Rigid and Deformable Motion and Disassembly Planning

with a Focus on the Digital Mockup Process in the Automotive Industry

Dissertation
zur Erlangung des Grades
"Doktor der Naturwissenschaften"

am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität in Mainz,

vorgelegt von
Daniel Schneider, M. Sc.
geboren in Reutlingen

Mainz, den 25. April 2017

Tag der mündlichen Prüfung: 10. Juli 2017

D77

ABSTRACT

In this work, we study algorithms for the virtual disassembly motion planning. In the course of this work, we address three topics from rigid and deformable disassembly motion planning in the automotive industry. The focus is the digital mockup process of a vehicle and we improve existing algorithms but also solve new problems. In the first part of our work, we study rigid body motion planning. In particular, we consider the problem-dependent parameters of state-of-the-art planners. In all planners, these parameters have to be defined *a priori* by the user individually for each dataset. We present an approach that avoids the need for user-adjusted parameters by sampling the parameters randomly during runtime and additionally improves the performance of motion planning. We present a comprehensive experimental analysis of the parameters and the resulting performance. The performance is evaluated in comparison to the well-established open source motion planning library OMPL (Şucan *et al.*, 2012) as well as to the commercial motion planning software Kineo^{TM} Kite Lab (Kineo^{TM}, 2016). In the second part of our work, we study the performance of collision detection in disassembly motion planning for rigid bodies. In this application, collision detection is the major subroutine of the planner and is used to compute the samples for the motion planner. We study in detail the characteristics of the collision detection algorithms in the literature as well as the characteristics of the computed samples. Based on our findings we present a novel algorithm for collision detection in a disassembly motion planner. We compare our approach to well-established collision detection libraries like the Proximity Query Package (Larsen *et al.*, 1999) and the Flexible Collision Library (Pan *et al.*, 2012). In the last part of this work, we address the task that the engineers have to validate the disassembly of a vehicle. We study in detail this task and evaluate the contribution of state-of-the-art algorithms. We present a motion planning problem which cannot be solved by these methods. We call this new motion planning problem, the Invalid Initial State Disassembly Motion Planning Problem. This problem is about computing disassembly paths for deformable objects that are modeled as rigid bodies and are in collision at the beginning of the planning. Besides a study of this problem, we also present an algorithm that can solve the problem.

ZUSAMMENFASSUNG

In dieser Arbeit werden Algorithmen zur virtuellen Montageplanung untersucht. Der Fokus liegt hierbei auf dem Digital Mock-Up Prozess eines Automobilfahrzeugs. Es werden drei Problemstellungen aus dem Bereich der Pfadplanung mit starren und deformierbaren Objekten behandelt. Im ersten Teil der Arbeit werden Algorithmen zur starren Pfadplanung untersucht. Wir befassen uns mit den problemabhängigen Parametern, mit welchen der Pfadplaner realisiert wird. Diese Parameter werden bisher *a priori* durch den Nutzer separat für jeden einzelnen Datensatz festgelegt. Wir werden einen neuen Ansatz vorstellen, der das Festlegen dieser Parameter durch den Nutzer vermeidet und dabei sogar noch die Geschwindigkeit des Planungsprozesses verbessert. Hierzu werden wir eine umfassende experimentelle Analyse der Parameter vorstellen. Die Geschwindigkeit unseres Ansatzes wird mit der Open Source Motion Planning Library OMPL (Şucan *u. a.*, 2012) und der kommerziellen Lösung Kineo^{TM} Kite Lab (Kineo^{TM}, 2016) verglichen und evaluiert. Im zweiten Teil der Arbeit befassen wir uns mit der Laufzeit der Kollisionserkennung eines Pfadplaners und optimieren diese für die Ausbauplanung von starren Objekten. Dazu untersuchen wir die Charakteristik moderner Kollisionsalgorithmen und der für die Planung berechneten Konfigurationen. Auf Basis dieser Untersuchung leiten wir einen Ansatz ab, der die Kollisionserkennung beschleunigt und vergleichen die Geschwindigkeit unseres Ansatzes mit moderner Open Source Software wie dem Proximity Query Package (Larsen *u. a.*, 1999) und der Flexible Collision Library (Pan *u. a.*, 2012). Im letzten Teil dieser Arbeit befassen wir uns mit einer Aufgabe des Ingenieurs innerhalb des DMU Prozesses. Das Thema ist hierbei die Validierung des Montageprozesses eines Fahrzeugs. Wir untersuchen, in wieweit Pfadplaner zur Lösung des Problems verwendet werden und extrahieren eine Fragestellung, welche mit modernen Pfadplanern nicht gelöst werden kann. Wir nennen dieses Problem das Invalid Initial State Disassembly Motion Planning Problem. Bei dieser Fragestellung geht es darum, wie man Ausbaupfade für deformierbare Objekte berechnen kann, welche starr modelliert sind und sich zu Beginn der Planung in Kollision befinden. Wir untersuchen diese Fragestellung und stellen einen Algorithmus vor, welcher in der Lage ist, für dieses Problem Pfade zu berechnen.

DANKSAGUNG

An dieser Stelle möchte ich mich herzlich bei meinem Betreuer/Doktorvater als auch in gleichem Maße bei meiner Betreuerin/Zweitgutachterin bedanken. Sie beide haben mir die Erstellung der Disseration ermöglicht und mit ihren wertvollen Ratschlägen, Ideen und konstruktiver Kritik maßgeblich zum Gelingen dieser Arbeit beigetragen. Dank geht auch an alle Kollegen und Professoren, die mich in Diskussionen mit Ihrem Fachwissen oder in anderer Weise unterstützt haben.
Nicht zuletzt möchte ich mich auch bei meiner Familie und meiner Freundin bedanken. Sie sind mir in der gesamten Zeit zur Seite gestanden, haben Verständnis gezeigt und mich in jeder Situation ermutigt und unterstützt.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

Our work is motivated by the digital mockup (DMU) (Döllner *et al.*, 2000) process in the automotive industry. The DMU process is a tool for all stages of the product lifecycle (Day, 1981) and part of the product lifecycle management (Wei Liu and Brisson, 2009). The DMU process starts in the introduction stage of the product lifecycle,

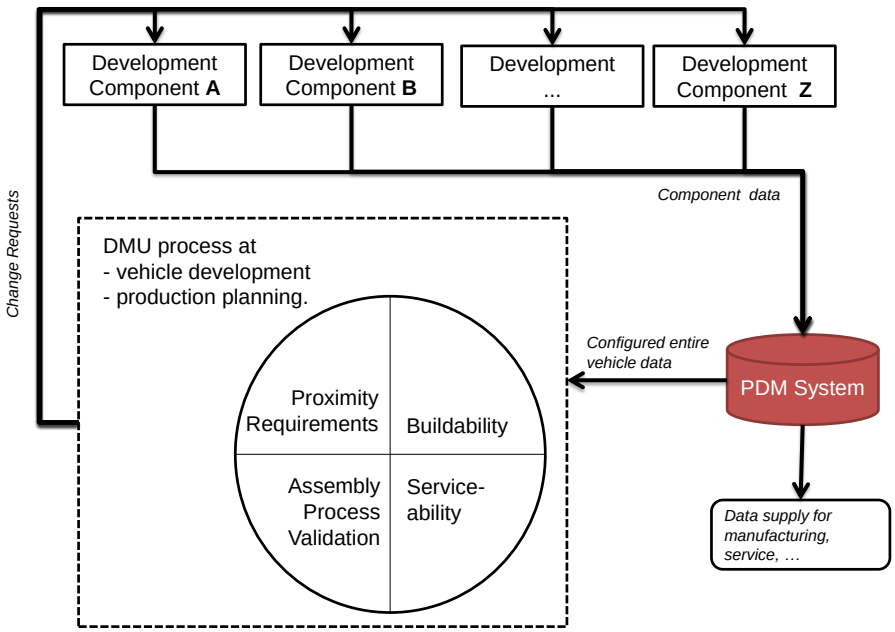


Fig. 1: Context of the DMU process.

where it is used to control the development of the product digitally. In this stage, the goal is that a virtual prototype replaces a manufactured prototype. This carries over to the actual growth and stabilization stage where the methods and models of

the DMU process are used to configure and further improve the product. In the decline stage, the DMU process is used to support a new iteration of the product or the introduction of a entirely new product. An overview of methods and models of the DMU process in the automotive product development was presented by Döllner *et al.* (2000). The DMU process operates on a three-dimensional representation of all the components of the product. This representation is stored in the product data management system (PDM system) (Wei Liu and Brisson, 2009) which takes track of the product data. Besides a three-dimensional representation, the PDM system stores additional meta information like part numbers and suppliers but also valuable information for the release to manufacturing, like the spatial position and orientation of the components and the assembly sequence. In Figure 1, one can see the context of the DMU process. The development teams release part versions of the individual components to the PDM system which stores the progress in a database. The task of the engineers working on the DMU process is to frequently check out the entire vehicle from the PDM system including the components and information mentioned above. They have to ensure the validity of the product and submit change requests to the development teams.

The DMU process includes several tasks. In the proximity control, components of the product are checked for violations of tolerance values, collisions with their environment or spatial conflicts with other components. In the buildability control, the components are checked whether a disassembly path from their installed position exists. In the assembly process validation, the path from the buildability control is validated whether it can be processed for example whether there are tools to realize it. In the serviceability control, the focus is to check whether a component can be serviced and to evaluate the complexity of the service. In this setting, complexity means the spatial free-space of the disassembly path and the tools that can be used for the servicing. In the DMU process, the engineers are supported by a visualization of the scenario, peripherals like a 3D mouse and software tools like a motion planning software. The motion planning software plans a path for a three-dimensional object from an initial configuration that is free of collision to a given goal configuration that is also free of collision. In general, the goal configuration is a configuration that represents a disassembled state, e.g. the separation of the components bounding box and its environment. An example for a disassembly path is given in Figure 2. In yellow we see the near environment of the component, and in green, we see the component itself that has to be disassembled from its installed initial configuration. From left to right we see the path that has to be computed and verified.

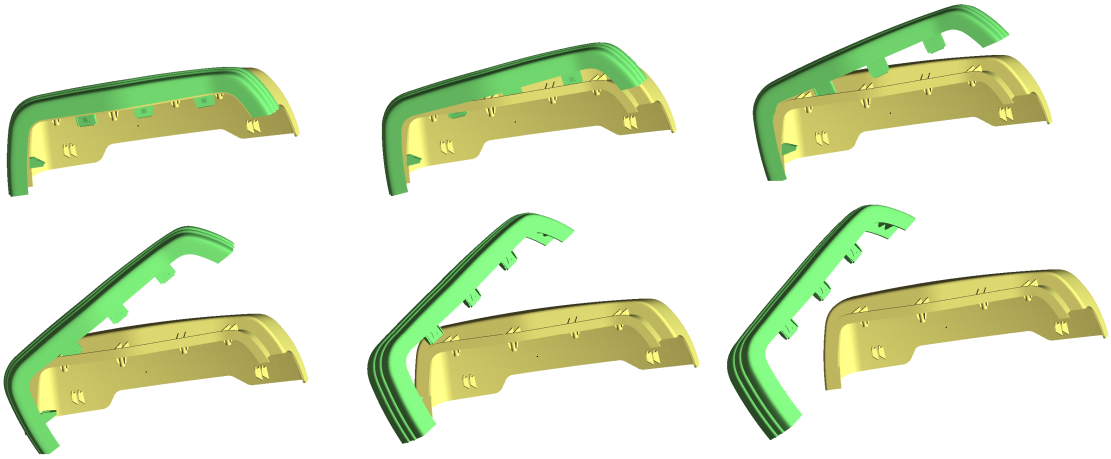


Fig. 2: Example of a disassembly path.

Motion planning has a significant role in the buildability control of the DMU process and is the focus of our work. The methods, algorithms and data structures for motion planning that we present in this work are focused on the usage in the buildability control but can also be used in the other tasks of the DMU process. The application of motion planning algorithms in the DMU process demands several basic conditions and requirements for our work that we will discuss in the following.

1.1 Basic Conditions and Requirements

The DMU process is a part of the early stage of the development of a product. This results in the following conditions and requirements for our presented methods. First, in this stage of the development, one cannot ensure the presence of meta information. In particular meta information like

- **Material of the Data:** For example, part A consists of steel and part B consists of polypropylene.
- **Labeling of the Data:** For example, part A is a screw, part B is a cable and part C is a bolt.
- **Hierarchy of the Data:** For example, part A has two sub-parts A1 and A2.
- **Fastening of the Data:** For example, part A has two sub-parts A1 and A2 and A1 is welded to A2.

The only data one can ensure at this early stage are the CAD models. From this CAD models, one can compute the triangle meshes, given a particular resolution. A fine resolution results in a large triangle set and *vice versa* for a coarse resolution. These meshes are the input to our work as they are the only reliable source in the DMU process. The quality of the computed meshes varies. Examples of quality measures of a mesh are

1. whether all objects have a volume, in particular, thin sheets are represented as a volume,
2. whether the mesh has no large holes,
3. whether the mesh is a watertight/ 2-manifold mesh
4. whether numerical close vertices are merged,
5. whether there are no degenerated triangles and
6. whether state-of-the-art algorithms can compute a numerical stable tetrahedral mesh.

The quality of a mesh is determined by the algorithms that compute the triangular discretization of the CAD model as well as various post-processing algorithms that are applied to the mesh. In the industry, data that is supplied to external software/suppliers often undergoes post-processing algorithms that remove data that is relevant to the construction or that are affected by copyrights. These post-processing algorithms can decrease the quality of the mesh.
In our work, we will focus on algorithms that are not limited to high-quality meshes. Especially we want to avoid the need for an automatic computation of tetrahedral meshes. Our experience showed that the computation of tetrahedral meshes for data in the DMU process is challenging. Even with highly optimized state-of-the-art approaches, it is by no means an automatic procedure. In our work, we designed our algorithms so that the mesh only needs to fulfill the quality measures 1 and 2. This ensures that the engineer can apply our algorithms to a large portion of his datasets without manually analyzing and repairing the input datasets in the DMU process.
A primary requirement for algorithms in the DMU process is a high level of automation. This means that there should be no user interaction needed for the computation as well as no tuning of algorithmic parameters. For the tuning of algorithmic parameters, it is essential to have knowledge about the internal algorithms to determine suitable parameters. Due to the variety of algorithms that are used in the DMU process, the

engineers do not necessarily have a full understanding of the underlying algorithms. Therefore algorithms that are free of any parameter adjustments are advantageous.
At last, another requirement for our methods is the performance. In most cases, the algorithms in the DMU process are executed on the workstations of the engineers rather than on a computer cluster. Moreover, online algorithms and algorithms that can be integrated into the workflow of the engineers are advantageous. Therefore a major topic of our work is the performance of motion planning which we will be highlighted in various aspects.

1.2 Our Contribution

In this section, we show in detail the contribution of our work. First, we want to describe the focus points which are

1. low requirements for the input data supplied in the DMU process,
2. ease of use of the presented approaches, like no parameter adjustments,
3. the performance of online interaction with the user as well as the performance in offline computations,
4. the stability of the algorithms,
5. a simple implementation of the presented methods,
6. a specialization and improvement of existing methods to the DMU process, and
7. the presentation of algorithms for the solution to a new problem.

In the last section, we described the basic conditions for the DMU process. The first three focus points of our work are derived directly from these conditions of the DMU process. Besides these three focus points, we also focus our approaches on stability as it ensures a continuous workflow for the engineer. If algorithms are unstable in the DMU process, it results in manual work. For example, if a motion planning algorithm fails, the engineer has to manually define the disassembly path by sequentially defined translations and rotations with the aid of a visualization of the scenario and a 3D mouse. Another focus of our work is on simple implementation which can easily be added to the existing software. It is motivated by the fact that we want our approaches to be easily reproducible. Next, we want to focus on the specialization and

improvement of existing methods. In the DMU process, many problems are solved by state-of-the-art algorithms which are not necessarily tuned for the application in the DMU process. We will focus on investigating these state-of-the-art approaches and present new approaches that specialize and improve them to the DMU process. At last, we will also study new problems. We present our methodology and case studies. Based on these results we describe solutions to the problems and evaluate them on several benchmark datasets.
In our work, we present three results. They are called

- the Completely Randomized Rapidly Exploring Random Tree Connect (short: CR-RRTC) (Schneider *et al.*, 2015b),
- the Fast Alternating Random Ray Approach (short: FARR) (Schneider *et al.*, 2017) and
- the Invalid Initial State Disassembly Motion Planning algorithm (short: InIState Motion Planner) (Schneider *et al.*, 2015a).

Each of these approaches has different focus points which we will address in the following together with their contributions. We will discuss the focus points of all our approaches with a spiderweb diagram. The corners represent our focus points. Each of our algorithms is visualized by a path through the spiderweb. The path for each approach is defined by the evaluation. We evaluate each approach on a scale from No to Low to Medium to High focus.

Our first approach is called the Completely Randomized Rapidly Exploring Random Tree Connect. This approach presents a solution to three-dimensional rigid body motion planning. Nowadays sampling-based motion planners use the power of randomization to compute multi-dimensional motions at high performance (LaValle, 2011). This performance is based on problem-dependent parameters like the weighting of translation versus rotation and the planning range of the algorithm. Former work uses constant user-adjusted values for these parameters which are defined *a priori*. The contribution of this work is to present a new approach which extends the power of randomization by varying the parameters randomly during runtime. This approach is the first work that introduces a random, and especially variable, selection of the motion planning parameters during the planning time. So with our approach we improve an existing method on a well-known problem. Selecting the motion planning parameters randomly avoids a pre-processing step to adjust them and avoids the need

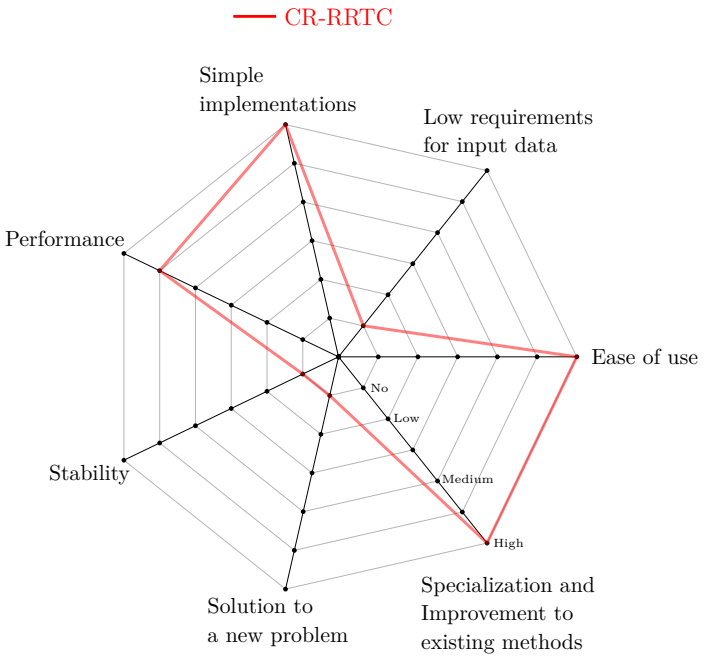


Fig. 3: Focus of the CR-RRTC approach.

for any algorithmic knowledge about the algorithms for the engineer in the DMU process. With this approach, we focus on the ease of use of three-dimensional rigid body motion planning. The requirements on input data for our approach are the same as for a general motion planning algorithm and not focused. In general, the method for adjusting the parameter is straightforward to understand and implement and can be applied to the many existing motion planning algorithms. Moreover, our approach improves the performance in comparison to existing methods in the majority of the benchmarks. To compare our approach, we present a comprehensive experimental analysis and study of the parameters and the resulting performance. The algorithms and data structures were implemented in the library (RASAND, 2010-2017), but we also compare the results of our work with the open source motion planning library OMPL (Şucan *et al.*, 2012) as well as the commercial motion planning software Kineo^{TM} Kite Lab (Kineo^{TM}, 2016). Stability was not focused in this approach as rigid body motion planning algorithms are stable.

Our second approach is called the Fast Alternating Random Ray Approach. The topic is the improvement of performance for collision detection in motion planning algorithms. So the primary focus is again on improving existing methods. In sampling-based three-dimensional rigid body motion planning, one of the major subroutines is

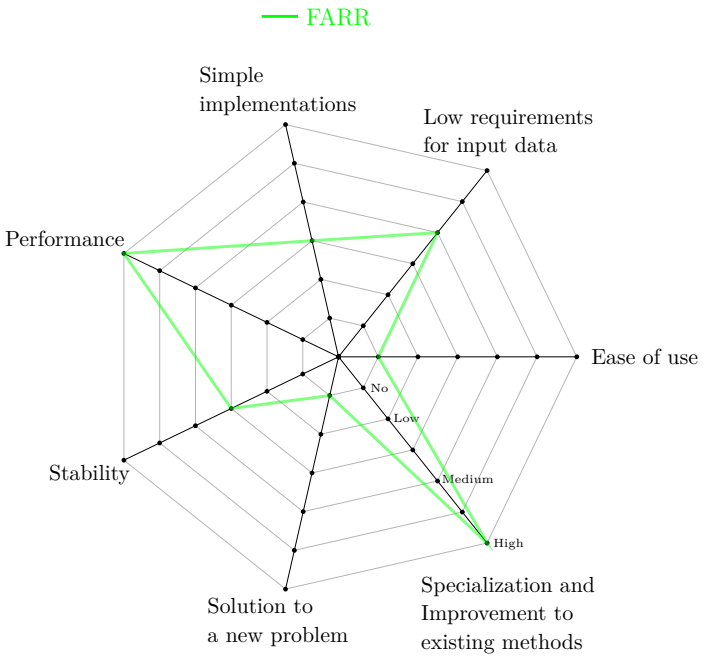


Fig. 4: Focus of the FARR approach.

collision detection. In particular for motion planning problems with narrow passages like in disassembly motion planning, many samples have to be checked by a collision detection algorithm. In this application, the runtime of the motion planning algorithm is even dominated by collision detection. We study in detail the characteristic of samples in motion planning scenarios and show that the samples in this application have a very specific characteristic. The characteristic is that many of the samples are in collision and have small penetration volumes. Collision detection for samples with this characteristic has never been addressed in the literature. We introduce a data structure as well as an algorithm that makes use of this characteristic by combining well-know data structures like a distance field and an octree with the swap algorithm (Llanas *et al.*, 2000). The collision detection algorithm we present is used by a motion planning algorithm and not the by the user. Therefore ease of use is not in focus. State-of-the-art approaches to collision detection have little requirements to the mesh. Some of them, like the bounding volume hierarchies, can even handle triangle soups. To stay comparable to these approaches, we focus on presenting an approach that has little requirements on the input data. As our approach is based on these well-known data structures, we can ensure a simple implementation of the underlying data structures. Nevertheless, the collision detection algorithm itself is quite challenging to implement. The primary focus of our FARR approach is on performance. For

three-dimensional rigid body motion planning with narrow passages, we compare our approach to well-established collision detection algorithms. The commonly available collision detection libraries that we use as comparison are the Proximity Query Package (PQP) (Larsen *et al.*, 1999) and the Flexible Collision Library (FCL) (Pan *et al.*, 2012). Besides the collision detection itself, we will also benchmark the application to motion planning. In general motion planning algorithms can be categorized into two subcategories. The tree-based (Lavalle, 1998) and roadmap-based planners (Kavraki *et al.*, 1996). We will show the impact of our approach on both types of motion planning algorithms. At last, we also focused on stability. We ensured that the results of our approach are the same as in the above-mentioned state-of-the-art approaches.

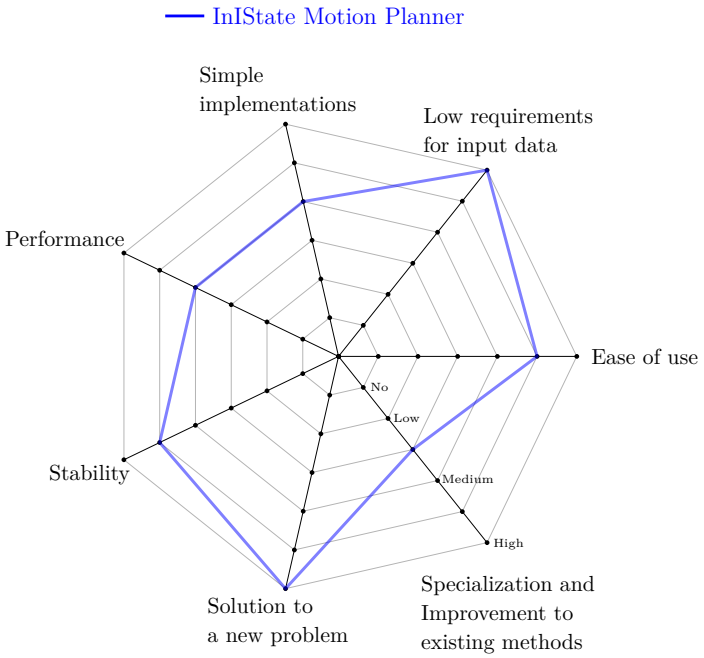


Fig. 5: Focus of the InIState Motion Planner.

The last result that we present in this work is called the Invalid Initial State Disassembly Motion Planning algorithm. In this approach, we will deal with a new motion planning problem from the DMU process. We will study, define and solve this new problem. In the DMU process, the contribution of the motion planning algorithm for the computation of a disassembly path is quite limited. The main work is done manually by the engineer rather than by an automatic computation of the motion planning software. The main reason for this is the following:
In order to compute a path with a motion planning algorithm the initial configuration

has to be free of collision. But in practice, almost every component is in collision at its initial state. The reasons for these collisions are analyzed and categorized in detail in this work. In order to start a state-of-the-art motion planning algorithm, the engineer first has to manually place the object into a state that is free of collision. In most cases, this means that the complex part of the disassembly path, where the maneuverability of the component is quite restricted, must be defined manually. We call this problem of finding a path from the initial invalid position to a position that is free of collision, the Invalid Initial State Disassembly Motion Planning Problem. We will also derive a more formal definition of the InIState problem later in this work.
In the literature, there exists no algorithm that can calculate a reasonable disassembly path for an invalid initial state. We will present a novel algorithm which overcomes this limitation by computing information about the flexible parts of the dynamic object and incorporating this information into the disassembly planning. Therefore the primary focus of this result is the solving of a new problem. But we also specialized our approach for the DMU process. Deformation models and also their integration to motion planning algorithms is challenging. In our approach, we evaluate an approach for the deformation model, which is originated in the field of computer graphics and animation, for its application in the DMU process. This deformation model has the advantages that the requirements to the mesh are little, the deformation model is very simple to implement, stable and computes deformations in real time. This all matches our focuses for the DMU process. But we also focus on the ease of use for the user. Therefore, we present a method that automatically determines the crucial parameter in the resulting new motion planning algorithm. As the input to our algorithm is only the triangle meshes, we have to put a high focus on low requirements for our approach. We put a focus on a simple implementation e.g. by using deformation model that is easy to implement. For analyzing the performance and stability of our approach, we evaluated the approach on a benchmark dataset that is modeled after real world motion planning scenarios from the DMU process and shows that our new approach can solve the InIState problem.

1.3 Structure

Our work has six chapters. After this introduction, we will summarize the background that is needed for our work in the second chapter. This chapter contains the background about rigid and deformable motion planning algorithms as well as the basic

data structures that are needed for our three approaches. In the three following chapters we deal with our three approaches. In each chapter, we describe our methodology and in particular the studies that we initiated. Moreover, we describe the approaches and the experiments. In the experiments, we present the benchmarks that we use for evaluation as well as the results for these benchmark sets. At the end of our work, we draw a conclusion and give insights about future work in the field of motion planning in the DMU process.

2

BACKGROUND

In this chapter, we will focus on the background of our work. The topic is rigid and deformable motion planning. We deal with motion planning for three-dimensional bodies, a problem with at least six degrees of freedom. For such high degrees of freedom sampling-based motion planners are needed to solve the problems efficiently. We will introduce two well-known sampling-based motion planning algorithms. A primary ingredient of these motion planning algorithms is spatial data structures for tasks like collision detection or near neighbor search. The basis of these data structures is also covered in this chapter. At last, we introduce deformable motion planning and how algorithms solve deformable motion planning problems in general which include the background of various deformation models.

2.1 Motion Planning

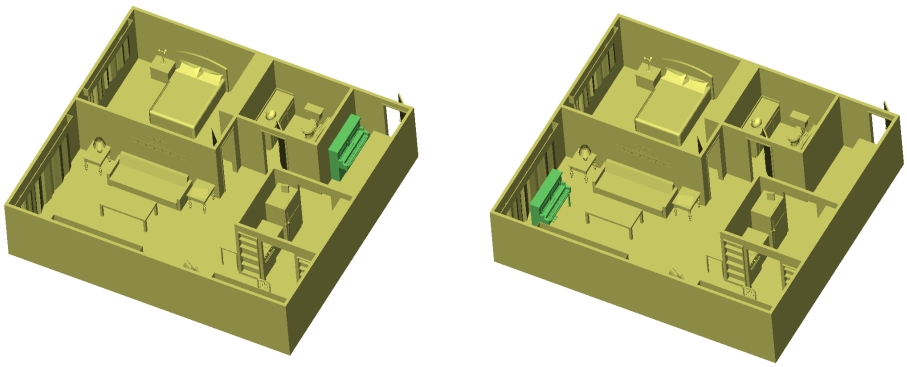


Fig. 6: An example of a single-query motion planning problem. Move the piano from the entrance to the living room (Piano Movers Problem).

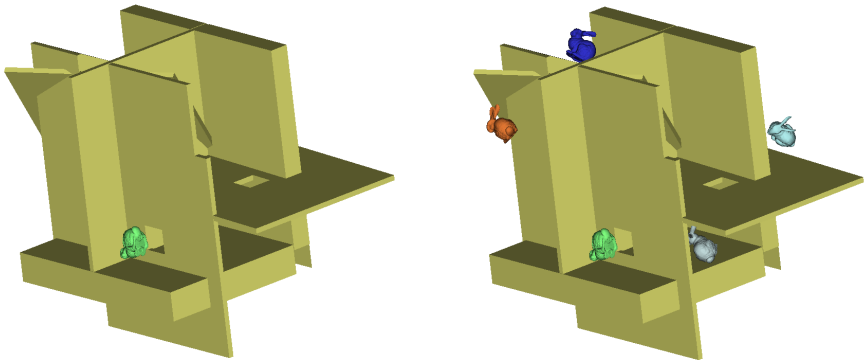


Fig. 7: An example of a multi-query motion planning problem. Move the bunny to different locations in the mace.

The motion planning problems studied in our work are defined by a scene consisting of a set of static objects S_i that we can join to one object S and a dynamic object D. Each object is represented by a set of primitives. In this work, we consider triangles as primitives. The translations and orientations of D are mapped uniquely to a configuration c. Applying a configuration c to D is denoted by $D(c)$. The space of all possible configurations is called the configuration space C. For C we define the validity function $v : C \to \{0, 1\}$. A well-known example for v is:

$$v(c) = \begin{cases} 1 & \text{if } D(c) \cap S = \emptyset \\ 0 & otherwise \end{cases} \tag{1}$$

The configuration space C is the union of the set of all valid configurations, called $C_{free} = \{c \in C \mid v(c) = 1\}$, and the set of all invalid configurations, called $C_{obstacle} = \{c \in C \mid v(c) = 0\}$. For C we define a metric with the distance function $d : C \times C \to \mathbb{R}$ that is used to define nearest neighbors. For a set of n configurations $C_{sampled} = \{c_0 \dots c_{n-1}\}$ and a configuration $c \in C$ the nearest neighbor is defined as $NN(c, C_{sampled}) := \min_{0 \le i \le n-1} d(c_i, c)$. The motion planning problem is defined as follows:

Definition 1 (*Motion Planning Problem (Single-Query)*)

Given $c_{init} \in C_{free}$, the valid initial configuration of D, and $c_{goal} \in C_{free}$, the valid goal configuration of D. The single-query motion planning problem is to find a continuous path $\sigma : [0, 1] \to C_{free}$ with $\sigma(0) = c_{init}$ and $\sigma(1) = c_{goal}$, if one exists.

This definition is the single-query version of the motion planning problem. An example for a single-query motion planning is shown in Figure 6. There, we see the problem of moving a piano the entrance of an apartment to the living room. We also define the multi-query motion planning problem as follows.

Definition 2 (*Motion Planning Problem (Multi-Query)*)

Given n tuples $(c^i_{init}, c^i_{goal}), i \in [0, n-1]$ with $c^i_{init} \in C_{free}$, the valid initial configurations of D, and $c^i_{goal} \in C_{free}$, the valid goal configuration of D. The multi-query motion planning problem is to find n continuous paths $\{\sigma_i : [0,1] \to C_{free}\}$ with $\sigma_i(0) = c^i_{init}$ and $\sigma_i(1) = c^i_{goal}$, if they exist.

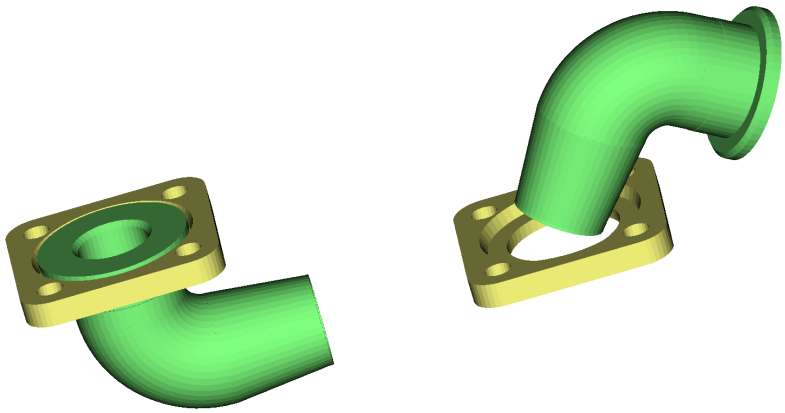


Fig. 8: An example of a disassembly motion planning problem. Separate the tube/flange from its environment (Flange Problem).

An example for a multi-query problem is shown in Figure 7. There, we see the problem of moving a bunny to multiple positions in a maze. In our work, we consider three-dimensional rigid bodies. Therefore C is defined as

$$SE(3) = \left\{ A \mid A = \begin{pmatrix} R & t \\ 0_{1\times 3} & 1 \end{pmatrix} \in \mathbb{R}^{4\times 4}, R \in \mathbb{R}^{3\times 3}, t \in \mathbb{R}^3, \right.$$
$$\left. R^T R = R R^T = I, det(R) = 1 \right\},$$

the special Euclidean group $SE(3)$ that represents three-dimensional rigid body motions (Kumar, 2007). Each configuration $c_i \in C$ defines a translation t_i and a rotation R_i of the dynamic object D. Note that the rotation could also be defined by a quaternion $q \in \mathbb{H}$. Note that in some algorithms the translation t is restricted to a three-dimensional interval B that is defined by $t_{min}, t_{max} \in \mathbb{R}^3$. This done to limit

the configuration space and speed up the computation. In our work, we also consider disassembly planning, a subproblem of the single-query motion planning problem. In the disassembly planning problem, the validity function v is defined as in Equation 1. Moreover, the goal configuration c_{goal} is defined as a set of configurations C_{goal}. Two common definitions of C_{goal} are

- all configurations that separate the bounding boxes of D and S or
- all configurations that realize a minimal distance of D and S larger than a given threshold.

