

Bachelor Thesis



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Improving Sampling-Based Motion Planning Using Library of Trajectories

Michal Minařík

**Supervisor: Ing. Vojtěch Vonásek, Ph.D.
Field of study: Cybernetics and Robotics
May 2021**

Acknowledgements

First and foremost, I would like to thank my supervisor Ing. Vojtěch Vonásek, Ph.D. for his assistance and support. I would also like to thank my family and friends for supporting me during my studies and proofreading this thesis.

Declaration

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

Prague, 19. May 2021

Michal Minařík

Abstract

Motion planning is one of the fundamental problems in robotics. This thesis combines the advances in motion planning and shape matching to improve planning speeds in static environments. The first part of this thesis covers current methods used in object similarity evaluation and motion planning. The middle part describes how these methods are used together to improve planning speeds by utilizing prior knowledge about the environment, along with additional modifications. In the last part, the proposed methods are tested against other state-of-the-art planners in an independent benchmarking facility. The proposed algorithms are shown to be faster than other planners in many cases, often finding paths in environments where the other planners are unable to.

Keywords: motion planning, sampling-based planners, guided planning, 3D object similarity, library

Supervisor: Ing. Vojtěch Vonásek, Ph.D.

Abstrakt

Plánování pohybu je jedním z podstatných problémů robotiky. Tato práce kombinuje pokroky v plánování pohybu a hodnocení podobnosti objektů za účelem zrychlení plánování ve statických prostředích. První část této práce pojednává o současných metodách používaných pro hodnocení podobnosti objektů a plánování pohybu. Prostřední část popisuje, jak jsou tyto metody použity pro zrychlení plánování s využitím získaných znalostí o prostředí. V poslední části jsou navržené metody porovnány s ostatními plánovači v nezávislém testu. Námi navržené algoritmy se v experimentech ukázaly být často rychlejší v porovnání s ostatními plánovači. Také často nacházely cesty v prostředích, kde ostatní plánovače nebyly schopny cestu nalézt.

Klíčová slova: plánování pohybu, pravděpodobnostní plánovače, informované plánování, podobnost 3D objektů, knihovna

Překlad názvu: Randomizované plánování pohybu s využitím knihoven trajektorií

Contents

1 Introduction	1
1.1 Goals	2
1.2 Thesis structure	4
2 Motion planning	5
2.1 Problem definition	5
2.2 Sampling-based motion planning	6
2.2.1 Probabilistic Roadmaps	6
2.2.2 Rapidly-exploring Random Trees	8
2.2.3 Bidirectional RRT	9
2.3 Guided planning	10
2.3.1 RRT with Inhibited Regions	11
2.4 Motion planning with experience database	13
2.5 Summary	14
3 Shape matching	15
3.1 Problem definition	15
3.2 Methods	16
3.2.1 Symmetric Flips Tracking	16
3.2.2 Möbius Voting	16
3.2.3 Deep Deformations	18
3.2.4 Genetic Algorithms with Adaptive Sampling	18
3.3 Summary	19
4 Proposed solution	21
4.1 Library of Trajectories	21

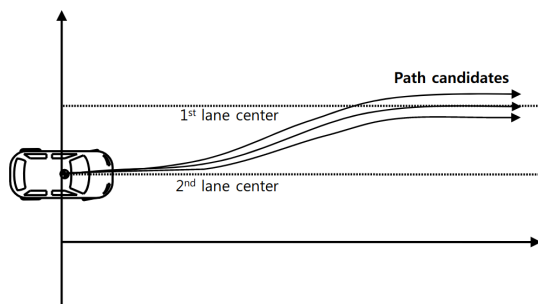
4.2 Tracking path diversity	23
4.3 Mesh similarity	25
4.4 Using guiding paths for similar objects	26
4.5 Summary	28
5 Selecting the shape matching method	29
5.1 Runtime	30
5.2 Reduced number of vertices	30
5.3 Selected method	32
5.4 Transformation invariance	32
5.5 Similarity evaluation	33
5.6 Summary	33
6 Experimental verification	35
6.1 Data preparation	35
6.2 Results	37
6.2.1 Generating guiding paths	37
6.2.2 Using ICP to transform the similar object	39
6.2.3 Finding paths for similar objects	42
6.2.4 Specifying a guiding path	42
6.3 Summary	44
7 Comparison with other planners	45
7.1 Selected planners	45
7.2 Parameter and environment setup	46
7.3 Preparation	47
7.4 Benchmark results	48

7.4.1 Easy scenarios	49
7.4.2 Medium scenarios	49
7.4.3 Hard scenarios	51
7.5 Summary	52
8 Conclusion	53
A Appendix	55
A.1 Metric	55
A.2 Object labelling test	56
B Attachments	59
C Bibliography	61
D Project Specification	65

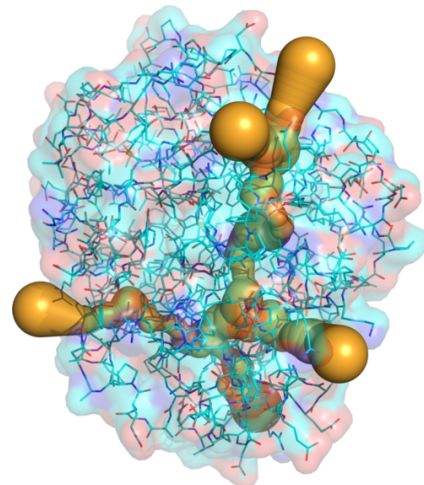
Chapter 1

Introduction

Motion planning is one of the fundamental problems and a widely studied area in robotics. The problem consists of finding a path for a manipulated object through a specified environment. One famous example is the *Piano Mover's Problem*, where a path of a piano through a house is searched for, such that no walls or other obstacles are hit. Engineers come across many other real-world problems, where an object needs to move through a given environment without colliding with obstacles, ranging from the navigation of autonomous cars (Figure 1.1a) [1] and drones [2] to the analysis of tunnels within a protein structure (Figure 1.1b) [3].



(a) : Lane changing of an autonomous car. Image courtesy of [1].



(b) : Tunnels (orange) leading from protein surface to the active site. Image courtesy of [3].

Figure 1.1: Motion planning is a widely studied area in robotics with many real-world applications, such as car navigation (1.1a) and protein analysis (1.1b).

Motion planning is an NP-hard problem [4]. One of the main approaches, which has recently been very successful, is to use sampling-based algorithms. They aim to find the path by iteratively sampling random configurations (position and rotation) of the manipulated object and checking for collisions. When enough valid configurations are found, a path through the environment is constructed [4]. Many algorithms have been developed to address specific problems arising from using sampling-based methods. One such problem is moving an object through a narrow passage (illustrated in Figure 1.2). In the narrow passage, the object's movement needs to be very precise. Therefore, using uniform sampling methods may require a lot of time to sample enough configurations that do not result in collisions. By identifying the narrow passages prior to the planning, we can increase the sampling rate inside the narrow passage. This consequently increases the probability of finding a path through the narrow passage.

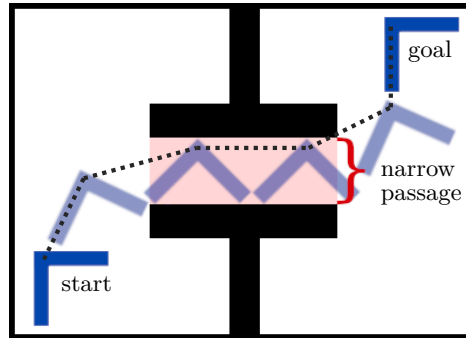


Figure 1.2: To pass through the narrow passage (red) without collision, the manipulated object (blue) needs to be moved very precisely.

Modifying the sampling probability is the idea of guided planning methods such as *Guided Rapidly exploring Random Tree* (RRT-Path) [5] or *Rapidly exploring Random Trees with Inhibited Regions* (RRT-IR) [6]. Guiding-based planners utilize so called *guiding paths* — paths through the workspace that are available for the planner and serve as hints of possible paths through the environment. Such paths can be found by analyzing the workspace geometry (e.g., the Voronoi diagram [5]), or by planning under relaxed constraints (e.g., scaled-down object [6]). The computed paths consequently guide the planner when solving the original problem by increasing the sampling probability along them.

1.1 Goals

The goal of this thesis is to understand modern sampling-based motion planning and shape matching methods, with the purpose of combining them to improve the efficiency of sampling-based motion planners.

We propose a novel method named *Rapidly exploring Random Trees with Library of Trajectories* (RRT-LIB), consisting of two phases. In the *preparation phase* illustrated in Figure 1.3, we compile a library containing paths¹ for multiple object classes. The paths are generated using a planner able to quickly approximate paths through a static environment with narrow passages. This prior knowledge is then leveraged by the planner in the *planning phase*, illustrated in Figure 1.4. When a path for a new object is required, we retrieve the paths of the most similar object in the library. These paths will serve as guiding paths, hinting at the possible paths through the environment.

With the proposed method, we aim to improve the speed of the sampling-based planners, particularly in situations when we plan in the same workspace repeatedly. In the case that the planner will be given multiple tasks in a known set of static environments, we aim to increase the chances of success by collecting information about the environment prior to the planning. This knowledge, in the form of possible paths, is saved to the library and retrieved when a new planning task is given to the planner.

¹More precisely, trajectories (configurations of an object over a period of time) are contained in the library — hence the name, Library of Trajectories. However, we are mainly interested in the configurations of the object moving through the environment. Therefore, the word “path” will be used further in this thesis, indicating that we are not considering the time aspect of the movements.

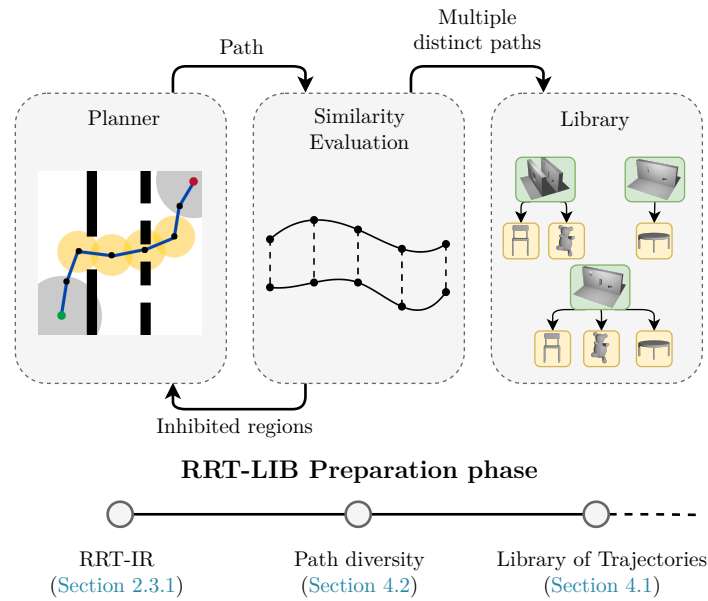


Figure 1.3: The preparation phase of the RRT-LIB algorithm. Multiple paths through the environment are iteratively computed by the RRT-IR planner. Using the already found paths as an input to the planner increases the probability of finding distinct paths. After enough distinct paths are found, the planning is terminated, and the found paths are saved into the library.

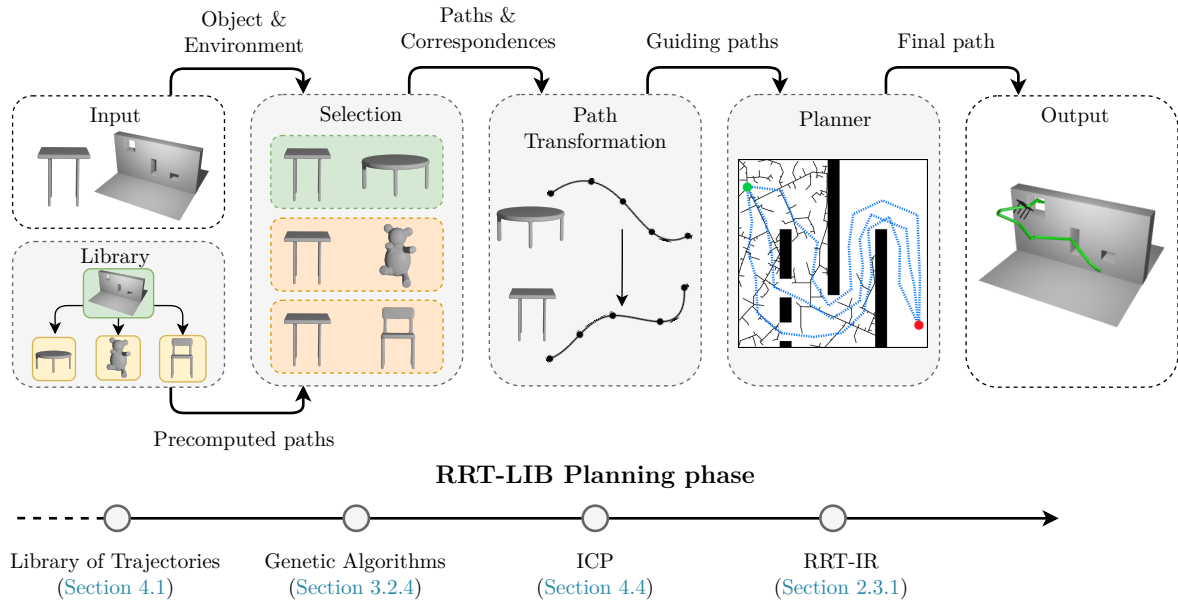


Figure 1.4: The planning phase of the RRT-LIB algorithm. The input consists of a manipulated object and an environment. Based on the inputs, paths computed for the most similar object in the library are loaded. After transforming the paths to account for possibly different positions of the manipulated object and the object from the library, we gain paths hinting at the possible paths through the environment. Increasing the sampling rate along these guiding paths should increase the probability of finding a solution.

1.2 Thesis structure

[Chapter 1](#) serves as an introduction to the areas of concern and outlines the structure of this thesis. The motivation and the goals of this thesis are presented. We also introduce a novel method RRT-LIB, aiming to achieve the specified goals — specifically, improve the efficiency of the sampling-based planners in environments where a narrow passage is present.

A brief description of motion planning is presented in [Chapter 2](#). We define the motion planning problem and all needed terms, along with a more in-depth description of various sampling-based motion planning methods.

A crucial part of the proposed solution is selecting an object from the library, which is the most similar to a given object. In [Chapter 3](#), we describe current approaches to shape matching and measuring the similarity of objects. We identify the capabilities of the current methods, mainly aimed at features that will allow us to reach the specified goals.

In [Chapter 4](#), we present the structure of the library. The library is designed with convenience in mind, allowing us to retrieve useful data efficiently. We also propose modifications to the selected sampling-based planner RRT-IR, allowing us to generate multiple distinct paths and store them in the library. However, the object in the library might be represented in a different coordinate system than the manipulated object. Therefore, the paths from the library cannot be used immediately as guiding paths for a similar object. This issue is discussed and solved by finding a transformation that aligns the two objects, ensuring the guiding path gives a reasonable estimate of the searched path.

Three of the shape matching methods presented in [Chapter 3](#) are implemented and tested in [Chapter 5](#). Based on the test results, we select one method that meets our criteria. We further test this method to ensure that no problems will arise when selecting the most similar object from the library. One such problem which would need to be addressed during the planning phase would be the dependence of the result on the query object scale.

Implementation details are presented in [Chapter 6](#), along with experimental results. The correct implementation is verified, and we show that the planner with all proposed modifications does indeed exhibit the expected behavior.

To test whether the proposed methods really improve the planning runtime, we implement our planner in an open-source planning library OMPL. Using OMPL benchmarking module, we compare our RRT-LIB planner to other state-of-the-art planners. This comparison is covered in [Chapter 7](#).

Achieved results and possible improvements are further discussed in [Chapter 8](#), which also serves as an overall summary of this thesis.

Chapter 2

Motion planning

In motion planning, we are given an object and search for a way to move it through an environment without colliding with obstacles. This chapter defines the motion planning problem and presents basic sampling-based methods.

2.1 Problem definition

We denote the manipulated *object* (or robot, agent) \mathcal{A} . The space through which is the object \mathcal{A} moved is represented by a *world* \mathcal{W} (usually a subset of \mathbb{R}^2 or \mathbb{R}^3). An *obstacle region* $\mathcal{O} \subseteq \mathcal{W}$ represents the obstacles in the environment we are planning in. In this thesis, \mathcal{A} will be a non-deformable object and the world \mathcal{W} will be \mathbb{R}^3 .

A *configuration* q of a non-deformable body in the standard three-dimensional Euclidean space uniquely describes the object's position $P \in \mathbb{R}^3$ and 3D rotation $R \in SO(3)$ (the group of all possible 3D rotations). We denote $\mathcal{A}(q)$ the set of all points occupied by the object in the configuration q . Many parametrizations of $SO(3)$ exist. Among the most commonly used are Euler angles — three angles describing the object rotation with respect to a chosen fixed coordinate system. Using Euler angles simplifies the visualization and manipulation of robots using the Denavit-Hartenberg (DH) notation. It also allows for a straightforward rotation matrix generation. However, problems arise when a metric or interpolation between two rotations needs to be defined, both crucial in motion planning [4]. Considering the disadvantages of the Euler angles, quaternions were used in this thesis. While sacrificing the intuitive visualization, we gain a well-defined metric (covered in [Section A.1](#)) and an interpolation technique (Spherical Linear Interpolation) [7].

Configuration space \mathcal{C} is defined as the set of all possible object configurations. In our case, \mathcal{C} is the special Euclidean group $SE(3) = \mathbb{R}^3 \times SO(3)$ with elements $q = ((x, y, z), (a, b, c, d))$. Two specific configurations are defined — q_{start} , specifying the starting position of the object, and q_{goal} , the target object position. We denote $\mathcal{C}_{free} \subseteq \mathcal{C}$, $q_{free} \in \mathcal{C}_{free}$ the subset of all configurations, where the manipulated object does not collide with the environment ($\mathcal{C}_{free} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} = \emptyset\}$). Conversely, $\mathcal{C}_{obs} = \mathcal{C} \setminus \mathcal{C}_{free}$ contains the colliding configurations. We also define $\mathcal{C}_{goal} \subseteq \mathcal{C}$, $q_{goal} \in \mathcal{C}_{goal}$ as the set of configurations we consider similar enough to the target configuration to be acceptable as a goal.

Path τ is a sequence of n waypoints $\{q_i \in \mathcal{C}_{free} \mid i = 1, \dots, n\}$, defining a collision-free movement of the manipulated object through the environment. The task of path planning is to find a path τ , such that $q_1 = q_{start}$ and $q_n \in \mathcal{C}_{goal}$.

2.2 Sampling-based motion planning

There exist two main principles of finding a solution to a motion planning problem. The first one is combinatorial motion planning, which aims to find a path through the continuous space without using approximations of the object, obstacles, or collisions. This approach allows for complete algorithms — algorithms that will find a solution if one exists, otherwise correctly evaluate that no such solution is available. Combinatorial planners can be based on Voronoi diagrams [8] or visibility graphs [9]. However, combinatorial motion planning needs an exact definition of the problem to properly evaluate interactions of the object with the obstacles (e.g., the obstacle representation needs to be explicit, which can be done using polygons). Creating such a representation is reasonable only when solving low-dimensional tasks (a point or a circular robot moving through a 2D world).

To allow for planning in more complex environments, we encapsulate the interaction evaluation in a collision detection module, which uses approximations to evaluate object collisions. This is the idea of sampling-based motion planning [4]. The collision detection module takes a configuration as an input and returns whether the manipulated object in this configuration collides with the obstacles. We need a way to select the configurations tested for collisions. In low-dimensional problems, choosing a resolution and performing an exhaustive (grid) search could be a viable solution. However, due to the configuration space being multidimensional, grid search becomes both time and memory consuming, often even impossible due to the latter. Therefore, the defined configuration space is sampled randomly, and a graph containing collision-free configurations is built. The alleviation of the constraints placed on the resolution allows finding paths where high-precision movement is needed [4].

Two of the most commonly used sampling-based methods, *Probabilistic Roadmaps* [10] and *Rapidly-exploring Random Trees* [4], are described below in [Section 2.2.1](#) and [Section 2.2.2](#), respectively. Many modifications of the RRT method have been developed, allowing better utilization of knowing the target configuration (*Bidirectional RRT* [4] in [Section 2.2.3](#)) or having additional info about the configuration space from previous searches (*RRT with Inhibited Regions* [6] in [Section 2.3.1](#)). A detailed survey of other RRT and PRM variants can be found in [11].

2.2.1 Probabilistic Roadmaps

The *Probabilistic Roadmap* algorithm (PRM) [10] works by randomly sampling the configuration space until a specified number of non-colliding configurations is found. A roadmap is then constructed by connecting subsets of the configurations (e.g., k-nearest neighbors or all neighbors in a specified distance). In the original implementation [10], the connection is made only if it connects two different roadmap components. However, a simplified version is used more often, where connections in the same component are allowed [12]. After all configurations are processed, the *construction phase* (the simplified version shown in [Figure 2.1](#) and [Alg. 1](#)) is completed, and the *query phase* follows ([Figure 2.2](#) and [Alg. 2](#)). During the query phase, the start and goal configuration pair is connected to the roadmap, and a path between them is found by a standard graph algorithm (such as the Dijkstra's shortest path or A*) [4].

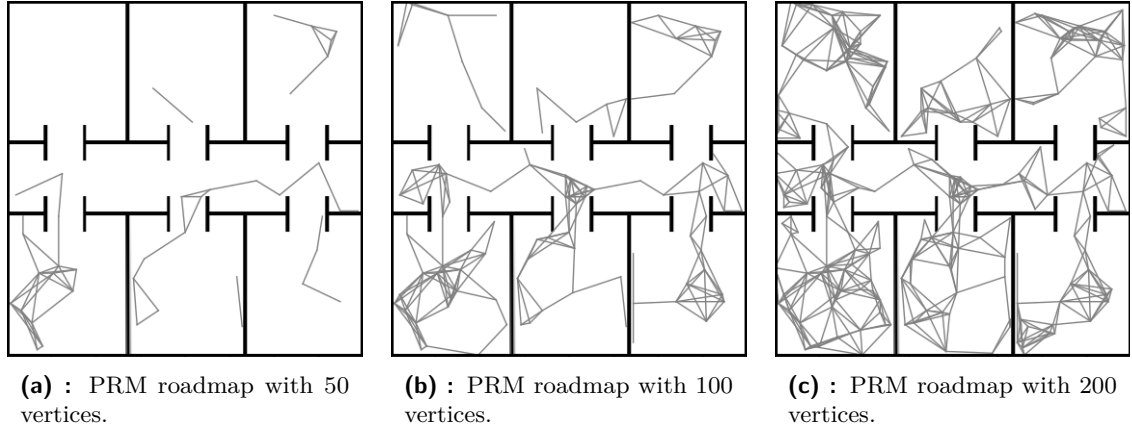


Figure 2.1: PRM construction phase. 2D configuration space \mathcal{C} is randomly sampled, with non-colliding samples from the free configuration space \mathcal{C}_{free} being added to the roadmap \mathcal{G} (gray), avoiding the obstacle region \mathcal{O} (black).

