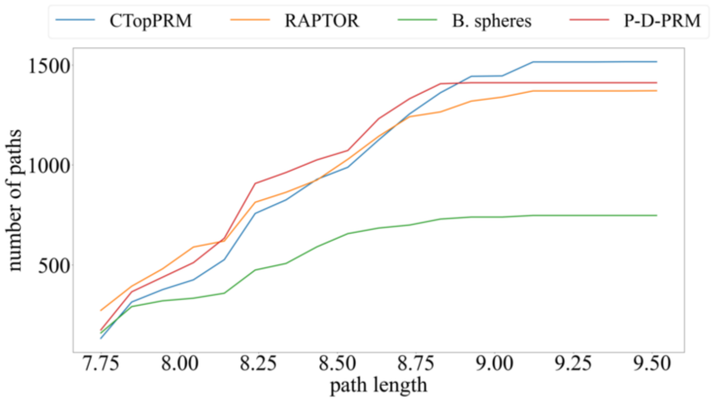
Path's quality evaluation is further shown in Figures 6.11-6.16. These figures depict cumulative histograms (b) which show the amount of paths found across 100 runs on y-axis, that are shorter than length on x-axis:

$$f(x) = |\{\pi \in \Pi \mid \pi.\text{length}() < x\}|, \tag{6.2}$$

where Π is a set of paths identified by a given method. These results show, that in poles environment all methods find the shortest paths with similar success rate, but CTopPRM manages to identify even the longer paths more consistently. On the other hand, in building environment, CTopPRM completely outclasses other methods, even finding four times the amount of paths identified by other methods.

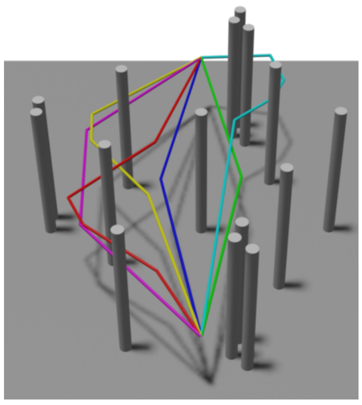


(a) scenario

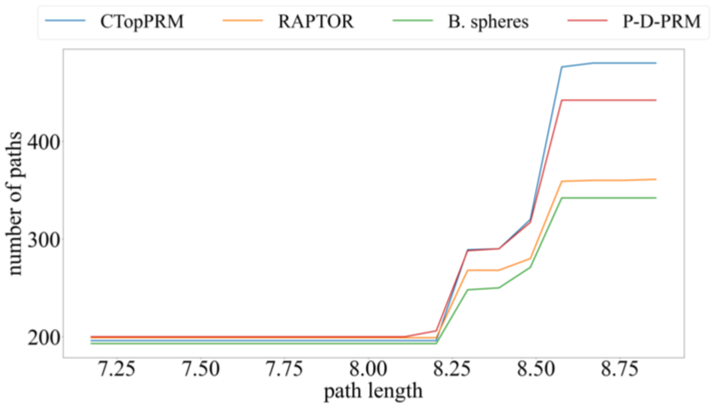


(b) cumulative histogram

Figure 6.11: Algorithm performance in poles scenario 1.

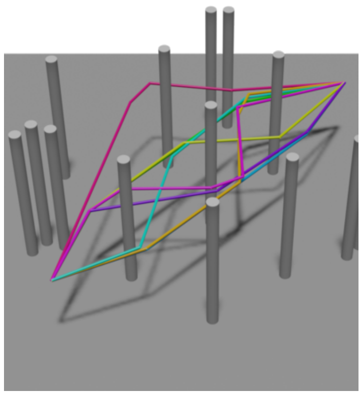


(a) scenario

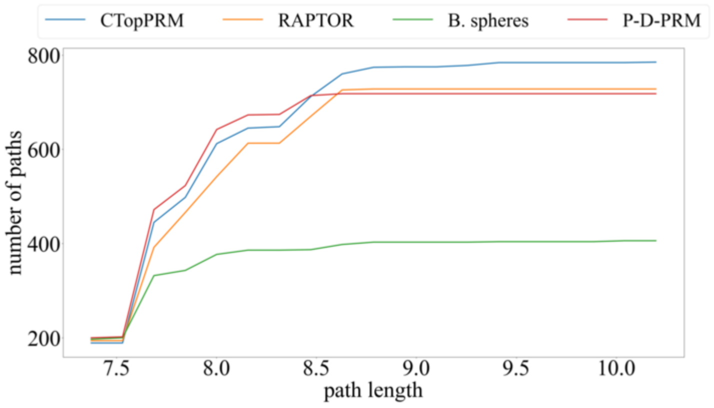


(b) cumulative histogram

Figure 6.12: Algorithm performance in poles scenario 2.