The disassembly motion planning problem is defined as:

Definition 3 (*Disassembly Motion Planning Problem*)

Given $c_{init} \in C_{free} \subset SE(3)$, the valid initial configuration of D, and $C_{goal} \subset C_{free} \subset SE(3)$, the valid set of goal configurations of D that represents the disassembled states. The disassembly motion planning problem is to find a continuous path $\sigma : [0,1] \rightarrow C_{free}$ with $\sigma(0) = c_{init}$ and $\sigma(1) \in C_{goal}$, if one exists.

2.2 Sampling-based Motion Planning

To solve motion planning problems, especially for high dimensions of freedom, one can apply sampling-based motion planners. Those algorithms use samples of C to build up a data structure that characterizes the free space C_{free} of the problem. Due to this discretization, the solution is a finite sequence of configurations. In recent years sampling-based motion planners became popular, especially for disassembly planning. There are tree-based approaches like the rapidly-exploring random tree (RRT) (Lavalle, 1998) for single-queries and roadmap-based methods like the probabilistic roadmap methods (PRM) (Kavraki *et al.*, 1996) for multiple queries. For a deep insight into these algorithms, we refer to the work above and the comprehensive work from LaValle (2011) and LaValle (2006). In the recent years, researchers were focused on solving motion planning problems with narrow passages (Hsu *et al.*, 2003) (Zhang and Manocha, 2008) (Hsu *et al.*, 1998) (Sam Rodriguez and Amato, 2006). A narrow passage is a part of the solution path which is highly restricted in all degrees of freedom. Especially disassembly motion planning problems are often characterized by narrow passages near

the initial configuration. In the following sections, we will briefly sum up the main approaches in sampling-based motion planning that are essential for our work.

2.2.1 Rapidly-Exploring Random Trees

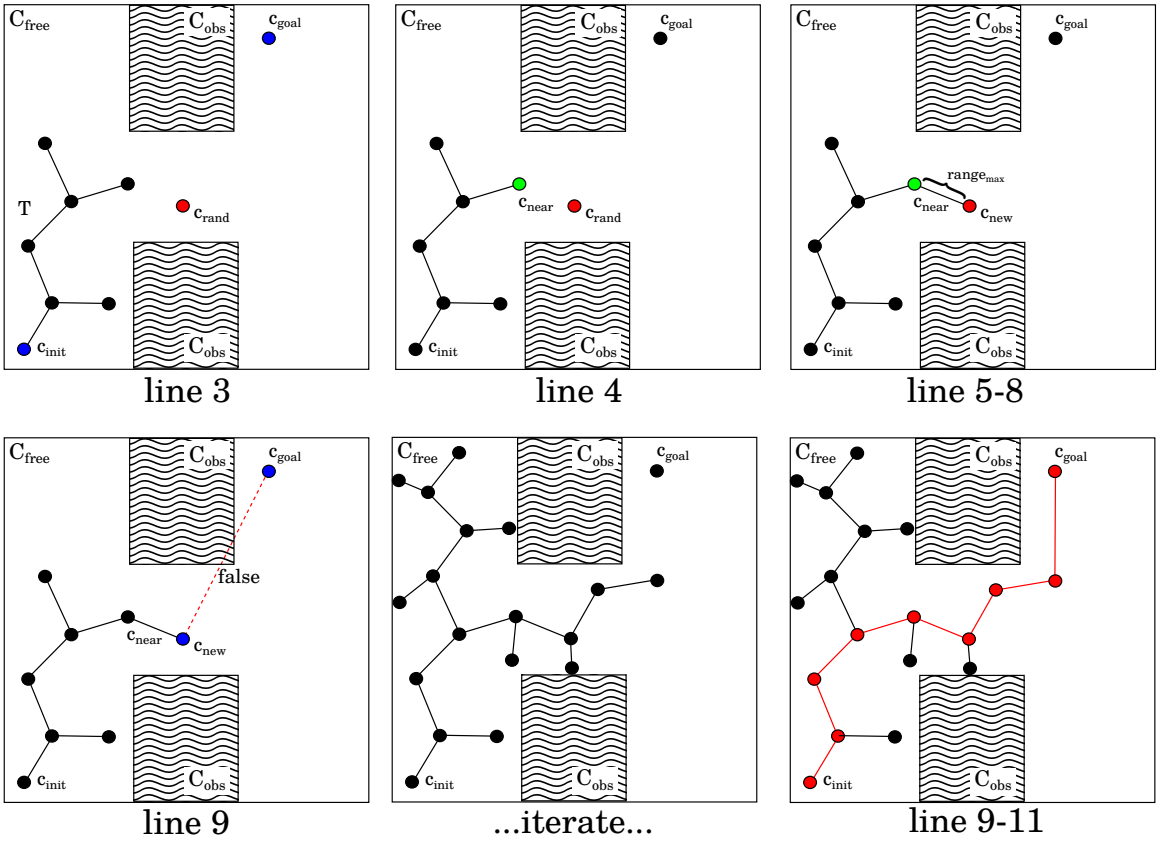


Fig. 9: An example of the Rapidly-Exploring Random Tree.

The Rapidly-Exploring Random Tree (RRT) (Lavalle, 1998) is a very common algorithm for single-query motion planning problems. The main idea behind an RRT is to sample random configurations c_{rand} to build up a tree T and find a path in T. The RRT algorithm is shown in Figure 9 and Algorithm 1. The tree T is rooted in the initial state c_{init} and consists of configurations and edges. When a random configuration c_{rand} is computed, the algorithm determines the nearest neighbor $c_{near} \in T$. Afterwards, the $getValidEdge$ function tries to draw a valid edge from c_{near} to c_{rand} with at most $range_{max}$ length. The edge is sampled with a given resolution. The validity of each of this sampled configurations on the edge is ensured by a collision detection algorithm. The valid part of the edge from c_{near} to c_{new} is inserted into the tree T. In

Algorithm 1: RRT (c_{init}, c_{goal}, $range_{max}$, $time_{max}$)

```
T.init(c_init)                                                          1
while (! TimeElapsed(T_max)) do                                         2
    c_rand ← RandomState()                                              3
    c_near ← NearestN(c_rand, T)                                        4
    c_new ← getValidEdge(c_near, c_rand, range_max)                     5
    if (! Trapped) then                                                 6
        T.addVertex(c_new)                                              7
        T.addEdge(c_near, c_new)                                        8
        if goalReached(c_new,c_goal) then                               9
            return Path(T)                                              10
return ApproxPath(T)                                                    11
```

Algorithm 2: RRTC ($c_{init}, c_{goal}, range_{max}$)

```
T_1.init(c_init), T_2.init(c_goal)                                      1
while (! TimeElapsed(T_max)) do                                         2
    T_current, T_other ← Swap(T_1, T_2)                                 3
    c_rand ← RandomState()                                              4
    c_near ← NearestN(c_rand, T_current)                                5
    c_new ← getExtend(c_near, range_max)                                6
    if (! Trapped) then                                                 7
        T_current.addVertex(c_new)                                      8
        T_current.addEdge(c_near, c_new)                                9
        c_near' ← NearestN(c_new, T_other)                              10
        c_new' ← getExtend(c_near', range_max)                          11
        if (! Trapped) then                                             12
            T_other.addVertex(c_new')                                   13
            T_other.addEdge(c_near', c_new')                            14
            if (Reached) then                                           15
                return Path(T_1, T_2)                                   16
return ApproxPath(T_1, T_2)                                             17
```

case the entire edge is not valid the algorithm is trapped and a new configuration is sampled. As the configurations c_{new} are selected based on nearest neighbors, the RRT has a space-exploring behavior. The exploration takes place until a given goal state is reached and a path is found or the maximum time is elapsed. In the second case, only a portion of the path or an approximated path can be returned. The RRT algorithm is not biased towards the goal, it only evaluates if the goal is reached. Therefore the RRT algorithm is well suited for disassembly motion planning problems because they have a large set of goal configurations. But if only a single configuration c_{goal} is given, algorithms without a goal bias have a low performance. Therefore, Kuffner and LaValle (2000) presented the Rapidly-Exploring Random Tree-Connect (RRTC) which is very similar to the RRT. It builds up two trees with one located at c_{start} and one located at c_{goal}. The second tree in c_{goal} realizes the goal bias. Moreover, the RRTC proved to perform well in many challenging benchmarks (Kuffner and LaValle, 2000). The RRTC algorithm is given in Algorithm 2. The RRTC samples a configuration c_{rand} uniformly in C in each iteration and selects one of the two trees to expand. A possible selection criterion is the sum of edge lengths in each tree. The sample point c_{rand} is then tried to be connected to its nearest neighbor c_{near} in the selected tree. After extending c_{near}, the algorithm tries to connect the trees by using the last configuration of the edge as new sample point c_{new} for the other tree. This process is repeated until both trees are connected, hence a valid path is found.

2.2.2 Probabilistic Roadmap Methods

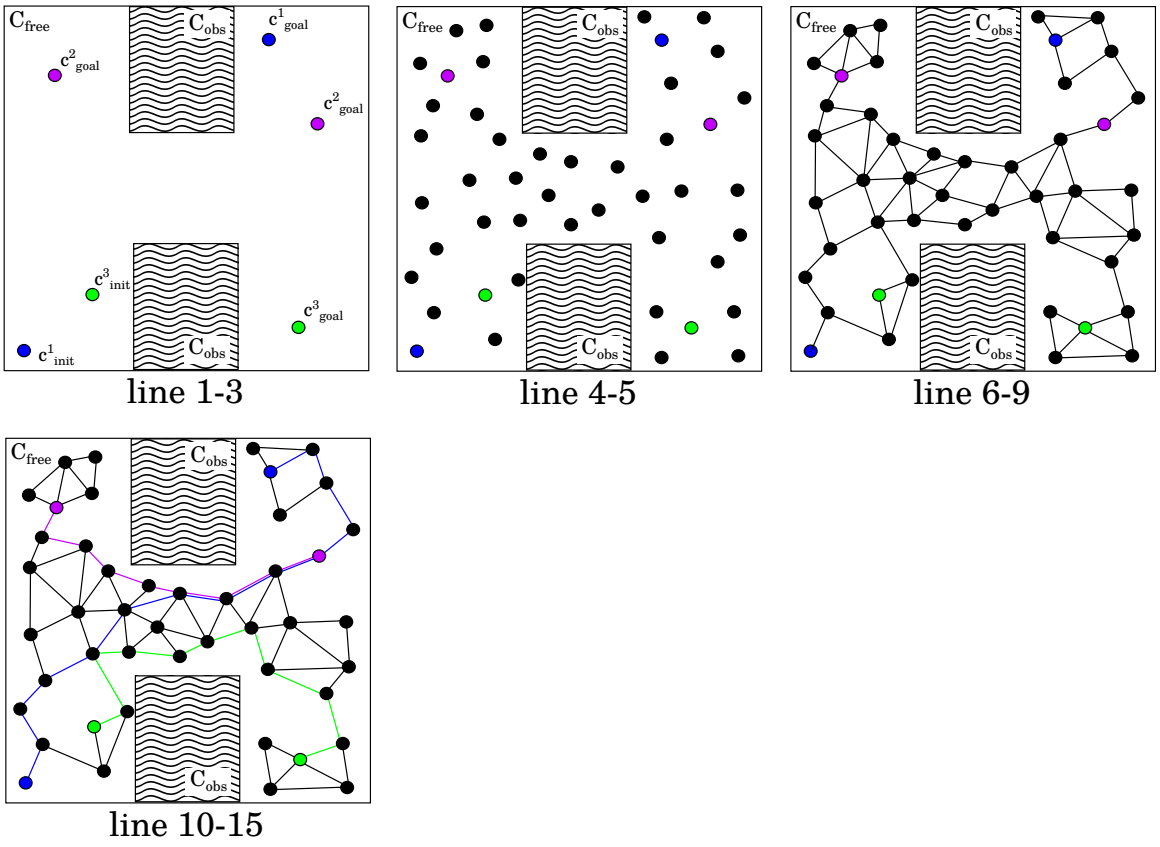


Fig. 10: An example of the Probabilistic Roadmap Method.

Algorithms based on the RRT approach are not suitable for multi-query motion planning as the data structure depends on the initial and goal configuration. For multi-query motion planning Kavraki *et al.* (1996) presented the Probabilistic Roadmap Methods (PRM). The parallelizable version of the PRM algorithm is given in Algorithm 3 and Figure 10. The algorithm builds up an undirected graph of vertices V and edges E in order to compute paths σ_i for each tuple (c^i_{init}, c^i_{goal}). The inputs to the algorithm are the initial and goal configurations, the number of samples n and the radius r. After initializing E and V in line 1-3, the algorithm computes n sample vertices from C_{free} in line 4-5. In line 6-10 the algorithm computes the edges E for all the vertices $v \in V$. Therefore, the algorithm computes for each vertex v the set of vertices U with a distance smaller than r. For each vertex $u \in U$ the algorithm tries to draw a valid edge from u to v. At last the algorithm computes for each tuple (c^i_{init}, c^i_{goal}) the shortest path σ_i in the graph if one exists. This is the result of the algorithm.

Algorithm 3: PRM $(\{(c^i_{init}, c^i_{goal})\}, r, n)$

```
E ← ∅, V ← ∅                                                          1
foreach ((c^i_init, c^i_goal) ∈ {(c^i_init, c^i_goal)}) do            2
    V ← V ∪ c^i_init ∪ c^i_goal                                       3
for j ← 0 to n do                                                     4
    V ← V ∪ CFreeSample()                                             5
foreach v ∈ V do                                                      6
    U ← Neighbors(v, V, r)                                            7
    foreach u ∈ U do                                                  8
        if (edgeIsValid(u, v)) then                                   9
            E ← E ∪ (u, v)                                            10
foreach ((c^i_init, c^i_goal) ∈ {(c^i_init, c^i_goal)}) do            11
    if connected(c^i_init, c^i_goal, V, E) then                       12
        σ_i = shortestPath(c^i_init, c^i_goal, V, E)                  13
    else                                                              14
        σ_i ← ∅                                                       15
return {σ_i}                                                          16
```

2.2.3 Expansive Space Tree

Algorithm 4: EST $(c_{init}, c_{goal}, r_1, r_2)$

```
T.init(c_init)                                                    1
while (! TimeElapsed(T_max)) do                                   2
    c_way ← chooseWaypoint(T)                                     3
    c_new ← expandWaypoint(c_way, r_1)                            4
    if (! Trapped) then                                           5
        T.addVertex(c_new)                                        6
        T.addEdge(c_way, c_new)                                   7
        T.updateWeights(r_2)                                      8
        if goalReached(c_new,c_goal) then                         9
            return Path(T)                                        10
return ApproxPath(T)                                              11
```

At last, we present the Expansive Space Tree approach (EST) (Hsu *et al.*, 1997). The work of Erbes (2013) showed that an extended version of this algorithm performs well for disassembly planning with narrow passages. As we use the EST approach later in this work, we explain it in the following. The algorithm is very similar to the RRT presented in Section 2.2.1. A pseudo code of the EST is shown in Algorithm 4 and we will explain the algorithm using Figure 11. In the first picture of Figure 11 we see an intermediate state of the algorithm. The algorithm builds up a tree and for each node of the tree a *weight* is assigned. These weighted tree nodes are called waypoints. An example for the *weight* is the inverse of the number of neighbors in a radius r_2:

$$weight = \frac{1}{\#neighbors}.$$

In the second image and in line 3 of Algorithm 4 the waypoint c_{way} with the smallest weight is computed. In the following, this waypoint is expanded by sampling a configuration c_{new} in a radius r_1 around c_{way}. In the fourth image and in line 6-8 of Algorithm 4, the configuration c_{new} is added to the tree T, if possible, and the weights are updated. In the following, the algorithm is iterated until the algorithm can connect the data structure to c_{goal} similar to the RRT approach.

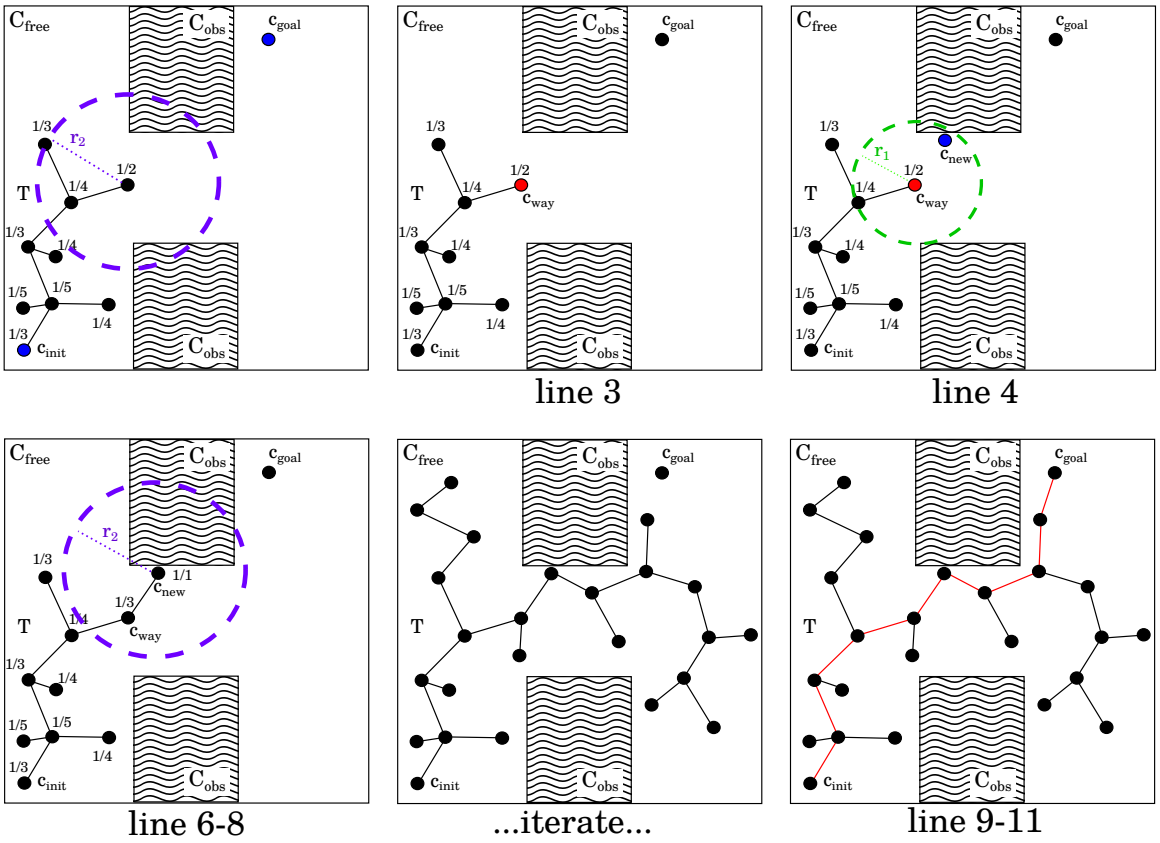


Fig. 11: An example of an Expansive Space Tree approach.

2.2.4 Motion Planning Parameters

The performance of motion planners is very input sensitive. It depends on the dataset, on the selected planner as well as on several parameters. The RRTC is a motion planner which needs less parameter tuning than other planners (see Lindemann and LaValle (2005, page 4,7)). In the following, we will discuss the two main parameters. First, the metric used in the algorithm is important for the performance of the motion planner. A metric for motion planning was proposed by LaValle (2006) and Yershova and LaValle (2007):

$$d_c(c_1, c_2, u_1, u_2) = u_1 \cdot d_t(t_1, t_2) + u_2 \cdot d_q(q_1, q_2) \tag{1}$$

with the Euclidean distance $d_t(t_1, t_2) = ||t_1 - t_2||_2$ and the distance $d_q(q_1, q_2) = min\{acos(q_1 \cdot q_2), acos(q_1 \cdot (-q_2))\}$. LaValle and Yershova describe the importance of a proper scaling of the distances d_t and d_q with some non-zero real constants u_1 and

u_2. These parameters control the importance of the translation versus the importance of the rotation of D. Note that (1) can also be expressed by only one parameter u if the equation is divided by u_2. Amato *et al.* (1998) studied the performance for several metric spaces on the PRM. They proposed a similar metric and investigated its parameters. Amato *et al.* showed that the importance of translation increases in highly restricted C-space regions. Vahrenkamp *et al.* (2011) investigated the topic of parameter-free motion planning algorithms, focusing on the resolution parameter to ensure exact motion planning. In this manner, exact motion planning means that the edges are verified exactly and not by a sampling resolution. They proposed a unification of the C-space for robotic applications. A similar unification we will use later in our work.
The second parameter is the parameter for the maximal length of an edge $range_{max}$ in a tree-based planner. This parameter was rarely studied in the literature. Often greedy approaches for $range_{max}$ were proposed, e. g. by Sam Rodriguez and Amato (2006). Greedy approaches try to extend the configuration c_{near} until the sampled configuration c_{rand} or $C_{obstacle}$ is reached. They perform well in many benchmarks, especially in the famous Alpha Puzzle benchmarks. Nevertheless, there are examples where a greedy approach decreases performance dramatically.
To deal with the parameters, previous work proposed methods to vary the planner during runtime. For example, Morales *et al.* (2004) presented a Probabilistic Roadmap Method (PRM) that varies the planner depending on a partition of C. For each part of the partition, the planner is selected based on features like the ratio of free nodes. The drawback of this work is the need of user interaction and parameter tuning. Tapia *et al.* (2009) extended this work to overcome this disadvantage.Later in our work, we will also present an approach that deals with the two parameters of the RRT efficiently.

2.2.5 Sampling for Motion Planners

Every sampling-based motion planner needs to compute samples of C_{free} in a predefined sampling region. An example of a sampling region is shown on the left of Figure 12. There are many sampling strategies besides a uniform sampling. Especially for problems with narrow passages the Gaussian sampling and the bridge test are often used. Examples of the sampling strategies mentioned above are shown in Figure 12 and Figure 13. In the Gaussian sampling (Boor *et al.*, 1999) the border of C_{obs} is sampled with a higher probability. Therefore a uniform sample c_{rand} is com-

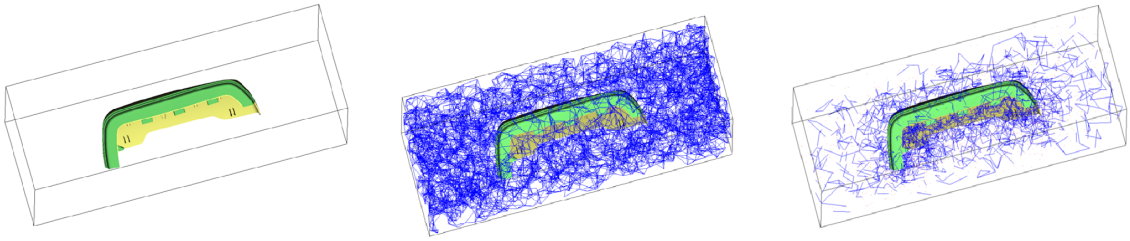


Fig. 12: Examples for the sampling region (left), uniform sampling (middle) and bridge test sampling (right) visualized in the workspace.

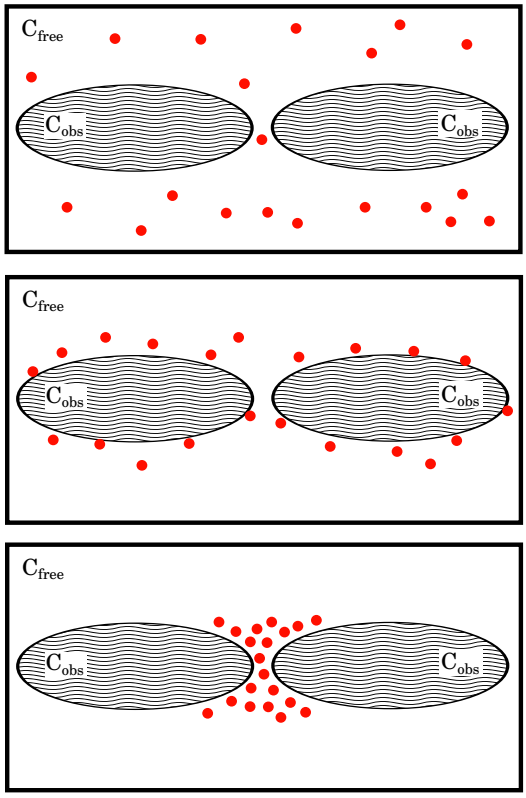


Fig. 13: Examples for uniform sampling (top), Gaussian sampling (middle) and bridge test sampling (bottom) in the configuration space.

puted. If c_{rand} is valid, c_{rand} is accepted as Gaussian sample if a second configuration c_{normal} is invalid. The second configuration c_{normal} is sampled from a normal distribution $N(c_{rand}, \sigma^2)$ with mean c_{rand} and variance σ^2. The variance σ^2 is a parameter of the sampling. If c_{rand} is invalid, its is accepted as a Gaussian sample if configuration c_{normal} is invalid. The bridge test (Hsu *et al.*, 2003) is very similar. In the bridge test, a bridge is defined by two invalid configurations c_{rand}, c_{normal} and the valid midpoint configuration c_{mid}. Like in the Gaussian sampling, c_{rand} is uniformly sampled and c_{normal} is drawn from a normal distribution. But in contrast, both samples need to

be invalid and c_{mid} must be valid. Only if this holds true, c_{mid} is accepted as a bridge test sample and added to the data structure. Otherwise, the sampling restarts right from the beginning. In contrast to a uniform sampling, the Gaussian and bridge test sampling strategies share the same characteristic. They use the structure of C_{obs}. In practice, a mixture of all three sampling strategies is used.

2.3 Spatial Data Structures

Every motion planner has a validity function $v : C \rightarrow \{0, 1\}$ that defines, whether a configuration is valid or not. In the general motion planning problem validity is checked by a collision test (see Section 2.1 Equation1). A common collision detection algorithm is the Voxmap PointShell^{TM} algorithm by McNeely *et al.* (2005). The main

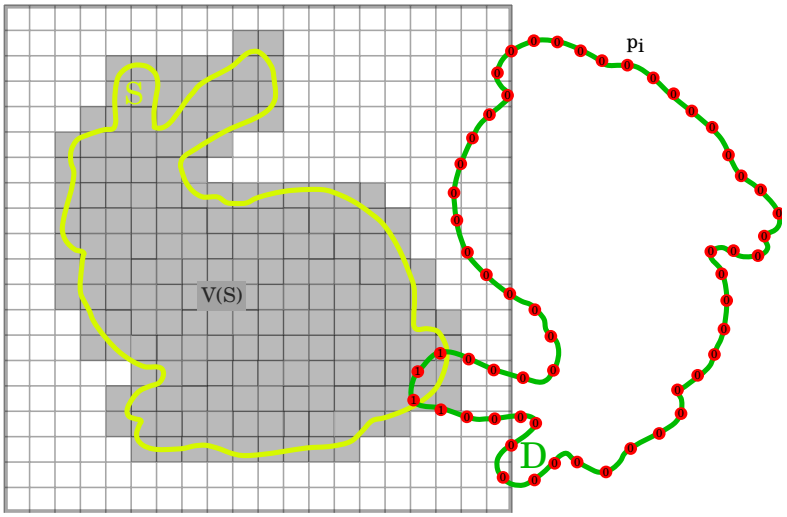


Fig. 14: Example for the Voxmap PointShell^{TM} algorithm.

idea of this algorithm is to voxelize S and sample the surface of D. We denote the voxelization of S by $V(S)$. Each voxel of $V(S)$ intersects S or is inside of S. The collision detection algorithm takes every point p_i on the surface of D and tests if it lies inside $V(S)$. If so, a collision is detected. If no point lies inside $V(S)$, the objects do not collide. For more information about VPS and how to speed up the computation, we refer to Mikel Sagardia and Hirzinger (2008). The general application of the VPS algorithm to assembly planning is evaluated in (Johnson and Vance, 2001).

The algorithm is important for our work, as we extend the VPS algorithm to calculate a special intersection volume instead of a simple collision.
Another algorithm that can be used for collision detection is the Gilbert-Johnson-Keerthi (GJK) Algorithm (Gilbert *et al.*, 1988) which is based on the computation of Minkowski differences. This algorithm can be applied to convex objects or to non-convex objects in conjunction with a convex decomposition. This algorithm is applied to motion planning if additional information like the minimum distance or maximum penetration depth (Bergen, 2001) is used in the motion planning algorithm.
At last, there are Bounding Volume Hierarchies (Gottschalk *et al.*, 1996) which are very popular and which we also use in our work. A BVH is a hierarchical partition of

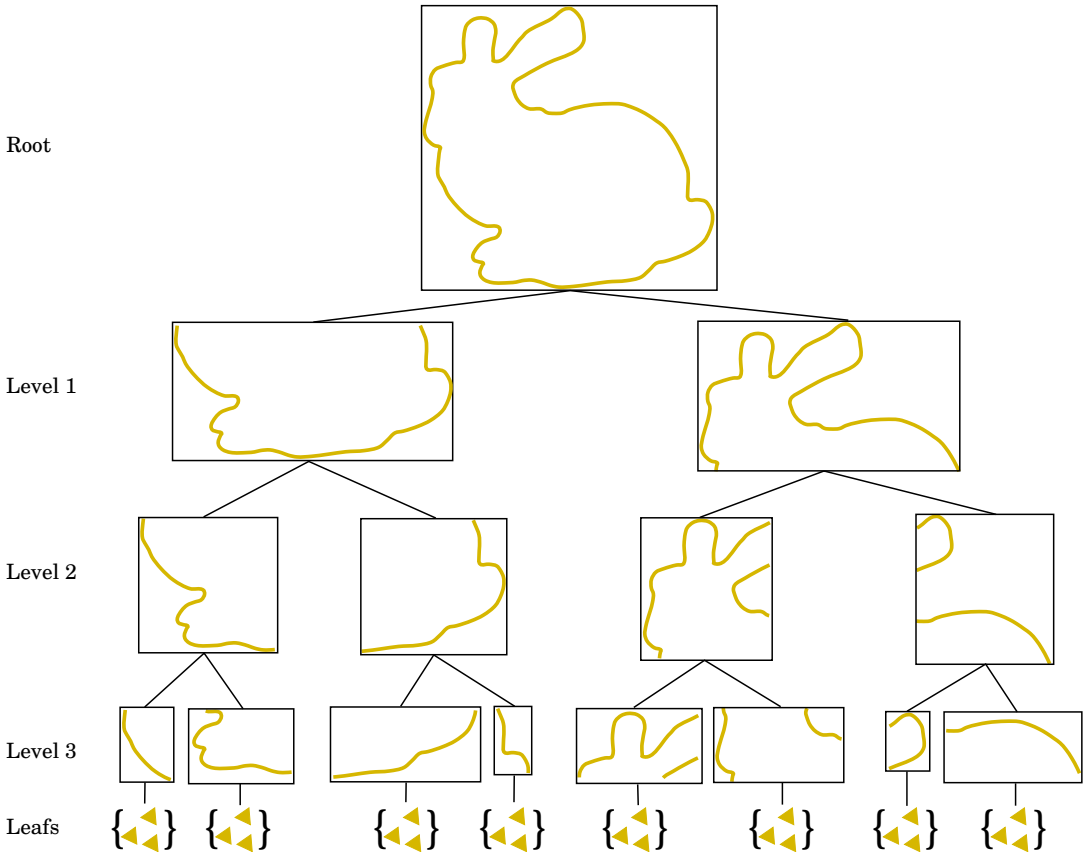


Fig. 15: Example for a Bounding Volume Hierarchy.

a given object that is stored in a tree structure as shown in Figure 15. The object is defined by a set of primitives, e.g. triangles. The root of the tree stores the bounding volume of all triangles. Each subsequent child node stores the bounding volume of the subset of all triangles belonging to its subtree. In our work, we use oriented bounding boxes (Gottschalk *et al.*, 1996) as bounding volumes for the hierarchies. For

the objects D and S this data structure is precomputed. For the collision detection query the trees of the objects are traversed. In the traversal, the bounding volumes in the nodes are tested whether they are free of collision or not. In case the node is a leaf, the corresponding triangles are tested for intersection. For our work, it is important to notice that there are two early outs in the traversal of the BVH: One for objects that are in collision and one for objects that are free of collision. In the first case, the early out is the first intersecting triangle pair that is found by the algorithm. As soon as this witness for the intersection is found, the algorithm can return the result. In our work, we present a speedup for this first early out. The second early out is the separation of the trees. When the tree is traversed only intersecting nodes are tested. In case there are no intersecting nodes, the bounding volumes separate the objects and the objects are free of collision. A deeper insight and summary about collision detection with BVH's can be found in (Ericson, 2004). A common approach to accelerate collision detection is to use problem specific bounding volumes in the hierarchy. This can range from simple bounding volumes like spheres and axis-aligned bounding boxes to complex shapes like convex hulls (Ehmann and Lin, 2001). Moreover, one can accelerate the collision detection by using the Edge Volume Heuristic (Dammertz and Keller, 2008) which subdivides triangles in order to accelerate collision detection. The Early Split Clipping (Ernst and Greiner, 2007) uses a similar approach. There, the primitives are also subdivided but only for the construction of the BVH. In the query phase, the original triangles are used. Therefore, the primitives are referenced in multiple bounding volumes. In contrast to the basic BVH approach where each primitive is only referenced once. Another well-known idea to accelerate collision detection is presented in (Leonidas Guibas and Zhang, 1999) (Gottschalk *et al.*, 1996) and (Ehmann and Lin, 2001). There, the authors use the caching heuristic which assumes that sequential configurations are only a small distance away from each other and therefore the last tested configuration can be used to accelerate the current query. The caching methods are not usable for the sampling in a motion planner as in general, the distance between two sampled configurations is not small enough. At most one could use this heuristic for the collision detection in the edge verification of the motion planner. But in challenging real-world examples with narrow passages the number of collision detection calls along the path is negligible in comparison to the number of collision detection calls in the sampling. In this work, we present a workspace acceleration data structure and an algorithm on top of the BVH for collision detection in the sampling computation of any motion planner.

