Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor Project

Disassembly Path Planning

Petr Ježek

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
May 2021

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ježek Petr** Personal ID number: **483566**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Disassembly Path Planning

Bachelor’s thesis title in Czech:

Plánování cest v úloze rozkládání objektů

Guidelines:

1. Get familiar with the assembly and disassembly path planning [1] and sampling-based planning [2].
2. Implement the matrix-based disassembly planning Breakout local search [6]. Consider 2D solid objects (represented by triangle mesh) without rotations.
3. Extend [6] by consider rotations of the objects.
4. Implement a sampling-based planner for disassembly planning,e.g., [5]. Consider 2D solid objects with rotations.
5. Extend method from 4) by precomputing manipulation paths for the active objects using a fast sampling-based planner, e.g. [3,4].
6. Experimentally verify performance of methods from 3), 4) and 5) in a suitable benchmark inspired by [5].

Bibliography / sources:

[1] Ghandi, Somayé, and Ellips Masehian. "Review and taxonomies of assembly and disassembly path planning problems and approaches." Computer-Aided Design 67 (2015): 58-86.

[2] LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.

[3] J. Denny, R. Sandström, A. Bregger, and N. M. Amato. Dynamic region-biased rapidly-exploring random trees. In Twelfth International Workshop on the Algorithmic Founda-tions of Robotics (WAFR), 2016.

[4] Vonásek, V., Pěnička, R. & Kozlíková, B. Searching Multiple Approximate Solutions in Configuration Space to Guide Sampling-Based Motion Planning. J Intell Robot Syst 100, 1527–1543 (2020). https://doi.org/10.1007/s10846-020-01247-4

[5] Duc Thanh Le, Juan Cortés, and Thierry Siméon. A path planning approach to (dis)assembly sequencing. In 2009 IEEE International Conference on Automation Science and Engineering, pages 286–291. IEEE, 2009.

[6] Somayé Ghandi and Ellips Masehian. A breakout local search (BLS) method for solving the assembly sequence planning problem. Engineering Applications of Artificial Intelligence, 39:245–266, 2015.

Name and workplace of bachelor’s thesis supervisor:

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

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

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

Assignment valid until: **30.09.2022**

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

III. Assignment receipt

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

.

Date of assignment receipt

Student’s signature

Acknowledgements

I wish to thank Ing. Vojtěch Vonásek, Ph.D. for supervising this thesis, valuable guidance and continuous encouragement. Moreover, my thanks go to my family and girlfriend for neverending support.

Declaration

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

Prague, date 18.5.2021

....
Petr Ježek

Abstract

Disassembly path planning is an important sector of robot manipulation and manufacturing of End-of-life product. This thesis focuses on developing suitable tools for 2D disassembly planning using a matrix heuristic method, sampling-based planners, such as Rapidly-exploring Random Tree algorithm and guided path search along the precomputed disassembly paths represented by the Reeb graph. Moreover, the goal is to provide a state-of-the-art of disassembly planning and disassembly path planning problems and design suitable benchmarks to properly evaluate the proposed disassembly planners.

Keywords: Disassembly, 2D assembly, Path planning, Rapidly-exploring Random Trees, Guided search, Matrix heuristic

Abstrakt

Plánování cest v úloze rozkládání objektů je důležitým prvkem při plánování pohybu a manipulaci robotů a zpracovávání a recyklaci End-of-life produktů. Tato práce se zaměřuje na návrh vhodných plánovacích algoritmů pro 2D prostředí za použití heuristických metod, algoritmů založených na vzorkování, jako například algoritmus Rychle rostoucích náhodných stromů a také na hledání cest podél předpočítaných trajektorií, reprezentované Reeb grafem. Dalším cílem je detailně popsat problém rozkládání objektů a navrhnout vhodné hlavolamy, na kterých bude možné otestovat implementované algoritmy.

Klíčová slova: Rozkládání objektů, 2D Hlavolamy, Plánování cest, Rychle rostoucí náhodné stromy, Řízené prohledávání, Maticová heuristika

Contents

Chapter 1

Introduction

The problem of disassembly planning is well known and highly important in industrial product manufacturing of End-of-life products [1, 2, 3], robot manipulation in robot simulation software, such as *Process Simulate*, and structural bioinformatics [4, 5]. The general concept is to take a given object apart, so to extract all parts from an assembly to a disassembled object — a disassembly. An assembly with a corresponding disassembly can be seen in Fig. 1.1. Nevertheless, in this thesis, only 2D assemblies are used, such as one in Fig. 1.2.

The problem can be divided into different sub-problems. The main two discussed in this thesis are Disassembly Sequence Planning and Disassembly Path Planning. The first selects a part most likely to be disassembled, and the second plans a disassembly path for the proposed part.

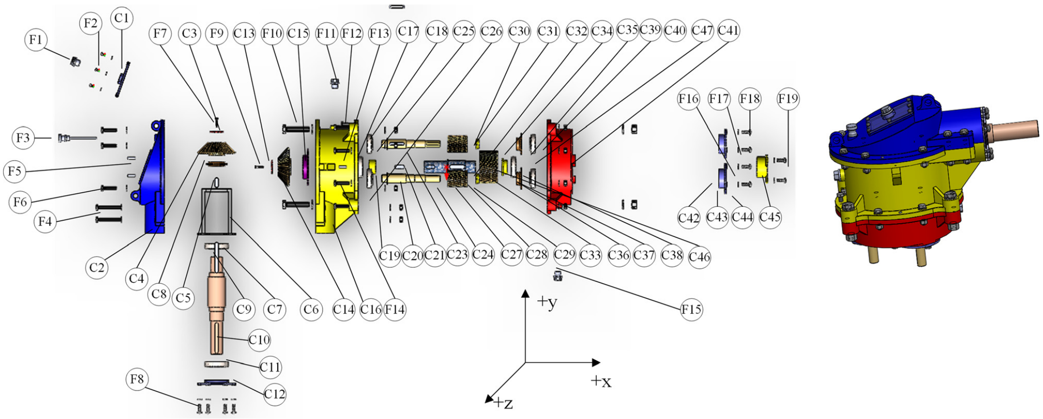
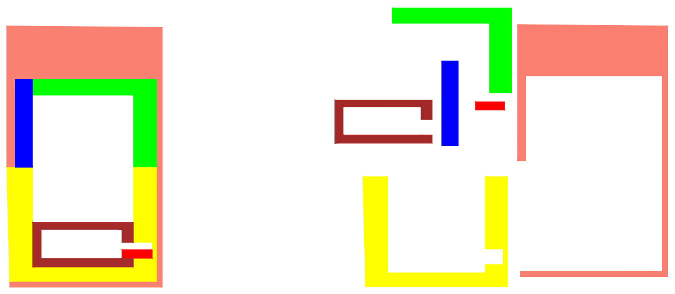


Figure 1.1: Example of complex 3D assembly (right) and disassembly (left), borrowed from [2]

Disassembly planning is applicable for a wide variety of assemblies differing in the number of parts, mobility of components or individual geometry. It follows that disassembly planning is a very complex problem. Generally, it is an NP-Hard problem [6], so for a fast performance, an effective method, often using reliable heuristic, is required. The solution methods are classified according to the completeness, representation of workspace, and way of searching the configuration space.

Many disassembly methods originally come from the motion planning problem, and usually, after some adjustments and improvements, the core idea is applicable to the problem of disassembly planning. The most suitable fast planners appropriate for various assemblies, even though not using any heuristic, are the sampling-based planners. The Rapidly-exploring Random Tree algorithm (RRT) [7] is considered as the breakthrough method whose core idea is used in plenty of contemporary path planners, such as Dynamic Region-biased Rapidly-exploring Random Tree algorithm (DRRRT) [8] or Manhattan-like Rapidly-exploring Random



(a) : Example of an assembly **(b) :** Example of a disassembly

Figure 1.2: Example of a complex 2D assembly and corresponding disassembly

Tree algorithm (ML-RRT) [9].

The solution methods based on the ML-RRT algorithm have great results for complex assemblies, even with many components, but fail in types of assemblies, where parts are close to each other (coherent or fine assemblies). In motion planning, this problem is called the narrow passage problem. Also, for disassembling an individual assembly, a reliable disassembly sequence planner is needed to correctly estimate a part most likely to be dismantled and eventually a disassembly direction along which to extract the part.

In this thesis, a suitable disassembly planner called the Matrix-based algorithm was designed. For selecting a proper part to disassemble, a heuristic algorithm inspired by [10] is used. Part is extracted in two phases: first is shifting along disassembly direction and second the sampling-based planning.

The second designed algorithm is the improved application of DRRRT in disassembly planning. It applies the same heuristic as the Matrix-based algorithm to determine part and disassembly direction, but after a guided sampling-based search is performed. The search is guided along a free space skeleton called the Reeb graph.

These designed algorithms are compared to the previously mentioned ML-RRT algorithm, which was adopted from [9] and evaluated on a wide variety of assemblies.

1.1 Goals of the thesis

The first goal is to investigate the problem of disassembly planning. In this thesis, a profound classification of types of assembly is provided, with individual examples of assemblies, partly borrowed from other publications, partly own designed. Also, several taxonomies for disassembly planning are presented.

The next goal is to implement a disassembly planner based on the Breakout Local Search Algorithm [10]. The implemented planner is called the Matrix-based algorithm and uses the matrix heuristic from [10]. The disassembly planning is performed by rotations and translations of individual parts. It also uses a sampling-based planning algorithm called RRT.

The fourth goal is to implement a sampling-based disassembly planner. The Matrix-based algorithm uses as a part of its two-phase move the RRT planning, so it can be classified as the sampling-based planner. Moreover, a planner called ML-RRT from [9] was implemented. It is further used for the evaluation and performance comparison of the designed algorithms.

The Matrix-based algorithm is further extended using the fast sampling-based planner from [8] and is called the DRRRT algorithm. It precomputes the manipulation paths in the form of a Reeb graph, and after the RRT planning is performed along the graph.

All previously mentioned algorithms are experimentally evaluated on the set of benchmarks. These assemblies are partly adopted, partly own designed.

Chapter 2

Related work for disassembly planning and disassembly path planning

This chapter focuses on the state-of-the-art of disassembly planning and one of its sub-problem — disassembly path planning. The specific terms for classifying the assemblies and solution methods are mentioned and explained. In the last section, the crucial key algorithm RRT is introduced. All the proposed solution methods in the third chapter are based on this algorithm.

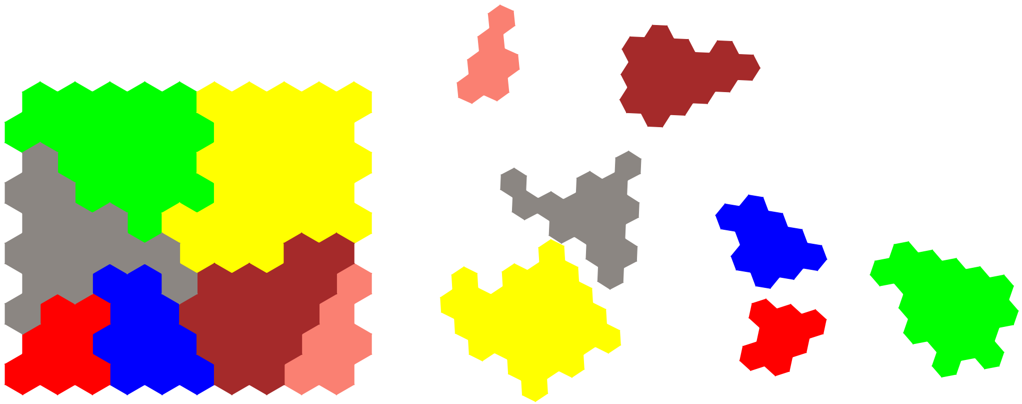
2.1 Disassembly planning

Disassembly planning (DAP) is a problem, whose task is to dismantle an object consisting of several parts called assembly (Fig. 2.1a) into a disassembled object called disassembly (Fig. 2.1b). Many applications rely on disassembling during their running process. For example, in robotics, DAP is widely used to automatically extract pieces from old devices and End-of-life products [2]. This recycling process is necessary because valuable materials can be extracted from the whole product in pure form, unlike shredding the parts as recycling is regularly done. A great advantage is that some sub-assemblies, a nonempty set of parts, can be refurbished and reused again [3]. For this purpose, partial disassembly planners are developed [11]. A review of approaches in product disassembly planning can be found in [1].

Assembling product method, known as Assembly by disassembly, is the subsequent application of disassembly planning since parts can be manipulated with more precise movements. The status of motion constraints is known better than in the disassembled state. The application is wide, mainly in the industry such as the automotive or aircraft sector. Some types of software have integrated the finding of collision-free paths of parts in digital mockups, such as *Process Simulate* by Siemens PLM Software [12]. The finding of collision-free paths is a crucial tool for robot manipulation during manufacturing operations. The advantages of finding the best disassembly sequence or optimizing the disassembly path can significantly reduce costs during manufacturing products.

In structural bioinformatics and molecular dynamics, disassembly planning is used for molecular motion tasks such as protein folding and protein-ligand interactions [4, 5].

Due to various DAP tasks of different characters and features, several taxonomies to classify disassembly problems were proposed in [12]. The first taxonomy divides the DAP problem into three sub-problems. Disassembly Sequence Planning (DASP) determines a sequence of collision-free operations, which remove a part from the assembly [13]. Disassembly Line Balancing (DALB) deals with partitioning all disassembly operations into elementary tasks, each at a specific time and assigning them to disassembly workstations. At each workstation, equal time is spent, and precedence constraints are satisfied [14]. This sub-problem is not



(a) : Example of an assembly **(b) :** Example of a disassembly

Figure 2.1: Example of assembly and corresponding disassembly

crucial for successful disassembly but for optimizing the manufacturing process and is not further discussed in this thesis.

Disassembly Path Planning (DAPP) creates collision-free sequences of positions, called paths, of parts of a given assembly from the start to goal position. The definition of these problems differs depending on the resource from which it is taken. In addition, the dissimilarity of DASP and DAPP in the survey [12] is not very noticeable. For example, in [15] DASP problem is rather defined as a listing of subsequent disassembly actions, where *actions* are defined as a separation of two or more sub-assemblies apart. According to the provided definitions, the most significant difference between DAPP and DASP is that the DAPP problem uses information about the part, which is most likely to be disassembled. This information comes from DASP. This explanation implies that each disassembly problem solution deals with these three problems in order DASP-(DALB)-DAPP.