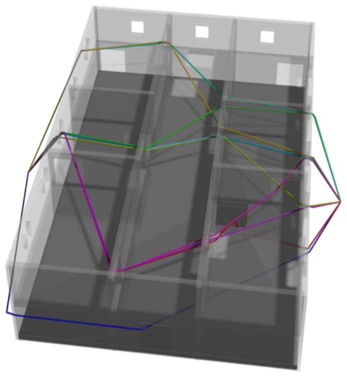


(a) scenario

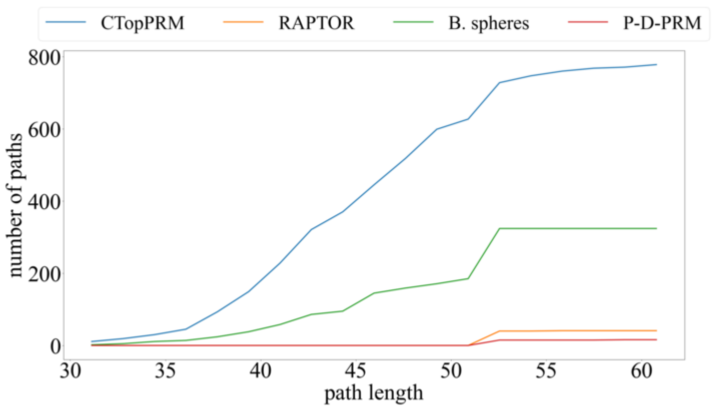


(b) cumulative histogram

Figure 6.13: Algorithm performance in poles scenario 3.

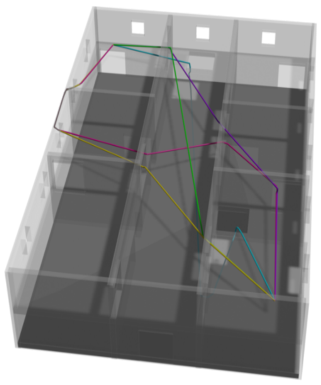


(a) scenario

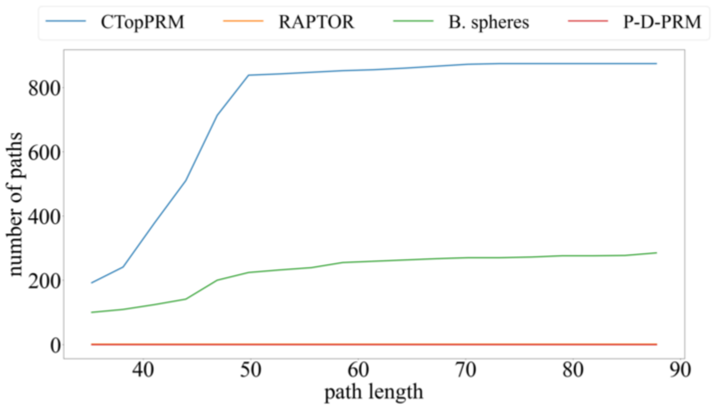


(b) cumulative histogram

Figure 6.14: Algorithm performance in building scenario 1.

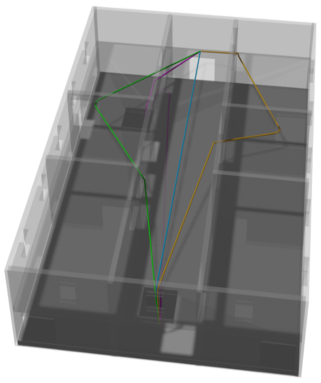


(a) scenario

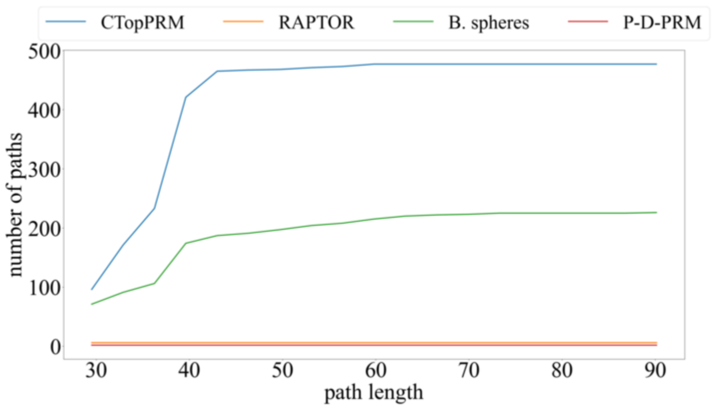


(b) cumulative histogram

Figure 6.15: Algorithm performance in building scenario 2.