At last, we sum up two basic spatial data structures that are used in this work. The

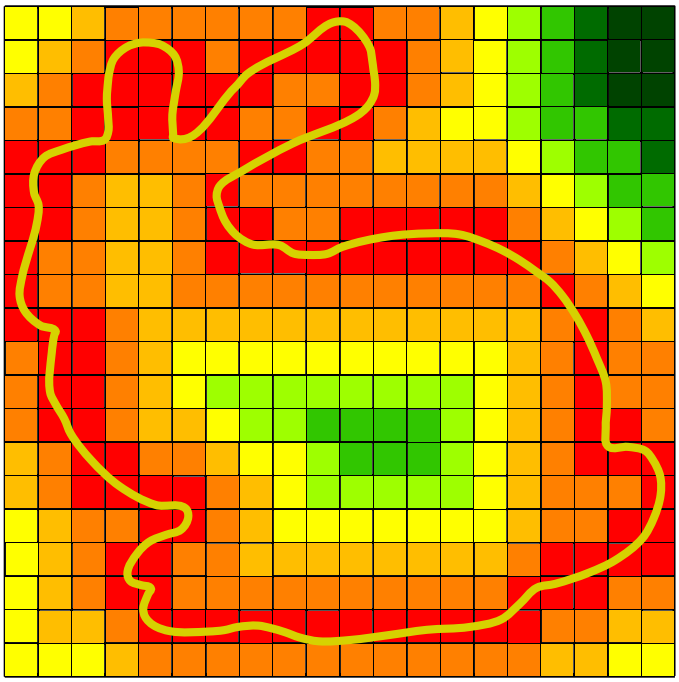


Fig. 16: Example for a Distance Field (red: low distance, green: high distance).

first data structure is the distance field. As shown in Figure 16, a distance field discretizes the objects and their environment with a grid. Each grid cell stores the minimal distance to the surface of the object. A survey on distance fields and their computation can be found in (Jones *et al.*, 2006). The second data structure is the octree. An octree is a quadtree in three dimensions. An example for a quadtree is shown in Figure 17. The resolution of the quadtree is defined by the maximal depth of the tree. To compute the quadtreewe start with the root quad, which contains the whole or parts of the object. It is subdivided into smaller quads in each level of the hierarchy. As long as the quads contain geometry and the hierarchy has not reached the maximal depth the quads are subdivided. An overview about octrees and their computation can be found in (Tang, 1992).

2.4 Nearest Neighbor Data Structures

Every sampling-based motion planner needs to calculate nearest neighbors. In our work, the particular requirements for the nearest neighborhood data structure are: High performance, dynamic updating and a query phase with arbitrary metric. Yershova and LaValle (2007) proposed a suitable nearest neighbor search. They use a

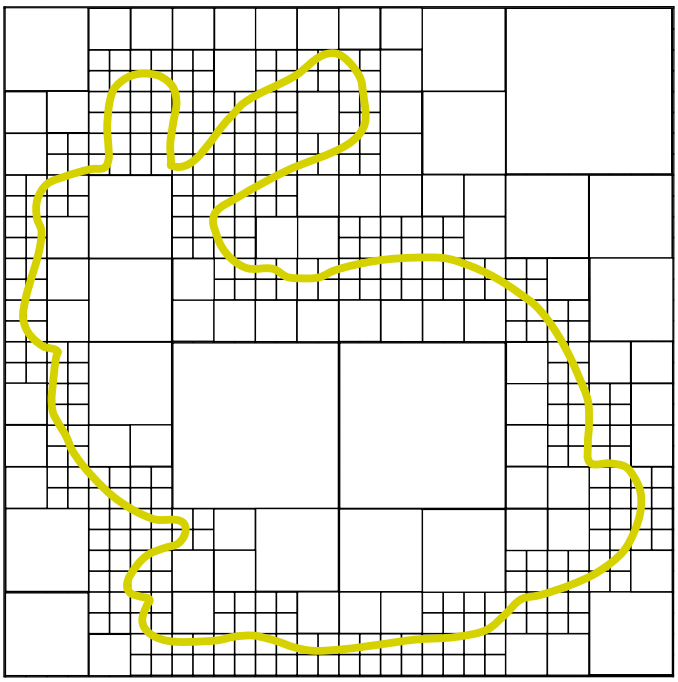


Fig. 17: Example for a quadtree with a depth of four.

spatial subdivision that is very similar to a kd-tree to find nearest neighbors in motion planning algorithms. All configurations are stored in a binary tree. The root node n_{root} of the tree stores the bounding box of the scene and each child node n_i in the tree represents a subspace. The resulting leaves n_{leaf} store at most k configurations each. The algorithm dynamically updates the data structure. During the query phase, the nearest neighbor $NN(c, C_{sampled}, u) = \min_i\{d_c(c_i, c, u) : c_i \in C_{sampled}\}$ to the configuration c with the current metric is determined. First, a point location of c takes place and identifies the leaf n_{leaf} containing c. Second, the metric is used to calculate the minimal distance d_{min} of c to all k configurations in n_{leaf}. Finally, the tree is recursively checked based on the current metric by visiting adjacent nodes and repeatedly updating d_{min}. For a deeper insight, we refer to the work of Yershova and LaValle (2007). Pan *et al.* (2010) used a data structure to find approximate nearest neighbors in a GPU-based PRM approach. Their Local Sensitive Hashing (Wang *et al.*, 2014) approach can be modified to fulfill our requirements. We examined the performance of both approaches and came to the conclusion that a Local Sensitive Hashing is only suitable for a large number of configurations as occurring in a PRM. Therefore, our work uses the data structure of Yershova and LaValle.

2.5 Deformation Models

In this work, we deal with a deformable motion planning problem. For deformable motion planning algorithms, one needs a deformation model to simulate the deformations. In the industry, the analysis of deformation properties is often realized by using deformation models based on the finite element methods (FEM). For an introduction to FEM we refer to the work of Sifakis and Barbic (2012). Determining the elastic parameters is crucial for the simulation. For an automatic determination of the parameters in the application of robot motion planning we refer to the work of Frank *et al.* (2014). For the disassembly motion planning in this work, we adjusted the FEM parameters manually as we use the FEM approach only for one reference dataset.
The simulation technique we use in this work is the position based dynamics (PBD) framework from Müller *et al.* (2007). This work uses geometric constraints to model plausible deformations in real-time. PBD is a geometric approach to a physical problem. In recent years the method proved well to approximately simulate physical behavior solely based on the geometry and the used constraints. Key benefits of the method are stability, speed, and a simple implementation. The method matches the application in this work very well as the input is plain geometry without any physical information. A comprehensive survey can be found in (Bender *et al.*, 2014). The primary ingredients of the PBD based algorithms are the geometric constraints. The constraints we use are the well-known position constraint and the bending constraint (Müller *et al.*, 2007), (Kelager *et al.*, 2010). Other well-know approaches for modeling deformations are the mass-spring systems (Liu *et al.*, 2013) or the projective dynamics approach (Bouaziz *et al.*, 2014). All those approaches mentioned above can also be used for our work. We choose the PBD approach, mainly due to performance and simplicity of the approach.

2.6 Deformable Motion and Disassembly Planning

In recent years deformable motion planning draws more and more attention. For an overview of deformable motion planning, we refer to the work of Frank (2013). In general, there are three possible approaches to solve deformable motion planning problems. First, there are approaches, like Rodriguez *et al.* (2006), Gayle *et al.* (2005) and Bayazit *et al.* (2002), which are based on physical simulations during the motion

planning algorithm. So the actual deformation is computed during the motion planner and forces resulting from the motion are applied to the deformation model. Due to the complexity and computation time for simulating deformations, these planners have a long computation time. Second, there are approaches like Frank *et al.* (2014) and Mahoney *et al.* (2010) that propose to precompute approximate deformations in a pre-processing step and use them during planning. Using precomputed deformation models speeds up the planning, but also approximates the deformation process. At last, there is the approach of motion planning without physical simulation. This is addressed by Phillips-Grafflin and Berenson (2014). Their approach plans motions for deformable objects with a voxel-based representation and measures the intersection of voxels. The intersection volume of voxels is assumed as a measure of deformation and integrated to the motion planner. In this work, we propose a motion planning algorithm for the disassembly of industrial parts that combines the second and third approach. This means we use a precomputed deformation data structure and apply ideas from the planning without physical simulation to solve the problem, Schneider *et al.* (2015a). The disassembly planning for industrial parts is also addressed by Jianwei Guoa *et al.* (2013). They use the contact information combined with fitting methods to plan the disassembly. In comparison to our work, their framework only addresses linear disassembly directions whereas the method that we present can handle arbitrary motions. Moreover, they solve the problem of fastening elements by including user interaction with a GUI whereas we propose an algorithm without user interaction.

2.7 Software and Libraries

This work is implemented in the RASAND (2010-2017) library. RASAND is a software solution for proximity queries, registration, packaging and motion planning. Additionally, there are a variety of external libraries that we use as a starting point but also as a comparison of our results with state-of-the-art algorithms. For motion planning, we compare the results of our work with OMPL (Şucan *et al.*, 2012) and the commercial software Kineo^{TM} Kite Lab (Kineo^{TM}, 2016). For comparing our approach on collision detection we use the open source Flexible Collision Library (FCL) (Pan *et al.*, 2012) and the Proximity Query Package (PQP) (Larsen *et al.*, 1999) as a state-of-the-art comparison. For the computation of the FEM results in our work we use the Vega FEM Library (Barbič *et al.*, 2012). The tetrahedral meshes that we use for the FEM simulations are computed by the TetGen library (Si, 2015).

3

THE COMPLETELY RANDOMIZED RRT-CONNECT

The approach that we will present in the following is based on our publication "Completely randomized RRT-connect: A case study on 3D rigid body motion planning". It was presented at the IEEE International Conference on Robotics and Automation in 2015 (Schneider *et al.*, 2015b).

3.1 Motivation and Main Idea

Nowadays sampling-based motion planners use the power of randomization to compute multi-dimensional motions at high performance. Nevertheless, the performance is based on two problem-dependent parameters. The first parameter u weights the importance of the rotation versus the translation of the dynamic object in the nearest neighbor search. The second parameter $range_{max}$ determines the maximal length of an edge in the tree. Former work uses constant user-adjusted values for these parameters which are defined *a priori*. In motion planning libraries these parameters are defined as constant values that proved well in many problems. However, to ensure a good performance, these parameters must be experimentally recalibrated *a priori* depending on the problem data. These experiments are very time consuming and take much longer than the actual planning and second, a certain level of algorithmic knowledge is necessary to perform the calibration.
The huge influence of constant parameters on the performance of a motion planning algorithm is exemplarily shown in Table 1. Park *et al.* (2016) used the RRTC algorithm of the open source motion planning library OMPL on a commodity personal computer using the OMPL benchmark scenarios shown in Figure 18. On a similar

Table 1: Performance comparison OMPL, Parameters.

Benchmark	**Not adjusted (Park *et al.*, 2016)**	**Manually Adjusted**
Apartment	20.15s	4.46s
AlphaPuzzle 1.5	19.92s	0.34s
Easy	0.12s	0.067s

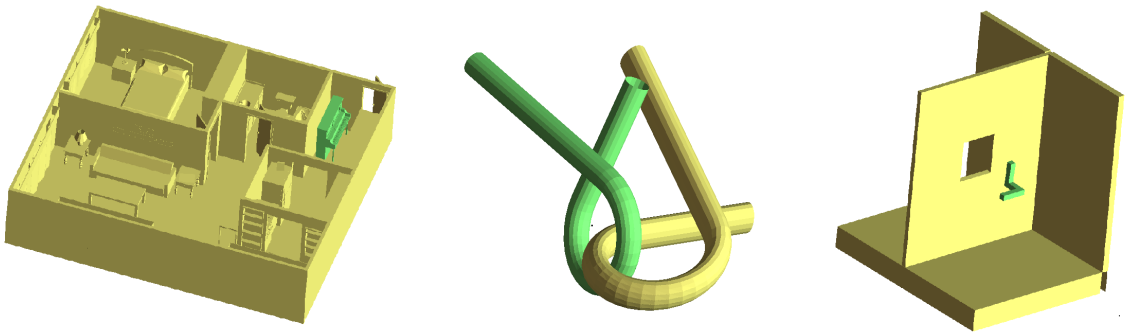


Fig. 18: OMPL benchmark scenarios. Left: Apartment; Middle: AlphaPuzzle 1.5; Right: Easy.

hardware configuration we were able to reproduce their results, but with manual adjustments of the parameters u and $range_{max}$ we obtained a much better performance which is shown in the third column of Table 1. The work of Park *et al.* (2016) did not focus on optimizing motion planning parameters as we did. Nevertheless the difference in performance shows the importance and potential that are hidden behind an optimization of these parameters. Our work proposes a new method to avoid an *a priori* specification of constant parameters. We extend the power of randomization by varying the parameters randomly during runtime. This avoids the pre-processing step to adjust parameters and moreover improves the performance in comparison to existing methods in the majority of the benchmarks. Our method is simple to understand and implement. This is the first work that introduces such a random selection of the motion planning parameters. In the following, we will extend the Rapidly-Exploring Random Tree-Connect (RRTC) (Kuffner and LaValle, 2000). Nevertheless, our work can be easily adapted to other tree-based motion planners. To compare our approach, we present a comprehensive experimental analysis of the parameters and the resulting performance. The algorithms and data structures were implemented in our library RASAND and we compare our approach with the results of OMPL (Şucan *et al.*, 2012) and the commercial software Kineo^{TM} Kite Lab (Kineo^{TM}, 2016).

3.2 Our Approach

As we randomize the parameters during runtime, we call our approach the Completely Randomized RRTC. But before we can present the overall motion planning we need to define the nearest neighbor metric and the range.

3.2.1 Nearest Neighbor Metric

In chapter 2.2.4 we presented the common metric d_c to measure distance in motion planning algorithms with $d_c(c_1, c_2, u_1, u_2) = u_1 \cdot d_t(t_1, t_2) + u_2 \cdot d_q(q_1, q_2)$. In our work we modify the neighborhood metric (1). We define a metric d'_c based on d_c with $d'_c \in [0, 1]$:

$$d'_c(c_1, c_2, u) = u \cdot d'_t(t_1, t_2) + (1 - u) \cdot d'_q(q_1, q_2); \quad u \in (0, 1). \tag{2}$$

The distance between two configurations is always a value in the interval $[0, 1]$. This unification is necessary to compare different problem data on a common basis, regardless the scaling of the problem data. To scale d_t to $d'_t \in [0, 1]$, we multiply d_t with ν^{-1} where ν is the maximum extension $||t_{max} - t_{min}||_2$ of B. B is the bounding box of the scene. With this we obtain $d'_t(t_1, t_2) = \nu^{-1} \cdot ||t_1 - t_2||_2$. To scale d_q to $d'_q \in [0, 1]$, we multiply d_q with π^{-1} and obtain $d'_q(q_1, q_2) = \pi^{-1} \cdot min\{acos(q_1 \cdot q_2), acos(q_1 \cdot (-q_2))\}$. To finally define d'_c we use a convex combination of d'_t and d'_q with the parameter $u \in (0, 1)$ and obtain (2). Equation (2) is used to determine nearest neighbors in C-space and the parameter u denotes the influence ratio of the translation versus the rotation in the distance computation.

3.2.2 Measuring the Range

The *range* is the length of an edge in the tree and $range_{max}$ is the maximal value for *range* that is allowed during the algorithm. The most straightforward approach is to use the nearest neighborhood metric (2) to measure the *range*. Obviously, in such an approach the *range* would depend on the parameter u. Our approach is different as we want to sample $range_{max}$ independently from the parameter u. In our approach

we propose to measure *range* as the maximum of the translation and the rotation of an edge:

$$range(c_i, c_j) := max\{d'_t(t_i, t_j), d'_q(q_i, q_j)\}. \tag{3}$$

3.2.3 Completely Randomized RRT-Connect

Algorithm 5: CR-RRTC (c_{init}, c_{goal})

```
1  T1.init(c_init), T2.init(c_goal)
2  while ! TimeElapsed(T_max) do
3      T_current, T_other ← SwapAmountOfSamples(T1, T2)
4      c_rand ← RandomState()
5      c_near ← NearestN(c_rand, T_current, rand(0,1))
6      c_new ← getExtend(c_near, rand(0,1))
7      if (! Trapped) then
8          T_current.addVertex(c_new)
9          T_current.addEdge(c_near, c_new)
10         c_near' ← NearestN(c_new, T_other, rand(0,1))
11         c_new' ← getExtend(c_near', rand(0,1))
12         if (! Trapped) then
13             T_other.addVertex(c_new')
14             T_other.addEdge(c_near', c_new')
15             if (Reached) then
16                 return Path(T1, T2)
17 return ApproxPath(T1, T2)
```

Algorithm 5 is a pseudo-code notation of the developed approach. We define the metric as in Section 3.2.1 and *range* as in Section 3.2.2. The only differences to the original RRTC are implemented in lines 5,6 and lines 10,11 where both parameters u and $range_{max}$ are now sampled uniformly in $(0, 1)$ in every iteration and a different swap function in line 3. The swap function uses the number of samples in each tree to determine which tree is expanded in the current iteration. We call the algorithm Completely Randomized RRTC as we do not only sample the translation and rotation

of D, but also the two major parameters u and $range_{max}$ of the algorithm. The idea behind this approach is explained in the following. Previous work uses a constant value for $range_{max}$ depending on the difficulty of the problem data. If a small value for $range_{max}$ is used, the performance of the algorithm decreases for simple benchmarks, since the expansion of the tree is slow. A large value or a greedy approach for $range_{max}$ results in a high performance for simple benchmarks, but to solve more complex benchmarks with restricted passages in C_{free}, small edges are needed to find a solution. Nevertheless, greedy approaches (see Section 2.2.4) are used frequently since the resulting lengths of the edges in an RRT planner will decrease automatically while the algorithm proceeds. This is due to the fact that the more configurations are added to the tree, the smaller the distance to the nearest neighbor or to the border of C_{free} becomes and therefore the length of the edges. The motivation behind sampling $range_{max}$ is to randomly allow large $range$ values to solve easy benchmarks. But also allow small $range$ values because short edges are needed to pass through restricted regions without solely exploring the border of C_{free}. The parameter u defines the importance of the translation versus rotation. A high value of u means that the translation is voted higher. Hence in the nearest neighbor search n_{near} has a small translational distance to c_{rand}. This is important if the translation is highly restricted and a rotation of D is needed to plan further (e.g in the Alpha Puzzles). The same goes *vice versa* for low values of u. But such knowledge is not known *a priori* and cannot be derived by an easy geometric guess. Even more, we note that the optimality of u is a local and not a global feature. For an iteration, it is not clear in advance whether translation or rotation should be favored. The key insight of our work is that using a variable value for u results in higher performance. We reason this as follows: In an RRTC many sample points are rejected. A sample point is rejected if the nearest neighbor c_{near} is trapped (see Algorithm 2, line 8). Those trapped c_{near} are configurations that are often, particularly in the beginning of the algorithm, far away from c_{start}. Which configurations are far away is defined by the metric that results from the parameter u. To expand the tree at those configurations is crucial for the algorithm to explore C. Sampling u in each iteration results in different ways, configurations can be extended and it reduces the number of trapped iterations which speeds up the expansion of the tree. Our approach increases the performance which is experimentally shown in the next chapter.

3.3 Experiments

In this section, we present the experimental results of our proposed algorithm as well as the results of our parameter study. All results presented are measured on a commodity personal computer with an Intel i7-950 processor and 8 GB of RAM, compiled under Linux with the g++ 4.8.1 compiler. All performance measures are done with at least 100 runs and the mean value is presented. All results are computed on one core of the CPU.

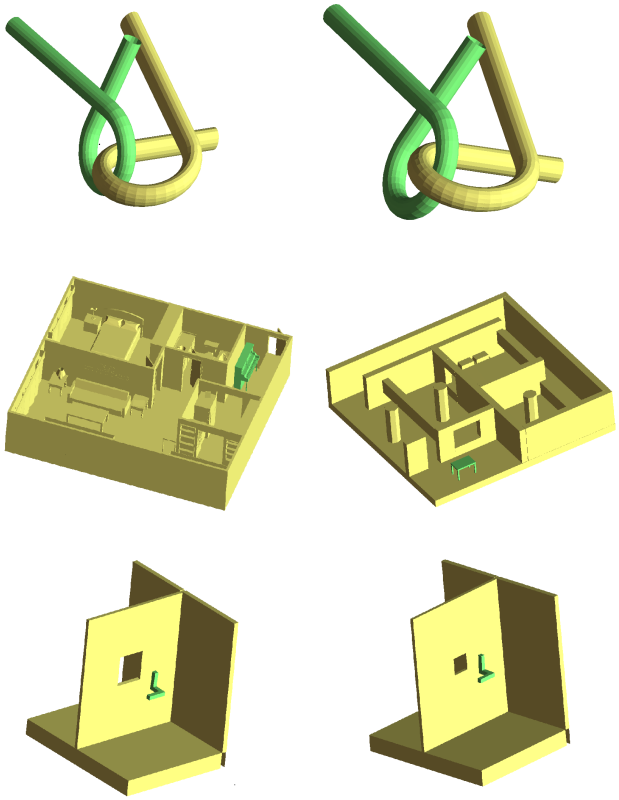


Fig. 19: Used OMPL Benchmarks (Easy/Hard): Alpha1.5/Alpha1.2, Apartment/Home, Easy/Twistycool.

As benchmark scenarios, we use six benchmarks that are supplied with OMPL (see Figure 19). Those six benchmarks are three pairs of two very similar benchmarks each with an easy and a more challenging version. Moreover, we use an easy and a more challenging version of an engine disassembly scenario that is shown in Figure 20.

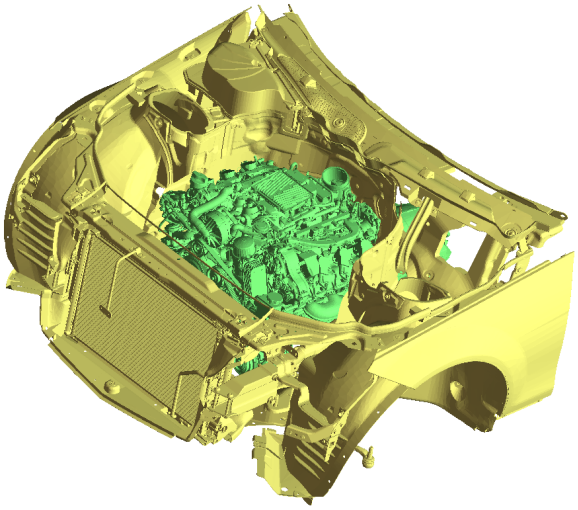


Fig. 20: Engine Disassembly Scenario.

3.3.1 Parameter Study

To show the impact of parameters on the performance, we implemented the RRTC with the metric defined in Chapter 3.2.1 and the *range* of the tree as in Chapter 3.2.2. We performed a parameter study on u and $range_{max}$ that evaluates the performance for varying values of u and $range_{max}$ with a resolution of 0.1. The performance is averaged over at least 50 runs. This parameter study was done for every benchmark, and we measure the percentage of the maximum runtime T_{max} that is defined in Algorithm 5. This provides us the optimal constant parameter pair for every single benchmark. We visualize the results of the parameter study with a heat-map. In Table 2 we see the result of the parameter study for the Alpha 1.2 puzzle. The best performance with fixed parameters for the Alpha 1.2 puzzle is achieved by choosing $u = 0.65$ and $range_{max} = 0.45$. In Table 3 the heat-map of the Apartment benchmarks is shown. The best performance with fixed parameters for the Apartment is achieved by choosing $u = 0.25$ and $range_{max} = 0.05$.

The results for the remaining benchmarks are shown in Figure 4 to Figure 9. First, we see that the heat-maps differ depending on the benchmark. This means there are no single optimal values for all benchmarks. Second, we notice that the influence of $range_{max}$ is less than the influence of u. Nevertheless, we can see our statement from Chapter 3.2.3. For simple benchmarks, like the Easy Benchmark (Figure 6) or the Alpha 1.5 benchmark (Figure 4), high values for $range_{max}$ should be favored.

Table 2: Heatmap of Alpha 1.2, Values in %, $T_{max} = 35s$.

u \ **range**$_{max}$	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	99.81	98.26	99.28	99.73	99.58	99.94	99.51	99.85	99.3	97.26
0.15	94	85.03	88.62	91.31	90.99	91.61	89.59	91.93	90.65	86.8
0.25	56.47	55.36	51.82	50.25	47.07	47.8	50.18	50.74	52.34	49.72
0.35	30.11	25.11	28.92	25.37	28.86	28.61	29.24	26.69	28.54	30.17
0.45	20.27	16.95	19.81	17.84	15.98	18.37	17.77	17.78	17.85	16.72
0.55	15.35	14.02	12.53	12.78	13.58	12.74	13.17	12.29	13.12	13.61
0.65	10.93	10.67	8.98	10.83	8.9 (min.)	9.65	9.64	10.44	9.98	10.58
0.75	13.91	10.25	11.89	10.41	9.18	10.66	10.39	11.46	9.32	10.22
0.85	34.95	27.74	25.21	22.19	19.55	19.22	24.58	25.91	27.35	20.31
0.95	75.53	64.26	67.8	71.63	68.41	67.29	61.23	59.5	59.42	67.4

Table 3: Heatmap of Apartment, Values in %, $T_{max} = 2.5s$.

u \ **range**$_{max}$	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	9.78	11.13	12.65	12.42	12.44	12.96	12.44	12.79	12.4	11.78
0.15	9.39	10.82	13.19	13.08	12.49	12.98	12.86	11.59	11.18	12.38
0.25	8.82 (min.)	10.93	13.34	13.96	13.75	12.15	13.53	12.18	12.19	12.27
0.35	10.71	16.81	17.03	15.98	16.32	15.92	16.18	15.1	16.94	15.32
0.45	15.75	17.8	21.23	19.8	18.69	22.56	20.93	19.42	19.14	22.5
0.55	19.29	22.23	22.94	20.82	21.85	21.56	22.31	20.61	22.12	19.79
0.65	19.45	22.9	21.82	20.57	21.66	24.26	24.22	22.75	21.8	22.47
0.75	28.52	27.21	26.7	29.19	28.92	28.62	28.29	26.47	26.54	25.3
0.85	42.73	37.58	35.84	35.62	37.45	38.92	38.45	40.96	41.94	40.32
0.95	100	73.92	69.5	68.62	68.22	70.43	68.1	65.56	73.63	73.02

Table 4: Heatmap of Alpha 1.5, Values in %, $T_{max} = 2.5s$.

	range$_{max}$									
u	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	100	63.16	49.38	50.53	47.98	49.95	47.49	45.29	47.34	45.84
0.15	81.45	33.33	26.13	26	25.18	24.7	26.48	25.85	24.84	27.33
0.25	31.03	16.78	15.31	14.93	16.06	15.67	14.92	16.35	14.51	15.71
0.35	17.19	11	9.92	10.1	9.84	9.93	9.47	10.19	9.83	9.62
0.45	10.57	7.65	7.19	7.03	6.89	6.86	6.84	6.48	7.05	6.37
0.55	7.11	5.09	5.13	5.14	5.31	5.26	4.78	5.13	5.04	4.95
0.65	6.3	5.1	4.83	4.28 (min.)	4.57	4.37	4.82	4.5	4.46	4.42
0.75	8.3	6.08	5.74	5.71	5.72	5.47	5.48	5.53	5.92	5.72
0.85	11.5	8.67	8.99	8.57	9.04	9.04	9.1	9.62	9.37	9.13
0.95	15.48	21.01	16.07	19.76	17.61	19.04	18.76	18.93	18.29	20.48

Table 5: Heatmap of Home, Values in %, $T_{max} = 20s$.

	range$_{max}$									
u	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	100	100	100	100	100	100	100	100	100	100
0.15	100	99.77	98.11	96.8	95.6	96.75	96.84	96.98	97.06	96.18
0.25	98.31	69.58	67.74	57.38	64.48	63.95	63.97	64.85	67.51	65.15
0.35	82.94	36.5	37.55	33.43	33.98	32.57	31.86	35.18	34.6	30.89
0.45	45.49	19.81	19.49	18.63	18.48	19.27	18.27	18.16	20.3	19.49
0.55	29.71	13.85	11.14	13.78	13.42	12.44	12.31	12.53	10.88	11.86
0.65	17.42	9.36	8.48	8.29	7.9	8.63	8.54	8.14	8.75	8.03
0.75	11.01	7.78	6.46	5.98	6.78	6.55	6.01	7.38	5.9	7.25
0.85	9.85	6.14	5.69	5.55	5.94	5.5	5.27 (min.)	5.82	5.92	5.89
0.95	13.49	7.86	7.04	7.41	7.54	7.66	7.3	7.72	7.25	7.23

Table 6: Heatmap of Easy, Values in %, $T_{max} = 1s$.

	range$_{max}$									
u	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	13.25	11.14	7.29	6.93	7.19	6.56	6.28	6.47	6.27	6.3
0.15	11.18	10.87	7.53	6.98	6.95	6.5	6.44	6.54	6.28	6.39
0.25	10.84	10.29	7.62	6.6	6.6	6.35	6.46	6.35	6.52	6.17
0.35	10.21	10.01	7.64	6.62	6.78	5.9 (min.)	6.01	6.23	6.07	6.14
0.45	10.16	9.55	6.96	6.58	6.54	6.21	6.22	6.3	6.08	5.97
0.55	9.53	9.6	7.55	6.84	6.6	6.19	6.19	6.3	6.22	6.36
0.65	8.61	9.48	7.6	6.92	6.62	5.96	6.22	6.23	6.41	6.04
0.75	7.56	9.13	7.94	6.89	6.6	6.29	6.41	6.07	6.03	6.09
0.85	7.24	8.88	8.07	7.03	6.76	6.35	6.26	6.18	6.18	6.18
0.95	7.03	8.71	7.43	7.12	6.65	6.64	7.12	6.23	6.45	6.29

Table 7: Heatmap of Twistycool, Values in %, $T_{max} = 35s$.

	range$_{max}$									
u	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	59.73	84	94.84	96.34	92.33	90.83	93.87	93.93	94.93	94.51
0.15	37.28	72.19	77.2	71.43	76.34	74.43	73.98	77.66	70.81	68.61
0.25	37.62	51.49	51.55	59.85	60.29	60.85	57.48	62.51	57.05	62.56
0.35	35.95	40.95	43.76	50.6	44.02	40.36	43.96	41.96	45.78	51.09
0.45	35.33	38.99	31.55	31.07	31.68	38.65	30.27	31.7	29.67	35.16
0.55	26.63	24.14	24.65	21.31	19.93	23.57	24.3	22.8	21.51	19.84
0.65	23.27	15.72	16.77	18.14	13.41	17.21	14.2	15.36	16.04	16.26
0.75	21.85	13.71	12.03	11.72	10.96 (min.)	11.46	14.12	13.34	12.77	10.97
0.85	28.36	15.5	11.7	11.68	12.36	12.65	13.22	11.88	11.71	14.18
0.95	53	25.49	24.57	21.08	20.2	19.32	21.66	15.56	23.03	23.49

Table 8: Heatmap of Engine1, Values in %, $T_{max} = 40s$.

	range$_{max}$									
u	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	33.44	34.49	36.68	40.5	29.2	35.02	39.66	31.54	33.36	38.53
0.15	29.46	32.58	32.36	31.83	34.46	37.48	32.75	32.67	26.03	35.9
0.25	33.3	27.32	30.28	33.18	28.28	34.45	34.43	29.37	28.76	32.87
0.35	32.92	26.64	33.25	26.59	26.11	30.04	30.41	25.34	30.78	33.75
0.45	17.93	21.07	24.08	20.97	24.14	27.24	23.42	25.15	27.07	19.84
0.55	15.98	16.39	15.3	12.78	14.17	16.61	14.11	18.58	17	12.13
0.65	10.48	13.17	13.98	14.24	12.95	10.62	9.77	11.5	15.02	13.54
0.75	12.54	12.4	11.99	11.89	12.07	13.39	8.92 (min.)	10.35	12.77	10.72
0.85	13.49	11.64	12.79	13.58	16.54	13.88	13.79	12.66	9.28	12.75
0.95	12.96	15.06	19	15.7	19.28	21.41	23.62	16.49	22.35	17.99

Table 9: Heatmap of Engine2, Values in %, $T_{max} = 40s$.

	range$_{max}$									
u	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
0.05	71.96	65.42	64.09	68.34	60.74	67.16	64.85	65.81	68.38	65.07
0.15	55.8	65.37	58.61	64.39	61.78	68.36	65.92	66.55	66.31	63.78
0.25	61.41	61.5	58.67	58.68	57.49	58.58	60.71	63.75	62.07	57.48
0.35	55.53	55.66	56.66	60.62	63.09	59.93	58.03	60.19	64.03	60.19
0.45	59.59	58.81	54.46	53.92	54.99	61.79	53.52	50.93	47.76	58.57
0.55	48.8	55.03	50.35	51.72	56.23	50.44	52.68	57.53	52.13	53.9
0.65	52.22	48.18	46.27	49.1	46.83	55	49.31	45.88	55.07	52.23
0.75	40.13	43.36	39.06	41.81	41.83	43.93	40.43	42.06	45.5	38.46
0.85	35.43	33.88	30.48	32.24	28.89	30.12	30.99	28.86	29.64	36.3
0.95	27.83	29.03	24.48	24.21 (min.)	27.57	31.74	25.3	25.36	24.85	26.6

3.3.2 Examined Algorithms

To show the performance of our CR-RRTC approach, we compare three algorithms that are implemented in our RASAND Library, one algorithm provided by OMPL and one provided by the Kineo^{TM} Kite Lab. Our algorithms, as well as the OMPL algorithm, use the same collision detection algorithm from Erbes (2013). Our approach from Algorithm 5 is denoted with CR-RRTC. Another algorithm that we use is our implementation of the RRTC with our metric (see Section 3.2.1) and *range* (see Section 3.2.2) and is denoted with Def-RRTC. This implementation uses two constant default values for $range_{max}$ and u. We determined the constant parameter values for the Def-RRTC by the mean value of all optimal parameters that we analyzed in the benchmarks of Chapter 3.3.1. We and set $u = 0.583$ and $range_{max} = 0.417$. The third algorithm is the same implementation as the Def-RRTC, but instead of using constant default values for $range_{max}$ and u it uses for each benchmark the constant optimal parameters determined by the parameter study in Chapter 3.3.1. We will denote this algorithm with Opt-RRTC. It is evident that comparing the Opt-RRTC is not fair as the optimal constant parameters are unknown in advance and can only be obtained by such a parameter study. Nevertheless, we benchmarked the Opt-RRTC to present the best performance that is possible by choosing constant values. We use the implementation of the RRTC from OMPL and denote it with OMPL-RRTC. As OMPL does not use scaling and sets $u = 0.5$, using the OMPL-RRTC without any adjustments of $range_{max}$ is not competitive to our Def-RRTC. However, we wanted to give a reference to OMPL, and so we adjusted for each benchmark $range_{max}$ as proposed in a file supplied by OMPL, which even improves performance. For Engine1 and Engine2 we set $range_{max}$ to 1.0. At last, we used the default algorithm of Kineo^{TM} Kite Lab. Kite is a closed source software, and we used it without any parameter adjustments.