Algorithm 1: PRM construction phase

Input: Configuration space \mathcal{C} , Desired roadmap size N

Output: Roadmap \mathcal{G}

```

1  $\mathcal{G} = \text{sample\_free}(\mathcal{C}, N)$  // Sample  $N$  non-colliding configurations
2 foreach  $q$  in  $\mathcal{G}$  do
3   foreach  $q_{near}$  in  $\text{neighborhood}(\mathcal{G}, q)$  do // e.g., k-nearest neighbors
4     if  $\text{can\_connect}(q, q_{near})$  then
5        $\mathcal{G}.\text{add\_edge}(q, q_{near})$ 
6 return  $\mathcal{G}$ 

```

Algorithm 2: PRM query phase

Input: Roadmap \mathcal{G} , Configurations q_{start} and q_{goal}

Output: Path τ from q_{start} to q_{goal} or empty list

```

1  $\mathcal{G}.\text{add\_vertex}(q_{start})$  // Add  $q_{start}$  and connect to other vertices
2 foreach  $q$  in  $\text{neighborhood}(q_{start})$  do
3   if  $\text{can\_connect}(q_{start}, q)$  then
4      $\mathcal{G}.\text{add\_edge}(q_{start}, q)$ 
5  $\mathcal{G}.\text{add\_vertex}(q_{goal})$  // Same for  $q_{goal}$ 
6 foreach  $q$  in  $\text{neighborhood}(q_{goal})$  do
7   if  $\text{can\_connect}(q_{goal}, q)$  then
8      $\mathcal{G}.\text{add\_edge}(q_{goal}, q)$ 
9 if  $\mathcal{G}.\text{same\_component}(q_{start}, q_{goal})$  then
10   $\tau = \mathcal{G}.\text{find\_path}(q_{start}, q_{goal})$  // e.g., Dijkstra's shortest path
11 return  $\tau$  // Empty list when no path found

```

One of the most significant advantages of roadmaps is their reusability — once we compute the roadmap for a given environment, we can find paths for multiple query pairs with the runtime constrained only by the graph search algorithm. However, in many implementations, the connectivity of the roadmap is not checked during its creation. This can lead to a graph which does not form a single connected component (Figure 2.2).

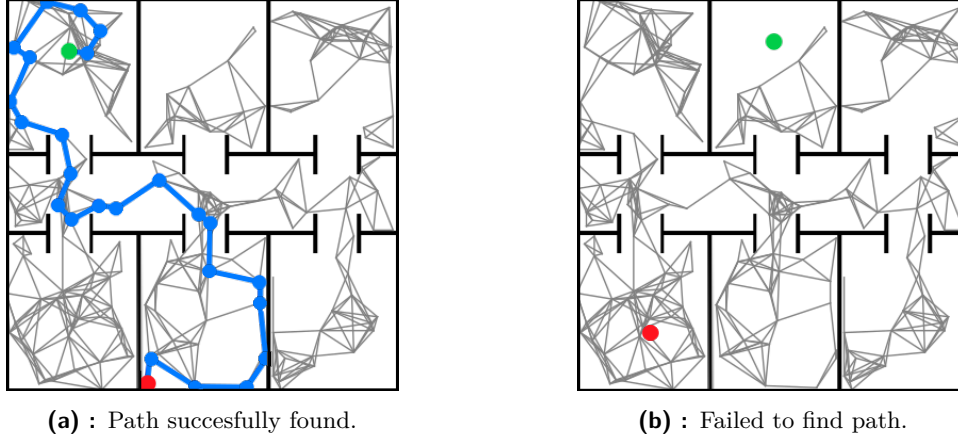


Figure 2.2: PRM query phase. A query consists of the start configuration q_{start} (green) and goal configuration q_{goal} (red). After connecting them to the roadmap, a path is searched for by a selected graph algorithm. In 2.2a, a path between q_{start} and q_{end} is successfully found (blue). However, due to the graph not forming a single connected component, a path for the second query in 2.2b is not found.

2.2.2 Rapidly-exploring Random Trees

The *Rapidly-exploring Random Tree* (RRT) [4] algorithm starts, contrary to PRMs, with a specified initial q_{start} and target q_{goal} configurations. A single tree starting from q_{start} is built by adding new, non-colliding configurations to the previously found ones. In each iteration, a random configuration $q_{rand} \in \mathcal{C}$ is generated, and the nearest configuration in the tree q_{near} is found. A straight line is constructed from q_{near} to q_{rand} , on which a non-colliding configuration q_{new} is found and added to the tree. The growth of the search tree is illustrated in Figure 2.3. After adding a configuration near the goal to the search tree, the path from the start to the goal is found easily by reverse tracking vertices in the tree (Figure 2.3c). If no such configuration is found after a specified number of iterations I_{max} , the search is terminated. The pseudocode of the RRT algorithm is listed in Alg. 3.

Algorithm 3: RRT

Input: Configuration space \mathcal{C} , Configurations q_{start} and q_{goal}

Params: Maximum iterations I_{max}

Output: Path τ from q_{start} to q_{goal} or empty list

```

1 Initialize  $\mathcal{T}(q_{start})$  // Search tree seeded at  $q_{start}$ 
2 for iteration = 1, ...,  $I_{max}$  do
3    $q_{rand} = \text{sample\_random}(\mathcal{C})$ 
4    $q_{near} = \text{nearest\_point}(\mathcal{T}, q_{rand})$ 
5    $q_{new} = \text{stopping\_configuration}(q_{near}, q_{rand})$ 
6   if  $q_{new} \neq q_{near}$  then
7      $\mathcal{T}.\text{add\_vertex}(q_{new})$ 
8      $\mathcal{T}.\text{add\_edge}(q_{near}, q_{new})$ 
9     if  $\text{is\_near\_goal}(q_{new})$  then
10      return Path  $\tau$  from  $q_{start}$  to  $q_{goal}$ 
11 return {} // Empty list when no path found

```

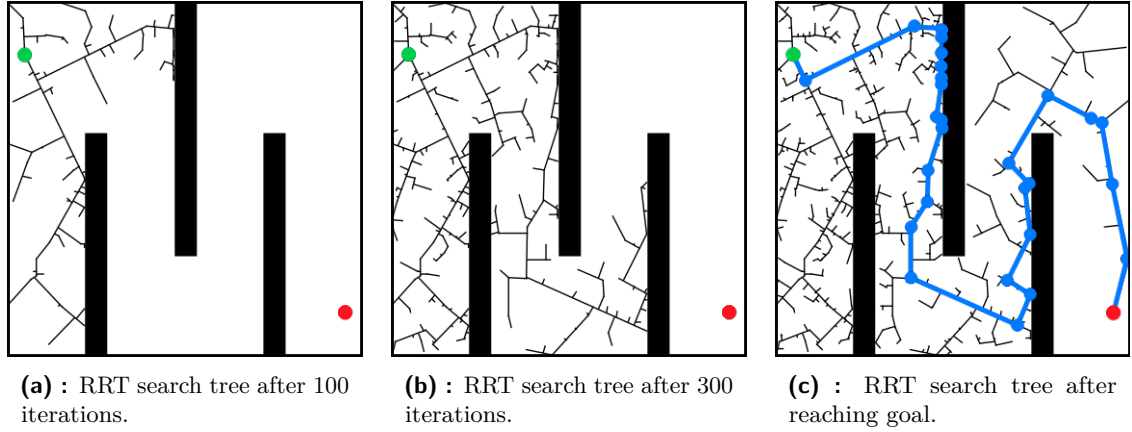


Figure 2.3: Illustration of the progress of the RRT algorithm in a 2D configuration space \mathcal{C} . The obstacle region \mathcal{O} is marked by black, \mathcal{C}_{free} by white pixels. The search tree is expanded from the starting configuration q_{start} (green) until the goal q_{goal} (red) is reached. The path τ (blue) is found by reverse tracking vertices in the tree.

2.2.3 Bidirectional RRT

A basic RRT modification uses the knowledge of q_{goal} and starts building two search trees from both q_{start} and q_{goal} , returning a path when the two trees meet [4]. This usually results in a faster runtime, especially when one of the start or goal configurations is located in an enclosed space (e.g., the bug trap problem shown in Figure 2.4). The pseudocode of the *Bidirectional RRT* algorithm is listed in Alg. 4.

Algorithm 4: Bidirectional RRT

Input: Configuration space \mathcal{C} , Configurations q_{start} and q_{goal}

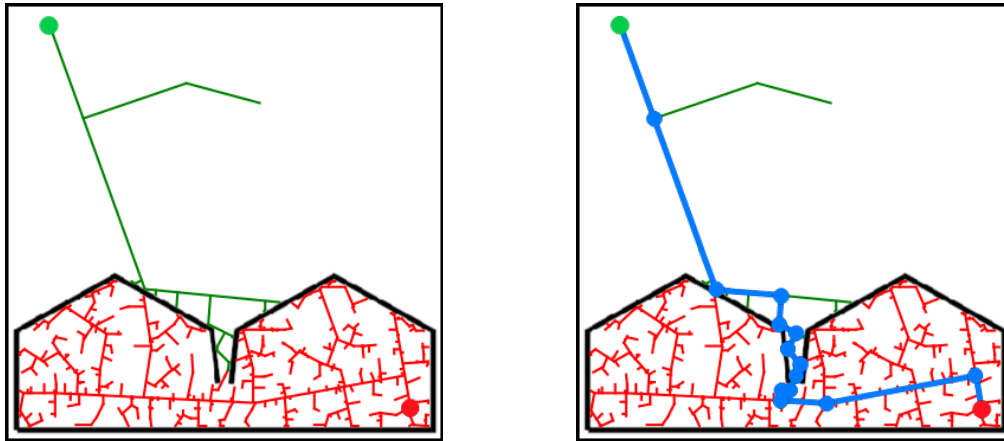
Params: Maximum iterations I_{max}

Output: Path τ from q_{start} to q_{goal} or empty list

```

1 Initialize  $\mathcal{T}_a(q_{start})$  // Search tree seeded at  $q_{start}$ 
2 Initialize  $\mathcal{T}_b(q_{goal})$  // Search tree seeded at  $q_{goal}$ 
3 for iteration = 1, ...,  $I_{max}$  do
4    $q_{rand} = \text{sample\_random}(\mathcal{C})$ 
5    $q_{near} = \text{nearest\_point}(\mathcal{T}_a, q_{rand})$ 
6    $q_{new} = \text{stopping\_configuration}(q_{near}, q_{rand})$ 
7   if  $q_{new} \neq q_{near}$  then
8      $\mathcal{T}_a.add\_vertex(q_{new})$ 
9      $\mathcal{T}_a.add\_edge(q_{near}, q_{new})$ 
10     $q'_{near} = \text{nearest\_point}(\mathcal{T}_b, q_{new})$  // Expand towards the nearest
11     $q'_{new} = \text{stopping\_configuration}(q'_{near}, q_{new})$  // vertex in the second tree
12    if  $q'_{new} \neq q'_{near}$  then
13       $\mathcal{T}_b.add\_vertex(q'_{new})$ 
14       $\mathcal{T}_b.add\_edge(q'_{near}, q'_{new})$ 
15    if  $q_{new} = q'_{new}$  then // When the trees meet,
16      return Path  $\tau$  from  $q_{start}$  to  $q_{goal}$  // construct and return a path
17  if  $|\mathcal{T}_b| > |\mathcal{T}_a|$  then // Ensure that the trees are expanded equally
18    swap( $\mathcal{T}_a, \mathcal{T}_b$ ) // by expanding the smaller tree
19 return {} // Empty list when no path found

```

(a) : Two search trees seeded at q_{start} and q_{goal} .

(b) : Final path.

Figure 2.4: Bidirectional RRT. Two search trees are built, starting from q_{start} (green) and q_{goal} (red). In each iteration, the algorithm expands the smaller tree and tries to make a connection to the nearest node of the other tree, returning a path (blue) on success.

2.3 Guided planning

Standard sampling-based planners sample the configuration space uniformly. This makes generating a collision-free path challenging in situations where the critical areas have a tiny volume. One such situations are narrow passages — small collision-free regions in the configuration space, that contain a part of the solution. Due to their low volume (compared to the volume of the configuration space), the probability of generating uniformly distributed samples in them is low. Consequently, the presence of narrow passages requires to significantly increase the number of samples, which increases the runtime. From the practical point of view, the narrow passages make the planning problem more difficult. Guiding-based planners aim to modify the sampling probability inside the key areas by introducing so called *guiding paths*. By increasing the sampling rate along the guiding paths, the probability of generating valid configurations in the key regions is increased (Figure 2.5).

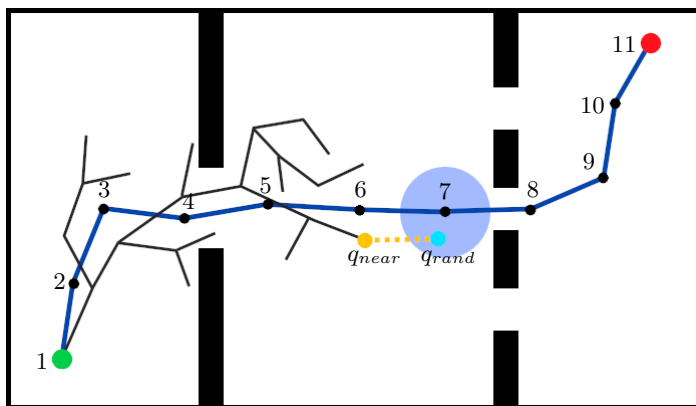


Figure 2.5: Guided planning. The guiding path (dark blue) is used to guide the expansion of the search tree (grey). The sampling rate is increased around the active waypoint 7 (blue area). The tree attempts to grow towards the random sample q_{rand} (light blue) generated around the active waypoint. After the tree approaches the active waypoint (e.g., to a predefined distance), the active waypoint is moved along the guiding path (to the waypoint 8).

When solving a low-dimensional problem (i.e., problem with less than 3 degrees of freedom), the medial axis of the workspace [13] or a path through the environment [5, 6] can be used as the guiding path. In [5], the guiding path is computed from the Voronoi diagram of a 2D workspace. Guiding paths for a high-dimensional problem can be obtained by relaxing the problem constraints [14, 15, 16]. In [14], the size of the robots or obstacles is modified to widen the narrow passage for the robot, making the problem easier to solve. The path found for the easier problem is then used to guide the planner when solving the original problem.

However, most of the approaches utilize only one guiding path. This can lead to problems when the solution belongs to a different homotopy class than the guiding path. Utilizing more guiding paths from different homotopy classes should increase the planner’s performance. However, evaluating path homotopy becomes very challenging in spaces with more dimensions than 2D. Furthermore, approaches that use only one guiding path can fail when the solution is far from the guiding path. This can be also avoided by using multiple guiding paths.

2.3.1 RRT with Inhibited Regions

Inhibited regions are an extension to the RRT algorithm (described in Section 2.2.2) proposed in [6], aiming to speed up the planning when a narrow passage is present. Inhibited regions are subsets of the configuration space, where the exploration is suppressed. By imposing such restrictions around already found paths, the planner is forced to find different paths through the environment. The main idea of this method is to find multiple guiding paths.

First, the manipulated object is scaled down, making it easier for the planner to find a path through the narrow passage. The waypoints of each computed path generate inhibited regions, and the planner tries to avoid these regions when searching for additional paths. Then, after multiple paths through the environment are found, the manipulated object is returned to its original scale, and the search is started again. During the subsequent search, the paths are used to guide the planner by increasing the sampling probability along them.

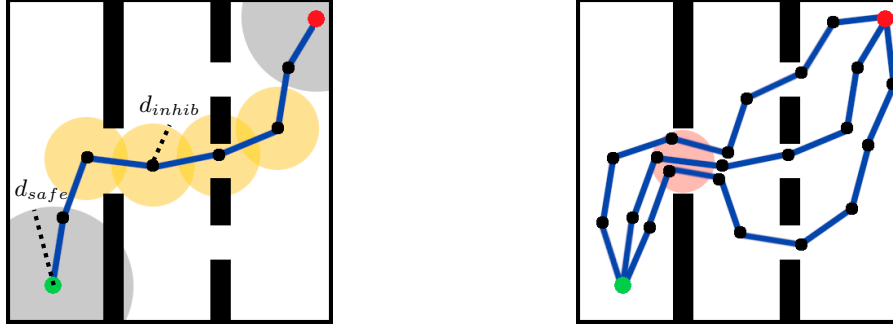
In the algorithm outlined in Alg. 5, the standard RRT planner loop presented in Alg. 3 is expanded. With the probability p_{bias} , the samples are chosen around the guiding paths \mathcal{G} . To track the search progress along each guiding path, active waypoints v_i are introduced, representing the first point of the i -th guiding path that has not yet been reached. In the case that the expansion towards the sampled configuration enters an inhibited region, it is not terminated directly (doing so would permit finding paths in environments where different paths need to go through the same bottleneck, as shown in Figure 2.6). Instead, entering the inhibited region is controlled by the following probability function

$$p_{i,j} = \begin{cases} \exp\left(-\frac{B(i,j)}{a_{sum}}\right) & \text{if } A(i,j) = 0 \\ 0 & \text{otherwise,} \end{cases} \quad (2.1)$$

where

$$A(i,j) = \max_{k>j}(a_{i,k}), \quad B(i,j) = \max_{k\leq j}(a_{i,k}) \quad (2.2)$$

and a_{sum} is the total number of attempts to enter the inhibited regions.



(a) : Each path generates inhibited regions around its configurations (yellow).

(b) : Sometimes, paths are forced to go through the same bottleneck (red).

Figure 2.6: RRT with Inhibited Regions. When searching for the guiding paths, each new path generates inhibited regions, in which the exploration is suppressed (2.6a). In environments where different paths go through the same bottleneck (2.6b), permitting the planner from entering the inhibited regions would result in the planner failing to find multiple paths. This is solved by introducing the probability Eq. 2.1.

Algorithm 5: RRT-IR find path

Input: Configuration space \mathcal{C} , Configurations q_{start} and q_{goal} , Guiding paths $\mathcal{G} = (g_1, \dots, g_m)$, Inhibited regions $\mathcal{R} = (r_1, \dots, r_n)$

Params: Inhibited region radius d_{inhib} , Maximum iterations I_{max} , Guiding bias p_{bias}

Output: Path τ from q_{start} to q_{goal} or empty list

```

1 Initialize  $\mathcal{T}(q_{start})$  // Search tree seeded at  $q_{start}$ 
2  $v_i = 0$  for  $i = 1, \dots, m$  // Index of active waypoint on guiding path  $g_i$ 
3  $a_{i,j} = 0$  for  $i = 1, \dots, n; j = 1, \dots, n_i$  // Number of attempts to enter the  $j$ -th
// configuration in the inhibited region  $r_i$ 
4  $a_{sum} = 0$  // Total attempts to enter all regions
5 for  $iteration = 1, \dots, I_{max}$  do
6   if  $|\mathcal{G}|$  and  $random(0, 1) < p_{bias}$  then
7      $i = random\ 1 \leq i \leq m$ 
8      $q_{v_i} =$  active waypoint on path  $P_i$ 
9      $q_{rand} = sample\_around(\mathcal{C}, q_{v_i})$ 
10  else
11     $q_{rand} = sample\_random(\mathcal{C})$ 
12     $q_{near} = nearest\_point(\mathcal{T}, q_{rand})$ 
13     $q_{new} = stopping\_configuration(q_{near}, q_{rand})$ 
14    if  $q_{new} \neq q_{near}$  then
15       $p_{enter} = 1$ 
16      if  $|\mathcal{R}| > 0$  then
17         $q_{inhib}, i, j = nearest\_point(\mathcal{R}, q_{new})$  // Nearest configuration in  $\mathcal{R}$ 
//  $q_{inhib} = r_i[j]$ 
18        if  $dist(q_{new}, q_{near}) < d_{inhib}$  then
19           $a_{i,j} = a_{i,j} + 1$ 
20           $a_{sum} = a_{sum} + 1$ 
21           $p_{enter} = p(i, j, a, a_{sum})$  // Eq. 2.1
22        if  $|\mathcal{R}| = 0$  or  $random(0, 1) < p_{enter}$  then
23           $\mathcal{T}.add\_vertex(q_{new})$ 
24           $\mathcal{T}.add\_edge(q_{near}, q_{new})$ 
25          if  $is\_near\_goal(q_{new})$  then
26            return Path  $\tau$  from  $q_{start}$  to  $q_{goal}$ 
27          if  $is\_near(q_{new}, q_{v_i})$  then
28             $v_i = v_i + 1$  //Progress to the next waypoint
29 return {} // Empty list when no path found

```

2.4 Motion planning with experience database

In a series of papers published under Kavraki Labs¹, researchers aimed to exploit the experience of robots performing tasks in a mostly static workspace (e.g., mobile robots used for shelf stacking in a warehouse) to improve their performance over time. The experience of a robot working in an environment is saved to a database in the form of paths or roadmaps. When a new planning task is given to the robot, planning guided by the data from the database is used in combination with planning from scratch. If the raw planning is successful, a new experience is created, expanding the database. Otherwise, the prior knowledge is used to support the planner.

In [17], *Experience-Driven Random Trees* (ERT) are proposed. When a robot is given multiple planning tasks in an environment, the solution paths are saved into a database. Over time, multiple paths are accumulated in the database, representing the robot’s experience gained by moving through the environment. During consequent planning tasks, the saved paths are fetched from the database and split into multiple segments (*micro-experiences* as named by the authors), representing high-level robot movements. The algorithm tries to expand the search tree by these segments or small deformations of them, as illustrated in Figure 2.7. This allows to grow the tree by larger segments instead of expanding by a single configuration.

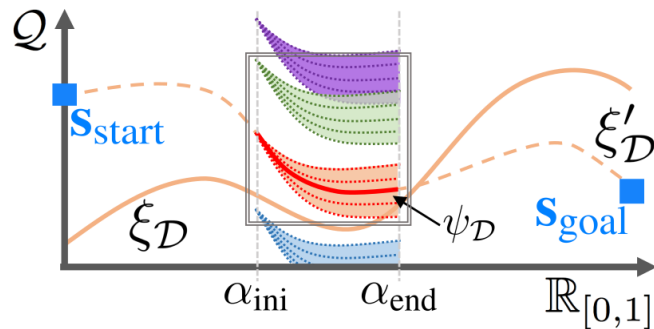


Figure 2.7: Illustration of the *Experience-Driven Random Trees* proposed in [17]. The experience from the database $\xi_{\mathcal{D}} : \alpha \in [0, 1] \rightarrow \mathcal{C}_{free}$ is mapped onto the current planning problem given by the start and the goal configurations s_{start} and s_{goal} . This mapped experience $\xi'_{\mathcal{D}} : \alpha \in [0, 1] \rightarrow \mathcal{C}_{free}$, $\xi'_{\mathcal{D}}(0) = s_{start}$ and $\xi'_{\mathcal{D}}(1) = s_{goal}$ is split into multiple segments — micro-experiences. One such segment $\psi_{\mathcal{D}} = \xi'_{\mathcal{D}}(\alpha_{ini}, \alpha_{end})$ is shifted (illustrated by colors) and sheared (illustrated by dotted lines). The algorithm then tries to expand the search tree with the deformed segments. Image courtesy of [17].