(a) scenario



(b) cumulative histogram

Figure 6.16: Algorithm performance in building scenario 3.

6.3 Controllable Trade-off in Performance

The third and final round of experiments examines how the performance of the CTopPRM algorithm is affected by the trade-off parameter κ_p, the DFS termination parameter. This set of experiments was conducted in one scenario in poles environment and one scenario in the building environment. To evaluate the algorithm's performance, each κ_p configuration was run 50 times in each of the two scenarios. The evaluation metrics used in these experiments are the same as those used in the experiments described in Section 6.2: computational time, the number of paths found (highest number found in a single run and average number found across all 50 runs), and the quality of paths (measured as the average length of the n-shortest paths found over all 50 runs, where n is specified in the table, see Equation (6.1)). The results of these experiments are presented in Table 6.5.

Table 6.5: Effect of trade-off parameter κ_p on CTopPRM's performance across 50 runs

	poles scenario 1, n=200				**building** scenario 1, n=150			
κ_p	c.t.[ms]	best	avg.	n-short.	c.t.[ms]	best	avg.	n-short.
1.2	9	12	**5.86**±2.68	8.04	88	11	**4.50**±2.81	46.25
1.4	13	18	**13.06**±3.33	7.84	93	25	**7.26**±4.53	43.56
1.6	19	19	**16.26**±2.30	7.81	97	25	**7.46**±4.78	43.17
1.8	28	19	**16.64**±3.42	7.81	102	25	**9.24**±5.19	42.03
2.0	40	19	**16.94**±2.00	7.81	109	25	**9.84**±5.58	41.79

The results demonstrate that increasing the value of κ_p results in a corresponding increase in computational time, but also an increase in both the quality and quantity of paths found. This supports the claim that the CTopPRM algorithm can effectively balance these metrics. However, as κ_p is further increased, computational time continues to rise steadily while improvements in path quality and quantity become less significant. This is because the pruning parameter κ_s, which removes paths that are too long or convoluted, is still in effect, which prevents some paths from being included in the final output despite being identified by CTopPRM. It is therefore advisable to carefully select both κ_p and κ_s in order to achieve the desired trade-off between these metrics.

Overall, CTopPRM algorithm outperformed other related methods in vast majority of scenarios in quality and quantity of found paths, within the same computational time. Moreover, it has the best trade-off between computational time and number of paths found. CTopPRM additionally allows for control of the trade-off between computational time and number of paths found, making it suitable for both online planning within tens of milliseconds, and for offline planning with narrow passages.

Chapter 7

Conclusion

This thesis introduced a new sampling-based method named CTopPRM for finding multiple paths with distinct UVD classes in cluttered environments. The CTopPRM clusters initially sampled dense roadmap in order to efficiently simplify the search of multiple distinct paths. Through testing in a variety of environments, we demonstrated that CTopPRM is both efficient and robust. In majority of test cases, it surpassed other related methods in number of found distinct paths, and their length, during similar computational time. We improved the average number of homotopy classes detected within the same run-time by 30-300%, depending on the scenario. Additionally, the CTopPRM allows controlling the balance between computational time, quantity and quality of found paths, making it highly adaptable for online planning. As future work, we aim to extend CTopPRM to enable fast trajectory re-planning, and to deploy it online on Unmanned Aerial Vehicles (UAVs) to test high-speed flight in partially unknown environments.

References

[1] M. Novosad, R. Pěnička, and V. Vonásek, *CTopPRM: Clustering topological PRM for planning multiple distinct paths in 3D environments*, 2023. arXiv: 2305.13969 [cs.RO].

[2] D. Belter, "Informed guided rapidly-exploring random trees-connect for path planning of walking robots," in *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2022, pp. 709–714. DOI: 10.1109/ICARCV57592.2022.10004330.