This thesis focuses mainly on the DAPP and partly on the DASP problem and presents solutions to these tasks. The main goal is to investigate DAP methods and implement and use a suitable solver for simplified disassembly planning, considering that only one piece can move at one time and implement a suitable path planner to achieve the locomotion of the objects. Before describing the possible solution methods, the disassembly will be classified in other taxonomies from [12] and specific terms and expressions for further investigation of the disassembly task will be explained.

2.2 Notation and used terms

For classifying and understanding the disassembly tasks, the basic terms are introduced. Parts of assembly/disassembly are also called *components*. The disassembly problem can be classified depending on its geometry. It can be *unique* if the components are entirely rigid. However, the term rigid is frequently used as well. Geometry is *toleranced*, if parts are modelled as rigid with known inaccuracies such as deformation due to gravitation force, *articulated*, if several rigid parts are connected with joints, for example, problems presented in [16] or *flexible*, if parts can reversibly and freely change shape. In this thesis, all parts are considered rigid.

The critical parameter of the disassembly is its *dimension*, which is the sum of degrees of freedom of each component in the disassembly. The dimension is the main parameter of the *Configuration Space* (C-space) used as the concept for the DAPP problem. The dimensionality of the space depends on the representation of the models (assemblies). For example, the assembly in Fig. 2.1a has seven components. Each is rigid, and each has three degrees of freedom (2D translation and rotation), so the total number of DOFs and final dimensionality of the assembly and the C-space is 21. The earlier mentioned C-space is defined as the set of all configurations, where the *configuration* is the minimal list of parameters defining the location of the mobile system in the world [12] and the minimal number of parameters is the counted dimensionality. Part of C-space occupied by obstacles is called C_{obst} and part which contains all possible configurations of all parts in the assembly/disassembly is C_{free}. It goes that $C_{free} = C \setminus C_{obst}$.

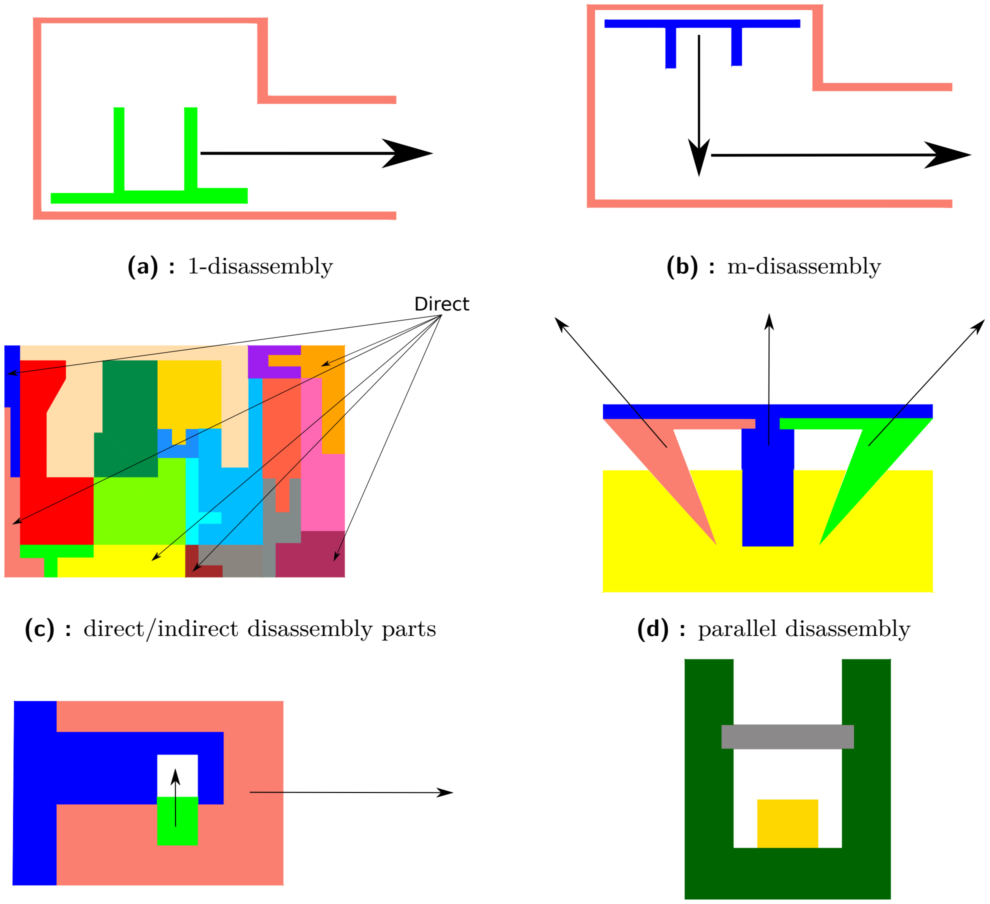
The assembly that is processed in the general DAP problem can be classified depending on the method of removing parts to achieve the final disassembly, according to [17]. The *1-/m-disassembly* means that a maximum of m motions is needed to extract a part from the whole assembly. The comparison is given in Fig. 2.2a and 2.2b. The *direct/indirect* disassembly is a problem, where a part can/cannot be removed without removing other parts, shown in Fig. 2.2c. The *sequential/non-sequential* disassembly depends on number of movable sub-assemblies at one time. Non-sequential can be also called *parallel*. Assembly in Fig. 2.2d shows that green, blue and salmon parts have to be disassembled simultaneously. In the *monotone* disassembly, a part can be directly moved out of the assembly fully. In *non-monotone* one, the part is removed after partial disassembling of one or more parts, demonstrated in Fig. 2.2e.

The disassembly is *complete* when all parts are removed from the assembly and *selective* or *partial* if only a specific subset of parts is removed. The disassembly method is *destructive* if any part has to be destroyed during disassembly and *non-destructive* in the opposite case. An example of destructive disassembly is in Fig. 2.2f, where to extract the yellow part, the grey one has to be eliminated.

2.3 Disassembly Path Planning

The path planning problem is the sub-task of disassembly planning. According to [12, 6], DAP sub-tasks, including DAPP, are either NP-hard or NP-complete. For classifying the DAPP problem, similar features according to [12] are used for the problem nature as for the more general DAP problem.

The *scale* of the problem can be *gross* if free space between assembled parts is significant, meaning that minor positional errors in disassembling the parts do not lead to losing a feasible solution. The scale is *fine* if there is no or very little free space between the parts and positional errors can lead to collision detection and losing a feasible solution. The examples of gross and fine assemblies are given in Fig. 2.3. *Sequentiality* determines the maximal number of moving sub-assemblies respect to one another during the disassembly operation (number of hands disassembling, including a hand holding the worktable). *Monotonicity* expresses if any part must be transferred to several intermediate placement positions, which lead to a feasible solution. Therefore, a non-monotone problem requires some parts to move more than once to solve the problem. *Linearity* determines if only one or more parts (a sub-assembly) are removed from the assembly at the same time. Therefore, a linear problem consists of disassembly operations, each extracting only one part from the assembly. A non-linear problem includes extracting several connected parts (a sub-assembly) from the assembly. After this extraction, each sub-assembly is disassembled separately [13]. The last feature, *coherency* is more suitable for classifying the assembly problems because it determines whether an inserted part will touch any previously inserted part. For disassembly problems, the problem is *coherent* if the parts are attached in the sub-assembly before being disassembled. The



(a) : 1-disassembly

(b) : m-disassembly

(c) : direct/indirect disassembly parts

(d) : parallel disassembly

(e) : non-monotone disassembly

(f) : destructive disassembly

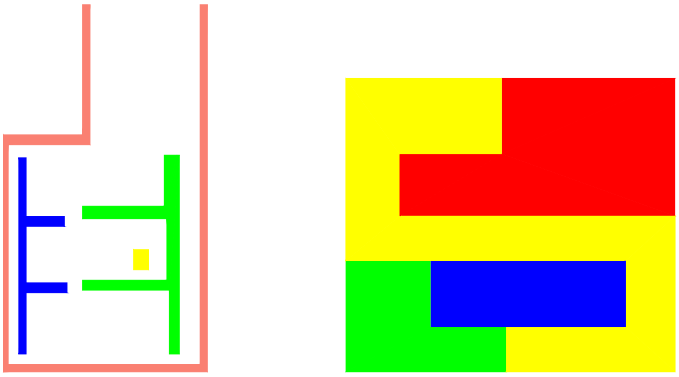
Figure 2.2: Example of disassembly classifications

other features were also originally defined for assembly problems but were easily applicable for classifying disassembly problems.

The example classification is made for several assemblies, later also used for evaluation:

- Fig. 2.13a is a four part sequential, linear, monotone and non-coherent puzzle
- Fig. 2.13c is a four part sequential, linear, monotone and coherent puzzle
- Fig. 2.13f is a four part sequential, linear, non-coherent and non-monotone puzzle, because the yellow part has to be moved to the left to successfully extract the green part
- Fig. 2.13h is a three part sequential, linear, coherent and non-monotone puzzle
- Fig. 2.13j is a four part sequential, non-linear, coherent and monotone puzzle
- Fig. 2.13l is a four part non-sequential puzzle

Finding the solution to the DAPP task depends on the completeness of the selected method. There are two conditions for drawing a line between the types of completeness. The first is that if the solution exists, the method must find it. The second is that if the solution does not exist, the method must report the non-existence of the solution. The method which accomplishes both conditions is called *complete*. This method works with the exact representation of the assembly, mainly using graph-based methods.



(a) : gross assembly

(b) : fine assembly

Figure 2.3: Scale of the problem

The methods, which do not fully accomplish one or both conditions are called *resolution complete* and *probabilistic complete*. Finding the solution with the resolution complete method depends on the considered accuracy of sampling the C-space. Usually, methods based on grid search are resolution complete because the grid is divided into small units, and the completeness depends on their size. Let a box with a tiny hole and a small ball inside be an assembly. The disassembly is achieved by pulling the ball out through the hole. The solution does not have to be found due to the low accuracy and resolution of the considered solution method.

The *probabilistic complete* method has guaranteed solution in infinite time if it exists, due to random sampling of the C-space. It implies that after an infinite amount of time, the C-space will be searched whole, and a solution will be found if it exists. These methods try to sample the configuration space without explicit representation. They attempt to sample it to a graph with a finite number of nodes and search for disassembly paths. *Probabilistic resolution complete* methods are both dependent on random sampling and resolution of the space where the method is used. It is usually some method with grid space and random sampling planning. *Incomplete* methods are those whose successful solving the problem depends on a heuristic, which does not have to find the solution, even if it exists.

2.3.1 Methods for disassembly path planning

Various approaches can achieve the solving of DAPP. Each of these approaches is considered as a separate group of solution methods. These are Graph-based, Grid-based, Sampling-based, Space Decomposition and Interactive approaches. The Graph-based methods try to represent the assembly as a graph, where the solution can be searched. The Grid-based methods discretize the C-space to a grid, which is easier to search than continuous space. The sampling-based methods represent the configuration space as a finite number of sampled points in it and search for the disassembly path in this representation. Because these algorithms are widely used in this thesis, there is a particular chapter 2.4 describing their nature. Space Decomposition methods divide the whole configuration space into several sub-spaces. The interactive methods require human-computer interaction, which may speed up the planning process with the help of human intelligence in optimizing and correcting the paths proposed by computer.

Graph-based planners can employ three different concepts for path planning. The first considers the assembly parts as nodes of the graph and the edges as their relations. These

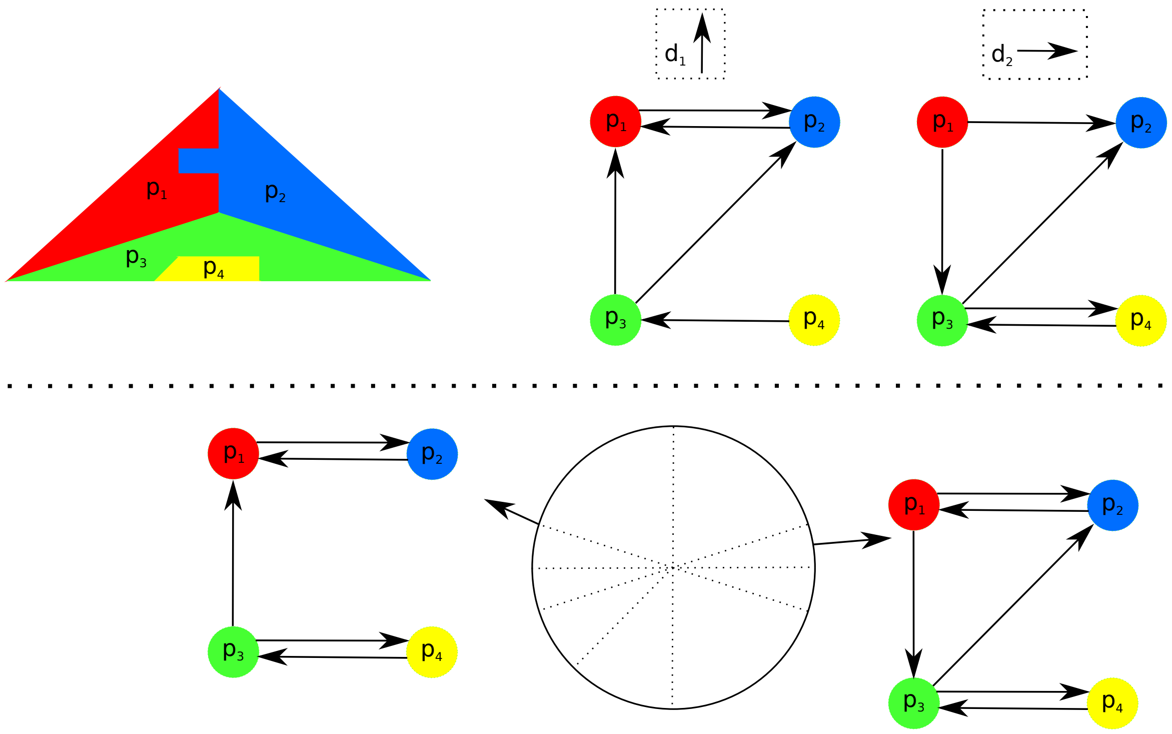


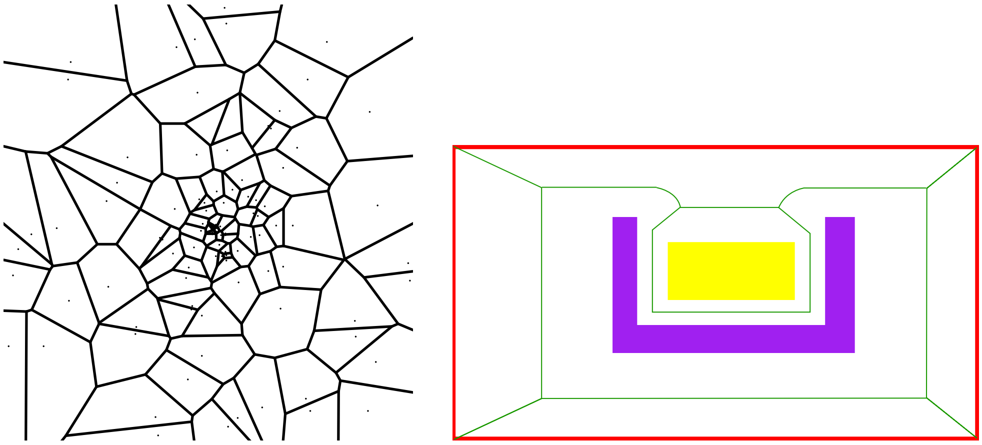
Figure 2.4: The directional blocking graph of the directions d_1 and d_2, where the nodes are parts and the oriented edges from part p to part q mean that part q hinders motion of part p in direction d. Beneath is the example of a part of Non-directional blocking graph.

relations can be, for example, connectivity, blocking, or force. The method using this concept is, for instance, the Blocking graph [18]. The second concept constructs the graph based on the geometry of the parts in the C-space. The methods using this concept are, for example, the Visibility graph method [19] and the Roadmap-based method. The third concept creates a graph from contacts and relations as nodes and the edges as transitions between them. Only the first two concepts are further explained, also with specific examples.