In [18], the experience is a movement through a specific obstacle region (e.g., a robot’s arm moving through a shelf of a bookcase). The workspace is divided into smaller obstacle regions (local primitives). At first, a standard sampling-based planner (RRT, PRM, etc.) is used to find a path through the obstacle region. When the robot comes across a similar local primitive during consequent planning, its experience is used to guide the planner locally. However, to use this approach, an explicit model of the obstacles in the workspace is needed to create the local primitives. Because the experience depends on both the robot and the workspace, this method is suited for situations where one robot repeatedly works in the same environment.

¹<http://www.kavrakilab.org/nsf-ri-1718478.html>

The results show improvements over other sampling-based planners that start each planning task from scratch. However, the proposed algorithms work with the assumption that the same robot is used for all tasks. The database is filled gradually by the robot's experience gained by finishing specific tasks in the environment. In the case that a different robot would be used, the experience gained by the original robot would not have to stay valid. This thesis also aims to create a database of paths, serving as the planner's experience. However, our goal is to design a method that will improve the planner's performance even when planning for an object different from the one used to generate the experience.

■ 2.5 Summary

In this chapter, motion planning was described, along with the definition of needed terms. From the most common methods developed to solve the motion planning problem, the sampling-based approach was presented. Sampling-based algorithms such as Rapidly Exploring Random Trees (Section 2.2.2) and Probabilistic Roadmaps (Section 2.2.1) work by randomly sampling the configuration space and using the collision detection module to evaluate the collisions between the manipulated object and the obstacles.

Sampling-based algorithms can become inefficient when a narrow passage is present. In such environments, we need to find a specific configuration in which the manipulated object does not collide with the obstacle region. When the sampling is uniform, the probability of generating samples in narrow passages is low, which increases the runtime. To address this issue, guiding-based methods such as RRT-Path [5] or RRT-IR [6] can be used. First, approximate paths through the narrow passage are found under relaxed conditions (e.g., a scaled-down object). By increasing the sampling rate along these paths, we guide the planner and increase the probability of finding the specific configuration needed. However, the computation of the guiding paths introduces a considerable overhead and needs to be done prior to every search.

A considerable amount of research has been performed on including the prior experience of a robot into consequent planning tasks, with the goal to increase the planning speed [17, 18]. Although this research tries to leverage the prior experience and is therefore similar to our thesis, an important difference needs to be pointed out. In the research mentioned, the experience was generated for a specific robot, and the same robot should be used during the consequent planning. On the other hand, we aim to use the gained experience when planning for objects that are not necessarily the same. Our goal is to use guiding paths computed for similar objects. We create a library containing information about the objects and their paths through various environments. To solve the path planning problem for a particular object, we first find the most similar object in the library and use its paths to guide the search for the manipulated object.

Chapter 3

Shape matching

Guiding paths can be used to overcome the inefficiency of sampling-based planners when the manipulated object needs to pass through a narrow passage as mentioned in [Chapter 2](#). This thesis aims to further utilize the concept of guiding paths by using paths that have already been computed for a different object. However, using paths computed for an arbitrary object could lead to problems when the manipulated object is of a completely different shape. Therefore, to select an appropriate (i.e., the most similar) object from the library, we need a way to evaluate the shape similarity. In this chapter, we present methods for matching 3D objects and computing their similarity, with the goal to identify and understand current capabilities.

3.1 Problem definition

In shape matching, the problem is to find a correspondence map between two objects. This has many applications in fields such as computer graphics (motion interpolation [\[19\]](#)) and medical imaging [\[20\]](#). The area of our concern will be 3D shape matching, where we search for a map between two 3D objects represented by their meshes. The methods can be divided into two main categories, depending on the type of the output correspondence map. Dense maps assign each vertex from one object a counterpart from the other object. Sparse maps, on the other hand, connect only a subset of the object vertices. This difference is visualized in [Figure 3.1](#).



Figure 3.1: When evaluating vertex-to-vertex correspondence, either sparse or dense maps can be computed. Dense maps match all vertices between the two objects (illustrated by vertex color). With sparse maps, only a subset of all vertices is selected and matched instead (illustrated by points and lines). Image courtesy of [\[21\]](#).

The correspondences are often found as a result of a minimization task, meaning a measure of the match quality needs to be defined. One common technique comes from the idea of analyzing the change of the geodesic distance. The geodesic distance between two points

on an object is the shortest path along the object surface. Since we are working with discrete mesh representations, the search for the shortest path along a surface is reduced to finding the shortest path in a graph and can be solved by the Dijkstra’s shortest path algorithm. Under isometric transformations (transformations where the distances and angles on the object surface are preserved), the geodesic distance between any two points does not change. Therefore, when a correspondence map for which the geodesic distance between points stays unchanged is found, we know that the two objects are isometric transformations of each other, and we found the best correspondence map.

As an example, in [22], a subset of vertices $\mathcal{S} = \{s_i\}$ from one (source) mesh and $\mathcal{T} = \{t_j\}$ from the other (target) mesh are matched, both of size N . The goal is to find a bijection $\phi : \mathcal{S} \rightarrow \mathcal{T}$ minimizing the *Average Isometric Distortion*, defined as

$$\mathcal{D}_{iso}(\phi) = \frac{1}{|\phi|} \sum_{(s_i, t_j) \in \phi} d_{iso}(s_i, t_j), \quad (3.1)$$

$$d_{iso}(s_i, t_j) = \frac{1}{|\phi'|} \sum_{(s_l, t_m) \in \phi'} |d_g(s_i, s_l) - d_g(t_j, t_m)|, \quad (3.2)$$

where d_g is the geodesic distance and ϕ' is a set of all correspondence pairs except the one, which contribution is calculated ($\phi' = \phi \setminus \{(s_i, t_j)\}$).

3.2 Methods

Shape comparison and matching is a complex task for which many algorithms have been developed. A broad and detailed comparison of the available methods is presented in [23] and [24]. Some examples are described below.

3.2.1 Symmetric Flips Tracking

A common problem in shape matching occurs when the shapes contain symmetrical parts (e.g., the left and the right arm of a human) [21]. Often, the symmetrical parts are matched incorrectly to each other — in the example of the human model, the left arm in the initial human shape is matched to the right arm in the target shape (Figure 3.2).

In [21], the author tries to handle the symmetric flip problem by tracking multiple correspondence maps up to a level of detail. This approach works in most cases but is very slow compared to other methods. The increased runtime comes from the fact that dense maps are computed instead of subsampling and finding correspondences between a small number of vertices.

3.2.2 Möbius Voting

In [25], subsets of vertices are sampled and mapped to a complex plane. Random triples from both subsets are iteratively selected, and three correspondence pairs are generated. In each

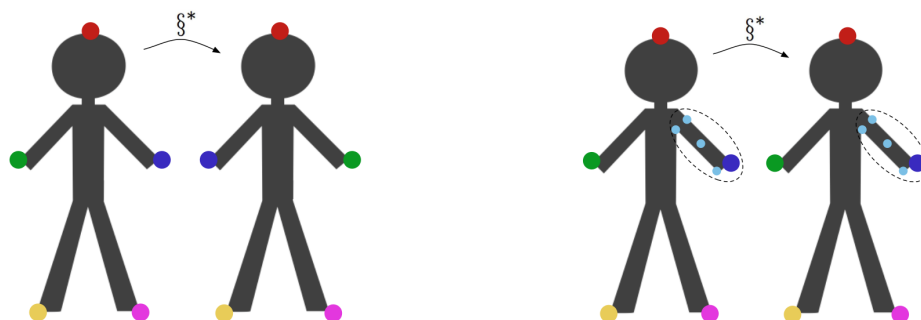


Figure 3.2: When using sparse maps, the sampled subsets can contain vertices symmetric in terms of geodesic distances. On the left image, matching arms left to right (green) and right to left (blue) does not change the average geodesic distance. By tracking multiple correspondence maps, the ambiguities can often be resolved. Thanks to the added vertices on the right arm (cyan), the arms are correctly matched (left to left in green and right to right in blue). Image courtesy of [21].

iteration, a Möbius transformation that maps one triplet to the other is found (a closed-form is guaranteed to exist — three points on a plane can be mapped to the other three points with a Möbius transformation unambiguously). All other points are also transformed, and the deformation error (how far are the transformed vertices from the target vertices) is calculated. When the error is low (the transformed vertices are close to the target vertices), the three pairs (or some of them) are likely to be correct. Visualization of the embedding and transformation is shown in Figure 3.3.

Möbius Voting outputs a measure of shape similarity (Eq. 3.1) and the corresponding vertex pairs. However, it suffers from the symmetric flip problem due to sparse sampling and is restricted to a sphere topology.

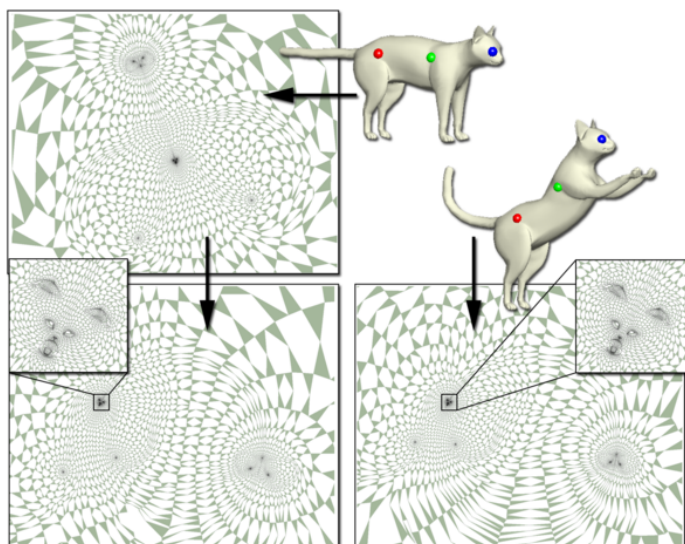


Figure 3.3: In [25], the objects are embedded into the extended complex plane. Triplets of points from both objects are sampled randomly, each time defining a unique Möbius transformation mapping the first triplet to the other. When the deformation error is low, the pairs are likely to be correct. Image courtesy of [25].

3.2.3 Deep Deformations

In [26], deep learning approaches are used to match an input shape to a template representing a specific class of shapes. Namely, an encoder-decoder style deep neural network *Shape Deformation Network* is used. For each shape class, a template is created. The encoder creates a feature embedding of the input shape. This feature embedding and the template are fed into the decoder, which deforms the template into a shape similar to the input shape. By comparing the deformed template to the input, we gain correspondences between the two shapes.

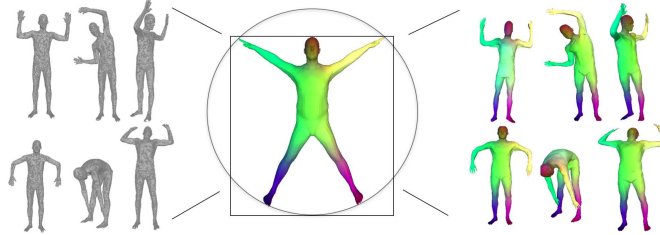


Figure 3.4: Using Deep Deformations [26], an unknown shape (left) is matched to one of the class templates (middle). Correspondences with the template are then evaluated (right). Image courtesy of [26].

Although the result quality was shown to be very high, this method comes with multiple downsides. As with other deep learning applications, a lot of hand-crafted training data is needed. For that reason, it is optimized mainly for human models, for which many high-quality datasets are available. Moreover, the class of the input shape needs to be specified. Therefore, this method can not be used to label unknown objects.

3.2.4 Genetic Algorithms with Adaptive Sampling

One of the recent methods presented in [22] uses genetic algorithms to find the correspondence map between two objects (summary presented in the SHREC'19 contest paper [27]).

Genetic algorithms are inspired by the process of natural selection. First, a subset of all possible solutions to the given problem (initial population) is selected. In each iteration, the candidates are evaluated by a fitness function measuring the solution quality. The goal can be to maximize this function (e.g., achieved score in a game) or to minimize it (e.g., obstacles hit by a car). The fittest candidates are then used to create a new generation of candidates (Figure 3.5). After a terminating condition is met (e.g., a specified number of iterations elapsed), the candidate with the highest fitness value is declared as the solution [22].

Here, a candidate represents a bijection between vertices sampled from the initial and target shape (a correspondence map). The goal is to find a bijection that minimizes the fitness function — in this case, the average isometric distortion (Eq. 3.1). After a satisfying solution is found, the sampled points are further moved by alternating optimization (*Adaptive Sampling* as named by the author), which should result in even better correspondences.

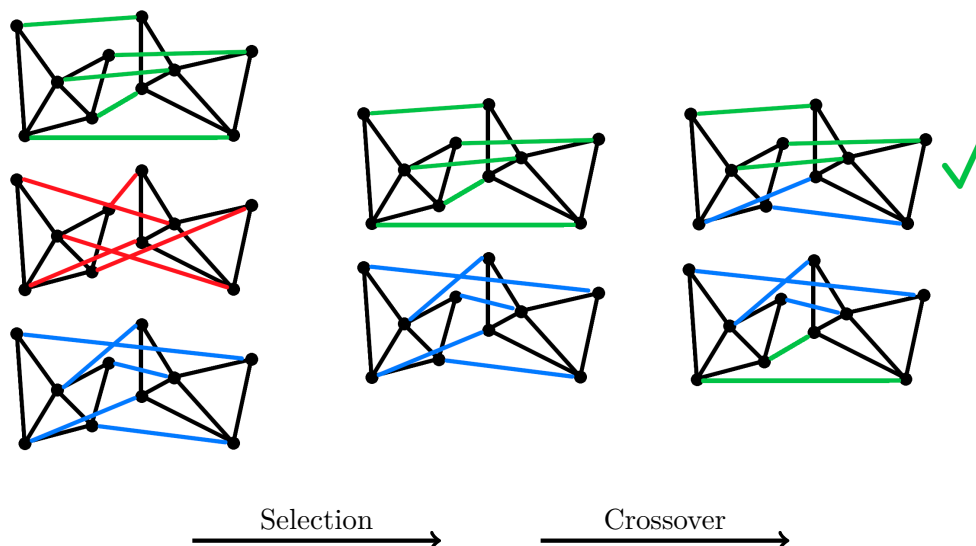


Figure 3.5: Illustration of shape matching using genetic algorithms. We search for a map between the vertices of two meshes (black). A solution candidate is a bijection between the vertices (colored lines). The green, red, and blue lines represent three different solution candidates in one generation. The fittest candidates in the generation are selected (green and blue). From these, a new generation is created by crossovers (swapping parameter values between the candidates) and mutations (random perturbations of the values). The correct solution appears in the new generation (indicated by the green checkmark).

■ 3.3 Summary

This chapter introduced the problem of shape matching and current approaches to solving this task. We identified that utilizing shape matching methods can provide us with two key capabilities when working with 3D objects. First, the similarity of two objects can be evaluated, giving us information about how similar or different the two objects are. This also means we are able to select among multiple objects the most similar to a specified object. Second, a list of corresponding points between two objects can be generated up to a variable level of detail. Both abilities will become useful in the following chapters.

Chapter 4

Proposed solution

In [6], it was shown that using approximations of possible paths can improve the inefficiency of the sampling-based planners when planning in environments with narrow passages. However, computing the approximations prior to every planning introduces a considerable overhead and slows the planning process. To solve this issue, we propose a novel method named RRT-LIB, consisting of two main phases, the *preparation phase* and the *planning phase*. The main idea is to store the already computed paths in a library and use them during later searches.

At the beginning of this chapter, the structure of the library used to store the precomputed paths is specified. We also introduce modifications to the state-of-the-art sampling-based planner RRT-IR described in Chapter 2, which allows us to fill the library in the preparation phase of RRT-LIB.

Storing the paths computed for multiple objects allows us to use them as hints during the planning phase, when a path for a query object is searched for. We use the shape matching methods presented in Chapter 3 to select an object from the library, which is the most similar to the query object. Finally, we introduce a method to transform the paths from the library into the guiding paths for the query object.

4.1 Library of Trajectories

At first, the structure of the underlying path library needs to be defined. The library will serve as the foundation for our subsequent work by providing paths precomputed for multiple different objects. These paths will be used as a suggestion to the planner of the possible paths through the given environment (Figure 4.2).

The main idea is to save the paths which have already been computed in a structure that allows for easy querying. Each path has two important parameters, the object, and the environment, which we will use to index our database. A composite index¹ raises the question of key ordering. We can choose the object most similar to the query object and look at whether the library contains computed paths for that object through the given environment. The second option is first to select the required environment and then choose the most similar object among the objects for which paths through the environment are available.

¹Also known as *Multiple-Column Index* in SQL languages <https://dev.mysql.com/doc/refman/8.0/en/multiple-column-indexes.html>

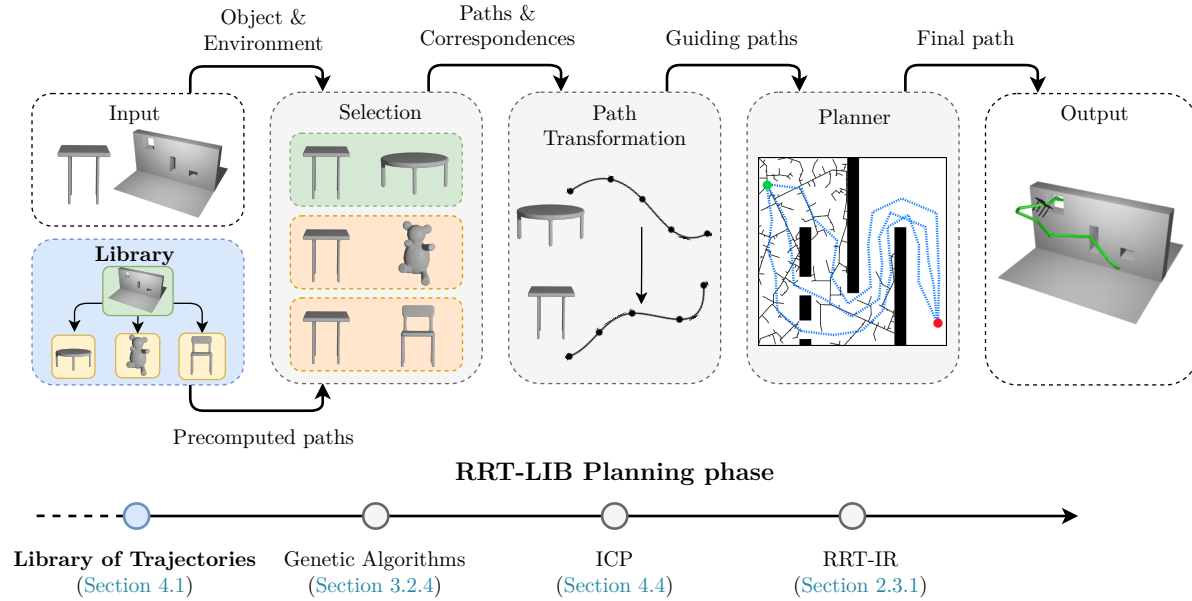


Figure 4.2: For each environment, the library contains precomputed paths for multiple objects. These paths will be used to guide the planner when planning for a different object, hinting at the possible paths through the environment.

The answer comes quite naturally after analyzing the selection workflow. The former option would not be advantageous because in case the library does not contain paths through the required environment for the most similar object, we do not gain any valuable knowledge. Considering the latter option, first selecting the environment and then selecting the most similar object is more convenient. Even in case the library does not contain paths for an object particularly similar to the query object, we still get paths giving us an estimate of possible paths through the environment. The library structure is illustrated in [Figure 4.3](#).

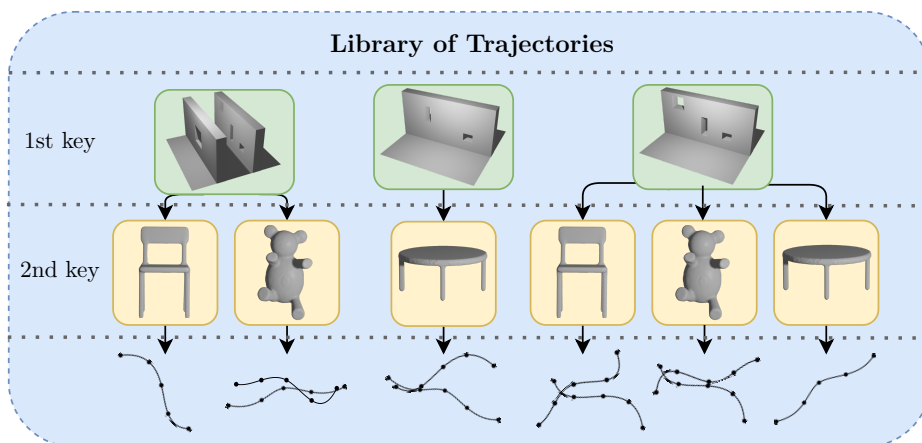


Figure 4.3: Structure of the library. For each environment (green), paths for different objects (yellow) are computed and saved. During the RRT-LIB planning phase, the paths are retrieved from the library and used as guiding paths.

4.2 Tracking path diversity

Since the probability of finding similar paths during the first phase is nonzero (due to the probability Eq. 2.1), some method of tracking path diversity needs to be implemented. Otherwise, we would end up with groups of paths being similar to each other, which would result in a decreased efficiency of the second phase, where speed is crucial. One such method is presented in [28], a modification of which is used here. By combining the diversity tracking with the RRT-IR algorithm presented in Section 2.3.1, we are able to fill the library (Figure 4.4).

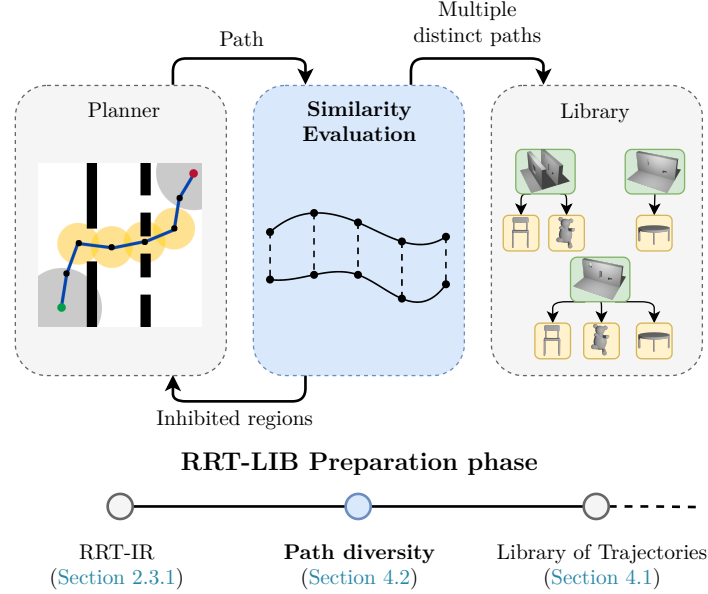


Figure 4.4: Path generation is the main part of the preparation phase of the RRT-LIB algorithm. Each path found by the RRT-IR algorithm is compared to the paths that have already been computed during the search. By filtering the paths by their similarity, only distinct paths are saved to the library. This increases the efficiency of the RRT-LIB planning phase.