[3] J. Ortiz, A. Clegg, J. Dong, *et al.*, *iSDF: Real-time neural signed distance fields for robot perception*, 2022. arXiv: 2204.02296 [cs.RO].

[5] R. Pěnička and D. Scaramuzza, "Minimum-time quadrotor waypoint flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, Apr. 2022. DOI: 10.1109/LRA.2022.3154013.

[4] R. Pěnička, Y. Song, E. Kaufmann, and D. Scaramuzza, "Learning minimum-time flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7209–7216, 2022. DOI: 10.1109/lra.2022.3181755.

[6] M. Aria, "Optimal path planning using informed probabilistic road map algorithm," *Journal of Engineering Research*, Dec. 2021. DOI: 10.36909/jer.ASSEEE.16105.

[7] B. Zhou, J. Pan, F. Gao, and S. Shen, "RAPTOR: Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1992–2009, 2021. DOI: 10.1109/TRO.2021.3071527.

[8] J. Denny, R. Sandström, A. Bregger, and N. Amato, "Dynamic region-biased rapidly-exploring random trees," in May 2020, pp. 640–655, ISBN: 978-3-030-43088-7. DOI: 10.1007/978-3-030-43089-4_41.

[9] V. Vonásek, R. Pěnička, and B. Kozlíková, "Searching multiple approximate solutions in configuration space to guide sampling-based motion planning," *Journal of Intelligent & Robotic Systems*, vol. 100, pp. 1527–1543, Dec. 2020. DOI: 10.1007/s10846-020-01247-4.

[10] L. Han, F. Gao, B. Zhou, and S. Shen, *FIESTA: Fast incremental euclidean distance fields for online motion planning of aerial robots*, 2019. arXiv: 1903.02144 [cs.RO].

[12] V. Vonásek and R. Pěnička, "Sampling-based motion planning of 3D solid objects guided by multiple approximate solutions," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1480–1487. DOI: 10.1109/IROS40897.2019.8968578.

[11] V. Vonásek, R. Pěnička, and B. Kozlíková, "Computing multiple guiding paths for sampling-based motion planning," in *2019 19th International Conference on Advanced Robotics (ICAR)*, Dec. 2019, pp. 374–381. DOI: 10.1109/ICAR46387.2019.8981589.

[13] X. Zhang, B. Zhang, C. Qi, Z. Li, and H. Li, "An online motion planning approach of mobile robots in distinctive homotopic classes by a sparse roadmap," in Aug. 2019, pp. 722–734, ISBN: 978-3-030-27537-2. DOI: 10.1007/978-3-030-27538-9_62.

[14] E. Huang, M. Mukadam, Z. Liu, and B. Boots, "Motion planning with graph-based trajectories and gaussian process inference," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5591–5598. DOI: 10.1109/ICRA.2017.7989659.

[15] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D euclidean signed distance fields for on-board MAV planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1366–1373. DOI: 10.1109/IROS.2017.8202315.

[16] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2016.11.007.

[17] J. Denny, J. Colbert, H. Qin, and N. M. Amato, "On the theory of user-guided planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4794–4801. DOI: 10.1109/IROS.2016.7759704.

[18] R. Kala, "Homotopic roadmap generation for robot motion planning," *Journal of Intelligent & Robotic Systems*, vol. 82, no. 3, pp. 555–575, 2016, ISSN: 1573-0409. DOI: 10.1007/s10846-015-0278-z.

[19] C. Rösmann, F. Hoffmann, and T. Bertram, "Planning of multiple robot trajectories in distinctive topologies," in *2015 European Conference on Mobile Robots (ECMR)*, 2015, pp. 1–6. DOI: 10.1109/ECMR.2015.7324179.

[20] M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard, "Online generation of homotopically distinct navigation paths," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6462–6467. DOI: 10.1109/ICRA.2014.6907813.

[21] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, no. 3, pp. 273–290, Oct. 2012. DOI: 10.1007/s10514-012-9304-1.

[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotic Research - IJRR*, vol. 30, pp. 846–894, Jun. 2011. DOI: 10.1177/0278364911406761.

[23] S. Bhattacharya, V. Kumar, and M. Likhachev, "Search-based path planning with homotopy class constraints," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, ser. AAAI'10, Atlanta, Georgia: AAAI Press, 2010, pp. 1230–1237.