The Blocking graph was firstly introduced in [18]. The related work appears in two forms: the Non-directional blocking graph (NDBG) and the Directional blocking graph (DBG). The DBG has nodes corresponding to parts of the assembly, and oriented transition links from part p to part q representing that part q hinders the part p in a direction d. These graphs can be generated for each possible direction. The NDBG is created by separating the whole space of possible disassembly directions into subsets with similar DBGs. An example of Blocking graphs is shown in Fig. 2.4. The concept of blocking graph is used in the proposed algorithm in section 3.2 as the Assembly interference matrix, which is a slightly modified version of DBG.

The Roadmap-based (or Constraint-based) method, introduced in [20], integrates the Generalized Voronoi Diagram (GVD) to help to guide the parts towards the goal configuration. GVD is a locus of points that are equidistant from two or more obstacle boundaries, including the workspace boundary [21]. The example of Voronoi Diagram in 2D for obstacles represented by dots is shown in Fig. 2.5a and for a simple assembly in Fig. 2.5b.

There is another structure called Reeb Graph, which has similar features to GVD and is often used instead of it. The Reeb graph is a structure, which encodes the topology of the given object [22]. This type of graph consists of nodes and edges. The nodes are the critical points of a smooth function defined on the whole area of the object called Morse function.



(a) : Voronoi diagram (black lines) of the set of dots

(b) : Voronoi diagram (green lines) of a simple assembly consisting of the yellow and purple part. The red rectangle is the surrounding image border

Figure 2.5: Examples of Voronoi diagram

The edges are the topological conjunction between two nodes. For disassembly planning, the object is represented by the free space in the assembly. An example of Reeb Graph is shown in Fig. 2.6a and the assembly transferred to Reeb graph in Fig. 2.6b.

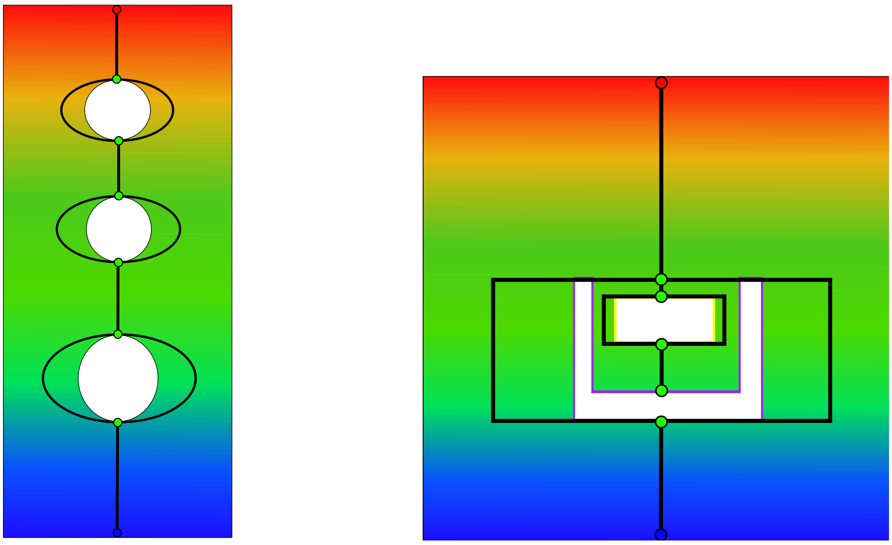
The Grid-based planners are similar resolution complete methods that explore a configuration space made of a grid. All grid-based methods have a related feature, approaching the goal configuration by a gradient search. A possible disadvantage of these methods is that they can get stuck in the local minima and never reach the goal configuration.

The Space decomposition planners try to disassemble by dividing the C-space into specific sub-spaces. These sub-spaces are subsequently searched to get the disassembly paths. The separation is made, for instance, by determining specific disassembly directions. The concept of separating direction is also used in Assembly Interference Matrix in the proposed algorithm in section 3.2.

The Interactive planners apply human intelligence through a virtual reality interface to correct, optimize, and verify the paths proposed by the disassembly method. The individual human operations are logged, and these data are used for generating instructions in the automatic disassembly process. The correction is primarily done after planning using one of the disassembly methods, which finds the collision-free path solution. However, this solution might not be optimal due to some resolution errors, high-speed rotations, or too long path possible to shorten. After finding this path, the human corrects and optimizes the path where it is possible.

2.4 Sampling-based path planning algorithms

In the last decades, sampling-based algorithms have attracted attention in motion planning, mainly due to computational efficiency and high-dimensional space operational flexibility. Sampling-based algorithms such as PRM [23], RRT [7] or more recent ones are briefly described in [24]. All sampling-based algorithms are based on constructing a graph containing feasible paths. The paths are made out of individual configurations. Those made from collision-free configurations are added to the graph. Compared to other planning algorithms, the big advantage is that sampling-based algorithms do not depend on the explicit representation of



(a) : Reeb graph of a workspace, with three circular objects (transfered to three loops of Reeb Graph) and y-value as a Morse function. Red point is its maximum, blue point the minimum and green point the saddle point.

(b) : Reeb graph of an assembly similar to the Fig. 2.5b

Figure 2.6: Examples of Reeb graph

the obstacles. The only information needed about the obstacles is if a specific part collides with any obstacle. Although most of the computational time is spent on collision detection, they are considered as fast planners in many cases. They also have a good performance in high-dimensional planning and configuration spaces with many obstacles, where classical (geometric-based) methods cannot be used at all [24].

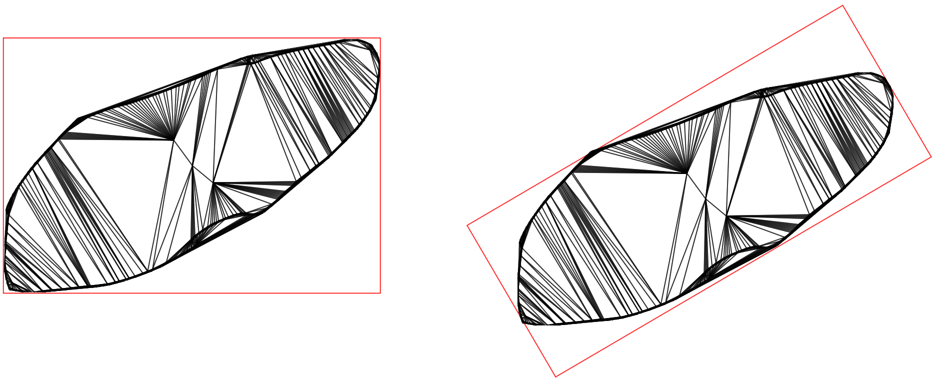
2.4.1 Collision detection

Some of the previously mentioned algorithms, such as RRT or PRM, depend mainly on fast collision detection. In this thesis, the obstacles and the moving part are considered as two separate polygonal meshes. For this type of representation, a fast solution for collision checking was introduced in [25] using Oriented Bounding Box Tree (OBBTree).

The Oriented Bounding Box wraps a complex object with a rotated rectangular box. There is a similarity to Axis-Aligned Bounding Box (AABB), which wraps the object to a box, which can not be rotated. The comparison of these two boxes is depicted in Fig. 2.7. Using OBBs, the collisions do not have to be evaluated for the enormous polygonal mesh representing the object, but only for a box, which surrounds it. If the collision is not detected with the box, there is no collision with the mesh.

Nevertheless, if the collision is detected, the box has to become tighter. The principle is in constructing the tree structure, which consists of several smaller OBBs divided from the greater one, which wrap the mesh even tighter. The collision check is done again and evaluated the same. If a collision is detected, the boxes are divided again. The depth, how many divisions are done are determined by the resolution of the algorithm.

The collision detection between the OBBs is also optimized by the separating axis theorem proposed in [26], which outperforms other collision detection algorithms. The performance is about one order of magnitude faster.



(a) : Axis Aligned Bounding Box (AABB) **(b) :** Oriented Bounding Box (OBB)

Figure 2.7: Example of an oriented and axis-aligned bounding box of a triangle mesh

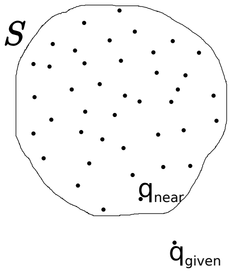


Figure 2.8: Nearest-neighbour search to q_{given} from the set S of k-dimensional points

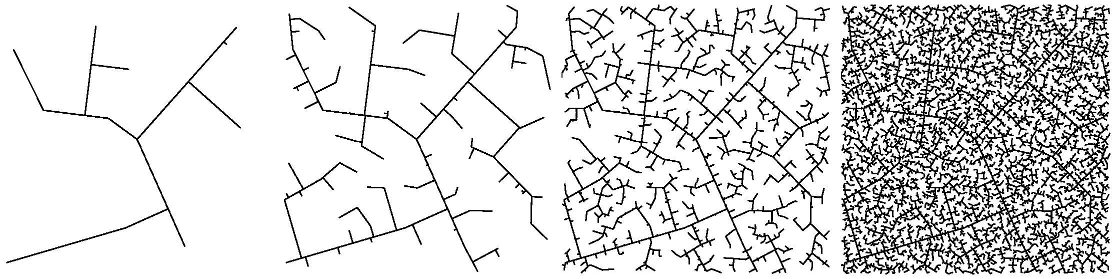
2.4.2 The nearest-neighbour search

Sampling-based planners, such as RRT, require a fast nearest-neighbour search. The nearest-neighbour search algorithm is used for finding the closest configuration q_{near} from the given set of all configurations to a given q_{given}, shown in Fig. 2.8. The naive solution is to measure Euclidean distance between each configuration in the set and the given configuration and select the nearest-neighbour as the one configuration with the smallest distance to q_{given}. This naive solution has the complexity of $\mathcal{O}(n)$, where n is the number of configurations in the set.

In this thesis, a faster solution with an average complexity of $\mathcal{O}(\log n)$ is used. The solution is based on a better configuration organization using k-dimensional tree data structure (k-d tree). A k-d tree is a binary tree structure for space partitioning.

2.4.3 Rapidly-exploring Random Tree algorithm

This sub-section introduces the Rapidly-exploring Random Tree algorithm [7] which moves the part from the start position $q_{start} \in C_{free}$ to the final position $q_{goal} \in C_{free}$. The algorithm incrementally searches the configuration space and it is probabilistic complete [27]. The collision-free configurations are stored in the tree $\mathcal{T}$. The tree is rooted in q_{start}. Each iteration, a random point q_{rand} is generated in the C-space. The tree is expanded towards q_{rand} from the nearest neighbour q_{near} found in $\mathcal{T}$. During q_{rand} approaching, the collisions of the robot (moving part) with obstacles (other parts) are detected. The collision-free path is added to the tree. These operations are repeated until the tree approaches q_{goal} to a predefined distance or after I_{max} iterations are reached. Depending on the number of iterations, the algorithm scans the workspace more or less thoroughly. An example of several RRTs depending on the number of iterations is depicted in Fig. 2.9. The visualization of the RRT algorithm is shown



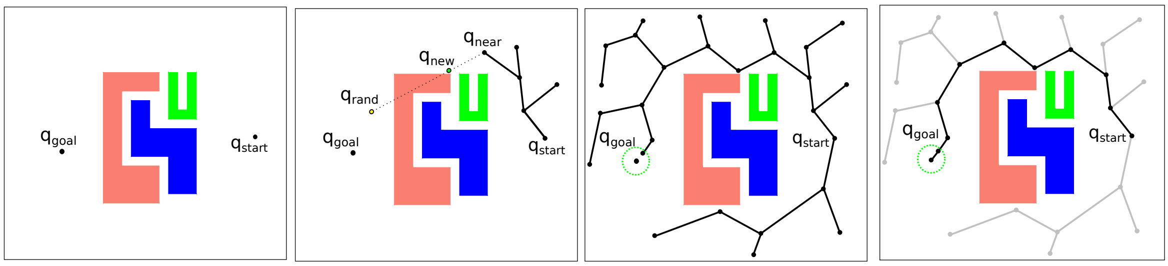
(a) : RRT Tree after 10 iterations

(b) : RRT Tree after 100 iterations

(c) : RRT Tree after 1000 iterations

(d) : RRT Tree after 10000 iterations

Figure 2.9: Visualization of RRT expansion after a various number of iterations



(a) : Start and goal position

(b) : Expansion from q_{near} to q_{rand}

(c) : The tree has grown near q_{goal}

(d) : The path from q_{start} to q_{goal} is extracted

Figure 2.10: Visualisation of RRT algorithm. In 2.10a, the tree is initialized. The expansion process is shown in 2.10b. The final grown tree, where the last expanded point in the region around q_{goal} is, is in 2.10c. The final path extracted from the tree is in 2.10d.

in Fig. 2.10.