The distance between paths τ_1 and τ_2 can be calculated as the average distance between each waypoint in the first path and the nearest waypoint in the second path

$$\rho(\tau_1, \tau_2) = \frac{1}{|\tau_1|} \sum_{q \in \tau_1} \rho(q_1, q_2^*) \quad (4.1)$$

$$q_2^* = \arg \min_{q_2 \in \tau_2} \rho(q_1, q_2). \quad (4.2)$$

The distance of a path τ from a set containing multiple paths $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_n\}$ can then be computed as

$$\rho(\tau, \mathcal{T}) = \min_{i=1, \dots, n} \max\{\rho(\tau, \tau_i), \rho(\tau_i, \tau)\}. \quad (4.3)$$

In order for a path to be considered *distinct* from the paths already found, its distance to the set containing the paths needs to be higher than a specified threshold. One reasonable threshold value, and the one that will be used, is the d_{inhib} radius.

Tracking the diversity of the guiding paths comes with two advantages. Adding a nondistinct path to the set of guiding paths would not give us any new information. Removing such

paths results in a smaller number of guiding paths, preserving the diversity. This improves the speed of the second phase, where the guiding paths are used to guide the path search. However, to enforce searching for new, distinct paths, the nondistinct path is still added to the inhibited regions.

The second advantage comes from the fact that searching for new guiding paths can be automatically stopped when the path diversity reaches a plateau, in contrast to having to specify the number of guiding paths manually prior to the search phase. In reality, stopping as soon as the first nondistinct path is found could lead to missing more complicated paths. Therefore, a parameter $n_{\text{diversity_patience}}$ is introduced, and the search is terminated once no distinct path has been found in the last $n_{\text{diversity_patience}}$ steps.

Of course, methods that use a heuristic to decide whether two paths are distinct come with some issues. In our case, using the average distance between paths, even two homotopic paths (i.e., paths that can be continuously deformed from one to the other without passing through the obstacle region) [4] can be declared distinct, if they are far enough in the free space. Among the more advanced methods for filtering paths based on their path through the environment is topological clustering [29]. Using such methods could lead to further improvements. However, it is out of the scope of this thesis.

Combining the RRT-IR *find path* method presented in Section 2.3.1 with a control loop, we get the final algorithm for the RRT-LIB preparation phase outlined in Alg. 6.

Algorithm 6: RRT-LIB preparation phase

Input: Configuration space \mathcal{C} , Configurations q_{start} and q_{goal}

Params: Diversity patience $n_{\text{diversity_patience}}$, Inhibited region radius d_{inhib} ,
Safe distance d_{safe}

Output: Set \mathcal{T} containing paths τ_i from q_{start} to q_{goal} or empty set

```

1  $\mathcal{T} = \{\}$  // Generated paths
2  $\mathcal{R} = \{\}$  // Inhibited regions
3  $n_{\text{same}} = 0$  // Number of iterations no distinct path was found
4 while  $n_{\text{same}} < n_{\text{diversity\_patience}}$  do
5    $\tau = \text{find\_path}(q_{\text{start}}, q_{\text{goal}}, \{\}, \mathcal{R})$  // Restrict planning around  $\mathcal{R}$  (Alg. 5)
6   if  $\tau \neq \{\}$  then
7      $\text{dist} = \rho(\tau, \mathcal{T})$  // Eq. 4.3
8     if  $\text{dist} > d_{\text{inhib}}$  then
9        $\mathcal{T}.\text{add}(\tau)$  // Add  $\tau$  to  $\mathcal{T}$  only if distinct enough
10       $n_{\text{same}} = 0$ 
11     else
12        $n_{\text{same}} = n_{\text{same}} + 1$ 
13      $\mathcal{R}.\text{add}(\{q \in \tau \mid \text{dist}(q, q_{\text{start}}) > d_{\text{safe}} \wedge \text{dist}(q, q_{\text{goal}}) > d_{\text{safe}}\})$ 
14 return  $\mathcal{T}$ 

```

4.3 Mesh similarity

After the library is filled by the computed paths, we need to have a way to choose the most similar object to the query object among the available ones (Figure 4.5).

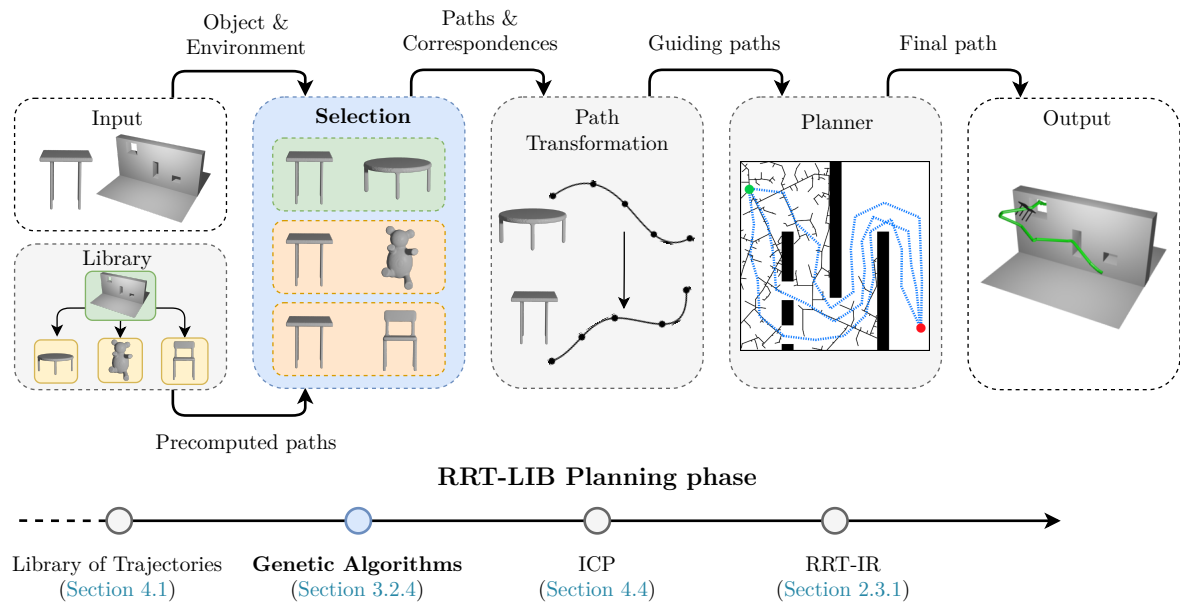


Figure 4.5: The most similar object to the input object is selected by a 3D shape matching method. Its paths through the input environment will provide us with an estimate of the possible paths.

We will base the selection of an appropriate method for evaluating the shape similarity on the following criteria:

1. the input is a pair of objects represented by their triangulated meshes,
2. one of the outputs is some form of a similarity metric, a number that allows for comparison,
3. another output is a list of corresponding vertices,
4. the method is reasonably fast and reliable.

The first requirement is given by the structure of our data. If the input object representation was different than the representation of our data, a complicated conversion would need to be implemented. The second and third requirements come from the presumed use cases. Similarity evaluation will be used to label an unknown object by comparing its similarity with multiple labeled objects. After finding the most similar object, a transformation between the two objects could lead to further improvements. Therefore, both the similarity metric and the correspondences are required. In Chapter 5, we implement and test three shape matching methods. Based on the results, one will be selected and used to evaluate the mesh similarity in the RRT-LIB pipeline.

4.4 Using guiding paths for similar objects

The RRT-IR algorithm (Section 2.3.1) with modifications proposed in Section 4.2 allows us to compute a diverse set of paths. These paths can then be used as guiding paths when planning for a similar object. However, using paths precomputed for a different object works only when the object coordinates are in the same reference frame and the objects have a similar rotation. This is a strong assumption, which in reality will not be met. The most natural way could be to represent all objects with respect to their “center of gravity”. However, this method allows only translation and will fail when objects are rotated differently. In this section, we alleviate these constraints by presenting a method of transforming the paths retrieved from the library before using them as guiding paths (Figure 4.6).

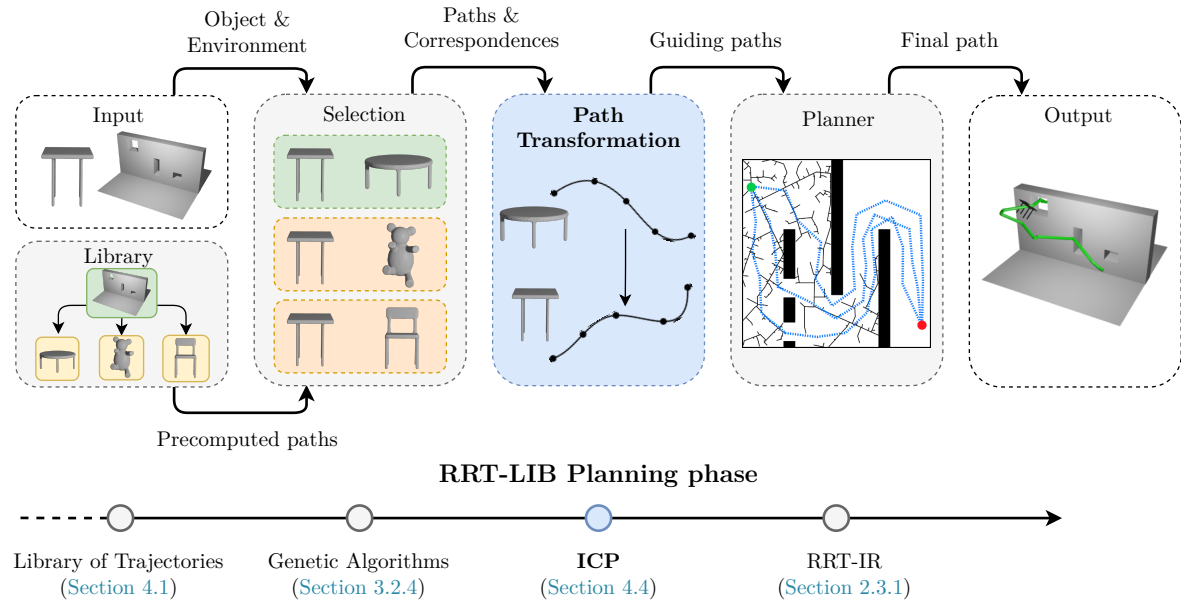


Figure 4.6: After the most similar object is selected from the library, we need to evaluate its position relative to the input object. Using the ICP algorithm, we compute a transformation minimizing the mutual distance between the two objects. By transforming the paths from the library, we can use them as guiding paths for the planner.

Let two 3D objects be represented by their meshes $\mathcal{M}_1 \in \mathbb{R}^{n \times 3}$ and $\mathcal{M}_2 \in \mathbb{R}^{m \times 3}$, where n and m are the numbers of vertices of the first and the second object, respectively. The problem consists of finding a 3D transformation that minimizes the mutual distance between the objects. Therefore, we are searching for a rotation matrix R^* and a translation vector t^* satisfying

$$d(\mathcal{M}_1, \mathcal{M}_2^{R^*, t^*}) = \min\{d(\mathcal{M}_1, \mathcal{M}_2^{R, t}) \mid R \in \mathbb{R}^{3 \times 3}, t \in \mathbb{R}^3\}, \quad (4.4)$$

where $\mathcal{M}^{R, t}$ is a mesh formed by transforming each point from \mathcal{M}

$$\mathcal{M}^{R, t} = (m_1^{R, t}, \dots, m_k^{R, t}) \quad m_i^{R, t} = Rm_i + t \quad (4.5)$$

and d is a mesh distance metric (e.g., mean squared difference). Multiple methods for solving (or finding an approximate solution to) such a problem have been proposed.

In this thesis, Iterative Closest Point (ICP) [30] will be used. ICP is an iterative algorithm that for a given *source* and *target* mesh finds a transformation of the *source* mesh, such

that the mesh distance (e.g., the sum of squared differences between the closest points) is minimized. In each iteration, the closest point in the *target* mesh is found for each point in the *source* mesh. A transformation is then found, which minimizes the sum of squared distances between the matched points². These steps are iterated until the meshes are close enough or a specified number of iterations has elapsed.

ICP is primarily used for correcting small transformations, and it is recommended to provide the algorithm with an initial guess of the transformation. We can extract this initial guess from the similarity evaluation presented in Section 4.3. From the output correspondences between the meshes, a rigid transformation minimizing the squared distance between the corresponding vertices can be found. This is, in fact, another iteration of ICP, where the correspondences in the first step are specified. By finding a transformation between the *source* and *target* mesh, we can use this information to transform the guiding paths. The algorithm is outlined in Alg. 7, with an illustration shown in Figure 4.7. By adding the ICP transformation into the planner, we get the RRT-LIB planning phase outlined in Alg. 8.

Algorithm 7: ICP with initial guess

Input: Source mesh $\mathcal{S} = (s_1, s_2, \dots, s_n)$, Target mesh $\mathcal{T} = (\tau_1, \tau_2, \dots, \tau_n)$, List of initial correspondences $L = [(s_{c_1}, \tau_{c_1}), (s_{c_2}, \tau_{c_2}), \dots, (s_{c_l}, \tau_{c_l})]$
Params: Maximum number of iterations N , Minimal error ε_{min}
Output: Rotation matrix R^* , Translation vector t^*

```

1  $R^*, t^* = \min_{R,t} \sum_{i=1}^l \|\tau_{c_i} - (R s_{c_i} + t)\|^2$  // Compute the initial guess
2  $\varepsilon = \sum_{i=1}^l \|\tau_{c_i} - (R^* s_{c_i} + t^*)\|^2$ 
3  $n = 0$ 
4 while  $n < N$  and  $\varepsilon > \varepsilon_{min}$  do
5   for  $i = 1, \dots, n$  do // Find the nearest vertex
6      $\tau_{s_i} = \text{nearest\_point}(s_i, \mathcal{T})$  // to each vertex in the source mesh
7    $R^*, t^* = \min_{R,t} \sum_{i=1}^n \|\tau_{s_i} - (R s_i + t)\|^2$  // Compute the optimal transformation
8    $\varepsilon = \sum_{i=1}^n \|\tau_{s_i} - (R^* s_i + t^*)\|^2$ 
9    $n = n + 1$ 
10 return  $R^*, t^*$ 

```

Algorithm 8: RRT-LIB planning phase

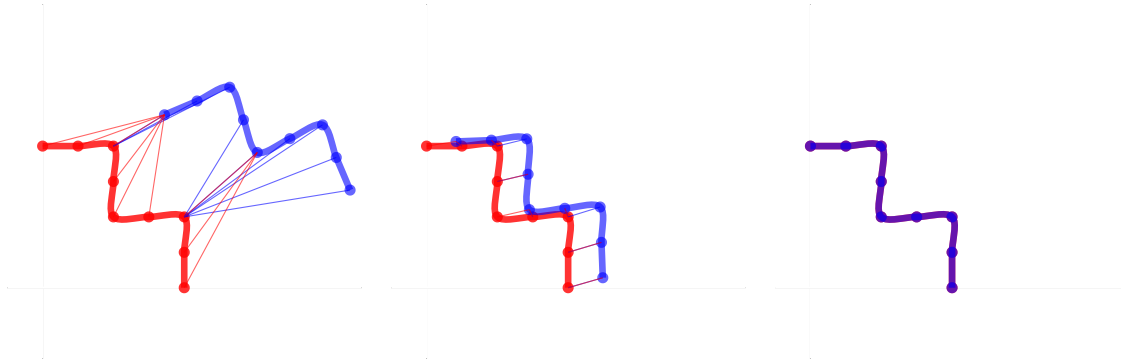
Input: Object \mathcal{O} , Map \mathcal{M} , Configurations q_{start} and q_{goal} , Library of Trajectories \mathcal{L}
Output: Path τ from q_{start} to q_{goal} or empty list

```

1  $\mathcal{O}_G, \mathcal{G}, L = \mathcal{L}.\text{get\_guiding\_paths}(\mathcal{O}, \mathcal{M})$  // Retrieve paths for similar object in  $\mathcal{L}$ 
// L - list of  $\mathcal{O}$  and  $\mathcal{O}_G$  correspondences
2  $R, t = \text{ICP}(\mathcal{O}, \mathcal{O}_G, L)$  // Find the mutual transformation
// between  $\mathcal{O}$  and  $\mathcal{O}_G$  (Alg. 7)
3  $\text{align\_objects}(\mathcal{O}, \mathcal{O}_G, \mathcal{G}, R, t)$ 
4  $\tau = \text{find\_path}(q_{start}, q_{goal}, \mathcal{G}, \{\})$  // Plan along the guiding paths (Alg. 5)
5 return  $\tau$ 

```

²Because the problem is formulated as the minimization of a sum of squares, a closed form solution exists and can be found by the method of least squares.



(a) : Initial position and rotation. (b) : *Source* transformation after the first iteration. (c) : Final *source* transformation.

Figure 4.7: Illustration of the ICP algorithm. As an input, the *source* mesh (in blue) and the *target* mesh (in red) are provided. ICP iteratively tries to find a transformation that maps the *source* mesh to the *target* mesh.

4.5 Summary

In this chapter, we introduced a sampling-based motion planning method named RRT-LIB, which consists of two phases. During the preparation phase outlined in [Alg. 6](#), paths for multiple object classes are computed by the modified RRT-IR algorithm described in [Section 2.3.1](#). The paths are stored in a library to use them later when planning for a similar object. The choices made when designing the library’s structure were covered in [Section 4.1](#). In [Section 4.2](#), we proposed a method to filter the paths by their similarity, saving only distinct paths in the library. A decreased number of redundant paths should increase the efficiency of the second phase.

When a path for a new query object in the same environment is required during the planning phase, we retrieve the paths of the most similar object in the library. The most similar object is selected by a 3D shape matching method, as described in [Section 4.3](#). The retrieved paths are transformed into guiding paths for the query object using the ICP algorithm covered in [Section 4.4](#) in combination with the correspondences found during the shape matching phase. Transformation of the paths is needed to resolve the possible discrepancy between the coordinate system of the query object and the object saved in the library. Finally, the guiding paths are used by the RRT-IR algorithm to guide the search tree. This is done by increasing the sampling probability along the guiding paths, hinting at the possible paths through the environment. The complete RRT-LIB planning phase pseudocode is listed in [Alg. 8](#).

Chapter 5

Selecting the shape matching method

In Chapter 4, we presented the RRT-LIB algorithm. The similarity of two 3D objects needs to be evaluated as a part of the planning phase, when the most similar object to a given object needs to be selected from the library. In this chapter, we test the available state-of-the-art shape matching methods and select one meeting the criteria specified In Section 4.3. The selected method will be used in our implementation of the RRT-LIB algorithm.

Three methods were selected and implemented¹: *Möbius Voting* (MV) [25], *Symmetric Flips Tracking* (SFT) [21], and *Genetic Algorithms with Adaptive Sampling* (GA) [22]. All the methods take a pair of triangulated meshes as the input and output both a similarity metric and a list of corresponding vertices. Since the similarity will need to be evaluated at every run of the RRT-LIB planner, our first concern will be the time and memory requirements.

Two datasets containing triangulated meshes were used for testing, TOSCA² [31] and PSB³ [32]. TOSCA contains isometric transformations of human and animal meshes, PSB contains non-isometric transformations of meshes divided into 19 categories. Examples of found correspondences can be seen in Figure 5.1.

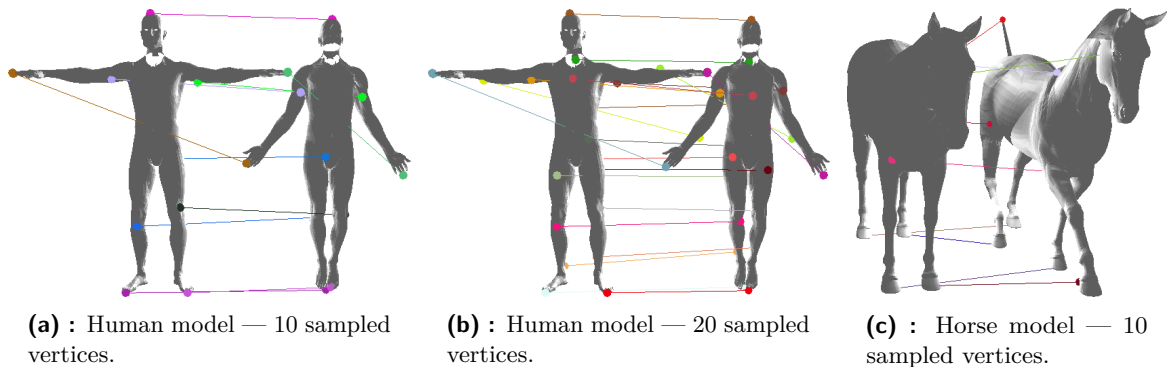


Figure 5.1: Visualization of the corresponding vertices found by the Genetic Algorithms on the TOSCA database.

¹Some parts of the author's implementation publicly available from <https://user.ceng.metu.edu.tr/~ys/pubs/> were used during the implementation of the SFT and GA methods.

²Available from http://tosca.cs.technion.ac.il/book/resources_data.html (TOSCA high-resolution)

³Available from <https://shape.cs.princeton.edu/benchmark>

5.1 Runtime

To evaluate the runtime, the algorithms are run on objects with different vertex counts and under different settings. The results are shown in [Table 5.1](#).

For some settings, memory constraints (MV for vertex count ~ 5500) or time constraints (GA for $N=40$ and higher vertex count) were exceeded. The reason for the increased runtime when increasing the vertex count in MV and GA methods is that a sample of vertices (of size N) has to be found prior to the main correspondence search. SFT works with all vertices, the runtimes are therefore higher and depend heavily on the original object vertex count.

We see that MV and SFT are usable only for models with a low vertex count, while GA speed is kept high despite increasing the model quality (while leaving N low).

Vertices	MV (10)	MV (20)	MV (40)	GA (10)	GA (20)	GA (40)	SFT
~ 5500	-	-	-	5	15	-	980
~ 4000	102	110	120	3	8	-	860
~ 2500	48	54	55	3	7	-	525
~ 1250	15	18	20	2	5	-	220
~ 500	4	4	10	1	5	2380	47
~ 100	0.6	0.8	6	1	4	230	4

Table 5.1: Runtime comparison for selected methods (in seconds).

5.2 Reduced number of vertices

Due to the runtime limitations, the number of vertices needs to be reduced prior to running additional tests. However, this could lead to a worse result quality for the GA method, which is able to cope even with high vertex count meshes. To see how reducing the number of vertices in an object changes the quality of the output correspondences, a model is selected and downsampled by *Blender's* `Decimate geometry` feature.