[24] L. Jaillet and T. Siméon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *The International Journal of Robotics Research*, vol. 27, pp. 1175–1188, Nov. 2008. DOI: 10.1177/0278364908098411.

[25] S. M. LaValle, *Planning Algorithms.* Cambridge University Press, 2006. DOI: 10.1017/CBO9780511546877.

[26] Y. Fujita, Y. Nakamura, and Z. Shiller, "Dual dijkstra search for paths with different topologies," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, Oct. 2003, pp. 3359–3364, ISBN: 0-7803-7736-2. DOI: 10.1109/ROBOT.2003.1242109.

[27] A. Hatcher, *Algebraic topology.* Cambridge: Cambridge University Press, 2002, pp. xii+544, ISBN: 0-521-79160-X; 0-521-79540-0.

[28] E. Schmitzberger, J. Bouchet, M. Dufaut, D. Wolf, and R. Husson, "Capture of homotopy classes with probabilistic road map," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2317–2322. DOI: 10.1109/IRDS.2002.1041613.

[29] J. Nešetřil, E. Milková, and H. Nešetřilová, "Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history," *Discrete Mathematics*, vol. 233, no. 1, pp. 3–36, 2001, ISSN: 0012-365X. DOI: 10.1016/S0012-365X(00)00224-7.

[30] O. Brock and O. Khatib, "Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, pp. 550–555. DOI: 10.1109/ROBOT.2000.844111.

[31] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, pp. 477–493, Jan. 2000. DOI: 10.1163/156855300741960.

[32] O. Brock and O. Khatib, "Elastic strips: A framework for integrated planning and execution," in *The Sixth International Symposium on Experimental Robotics VI*, Berlin, Heidelberg: Springer-Verlag, 1999, pp. 329–338, ISBN: 1852332107. DOI: 10.5555/645626.662219.

[33] S. M. LaValle, "Rapidly-exploring random trees : A new tool for path planning," *The annual research report*, 1998.

[34] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. DOI: `10.1109/70.508439`.

[35] F. Aurenhammer, “Voronoi diagrams—a survey of a fundamental geometric data structure,” *ACM Comput. Surv.*, vol. 23, pp. 345–405, 1991. DOI: `10.1145/116873.116880`.

[36] H. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 116–121. DOI: `10.1109/ROBOT.1985.1087316`.

[37] T. Lozano-Pérez and M. A. Wesley, “An algorithm for planning collision-free paths among polyhedral obstacles,” *Commun. ACM*, vol. 22, no. 10, pp. 560–570, 1979, ISSN: 0001-0782. DOI: `10.1145/359156.359164`.

[38] R. Dial, “Algorithm 360: Shortest-path forest with topological ordering [h],” *Commun. ACM*, vol. 12, pp. 632–633, Nov. 1969. DOI: `10.1145/363269.363610`.

[39] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: `10.1109/TSSC.1968.300136`.

[40] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math*, vol. 1, 1959. DOI: `10.1007/BF01386390`.

[41] R. Bellman, “On a routing problem,” *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958, ISSN: 0033569X, 15524485. DOI: `10.2307/43634538`.

[42] R. C. Prim, “Shortest connection networks and some generalizations,” *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957. DOI: `10.1002/j.1538-7305.1957.tb01515.x`.

[43] J. B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956, ISSN: 00029939, 10886826. DOI: `10.2307/2033241`.

[44] O. Borůvka, “O jistém problému minimálním (About a certain minimal problem),” *Práce Moravské přírodovědecké společnosti*, vol. 3, pp. 37–58, 1926.

[45] O. Borůvka, “Příspěvek k otázce ekonomické stavby elektrovodných sítí (Contribution to the solution of a problem of economical construction of electrical networks),” *Elektrotechnický obzor 15*, pp. 153–154, 1926.

Appendix

Attachments

The attached file `topological_planning.zip` contains a collection of the software used in this thesis, along with testing environments. To install and use the software, follow the `README.md` file in the attached folder. `src/` and `include/` contain source codes for all implemented methods. Source code for the proposed CTopPRM method is implemented in the `topological_prm_clustering.hpp` file. Files specifying test cases are in `config_files/`. `blender/` contains the testing environments as `.obj` files, and scripts used for visualization of results in blender.

The attached file `multimediamaterial.zip` contains a video which describes the proposed method and shows the experimental results. The same video is available online at `https://www.youtube.com/watch?v=azNrWBU5cAk`.