After finishing iterating from start to goal, the path is pruned. The pruning attempts to connect the start and goal configuration (Fig. 2.11a). If the path is collision-free, the algorithm ends, otherwise the configuration q_{middle} in the half of the path is taken and the sub-paths (q_{start}, q_{middle}) and (q_{middle}, q_{goal}) are processed similarly as the start-goal path, shown in Fig. 2.11b. The algorithm ends when the pruned path from the start to the goal is completed, depicted in Fig. 2.11c. The algorithm for pruning is Alg. 1.

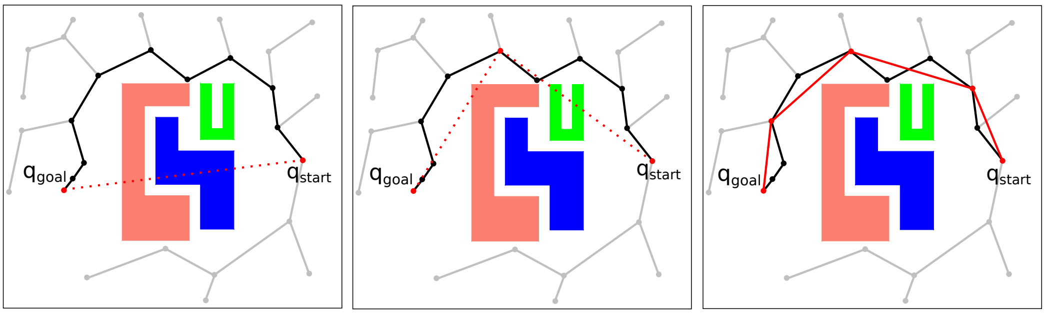
Algorithm 1: `PathPrunning`

```
Input: Path $P = \{q_{start}, ..., q_{goal}\}$, Pruned path $P_p$, begin configuration $q_{begin}$, end
       configuration $q_{end}$
Output: Updated pruned path $P_p$

if sub-path $(q_{begin}, q_{end})$ is collision-free then
    $P_p$.add(($q_{begin}, q_{end}$));
else
    $q_{middle} \leftarrow$ FindMiddlePoint($pp$);     // middle config. $P$ between $q_{begin}$ and $q_{end}$
    PathPrunning($P$, $P_p$, $q_{begin}$, $q_{middle}$);
    PathPrunning($P$, $P_p$, $q_{middle}$, $q_{end}$);
```

2.4.4 Dynamic Region-biased Rapidly-exploring Random Tree algorithm

This algorithm belongs to a family of RRT-based methods. Its main advantage is that it requires fewer iterations for completing the path planning task than basic RRT. It also explores



(a) : Attempt to connect q_{start} with q_{goal}

(b) : Algorithm depth one sub-paths

(c) : Final successful connection from q_{start} to q_{goal}

Figure 2.11: Visualization of the path pruning method. In 2.11a the first attempt to prune the path by connecting q_{start} and q_{goal} is made (red). Because it failed (it crosses obstacles), the path is halved, and the attempt is repeated for each sub-path in depth one of the recursive algorithm (2.11b). Finally in depth of two, the complete collision-free path from q_{start} to q_{goal} is found in 2.11c.

parts of space where the probability of selecting a random point is low. These parts of free space are called narrow passages. Dynamic Region-biasing helps sample these narrow passages by constructing the Reeb graph as a skeleton of the input workspace. After constructing the graph, the dynamic region (a circle with centre on the Reeb graph) is moved along the graph, and the RRT algorithm generates nodes inside these regions [8].

The DRRRT method takes the input environment and start and goal configuration of the part to be disassembled (Fig. 2.12a). Firstly, the free space is triangulated using the Delaunay triangulation of the free space (Fig. 2.12b). The obstacles (other parts of the assembly, which are not currently disassembled) are taken as holes in the workspace. The triangulated mesh is used to compute the Free Space Skeleton represented as the Reeb Graph (Fig. 2.12c), and then the start and goal are connected to this skeleton, shown in Fig. 2.12d. Then, the first region is initialized in the start configuration (Fig. 2.12e), and RRT explores the environment by generating points in the advancing regions along the Reeb graph, depicted in Fig. 2.12f and Fig. 2.12g. The final path is extracted (Fig. 2.12h), and the active part is moved to the final position, shown in Fig. 2.12i.

The overview of this planner is shown in Alg. 4. The method `ComputeFreeSpaceSkeleton` computes the Reeb graph. For biased tree growing, a search tree and initial region are initialized, and after, the method `Region-biasedRRTGrowth` attempts to reach the goal configuration until I_{max} iterations. When finished, the final path of the disassembled part is extracted from the tree and pruned.

Compute Free Space Skeleton

This method overview is shown in Alg. 2. Firstly, the Delaunay triangulation converts the environment e into mesh M of triangles. It is crucial to triangle only free space, not obstacle space, because the disassembly manipulation is executed only in free space.

Line 2 computes the Reeb graph from the triangulated mesh using the VTK library [29]. This library employs the algorithm from [22]. The algorithm processes the triangles online from the mesh M. Each new vertex in the mesh is added to the Reeb graph. For each edge of the triangle, which is new to the Reeb graph, Arc's structure is created. Arc connects two points in the Reeb graph and is made of edges. After adding nodes and arcs, the surface of the triangle is added. Inserting the surface of a triangle does merging of arcs into one containing edges from both arcs. After processing all triangles, the final Reeb graph is finished. Then

```
Algorithm 2: ComputeFreeSpaceSkeleton
  Input: Environment $e$, Positions $q_{start}$ and $q_{goal}$
  Output: Free Space Skeleton $S_{free}$
  $M \leftarrow$ Triangulation($e$);                               // using [28]
  $RG \leftarrow$ ComputeReebGraph($M$);              // using [29], who are using [22]
  $S_{free} \leftarrow$ ConnectReebGraphToStartAndGoal($RG, q_{start}, q_{goal}$);
  return $S_{free}$;
```

the nodes q_{start} and q_{goal} are connected to the Reeb graph.

Region-biased RRT Growth

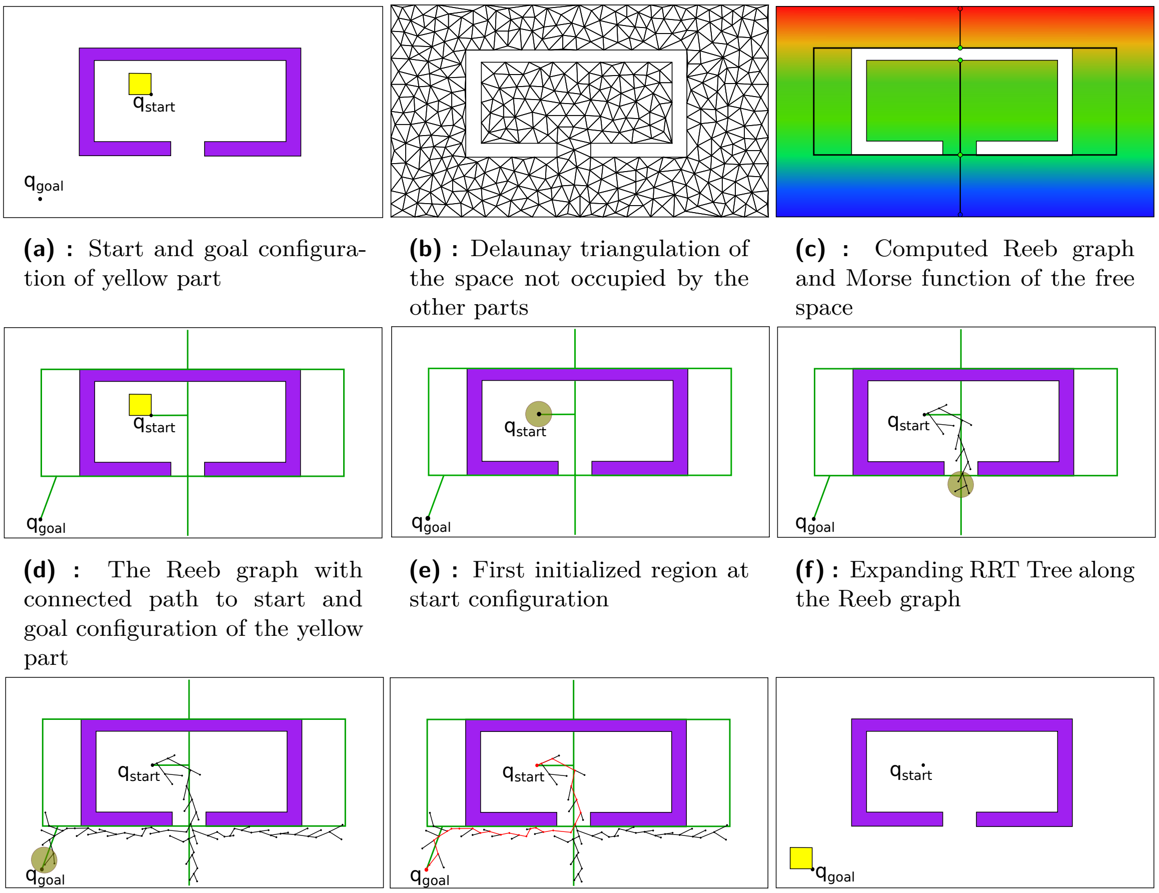
In Alg. 3 the strategy of growing RRT Tree in the dynamic region is shown. On lines 2 - 5, a standard RRT algorithm is performed with the alteration of generating random points. Regular RRT selects q_{rand} from the whole configuration space, but Region-biased RRT selects q_{rand} from the dynamic region R with the center on S_{free}. After I_{max} attempts to expand the tree are executed, the region is advanced along S_{free} to another node of the Reeb graph.

```
Algorithm 3: Region-biasedRRTGrowth
  Input: Tree $\mathcal{T}$, Region $R$, Free Space Skeleton $S_{free}$
1 for $i = 1, \ldots, I_{max}$ do
2     $q_{rand} \leftarrow$ RandomPointFromRegion($R$);
3     $q_{near} \leftarrow$ NearestNeighbour($q_{rand}, \mathcal{T}$);
4     $q_{new} \leftarrow$ Expand($q_{near}, q_{rand}$);
5     $\mathcal{T}$.AddNewNodeAndEdge($q_{near}, q_{new}$);
6 $R \leftarrow$ ChangeRegion($S_{free}$);
```

2.5 Conclusion

This chapter introduces the problem of disassembly planning. Several taxonomies to classify assemblies and the disassembly methods are introduced. Also, the nature of the problem in terms of complexity and resolution is discussed. The representation of assembly as a graph and the blocking relations between parts are mentioned and are further used in the Matrix-based algorithm in section 3.2. For the DRRRT algorithm, a particular assembly representation called the Reeb graph is described, and several examples are given. From the disassembly methods, the sampling-based planners are explained more profoundly because they are primarily used in this thesis. Also, the tools, such as fast collision checking and nearest-neighbour search, used for sampling-based planning are introduced and described. The RRT and DRRRT algorithms are explained in detail, including their overviews.

```
Algorithm 4: DRRRT
  Input: Environment $e$, Positions $q_{start}$ and $q_{goal}$
  Output: Path from $q_{start}$ to $q_{goal}$, Tree $\mathcal{T}$
  $S_{free}$ ← ComputeFreeSpaceSkeleton($e$, $q_{start}$, $q_{goal}$);        // in Alg. 2
  $R$ ← InitialRegion($q_{start}$) ;                          // first region for biased growing
  $\mathcal{T}$ ← InitializeRRTTree($q_{start}$);
  for $i = 1, \ldots, I_{max}$ do
      Region-biasedRRTGrowth($\mathcal{T}, R, S_{free}$);                        // in Alg. 3
      if near $q_{goal}$ then
          break;
  $p$ ← ExtractStartGoalPathFromRRTTree($\mathcal{T}$);
```



(a) : Start and goal configuration of yellow part

(b) : Delaunay triangulation of the space not occupied by the other parts

(c) : Computed Reeb graph and Morse function of the free space

(d) : The Reeb graph with connected path to start and goal configuration of the yellow part

(e) : First initialized region at start configuration

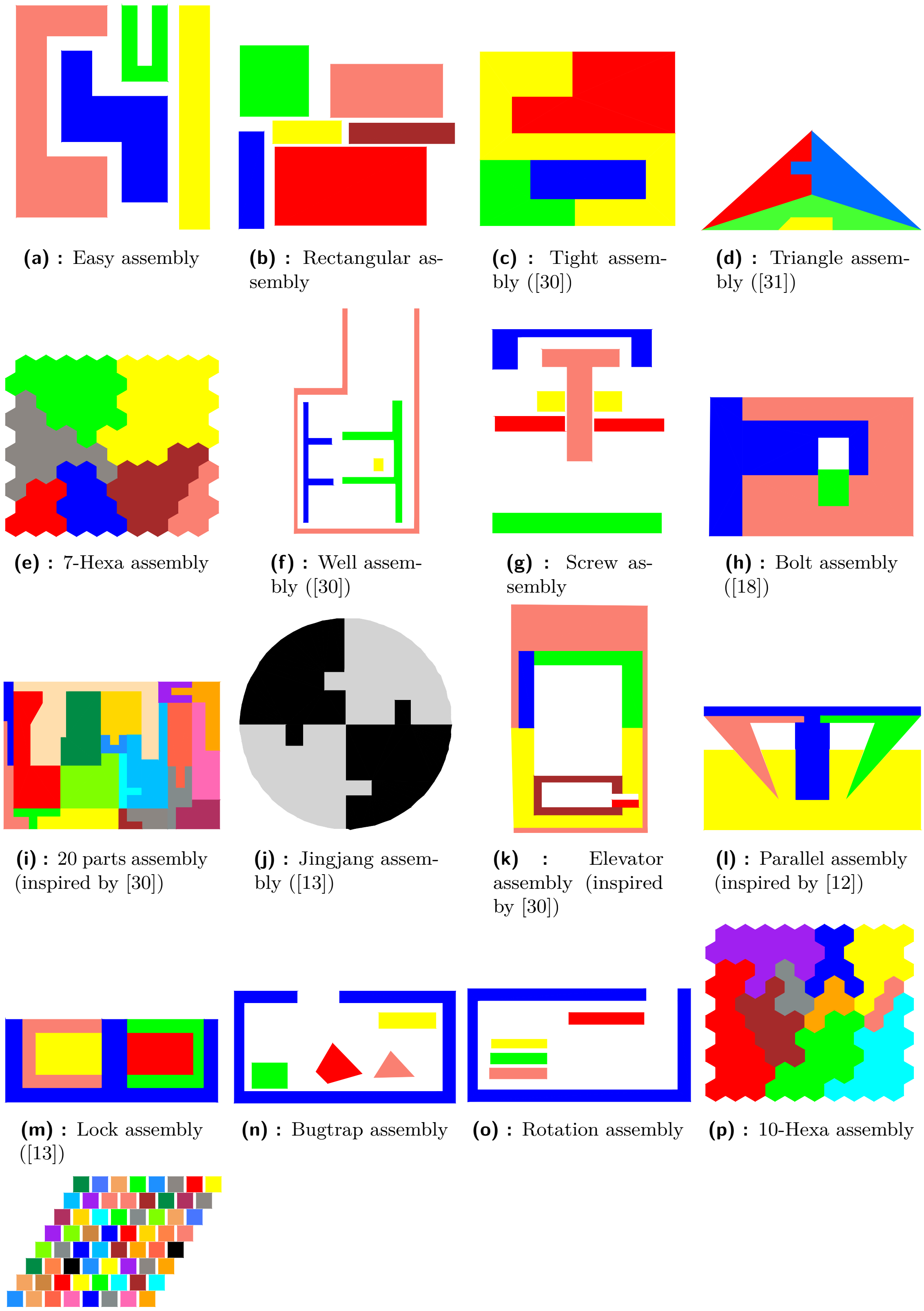
(f) : Expanding RRT Tree along the Reeb graph