The results are shown in [Figure 5.2](#) and [Figure 5.3](#). Even though the result changes depending on the number of vertices, there is no clear trend that would indicate a change in the result quality. The results on the low vertex count versions are compared to SFT in [Figure 5.4](#).

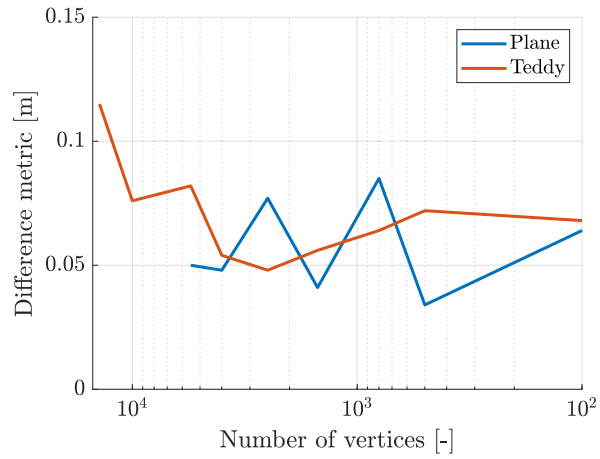


Figure 5.2: Average distortion in mesh units (defined in Eq. 3.1) for different number of mesh vertices.

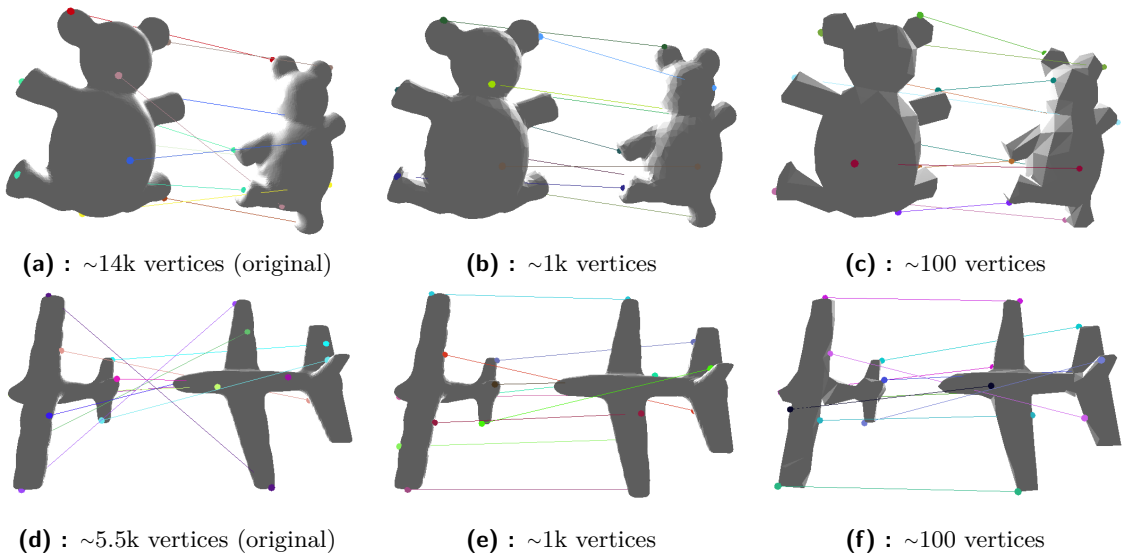


Figure 5.3: GA results on downsampled objects.

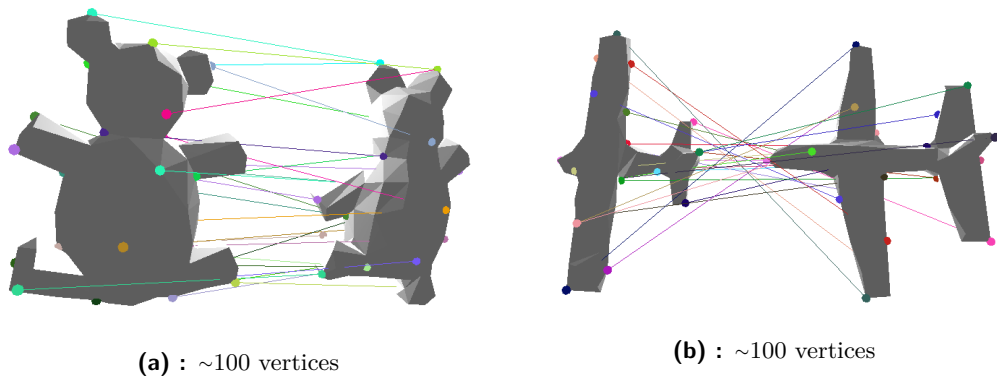


Figure 5.4: SFT results on downsampled objects.

5.3 Selected method

The Genetic Algorithms method results are generally better than the results of the two other methods. Moreover, the GA method is faster in all cases, even on higher vertex count meshes, where the other two methods fail. Considering the test results, it will be the method of our choice for further testing.

5.4 Transformation invariance

To examine how the GA method reacts to different object transformations, we rotate and scale the target object while keeping the initial object unchanged. For each transformation, the algorithm is run and the found correspondences are evaluated.

We see that the resulting correspondences found between the initial and the target object are not affected by either rotation (Figure 5.5) or scaling (Figure 5.6).

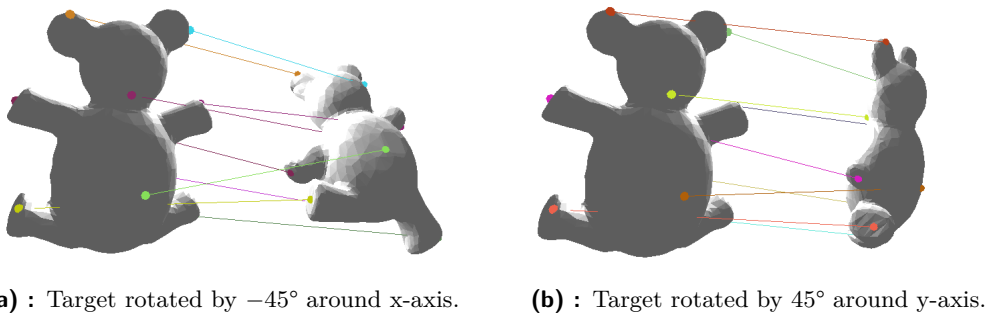


Figure 5.5: Matching under rotation. We observe that the rotation of the object does not change the result.

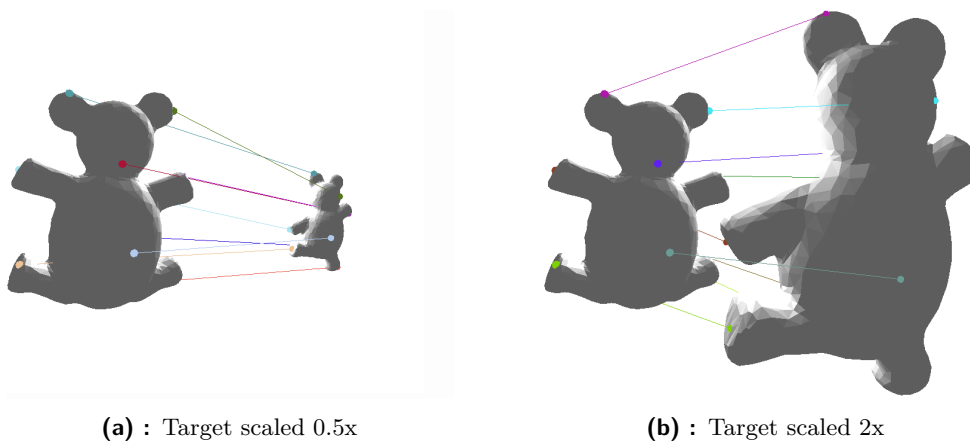


Figure 5.6: Matching under scaling. We observe that the scale of the object does not change the result.

5.5 Similarity evaluation

In addition to vertex matching, the similarity needs to be evaluated to label an unknown object. GA implements the *maximum isometric distortion* ($maxIso$) metric. This metric is suitable only for isometric or nearly isometric deformations. However, the objects in PSB are not isometric deformations of each other — a more suitable approach is to compute the average over the vertices. Therefore, we use the *average isometric distortion* ($avgIso$). Using this metric, the results are improved — full results can be found in the appendix (Section A.2). Two examples of objects mislabeled by the $maxIso$ metric are presented in Figure 5.7, one where switching to $avgIso$ helped and one where the label stayed wrong.

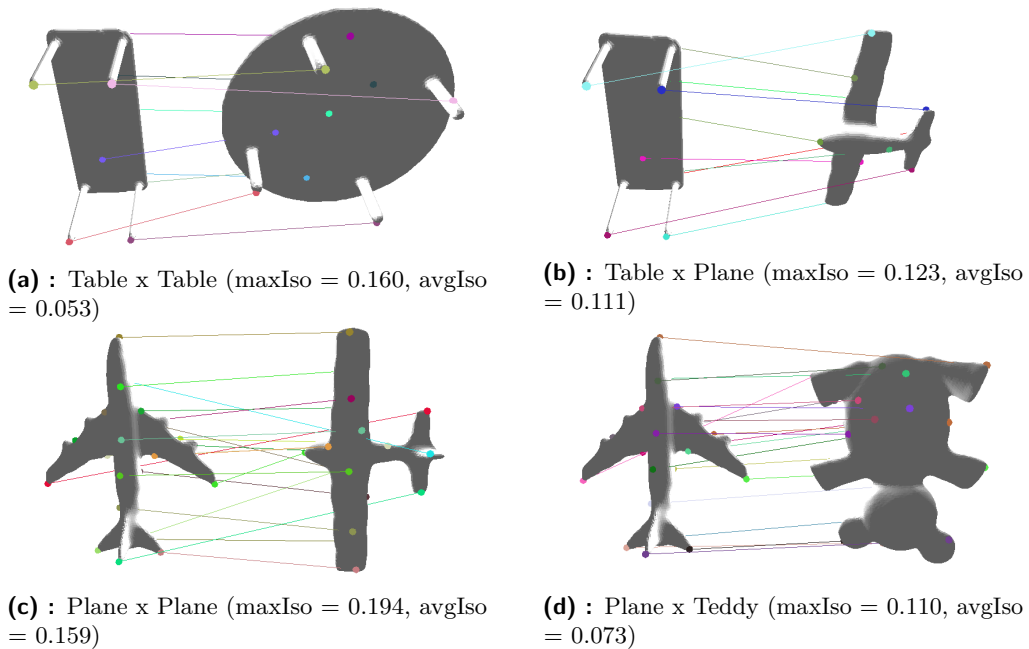


Figure 5.7: Examples of mislabeled pairs. In the first row, the table is labeled as a plane by the $maxIso$ metric (Figure 5.7b). After switching to $avgIso$ metric, it is correctly labeled as a table (Figure 5.7a). In the second row, the plane is reported to be more similar to a teddy (Figure 5.7d) than to another plane (Figure 5.7c) by both metrics.

5.6 Summary

In this chapter, we tested three 3D shape matching methods, namely *Möbius Voting*, *Symmetric Flips Tracking*, and *Genetic Algorithms with Adaptive Sampling*. From these three, the Genetic Algorithms have shown to be the fastest with a high result quality. In the other tests, we have shown that the results do not change when one of the objects is scaled and rotated, which is a very convenient feature — we do not have to transform the objects prior to the matching for them to have similar sizes or positions. The last test proved that the selected method can be used to label an unknown object by selecting the most similar object from a list. The test results were satisfactory, leading us to the decision that the Genetic Algorithms method will be implemented as a part of the RRT-LIB pipeline.

Chapter 6

Experimental verification

In this chapter, the proper function of the RRT-LIB method proposed in [Chapter 4](#) is tested. The tests are designed to confirm a correct implementation of the algorithms and validate the expected behavior.

First, we test whether the method of tracking the path diversity proposed in [Section 4.2](#) ([Alg. 6](#)) helps the RRT-IR algorithm to find distinct paths. We also test whether transforming the paths from the library by the ICP algorithm presented in [Section 4.4](#) ([Alg. 7](#)) makes the paths more relevant for the current planning.

Finally, we combine all methods and test the complete RRT-IR planning phase by running the planner in the test environments. Some test results contain runtimes, however, measuring the precise runtime is not the purpose of the tests presented in this chapter. Due to the overhead introduced by logging and saving planner progress, the runtimes are only indicative but are still included to give the reader a notion of the time required. An extensive comparison with other planners is presented in [Chapter 7](#).

6.1 Data preparation

All implemented methods are tested using objects from the PSB dataset [\[32\]](#) and manually created maps. Objects and maps are represented as triangulated meshes. To make the results comparable, each object is scaled so that its bounding box fits into a cube with an edge size of 2 map units. From three categories, 11 objects in total were selected: four desks¹, four chairs, and three teddy bears. Subscripts will be used when referencing a specific model in the text (e.g., desk₁ and chair₃). The models are shown in [Figure 6.1](#), along with their bounding box dimensions.

Width and depth of each map is 10 units, the height is 5 units. In the first two scenarios, the wall contains one window that is traversable by all objects. This is to test the correct implementation of the planning algorithms and to provide a benchmark for further testing. In the third scenario, the wall contains three windows, each traversable by only some test objects. To increase the planning difficulty, a single test with two walls 3 units apart is introduced. The maps are shown in [Figure 6.2](#), where the window dimensions are given in (x, y, z) format. Each window has a depth of 1 unit (depth of the walls), the width and height are specified by the y and z dimensions, respectively.

¹The word “desk” is used instead of “table” to prevent possible ambiguities with data tables.

6.2 Results

The following algorithm implementations were used for the tests.

To detect collisions (needed in [Alg. 5](#), line 13), the *Robust and Accurate Polygon Interference Detection* (RAPID) library was used [33]. This library allows for fast collision checking between objects represented as triangulated meshes. The nearest point from a list to a given point needs to be found in [Alg. 5](#) (lines 12 and 17) and [Alg. 7](#) (line 6). A naive implementation of the nearest neighbor search (i.e., computing the distance to each tree node and selecting the node with the lowest value) would be extremely slow and inefficient. Instead, k-d trees [34] are used — a multidimensional data structure used for efficient k-nearest neighbor searches. The *nanoflann* library [35] was selected due to its high speed, easy integration and modification, and popularity. Many open-source implementations of the Iterative Closest Point algorithm used in [Alg. 8](#) (described in [Section 4.4](#)) exist. In this work, the library *LIBICP* [36] is used.

6.2.1 Generating guiding paths

The manipulated object is first scaled down to find the guiding paths. Making the object smaller will increase the planning speed and the path diversity [6]. To evaluate the efficiency of the diversity tracking method proposed in [Alg. 6](#), 20 paths for the scaled-down desk₁ are generated first without evaluating their similarity. The planner is then run again, with the paths filtered by their similarity (the parameter $n_{\text{diversity_patience}}$ set to 10). The progress of the distinct path count with increasing iterations is shown in [Figure 6.3](#), and the comparison to the method without filtering is made in [Table 6.1](#). Found paths are visualized in [Figure 6.4](#).

It is also worth mentioning that even though some paths look similar in the 3D visualization, they might be different due to the rotation being also included in the metric evaluation as defined in [Section A.1](#) (see [Figure 6.5](#)). This behavior turns out to be useful because often the manipulated object can pass through the window only under a very specific rotation, which could be different than the scaled-down object rotation. Therefore, having more paths increases the probability of finding the right one allowing the object to pass through.

	All paths	Filtered by diversity
Total found paths [-]	20	44
Kept paths [-]	20	9
Cumulative time [s]	24.53	33.51
Average time per path [s]	1.23	0.73
Visualization	Figure 6.4a	Figure 6.4b

Table 6.1: Guiding paths search runtime for a search without filtering (left) and with filtering (right). The runtime is affected mainly by outliers (i.e., runs taking exceptionally long to finish). In this case, three runs taking over two seconds have occurred during the search without filtering, compared to four that occurred during the search with filtering. This random nature is the cause of the average time per run being lower for the search with filtering.

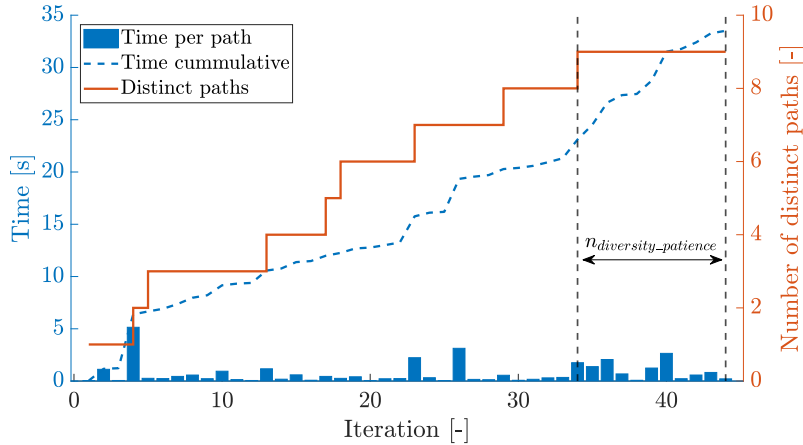
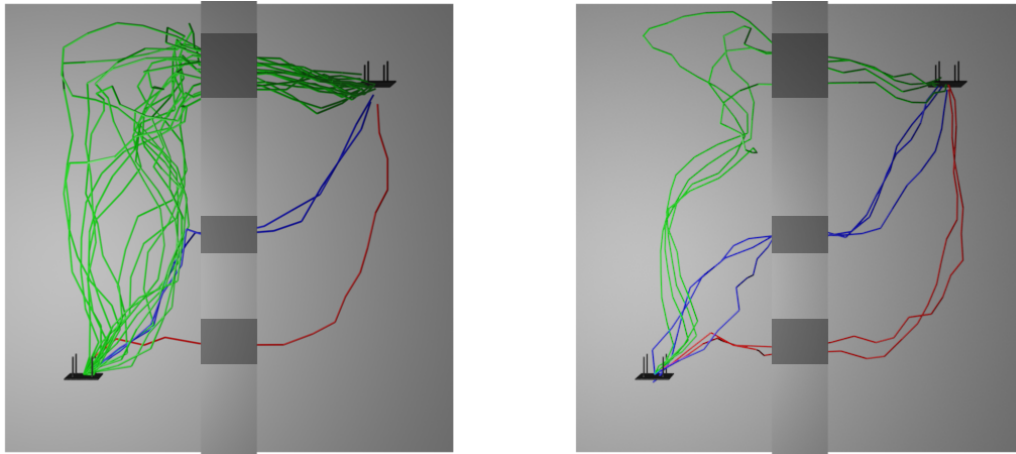
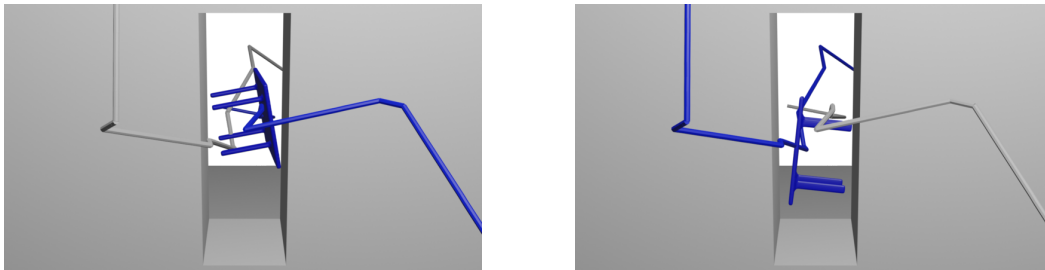


Figure 6.3: As the planner continues searching for new paths, the distinct paths are added to the guiding path set. After multiple (9) distinct paths are found, no new paths are classified as distinct in the last $n_{diversity_patience}$ (10) steps, and the search is stopped.



(a) : Without filtering ($n_{guiding_paths} = 20$). (b) : Filtered by diversity ($n_{diversity_patience} = 10$).

Figure 6.4: Guiding paths were generated for a scaled-down desk₁ through a map with three windows in the wall (map₃, Figure 6.2c), first with the number of paths specified prior to the search (6.4a), then with filtering the paths by their diversity (6.4b). Keeping only the distinct paths results in fewer paths in total, which will increase planner efficiency when planning for similar objects.



(a) : The first path.

(b) : The second path.

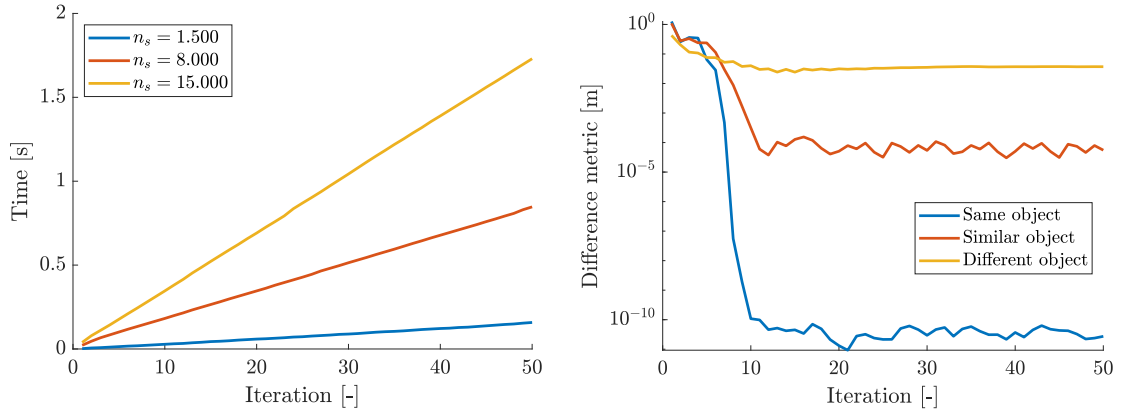
Figure 6.5: Even though the two paths 6.5a and 6.5b are similar in 3D, they are different in 6D due to the object rotation.

6.2.2 Using ICP to transform the similar object

In the previous section, it was assumed that the two objects (the original, for which the guiding paths were computed, and the similar, for which a path is searched) would be represented in the same reference frame and, moreover, would be positioned similarly in this frame. In this section, we show that when this assumption does not hold, the raw guiding paths from the library cannot be used unless a transformation between the two objects is found and the similar object is transformed accordingly.

First, the time complexity of the ICP algorithm needs to be analyzed. Because it is a part of the RRT-LIB planning phase, the time needed to transform the paths from the library into the guiding paths is counted towards the total planning time. Therefore, ICP needs to be fast enough not to substantially affect the resulting search duration. The time complexity depends on the number of vertices in the *source* mesh n_s , vertices in the *target* mesh n_t and iterations n_i . In each iteration, the closest point in the *target* mesh to each point in the *source* mesh needs to be found. By utilizing kd-trees, the nearest neighbor search complexity reduces to $\mathcal{O}(\log n_t)$ [37] for one vertex. Therefore, the time complexity of one iteration is $\mathcal{O}(n_s \log n_t)$ and the total time complexity is $\mathcal{O}(n_i n_s \log n_t)$. This relationship can be observed in Figure 6.6a. We see that even for a mesh with a large number of vertices (15 000), the transformation is found in a matter of seconds, which is acceptable.