3.3.3 Comparison

The results of our comparison are shown in Figure 21. First, we note that in almost all benchmarks our motion planners based on the RASAND library (CR-RRTC, Opt-RRTC and Def-RRTC) outperform the OMPL-RRTC although the same collision detection and sampling resolution is used. This especially shows the effectiveness of our

scaled metric from Chapter 3.2.1. Second, to highlight the good performance of our CR-RRTC approach, we compare it with the Def-RRTC that uses default parameter values as in previous work. We see that with exception of the Alpha puzzles, our algorithm always outperforms the Def-RRTC. For the Alpha 1.2 Puzzle we have to point out that the default constant parameter value of Def-RRTC is, by pure chance, nearby the optimal constant value (compare Table 2 and Chapter 3.3.2) which results in the good performance of the Def-RRTC. Comparing the CR-RRTC approach to the Opt-RRTC in the Twistycool and Engine1 benchmarks shows that our approach can outperform the Opt-RRTC in single benchmarks. This implies that in some benchmarks randomly selected parameter values can even be better than the optimal constant parameter values. Again we point out that those optimal values are unknown in advance. In the remaining benchmarks (Apartment, Home, Easy, Alpha 1.5, Alpha 1.2, Engine2) we see that the CR-RRTC proves a good performance that is near the Opt-RRTC. At last, the comparison to Kite and the OMPL-RRTC shows that our CR-RRTC implementation is competitive to other implementations.

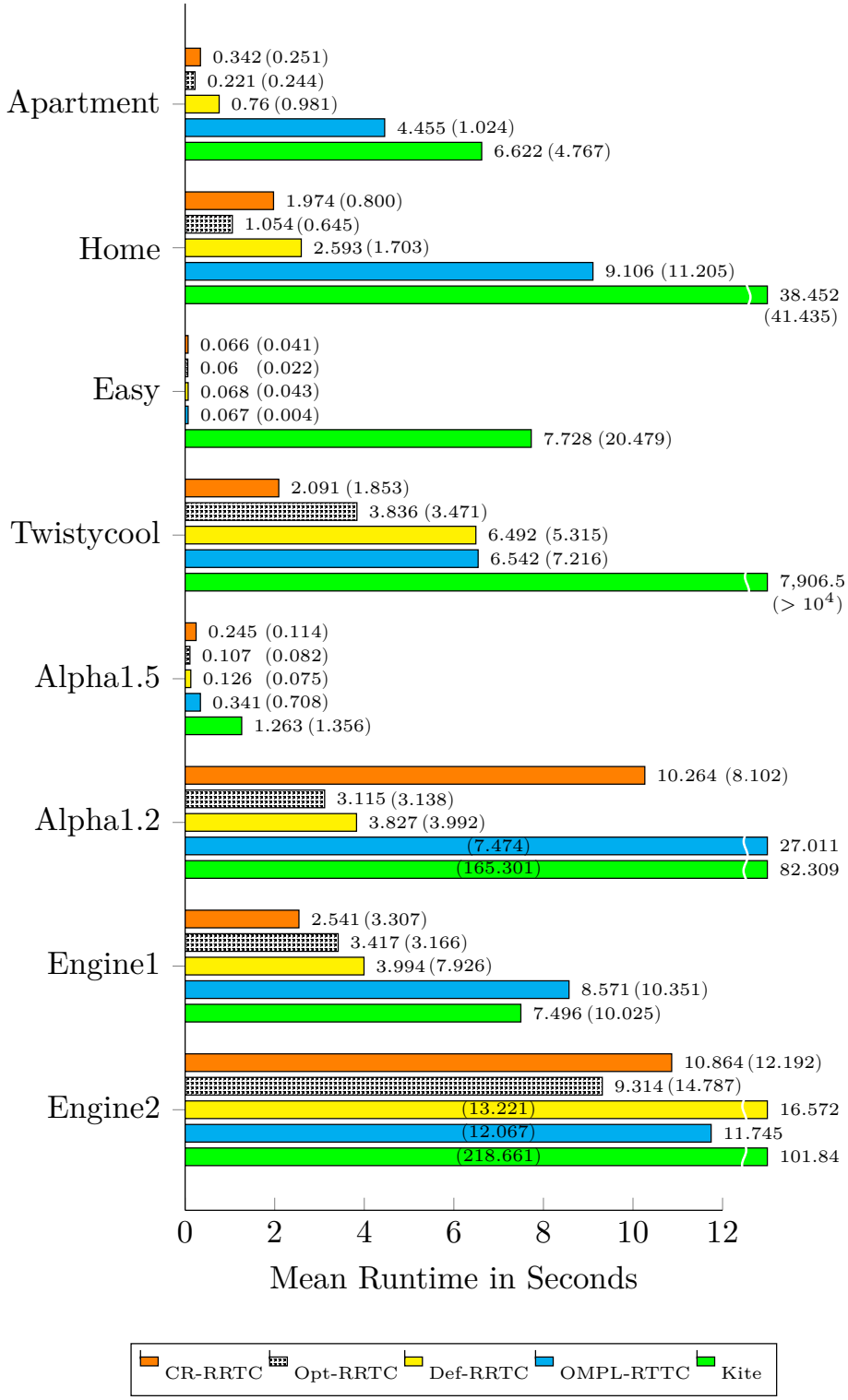


Fig. 21: Performance comparison of our approach based on the mean value (standard deviation) in seconds.

4

THE FAST ALTERNATING RANDOM RAY APPROACH

The approach addressed in this chapter is based on our publication "Collision Detection for 3D Rigid Body Motion Planning with Narrow Passages". This paper is accepted for publication on the IEEE International Conference on Robotics and Automation 2017 and will be presented at the upcoming conference in May 2017 (Schneider *et al.*, 2017).

4.1 Motivation and Main Idea

In Section 2.2 we presented the rapidly-exploring random tree (RRT) (Lavalle, 1998) for single-queries and roadmap-based methods like the probabilistic roadmap methods (PRM) (Kavraki *et al.*, 1996) for multiple queries. These motion planners need to perform the following four major subroutines:

1. Sampling valid configurations.
    a) Sampling strategy.
    b) Workspace data structure and algorithm for collision detection.
2. Near neighbor search.
3. Edge verification.
4. Shortest path in a graph.

The focus of current research is the acceleration of the subroutines noted above. Subroutine 4) was studied comprehensively over the last decades in many research fields. Subroutine 3) is addressed by lazy approaches like Lazy Probabilistic Roadmap Methods (Bohlin and Kavraki, 2000) and by motion planning using the Lower Bounds Algorithm (Salzman and Halperin, 2015) which reduces the number of edge verifications. These approaches proved to increase the performance by delaying the edge verification until it is necessary. With such lazy approaches the runtime of subroutine 3) is almost negligible and the main tasks remain subroutines 1) and 2) (Salzman and Halperin, 2015, page 4171, Figure 5). Subroutine 2) is also well-studied in many research fields. In the field of motion planning an algorithm to increase the performance of the near neighbor search by applying random grids is presented in (Kleinbort *et al.*, 2015). In subroutine 1) we have to compute valid samples. Therefore, we have to choose a sampling strategy (1a) like the uniform sampling, the Gaussian sampling (Boor *et al.*, 1999) or the bridge test (Hsu *et al.*, 2003) (see Section 2.2.5). Also, we need a data structure and an algorithm to classify samples as valid/invalid (1b). The topic of this chapter is collision detection in subroutine 1b).

As presented in Chapter 2, two well-studied algorithms for collision detection are the bounding volume hierarchy (BVH) (Gottschalk *et al.*, 1996) and the Voxmap PointShell^{TM} algorithm (VPS) (McNeely *et al.*, 2005). The decision which algorithm to choose for optimal performance in motion planning depends on the problem.

The advantage of the BVH is that in the case that the objects are free of collision, the algorithm can detect this fast with the separation of the hierarchies. In the case that the objects are in collision, the traversal of the hierarchy enables the algorithm to focus on regions of the objects that are close to each other. Note, that in the latter case this comes at the price of at least as many expensive bounding volume intersection tests as the depth of the BVH indicates. The VPS algorithm, in contrast, has the advantage that it heavily relies on simple look-ups in a three-dimensional array. Single look-ups are inexpensive, and if the objects have a large intersection volume, the VPS algorithm can detect collisions fast. But the VPS algorithm (without an additional broad phase) does not focus on regions that are near each other. Therefore, the algorithm might end up testing many or all points of the pointshell in the case of a configuration with a small intersection volume or free of collision. For disassembly motion planning with narrow passages, most of the configurations are in collision with small intersection volumes. The BVH approach is mostly chosen for motion planning problems with this characteristic although the BVH has to be traversed for almost every configuration from the root to at least one leaf.

In this chapter, we introduce a data structure and an algorithm that makes use of the characteristic of the samples. To solve the collision query for motion planners efficiently it is important to have an approach that has a very fast early out if the objects are slightly in collision. Neither the BVH nor the VPS algorithm provides this. Our approach achieves such an early out by taking the basic idea of the VPS algorithm, combining it with a custom data structure based on an octree and a distance field, using a custom variant of the swap algorithm (Llanas *et al.*, 2000) and applying this to a BVH computation. Our main idea is to introduce a data structure and an algorithm that is executed *a priori* to the BVH traversal. Our approach can detect a witness for a collision in a fraction of the time a BVH needs because it is solely based on array look-ups. This data structure and algorithm without the backup of a subsequent BVH traversal is semi-approximate. This means, if our algorithm computes that the objects are in collision, they are really in collision and we have a witness. If our algorithm states that they are free of collision, we cannot guarantee this result. Only in this case, we execute a subsequent BVH traversal to verify the result. Obviously, the results of our approach together with the subsequent BVH traversal are exactly the same results as using solely a BVH. Later we will show that for three-dimensional rigid body motion planning with narrow passages our approach achieves a speedup of up to 5.0 compared to well-established collision detection libraries like the Proximity Query Package (PQP) (Larsen *et al.*, 1999) and the Flexible Collision Library (FCL) (Pan *et al.*, 2012).

4.2 Our Approach

4.2.1 Fast Alternating Random Rays Algorithm

The algorithmic idea of our approach is based on the Swap algorithm of Llanas *et al.* (2000). This is an algorithm for the distance computation between convex objects. Algorithm 6 is a modified version of the Swap algorithm. Given two convex polyhedra O_1, O_2, the general idea is that the algorithm starts at a point a of object O_1. From this point on we alternate between the object surfaces in the following way. We iteratively compute the nearest point b to a on O_2 as well as the nearest point a' to b on O_1 (line 6-7). In our work, we call all the line segments between a and b rays. The algorithm converges to the unsigned distance minimum between the two objects (Llanas *et al.*,

Algorithm 6: ModifiedSwapAlgorithm(O_1, O_2, a)

```
if (inside(a,O2)) then                                   1
    return true                                          2
else                                                     3
    b ← 0                                                4
    while (a ≠ b) do                                     5
        b ← nearestPoint(a, O2)                          6
        a′ ← nearestPoint(b, O1)                         7
        if (inside(b, O1)) then                          8
            return true                                  9
        if (inside(a′, O2)) then                         10
            return true                                  11
        if (a = a′) then                                 12
            return false                                 13
        a ← a′                                           14
    return true                                          15
```

2000). The difference between Algorithm 6 and the Swap algorithm from (Llanas *et al.*, 2000) are several early outs we introduce as our objective is collision detection instead of distance computation. All possible outcomes of Algorithm 6 are shown in Figure 22.

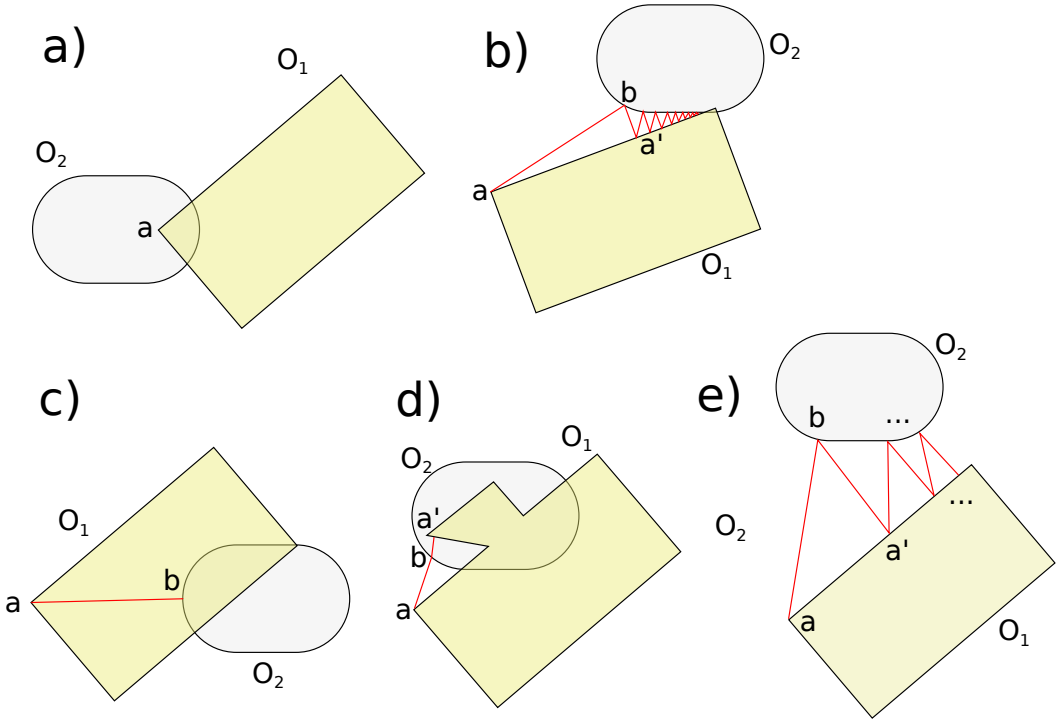


Fig. 22: Collision detection with a random ray.

Case a) results if the starting point a of O_1 is inside O_2 which witnesses the collision (Algorithm 6, line 1). In case b) the random rays converge and for some iteration a and b are equal and the objects intersect (Algorithm 6, lines 5 and 15). In case c) an intersection is found if in some iteration the point b is inside O_1 (Algorithm 6, line 8-9). The same holds in case d) for the point a' inside O_2 (Algorithm 6, lines 10-11). Note that case d) can only occur for non-convex objects. In the last case, if in some iteration a and a' are equal, the objects are free of collision (Algorithm 6, lines 12-13).

Algorithm 6 works correctly for convex objects but can also be used to semi approximately detect collisions for arbitrary non-convex objects. If we apply the algorithm to arbitrary objects, we can no longer guarantee that we find a collision of the objects, as the rays converge to a local distance minimum and not necessarily to the global minimum. But if the algorithm reports a collision, then this outcome is definitely correct and we have a witness for collision. For objects with a complex geometry, the probability of detecting a collision with a single starting point is not sufficiently high.

Algorithm 7: SwapAlgorithmNonConvex(O_1, O_2, k)

```
for i ← 0 to k do                                              1
    a ← randomVertex(O1)                                       2
    if (ModifiedSwapAlgorithm(O1, O2, a)) then                 3
        return true                                            4
return false;                                                  5
```

Algorithm 8: FastAlternatingRandomRays(O_1,O_2, k, T)

```
applyTransformation(O2, T)                                     1
if (SwapAlgorithmNonConvex(O1, O2, k)) then                    2
    return true                                                3
else                                                           4
    return CDwithBVH(O1, O2)                                   5
```

Therefore we execute the algorithm k times with different randomly selected starting points. This is shown in Algorithm 7. The question arises how large k needs to be, in order to achieve a sufficiently good semi-approximate collision detection. This question is addressed in the experiments.
In Algorithm 8 we present the overall collision detection algorithm. We use Algorithm 7 as an early out for a BVH. If Algorithm 7 computes an intersection, we can return the result. If not, we execute the bounding volume hierarchy traversal. We call Algorithm 8 the Fast Alternating Random Rays (FARR). By running Algorithm 7 *a priori* to the BVH traversal we obtain a high performance early out in case of collision, but also an overhead in case the objects are free of collision or the collision is not found. To keep the overhead minimal, we have to supply Algorithm 6 with a data structure that can compute at high performance

- whether a query point p is inside an object O and
- the nearest point to p on the surface of the other object.

This data structure is addressed in the next chapter.

4.2.2 Data Structure

In three-dimensional rigid body motion planning, we plan paths for a dynamic object through an environment with several static obstacles. Without the loss of generality, we can always assume one dynamic and one static object. If the environment consists of several obstacles, we can simply join these objects and treat them as one obstacle. The data structure we present in the following can be used with arbitrary translations and rotations. That means we need to compute the data structure only once for each of the objects for the whole planning. The goal of our data structure is to maximize the performance in case of collision and minimize the overhead in case of no collision. The most computationally expensive task in Algorithm 6 is the computation of the nearest point to a query point p on the surface of an object O as well as the inside test. We solve both tasks with a discretization. We build up a precomputed look-up table for the object O. The realization is a signed distance field D which is implemented using a three-dimensional array of cells. It stores the following information in each cell:

1. Whether the cell is inside/outside O or intersected by a triangle of O,
2. the nearest point on the surface of O to the cell center and
3. the triangle this nearest point lies on.

We define the size of D as 1.3 times the bounding box of O. With this limited size, we avoid the computation of a distance field for the whole scene. But a query point p might lie outside of D. Especially in Algorithm 6 we need to locate arbitrary points p in D and we need to take this into account. In case p is inside of D we can compute the corresponding cell with an integer division and get the information noted above. In case p is outside, we apply an orthogonal projection of p onto the boundary of D and get p_{ortho}. The cell for p_{ortho} is then used as the cell for p. For Algorithm 6 we need to build up two data structures D_1 and D_2 for O_1 and O_2 as we need to query both objects. The third information stored in the data structure are the triangles of the nearest points. We use this information in line 5 of Algorithm 6 if we reach $a == b$. Then we test the triangles of the nearest points to a and b for an intersection. This test is necessary as, due to performance, we do not want to test the criteria $a == b$ with floating point precision but with an ϵ. We choose ϵ to be as small as $6.0 \cdot 10^{-6}$ times the extension of the dynamic object. To avoid false positives with the introduction of

ϵ, we need to test the triangles for an intersection.
Although we limit the size of D, the problem of distance fields is their high pre-computation time and their storage. To reduce the computation time we propose to replace the three-dimensional array by an octree. This means we stop the subdivision as soon as the corresponding node is completely inside/outside the object. Using the octree, the cell size is large in empty regions and fine near the surface of the object. Note that the octree is only used for the construction of the data structure. In the query, we use the complete 3D array of cells to retain the constant look-up time. Each array cell stores the information provided by the corresponding octree node.

4.3 Experiments

In this chapter, we evaluate our approach. We analyze the number of random rays and the performance in motion planning. The results are benchmarked on a commodity personal computer with an Intel i7-950 processor and 8 GB of RAM, compiled under Linux with the g++ 4.8.1 compiler. All our algorithms as well as external algorithms use double precision and are executed on a single core of the CPU. For our octree-based distance field, we assume a hierarchy level of 8 which corresponds to a resolution of $2^8 = 256$.

4.3.1 Benchmarks

To benchmark our approach, we use six disassembly motion planning benchmarks which are shown in Figure 23. The dynamic objects are colored in green, whereas their environments are colored in yellow. The first three benchmarks are from our applications (Hofmann, 2015) and represent data similar to those in a car interior. The other three benchmarks are the well-known Flange problems in three decreasing difficulties 1.0, 0.95 and 0.9. Flange 1.0 is the original dataset and is quite challenging. For Flange 0.95 and Flange 0.9 the diameter of the tube is shrunken by 5 and 10 percent which reduces the difficulty. In Figure 23 the initial, free of collision, configurations are shown. The motion planning problem is to find a collision free path from this initial configuration to a configuration which represents a disassembled state, e.g. the separation of the bounding boxes. An example of a solution path for dataset 3 is

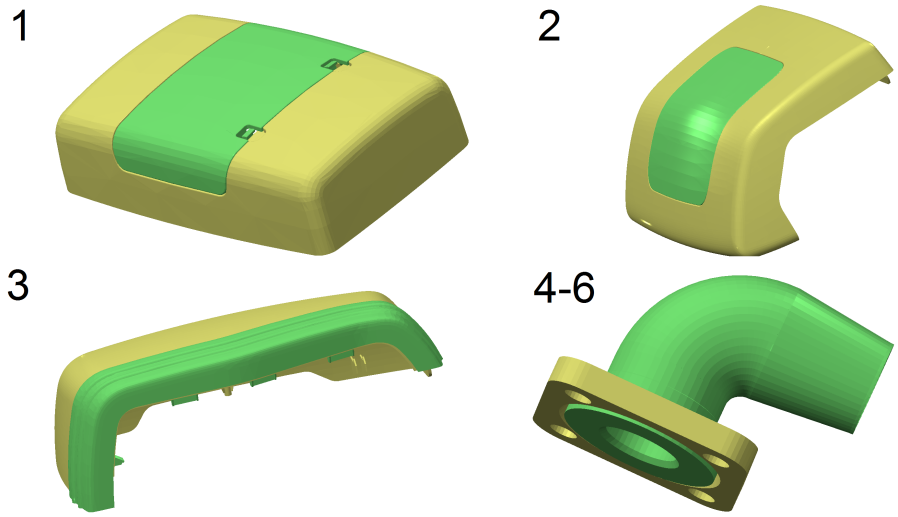


Fig. 23: Used benchmark dataset.

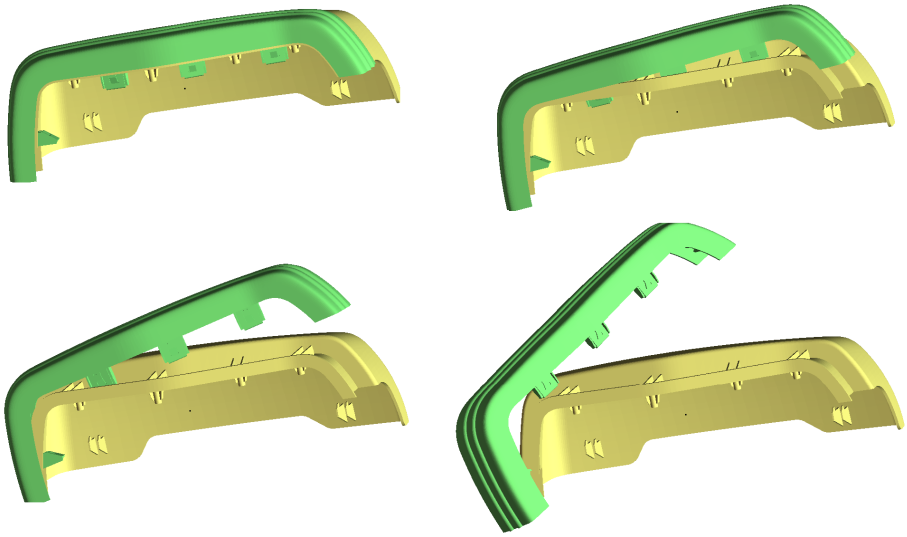


Fig. 24: The Solution for dataset 3.

shown in Figure 24. The important motion planning properties of the benchmarks are shown in Table 10. First, we consider the number of triangles. Dataset 1 and 3 are exported from a CAD Software with a precision of 0.2 mm. This is a very coarse but common precision for motion planning in the car industry. The intention is to keep the number of triangles as small as possible to reduce the computation time for collision detection in the motion planner. In dataset 2 we increased the precision to analyze the performance of our approach with large numbers of triangles. The Flange problems, dataset 4 to 6, also have a small number of triangles, similar to dataset 1. Next, we consider the number of samples that are needed to plan a path with a motion

Table 10: Properties of the benchmark set.

Dataset	#1	#2	#3	#4	#5	#6
Triangles	6,684	280,589	35,698	6,296	6,296	6,296
Samples	1,374,084	1,023,147	84,232	168,618	26,141	17,908
Invalid Portion	99.80%	99.80%	96.22%	89.55%	44.48%	30.10%

planner. The two most challenging problems are datasets 2 and 3. Dataset 1 and 4 have a medium difficulty, and dataset 5 and 6 are easy problems. This is also reflected by the invalid portion of the samples. The invalid portion is the percentage of all tested samples that are in collision. The higher this portion is, the more challenging is the planning of a path through the narrow passages of the motion planning problem. The motion planner we use is from Erbes (2013) and is based on an Expansive Space Tree (LaValle, 2006). This algorithm can solve all six benchmarks in less than an hour. But we also tested the results for the Flange benchmarks with the Open Motion Planning Library (OMPL) (Şucan *et al.*, 2012) and obtained similar results.

4.3.2 Number of Random Rays

In this section, we answer the question how to choose a good value for k. The number of random rays k has an influence on two subjects. First, it has an influence on how many collisions are detected by Algorithm 7. In the following, we call this the detection rate. The detection rate is the percentage of all invalid configurations that are detected as invalid configurations. Second, it has an influence on the overhead that is produced in the case Algorithm 7 does not find a collision. With a high value for k, we expect the algorithm to have a high detection rate but also a large overhead and *vice versa* for a small value of k.

Detection rate: The detection rate of Algorithm 7 for each benchmark is presented in Figure 25. We used for each benchmark 10,000 invalid samples which are generated by the motion planner. We measured the detection rate with an increasing number of k. We observe that for the datasets 1,2 and 4 we achieve a high detection rate of $> 90\%$ with only 30 random rays. For the other three datasets, the detection rate is worse. At first glance, one could think that the shape of the geometry is the primary factor for the detection rate. That this is not the case can be concluded from the fact that the Flange 1.0 problem (#4) has a significantly better detection rate than the other

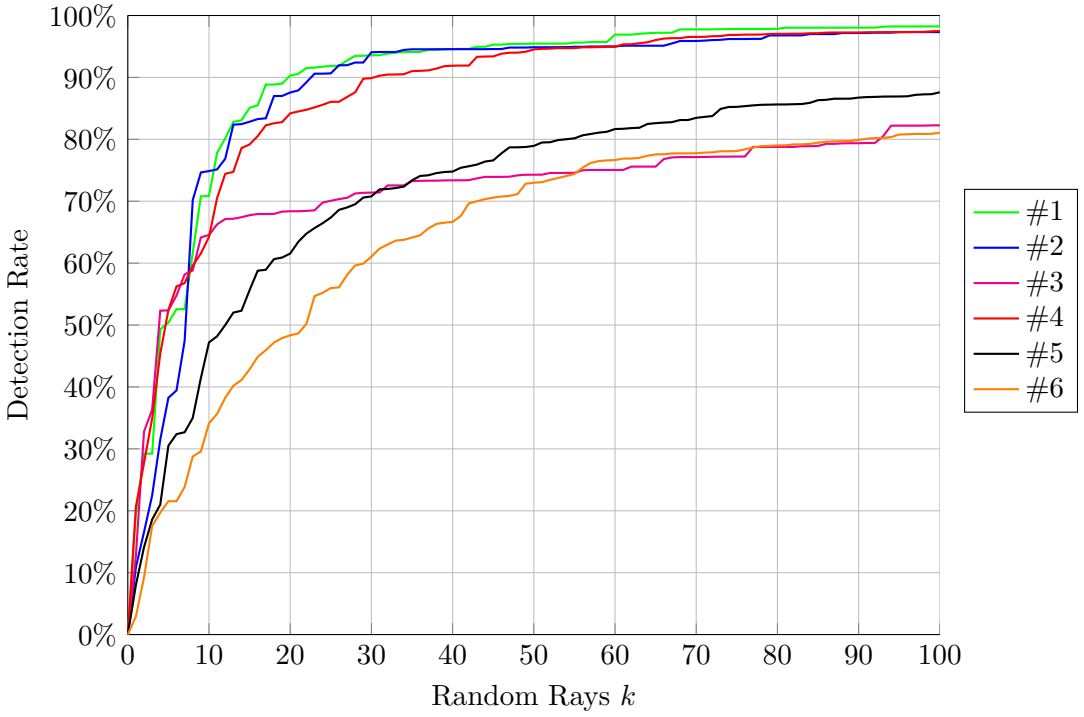


Fig. 25: Detection rate for each benchmark set with a disassembly planner.

two Flange problems (# 5 and #6) although the shape of the geometry is almost the same. The main contribution to the detection rate comes from the characteristic of the tested samples. If the configurations have small contacts at multiple local positions, Algorithm 7 can miss intersections. So the detection rate depends on the motion planning problem. It also depends on the used motion planner. In this work, the samples of Figure 25 are generated by a tree-based motion planner. The other large group of motion planners is the roadmap-based motion planners like the Probabilistic Roadmap Method (PRM) (Kavraki *et al.*, 1996). In this motion planners, the samples are drawn uniformly from a predefined bounding box around the environment. In such a sampling strategy more configurations present large intersection volumes. We used the PRM from the Open Motion Planning Library (Şucan *et al.*, 2012) and evaluated the detection rate similar to Figure 25. As expected, the result is that for the samples of such roadmap-based motion planners only 15 random rays are enough to ensure a detection rate of >90% for all benchmarks. But roadmap methods are not suited for disassembly problems and therefore are not able to solve our problems in a reasonable time and on a commodity personal computer with only 8 GB of RAM. Nevertheless, we want to point out that our approach translates well to other planners like the PRM

as we will show later in this chapter.

Overhead: We investigated the worst-case overhead of our approach. We have done this by executing our approach for each benchmark on 10,000 valid samples. So our approach produces an overhead in each single sample. We observed two things. First, the overhead is, as expected, linear in the number of random rays. But second, the absolute overhead is similar for all benchmarks regardless of the number of the triangles. This is explained by the fact that Algorithm 1 is only based on a constant lookup time in a 3D array and that we stop the algorithm (Algorithm 1, line 5) after 19 iterations.

Number of Random Rays: To choose an appropriate value for k, we analyzed the overhead relative to the BVH runtime and decided to choose $k = 30$. With this value, the overall FARR algorithm is, in our benchmarks, not slower than the pure BVH approach even if we consider simple examples with a small number of triangles like in dataset 6. The overall performance for all benchmarks with $k = 30$ is presented in the next section.

4.3.3 Performance for the Disassembly Planner

To measure the performance of our approach, we compare it to the BVH implementation of the Flexible Collision Library (FCL) (Pan *et al.*, 2012) and the Proximity Query Package (PQP) (Larsen *et al.*, 1999). Both libraries are from the same group, and our investigation showed that they perform almost the same and the results are equally good. Therefore we always state them together in the following. For the

Table 11: Performance speedup to FCL/PQP.

Dataset	#1	#2	#3	#4	#5	#6
Speedup	9.47	7.63	2.35	2.12	1.15	1.04

performance comparison, we executed our motion planner on all our benchmarks. We first checked all generated samples for collision with FCL/PQP and measured the runtime for collision detection on each dataset. Afterward, we enhanced FCL/PQP with our FARR approach and measured the computation time again. The speedups are presented in Table 11. We observe that our approach can accelerate the computation by a factor of 2.0 to 9.5 for the medium and hard benchmarks (#1 - #4). For simple benchmarks (#5 - #6) we are able we to achieve similar performance as the pure BVH implementation of FCL/PQP. In the later cases, the acceleration at the invalid

Table 12: Overall motion planning performance.

Dataset	#1	#2	#3
FCL/PQP: Pre-Processing	0.03s	1.47s	0.16s
FCL/PQP: Other MP Tasks	0.71s	0.68s	0.14s
FCL/PQP: Collision Detection	344.41s	1037.86s	36.57s
Sum:	**345.15s**	**1040.01s**	**36.87s**
FARR: Pre-Processing	28.27s	73.88s	21.01s
FARR: Other MP Tasks	0.71s	0.68s	0.14s
FARR: Collision Detection	36.36s	136.03s	15.56s
Sum:	**65.34s**	**210.59s**	**36.71s**
Speedup:	5.28	4.94	1.00

Dataset	#4	#5	#6
FCL/PQP: Pre-Processing	0.03s	0.03s	0.03s
FCL/PQP: Other MP Tasks	0.50s	0.11s	0.04s
FCL/PQP: Collision Detection	118.84s	30.91s	37.43s
Sum:	**119.37s**	**31.05s**	**37.5s**
FARR: Pre-Processing	8.70s	11.31s	12.19s
FARR: Other MP Tasks	0.50s	0.11s	0.04s
FARR: Collision Detection	56.18s	26.87s	35.93s
Sum:	**65.38s**	**38.29s**	**48.16s**
Speedup:	1.83	0.81	0.78

configuration balances the overhead for the valid configurations.
At last, for the overall motion planning performance, we must take the pre-computation time for the data structures as well as the time of the other motion planning tasks into account. This is presented in Table 12. As expected, computing a BVH is much faster than additionally computing our octree-based distance field. This has an influence on the overall motion planning performance. For the simple datasets (#5 - #6) our approach should not be used. But for the real challenging problems with narrow passages (#1 - #4) our approach has a substantial performance advantage. We achieve a speedup of up to 5.0 in these benchmarks. Note, that we assume the motion planner to be queried only once. With multiple queries, the overall motion planning performance of our approach improves even further.