(g) : The region and the RRT Tree has reached the goal configuration

(h) : The extracted red final path

(i) : The yellow part is moved to goal configuration

Figure 2.12: Visualisation of the DRRRT algorithm. In 2.12a q_{start} and q_{goal} are depicted. Then, the free space, where the yellow part can move, is triangulated. It can be seen in 2.12b. The obstacle (purple part) is excluded in triangulation and creates the Reeb graph, shown in 2.12c. The Morse function determining the Reeb graph is the y-value of the workspace, depicted as the gradient spectrum in 2.12c. q_{start} and q_{goal} are connected to the Reeb graph by a straight line, shown in 2.12d. The first dynamic region is initialized in q_{start}, shown at 2.12e. The RRT grows in the regions, and the regions are advanced along the Reeb graph (2.12f). Until the region reaches q_{goal}, the regions are advanced, and the tree is expanded, shown in 2.12g. The final path from the tree is extracted (the red path in 2.12h). Finally, the yellow part is moved along the red path from q_{start} to q_{goal}, depicted in 2.12i.



(a) : Easy assembly

(b) : Rectangular assembly

(c) : Tight assembly ([30])

(d) : Triangle assembly ([31])

(e) : 7-Hexa assembly

(f) : Well assembly ([30])

(g) : Screw assembly

(h) : Bolt assembly ([18])

(i) : 20 parts assembly (inspired by [30])

(j) : Jingjang assembly ([13])

(k) : Elevator assembly (inspired by [30])

(l) : Parallel assembly (inspired by [12])

(m) : Lock assembly ([13])

(n) : Bugtrap assembly

(o) : Rotation assembly

(p) : 10-Hexa assembly

(q) : Chessboard assembly

Figure 2.13: Puzzles used for evaluation

Chapter 3

Disassembly methods

This chapter summarizes the solution methods, which were implemented and evaluated in this thesis. Firstly, the ML-RRT method [9] is presented. This algorithm was implemented (not designed) as a benchmark comparison to the other two own designed algorithms. These are the Matrix-based algorithm and the DRRRT algorithm. All of these algorithms are explained, and the overview of pseudo-codes is presented as well.

3.1 ML-RRT algorithm

The first implemented algorithm in this thesis is a Manhattan-like Rapidly-exploring Random Tree algorithm (ML-RRT) taken from [9]. The fundamental concept of the ML-RRT algorithm is based on dividing the C-space, with dimensionality computed as the sum of all DOFs of all parts, to specific sub-manifolds and letting the RRT algorithm search in those.

3.1.1 C-space representation in ML-RRT algorithm

The sub-manifolds are called active (C_a) and passive (C_p). The active sub-manifold consists of variable parameters (motion configurations) of the selected part (active part). The passive sub-manifold has varying passive part parameters. Thus, as an example in Fig. 3.1, the red part is chosen as the active part, the configuration space is separated into the active red parameters and passive yellow, blue and green parameters.

3.1.2 ML-RRT algorithm overview

The algorithm tries to move with the selected active part. After the first collision, the passive parts are moved a little, not to block the active part's potential movement. After the passive parts finish moving, the active part moving repeats. After the part is disassembled, the algorithm repeats for another selected active part.

The main loop is depicted in Alg. 5. The parts are chosen one by one until all are attempted to disassemble. In the method `SelectPart` a part to be disassembled is selected. The order of parts is chosen as the one given by the user. In `SetPartition` the C-space is divided into active and passive sub-manifold as described in the previous subsection. Next, the search tree $\mathcal{T}$ for RRT planning is initialized with an initial configuration consisting of the actual position of the active part. The tree $\mathcal{T}_{all}$ contains an initial configuration consisting of positions of all parts. The selected part is then disassembled, or after I_{max} iterations, the failure to disassemble is reported.

The method for planning the path of the active part is executed in function `ExpandML-RRT` in Alg. 6. The main task is to move with the active part. If it is not possible due to collisions,

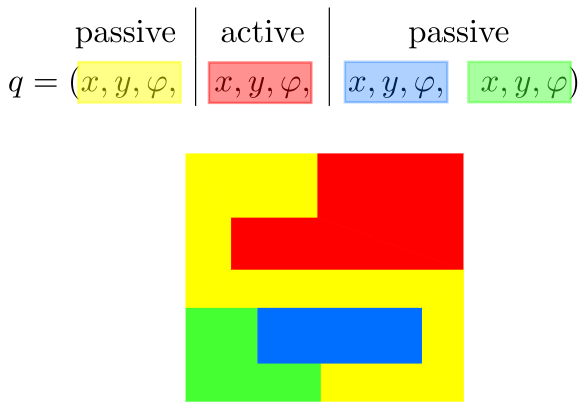

passive
active
passive
$q = (x, y, \varphi, \; x, y, \varphi, \; x, y, \varphi, \; x, y, \varphi)$

Figure 3.1: Active and passive part of the configuration

```
Algorithm 5: IterativeML-RRT
Input: Parts $P$, Configuration Space $C$, initial configuration $q_{init}$
$n$ ←number of parts;
for $i = 1, \ldots, n$ do
    $p_i$ ← SelectPart($P$);
    $(C_a, C_p)$ ← SetPartition($C, p_i$);
    $\mathcal{T}$.add($q^{act}_{init}$);
    $\mathcal{T}_{all}$.add($q_{init}$);
    for $j = 0, \ldots, I_{max}$ do
        Disassembled ← false;
        Disassembled ← ExpandML-RRT($P, C, \mathcal{T}, \mathcal{T}_{all}, C_a, C_p$);    // method from Alg. 6
        if Disassembled then
            break;
```

then the passive blocking ones are relocated. The algorithm is based on the RRT algorithm. The key difference is the previously mentioned sub-manifold planning. Unlike sampling a random configuration in the whole configuration space, the configuration q^{act}_{rand} is sampled only in C_a, shown at line 1. The nearest-neighbour search for finding q_{near} is also executed in C_a. The `Expand` function at line 3 proceeds from q_{near} towards q^{act}_{rand} until collision. The last valid collision-free configuration q_{new} with list P^p_{col} with all colliding parts stopping the expansion is returned for further steps. The configuration q_{new} is added to $\mathcal{T}_{all}$, also the active part to $\mathcal{T}$ and the reference between these two nodes is made.

Subsequently, until the list P^p_{col} contains colliding parts, the active part is attempted to unblock further moves by perturbing the passive parameters of parts associated with P^p_{col}. Note that the perturbation is carried out in the nearby neighbourhood of the passive parameters. The `Expand` function moves passive parts from q_{near} towards q^{pas}_{rand} until collision. During this expansion process, the collisions are detected and added to a list $P^{p'}_{col}$. The new node q_{new} has only the passive part perturbed so that the new node is added only to $\mathcal{T}_{all}$. The `ChangeReference` function is further discussed in the next section because it requires an introduction to sub-manifold nearest-neighbour search. The list $P^{p'}_{col}$ is relatively compared to P^p_{col} and only the blocking parts, which are not in P^p_{col} yet are considered further as parts hindering the motion of active parts.

3.1.3 Sub-manifold nearest-neighbour search

There is a noticeably unclear issue with the nearest-neighbour search in the sub-manifold. Several configurations with the same active parts but different passive ones can be selected as the nearest-neighbour when using the distance metric in C_a. In the article where ML-RRT is introduced ([9]), there is no information about how to handle this problem.

The solution designed in this thesis (already mentioned in the Algorithm overview) selects one neighbour, which was expanded the last. It is achieved by a tree $\mathcal{T}_{all}$, where all configurations are saved, also with passive parts. Each configuration in $\mathcal{T}$ has a reference to a leaf node in $\mathcal{T}_{all}$ to determine the correct last expanded passive part configuration successfully. This determination is made at line 2 of Alg. 6. After adding q_{new} to $\mathcal{T}_{all}$ in the second part of the Alg. 6, reference of the node q^{act}_{near} in $\mathcal{T}$ has to be changed to the leaf node (q_{new}) in $\mathcal{T}_{all}$.

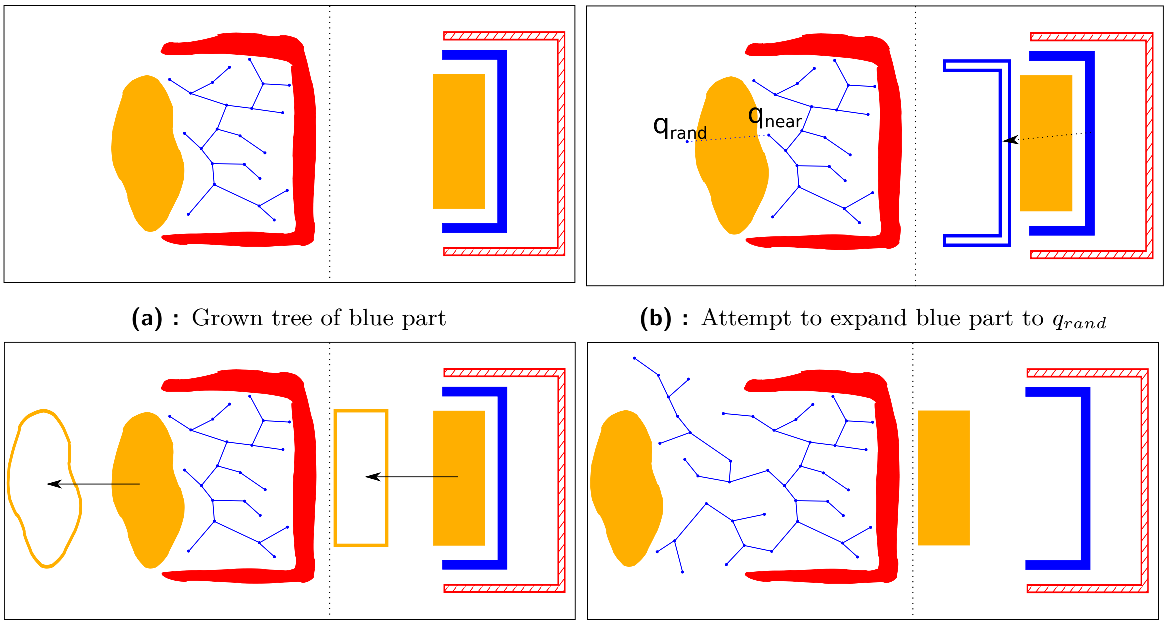
Algorithm 6: `ExpandML-RRT`

```
Input: Parts $P$, Configuration Space $C$, tree $\mathcal{T}$, all configuration tree $\mathcal{T}_{all}$, active partition
       $C_a$, passive partition $C_p$
Output: updated trees $\mathcal{T}$ and $\mathcal{T}_{all}$
1  $q^{act}_{rand} \leftarrow$ SampleConf$(C, C_a)$;
2  $q_{near} \leftarrow$ BestNeighbor$(\mathcal{T}, \mathcal{T}_{all}, q^{act}_{rand}, C_a)$;     // the sub-manifold nearest-neighbour
3  $(q_{new}, P^p_{col}) \leftarrow$ Expand$(q_{near}, q^{act}_{rand})$;
4  $\mathcal{T}_{all}$.AddNewNodeAndEdge$(q_{near}, q_{new})$;
5  $\mathcal{T}$.AddNewNodeAndEdge$(q^{act}_{near}, q^{act}_{new})$;
6  $q_{near} \leftarrow q_{new}$;
7  while $P^p_{col} \neq \emptyset$ do
8      $q^{pas}_{rand} \leftarrow$ PerturbConf$(C, q_{near}, P^p_{col})$;
9      $(q_{new}, P^{p'}_{col}) \leftarrow$ Expand$(q_{near}, q^{pas}_{rand})$;
10     $\mathcal{T}_{all}$.AddNewNodeAndEdge$(q_{near}, q_{new})$;
11     $\mathcal{T}[q^{act}_{near}]$.ChangeReference$(\mathcal{T}_{all}[q_{new}])$;
12     $q_{near} \leftarrow q_{new}$;
13     $P^p_{col} \leftarrow P^{p'}_{col} \setminus P^p_{col}$;
14 $Disassembled \leftarrow$ false;
15 if $q_{near}$ $near$ $q^{act}_{goal}$ then
16     $Disassembled \leftarrow$ true;
17 return $Disassembled$;
```

An example visualization of the ML-RRT principle is shown in Fig. 3.2. It is supposed that the tree of the blue part is expanded until the state shown in Fig. 3.2a. After that, the algorithm samples q_{rand} configuration and the nearest configuration from the blue part tree q_{near} is found. The sampling is depicted in Fig. 3.2b. The expansion is not possible because the orange part is hindering the motion of the blue one. The algorithm detects the collision and the blocking part. Then the parameters of the blocking part (in this example, the orange one) are perturbed and expanded to a position shown at Fig. 3.2c. After blocking part perturbation and expansion, the blue part is released, and its tree can be further expanded as shown in Fig. 3.2d.