The result error is calculated as the sum of squared distances between corresponding vertices (Alg. 7, line 8). In Figure 6.6b, we see that ICP is able to correct a small transformation of the same object while also being able to transform a similar object to fit the original object in around 15 iterations.



(a) : The ICP runtime linearly increases with the increasing number of iterations. For this experiment, the same *target* mesh with $n_t = 10000$ has been used for all tests.

(b) : The difference metric (in mesh units) gradually decreases and converges in the first 15 iterations. For the same object (blue), zero error is achieved. Final errors for a similar object (red) and a completely different object (yellow) are higher. This can be expected since the transformation is isometric and cannot deform the *target* object in any way.

Figure 6.6: Testing the ICP time (6.6a) and result quality (6.6b) over time.

To determine how the result is affected by the initial transformation, another test is performed. The magnitude of a transformation can be determined by adding the magnitude of the translation and the rotation making up the specific transformation. In order for the magnitude

of rotation to be measured, the axis-angle rotation representation is used. In Figure 6.7, we see that without the initial guess, the error increases when increasing the transformation magnitude. When the initial guess extracted from the similarity evaluation presented in Section 4.3 is used, the results are greatly improved, as shown in Figure 6.8. An illustration is shown in Figure 6.9.

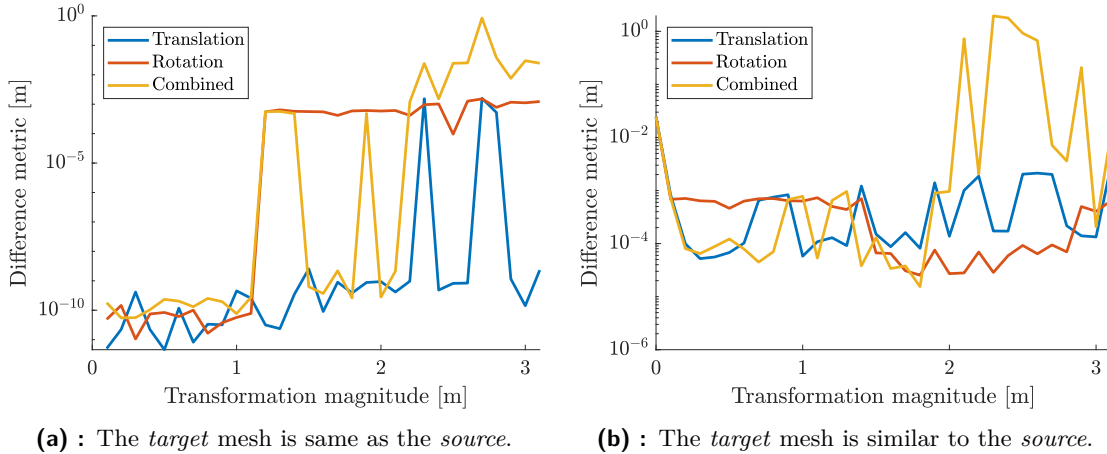


Figure 6.7: The result difference metric increases with increasing transformation magnitude for the same object (both in mesh units). The spikes show that ICP is sensitive to initial conditions.

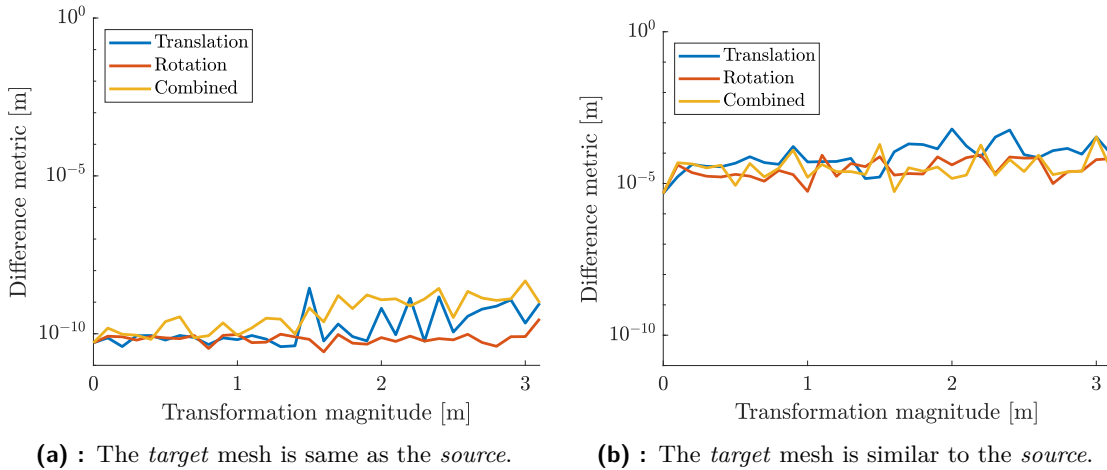


Figure 6.8: When an initial guess of the transformation obtained from the similarity evaluation is provided, ICP results are better and more consistent.

A similar object to the original object used in Section 6.2.1 is selected and its mesh is transformed by a random transformation. Thanks to the fact that the similarity evaluation method is invariant to affine transformations, as shown in Section 5.4, the corresponding vertices are not affected by the random initial transformation. Using methods from Section 4.4, a transformation mapping the similar object to the original is found (visualization is shown in Figure 6.10). The planner is first run without using the found transformation (Figure 6.10a). Due to the difference in position and rotation, the guiding paths do not guide the object through the windows and go through the wall instead (Figure 6.11a). As expected, no solution is found in the 2-minute limit. Then, the similar object is transformed prior to performing a search (Figure 6.10b). Transforming the object makes the guiding path relevant (Figure 6.11b), and a solution is found very quickly (in under two seconds).

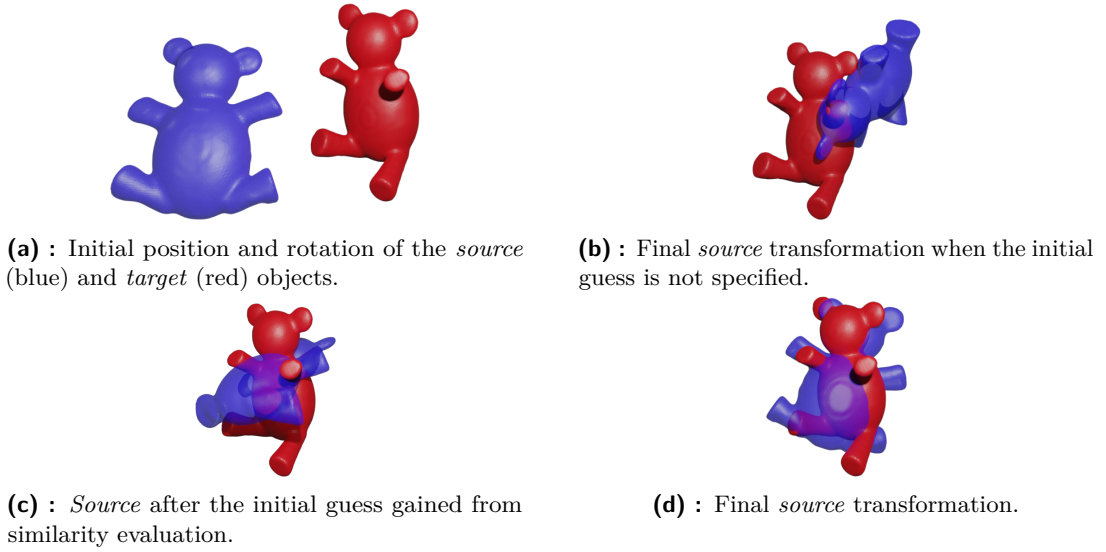


Figure 6.9: Illustration of the ICP algorithm in 3D with additional knowledge gained from the similarity evaluation. The *source* mesh (in blue) and the *target* mesh (in red) are provided as the input, along with a list of the corresponding vertices (6.9a). First, a transformation minimizing the distance between corresponding vertices is found and used as an initial guess of the transformation (6.9c). Then, ICP iteratively tries to find a transformation that maps the *source* mesh to the *target* mesh (6.9d). Without the initial guess, the results are much worse (6.9b).



Figure 6.10: The initial and final position of the original (red) and the similar (blue) object. The guiding paths were computed for the original object. To use them, the similar object needs to be mapped to the original and transformed accordingly.

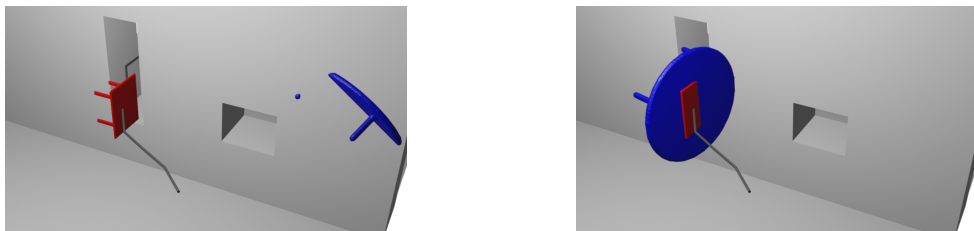


Figure 6.11: When the mutual position and rotation of the guiding object (red) and the similar object (blue) are not taken into account, the guiding paths become useless because they guide the similar object through the wall instead of the middle window (6.11a). After the guiding paths are transformed, planning along them becomes viable (6.11b).

6.2.3 Finding paths for similar objects

After the guiding paths for the scaled-down desk_1 are generated, we can use them to guide the planner searching for a path for similar, fully scaled objects. desk_2 and desk_3 are used in this example. Both can fit through the middle window, desk_2 can also tightly fit through the larger (top-left) window. The results are presented in Table 6.2 and in Figure 6.12. We see that the planner successfully finds a path in a reasonable time. When the planner is run without using the guiding paths, the planning is terminated after two minutes without finding a path.

	desk_2	desk_3
Runtime [s]	26.58	1.42

Table 6.2: Path search runtime guided by already found paths.

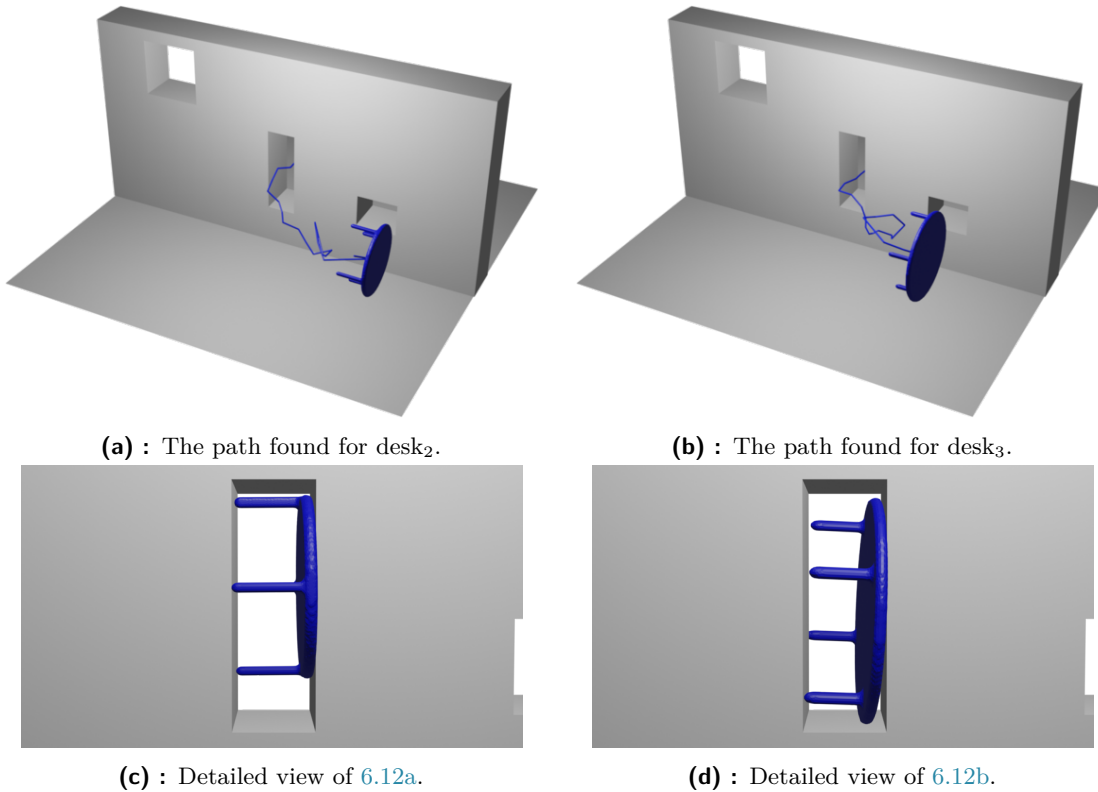


Figure 6.12: A valid path is found in a reasonable time even though the objects desk_2 and desk_3 barely fit through the window, thanks to the paths found for a similar, scaled-down object (desk_1). Without the guiding paths, it would be very hard for the planner to find a path through.

6.2.4 Specifying a guiding path

By specifying one guiding path, we can effectively choose a path for the manipulated object to follow. As an example, one path for each of the three windows is selected and used as a guiding path for the desk_4 . Even though the guiding path is specified, there exists a nonzero probability that a path not following this guiding path will be found — this probability is

affected by the p_{bias} value. By setting p_{bias} to 1.0, the probability of finding a path similar to the guiding path should be theoretically maximized. However, since the original object was scaled down and had a different shape, setting p_{bias} to 1.0 (forcing the planner to plan only around the guiding paths) could result in a failure to find a path. Practically, values around 0.8 – 0.9 yield better results. The experiment results are shown in Figure 6.13 and Table 6.3.

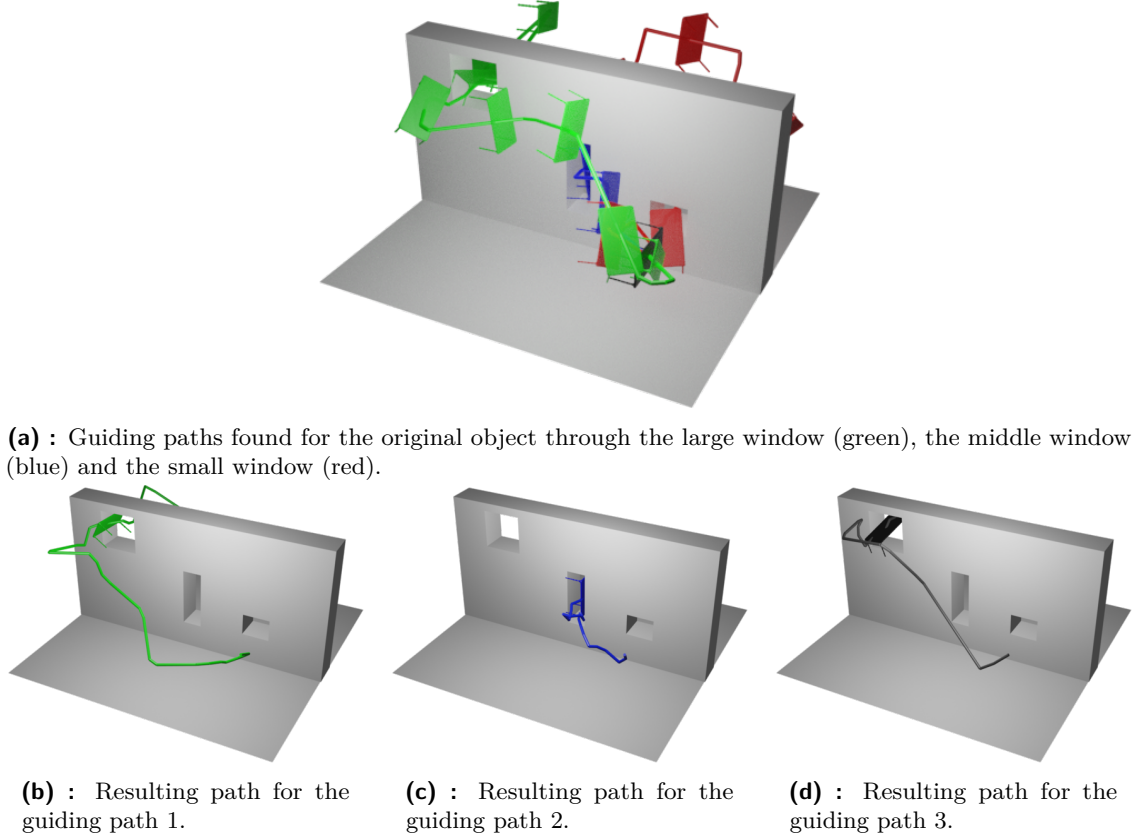


Figure 6.13: By specifying a guiding path from the paths found for a similar object (6.13a), the path for the manipulated object can be successfully selected most of the time (6.13b and 6.13c). However, due to a possible difference in the required rotation or the existence of easier paths, the planner can still find a path different from the guiding path selected (6.13d).

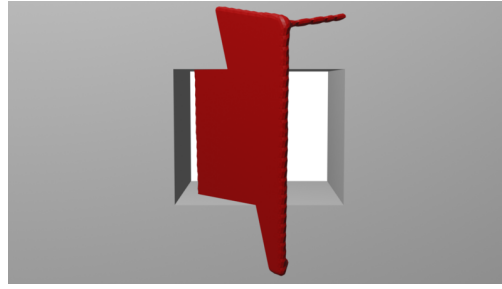
	Path 1	Path 2	Path 3
Runtime [s]	0.80	0.73	5.46

Table 6.3: Runtime of finding the resulting paths.

When searching along paths 1 and 2 (the largest and the middle window), we see that a path is found in a short time. However, when the third guiding path was used (the smallest window), the planner was not able to find a solution and went through one of the larger windows instead. By visualizing and comparing the configuration of the guiding and the manipulated object along the guiding path (Figure 6.14), we can understand why a path could not be found. Even though the guiding path went through the smallest window, the rotation of the guiding object is far from the rotation needed for the manipulated object to pass through (as shown in Figure 6.5). By manually selecting a guiding path with a correct rotation in addition to going through the smallest window and setting p_{bias} to 0.99, the planner can find a path through the smallest window in a reasonable time of 12.30 s (Figure 6.15).

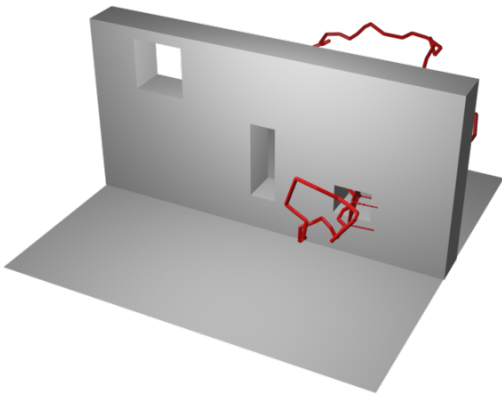


(a) : Rotation of the guiding object following the guiding path.

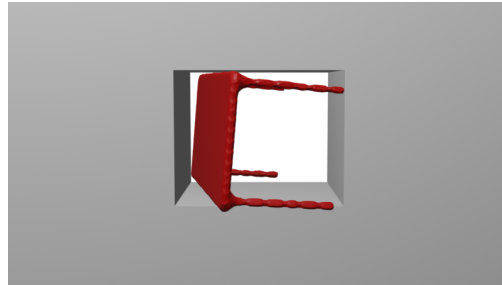


(b) : Rotation of the manipulated object following the guiding path.

Figure 6.14: Planner guided by a selected guiding path is unable to find a solution. This happens because the rotation needed for the manipulated object to pass through the smallest window is substantially different from the rotation proposed by the guiding path.



(a) : The path found for desk₄.



(b) : Detailed view of 6.15a.

Figure 6.15: By manually selecting a guiding path going through the smallest window with the correct rotation, the desired path is found (6.15a), even though the manipulated object barely fits through the window (6.15b).

6.3 Summary

In this chapter, we tested the correct implementation and behavior of all parts of the RRT-LIB planner. Although the tests confirmed our initial ideas, they can not provide us with the confirmation of improved efficiency compared to other state-of-the-art planners. This comparison is presented in the following chapter.

Chapter 7

Comparison with other planners

To compare the performance of our RRT-LIB planner with other planners, The Open Motion Planning Library (OMPL) [38] and its benchmarking facilities [39] were used. OMPL is an open-source software library for testing various sampling-based planners. In recent years, it has become widely used for comparing new sampling-based planners in the literature [40, 41, 42]. Our planner needed to be rewritten using data structures from the OMPL library (state representation, nearest neighbors, etc.). A collision checking function also needed to be implemented, taking our data as input and determining the validity of the sampled states.

By implementing the planner directly into the library and compiling from the source, the results are not affected by different compilers and possible optimization. Moreover, using the same validity checking function and nearest neighbor libraries eliminates the influence of external libraries on the planner results.

7.1 Selected planners

The OMPL library offers multiple implemented and preconfigured planners¹, which will be used in the benchmark. Among the algorithms are Rapidly-exploring Random Trees (RRT) described in Section 2.2.2 along with multiple modifications. The standard Probabilistic roadmaps (PRM) method described in Section 2.2.1 could not be used due to the incompatibility of the implementation with the compiler used. Therefore, only a modified variant (LazyPRM) is included. Compared algorithms are listed in Table 7.1.

More available planners could be used in the benchmark. However, it needs to be considered that many algorithms do not terminate immediately when a path is found. Instead, algorithms such as RRT* or SPARSE aim to find the optimal (typically the shortest) path by improving the solution until a satisfactory one is found or a specified time limit is reached. Because we are mostly interested in the runtime, the comparison with optimizing planners is not relevant as they have a different planning objective.

¹The list of available planners can be found at <https://ompl.kavrakilab.org/planners.html>

Abbreviation	Name
RRT-LIB	RRT with Library of Trajectories
RRT	Rapidly-Exploring Random Trees [4]
RRTConnect	Bidirectional RRT [43]
Lazy RRT	Lazy vertex and edge evaluation RRT [44]
LazyPRM	Lazy vertex and edge evaluation PRM [45]
KPIECE	Kinematic Planning by Interior-Exterior Cell Exploration [46]
BKPIECE	Bidirectional KPIECE [46]
LBKPIECE	Lazy vertex and edge evaluation KPIECE [46]
EST	Expansive Space Trees [47]
BiEST	Bidirectional EST [47]
SBL	Single-query Bi-directional Lazy collision checking planner [48]
STRIDE	Search Tree with Resolution Independent Density Estimation [49]

Table 7.1: Planners used in the benchmark.

7.2 Parameter and environment setup

The parameter settings for our RRT-LIB planner are listed in Table 7.2. The default² configuration is used for all other planners.