4.3.4 Performance for Roadmap Methods

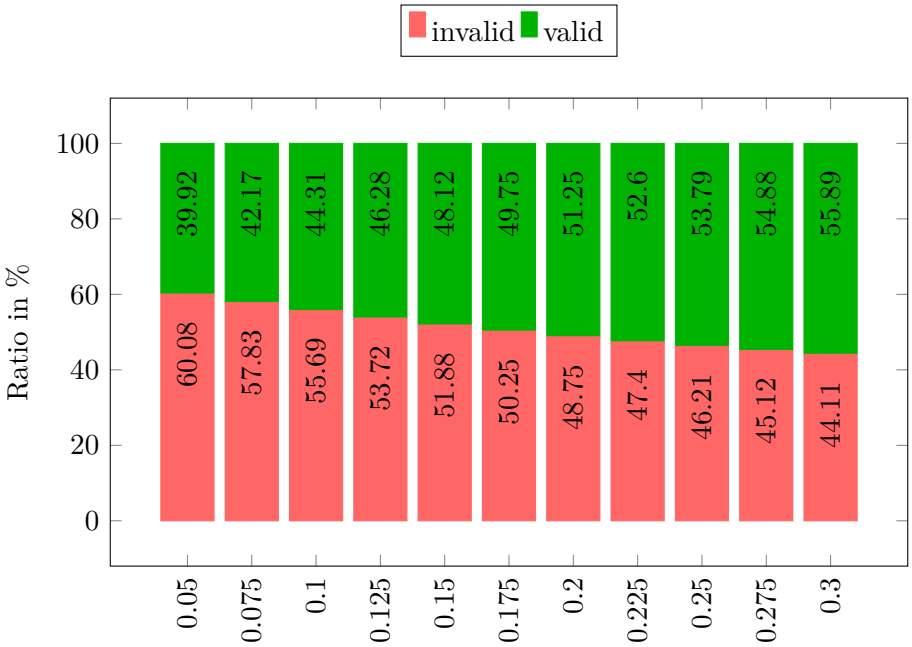
In this chapter, we want to address two additional topics. First, we have already evaluated our approach on a tree-based planner that is tuned for narrow passages in disassembly problems. The invalid portion of over 96% is characteristic for this kind

of planners. We now show that our approach is also able to improve the performance with roadmap methods.
Second, we will show that we can also improve the performance of a highly optimized BVH implementation from Erbes (2013) in addition to the public available BVH implementation of the Flexible Collision Library (FCL) (Pan *et al.*, 2012) and the Proximity Query Package (PQP) (Larsen *et al.*, 1999). As reference benchmark in this chapter, we use benchmark #3 from Figure 23. We use benchmark #3 as this is the only real world benchmark dataset from Figure 23 that can be solved by a roadmap method using 1,000,000 samples.

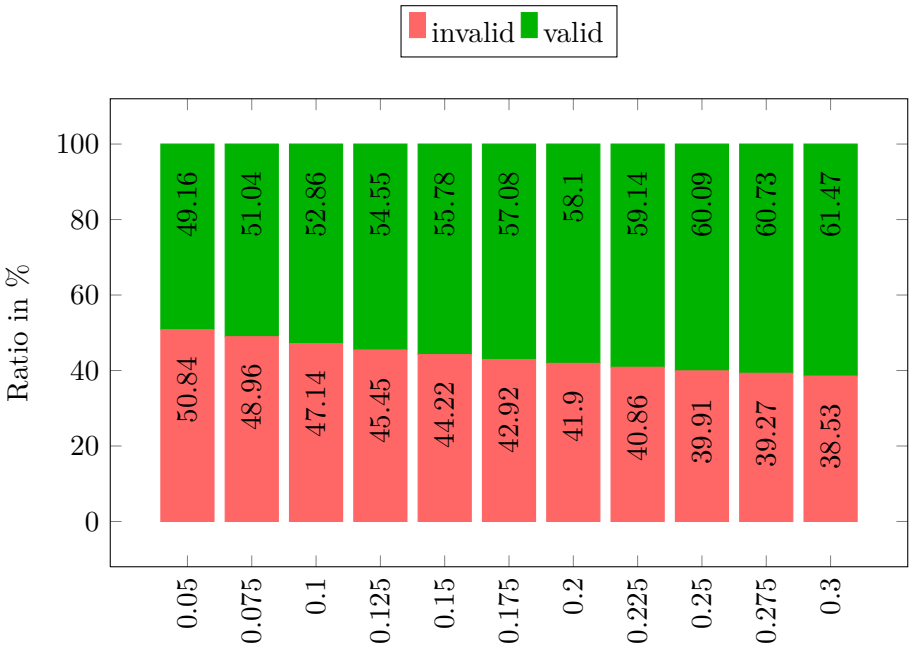
Table 13: Computation of 1,000,000 valid uniform samples.

Algorithm	**Speedup**	**Invalid Samples**	**Inv. Portion**
FCL	1.0	$\sim$ 613.600	$\sim$ 38.03%
Erbes (2013)	1.31		
FCL+FARR	1.97		
Erbes (2013)+FARR	2.28		

The invalid portion in the sampling of a motion planner differs, depending on the problem data as well as the sampling strategy. For roadmap methods with a uniform sampling the ratio of valid and invalid configurations depends on the size of C_{free} and C_{obs}. This is related to the definition of the planning region, the region where a path is computed by the motion planner. We assume in the following experiments that the planning region is reasonably defined. We choose two times the extension of an axis aligned bounding box around the objects. We now benchmark our approach with various sampling strategies. First, we uniformly sample the planning region of the car part benchmark. The results are shown in Table 13. We generated one million uniform samples and observed a speedup of up to 2.28. The invalid portion of our planning problem was about 38%. For most challenging motion planning scenarios, a uniform sampling strategy is not sufficient and different sampling strategies, like the Gaussian sampling and the bridge test, are applied. For the Gaussian sampling and the bridge test, the invalid portions are different. Both sampling methods use the parameter σ^2 for the normal distribution. In the Gaussian sampling, σ determines the distance to C_{obs} and in the bridge test the length of the bridge. In Figure 26 we investigate the invalid and valid portions of the sampling for our car part scenario for different values of σ^2. We observe that the portion of invalid samples is always higher than the 38% from the uniform sampling. Moreover, for small σ^2, the invalid portion is up to 60%. This makes our approach especially efficient for this kind of sampling as shown by the performance results in Figure 27. There we investigate the speedup with the two

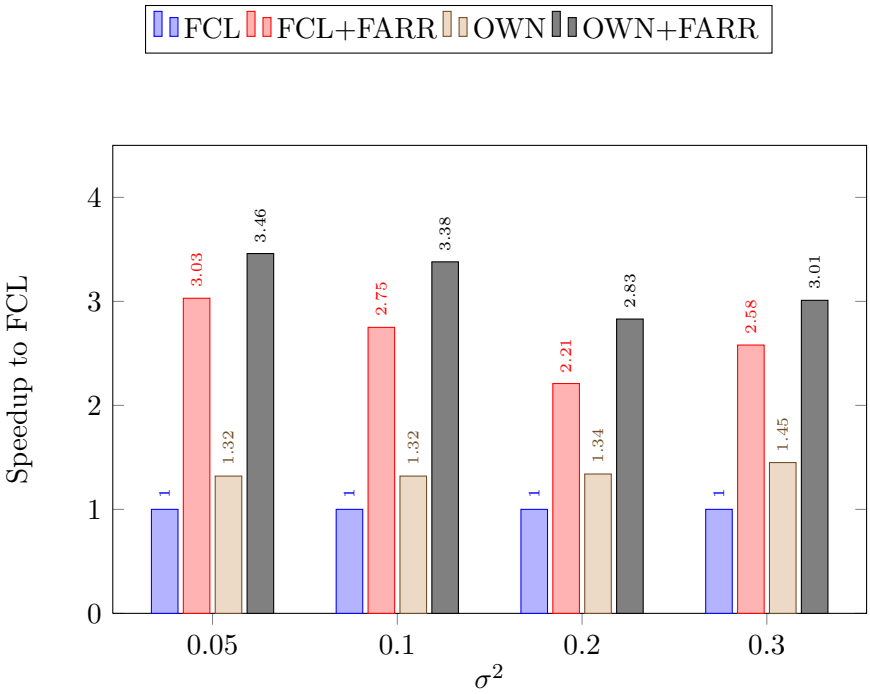


(a) Bridge Test

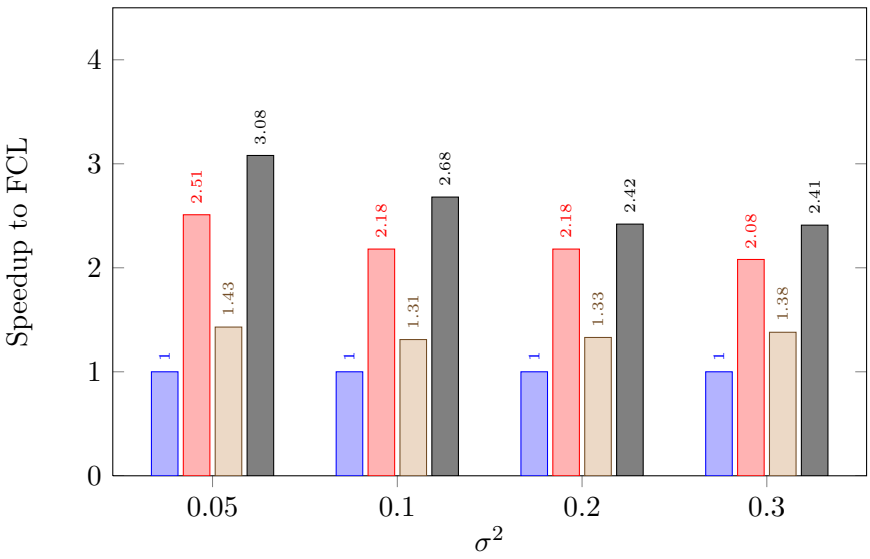


(b) Gaussian Sampling

Fig. 26: The valid and invalid portion for different values of σ^2.



(a) Bridge Test



(b) Gaussian Sampling

Fig. 27: The FARR performance in the motion planning scenario.

sampling strategies. We see that our approach can accelerate the computation of the samples by a factor of 2 to 3 depending on the chosen σ^2. With this result, we showed that our approach is not only practical for tree-based motion planner but is also able to accelerate the collision detection in the sampling of roadmap-based methods.

4.3.5 Discussion

In this section, we will discuss our approach in a wider field of applications. For motion planning, collision detection algorithms are used for the sampling of valid configurations but also for the edge verification. There, one discretizes the edge and uses a collision detection algorithm for the verification of the resulting configurations. The question arises whether the FARR approach can also be used here. We have shown that our approach is dependent on the invalid portion of the tested configurations. The edges in a motion planner are built up in C_{free}. This means that a high portion of the tested configurations is valid, which in consequence causes a lot of overhead for the FARR approach. We tested this scenario using an RRT approach and observed a performance decrease. Therefore, we do no recommend to apply our approach for the edge verification of a motion planner. In general, we do not recommend using our approach in the motion planning for environments that are not cluttered with obstacles. In these scenarios, there are few invalid configurations that need to be tested by the motion planner, and therefore the overhead is too high.
For applications other than motion planning, the performance of our approach mainly depends on the invalid portion of the configurations that are tested. But it also depends on the number of triangles of the models. A high number of triangles favors our approach in comparison to a state-of-the-art BVH approach, especially in scenarios that need a lot of collision queries. With a higher number of triangles, the depth of the BVH increases and herewith the time to traverse the data structure in the collision query. For the FARR approach, with a higher number of triangles, the pre-computation time for the distance field increases. But in the collision query, the paths of the random rays are basically the same. So in conclusion, to decide whether one should use the FARR approach on their application depends on the scenario and the input data. For the disassembly planning with narrow passages, we showed that the FARR approach can achieve a substantial speedup.

5

INVALID INITIAL STATE DISASSEMBLY MOTION PLANNER

The main algorithmic idea that we present in the course of this chapter is based on our publication "A motion planning algorithm for the invalid initial state disassembly problem". It was presented at the 20th International Conference on Methods and Models in Automation and Robotics in 2015 (Schneider *et al.*, 2015a). In this chapter, we present the approach in detail. First, we present our methodology for the problem. Second, we give an extended insight into the background and application of our approach. At last and foremost, we present the final version of our approach that we highly improved regarding performance compared to the published results.

5.1 Study of the InIState Problem

In this chapter, we put our focus on disassembly planning in the DMU process. In particular, we put the focus on the task where the engineers have to validate that given components of a vehicle can be disassembled from their installed position. They are supported by software tools like a three-dimensional rigid body motion planning software. In cooperation with Schneider *et al.* (2013), we extracted and analyzed the main challenge the engineers are facing in their work on the disassembly planning. They are facing the challenge that the objects are not free of collision in their installed position, although state-of-the-art motion planning algorithms have the requirement of a valid, free of collision, initial configuration. We call this problem the Invalid Initial State Disassembly Motion Planning Problem (short: InIState problem), which will be defined in a more formal way in the course of this chapter. State-of-the-art motion planning algorithms cannot be applied to scenarios that have this problem and the

engineers are left alone with a lot of manual tasks. In the following, we describe our methodology to face this problem.
To get a comprehensive understanding of the problem we initiated a study (Schneider *et al.*, 2013). In this study, we investigated motion planning scenarios that have to be checked during the DMU process of a vehicle. In cooperation with engineers from the Daimler AG, we analyzed over 800 motion planning scenarios. In each scenario, we investigated the disassembly path with a CAD software and an integrated state-of-the-art motion planner. We documented various aspects regarding the disassembly process. One aspect that we documented is the number of InIState problems in the investigated motion planning scenarios as well as the reasons for the InIState problem. Moreover, we documented the individual characteristics of the problem regarding the motion planning algorithm as well as the characteristics of the disassembly paths that the engineer himself has to define manually.
In the further course of the work and based on the initiated study of the problem we define five representative benchmark scenarios. This benchmark dataset is used for evaluation of different approaches to this problem. Therefore, we will analyze the current algorithmic approaches to the problem and extract the existing scientific gap in the literature. To close this gap, we propose a new approach that we developed. We describe, analyze and benchmark this approach in the course of this chapter in detail.

5.1.1 The DMU Process

The flowchart of the DMU process where our study took place is shown in Figure 28. This process is not automated. The work units that have to be done manually by an engineer are highlighted in yellow. The process starts by loading the current scenario from the PDM system into the workbench of the software. Then the scenario is tested whether the initial state is valid or not. This is done by a collision detection algorithm. In most cases, the initial state is not valid, and the engineer has to manually solve the InIState problem. With the support from the visualization of the motion planning software, the engineer navigates the dynamic object to an intermediate valid configuration c_{inter} that is as near as possible to the installed position. In the next step, he manually defines a path σ_1 with successive translations and rotations from the initial to the previously defined valid configuration c_{inter}. With this manually defined path, the InIState problem is solved. In the next steps, a state-of-the-art motion planner is

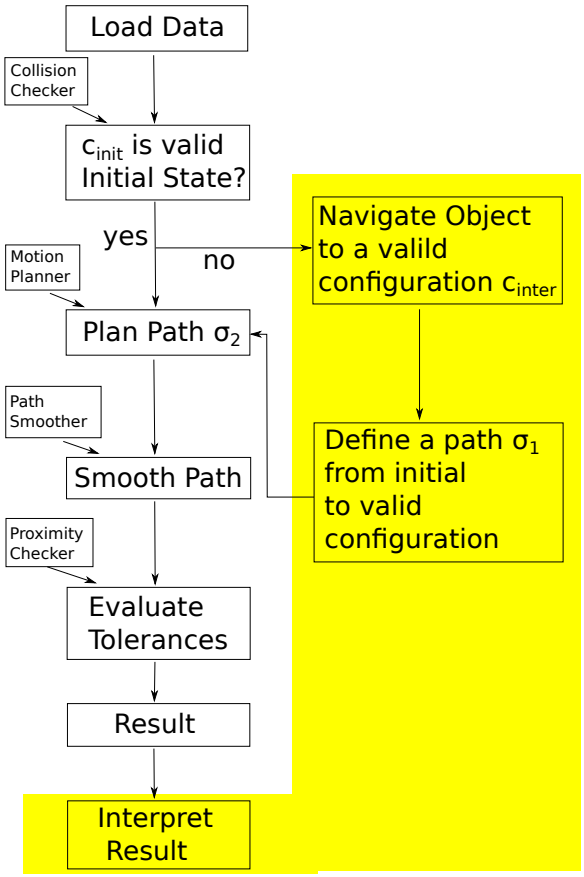


Fig. 28: Disassembly Planning in the DMU Process.

executed and plans a path σ_2 from the valid configuration c_{inter} to a configuration that represents the disassembled state. In the next two steps, the path is post-processed. It is smoothed, and tolerance values for the path are computed. This means that the distances of the object to its environment along the path are evaluated. At last the results are interpreted by the engineer. He evaluates whether the component can be disassembled or not. These tasks are repeated for each disassembly scenario of the vehicle.

5.1.2 Results of the Study

In the following, we will describe the insights about the InIState problem we gained from studying the DMU process and derive a formal definition of the InIState problem.

Amount of Invalid Initial States

The first insight we made is about the amount of invalid initial states. The invalid initial states are the cause for the InIState problem. The initial state of the dynamic object is the assembled state of the scenario. If the geometry of the dynamic and static object intersect in the assembled state, state-of-the-art motion planners are not able to compute motion paths. In our study, we found out that every scenario we investigated presented an invalid initial state. The only scenarios where the dynamic object and its environment were not in collision were those where the environment or the object itself were missing due to the lack of data. We investigated scenarios from the early development stage. Therefore about 10% of the scenarios had some missing data.

Reasons for the Invalid Initial States

The primary purpose of the DMU process is to verify a product in the introduction stage of the product lifecycle. In this stage, meta information for the object are incomplete or even missing. For example

- the exact surface of the object,
- the material properties of the object,
- the differentiation of rigid and deformable parts
- and even the kinematics of the objects.

For several reasons, these missing information leads to invalid initial states. First, in the interior of a vehicle, the majority of the components are fixed to their environment with fastening elements like clips. The suppliers of such fastening elements offer the CAD models of the clips in a rigid version. This rigid version shows the clips in their relaxed state. These relaxed fastening elements cause invalid initial states. Other invalid initial states are caused by bolts and screws. Those fastening elements are used to fix components of the vehicle that are not visible to the customer or relevant for the stability of the vehicle. In CAD systems the threads of the screws are rarely modeled in detail. Often they are represented as colliding over-sized tubes. These tubes, together with the bolts, cause invalid initial states. Another reason for the invalid initial states are the surfaces of the CAD models. In CAD data, the surfaces

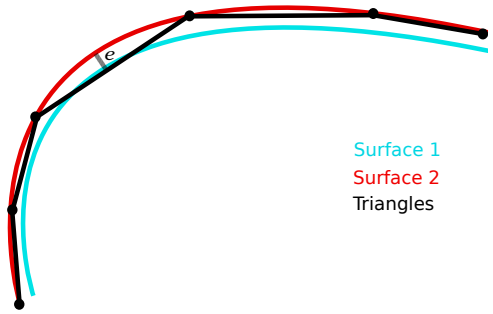


Fig. 29: Surface discretization and invalid initial state.

are modeled with Bezier and/or NURBS surface patches. For an efficient performance in the DMU process, these surfaces are discretized to triangle meshes with a certain precision e. A common approach is to define e as the maximum distance of the surface to its approximated triangle mesh. This precision is coarse in order to get meshes that have a small number of triangles. In Figure 29, we see how the precision is defined for Surface 2. But such a low precision can cause an intersection with other components, consider Surface 1 in Figure 29. But even when neglecting the discretization, some components are fixed to their environment by friction-based connections. In those cases even the exact surfaces intersect. In Figure 30, we show the distribution of the

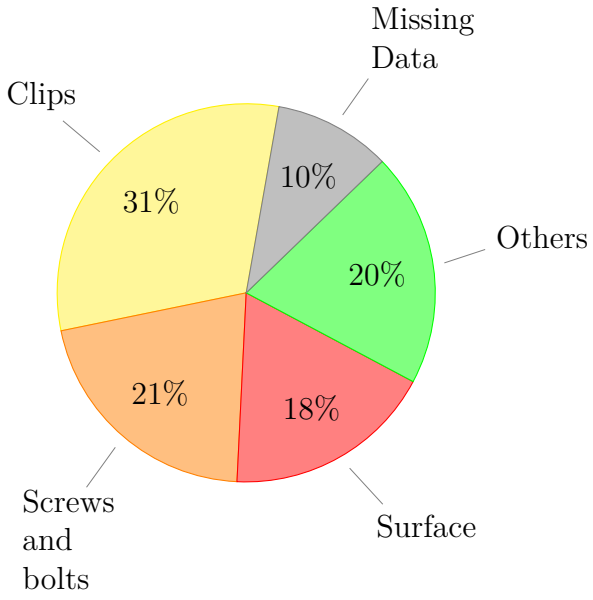


Fig. 30: The distribution of the causes for the invalid initial states.

above-mentioned reasons for the invalid initial states within our 800 motion planning scenarios. With 31%, the largest portion of the invalid initial states is caused by clips.

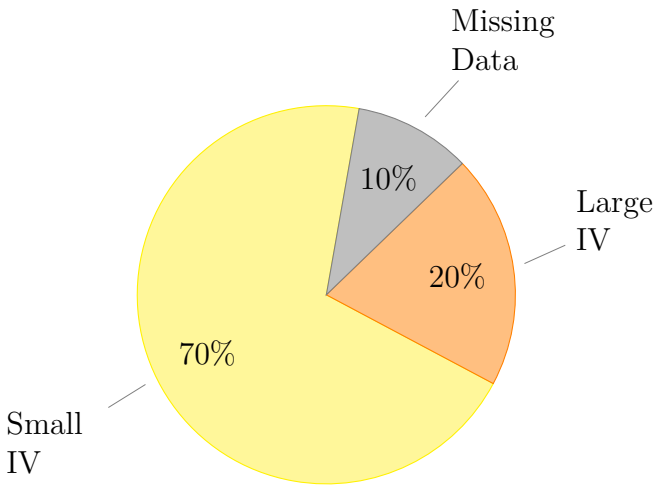


Fig. 31: The characteristic of the invalid state.

The second largest group with 21% is the group of screws and bolts. The invalid states caused by surface discretization and friction-based connections have an amount of 18%. With a sum of 70%, we see that the main contributors to the invalid initial states are fastening elements. In the last group with 20%, there are invalid states that are not related to the fastening of the components but to the fact that we considered a three-dimensional model under development. An example is cable harnesses that are not correctly modeled in the CAD data or dummy objects that are added by the engineers to allocate construction space for future components.

Characteristics of the Invalid Initial States

Next, we want to address the characteristics of the invalid initial states near the installed position. The primary characteristic we evaluated is the size of the intersection volume (IV) that causes the invalid initial state. As we see in Figure 31, 70% of all invalid initial states have a small intersection volume. With small, we mean that the intersection volume is smaller than the volume of a screw. So most of the invalid initial states are not immediately apparent, and one needs the help of a collision detection algorithm to detect the collision. Although small, regarding the intersection volume, they prevent state-of-the-art motion planning algorithms to plan a path.

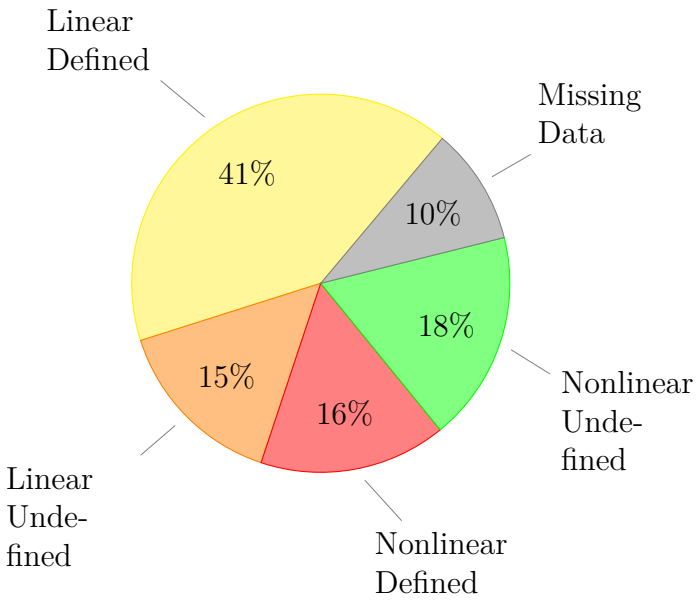


Fig. 32: Characteristics of the disassembly paths.

Invalid Initial State Disassembly Motion Planning Problem

From the results above we can conclude that planning the disassembly in real world scenarios leads to a new important type of disassembly motion planning problem, namely the Invalid Initial State Disassembly Motion Planning Problem (short: InIState problem). It can be defined as follows:

Definition 4 (***Invalid Initial State Disassembly Motion Planning Problem***)

Given $c_{init} \in C \subset SE(3)$, the initial configuration of D, and $C_{goal} \subset C_{free} \subset SE(3)$, the set of valid goal configurations of D that represents the disassembled states. The solution path $\tilde{\sigma} : [0,1] \rightarrow C$ with $\tilde{\sigma}(0) = c_{init}$ and $\tilde{\sigma}(1) = c_{goal} \in C_{goal}$ is the disassembly path for D if its exact surface, kinematics and deformations including material properties are taken into account. The Invalid Initial State Disassembly Motion Planning Problem is to find an realistic approximation σ of $\tilde{\sigma}$ in the absence of this information.

In reality, the dynamic object D would deform during the movement, leading to a disassembly path $\tilde{\sigma}$ that is free of intersections. The primary challenge of the InIState problem is that in the CAD data the dynamic object D is rigid and its exact surface, kinematics, and deformations are not available for planning. The only data that are

available at this early stage of the DMU process are the triangle meshes. Therefore methods and algorithms are needed that approximate the real path $\tilde{\sigma}$. The main difference to the disassembly motion planning problem is that in consequence the path σ we are looking for, including the initial configuration c_{init}, will consist of configurations of C_{free} and C_{obs}. That means that σ will cause intersections of the rigid object D with its environment.

Characteristics of the Disassembly Paths

At last, we consider the disassembly paths for the invalid initial states. We group the disassembly paths in linear and nonlinear disassembly paths. A linear disassembly path is a path where the object can be disassembled from its near environment by moving it along a straight line. A nonlinear path mostly consists of a complex motion including translations and rotations. As a second criterion, we consider whether the path can be derived from the fastening elements or not. If a path can be derived from a fastening element, the engineer can use the orientations of the fastening element to define the disassembly path. In Figure 32, we see the distribution of the categories. For the engineer, the easiest paths to define manually are the linear paths that can be derived by a fastening element. This category is represented with 41%. The most time-consuming paths to define manually are the two types of nonlinear paths and the linear paths that are not defined by any fastening element. These last three categories together represent 49% of the scenarios, showing the importance to design algorithms for the InIState problem to support the engineers in their task to plan and define paths in the DMU process.

Context of the InIState Problem

In this section, we consider the context of the InIState Problem in the field of motion planning. In the literature, the InIState problem is only addressed by Jianwei Guoa *et al.* (2013). But their work is limited to linear disassembly paths and puts a focus on the illustration of the disassembly and not primarily on the computation a path.
But in a wider context, one can consider the InIState motion planning problem as a motion planning problem with uncertainty (Dadkhah and Mettler, 2012). These problems have their origin in the motion planning for robots. There, one cannot ensure the exact configuration of the robot as each configuration can only be approximated.

In our case, one could consider the evaluation of a configuration as uncertain too. The classification of a configuration to the forbidden configuration space is uncertain, as the colliding parts of the geometry could be deformable and therefore the configuration could be valid.
In an even wider context, one could consider the InIState motion planning problem as a problem of constraint motion planning (Pivtoraiko and Kelly, 2006). In constraint motion planning, one considers motion planning problems with constraints on the path. Examples for a constraint are: The object

- is only allowed to move along paths with a low curvature,
- has to keep a minimum proximity to its environment or
- that the length of the path must not exceed a certain length.

For the InIState motion planning problem, one can define a vague constraint. One can define the constraint that the motion planner has to take into account the unknown properties of the dynamic object, like the deformable parts. The basic approach of constraint motion planning is to consider the constraint in the computation of the edges of the data structure.

5.2 Dataset

Table 14: Attributes of the datasets.

Dataset	#1	#2	#3	#4	#5
Triangles D	67,134	48,308	6,544	2,448	26,290
Triangles S	43,918	26,364	26,430	4,250	16,810
Disassembly Motion	nonlinear	nonlinear	nonlinear	nonlinear	linear

For the support of our work, Hofmann (2015) designed and constructed datasets for the InIState problem. The five datasets represent different challenges for the motion planning algorithms caused by clips. The datasets are given as triangle sets. They are modeled with respect to different scenarios we faced in our study. The datasets are shown in Figure 33 and the properties of the datasets are listed in Table 14. We have analyzed that the main contributors to the InIState problem are fastening elements. Therefore the datasets we present are representative for the InIState problem caused

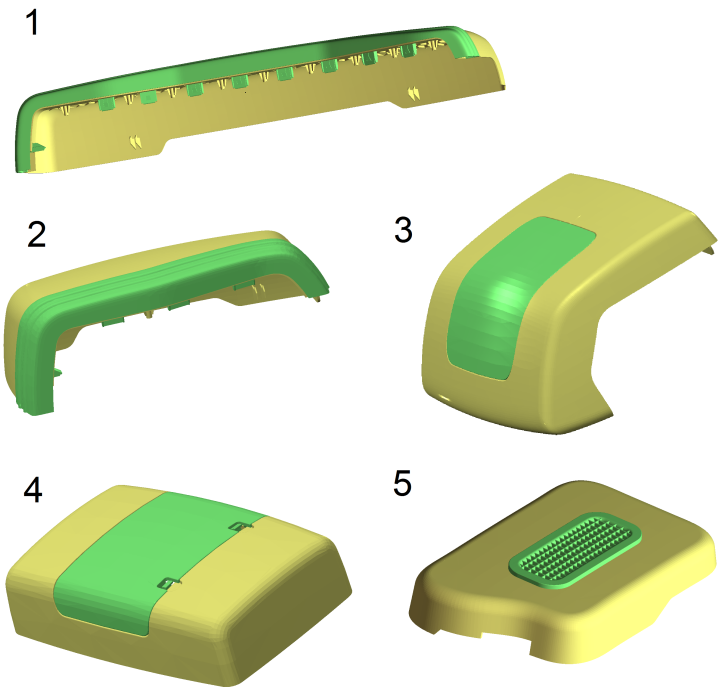


Fig. 33: Dataset 1-5 for the InIState problem.

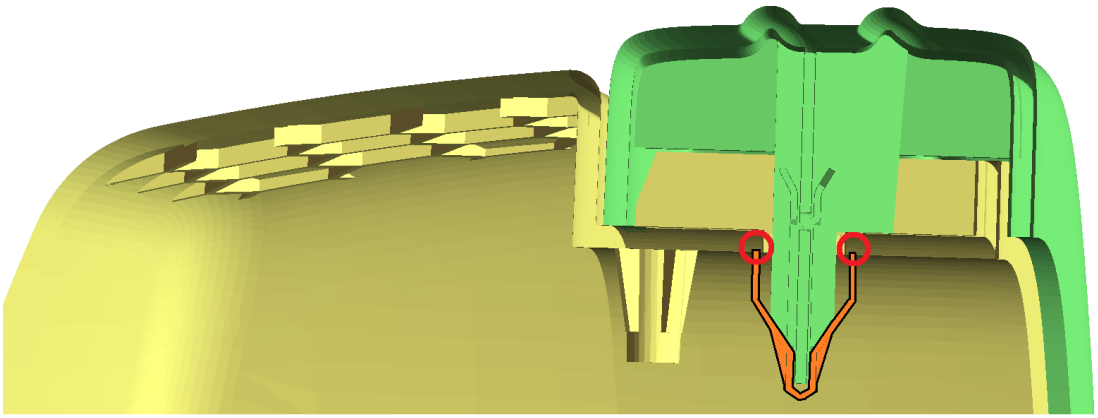


Fig. 34: Close-up view on dataset 2. Orange: One clip that causes intersections along the disassembly path, Red circles: Triangles that are in collision at the initial state.

by clips fastening elements. In Figure 34, we see a close-up view on dataset 2 that shows the intersection of the dynamic object and its environment. The first four datasets of Figure 33 have disassembly paths that are nonlinear. Dataset 5 has a linear disassembly path. In those five datasets various types of fastening elements are represented. In Figure 35, we show details of the datasets 2 to 5. Dataset 1 is very similar to dataset 2.

First, we see that dataset 2 has three flexible clips and a rigid guide rail on the

left. The outer parts of the clips are intersecting the environment at the initial state. Moreover, the clips will cause intersections all along the first part of the disassembly path. The guide rail has no intersection with the environment but determines the rotational disassembly motion we see in Figure 2. Dataset 1 from Figure 33 is a variation of dataset 2. Only the number of clips is increased from 3 to 8. In dataset 2 the intersection volume caused by the three clips is very small. A naive approach by simply allowing the motion planner a small threshold of intersection volume succeeds for dataset 2. In contrast, for dataset 1 the intersection volume of the 8 clips is larger than the volume of the guide rail. The naive approach fails in dataset 1 as it ignores the guide rail.

Dataset 3 has six clips and two guide rails. In this dataset, the clips are large. As we see in the first and second detailed view, the clips are even larger than the guide rails. This makes motion planning challenging for this dataset.

Dataset 4 is an example of friction-based tension. Although the object has two clips, the component is mainly fixed by the friction of the two guide rails. Therefore the surface of the dynamic object and its environment are intersecting.

Dataset 5 is a representative example for a linear disassembly path. As we see in the detailed view, the clips of dataset 5 are thick and almost rigid in comparison to the clips of the other datasets.

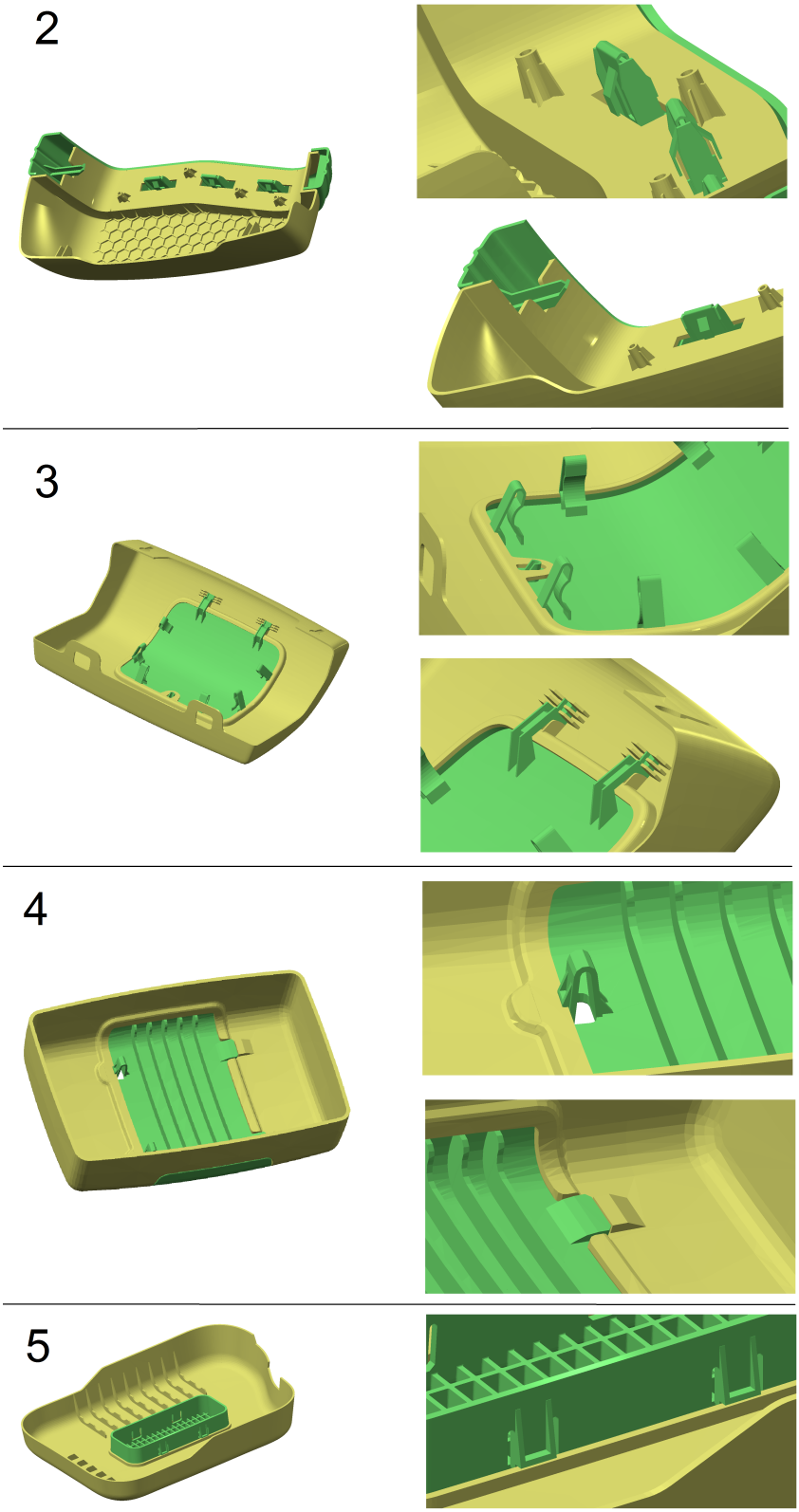


Fig. 35: Details of datasets 2-5. Left: Bottom View. Right: Detailed View.