3.2 Matrix-based algorithm

The main concept of the Matrix-based algorithm combines the matrix heuristic method used in Breakout Local Search algorithm [10] for finding the disassembly sequence, and RRT algorithm [7]. The heuristic method plans a sequence of pairs of parts and directions (disassembly sequence). The parts are then moved along the disassembly directions in the



(a) : Grown tree of blue part

(b) : Attempt to expand blue part to q_{rand}

(c) : Relocation of the orange part to clear the way for the blue part

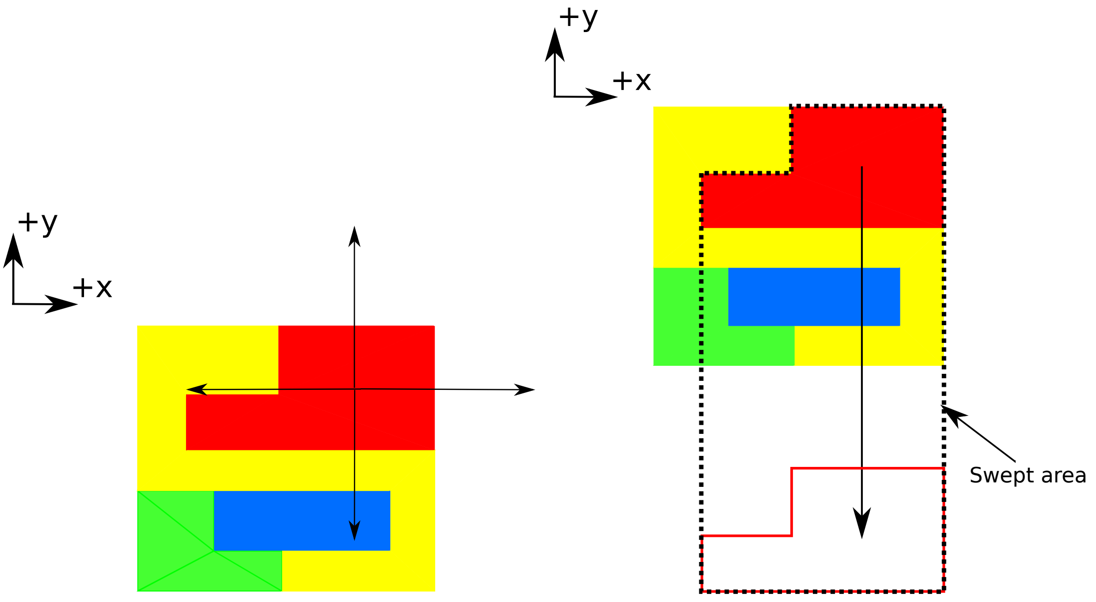
(d) : After the relocation, the blue part tree can be further expanded

Figure 3.2: Visualization of the ML-RRT algorithm, where the C-space is on the left and the workspace on the right. Blue is the active part, red is the stable part, which means that it can not be moved. Orange is another part blocking the blue one. In 3.2a the grown tree of blue part configurations is shown. In 3.2b another configuration of the blue part q_{rand} is sampled, and an attempt to expand the tree to this position is made. The blocking part is detected (the orange part), and in the second part of the Alg. 6 at line 8 the orange part position is perturbed and then at line 9, the orange part is expanded to the new sampled position, shown in 3.2c. The blue part is unblocked and can be further expanded, as shown in 3.2d.

disassembly sequence. Afterwards, the RRT algorithm plans the path towards a predefined final position. The first part, the assembly sequence computation, requires determining which part is restricted by which part in each direction. For example, in Fig. 3.3a, the red part is unrestricted in $+x$ direction (to the right), but the yellow part blocks any other direction, meaning $-x$ (to the left), $+y$ (up) and $-y$ (down). Moreover, $-y$ direction is not directly blocked by the blue and green part, but when the red part is moved along that direction, the swept area (black dotted space) collides with them (Fig. 3.3b). That is the reason why these parts are also considered blocking. From this information about blocking parts, a disassembly sequence using the heuristic algorithm is computed. Each part in the disassembly sequence is then moved along the corresponding direction for a predefined distance. The second phase, path-finding to the predefined goal position, is executed subsequently.

3.2.1 Matrix heuristic

The algorithm will be demonstrated step by step for the square puzzle, which is depicted in Fig. 3.3a and for four disassembly directions shown in Fig. 3.3a. The representation of individual components and their blocking parts is coded in a structure called Assembly Interference Matrix (AIM). AIM is an $n \times (n \times m)$ matrix, where n is the number of parts, and m is the number of disassembly directions. The AIM for the square puzzle is shown in Fig. 3.4. Each column in this matrix represents one disassembly direction d_k for particular part p_j written above the matrix and particular blocking part q_i as a row of AIM. Each



(a) : Disassembly directions

(b) : Swept area of the red part

Figure 3.3: Disassembly directions and swept area of the red part

element is either 1, meaning that the swept area of part p_j is blocked by part q_i in direction d_k, or 0 otherwise. More precisely, the matrix element can be expressed as $I_{q_i,p_j}^{sw(p_j,d_k)}$, where $sw(p_j, d_k)$ is the swept area of the part p_j along the direction d_k. For instance, the first column represents moving of the yellow part in the $-x$ direction. The yellow part's swept area collides with a green and blue part, so the third and the fourth element in the column are 1 and the others 0. Another example is given for the red part moving in $-y$ direction in section 3.2, where the term swept area is introduced.

There are another three matrices used in the further introduced algorithm. The Directional Assembly Interference Matrix ($DAIM$) shown in Fig. 3.5 sums up all columns in AIM one by one. So an element in $DAIM$ is total number of blocking parts for a part p_i in a direction d_k. The element can be also expressed as

$$DAIM(p_i, d_k) = \sum_{j=1}^{n} I_{q_i,p_j}^{sw(p_j,d_k)}. \tag{3.1}$$

As an example, the element $(p_1, -x)$ in $DAIM$ is sum of the first column in AIM.

For the second matrix, called Disassembled Blocking Parts Matrix (DBP), let p_2 and p_4 be the disassembled parts. By using AIM and already disassembled parts, the DBP of size $(n - c) \times m$ is computed, where c is the number of already disassembled parts, here 2. The disassembly would look as shown at Fig. 3.6 also with the DBP matrix. Each row in DBP is representing one non-extracted part p_i. For each of these parts, the total number of interferences with all disassembled parts π_j in all directions d_k is detected. Easier way to visualize the process of creating DBP is when all rows in AIM corresponding to disassembled parts are crossed out and then all disassembled parts sub-matrices are summed up. Example is in Fig. 3.6. So

$$DBP(p_i, d_k) = \sum_{all\ \pi_j} I_{q_i,\pi_j}^{sw(\pi_j,d_k)}, \tag{3.2}$$

where q_i does not belong to the set of disassembled parts.

$$AIM = \begin{array}{c}\\ \\ \end{array}\begin{array}{c|cccc|cccc|cccc|cccc|}
 & \multicolumn{4}{c}{p_1} & \multicolumn{4}{c}{p_2} & \multicolumn{4}{c}{p_3} & \multicolumn{4}{c}{p_4} \\
 & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y \\
p_1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
p_2 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
p_3 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
p_4 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{array}$$

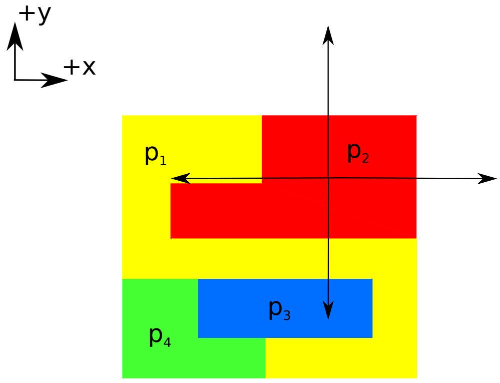


Figure 3.4: Example of Assembly Interference Matrix

$$DAIM = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 2 & 1 & 3 & 2 \\
p_2 & 1 & 0 & 3 & 1 \\
p_3 & 1 & 1 & 2 & 2 \\
p_4 & 0 & 2 & 0 & 3
\end{array}$$

Figure 3.5: Example of Directional Assembly Interference Matrix

The last matrix of the same size as DBP called Not Disassembled Blocking Parts Matrix (NBP) supposes as before at DBP matrix, that p_2 and p_4 are disassembled. Each row in NBP represents a non-extracted part. For each of these parts p_i and each direction d_k, we detect the number of interferences of all other not disassembled parts. For easier visualization, it is proceeded from the same concept of crossed rows of AIM. The difference is that after crossing, all columns corresponding to not yet disassembled parts are summed up, and the sum is written to NBP matrix. Note that when summing, we exclude the crossed rows of AIM. Example is given in Fig. 3.7. Therefore

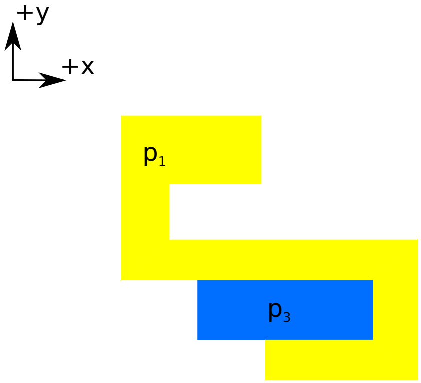
$$NBP(p_i, d_k) = \sum_{all\ q_i} I_{q_i,p_j}^{sw(p_j,d_k)}, \tag{3.3}$$

where p_j and q_i do not belong to the set of disassembled parts.

The heuristic algorithm is depicted in Alg. 7. The inputs are AIM and all parts the assembly is made of. At the beginning, $DAIM$ is created in `CreateDAIM`.

The first (random) disassembly direction d_1 is chosen in `RandomDirection`. According to selected direction a first part to be disassembled π_1 is selected in method `Min` as the one, which

$$AIM = \begin{array}{c|cccc cccc cccc cccc|}
 & \multicolumn{4}{c}{\overbrace{\qquad\qquad\qquad}^{p_1}} & \multicolumn{4}{c}{\overbrace{\qquad\qquad\qquad}^{p_2}} & \multicolumn{4}{c}{\overbrace{\qquad\qquad\qquad}^{p_3}} & \multicolumn{4}{c}{\overbrace{\qquad\qquad\qquad}^{p_4}} \\
 & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y \\
p_1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
\cancel{p_2} & \cancel{0} & \cancel{1} & \cancel{1} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} \\
p_3 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
\cancel{p_4} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0}
\end{array}$$



$$DBP = \begin{array}{c|cccc|}
 & -x & +x & -y & +y \\
p_1 & 1 & 1 & 1 & 2 \\
p_3 & 0 & 1 & 1 & 1
\end{array}$$

Figure 3.6: Example of the DBP matrix, the modified puzzle and the reduced AIM matrix

has a minimal number of blocking parts along the direction d_1 in $DAIM$. More precisely

$$\pi_1 = \underset{i}{\operatorname{argmin}}\{DAIM(p_i, d_1)\}. \tag{3.4}$$

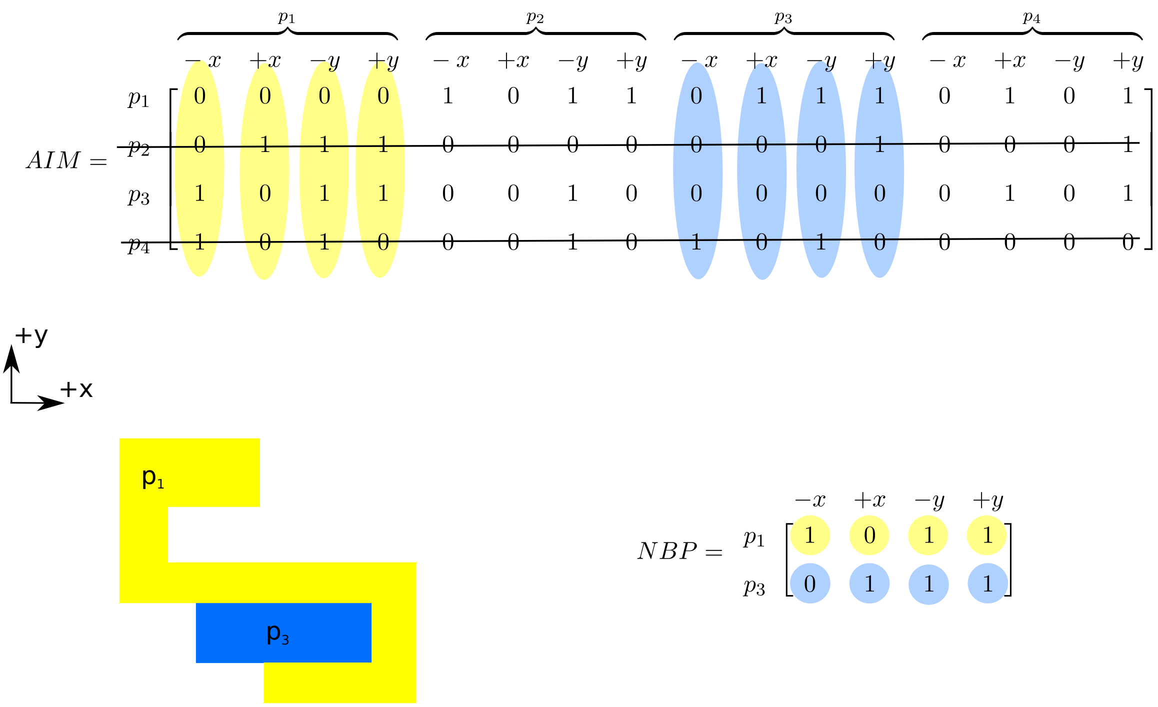
By first disassembling pair of direction d_1 and part π_1, the final disassembly sequence DS is initialized. The set of finished parts F contains only π_1. In the for-loop, the matrices DBP and NBP are created using methods `DisassembledBlockingParts` and `NotDisassembledBlockingParts` using the AIM and mainly the information about finished parts in F. Next disassembly pair of (π_i, d_i) is the one with minimal sum of DBP and NBP counted in method `Min`, mathematically as follows

$$(\pi_i, d_i) = \underset{k,j}{\operatorname{argmin}}\{DBP(p_k, d_j) + NBP(p_k, d_j)\}. \tag{3.5}$$

If there is more than one optimal pair, a random one from them is selected. At the end, disassembly sequence $\{(\pi_1, d_1), ..., (\pi_n, d_n)\}$ is returned. Further on, an example run of the above algorithm for an easy four-component puzzle is also provided with visualization at Fig. 3.8 and 3.9.