Parameter	Name	Value
p_{goal}	Goal bias	0.05
p_{bias}	Path bias	0.80
d_{guide}	Guiding radius	0.50
$d_{inhibited}$	Inhibited radius	1.20
d_{safe}	Safe distance	0.80

Table 7.2: Default parameters for the RRT-LIB planner

For each object and map pair, a benchmark test is run. Each planner is run 10 times with a 2-minute time limit, computing the average time after all 10 runs have finished. Since we are benchmarking 12 planners on 22 scenarios³, with each planner run 10 times for a maximum of 2 minutes for each scenario, the upper time estimate for the benchmark is $12 \cdot 22 \cdot 10 \cdot 2 = 5280$ min = 88 h. To make such computation more manageable, the benchmarks were run on a computing grid *MetaCentrum*⁴, allowing to run each benchmark scenario independently. The specifications of one computational node are listed in Table 7.3, along with the resources reserved for each job.

²As of version 1.5.2 available at <https://github.com/ompl/ompl/releases/tag/1.5.2>.

³11 objects in 2 maps

⁴<https://www.metacentrum.cz/cs/>

Parameter	Value	Reserved for one job
CPU	4x14 CPU Intel Xeon Gold 5120	1 thread
RAM	768 GB RAM	8 GB
OS	Debian10	—

Table 7.3: Specifications of one MetaCentrum node used to run the benchmarks.

7.3 Preparation

Before running the benchmarks, we need to generate the guiding paths and save them in the library (the RRT-LIB preparation phase, Alg. 6). From each object category (desk, chair, and teddy, as described in Section 6.1), one object is selected as the category representative. For scaled-down⁵ versions of the objects, the guiding paths are computed in two maps: map₃ and map₄. These maps were selected because they offer various paths through the wall, and the window dimensions are designed to make the problem challenging enough, but still solveable. The number of guiding paths for each map and guiding object are listed in Table 7.4 and visualized in Figure 7.1. We see that our proposed method of generating guiding paths is able to generate distinct paths quite well. However, no guiding path was found through the smallest window in Figure 7.1b and through the middle window in Figure 7.1f. Missing a possible path through the environment is one of the drawbacks of using a sampling-based planner to generate the guiding paths. The guiding path generation takes approximately 20 seconds per object and map.

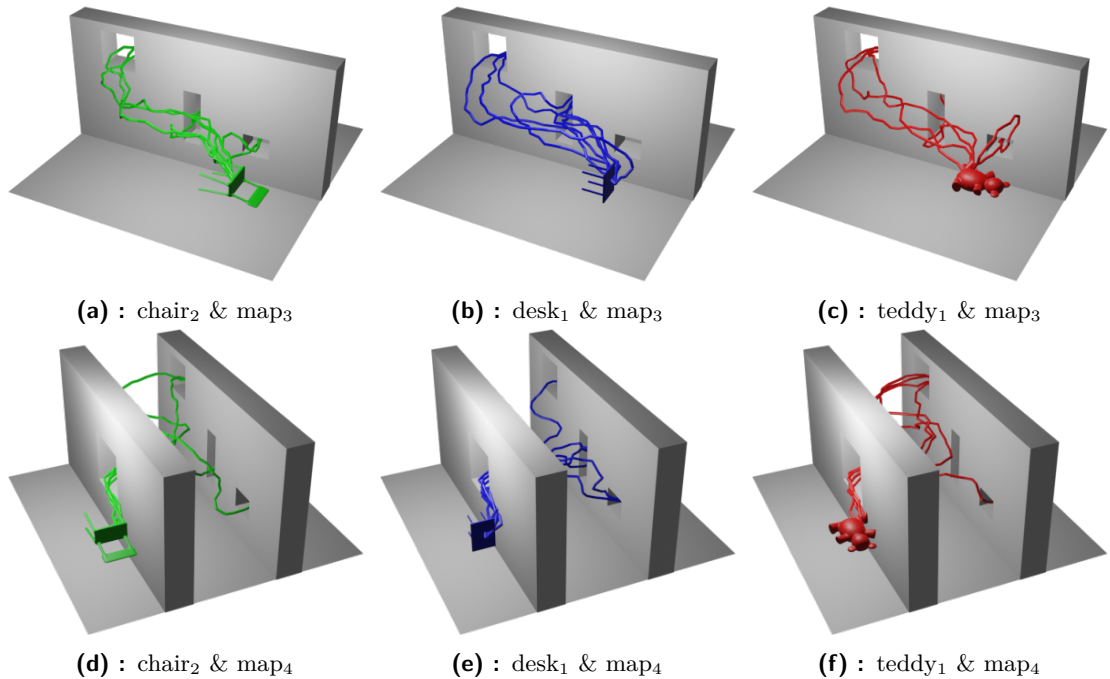


Figure 7.1: Guiding paths for the three guiding objects. Generated as a part of the RRT-LIB preparation phase.

⁵Objects were scaled-down to 40% of their original size.

Object	Guiding paths	
	map ₃	map ₄
chair ₂	7	4
desk ₁	7	6
teddy ₁	6	5

Table 7.4: Number of computed guiding paths.

7.4 Benchmark results

The guiding paths were computed for scaled-down objects by our RRT-LIB planner and stored in the library. In each benchmark scenario, a query object and a map are given as inputs to the planners. At first, our method selects the most similar object from the library and computes the mutual correspondences. The similarity evaluation takes on average one second per object in the library — therefore, in our case, it adds approximately 3 seconds to each planning task. The correspondences for three query objects are illustrated in Figure 7.2. In all cases, the genetic shape correspondence algorithm [22] presented in Chapter 5 successfully selected the guiding object belonging to the same class as the query object. The paths computed for the similar object are retrieved from the library, and a transformation between the similar object and the query object is found by ICP to ensure the objects have a similar position and rotation (Section 4.4). Finding the transformation using ICP is not time demanding and it takes less than a second to compute. The paths from the library are then used as the guiding paths for the planner, increasing the sampling probability along them (Alg. 5). If no solution is found after two minutes, the run is terminated.

The result of each benchmark test is presented on the following pages, with a bar graph representing the success rate of each planner (in how many of the ten runs was a path successfully found) and a box graph containing the time needed to find a solution (capped at the 2-minute threshold). It needs to be noted that the time needed to precompute the guiding paths (approximately 20 seconds) is not accounted for in the graphs. Moreover, due to the OMPL implementation, only the time used by the planner is measured and shown in the graph. Therefore, the overhead introduced by selecting the similar object from the library and transforming the object (~ 3 s) needs to also be taken into account.

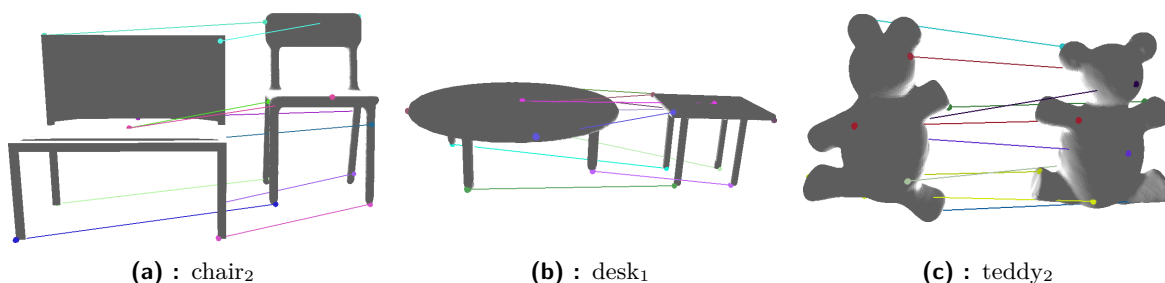


Figure 7.2: For every query object (left on each image), the most similar object from the library is selected (right on each image). The correct guiding object was selected every time. The correspondences between the objects are illustrated by the colored lines.

7.4.1 Easy scenarios

In the easier benchmarks (Figure 7.3 and Figure 7.4), most of the tested planners are able to find a solution within the time limit. The runtime of our planner is similar to the runtime of the standard RRT planner.

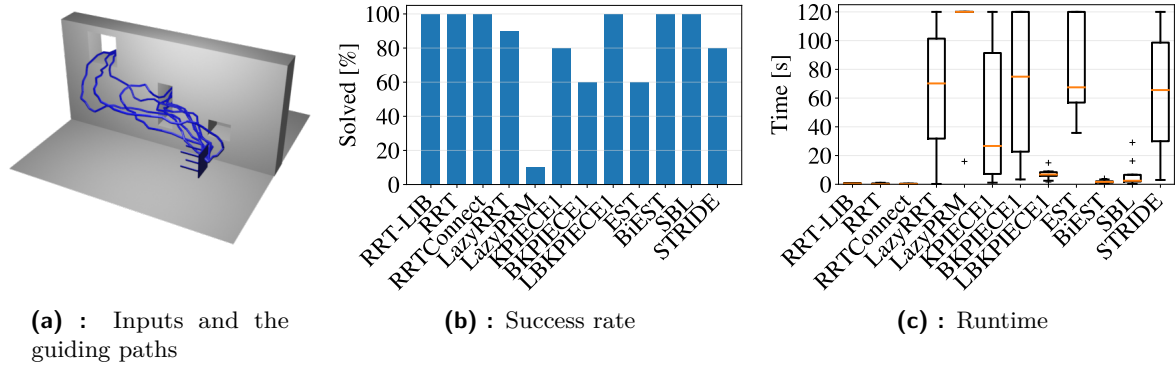


Figure 7.3: desk₁ & map₃ (easy)

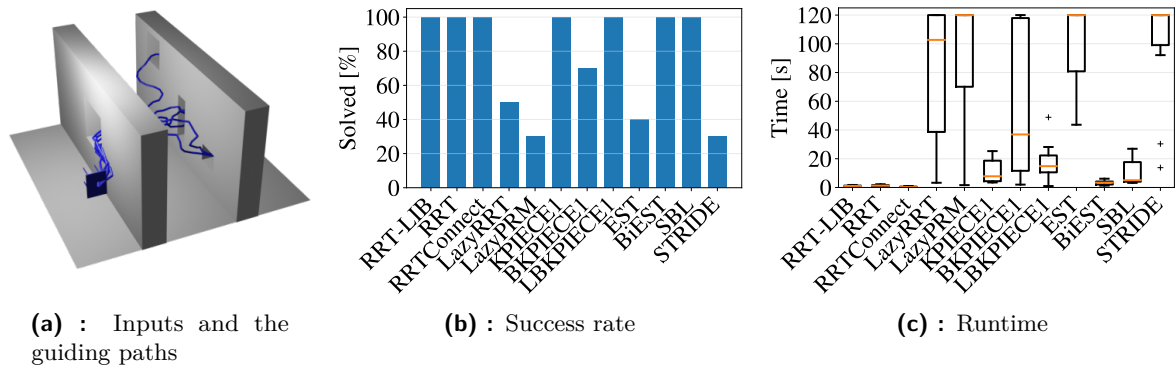


Figure 7.4: desk₁ & map₄ (easy)

7.4.2 Medium scenarios

By using larger objects, the difficulty is increased. The benchmarks with medium difficulty are shown in Figure 7.6, Figure 7.7, and Figure 7.8. Our planner retains the 100% success rate, and the results show that guiding paths for a similar object usually help to achieve greater speed and consistency, compared to the standard RRT. However, in some cases (Figure 7.8), RRT achieves faster runtime. This can happen when the guiding paths are far from the possible paths of the query object. Detailed view of three runtime results is shown in Figure 7.5. Considering the time added by generating the guiding paths in the preparation phase and selecting the most similar object from the library, using one of the other planners would be faster than using our planner in some cases.

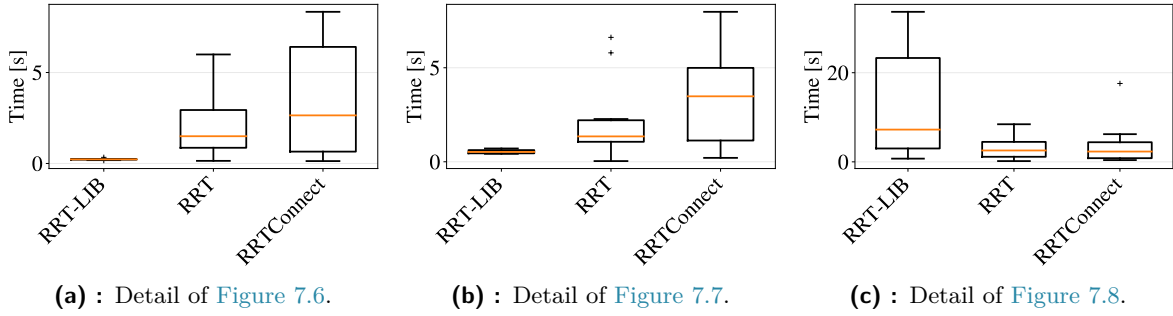


Figure 7.5: Detailed view of three benchmark results. The runtime of our RRT-LIB planner is compared to the standard RRT and the bidirectional variant (RRTConnect).

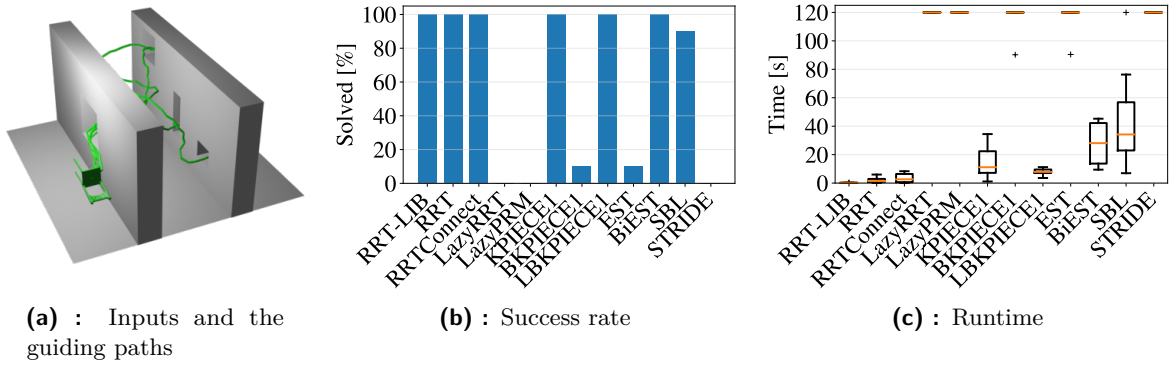


Figure 7.6: chair₁ & map₄ (medium)

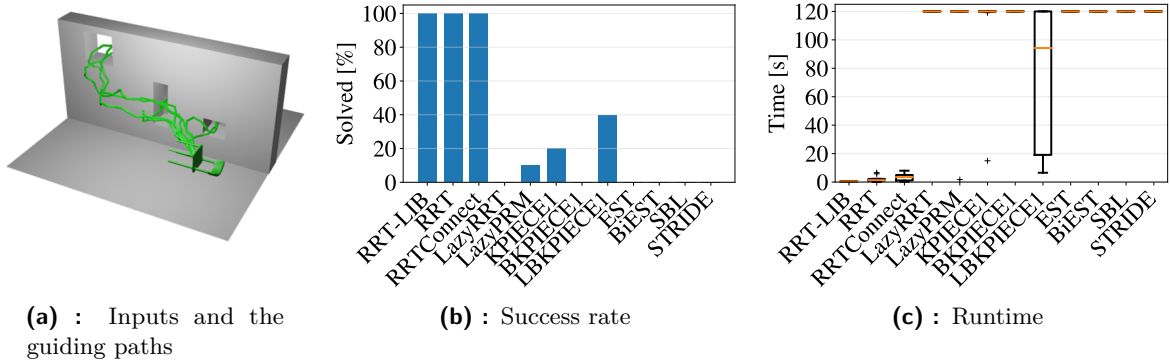


Figure 7.7: chair₄ & map₃ (medium)

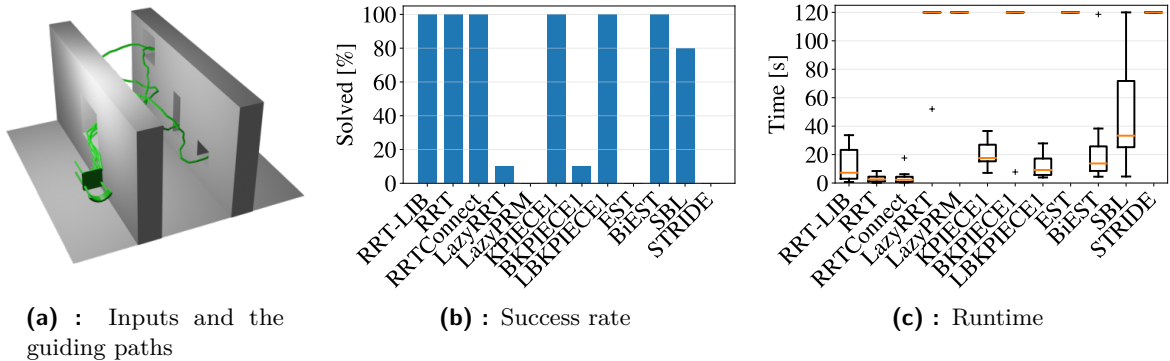


Figure 7.8: chair₄ & map₄ (medium)

7.4.3 Hard scenarios

Tests depicted in [Figure 7.9](#), [Figure 7.10](#), [Figure 7.11](#), and [Figure 7.12](#) are substantially harder and show the real advantage of using guiding paths. The objects barely pass through only one of the windows. Therefore, it is hard for the planners to find a path through. We can see that our planner is still able to find a solution most of the time, while the other planners fail every time. In [Figure 7.11](#), our planner failed to solve the problem in the time limit during two of the ten runs. This can be attributed to the randomness of the sampling-based planners. Even though the guiding paths guide the expansion of the tree, the sampling is still random, and situations where a solution is not found in a specified time can occur. In the other tests, the success rate remains 100%.

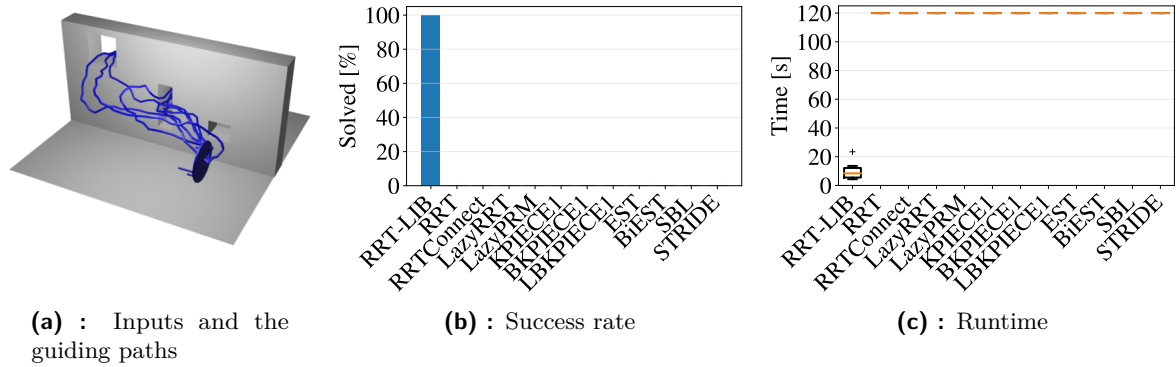


Figure 7.9: desk₂ & map₃ (hard)

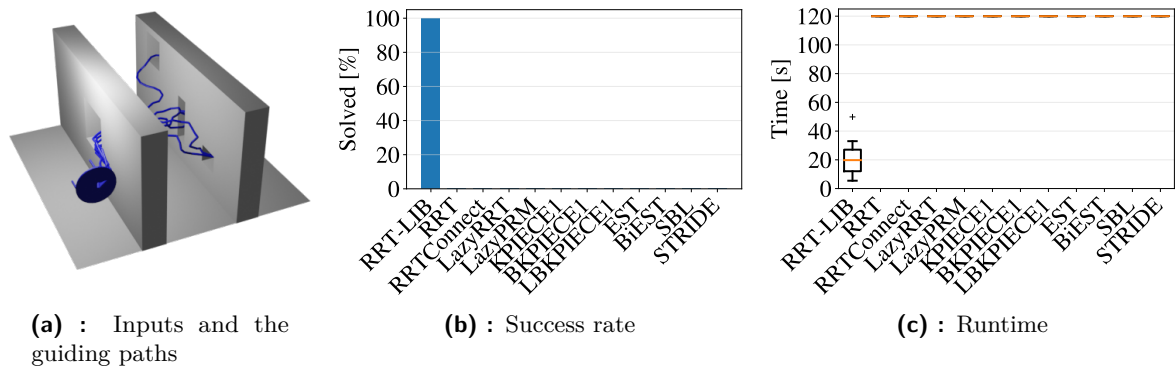


Figure 7.10: desk₂ & map₄ (hard)

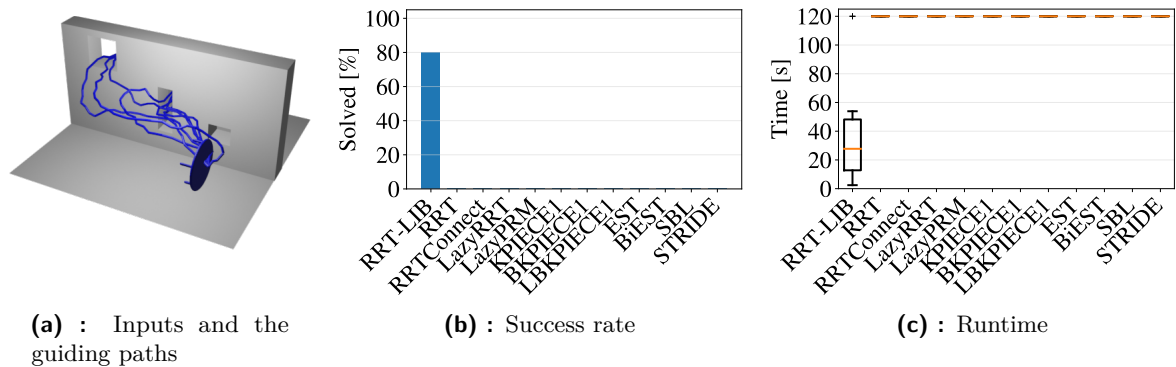
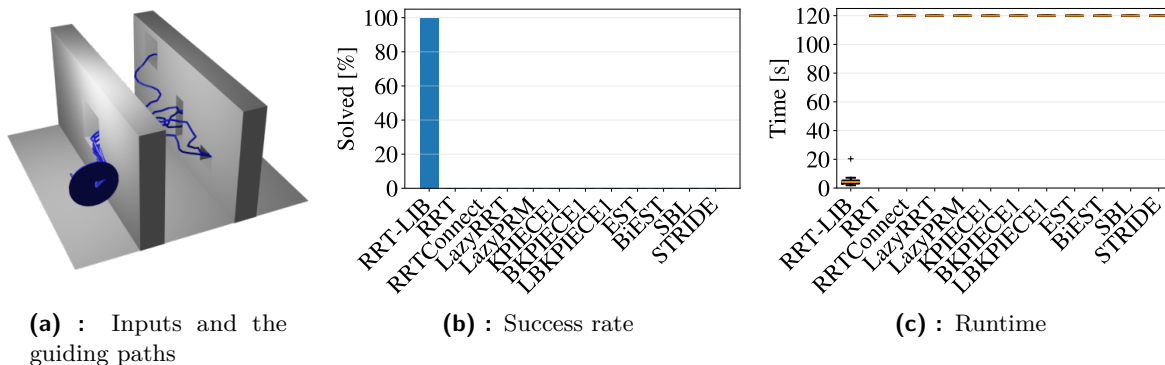


Figure 7.11: desk₃ & map₃ (hard)

Figure 7.12: desk₃ & map₄ (hard)

7.5 Summary

In this section, we compared our RRT-LIB planner with other state-of-the-art planners. The comparison was made using the benchmarking facilities of the OMPL library [39]. We introduced tests with an increasing level of difficulty: easy, medium, and hard.