5.3 State-of-the-Art Approaches

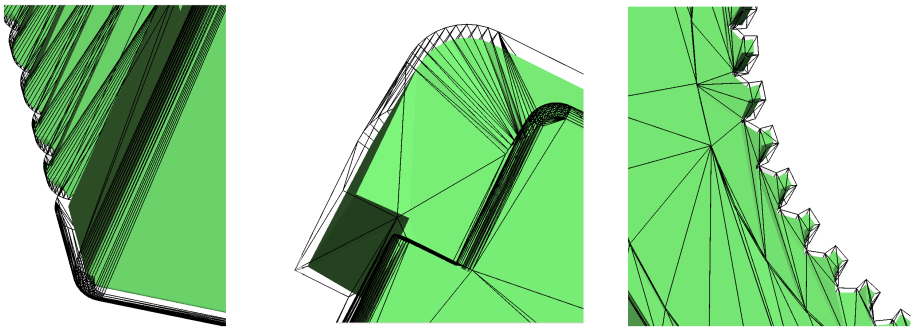


Fig. 36: Examples of shrinking an object.

In this section, we deal with state-of-the-art approaches to solve the InIState problem. The approaches to this problem are rather basic. The methods to deal with the InIState problem are all placed around the idea of softening the definition of collision. One well-known approach is to shrink the object. An example for the shrinking of an object is shown in Figure 36. A shrinking algorithm can be found in (Ma and Wan, 2001). The algorithm computes the shrunken object based on a voxelization. Another straightforward idea to solve the InIState problem is to replace the collision detection algorithm by an intersection volume or penetration depth algorithm. So a configuration is considered to be valid if the intersection volume or the penetration depth is below a certain threshold. In our experience, these two simple approaches can enable a motion planning algorithm to work for the InIState problem if the discretization of the CAD data to triangle sets or a friction-based connection cause the invalid initial state (Category "Surface" from Figure 30). For the category "Screws and Bolts" a naive approach is to detect cylinders in the components, for example by fitting algorithms like in (Beder and Förstner, 2006). For all categories, including the remaining large category of "Clips" fastening elements, we present a solution in the following.

5.4 Our Approach

As described earlier, the only input for the motion planning in the DMU process is the triangle meshes of the dynamic object and their environment. To solve the InIState problem, we developed a new approach, the InIState motion planner. This algorithm is the topic of this section and will be described and benchmarked in detail. The main

idea of our approach is to give the motion planner the knowledge about deformable regions and teach the algorithm to use this information properly during the planning. To achieve this, we enhance a state-of-the-art motion planner with a special validity test and a pre-processing step. The pre-processing step is the computation of a mesh labeling. We label flexible fastening elements of the dynamic object D by using a deformation model. The deformation model is arbitrary, but we will describe a deformation model that is tuned towards our needs. This information of the pre-processing is integrated to the motion planner. Also, we apply a special intersection volume test during the motion planning. The individual ingredients of our approach are described in the following subsections.

5.4.1 Labeling

Mesh labeling and the related mesh segmentation are widely used in applications from computer graphics, animation, and games but also in the field of industrial applications. A survey of methods for mesh segmentation can be found in (Shamir, 2008) and (Attene *et al.*, 2006). In (Attene *et al.*, 2006), Attene *et al.* categorize mesh segmentation into two groups:

1. Purely geometric segmentation and
2. semantics-oriented segmentation.

The labeling in this work falls into the second category. The semantic we address is the property of being more or less deformable than other regions of the mesh. The labeling measures how geometrically deformable the object is at a vertex in comparison to all other vertices.

Background and Goals

The straightforward approach for the computation of the labeling is to use a deformation model for the object, apply a unit force on each vertex of the object and measure the displacement of the vertex. Large displacements label high deformable regions and *vice versa* for small displacements.
Well-known deformation models that could be used for the computation of the labeling are based on the finite element method and a tetrahedral mesh of the object. This

usage has two major requirements. First, one needs a tetrahedral mesh. In our application the objects are represented by triangle meshes. To automatically compute the tetrahedral mesh, the triangle mesh has to be watertight and the computed tetrahedra must be non-degenerate. Second, one needs a stable implementation of a deformation model based on the finite element method.
In our work, we focus on an alternative approach for the computation of the labeling with the following goals:

1. Usage of a deformation model based on the surface in order to avoid the need of a tetrahedral or voxel mesh.
2. No need for strictly watertight meshes.
3. Similar results as in a finite element approach. Physically plausible but not necessarily physically correct behavior.
4. High performance and high stability of the deformation model.
5. Simple implementation for an easy addition to existing motion planners.

Several reasons motivate these goals, in particular for the application on deformable disassembly motion planning. In this application, the objects are often rather thin and a volumetric representation is challenging. Moreover, the triangle meshes are not necessarily a 2-manifold. This means that the meshes are not watertight, triangles may intersect or overlap, and even holes can occur. In the deformable disassembly planning, one is not primarily interested in the exact stresses. This is the topic of a modal analysis. One is more interested in a fast and stable computation of deformations to enable the motion planning algorithm to compute a path. At last, deformation models are a broad and complex field, and simple approaches facilitate the introduction to deformable motion planning for a user.
To achieve these goals, we present an approach that is based on the position-based dynamics framework (Müller *et al.*, 2007), a method from the field of computer graphics and animation. This method trades physical correctness for the following benefits: The deformation model is very stable and performs in realtime. Also, the method is very easy to implement and can be added easily to existing motion planners which fit our defined goals. We show in this work how to apply the position-based dynamics approach to the deformation labeling of an object using the surface representation solely, study the number of constraints for the model, compare the results to the ones achieved with a finite element method and evaluate the performance.

To use the surface for the deformation model, we project volumetric properties of the mesh onto its surface. There are many approaches for achieving this. One approach that assigns volumetric properties to mesh vertices is described by Shapira *et al.* (2008). In their work, they use the shape diameter function to compute a skeletonization and a segmentation of a mesh. The shape diameter function measures the diameter of an object at a given vertex. As described in Shapira *et al.* (2008), the distance from the vertex to the medial axis can be assigned to each vertex. This method is related to our method as we use an approximation of the medial axis. Another work from Amenta *et al.* focuses on an approximation of the medial axis (Amenta *et al.*, 2001). In our work, we use an alternative algorithm from Weller and Zachmann (2010). This algorithm is intended to build up an inner sphere tree which is a data structure for proximity queries. As a side effect, the first iteration of the algorithm computes an approximation of the medial axis. This algorithm is easy to implement, fast, and achieves sufficiently good results for our application. In particular, this method can deal with small errors in the mesh, like small holes and overlapping triangles.

Algorithm and Implementation

In this section, we describe how we apply and modify the PBD framework to solve the labeling problem. There are a few basic assumptions for our approach. First, we need a mesh because the mesh defines our constraints. The mesh does not need to be completely watertight. Small holes are allowed and only lead to some missing constraints which are not critical. Second, we assume that the vertex normals of the mesh are correct, which means they direct to the outside of the object. These two assumptions are rather weak. A third assumption comes from the incompleteness of the data in real world application where physical properties are often missing. There, the data does not always store the physical properties. So we assume that the object consists of one single material.

Using only the surface for modeling three-dimensional deformation is highly critical as the vertices have no information about the volume of the mesh, e.g. the thickness of the object at a certain vertex. We cannot assume that a straightforward usage of a surface model alone leads to the same results as a volumetric model. But we want to use solely the surface model due to the small number of constraints and the small storage requirements. So we try to map the volumetric information onto the mesh surface and use this as a parameter in the PBD algorithm to approximate the three-dimensional deformations sufficiently good. How we compute the mapping is shown

Algorithm 9: Phase I

Input: Mesh D with n vertices

Output: $\{v_i\} \rightarrow \{S_i\}$, $i \in [0, n-1]$

foreach $v_i \in D$ **do** 1

  $p_i = v_i - \delta \cdot \text{vertexNormal}(v_i)$ 2

  while *(!*converged(p_i)*)* **do** 3

    $p_{near} = \text{nearestPoint}(p_i, D)$ 4

    $p_i = p_i + \epsilon \cdot (p_i - p_{near})$ 5

  $v_i \rightarrow S_i(p_i, \|p_{near} - p_i\|)$ 6

Algorithm 10: Phase II

Input: D, $\{v_i\} \rightarrow \{S_i\}$, $i \in [0, n-1]$

Output: $\{v_i\} \rightarrow \{d_i\}$, $i \in [0, n-1]$

foreach $v_i \in D$ **do** 1

  $m_i = \rho \cdot \text{volume}(v_i)$ 2

$C = \emptyset$ 3

$C \leftarrow C \cup \text{positionsConstraints}(D)$ 4

$C \leftarrow C \cup \text{bendingConstraints}(D)$ 5

initPBD(solverIterations N, Damping da_l, Stiffness k, Δt, m_i, D, C) 6

foreach $v_i \in D$ **do** 7

  resetPBD() 8

  $f_{ext_i} = -\sigma \cdot \text{vertexNormale}(v_i)$ 9

  executePBD(t) 10

  update$(\{d_0 ... d_{n-1}\})$ 11

in Algorithm 9 which is similar to the algorithm for the inner sphere tree from (Weller and Zachmann, 2010). The input of the algorithm is a mesh D with n vertices. The main idea of this algorithm is to compute a mapping $\{v_i\} \rightarrow \{S_i\}$ that assigns a sphere S_i to each vertex v_i of D. The center of the sphere S_i is near the vertex v_i and centered approximately on the medial axis. To compute this mapping, we go over all vertices of the mesh. For each vertex v_i we need to compute the center p_i of S_i. First, we move v_i a small distance δ to the inside of the mesh with the usage of its negative vertex normal. The resulting point p_i is used as the starting point for the iterative process in line 3. In each iteration, we compute the point p_{near} that lies on the mesh and has the smallest distance to p_i. Now the point p_i is pushed to the opposite direction of p_{near}. The ϵ is a cooling function which, in our case, decreases linearly with every iteration. With this process, the point p_i approximates a point on the medial axis of the mesh (see (Weller and Zachmann, 2010)). The resulting sphere S_i is defined by the center p_i and the distance from p_i to p_{near}. This sphere S_i is stored for the vertex v_i. So for each vertex v_i the volumetric information is a sphere S_i approximately centered on the medial axis. S_i is near the vertex v_i because we choose the point p_i near the vertex v_i as starting point. The sphere S_i gives us an approximation of the diameter of the object at the vertex v_i.

Algorithm 10 shows the computation of the actual labeling with the PBD algorithm. For our labels we compute a mapping $\{v_i\} \rightarrow \{d_i\}$ that assigns a deformation value d_i to each vertex v_i. For each vertex v_i we have to define its mass. We approximate the vertex mass as described in the following. For each vertex v_i we define a volume. The base of the volume is a third of the area of all triangles that have v_i as vertex. The height of the volume is the diameter of the medial axis sphere S_i. This volume together with the density ρ define the masses m_i of v_i. Note that the mass of a vertex takes into account the thickness of the object at v_i as well as the triangulation of the mesh. Afterward, we compute the constraints. In Algorithm 10 a position constraint is computed for each edges of the mesh. Position constraints are applied to two vertices. E.g. the two vertices of an edge in the mesh. If the edge is stretched, the vertices are pushed towards each other by the constraint. If the edge is compressed, the vertices are pushed away by the constraint. In Algorithm 10 we also apply a bending constraint to every two triangles that share an edge. This is realized by applying a position constraint to the two vertices of the neighboring triangles that are not part of the shared edge. After the initialization of the parameters, the computation of the labels is executed for each vertex v_i on the surface. For each v_i we apply an external force in the opposite direction to the vertex normal and simulate a given time t with

the PBD algorithm. The size of the external force is determined by a parameter σ. The value σ is a scaling factor that is applied to the direction of the force. Afterward, we measure the displacements $\{d_0...d_{n-1}\}_i$ for all vertices in iteration i. The displacements $\{d_0...d_{n-1}\}_i$ are measured in Euclidean distance and summed over all iterations. So for each vertex v_i the value d_i measures the displacements over all iterations. This value d_i is used as a measure for the deformation. A high value means that the vertex can be deformed easily and a low value means that the vertex is nearly rigid. The overall computation time is dominated by the second phase. More precisely, it is dominated by the number of constraints that are used in the PBD algorithm. To keep the runtime low, we limit our approach to the position constraints and a single bending constraint for each pair of neighboring triangles. The bending constraint is realized by applying a position constraint to the opposing vertices of the neighboring triangles. For a mesh with E edges, we only need $2 \cdot E$ position constraints. We will prove in the experiments that this small number of constraints provides a sufficiently good labeling of the vertices and a high performance.

Next, we give some details about the implementation parameters. We scaled all datasets to an axis-aligned box with the largest extension of 1.0. For the computation of the deformation values, we simulated 1 second with the time interval $\Delta t = 30$ milliseconds. We used linear damping with a value of $da_l = 0.99$. For the number of iterations, we were able to choose a low value of $N = 3$ and still get plausible deformation values. For the stiffness, we used the formula from Müller *et al.* (2007) with $k' = 1 - (1 - k)^N$ and $k = 0.99$. The one remaining parameter is the value σ for the external force. This parameter is the same for all vertices and very important. If σ is too large, any vertex can be moved. Otherwise, if σ is too small, none of the vertices moves. To determine σ automatically, we chose a heuristic that is based on the local diameters of our object. We chose σ related to the smallest diameter of the object. This ensures that we can detect small elements of the object. To measure the diameter in each vertex, we used the results from the first phase where we computed the approximate medial axis spheres. We took the minimum radius r_{min} of all those spheres to determine σ by $\sigma = \kappa \cdot r_{min}$. We chose $\kappa = 5.25$ as this value showed good results in our tests. These parameters were used to obtain the results that are presented in the experiments chapter.

5.4.2 Maximum Local Intersection Volume

Besides the labeling, we need another ingredient for our algorithm. As noted before, an RRT can use the intersection volume for the *getValidEdge* function (see Algorithm 1) to compute paths for the invalid initial state problem. This approach is simple but too simple and fails in many benchmarks. In our algorithm, the motion planner uses a method that we call the maximum local intersection volume (MLIV). In contrast to a simple intersection volume an MLIV is defined as:

Definition 5 (*The (Maximum Local) Intersection Volume*)

Given two polyhedra D and S, the intersection volume is given as IV(D,S):= $D \bigcap S$. If $IV(D,S) \neq \emptyset$ partitions into n disjoint volumes $IV(D,S) = \biguplus_{i=1}^{n} V_i$, we define the maximum local intersection as $MLIV(D,S) := max\{V_1, V_2, ..., V_n\}$.

Thus, only the largest connected component of the intersection volume contributes to the MLIV. It is evident that the computation of the MLIV is more complex than an intersection volume computation. But our work shows that the MLIV achieves far better results because collisions are considered independently. For each configuration the MLIV is different. How we calculate the MLIV is described in the following:

Algorithm 11: Compute MLIV(c)

Input: $c \in C$-Space, $V(S)$, $V(D)$

Output: MLIV(c)

```
foreach s_j ∈ V(D) do                                   1
    if (!alreadyVisited(s_j)) then                      2
        if s_j ∈ V(S) then                              3
            CC_j ← startFloodFill(s_j)                  4
            setVisitedVoxels()                          5
n ← size(max(CC_j))                                     6
MLIV(c) ← voxelSize · n                                 7
```

To compute the MLIV for a configuration c we extend the VPS (Voxmap-PointShell Algorithm^{TM} (McNeely *et al.*, 2005)). Beside a voxelization of S, like in the VPS, we also use a voxelization of D. In order to compute the MLIV we test every voxel center s_j of $V(D)$ whether it is inside $V(S)$ or not. Note that we only have to test s_j if the

corresponding voxel intersects the surface of D. Only the inside test of s_j is similar to the VPS. If s_j is inside $V(S)$, we start a flood fill algorithm (see (Torbert, 2016)) with this voxel. During the flood fill algorithm, we check every neighbor of the currently visited voxel of D whether it is also covered by any voxel of S. Only if this holds true, the neighbor is added to the flood fill algorithm. At the end of the flood fill algorithm, we computed the first connected component CC_j. To avoid computing a connected component CC_j multiple times, we have to flag all voxels that are part of CC_j as visited. We repeat this process for all surface points s_j. Afterward, we determine the largest connected component which represents the MLIV. Due to the voxelization, the computation is only an approximation of the MLIV. Nevertheless, this approximation is sufficient for our application and shows a good performance.

5.4.3 Validity Computation

After precomputing the labeling, we now combine it with the MLIV from Algorithm 11. The labeling is computed by vertex and the MLIV is computed by voxel. Therefore, as the vertices have a coarser resolution than the voxels, we need to assign a labeling value for each voxel. This done by interpolating the labeling in the same resolution as the voxel field for the MLIV computation. With this, we assigned a color value for each voxel. Our goal now for the validity computation on the motion planner is that voxels with a low value in the labeling contribute more to the MLIV than voxels with a high value in the labeling. Therefore, the intersection volume of deformable parts is weighted lower than the intersection volume of rigid parts. For our planner, we use a linear relation between the value of the MLIV and the labeling. The computation is shown in Algorithm 12. The only difference to Algorithm 11 is the post-processing step in lines 6-9. There, we measure the volume of the connected component based on the labeling that we computed in the pre-processing. This is done by multiplying the size of a voxel with returned value of the function colorValue($voxel$). This function returns a value between 0 and 1, where 0 means deformable and 1 rigid. With this, deformable parts of the object do not contribute to the MLIV at the configuration c.

Algorithm 12: Compute Labeled-MLIV(c)

Input: $c \in C$-Space, $V(S)$, $V(D)$

Output: Labeled-MLIV(c)

```
foreach s_j ∈ V(D) do                                   1
    if (!alreadyVisited(s_j)) then                      2
        if s_j ∈ V(S) then                              3
            CC_j ← startFloodFill(s_j)                  4
            setVisitedVoxels()                          5
foreach CC_j do                                         6
    cv_j = 0                                            7
    foreach voxel ∈ CC_j do                             8
        cv_j += voxelSize · colorValue(voxel)           9
MLIV(c) ← max{cv_j}                                     10
```

5.4.4 The InIState Motion Planner

For our InIState motion planner, we alter the motion planner of Erbes (2013), which is based on the EST (Hsu *et al.*, 1997). In principle many motion planners can be used, our approach is not limited to the motion planner of Erbes. For example, we can use the CR-RRTC that we presented in Chapter 3 or the PRM (Kavraki *et al.*, 1996). This results in similar paths but a much lower performance than with the approach of Erbes. The reason is explained in the following. Due to the problem statement, we deal with motion planning problems that feature very restricted motions, the so-called narrow passages, near the initial state. Motion planners like the CR-RRTC or the PRM, draw samples from the whole C-Space. In the EST approach, one samples locally in a small sphere around the already computed data structure. In the beginning of the algorithm, the data structure only contains the initial state. This means that with the EST approach, it is more likely to sample the important configurations that lead the algorithm through the narrow passage. This results in higher performance. But the limited sampling radius of the EST has the consequence that it explores the C-Space at a lower speed in comparison to the other two mentioned algorithms. But this is not important for the performance of disassembly motion planning.
The two changes that one needs to apply to a motion planner in order to get our

InIState Motion Planner are described in the following. First, we need to replace the $getValidEdge$ function by the Labeled-MLIV computation of Algorithm 12. To get a validity function, we introduce a threshold τ. If the Labeled-MLIV value, the result from Algorithm 12, is below τ the configuration is considered as valid and invalid otherwise. Note that τ introduces an early out for Algorithm 12 and can speed up the computation substantially. If in the processing of Algorithm 12 the current value cv_j exceeds τ, the configuration is not valid and the computation is terminated.
The value of τ is critical for the path planning. For low values of τ the planner only allows small local intersections of the objects. Therefore only paths with little intersections are calculated, and the opposite holds true for large values of τ. For low values of τ there is possibly no path and for large values of τ the computed path may be not reasonable. There are two methods to set this value.
First, the threshold τ can be set by the user. The value for τ is related to the volume of the fastening elements. The engineer can set the value relative to the volume of the largest fastening element. This heuristic proved well in our benchmark datasets.
Second, one could define τ by a heuristic. We present an algorithm to determine a suitable value for τ automatically. How to achieve this is explained in the next section together with the overall algorithm.

Second, we need to define a goal state c_{goal} for our InIState Motion Planning algorithm. In a disassembly motion planner, the goal state is often defined as a set of configurations C_{goal} (e.g. all configurations that separate the bounding boxes of D and S). These goal states are needed in the $goalReached$ function in Algorithm 1 to determine if the planning algorithm can be terminated. This definition of C_{goal} can also be used for our planner. But in our algorithm, we want to terminate the computation with the Labeled-MLIV computation as validity function earlier to improve performance. Computing the Labeled-MLIV is more expensive than a collision detection. We propose to define $C_{goal} := \{c \mid \text{distance}(S, D) > \psi\}$ with ψ as the maximal extension of a fastening element. The value of ψ can be estimated by the labeling, e.g. by computing the largest connected component of deformable voxels in the labeling. After this computation, we can assume that the fastening elements do not affect the disassembly path. For the remaining path, we can use a state-of-the-art motion planner to plan the path for the separation of the objects. This approach divides the solution path σ into two parts σ_1 and σ_2, computes c_{inter} (see Section 5.1.1), and improves the performance of our overall motion planner.
We call any disassembly motion planner that is altered in the above-described way,

an InIState Motion Planning algorithm. How we use this approach for the overall algorithm in a DMU process is described in the next section.

5.4.5 Overall Algorithm

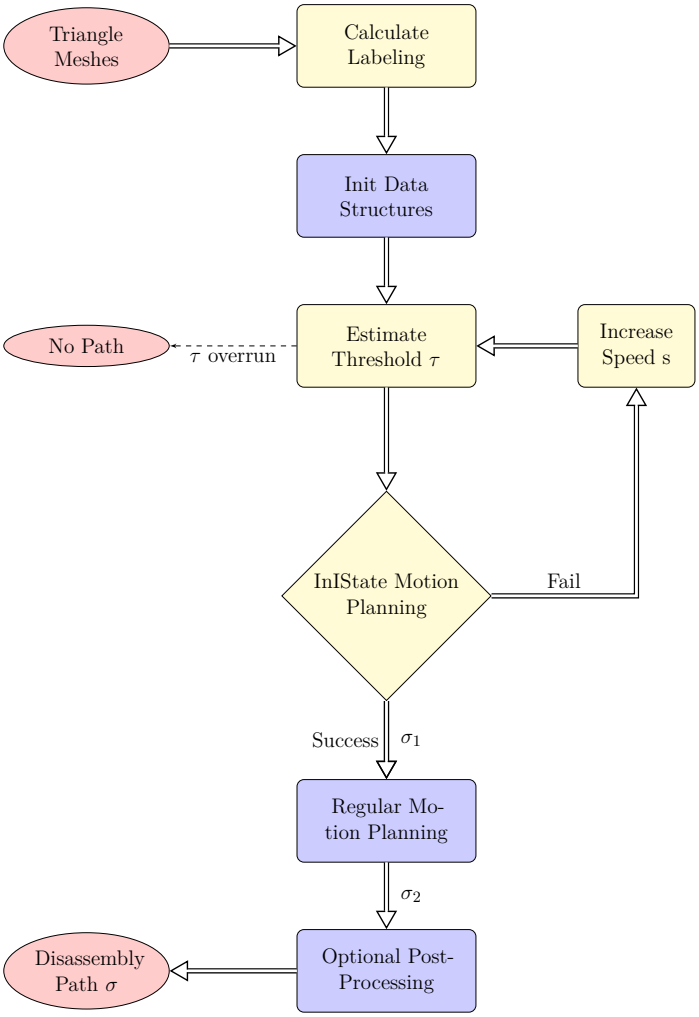


Fig. 37: The flowchart of our algorithm.

The structure of the overall algorithm for the DMU process is summarized in Figure 37. State-of-the-art parts are marked in blue whereas our new algorithmic parts are marked in yellow. The only inputs to our algorithm are the triangles of S and D. The output is the disassembly path σ which consist of invalid and valid configurations.
First, we calculate the labeling. Second, we initialize the motion planner by calculating other data structures (e.g. Bounding Volume Hierarchies). Afterward, the InIstate

Motion Planning loop starts. We want to ensure an automatic process. Therefore a suitable value for τ must be determined. The parameter is highly dependent on the data and must be computed individually for each dataset.
A straight forward approach is to start at the lowest possible value for $\tau := \tau_{lowest}$, namely the Labeled-MLIV value computed for c_{init}. We now increase τ_{lowest} linearly with a fixed step size δ_1 and test every value $\tau' = \tau_{lowest} + k \cdot \delta_1$ whether a path can be found in a given maximal planning time. As a small step size is needed, this results in many tests (> 25) each taking a full planning time. The entire computation time is high with this simple approach. Therefore we want to propose a method that is able to reduce the number of loops significantly. At first glance, a bisection would be a suitable approach. But due to the high performance variance of sampling-based motion planners, the bisection is unstable and could converge to a high value of τ. We propose a method that uses the expansion speed s of the RRT to determine a suitable value for τ. The expansion speed s of a planner is defined by the number of added configurations in T per second planning time (e.g. 4 configurations per second). So we start with an initial speed s_{init} and increase τ_{lowest} until the expansion speed s_{init} is reached. The resulting τ is then used to start the InIState Motion Planning with 2 minutes planning time. In case the InIState Motion Planning fails, the speed is linearly increased by $s' = s_{init} + k \cdot \delta_2$ and a new value for τ is computed. In our algorithm, the initial speed s_{init} is set to 4 configurations per second and δ_2 is set to 5 configuration per second. In the case of success, we calculated the path σ_1 for the disassembly problem. The next steps in the algorithm are the regular path planning to calculate σ_2 and some post-processing steps (e.g. smoothing).

5.5 Evaluation and Experiments

In this section, we will evaluate the approach for the labeling as well as the InIState motion planner. The goal of the evaluation of the labeling is to show that our approach, based on the position-based dynamics approach, is able to distinguish between deformable parts and rigid parts of the input geometry. In the evaluation of the InIState motion planner, our goal is to evaluate whether our approach is able to compute disassembly paths for our benchmarks from Figure 33.

5.5.1 Labeling

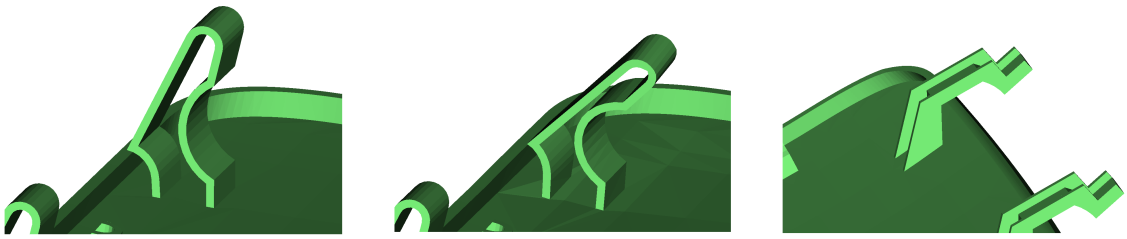


Fig. 38: Zoom-in on important areas of dataset 3.

All dynamic objects of Figure 33 contain some deformable clips. These clips are part of the dynamic mesh and are modeled as rigid objects although they are deformable. On the left-hand side and in the middle of Figure 38 we see one clip of dataset 3 in its relaxed and stressed position. The goal of the labeling is to provide the motion planner with the information that this part of the geometry, the clip, is more deformable than the remaining parts of the mesh. On the right-hand side of Figure 38 we see the guide rail of dataset 3. It is almost rigid due to the cross braces that attach the guide rail to the base plate. The labeling must be able to distinguish between the clips and the guide rails. For the comparison with a state-of-the-art FEM approach, we used a reference model. This reference model is described in the following section.

Reference Model

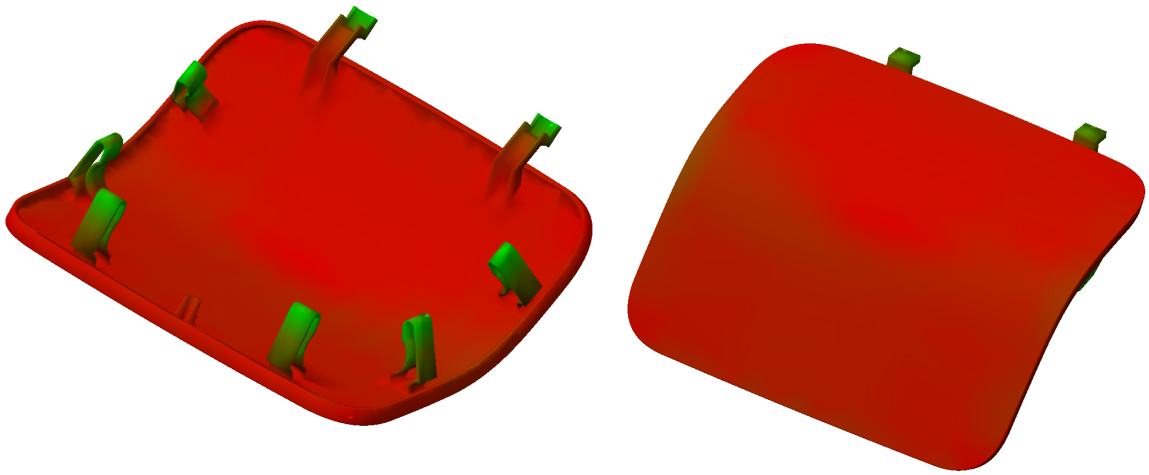


Fig. 39: Labeling of dataset 3 with a corotational linear elasticity model, using the finite element method.

Dataset 3 is used as a reference model. For dataset 3 we computed the labeling with a finite element method. The result is shown in Figure 39. It was computed using the algorithms from the Vega library (Barbič *et al.*, 2012) on a tetrahedral representation of the object with 4531 tetrahedra computed by TetGen (Si, 2015). As deformation model, we used the corotational linear elasticity model. For the material properties, specified by the mass density, Young's modulus and Poisson's ratio, we assumed the values of polypropylene (900 kg/m^3, 1300 MPa, 0.45). For all vertices we applied a force of 500 N in the opposite direction of the vertex normal and measured the displacement. We simulated 10 timesteps with 33ms each. Note that computing the tetrahedral mesh as well as applying the forces includes a lot of manual work due to degenerated tetrahedra, as well as instability in the linear solver (caused by the input data). Moreover, TetGen (Si, 2015) was not able to compute a tetrahedral mesh for all datasets. Therefore we use dataset 3 as reference model.
In Figure 39, a high displacement is colored in green and a low displacement is colored in red. We see that the FEM deformation model can distinguish the flexible clips from the rigid base plate and the rigid guide rails. In the following, we will show that our new method based on the PBD framework and working solely with the surface representation can compute a similar result.

Phase I

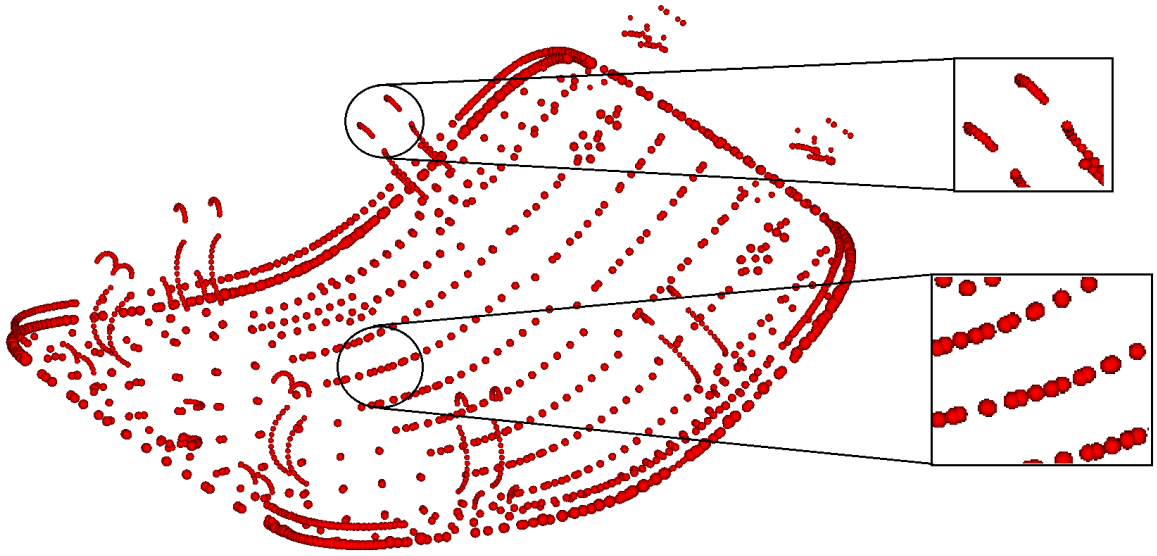


Fig. 40: The computation of the approximate medial axis spheres. Top: Small spheres of the clip. Bottom: Larger spheres of the base plate.