At the beginning of the algorithm, the $DAIM$ from AIM of a given puzzle is created, as shown in Fig. 3.8a. After that, a random direction d_1 is chosen and appropriate part π_1 with minimal value in $DAIM$ is selected, see Fig. 3.8b. Next step is creating DBP and NBP matrices considering p_2 as the disassembled part. All pairs of parts and directions corresponding to the smallest elements of $DBP + NBP$ matrix are found and a random pair is chosen as the next part and direction to be disassembled (π_2, d_2). Illustration is given in Fig. 3.8c. In Fig. 3.9a, parts p_2 and p_1 are already disassembled. DBP and NBP are created and best option from $DBP + NBP$ is chosen (part p_3). In Fig. 3.9b the last part is processed in a similar way as the last two parts. The final disassembly sequence is

$$(o_1, o_2, o_3, o_4) = \langle(p_2, +x), (p_1, -x), (p_3, +y), (p_4, +y)\rangle \tag{3.6}$$



$$AIM = \begin{array}{c|cccc|cccc|cccc|cccc} & \multicolumn{4}{c}{p_1} & \multicolumn{4}{c}{p_2} & \multicolumn{4}{c}{p_3} & \multicolumn{4}{c}{p_4} \\ & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y \\ p_1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ \cancel{p_2} & \cancel{0} & \cancel{1} & \cancel{1} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} \\ p_3 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \cancel{p_4} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} \end{array}$$

$$NBP = \begin{array}{c|cccc} & -x & +x & -y & +y \\ p_1 & 1 & 0 & 1 & 1 \\ p_3 & 0 & 1 & 1 & 1 \end{array}$$

Figure 3.7: Example of the NBP matrix, the modified puzzle and the reduced AIM matrix

3.2.2 Matrix-based algorithm overview

The main algorithm is Alg. 8. The main loop repeatedly computes and evaluates the disassembly sequences until the assembly is disassembled or I_{max} iterations are executed. Note that it is not guaranteed that this method will find a solution due to the heuristic approach, even if it exists.

In the beginning, AIM is computed using the previously mentioned process. The method `ComputeDisassemblySequence` is thoroughly described in Alg. 7 and in section 3.2.1. After computing the disassembly sequence DS, each part is moved along the disassembly direction. The part is considered as shifted after moving for a predefined distance without collision. Depending on reaching this distance, the part is considered shifted from the assembly or blocked. When the part is successfully shifted, the RRT Tree begins to grow from the part's position. The part is navigated towards the final position, and if reached, the part is considered finished, and the next part from DS is taken. Note that the finished parts have to be kept as obstacles for other still blocked parts because there might be unexpected collisions with them.

This two-phase movement helps to extract parts from narrow passages (first phase) and the RRT expansion helps to escape from more complex assemblies, where movements along the disassembly direction do not ensure successful extraction. The disassembly direction can be either straight-line movement or a specific angle rotation. The example of RRT expansion for a yellow part is in Fig. 3.10. It shows the whole process of RRT planning. The tree avoids configurations where the yellow part collides with blocked parts and finished parts (brown and salmon).

Algorithm 7: `ComputeDisassemblySequence`

```
Input: Assembly Interference Matrix $AIM$, Parts $P$
$DAIM \leftarrow$ CreateDAIM($AIM$);
$n \leftarrow$ number of parts;
$d_1 \leftarrow$ RandomDirection();
$\pi_1 \leftarrow$ Min($DAIM, d_1$);
$DS \leftarrow \{(d_1, \pi_1)\}$;
$F \leftarrow \{\pi_1\}$;
for $i = 2, \ldots, n$ do
    $DBP \leftarrow$ DisassembledBlockingParts($AIM$, $P$, $F$);
    $NBP \leftarrow$ NotDisassembledBlockingParts($AIM$, $P$, $F$);
    $(d_i, \pi_i) =$ Min($DBP + NPB$);
    $DS \cup (d_i, \pi_i)$;
    $F \cup \pi_i$;
return $DS$;
```

Algorithm 8: `Matrix-based heuristic algorithm`

```
Input: Parts $P$, Directions $D$, Parts final positions $F$
$n \leftarrow$ number of parts;
$m \leftarrow$ number of directions;
$Disassembled =$ false;
for $i = 1, \ldots, I_{max}$ do
    $AIM \leftarrow$ CreateAIM($P$);
    $DS \leftarrow$ ComputeDisassemblySequence($AIM, P$);
    $d \leftarrow$ size of DS;
    for $i = 1, \ldots, d$ do
        $Away \leftarrow$ false;
        $Away \leftarrow$ MoveAlongDisassemblyDirection($DS[i]$);
        if $Away$ then
            RRTExpandToFinalPositions($DS[i], F$);        // using RRT algorithm from
             section 2.4.3
            $n = n - 1$;
    if $n = 0$ then
        $Disassembled =$ true;
        break;
return $Disassembled$;
```

3.3 DRRRT planner

The third algorithm is based on the DRRRT algorithm (section 2.4.4) combined with the matrix heuristic algorithm described in section 3.2. The overall principle is to generate a disassembly sequence using the matrix heuristic. After, an attempt to disassemble the proposed parts one by one by constructing a free space skeleton and navigating the part along the skeleton towards the goal configuration is made. Until all parts are disassembled, or I_{max} iterations are executed, this whole process is iterated.

3.3.1 DRRRT planner overview

The main loop is depicted in Alg. 9. The main loop is iterated until I_{max} iterations are executed, or all parts are disassembled. The first part is the disassembly sequence computing using Alg. 7. Note that because of further use of a sampling-based algorithm, the disassembly directions are not further used. However, the more directions will be included in the disassembly sequence

Algorithm 9: `DRRRT planner with matrix-based heuristic`

```
Input: Environment $e$, Parts $P$, Directions $D$, Parts final positions $F$
$n \leftarrow$ number of parts;
$m \leftarrow$ number of directions;
for $i = 1, \dots, I_{max}$ do
    $AIM \leftarrow$ CreateAIM($P$);
    $DS \leftarrow$ ComputeDisassemblySequence($AIM, P$);          // method from Alg. 7
    $d \leftarrow$ size of DS;
    for $j = 1, \dots, d$ do
        $S_{free} \leftarrow$ComputeFreeSpaceSkeleton($e$, $q_{start}$, $q_{goal}$);     // method from Alg. 2
        $R \leftarrow$ InitialRegion($P, q_{start}$);          // first region for biased growing
        $\mathcal{T} \leftarrow$ InitializeRRTTree($T, q_{start}$);
        for $k = 1, \dots, I_{max}$ do
            Region-biasedRRTGrowth($\mathcal{T}, R, S_{free}$);          // method from Alg. 3
            if $near\ q_{goal}$ then
                MovePartToGoalPosition($P[j]$);
                break;
    if $all\ disassembled$ then
        break;
```

estimation, the more accurate movability of each part will be estimated. It means that if too few disassembly directions are considered, the matrix heuristic can determine the wrong part to be disassembled.

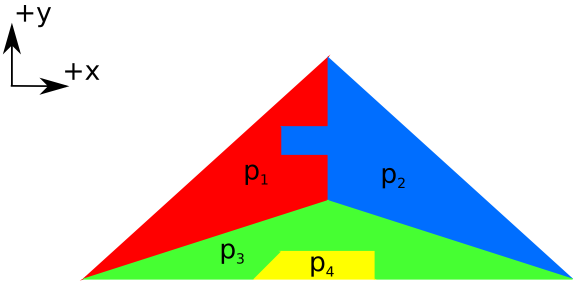
After computing the disassembly sequence, each part is attempted to disassemble. For each part, a free space containing holes symbolizing the remaining parts is formed and using the method `ComputeFreeSpaceSkeleton` in Alg. 2 the free space skeleton is computed. The part is advanced along the skeleton in the method `Region-biasedRRTGrowth` from Alg. 3 until reaching a goal or I_{max} iterations are executed.

3.4 Conclusion

In this chapter, three algorithms were introduced. The first, ML-RRT, borrowed from [9] is used as a comparison to the other algorithms. The first designed algorithm is the Matrix-based algorithm. It uses a matrix heuristic to compute the disassembly sequence of parts and corresponding disassembly directions. After that, a unique two-phase movement to extract the proposed parts in diassembly sequence is performed.

The second designed algorithm is the DRRRT planning algorithm. It uses the same heuristic as the Matrix-based algorithm, but only for specifying part to be disassembled. After this selection, a free space skeleton of the assembly called the Reeb graph is computed. Then it is used for a guided search of the RRT algorithm.

$$AIM = \begin{array}{c} \\ \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \overbrace{\qquad\qquad\qquad\qquad}^{p_1} \overbrace{\qquad\qquad\qquad\qquad}^{p_2} \overbrace{\qquad\qquad\qquad\qquad}^{p_3} \overbrace{\qquad\qquad\qquad\qquad}^{p_4} \\ \begin{array}{cccccccccccccccc} -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y & -x & +x & -y & +y \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

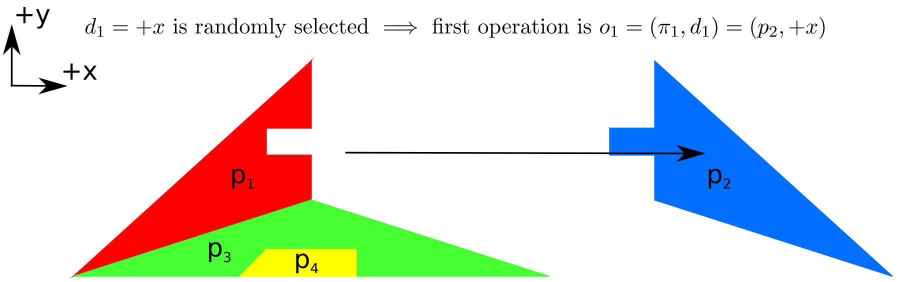


$$DAIM = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_1 & 0 & 3 & 3 & 1 \\ p_2 & 3 & 0 & 3 & 1 \\ p_3 & 2 & 2 & 1 & 2 \\ p_4 & 2 & 2 & 0 & 3 \end{array}$$

(a) : Created AIM and $DAIM$ for given puzzle

$$DAIM = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_1 & 0 & 3 & 3 & 1 \\ p_2 & 3 & 0 & 3 & 1 \\ p_3 & 2 & 2 & 1 & 2 \\ p_4 & 2 & 2 & 0 & 3 \end{array} \qquad \pi_1 = \begin{cases} p_1 & if \quad d = -x \\ p_2 & if \quad d = +x \\ p_4 & if \quad d = -y \\ p_1 \text{ or } p_2 & if \quad d = +y \end{cases}$$

$d_1 = +x$ is randomly selected $\implies$ first operation is $o_1 = (\pi_1, d_1) = (p_2, +x)$

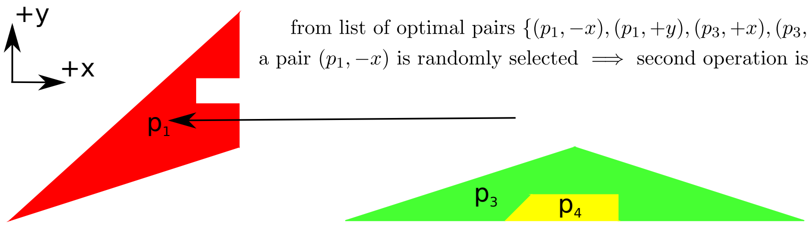


(b) : First part disassembling

$$DBP(p_2) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_1 & 1 & 0 & 1 & 1 \\ p_3 & 1 & 0 & 1 & 0 \\ p_4 & 1 & 0 & 1 & 0 \end{array}$$

$$NBP(p_2) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_1 & 0 & 2 & 2 & 0 \\ p_3 & 2 & 1 & 1 & 1 \\ p_4 & 2 & 1 & 0 & 2 \end{array} \qquad (DBP + NBP)(p_2) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_1 & 1 & 2 & 3 & 1 \\ p_3 & 3 & 1 & 2 & 1 \\ p_4 & 3 & 1 & 1 & 2 \end{array}$$

from list of optimal pairs $\{(p_1, -x), (p_1, +y), (p_3, +x), (p_3, +y), (p_4, +x), (p_4, -y)\}$
a pair $(p_1, -x)$ is randomly selected $\implies$ second operation is $o_2 = (\pi_2, d_2) = (p_1, -x)$

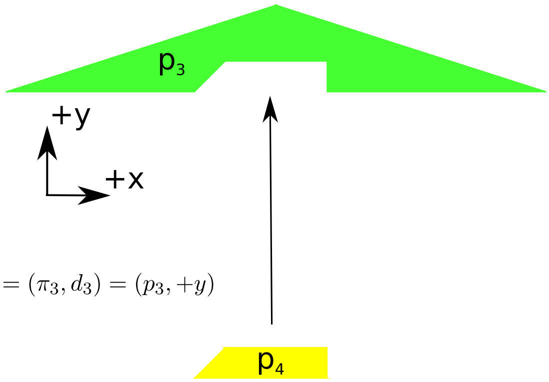


(c) : Second part disassembling

Figure 3.8: An example run of Matrix heuristic algorithm.

$$DBP(p_1,p_2) = \begin{array}{c} \\ p_3 \\ p_4 \end{array} \begin{array}{cccc} -x & +x & -y & +y \\ \end{array} \left[\begin{array}{cccc} 1 & 1 & 2 & 0 \\ 1 & 1 & 2 & 0 \end{array}\right] \quad (DBP+NBP)(p_1,p_2) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_3 & 2 & 2 & 3 & 0 \\ p_4 & 2 & 2 & 2 & 1 \end{array}$$

$$NBP(p_1,p_2) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_3 & 1 & 1 & 1 & 0 \\ p_4 & 1 & 1 & 0 & 1 \end{array}$$



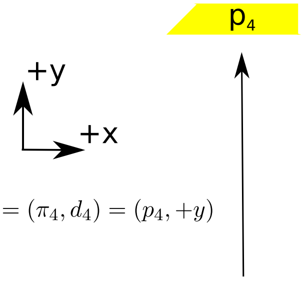
optimal pair $(p_3, +y) \implies$ third operation is $o_3 = (\pi_3, d_3) = (p_3, +y)$

(a) : Third part disassembling

$$DBP(p_1,p_2,p_3) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_4 & 2 & 2 & 3 & 0 \end{array}$$

$$NBP(p_1,p_2,p_3) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_4 & 0 & 0 & 0 & 0 \end{array}$$

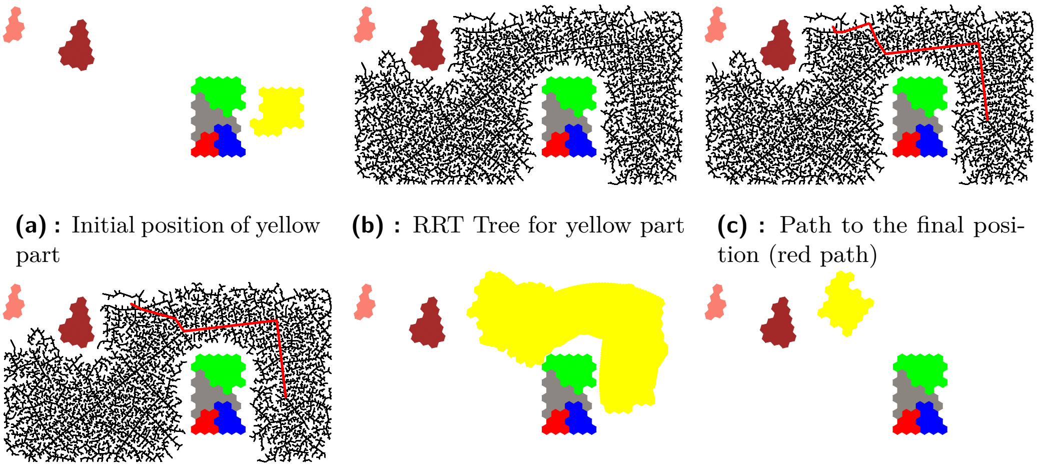
$$(DBP+NBP)(p_1,p_2,p_3) = \begin{array}{c|cccc} & -x & +x & -y & +y \\ \hline p_4 & 2 & 2 & 3 & 0 \end{array}$$



optimal pair $(p_4, +y) \implies$ fourth operation is $o_4 = (\pi_4, d_4) = (p_4, +y)$

(b) : Last part disassembling

Figure 3.9: An example run of Matrix heuristic algorithm.