The RRT-LIB pipeline consists of multiple steps. In the preparation phase, the guiding paths for multiple objects are computed. In the planning phase, a similar object from the library needs to be selected first. Then, a transformation between the query object and the object from the library is found by the ICP algorithm. Once the transformation is found, the sampling-based planning begins.

The overhead introduced by the various steps in the pipeline needs to be considered when evaluating the usefulness of the RRT-LIB planner given a specific task. When the problem is simple, the time required to select the similar object and find the mutual transformation is considerably longer than the time needed for the planning. Even though the planner finds the solution almost immediately thanks to the guiding paths, no significant amount of time is saved compared to the standard RRT planner.

The advantage of using the guiding paths computed for similar objects is shown in the medium and hard benchmarks. In most cases, our RRT-LIB planner achieves faster runtime compared to the other state-of-the-art planners. Moreover, it successfully finds paths even in the tasks in which the other planners fail.

However, the library needs to contain computed paths, and the computation of guiding paths in the RRT-LIB preparation phase requires a considerable amount of time. Making such an effort is reasonable only when we plan for similar objects in the same environment repeatedly. If we know that the planning will be performed multiple times with a defined set of static environments and object classes, devoting the time to compute paths to store in the library will prove advantageous in the long run. However, when the planning task is unique and will not be repeated, it is better to use other approaches.

Chapter 8

Conclusion

This thesis dealt with the task of motion planning for 3D objects. Motion planning is an NP-hard problem and is most commonly addressed by the sampling-based methods [4]. However, sampling-based planners struggle with situations in which a narrow passage is present. In such environments, guiding paths (i.e., available paths through the environment) can be used to improve the planning speed [6].

We introduced a novel RRT-LIB algorithm that aims to improve the efficiency of sampling-based planners by creating a library which stores already computed paths in an environment with narrow passages. These paths are then used as guiding paths for other, similar objects. To retrieve a set of paths that has the highest potential of correctly guiding the planner, a query system able to choose the most similar object to a given object was designed. By using paths that were computed for a similar object, the planning can get considerably faster.

The first step is to understand and correctly define the areas of concern. Introduction to motion planning and shape matching was presented in [Chapter 2](#) and [Chapter 3](#), along with current approaches to tackling these problems.

The Rapidly-Exploring Random Trees with Inhibited Regions (RRT-IR) [6] proved to be a robust algorithm for both path generation and guided planning. Using the RRT-IR capability to generate guiding paths allowed us to design and implement a library containing precomputed paths for objects through a specified environment, as described in [Chapter 4](#). To increase the efficiency of the library, a novel method to evaluate and track the path similarity was proposed. By tracking and filtering paths by their diversity, we were able to quickly find different guiding paths through the environment. This variability consequently helped us with finding a path for similar objects. However, implementing a more complex path filtering method, such as methods based on topological clustering, could lead to improved results in more types of environments.

To select the most similar object in the library, a state-of-the-art 3D shape matching algorithm was used. This gave us the ability to decide which known object from the library is the most similar to an unknown object, also outputting a list of corresponding vertices between the two objects. To ensure that the original and the similar object are positioned similarly in the coordinate frame used, a transformation between them is found by the ICP algorithm. This transformation is also used to modify the guiding paths.

Multiple tests were proposed in [Chapter 5](#) to choose a specific shape matching method. In the end, a method utilizing Genetic Algorithms [22] was used, which for two triangulated meshes outputs a similarity metric and a list of corresponding vertices. This approach turned out to be both fast enough and precise enough to allow further development. In comparison, the

other two tested methods were both very slow for meshes with a higher number of vertices and the matching quality was comparable to or even worse than the Genetic Algorithms method.

In [Chapter 6](#), we showed that our methods are able to leverage the prior knowledge about the environment in the form of precomputed guiding paths. Moreover, being able to specify the guiding paths and choose favorable object paths can be useful in numerous fields, including robotics.

After verifying the correct behavior of our algorithm, it was compared to other state-of-the-art path planning methods to see whether we have achieved some improvements. For that, a third-party open-source planning library OMPL [\[38\]](#) was used. The benchmarks have shown that having multiple precomputed paths through the environment for a similar object can substantially decrease the time needed for planning. More importantly, our RRT-LIB planner is able to find paths even in the tasks in which the other planners fail. The results are presented in [Chapter 7](#).

Due to the fact that computing the guiding paths in the RRT-LIB preparation phase takes a considerable amount of time, our approach is suited for situations when the planning is repeated in the same environment. When the task is to plan in a given environment once, devoting the time to build the library is unnecessary, and other approaches will yield better results. However, when we know ahead that the planning will be conducted multiple times in a set of static environments, filling the library with paths prior to planning can lead to increased speed and success rate of the planner.

Appendix A

Appendix

A.1 Metric

A metric definition is needed for motion planning. The metric is a real valued function ρ measuring the distance between two configurations in \mathcal{C} . The most common metrics come from the L_p metric family, defined for \mathbb{R}^n and $p \geq 1$ as

$$\rho(x, y) = L_p(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (\text{A.1})$$

For $p \in \{1, 2, \infty\}$

1. L_1 : The *Manhattan metric*
2. L_2 : The *Euclidean metric*
3. L_∞ :

$$\rho(x, y) = \lim_{p \rightarrow \infty} L_p(x, y) = \max_{1 \leq i \leq n} \{|x_i - y_i|\} \quad (\text{A.2})$$

The Euclidean metric is commonly used when planning in \mathbb{R}^3 . However, as we work with both translation and rotation, we need to extend the metric to $SE(3)$.

First, a $SO(3)$ metric needs to be defined. Using quaternions, one such metric comes from the angle between two quaternions, computed from their inner product

$$\rho_s(h_1, h_2) = \cos^{-1}(a_1 a_2 + b_1 b_2 + c_1 c_2 + d_1 d_2). \quad (\text{A.3})$$

Because h and $-h$ represent the same rotation, the $SO(3)$ metric is given by

$$\rho(h_1, h_2) = \min\{\rho_s(h_1, h_2), \rho_s(h_1, -h_2)\}. \quad (\text{A.4})$$

When working with Cartesian products of metric spaces ($\mathbb{R}^3 \times SO(3)$ in our case), a metric can be simply defined as a linear combination of the metrics from the underlying metric spaces. Therefore, the distance between two configurations $q_1 = (r_1, h_1)$, $q_2 = (r_2, h_2)$ can be computed as

$$\rho(q_1, q_2) = c_1 \rho_1(r_1, r_2) + c_2 \rho_2(h_1, h_2), \quad (\text{A.5})$$

where ρ_1 is the \mathbb{R}^3 Euclidean metric and ρ_2 is the $SO(3)$ metric defined in Eq. A.5. Coefficients c_1 and c_2 can be any real positive constants (with a common value being 1, which will be used in this thesis) [4].

■ A.2 Object labelling test

Multiple tests were concluded to verify the selected algorithm for selecting the most similar object from the library. The method which uses genetic algorithms was presented in [Chapter 3](#) and selected as a suitable method in [Chapter 5](#). In the implementation used, the *maximum isometric distortion* metric is computed. To see whether this metric can be used to compare objects and label them accordingly, three samples from each category are selected and compared to the category representatives — one for each category. The label of the object for which the *maximum isometric distortion* is minimal is selected as the label for the unknown object.

Using this method, we are able to correctly identify the label of most isometric deformations from the TOSCA dataset ([Table A.1](#)). The only exception happens for the second model of a wolf. However, this model is labeled as a dog, which is still satisfactory. The algorithm failed for the models of a gorilla because the mesh does not form a connected component. The result quality for non-isometric deformations from the PSB dataset is worse ([Table A.2](#)). This is the result of calculating the maximum over the sampled subset of vertices, which can be suitable for isometric or nearly isometric deformations. However, the objects in PSB are not isometric deformations of each other — a more suitable approach is to compute the average over the vertices. Using the *average isometric distortion* (*avgIso*), we see that our results have improved ([Table A.2](#)).





























							
	0.000	0.271	0.207	0.173	-	0.156	0.157
	0.170	0.264	0.212	0.170	-	0.120	0.213
	0.036	0.261	0.167	0.209	-	0.191	0.210
	0.347	0.021	0.286	0.186	-	0.205	0.424
	0.275	0.081	0.216	0.135	-	0.173	0.251
	0.305	0.055	0.322	0.206	-	0.178	0.209
	0.229	0.314	0.000	0.344	-	0.221	0.179
	0.199	0.307	0.023	0.211	-	0.224	0.191
	0.252	0.265	0.041	0.224	-	0.174	0.231
	0.176	0.233	0.226	0.087	-	0.092	0.121
	0.103	0.257	0.273	0.049	-	0.110	0.115
	0.116	0.210	0.509	0.060	-	0.126	0.073
	-	-	-	-	-	-	-
	-	-	-	-	-	-	-
	-	-	-	-	-	-	-
	0.180	0.182	0.236	0.090	-	0.000	0.104
	0.133	0.209	0.403	0.106	-	0.054	0.149
	0.181	0.242	0.420	0.136	-	0.024	0.088
	0.080	0.360	0.488	0.089	-	0.139	0.060
	0.128	0.387	0.154	0.048	-	0.138	0.075
	0.149	0.177	0.428	0.095	-	0.129	0.038

Table A.1: Labelling test data from the TOSCA database. All objects except one are labelled correctly. The gorilla models are not connected, which is a prerequisite for the algorithm to work.

















											
	0.000	0.222	0.154	0.204	0.185		0.000	0.083	0.082	0.086	0.075
	0.194	0.279	0.169	0.278	0.110		0.159	0.124	0.058	0.068	0.073
	0.227	0.122	0.126	0.057	0.102		0.122	0.110	0.055	0.056	0.089
	0.363	0.000	0.109	0.207	0.121		0.153	0.000	0.054	0.063	0.074
	0.149	0.116	0.152	0.197	0.120		0.112	0.041	0.057	0.047	0.053
	0.248	0.017	0.102	0.239	0.278		0.123	0.015	0.059	0.073	0.082
	0.436	0.205	0.000	0.288	0.237		0.149	0.117	0.000	0.094	0.073
	0.164	0.228	0.094	0.300	0.118		0.127	0.130	0.061	0.098	0.076
	0.123	0.160	0.160	0.296	0.178		0.111	0.092	0.053	0.074	0.059
	0.146	0.194	0.118	0.000	0.154		0.131	0.074	0.096	0.000	0.120
	0.347	0.108	0.204	0.062	0.112		0.179	0.100	0.107	0.054	0.089
	0.214	0.172	0.206	0.100	0.076		0.103	0.105	0.049	0.040	0.043
	0.269	0.143	0.196	0.154	0.000		0.133	0.116	0.063	0.051	0.000
	0.070	0.244	0.213	0.224	0.150		0.065	0.173	0.079	0.092	0.055
	0.115	0.142	0.154	0.163	0.076		0.112	0.103	0.076	0.046	0.037

Table A.2: Using the *maxIso* (left) x *avgIso* (right) metric to label unknown objects. We see that the result improves when using the *avgIso* metric.



Appendix B

Attachments

The attached file `code.zip` contains a collection of the software used in this thesis, along with examples of the data. To install and use the software, follow the respective `README.md` files in the attached folders.

`genetic-algorithms-master/` contains scripts used to evaluate object similarity and correspondences.

`rrt_lib-master/` contains code of the RRT-LIB algorithm, both as a standalone version and as a part of the OMPL library. It also contains used maps and objects.

Appendix C

Bibliography

- [1] J. Kim, K. Jo, D. Kim, K. Chu, and M. Sunwoo, “Behavior and path planning algorithm of autonomous vehicle A1 in structured environments,” *IFAC Proceedings Volumes*, vol. 46, no. 10, pp. 36–41, 2013. 8th IFAC Symposium on Intelligent Autonomous Vehicles.
- [2] H. Heidari and M. Saska, “Collision-free trajectory planning of multi-rotor UAVs in a wind condition based on modified potential field,” *Mechanism and Machine Theory*, vol. 156, pp. 104–140, Feb. 2021.
- [3] V. Vonásek, A. Jurčík, K. Furmanová, and B. Kozlíková, “Sampling-based motion planning for tracking evolution of dynamic tunnels in molecular dynamics simulations,” *Journal of Intelligent & Robotic Systems*, June 2018.
- [4] S. M. LaValle, *Planning Algorithms*. USA: Cambridge University Press, 2006.
- [5] V. Vonásek, J. Faigl, T. Krajník, and L. Přeučil, “RRT-path – a guided rapidly exploring random tree,” in *Robot Motion and Control 2009* (K. R. Kozłowski, ed.), (London), pp. 307–316, Springer London, 2009.
- [6] 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)*, pp. 374–381, Dec. 2019.
- [7] E. Dam, M. Koch, and M. Lillholm, *Quaternions, Interpolation and Animation*. Rapport (Københavns universitet. Datalogisk institut), Datalogisk Institut, Københavns Universitet, 1998.
- [8] S. Fortune, “A sweepline algorithm for voronoi diagrams,” in *Proceedings of the Second Annual Symposium on Computational Geometry, SCG ’86*, (New York, NY, USA), p. 313–322, Association for Computing Machinery, 1986.
- [9] S. K. Ghosh and D. M. Mount, “An output sensitive algorithm for computing visibility graphs,” in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pp. 11–19, 1987.
- [10] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. 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.
- [11] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2, pp. 56–77, Jan. 2014.
- [12] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” 2011.

- [13] R. Sandström, D. Uwacu, J. Denny, and N. M. Amato, “Topology-guided roadmap construction with dynamic region sampling,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6161–6168, 2020.
- [14] V. Vonásek and R. Pěnička, “Path planning of 3D solid objects using approximate solutions,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 593–600, Sept. 2019.
- [15] D. Hsu, H. Cheng, and J.-C. Latombe, “Multi-level free-space dilation for sampling narrow passages in PRM planning,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.
- [16] O. B. Bayazit, D. Xie, and N. M. Amato, “Iterative relaxation of constraints: a framework for improving automated motion planning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3433–3440, 2005.
- [17] E. Pairet, C. Chamzas, Y. R. Petillot, and L. Kavraki, “Path planning for manipulation using experience-driven random trees,” *IEEE Robotics and Automation Letters*, vol. 6, p. 3295–3302, Apr. 2021.
- [18] C. Chamzas, A. Shrivastava, and L. E. Kavraki, “Using local experiences for global motion planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 8606–8612, May 2019.
- [19] O. Sorkine and M. Alexa, “As-Rigid-As-Possible surface modeling,” pp. 109–116, Jan. 2007.
- [20] S. Banerjee and D. Majumdar, “Shape matching in multimodal medical images using point landmarks with hopfield net,” *Neurocomputing*, vol. 30, p. 103–116, Jan. 2000.
- [21] Y. Sahillioglu and Y. Yemez, “Coarse-to-fine isometric shape correspondence by tracking symmetric flips,” *Computer Graphics Forum*, vol. 32, no. 1, pp. 177–189, 2013.
- [22] Y. Sahillioglu, “A genetic isometric shape correspondence algorithm with adaptive sampling,” *ACM Transactions on Graphics*, vol. 37, pp. 1–14, Oct. 2018.
- [23] S. Biasotti, A. Cerri, A. Bronstein, and M. Bronstein, “Recent trends, applications, and perspectives in 3D shape similarity assessment,” *Computer Graphics Forum*, vol. 36, pp. 87–119, Jan. 2016.
- [24] Y. Sahillioglu, “Recent advances in shape correspondence,” *The Visual Computer*, vol. 36, pp. 1705–1721, Aug. 2020.
- [25] Y. Lipman and T. Funkhouser, “Mobius voting for surface correspondence,” *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 28, Aug. 2009.
- [26] T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry, “3D-CODED : 3D correspondences by deep deformation,” in *ECCV*, 2018.
- [27] R. M. Dyke, C. Stride, Y.-K. Lai, P. L. Rosin, M. Aubry, A. Boyarski, A. M. Bronstein, M. M. Bronstein, D. Cremers, M. Fisher, T. Groueix, D. Guo, V. G. Kim, R. Kimmel, Z. Löhner, K. Li, O. Litany, T. Remez, E. Rodolà, B. C. Russell, Y. Sahillioglu, R. Slossberg, G. K. L. Tam, M. Vestner, Z. Wu, and J. Yang, “Shape Correspondence with Isometric and Non-Isometric Deformations,” in *Eurographics Workshop on 3D Object Retrieval* (S. Biasotti, G. Lavoué, and R. Veltkamp, eds.), The Eurographics Association, 2019.

- [28] 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, Dec. 2020.
- [29] F. T. Pokorný, K. Goldberg, and D. Kragic, “Topological trajectory clustering with relative persistent homology,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16–23, 2016.
- [30] K. S. Arun, T. S. Huang, and S. D. Blostein, “Least-squares fitting of two 3-d point sets,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 698–700, Sep. 1987.
- [31] A. Bronstein, M. Bronstein, and R. Kimmel, *Numerical Geometry of Non-Rigid Shapes*. Springer Science+Business Media, LLC, 2009.
- [32] “The Princeton shape benchmark,” in *Proceedings of the Shape Modeling International 2004, SMI '04, (USA)*, p. 167–178, IEEE Computer Society, 2004.
- [33] S. Gottschalk, M. Lin, and D. Manocha, “Obbtree: A hierarchical structure for rapid interference detection,” *Computer Graphics*, vol. 30, Oct. 1997.
- [34] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, p. 509–517, Sept. 1975.
- [35] J. L. Blanco and P. K. Rai, “nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees.” <https://github.com/jlblancoc/nanoflann>, 2014.
- [36] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [37] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, p. 509–517, Sept. 1975.
- [38] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, Dec. 2012. <https://ompl.kavrakilab.org>.
- [39] M. Moll, I. A. Şucan, and L. E. Kavraki, “Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization,” *IEEE Robotics & Automation Magazine*, vol. 22, pp. 96–102, Sept. 2015.
- [40] M. Moll, I. A. Şucan, and L. E. Kavraki, “Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization,” *IEEE Robotics & Automation Magazine*, vol. 22, pp. 96–102, Sept. 2015.
- [41] M. Althoff, M. Koschi, and S. Manzingger, “Commonroad: Composable benchmarks for motion planning on roads,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 719–726, IEEE, 2017.
- [42] A. Gandia, S. Parascho, R. Rust, G. Casas, F. Gramazio, and M. Kohler, “Towards automatic path planning for robotically assembled spatial structures,” in *Robotic fabrication in architecture, art and design*, pp. 59–73, Springer, 2018.

- [43] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995–1001 vol.2, Apr. 2000.
- [44] R. Bohlin and L. Kavraki, “A randomized algorithm for robot path planning based on lazy evaluation,” *Handbook on Randomized Computing*, Jan. 2001.
- [45] R. Bohlin and L. E. Kavraki, “Path planning using lazy PRM,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, pp. 521–528 vol.1, 2000.
- [46] I. Şucan and L. Kavraki, “Kinodynamic motion planning by interior-exterior cell exploration,” pp. 449–464, Jan. 2008.
- [47] D. Hsu, J.-C. Latombe, and R. Motwani, “Path planning in expansive configuration spaces,” *International Journal of Computational Geometry & Applications*, vol. 09, no. 04n05, pp. 495–512, 1999.
- [48] G. Sanchez-Ante and J.-C. Latombe, “A single-query bi-directional probabilistic roadmap planner with lazy collision checking,” pp. 403–417, Jan. 2001.
- [49] B. Gipson, M. Moll, and L. Kavraki, “Resolution independent density estimation for motion planning in high-dimensional spaces,” pp. 2437–2443, May 2013.

I. Personal and study details

Student's name: **Minařík Michal**

Personal ID number: **483465**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Improving Sampling-Based Motion Planning Using Library of Trajectories

Bachelor's thesis title in Czech:

Randomizované plánování pohybu s využitím knihoven trajektorií

Guidelines:

1. Get familiar with sampling-based motion planning [1]; with focus to guided sampling-based planners [2,3]. Study the problem of shape matching of 3D objects [5,6].
2. Implement a sampling-based planner for motion planning of 3D solid objects, e.g. RRT-IR [3].
3. Implement a shape matching method for 3D objects, e.g. [5,6]. Consider shapes from TOSCA, PSB and SHAPENET benchmarks (<http://yulanguo.me/dataset.html>). Create suitable test scenarios for the objects.
4. Create a library of motion trajectories for the objects using e.g. [3].
5. Design a sampling-based planner that utilizes the motion library to speed up the planning task. The trajectories from the library will be selected based on the similarity between the query objects and the objects in the database. The found trajectories will be used to speed up search in the configuration space of the query object. Start with the basic guiding (e.g. [3]) and adapt it for cases where query object has no 100% match in the database.
6. (Optional) Improve the method from 5) by considering deformations between the query object and objects from the database [5]. Alternatively, improve the guided sampling using the concept of multi-level motion planning [6].
7. Experimentally verify performance of all designed methods with suitable methods from OMPL benchmark.

Bibliography / sources:

- [1] LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.
- [2] J. Denny, R. Sandström, A. Bregger, and N. M. Amato. Dynamic region-biased rapidly-exploring random trees. In Twelfth International Workshop on the Algorithmic Foundations of Robotics (WAFR), 2016.
- [3] Vonásek, V., Pěnička, R. & Kozlíková, B. Searching Multiple Approximate Solutions in Configuration Space to Guide Sampling-Based Motion Planning. J Intell Robot Syst 100, 1527–1543 (2020). <https://doi.org/10.1007/s10846-020-01247-4>
- [4] I. A. Sucas, M. Moll and L. E. Kavraki, "The Open Motion Planning Library," in IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72-82, Dec. 2012, doi: 10.1109/MRA.2012.2205651.
- [5] Minh Khoa Nguyen. "Efficient exploration of molecular paths from As-Rigid-As-Possible approaches and motion planning methods". Theses. Université Grenoble Alpes, Mar. 2018.
- [6] Silvia Biasotti et al. "Recent Trends, Applications, and Perspectives in 3D Shape Similarity Assessment", In: Computer Graphics Forum 36 (Jan. 2016), pp. 87–119.
- [7] Orthey, A., Akbar, S., & Toussaint, M. (2020). Multilevel motion planning: A fiber bundle formulation. arXiv preprint arXiv:2007.09435.

Name and workplace of bachelor's thesis supervisor:

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

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

Date of bachelor's thesis assignment: **08.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Vojtěch Vonásek, Ph.D.
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

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

Date of assignment receipt

Student's signature