First, we present the results of the first phase. As previously described, we compute the approximated medial axis spheres for each vertex. For dataset 3 the spheres are shown in Figure 40. We can observe that the spheres are located approximately on the medial axis and provide a measure of the thickness of the object in each vertex. The spheres at the clips are smaller than the spheres at the base plate. In Figure 41 one can see the results of the first phase as a color coding for all four datasets. Marked in blue are vertices with large associated spheres and in yellow the vertices with small spheres. We observe that the presented method can capture the diameter and the local volume of the datasets. For dataset 4 the diameter is almost the same for every vertex. The computation time for the first phase is less than a second and therefore negligible.

Phase II

Next, we take a closer look at the results for the second phase, the labeling with the PBD approach. In Figure 42, we present three different outcomes for dataset 1. With this experiment, we want to analyze the type and number of constraints that are needed to obtain a similar result as with the FEM approach shown in Figure 39. The color code red describes a low deformation value, which means a small displacement d_i in the PBD simulation. The color code green describes a high deformation value which means large displacements d_i. The three results in Figure 42 differ in the realization of the PBD constraints.
The first labeling is computed by using the PDB framework only with a single position constraint for the two vertices of each edge in the mesh. We can see that the colors of the vertices are not distinct. Solely using the positions constraints of the surface is not sufficient and does not lead to plausible deformation values for our application. Next, we see the results when we add the bending constraints to the position constraints to get a certain amount of surface tension. We see that the labeling is now able to detect the clips as deformable. Nevertheless, the method cannot detect the base plate as rigid as the algorithm has no information about the thickness of the base plate. With the same reason, the rigid guide rails are labeled as deformable. The model can detect that the guide rails themselves are rigid, caused by the cross braces. But the method fails to classify the base plate as rigid and so the guide rails are labeled as completely deformable as they are attached to a deformable part. The last picture shows the actual result of the coloring by additionally using the thickness information from the first phase. Note that this does not add any additional constraints. We

see that the algorithm is now able to detect the base plate as rigid and therefore the guide rails are less deformable than the clips. This result is similar to the FEM result from Figure 39 with two small differences. First, the guide rails in Figure 42 are more deformable than in the FEM model from Figure 39. According to the FEM model, only the tip of the guide rail is deformable. Second, on the backside of the mesh, we see a light greenish coloring. These differences are caused by the fact that we use a model that approximates the deformations by geometric constraints and the reduction from three to two dimensions. In our benchmarks, having the application of motion planning in mind, these small artifacts did not turn out to be critical. We successfully tested the labeling with the motion planning algorithm. At last, we want to take a look at the results of the labeling for the remaining benchmarks. They are shown in Figure 43. We can see that the algorithm can detect the deformable clips for the first three benchmarks. Dataset 5 in Figure 43 shows the limitation of our approach. For this dataset, we are not able to detect the clips. This is reasoned by the fact that the clips in this example are rather thick and nearly rigid. Therefore the resulting displacements of all vertices are very similar. This is not a problem for the application as the intersection volume in the disassembly, caused by such a nearly rigid clips, is very small and our motion planner even without the labeling can handle such scenarios.

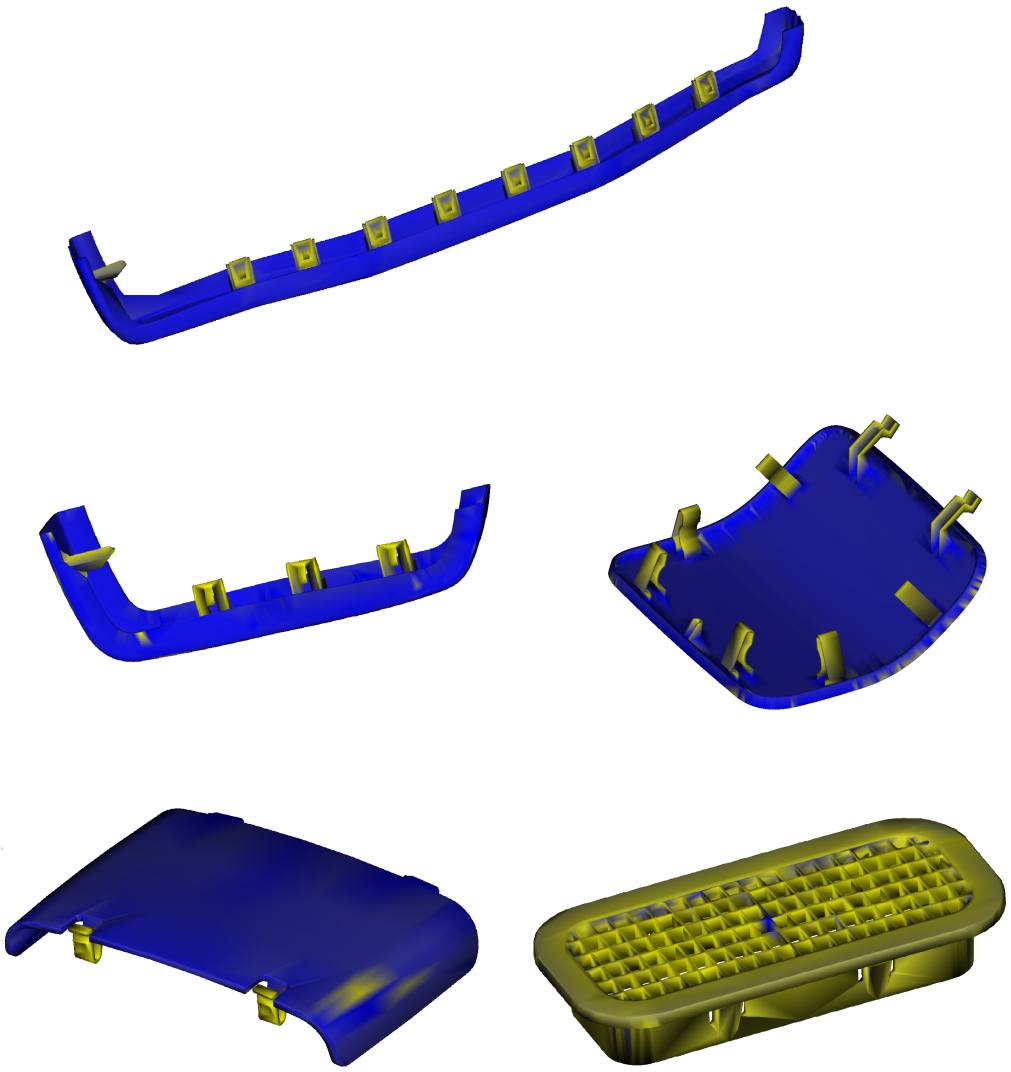


Fig. 41: Results of phase I. Vertices with a large medial axis sphere are marked in blue, vertices with a small sphere in yellow.

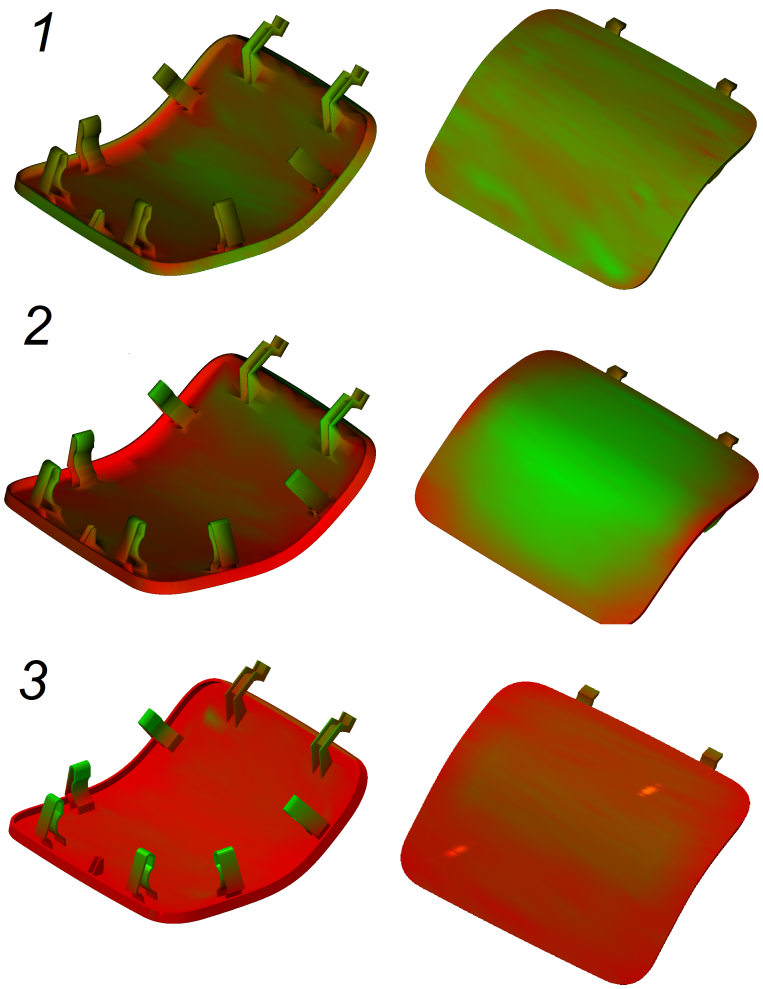


Fig. 42: Labeling with 1. position constraints, 2. positions + bending constraints and 3. positions + bending constraints + phase I.

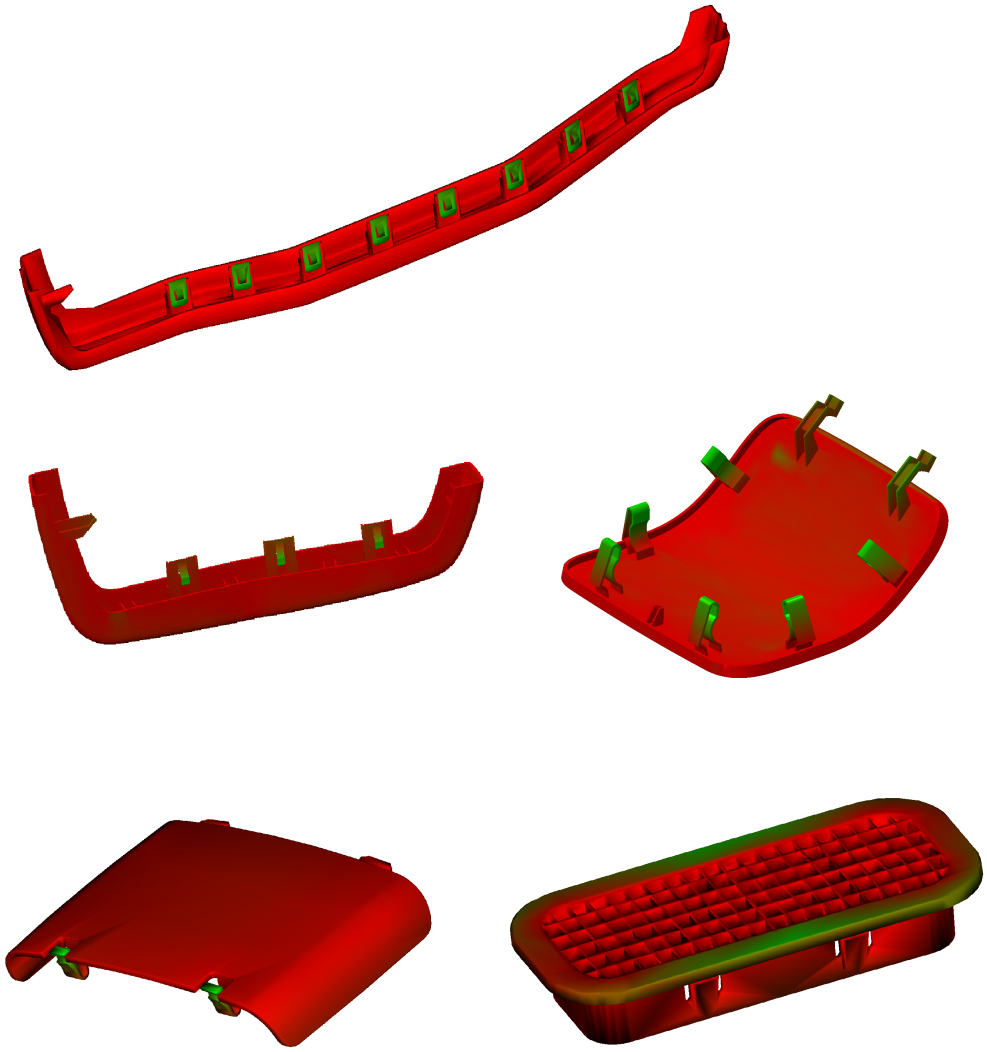


Fig. 43: The results of phase II. Vertices with a large displacement are marked in green and vertices with a small displacement in red.

Performance

The focus of the labeling is not on performance but on simplicity and small requirements of the approach. Nevertheless, we will shortly present the runtimes. For our reference model, our approach takes around 69 ms per loadcase. Computing one loadcase with the straightforward FEM approach on the tetrahedral mesh of the same reference model takes about 0.7s in average. With the 3119 loadcases of the reference model, this results in an overall computation time of over half an hour. But note that we have not optimized the FEM approach.

We want to have an approach that can compute the results so fast that it is reasonable to wait for it on a daily basis. Therefore we also parallelized our approach. In the fourth column of Table 15 we see the overall performance on a single core. The overall performance ranges from 1 to 5 minutes. In comparison, the reported single core performance for the volumetric representation in our previously published results (Schneider *et al.*, 2015a) ranges from 20 to 34 minutes. In the fifth column of Table 15 we see the overall performance if we enable CPU parallelization. Then the runtime ranges from 5 seconds to 1.5 minutes. We parallelized the computation of the labels such that each thread computes the label for one vertex. With this workload, we can achieve a speedup of 3.8 to 5.3 in our implementation which is a solid scaling for the used hyper-threaded quad-core CPU.

Table 15: Performance for the benchmark datasets.

Data	Loadcases	Time per loadcase (Singlecore)	Time Overall (Singlecore)	Time Overall (Multicore)	Speedup
Set 1	3970	69ms	274.9s	71.5s	3.84
Set 2	3262	62ms	202.2s	38.0s	5.32
Set 3	3119	67ms	208.9s	42.8s	4.88
Set 4	1233	22ms	27.7s	5.4s	5.13
Set 5	2254	49ms	111.3s	20.9s	5.33

5.5.2 InIState Motion Planner

This section presents the results of our InIState motion planner on all five benchmarks of Figure 33. In Table 16 we present the results of different approaches on all datasets.

Table 16: Results for the benchmark datasets.

Data	Shrink	IV	MLIV	Labeled-MLIV
Set 1	-	-	+	+
Set 2	-	+	+	+
Set 3	-	-	-	+
Set 4	-	-	-	+
Set 5	-	+	+	+

A negative result means that the algorithm was not able to compute a path or computed an incorrect path. We applied state-of-the-art approaches as we described in Section 5.3. First, we applied a motion planner that shrinks the dynamic object for planning and executes a state-of-the-art motion planner with a collision detection as validity function. This simple approach is not able to compute paths in any of our benchmarks as shrinking the object is only useful for invalid initial states reasoned by data inaccuracy. But in our benchmarks, the invalid states are caused by flexible fastening elements. Second, we evaluated a planner that uses the intersection volume for the validity function in a motion planning algorithm. This planner can compute paths for dataset 2 and 5 but fails in the remaining benchmarks. Third, we present a planner that uses the maximum local intersection volume (MLIV) for the motion planner solely. In addition to the intersection volume approach, this planner can compute a path for dataset 1 because the different connected components of the intersection volume are considered independently with the MLIV. But this algorithm still fails in dataset 3 and 4 as one single clip and the guide rail have the same volume, and the planner cannot distinguish between a guide rail and a fastening element. At last, we present the results of our algorithm that uses the MLIV approach combined with the labeling. This approach can compute paths for all five datasets.

In Table 17, we now see the performance of our InIState planner. In the second

Table 17: Performance for the computation of the solution path.

Data	Misc	Labeling	Estimate τ	Value for τ	Planning	Sum
Set 1	20.0s	71.5s	2.4s	5	2.1s	96.0s
Set 2	16.6s	38.0s	6.5s	20	19.3s	80.4s
Set 3	34.7s	42.8s	25.6s	646	66.8s	169.9s
Set 4	12.7s	5.4s	56.8s	2884	30.0s	104.9s
Set 5	19.8s	20.9s	(24.6s) +38.9s = 63.5s	(632) 1005	(120.0s) +23.9s = 143.9s	248.1s

column, we see the time that is needed to build up all the different data structures that are necessary for the motion planner. These data structures are, for example, the voxel fields for the static and dynamic object, the computation of the bounding volume hierarchies and other minor data structures. This task takes about 30 seconds and is executed on a single core of the CPU. In the second column, we see the runtime for the computation of the labeling. The performance is achieved by using the multicore version of our labeling approach and is the same as in Table 15. It ranges from 5 seconds to 1.5 minutes. In the third column, we see the runtime for the estimation of τ. For the parameter τ we presented a heuristic for the computation of this parameter. After applying the heuristic which is based on the expansion speed s of the tree, the planner tries to plan a path in a given planning time. If this planning fails, the process is looped until the planner finds a path. In the first four datasets the initial speed s_{init}

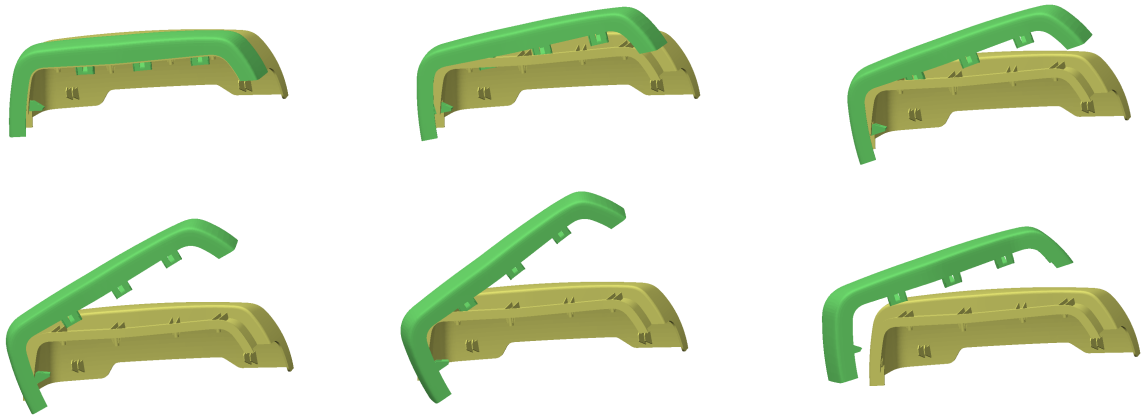


Fig. 44: The calculated path for dataset 2.

was enough to compute τ for the motion planning in the latter stage of the algorithm. For the fifth benchmark, one needs a second iteration to compute τ. The runtime of the first iteration is shown in brackets in Table 17. The runtime for the estimation of τ ranges from a few seconds up to a minute. This part of the algorithm is also parallelized. The actual values for τ are presented in the fourth column. In the fifth column we see the runtime of the actual planning with the estimated parameter τ. To get this planning time, we applied our InIState motion planner to an approach that is tuned for motion planning with narrow passages (Erbes, 2013). This algorithm is based on the Expansive Space Tree approach (Hsu *et al.*, 1997) and computes edges of the tree in parallel on the CPU. (see Section 2.2.3) So the planning time for our InIState motion planner ranges from a few seconds up to 3 minutes. In the last column we summed up the runtimes of all tasks and see that the overall runtime of our InIState

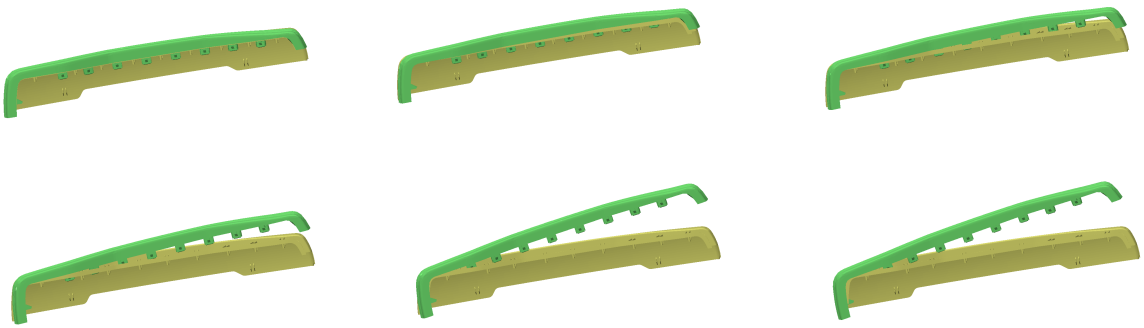


Fig. 45: The calculated path for dataset 1.

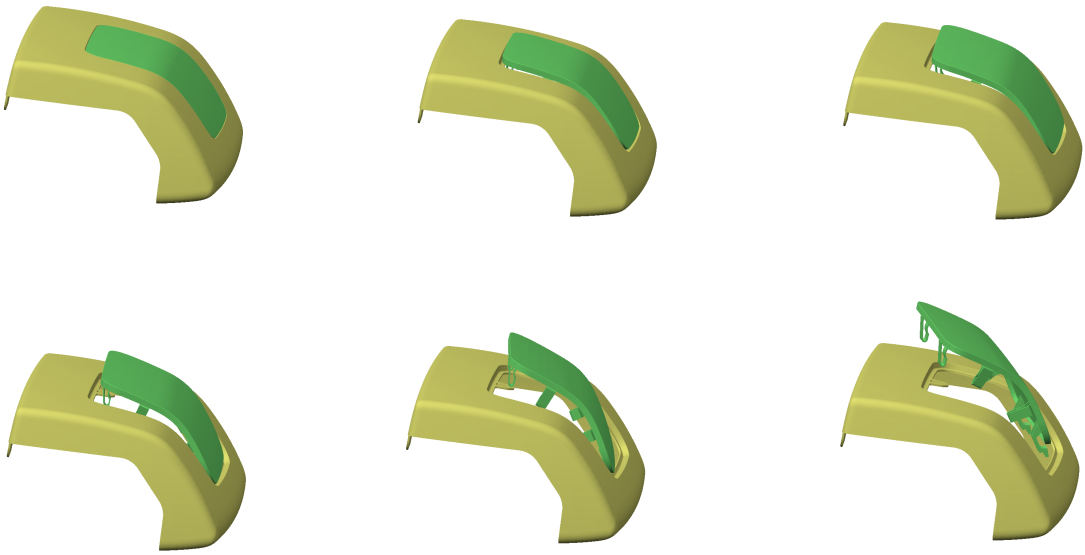


Fig. 46: The calculated path for dataset 3.

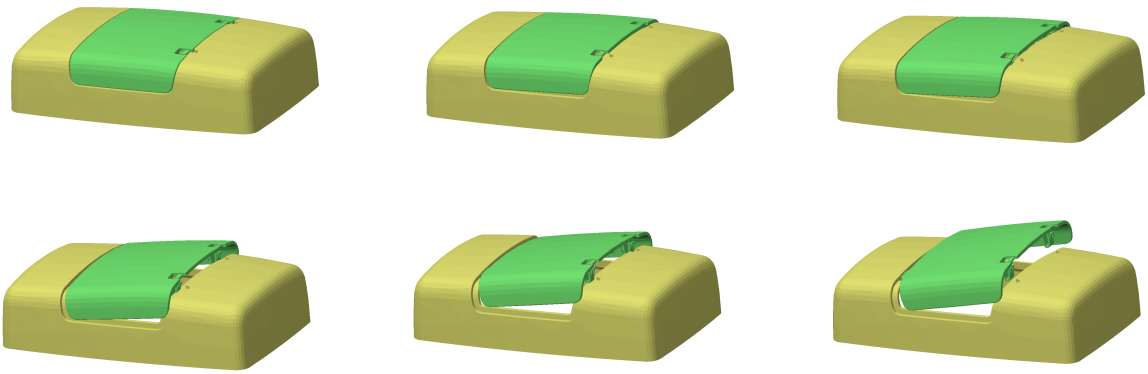


Fig. 47: The calculated path for dataset 4.

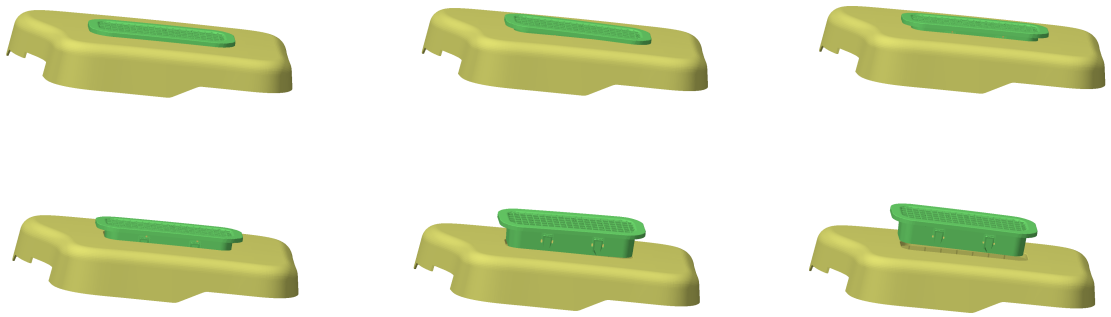


Fig. 48: The calculated path for dataset 5.

motion planner is about 1.5 to 4 minutes. At last in Figure 44 we see an example of a path that is calculated with our algorithm. It weaves out the dynamic object in a highly complex motion. The planner disassembles the dynamic object by lifting up the deformable clips one by one from right to left and by finally detaching the rigid rail on the left by a rotating downward motion. This is the correct motion path that disassembles the two objects. The paths for the remaining datasets are shown in Figure 45 to Figure 48. In those figures we see the disassembly paths that we wanted to compute for our benchmarks. For dataset 1 to 4 we see the complex motions that are needed to disassemble the dynamic objects from their environments. The disassembly path of dataset 1 is very similar to the one of dataset 2. For dataset 3 and 4 (Figure 46 and Figure 47) we see that our planner takes the rigid guide rails as well as the flexible clips into account and therefore computes the disassembly path correctly. In Figure 48 we see the linear disassembly motion (a movement along a straight line) of dataset 5. The motion path is less complex than the paths from dataset 1 to 4 but is still difficult to compute. The dynamic object in this benchmarks fits tightly into its environment which causes a narrow passage for the motion planner. This is also shown by the planning runtime for this dataset in Table 17.

5.5.3 Limitations

In the process of our work, we tested the InIState motion planner on the previously presented benchmark datasets but also on different other datasets, including original data from the automobile industry. Based on these experiments we will describe the limitations of our approach in the following. First, our approach is limited to

volumetric representations of the objects. We encountered datasets, especially very thin ones, that contain parts which are not extruded to a volumetric body in the CAD software. This means that only a surface represents this parts of the object instead of an actual volume. Therefore, one cannot compute an intersection volume and our approach ignores these parts. Such non-volumetric representations are common for older CAD software and are not permitted in modern CAD software. Second, our approach assumes that the environment has a reasonable size in comparison to the dynamic object as well as its deformable parts. Our approach computes a voxelization of the dynamic objects and its static environment. If the environment is unnecessarily large, the size of the needed RAM for the voxelization is bigger than the 8 GB of our test system. This problem can be solved by a heuristic that removes parts of the environment or by only exporting the relevant data from the CAD software. Third, for the computation of the labeling, we applied a certain amount of force to each vertex. We automatically determined the amount of force by a heuristic that is based on the thickness of the object. In some datasets, this heuristic fails. If the force is too large, the result can be that parts of the geometry are labeled as deformable, although they are rigid. And *vice versa* for the situations when the force is too small. But in our tests, the heuristic succeeded in many datasets. For the computation of the labeling, we also have to take into account, that we apply a geometric deformation model on the surface of the objects. Although we take into account the thickness, as well as the tessellation, of the mesh, it is an approximation. At last, our approach is limited to the computation of a reasonable path for the InIState motion planning problem. The evaluation of the path is left to the engineer. Approaches to evaluate the paths automatically by an algorithm have to be addressed in the future work of the InIState motion planner.

5.5.4 User Interface

At last, we want to present the user interface that was programmed, from ground up, for this work. The user interface is shown in Figure 49 and is called RasandViewer. The software is programmed in C++ using Qt 5 (Qt, 1995-2017) as the cross-platform application framework and OpenGL (OpenGL, 1997-2017) as the graphics API. The library that is used as basis and extended in this work is the RASAND library (RASAND, 2010-2017). The RasandViewer is programmed using the model-view-controller pattern to enable an easy addition to various other applications. Due to this flexibility

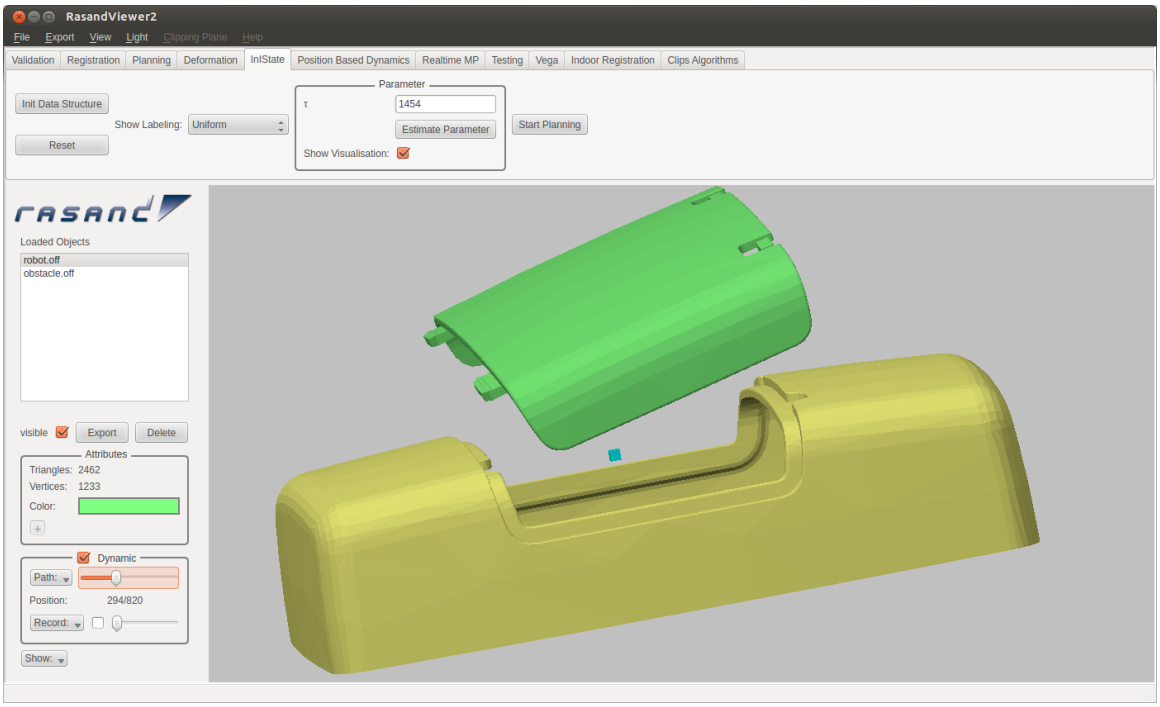


Fig. 49: RasandViewer

other applications of the RASAND library were added to the user interface. On the top of Figure 49, we see the tabs for each application like the distance validations of components, registration and much more. The forth tab contains the motion planning algorithms from Chapter 3 as well as the collision detection algorithms from Chapter 4. The fifth tab, that is visible in Figure 49, contains the user interface for our InIState motion planner that we presented in the current chapter. On the left of Figure 49, we see the loaded objects of the scene, some basic properties of the input data and at the bottom, we see a toolbox for the motion paths of the dynamic objects. In the center of Figure 49, we see the OpenGL window that visualizes the data.

In the following, we will describe the interaction with the software for the InIState motion planner. First, the user loads the geometric objects and initializes the data structures by pressing the ”Init Data Structure” button. The subsequent processing time is the sum of the second and third column of Table 17. Next, the user can visualize the result of the pre-processing, the thickness and deformation labeling, by using the drop-down box. In the next step, the user can decide whether he wants to set the parameter for the planning manually or if he intends to use the presented heuristic. In case the user sets the parameter manually, the software visualizes the parameter τ by a light-blue cube. This light-blue cube is shown in the center of Figure 49. It visualizes the intersection volume τ that is considered as valid by our InIState mo-

tion planner. After the parameter is determined, one can start the planning with the right-hand button. In the end, the computed solution path is loaded and the user can apply the path to the dynamic object and even manipulate it on the lower left-hand side. The user can manipulate the path by recording manual paths and adding them to the computed path. At last, the user can export and save the path a proprietary path file.

6

OUTLINE

6.1 Overall Conclusion

In the previous chapters, we addressed three problems in the field of motion planning in the DMU process. In the following, we want to quantify the contributions of all three approaches.

Our first approach, the Completely Randomized RRT-Connect (CR-RRTC), is an approach to three-dimensional rigid body motion planning. With a study of the main parameters of a state-of-the-art motion planning algorithm, we showed that the parameter selection is different for each benchmark dataset and that the parameters cannot be derived by geometric features of the datasets. These parameters are determined by the motion path which is unknown *a priori*. We presented an algorithm that samples the parameters $range_{max}$ and u in the motion planning algorithm uniformly during runtime. The CR-RRTC is the first motion planning algorithm that applies variable values to the parameters $range_{max}$ and u during runtime. Our approach contributes to the performance of motion planning in the majority of the presented benchmarks with a speedup up to 3.10 in comparison to the constant default values of state-of-the-art motion planning algorithms. Another important result of our approach is the findings that our new variable parameter selection, in some benchmarks, can even outperform the performance of the optimal constant parameters which can only be determined by a time-expensive study like the one that we initiated in our work. This shows us that the parameters should not be constant values over the whole planning phase of the algorithm but should be computed during the planning. Another contribution of the CR-RRTC is that the algorithm simplifies the motion planning in the DMU process. In our approach, there is no need for the engineer to define the parameters beforehand.

And at last, the CR-RRTC is easy to implement and can easily be added to existing motion planners.

In our second approach, we addressed collision detection. We presented an approach to collision detection for disassembly motion planning with narrow passages. The finding of our approach is that a huge portion of the samples in these applications are in collision with small penetration volumes. State-of-the-art collision detection algorithms which are used in common motion planners are not using this information. We presented an approach that makes use of this characteristic. For the application of three-dimensional rigid body motion planning with narrow passages, our approach achieves a speedup of 2.0 to 9.5 for the collision detection and a speedup of up to 5.0 for the overall motion planning performance in comparison to well-established collision detection libraries like the Proximity Query Package (Larsen *et al.*, 1999) and the Flexible Collision Library (Pan *et al.*, 2012). This is a substantial speedup for the motion planning in the DMU process. We evaluated our approach on a tree-based planner as well as on a state-of-the-art roadmap-based planner. For the data structures of our approach, we used well-known data structures like a distance field and a bounding volume hierarchy.

At last, we presented the Invalid Initial State Disassembly Motion Planning Problem in the context of the DMU process. We described this problem in the context of the DMU process and how it is addressed by state-of-the-art methods. We initiated a study and analyzed the main reason for this problem (Schneider *et al.*, 2013). The reason is that each and every part of the vehicle is fastened to its environment. This, in conjunction with the rigid modeling of the vehicle, causes the InIState problem. Besides the reasons and the amount of invalid initial states we also analyzed the characteristics of the invalid initial states and derived a formal definition of the problem. Based on the findings of this study a set of five datasets was presented (Hofmann, 2015) that we use in our work as benchmarks. Besides, studying the problem, we also presented a solution to the problem. We presented an algorithm that is able to compute paths for the InIState problem. These paths can be used to support the engineer in the DMU process in two major aspects. First, with our method, the engineer can judge whether the component can be disassembled or not. Second, the engineer does not have to define the disassembly path manually with sequential translations and rotations.
The key ingredients of the algorithm are a special intersection volume and a labeling of the dynamic object that is based on a deformation model. The two ingredients are included to a motion planner. Using a deformation model in the DMU process is quite

challenging due to the input datasets which only consist of triangle meshes. As we wanted to keep the requirements on the mesh low, we decided to add a deformation model from computer graphics and animation to our motion planner. This model trades physical correctness for simplicity, performance, and stability. We tested the labeling as well as the overall algorithm on the benchmarks that we presented. We showed that our InIState motion planning algorithm can compute complex non-linear disassembly paths for two objects although they collide in their initial state and along the disassembly path. The algorithm can automatically distinguish between flexible and rigid parts and can successfully incorporate this information together with the connected components of intersection volumes to a motion planner.

6.2 Future Work

In this chapter, we want to discuss the future work of our approaches. We also want to analyze which of the addressed problems we see as the most promising topic for the development of future methods in the DMU process. Therefore, we evaluate the future work of our three approaches individually in the following.

6.2.1 The Completely Randomized RRT-Connect

In the Completely Randomized RRT-Connect approach we showed that a variable selection of the parameters during runtime is advantageous. We chose to select the parameters randomly with a uniform distribution and by that we removed the need for user adjusted parameters. Also, we also improved the performance of the motion planning. The only topic for future work that is left is to investigate other variable parameter selections and improve the performance even further. Two topics could be the usage of a different distribution or a definition of the parameters based on the problem data. Regarding the first topic, we already tested different distributions during the process of our work. But these methods did not result in a consistent performance increase in comparison to the uniform selection. Regarding the second topic, we showed that the parameters are not depending on the geometry of the dynamic and static object but on the solution path. As the solution path is unknown beforehand, one

cannot use it for determining the parameters. At most, one could change the parameters during runtime based on the already computed data structure. In conclusion, we think that the benefits of the future work for the CR-RRTC approach are limited.

6.2.2 The Fast Alternating Random Ray Approach

In our FARR approach, we introduced a new data structure and algorithm to speed up collision detection in disassembly motion planning. We see the three following topics for future work. First, in our FARR approach, we use random starting points on the surface to achieve a substantial speedup in collision detection. A straightforward approach for future work is to investigate whether a different strategy for the selection of the starting points would result in an even more improved performance. In the process of our work, we already tested other strategies. For example the strategy of choosing points with a high curvature as starting points. The performance of this strategy varied in comparison to the random starting points. In some benchmarks, we obtained a higher performance than with random points but in some scenarios, the performance decreased. Therefore we stuck with random points because of simplicity of randomly selected points.
Second, in our approach, the result of one starting point is independent of the results of the other starting points. Moreover, the data structure that is used in the algorithm does not need to be altered during the motion planning. At first sight, this makes our approach a candidate for a GPU implementation. Nevertheless, the amount of starting points with round about 12 is very low. Moreover, in a tree-based motion planner, collision detection calls are sequential. This means that the collision query in the current iteration is dependent on the collision query of the previous iteration. To achieve a benefit from a GPU-based collision detection, one would need a roadmap-based planner instead of a tree-based planner. An example for such a roadmap-based planner, implemented on the GPU, is the one of Manocha *et al.* (2010). In this motion planner, a GPU version of our approach seems very promising. But roadmap-based planners are not suitable for disassembly motion planning which is the main focus of this work.
At last, our approach is focused on collision detection for rigid bodies. For future work, one can investigate whether it is possible to extend our approach to the collision detection of deformable shapes. The primary challenge in such an approach would be the update of the distance field data structure. One would need a method to update

the data structure in case the shape deforms. This update method needs to have such a high performance that it does not negate the performance benefit of the overall approach.

6.2.3 Invalid Initial State Disassembly Motion Planner

At last, we address the future work of the InIState problem. With the introduction of the Invalid Initial State Disassembly Motion Planner, we solved the problem of planning disassembly paths for objects that are not free of collision due to the various reason we presented. For future work, we want to address two topics that we will discuss in the following.

Online Approach

Classical motion planners, as well as our InIState motion planner, are offline algorithms. This means that the engineer starts the algorithm with the input data and waits till the computation is finished. A topic for future work could be the realization of an online approach. In an online approach, the engineer can follow the motion planning in realtime and even more, can interact with the motion planning algorithm. An online algorithm has benefits. Not only that the online methods are easier to integrate into the workflow of the engineer but also, the algorithm is no longer a black box for the engineer like state-of-the-art motion planners. This results in higher trust into the algorithm, helps the engineer to understand the computation of the disassembly path, and in the end, helps him to evaluate the disassembly.
Classical motion planning algorithms that can compute results in real-time are highly challenging even for low-dimensional configuration spaces. For example, Katrakazas *et al.* (2015) recently presented a survey on real-time motion planning for vehicles. This is a motion planning problem that has a three-dimensional configuration space whereas the motion planning of three-dimensional rigid bodies has a six-dimensional configuration space. Moreover, the configuration space for the motion planning of deformable objects has the same degrees of freedom as the deformation model. Nevertheless, we see a future work for real-time motion planning for the InIState problem. In the following, we will describe a method that could be used for the future development of real-time methods that address the InIState problem. We do not only describe

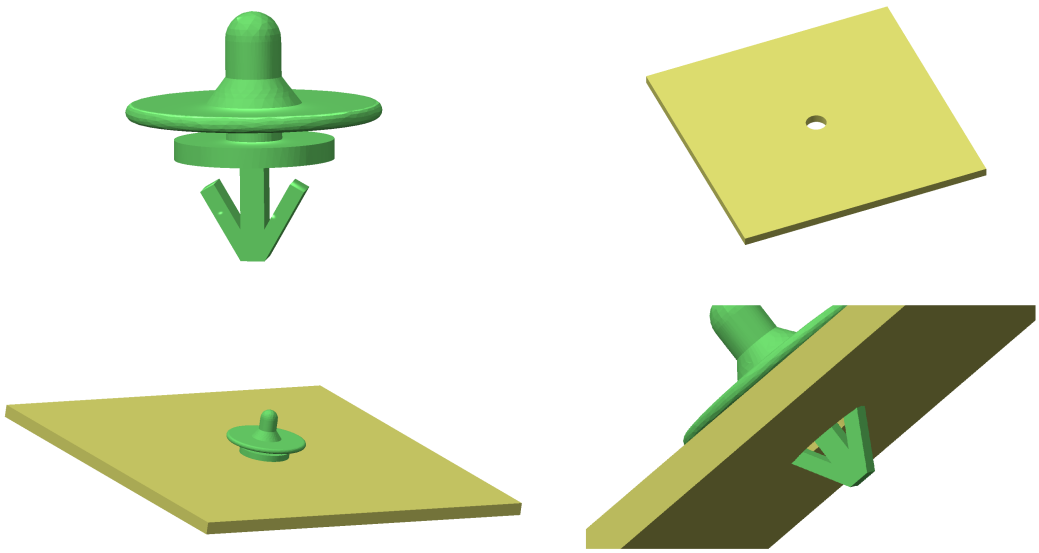


Fig. 50: Top left: Clip; Top right: Base plate; Bottom left: Initial Configuration; Bottom right: Intersection of the clip.

the method, but we also initiated a proof of concept. For the proof of concept, we use a simple dataset that is shown in Figure 50. The static object is a base plate with a hole in the center. The dynamic object is a clip that is deformable and fits the hole of the base plate. The goal of the motion planner is to compute a path for this InIState problem. We use this dataset for the following reasons. First, this dataset has a very small amount of only 2000 triangles which makes real-time deformation possible. Second, the disassembly path is approximately a motion along a straight line, which is a motion that is not as complex as the scenarios of the DMU process that we investigated in our work. The method we will present in the following has a major difference to a classical motion planner. The method works in the workspace of the problem and not on the configuration space. The basic idea is that the motion is computed by a deformation model and collision responses from the environment rather than by a data structure in the configuration space. Therefore, we start the computation from a disassembled state near the initial state. The path is computed by the interaction of the dynamic object and its environment. With the use of collision responses and the resulting forces, the dynamic object is guided from its disassembled state to its initial state. The structure of our method is shown in Figure 51. The first task is to sample a configuration c_{near} of the dynamic objects that represents a disassembled state. This means we look for a configuration that is free of collision and is close to the initial configuration.

A simple algorithm for the sampling is shown in Algorithm 13. Repeatedly the al-

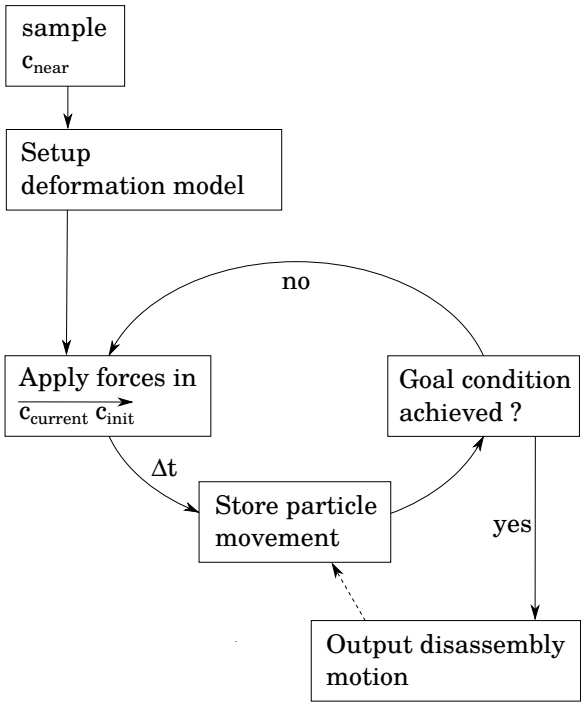


Fig. 51: The structure of the presented method.

gorithm samples a configuration, evaluates the distance to the initial configuration, and applies a collision detection until a certain time elapsed. For the measuring of the distance of two configurations in this algorithm, we measure the displacements of the vertices of the oriented bounding box of the object from the first configuration to the second configuration. After sampling c_{near} we apply forces to each vertex of the dynamic object. The forces are defined by the center of the object. We apply forces in the direction, pointing from the center of the object in the current state to the center of the object in the initial configuration. While these forces are applied to the object, the algorithm evaluates whether the dynamic object has reached its initial configuration. This evaluation is realized by measuring the distance difference of all vertices in the current and the initial configuration. If this difference reaches a local minimum, we assume that we reached the initial configuration. We detect this local minimum by interpolation over multiple timestep iterations. During the whole computation, we store the vertex displacements. These particle displacements define our path. As this algorithm is an online algorithm, the engineer could interact with the algorithm. But note, he does not necessarily need to. First, during the sampling of the disassembled configuration, the engineer could accept, reject or even guide the sampling in order to find a good starting point for the algorithm. Second, the engineer can apply custom forces to the object to guide the dynamic object to its initial configuration. We

Algorithm 13: Sampling(D, S)

```
d ← ∞                                                          1
c_sample ← 0⃗                                                   2
while !TimeElapsed do                                          3
    c_rand ← RandomState()                                     4
    if (distance(0⃗, c_rand) < d) then                          5
        if (! inCollision(D, S, c_rand)) then                  6
            c_sample ← c_rand                                  7
            d ← distance(0⃗, c_rand)                            8
return c_sample                                                9
```

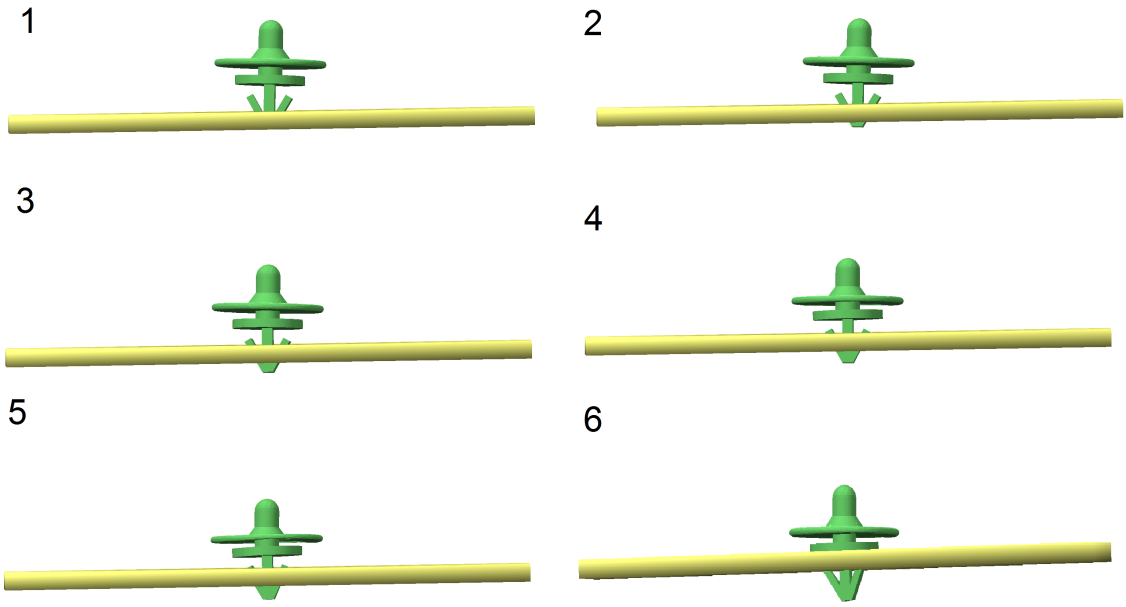


Fig. 52: Top Left: Sampled configuration; Bottom Right: Result configuration.

tested this method with our above-mentioned benchmark and obtained the results that are shown in Figure 52. The results are computed without user interaction. In the first image, we see the sampled configuration using Algorithm 13. The algorithm sampled a configuration that is free of collision and has a small distance to the initial configuration. In the following images, we see the movement of the dynamic object. This movement results from the contact forces and the directional forces to the initial configuration that we apply to the deformation model. In the last image, we see the

result configuration. This is the configuration when the algorithm decided to stop the computation of the path as the sum of all vertex distances is minimal. We see that the deformation model has deformed the outer parts of the clips as expected. Using this approach on datasets of the DMU process reveals the following two challenges for future work.

First, there is the challenge for the deformation model. In our proof of concept, we used a benchmark set with less than 500 triangles. Triangle datasets in the DMU process, like the one that we used in this work, have >>2.500 triangles. Even with modern hardware, our deformation model is not able to compute the deformations in realtime for these benchmarks. Moreover, the deformation model that we used is not tuned for the objects in the DMU process. So for future work, one could focus on different geometric-based deformation models that suit this approach.

The second challenge of the approach is the sampling of the configuration that represents the disassembled state. In our approach, we sampled a configuration that is free of collision and is near the initial configuration. This fails in more complex datasets. We explain this by the example that we see in Figure 53. In Figure 53, we see four

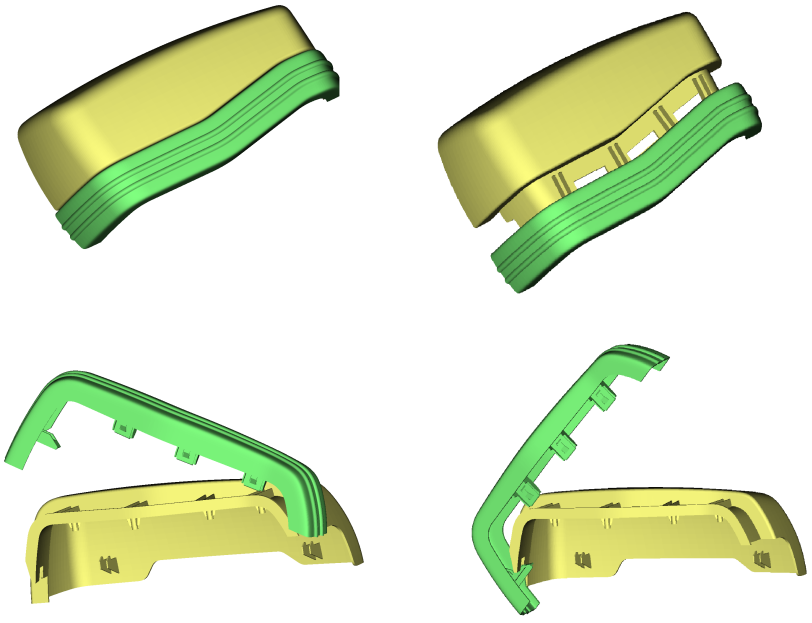


Fig. 53: The sampled configurations of dataset 2.

configurations of dataset 2. In the first image, we see the initial configuration. In the second and third image, we see two examples of configurations that are sampled by our approach. In the last image, we see the configuration that we actually need for motion planning. The problem is that the distance to the initial configuration in the

second and third image is smaller than in the fourth image. Even with a different distance measure, we obtain similar results. The statement runs as follows: For complex datasets like dataset 2 we do not only want a configuration that is free of collision and near the initial state but we want a configuration that is free of collision, near the initial state, and located on the disassembly path. In the fourth image of Figure 53 the configuration is the one that we are looking for as we know that this configuration is on the disassembly path. But as the disassembly path is the result that we want to compute, the sampling algorithm cannot distinguish between a correct and false configuration. So the future work of this method we need to take a closer look at the sampling approach.

Path Evaluation

The second topic for the future work of our InIState motion planner comes from the DMU process that we described in Figure 28. The topic that we addressed in our work is the automatic computation of the disassembly path. The remaining manual task in Figure 28 is the evaluation of the disassembly path. In this task, the engineer evaluates whether the objects can be disassembled or not. It would be beneficial to automate the path evaluation as well.
A method one could use is to use an upper threshold τ_{max} for the value τ. This threshold τ_{max} would be defined before the path computation and describes the maximal labeled intersection volume that is allowed for the path computation. If our approach can compute a path with a τ lower than τ_{max}, we consider the path as correct. But, we would need a proper heuristic for τ_{max}. As the evaluation of paths is crucial for the DMU process, one would need to evaluate the heuristic on a large number of datasets. Additionally, an approach to this problem could be to use a deformation model on the calculated disassembly path to automatically evaluate the stresses of the object. In this process, one could define certain thresholds for the stresses. Exceeding those thresholds would mean that the object breaks during the disassembly. As in our DMU process, the material properties of the objects are unknown one would need a geometric approach to this problem again.
In conclusion, we see that there are a few promising ways for future work of our InIState approach. With our InIState motion planner, we enabled the computation of the disassembly paths which creates the base for further development in the DMU process of automated disassembly planning.

OWN REFERENCES

Schneider, Daniel, Elmar Schömer, and Nicola Wolpert (2015a). "A motion planning algorithm for the invalid initial state disassembly problem". In: *Methods and Models in Automation and Robotics (MMAR), 2015 20th International Conference on*, pp. 35–40.

Schneider, Daniel, Elmar Schömer, and Nicola Wolpert (2015b). "Completely randomized RRT-connect: A case study on 3D rigid body motion planning". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2944–2950.

Schneider, Daniel, Elmar Schömer, and Nicola Wolpert (2017). "Collision Detection for 3D Rigid Body Motion Planning with Narrow Passages". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Forthcoming.

REFERENCES

Amato, Nancy M., Osman B. Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo (1998). *Choosing Good Distance Metrics and Local Planners for Probabilistic Roadmap Methods*. Tech. rep. Texas A & M University.

Amenta, Nina, Sunghee Choi, and Ravi Krishna Kolluri (2001). "The Power Crust, Unions of Balls, and the Medial Axis Transform". In: *Computational Geometry: Theory and Applications* 19.2-3, pp. 127–153.

Attene, Marco, Sagi Katz, Michela Mortara, Giuseppe Patane, Michela Spagnuolo, and Ayellet Tal (2006). "Mesh Segmentation - A Comparative Study". In: *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*. SMI '06. Washington, DC, USA: IEEE Computer Society, pp. 7–. ISBN: 0-7695-2591-1.

Bayazit, Osman B., Jyh-Ming Lien, and Nancy M. Amato (2002). "Probabilistic Roadmap Motion Planning for Deformable Objects." In: *2002 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2126–2133.

Beder, Christian and Wolfgang Förstner (2006). "Direct Solutions for Computing Cylinders from Minimal Sets of 3D Points". In: *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*. ECCV 06. Berlin, Heidelberg: Springer Verlag, pp. 135–146. ISBN: 3-540-33832-2, 978-3-540-33832-1.

Bender, Jan, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin (2014). "A Survey on Position-Based Simulation Methods in Computer Graphics". In: *Computer Graphics Forum* 33.6, pp. 228–251.

Bergen, Gino Van Den (2001). "Proximity queries and penetration depth computation on 3D game objects". In: *In Game Developers Conference*.

Bohlin, Robert and Lydia E. Kavraki (2000). "Path planning using lazy PRM". In: *2000 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1, 521–528 vol.1.

Boor, Valerie, Mark H. Overmars, and A. Frank van der Stappen (1999). "The Gaussian Sampling Strategy for Probabilistic Roadmap Planners." In: *1999 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1018–1023.

Bouaziz, Sofien, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly (2014). "Projective Dynamics: Fusing Constraint Projections for Fast Simulation". In: *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2014* 33.4, 154:1–154:11.

Dadkhah, Navid and Bérénice Mettler (2012). "Survey of Motion Planning Literature in the Presence of Uncertainty: Considerations for UAV Guidance". In: *Journal of Intelligent & Robotic Systems* 65.1, pp. 233–246.

Dammertz, Holger and Alexander Keller (2008). "Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance". In: *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing*, pp. 155–158.

Day, George S. (1981). "The Product Life Cycle: Analysis and Applications Issues". In: *Journal of Marketing* 45.4, pp. 60–67.

Döllner, Gernot, Peter Kellner, and Oliver Tegel (2000). "Digital Mock-Up And Rapid Prototyping In Automotive Product Development". In: *Journal of Integrated Design and Process Science* 4.1, pp. 55–66.

Ehmann, Stephen A. and Ming C. Lin (2001). "Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition". In: *Computer Graphics Forum* 20.3, pp. 500–511.

Erbes, Rainer (2013). "Effcient Parallel Proximity Queries and an Application to Highly Complex Motion Planning Problems with Many Narrow Passages". PhD thesis. Johannes Gutenberg - University Mainz.

Ericson, Christer (2004). *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558607323.

Ernst, Manfred and Gunther Greiner (2007). "Early Split Clipping for Bounding Volume Hierarchies". In: *2007 IEEE Symposium on Interactive Ray Tracing*, pp. 73–78.

Frank, Barbara (2013). "Techniques for Robot Motion Planning in Environments with Deformable Objects". PhD thesis. Department of Computer Science, University of Freiburg.

Frank, Barbara, Cyrill Stachniss, Rüdiger Schmedding, Matthias Teschner, and Wolfram Burgard (2014). "Learning object deformation models for robot motion planning". In: *Robotics and Autonomous Systems* 62.8, pp. 1153–1174.

Gayle, Russell, Ming C. Lin, and Dinesh Manocha (2005). "Constraint-Based Motion Planning of Deformable Robots". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1046–1053.

Gilbert, Elmer G., Daniel W. Johnson, and Sathiya S. Keerthi (1988). "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal on Robotics and Automation* 4.2, pp. 193–203.

Gottschalk, Stefan, Ming C. Lin, and Dinesh Manocha (1996). "OBBTree: A Hierarchical Structure for Rapid Interference Detection". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM, pp. 171–180. ISBN: 0-89791-746-4.

Hofmann, Andreas (2015). *Dataset created as student assistant at the University of Applied Science, Stuttgart*.

Hsu, David, Jean-Claude Latombe, and Rajeev Motwani (1997). "Path planning in expansive configuration spaces". In: *1997 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 3, 2719–2726 vol.3.

Hsu, David, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Stephen Sorkin (1998). "On Finding Narrow Passages with Probabilistic Roadmap Planners". In: *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic Perspective*. WAFR '98. Houston, Texas, USA: A. K. Peters, Ltd., pp. 141–153. ISBN: 1-56881-081-4.

Hsu, David, Tingting Jiang, John Reif, and Zheng Sun (2003). *The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners*. Tech. rep.

Jianwei Guoa Dong-Ming Yana, Er Lid, Weiming Donga, Peter Wonkab, and Xiaopeng Zhanga (2013). "Illustrating the disassembly of 3D models". In: *Computers and Graphics* 37, pp. 574–581.

Johnson, Tom C. and Judy M. Vance (2001). "The Use of the Voxmap Pointshell Method of Collision Detection in Virtual Assembly Methods Planning". In: *Design Engineering Technical Conferences and Computers and Information in Engineering Conference*.

Jones, Mark W., Andreas Baerentzen, and Milos Sramek (2006). "3D distance fields: a survey of techniques and applications". In: *IEEE Transactions on Visualization and Computer Graphics* 12.4, pp. 581–599.

Katrakazas, Christos, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka (2015). "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions". In: *Transportation Research Part C: Emerging Technologies* 60, pp. 416 –442.

Kavraki, Lydia E., Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *1996 IEEE International Conference on Robotics and Automation (ICRA)* 12.4, pp. 566–580.

Kelager, Micky, Sarah Maria Niebe, and Kenny Erleben (2010). "A Triangle Bending Constraint Model for Position-based Dynamics". In: *Proceedings of the Seventh Workshop on Virtual Reality Interactions and Physical Simulations*, pp. 31–37. ISBN: 978-3-905673-78-4.

Kleinbort, Michal, Oren Salzman, and Dan Halperin (2015). "Efficient high-quality motion planning by fast all-pairs r-nearest-neighbors". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2985–2990.

Kuffner, James J. and Steven M. LaValle (2000). "RRT-Connect: An efficient approach to single-query path planning". In: *2000 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 995–1001.

Kumar, Vijay (2007). *Rigid Body Motion and the Euclidean Group*. `http://www.seas.upenn.edu/~meam620/notes/RigidBodyMotion3.pdf`.

Lavalle, Steven M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep.

LaValle, Steven M. (2006). *Planning Algorithms*. New York, NY, USA: Cambridge University Press. ISBN: 0521862051.

LaValle, Steven M. (2011). "Motion Planning". In: *IEEE Robotics Automation Magazine* 18.1, pp. 79–89.

Leonidas Guibas, David Hsu and Li Zhang (1999). "H-Walk: hierarchical distance computation for moving convex bodies". In: *Proceedings of 15th ACM Symposium on Computational Geometry*, pp. 344–351.

Lindemann, Stephen R. and Steven M. LaValle (2005). "Current Issues in Sampling-Based Motion Planning". In: *Robotics Research. The Eleventh International Symposium: With 303 Figures*. Ed. by Paolo Dario and Raja Chatila. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 36–54. ISBN: 978-3-540-31508-7.

Liu, Tiantian, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan (Nov. 2013). "Fast Simulation of Mass-spring Systems". In: *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2013* 32.6, 214:1–214:7.

Llanas, Bernardo, Marian Fernandez de Sevilla, and Vicente Feliu (2000). "An iterative algorithm for finding a nearest pair of points in two convex subsets of Rn". In: *Computers and Mathematics with Applications* 40.8–9, pp. 971 –983.

Ma, Cherng-Min and Shu-Yen Wan (2001). "A medial-surface oriented 3-d two-subfield thinning algorithm". In: *Pattern Recognition Letters* 22.13, pp. 1439 –1446.

Mahoney, Arthur, Joshua Bross, and David Johnson (2010). "Deformable robot motion planning in a reduced-dimension configuration space". In: *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5133–5138. ISBN: 9781424450381.

Manocha, Dinesh, Christian Lauterbach, and Jia Pan (2010). "G-Planner: Real-time motion planning and global navigation using GPUs". In:

McNeely, William A., Kevin D. Puterbaugh, and James J. Troy (2005). "Six Degree-of-freedom Haptic Rendering Using Voxel Sampling". In: *ACM SIGGRAPH 2005 Courses*. SIGGRAPH '05. New York, NY, USA: ACM.

Mikel Sagardia Thomas Hulin, Carsten Preusche and Gerd Hirzinger (2008). "Improvements of the Voxmap-PointShell Algorithm — Fast Generation of Haptic Data-Structures". In: *53rd Ilmenau Scientific Colloquium*.

Morales, Marco, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato (2004). *A Machine Learning Approach for Feature-Sensitive Motion Planning*.

Müller, Matthias, Bruno Heidelberger, Marcus Hennix, and John Ratcliff (2007). "Position Based Dynamics". In: *Journal of Visual Communication and Image Representation* 18.2, pp. 109–118.

Pan, Jia, Christian Lauterbach, and Dinesh Manocha (2010). "Efficient nearest-neighbor computation for GPU-based motion planning". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, pp. 2243–2248.

Park, Chonhyon, Jia Pan, and Dinesh Manocha (2016). "Parallel Motion Planning Using Poisson-Disk Sampling". In: *IEEE Transactions on Robotics* PP.99, pp. 1–13.

Phillips-Grafflin, Calder and Dmitry Berenson (2014). "A representation of deformable objects for motion planning with no physical simulation". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 98–105.

Pivtoraiko, Mihail and Alonzo Kelly (2006). "Constrained Motion Planning in Discrete State Spaces". In: *Field and Service Robotics: Results of the 5th International Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 269–280. ISBN: 978-3-540-33453-8.

Rodriguez, Sam, Jyh-Ming Lien, and Nancy M. Amato (2006). "Planning motion in completely deformable environments". In: *2006 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2466–2471.

Salzman, Oren and Dan Halperin (2015). "Asymptotically-optimal Motion Planning using lower bounds on cost". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4167–4172.

Sam Rodriguez Xinyu Tang, Jyh-Ming Lien and Nancy M. Amato (2006). "An Obstacle-Based Rapidly-Exploring Random Tree". In: *2006 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 895–900.

Schneider, Daniel, Judith Essner, Anja Basler, Gerald Masan, Elmar Schömer, and Nicola Wolpert (2013). *Projekt Einbaupfade (eng: Project Assembly Paths)*.

Shamir, Ariel (2008). "A survey on Mesh Segmentation Techniques". In: *Computer Graphics Forum* 27 (6), pp. 1539–1556.

Shapira, Lior, Ariel Shamir, and Daniel Cohen-Or (2008). "Consistent Mesh Partitioning and Skeletonisation Using the Shape Diameter Function". In: *Vis. Comput.* 24.4, pp. 249–259.

Sifakis, Eftychios and Jernej Barbic (2012). "FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction". In: *ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. New York, NY, USA: ACM, 20:1–20:50. ISBN: 978-1-4503-1678-1.

Tang, Zesheng (1992). "Octree representation and its applications in CAD". In: *Journal of Computer Science and Technology* 7.1, pp. 29–38.

Tapia, Lydia, Shawna Thomas, Bryan Boyd, and Nancy M. Amato (2009). "An Unsupervised Adaptive Strategy for Constructing Probabilistic Roadmaps". In: *2009 IEEE International Conference on Robotics and Automation (ICRA)*. ICRA'09. Piscataway, NJ, USA: IEEE Press, pp. 2300–2307. ISBN: 978-1-4244-2788-8.

Torbert, Shane (2016). *Applied Computer Science*. 2nd. Springer Publishing Company, Incorporated. ISBN: 3319308645, 9783319308647.

Vahrenkamp, Nikolaus, Peter Kaiser, Tamim Asfour, and Rüdiger Dillmann (2011). "RDT+: A Parameter–free Algorithm for Exact Motion Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 715–722.

Wang, Jingdong, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji (2014). "Hashing for Similarity Search: A Survey". In: *Computing Research Repository (CoRR)* abs/1408.2927.

Wei Liu Yong Zeng, Michael Maletz and Dan Brisson (2009). "Product Lifecycle Management: A Survey". In: *2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/-CIE*.

Weller, René and Gabriel Zachmann (2010). "ProtoSphere: A GPU-Assisted Prototype Guided Sphere Packing Algorithm for Arbitrary Objects". In: *ACM SIGGRAPH ASIA 2010 Sketches*. SA '10. New York, NY, USA: ACM, 8:1–8:2. ISBN: 978-1-4503-0523-5.

Yershova, Anna and Steven M. LaValle (2007). "Improving motion-planning algorithms by efficient nearest-neighbor searching". In: *IEEE Transactions on Robotics* 23.1, pp. 151–157.

Zhang, Liangjun and Dinesh Manocha (2008). “An efficient retraction-based RRT planner”. In: *2008 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3743–3750.

LIBRARIES AND SOFTWARE

Barbič, Jernej, Fun Shing Sin, and Daniel Schroeder (2012). *Vega FEM Library*. `http://www.jernejbarbic.com/vega`.

Kineo$^{\text{TM}}$ (2016). *Kineo^{TM} Kite Lab 2.07.101, Kineo and KineoWorks are trademarks or registered trademarks of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries*. `http://plm.automation.siemens.com/de_de/products/open/kineo`.

Larsen, Eric, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha (1999). *Fast Proximity Queries with Swept Sphere Volumes*. `http://gamma.cs.unc.edu/SSV/`.

OpenGL (1997-2017). *Open Graphics Library (OpenGL)*. `http://www.opengl.org/`. [Online; accessed 3-February-2017].

Pan, Jia, Sachin Chitta, and Dinesh Manocha (2012). "FCL: A general purpose library for collision and proximity queries". In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3859–3866.

Qt (1995-2017). *Qt Library*. `http://www.qt.io/`. [Online; accessed 3-February-2017].

RASAND (2010-2017). *RASAND Library*. `http://rasand.info`. [Online; accessed 9-Novemeber-2016].

Si, Hang (2015). "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator". In: *ACM Transactions on Mathematical Software (TOMS)* 41.2, 11:1–11:36.

Şucan, Ioan A., Mark Moll, and Lydia E. Kavraki (2012). "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4. `http://ompl.kavrakilab.org`, pp. 72–82.