(a) : Initial position of yellow part

(b) : RRT Tree for yellow part

(c) : Path to the final position (red path)

(d) : Pruned path to final position (red path)

(e) : Record of all configurations of yellow part along the path to final position

(f) : Final position of yellow part

Figure 3.10: An example run of the RRT algorithm for the yellow part.

Chapter 4

Experiments and testing

This chapter covers details about the testing scenarios, used libraries and software for algorithm implementation and performed experiments with the proposed disassembly algorithms.

4.1 Used libraries

The proposed algorithms rely on several libraries and software packages:

1. MPNN: Nearest-neighbor Library for Motion Planning [32] — used in all sampling-based parts of the algorithms for the nearest-neighbour search
2. RAPID: Robust and Accurate Polygon Interference Detection [25] — used for fast collision detection between triangulated objects
3. VTK: Visualization Toolkit [29] — contains libraries for creating the Reeb graph from a polygonal mesh
4. Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator [33] — used for triangulating 2D objects

4.2 Implementation details

All algorithms were implemented in C++. The methods for processing files and other file transfers were implemented in Python. The program was executed on Linux Mint on processor Intel® Core™ i7 1.8 GHz, and GNU gcc compiler 5.4.0.

All three algorithms were tested on several proposed problems and with different setting parameters. Each algorithm was executed 100 times on each scenario. The runtime of each test is measured, and the whole experiment is evaluated as a distribution function. The distribution function of a number of iterations x is a function f, where $f(x)$ is a value determining the probability that the algorithm will find a solution in less or equal than x iterations. Each trial is run until success or until maximal number of iterations are executed. The maximal number of iterations is considered differently for each algorithm.

For the ML-RRT algorithm, each part has up to 10^6 attempts to sample the active configuration space (the sampling in the passive sub-manifold does not count to the iterations). For the DRRRT algorithm, the strategy is to attempt to dismantle the assembly repeatedly. Therefore, when the heuristic algorithm proposes the component, the RRT phase has up to $5 \cdot 10^4$ iterations to build a tree towards the goal. The disassembly cycle (all parts were attempted to disassemble) is repeated up to 10^3 times. For the Matrix-based algorithm, the strategy is similar to DRRRT with a modification that after selecting a part by the heuristic and the locomotion along the disassembly direction, maximally 10^6 iterations are executed to find path to goal. The reason is that the Matrix-based algorithm requires relatively more iterations when exploring the configuration space compared to DRRRT, due to narrow passage problem. The DRRRT requires fewer iterations because of sampling along the Reeb graph.

The implemented algorithm performance depends on the parameters other than the number of iterations and the preferred direction in the Matrix-based algorithm. These parameters are:

1. Number of points per arc of the final Reeb Graph highly affects the speed of computing the Reeb graph but does not affect the path planning performance. The higher the number is, the more precise the Reeb graph is, but the computational time increases. The experiments showed that the ideal number of points is from 5 to 50, but 5 points are used for evaluation.

2. Radius of a region in DRRRT affects the number of iterations needed to find the goal successfully. The experiments showed that it is better to choose a greater radius to achieve the solution faster.

3. Density of points in DRRRT region also affects the number of iterations. The density with the highest performance was experimentally determined with at least 20 points per region.

4. Maximal surface of the triangle during triangulation influences the runtime of the triangulation and does not affect the performance of the planning algorithm. It affects the Reeb graph's computational speed, so the parameter is let as default and not used in the evaluation.

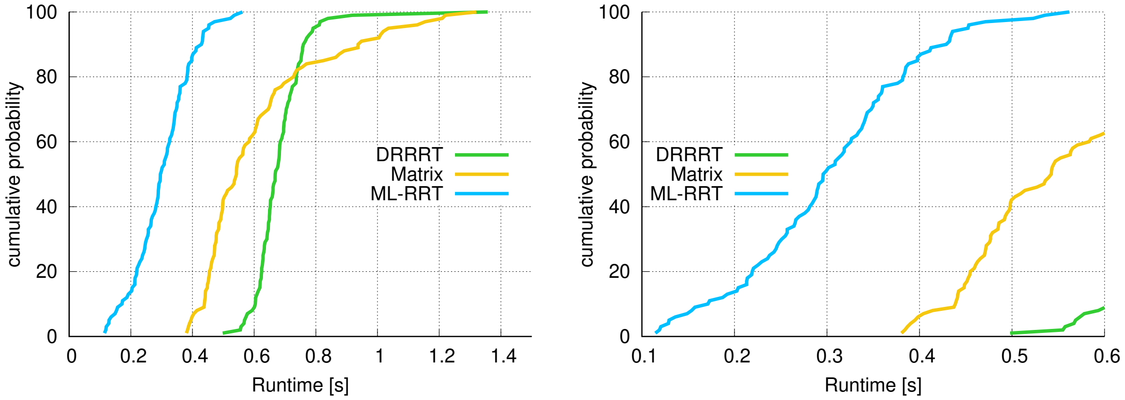
4.3 Proposed assembly problems

The experiments were performed on several scenarios that were prepared by hand for this thesis. Despite many existing publications related to the assembly and disassembly task, there is no publicly available library of benchmarks for 2D problems. It was supposed to create a wide diversity of benchmarks to determine the efficiency of each algorithm depending on a specific task. Each assembly is different in the number of parts, possible disassembly sequences, the number of obstacles (stable parts which cannot be moved), and other features described in section 2.3. Some of the assemblies were already shown in the thesis for explaining the algorithm functionality.

Each of the following benchmarks has been automatically generated using software to generate random hexagonal or square grid assemblies or designed in XFig software [34]. Some of them were inspired by a benchmark in an existing publication, referenced in the caption in Fig. 2.13.

4.4 Disassembly experiments evaluation

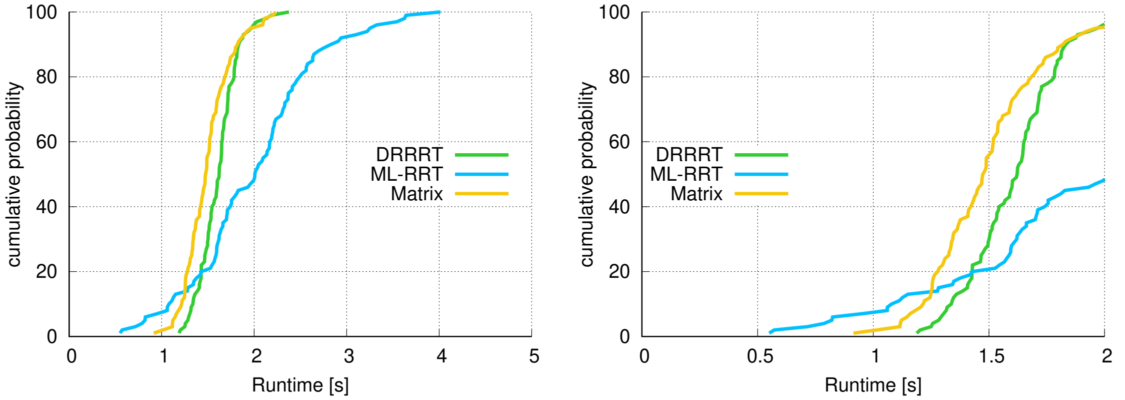
In this section the individual experiments are described in detail, each with a distribution function. Each experiment parameter, such as runtime and success rate are in Tab. 4.1 and Tab. 4.2. The runtime in seconds is determined by the average time and standard deviation separated by a vertical bar.



(a) : Evaluation of the Easy assembly for all three algorithms

(b) : Detail on the ML-RRT and the Matrix-based algorithm

Figure 4.1: Evaluation of the Easy assembly



(a) : Rectangular assembly for all three algorithms

(b) : Rectangular assembly zoomed on first 2 seconds

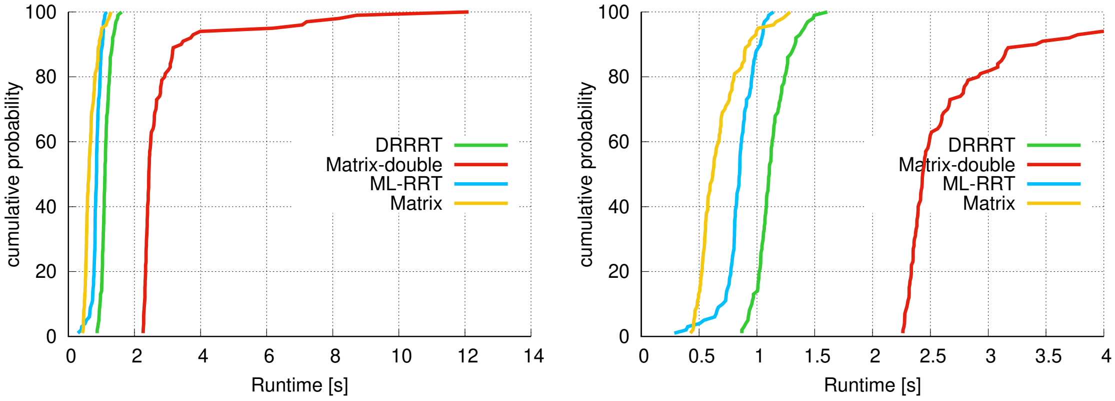
Figure 4.2: Evaluation of the Rectangular assembly

4.4.1 Easy assembly

The first problem is the easy four-part assembly shown in Fig. 2.13a. The evaluation depicted as a distribution function is shown in Fig. 4.1. The graph shows that all algorithms solved the problem comparably fast with a slightly better performance of ML-RRT. The DRRRT needed at least 0.5 seconds to solve the puzzle. All three algorithms succeeded the scenario in all tests.

4.4.2 Rectangular assembly

A similar problem to the previous one is the Rectangular assembly (Fig. 2.13b). The distribution shown in Fig. 4.2 is also similar to the one for the Easy assembly. Note that the time needed for disassembling has significantly increased, even though only two parts were added. For the ML-RRT, the average time (shown in table Tab. 4.1) has increased about six times in comparison with the Easy assembly. The DRRRT has similar performance to Matrix-based algorithm. The DRRRT has an average time only two times greater compared to the Easy assembly.



(a) : Tight assembly for all algorithms

(b) : Tight assembly detail on first 4 seconds

Figure 4.3: Evaluation of the Tight assembly

4.4.3 Tight assembly

The third assembly to evaluate is the Tight assembly (Fig. 2.13c). The first graph in Fig. 4.3 shows all three main algorithms. There is also a modification of the Matrix-based algorithm, where extraction of two parts simultaneously in the same direction is possible (Matrix-double). The algorithm modification computes the possible disassembly directions for each pair of parts. The number of parts + pairs of parts is then

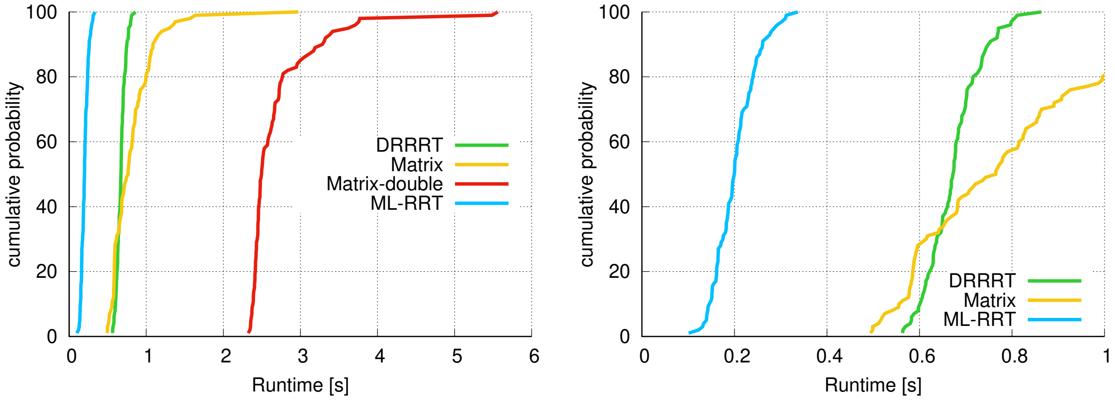
$$N_{all} = \binom{n}{2} + n. \tag{4.1}$$

For the Tight assembly, $N_{all} = 10$. That is the reason why this modification is slower than the single part version. More parts signify more exploring during AIM matrix creation.

In the Tight assembly scenario, the Matrix-based algorithm is in the lead. The reason is that ML-RRT has problems with narrow passages (and high-dimensional space, which is the issue shown in follow-up experiments). The probability of sampling a point directly in a disassembly direction is low, so more iterations (and resulting greater time complexity) are needed to disassemble the puzzle successfully. Unlike the Matrix-based, which firstly moves the part along the disassembly direction and then the sampling-based part quickly finishes the disassembly process. The DRRRT algorithm has a similar behaviour as in the Easy assembly, although about 10% of tests lasted more than 2 s. The DRRRT is slower because the Tight assembly was tested on a smaller workspace than the Easy assembly. Moreover, the obstacles are closer to each other, and it complicates sampling-based planning.

4.4.4 Triangle assembly

The following experiment is done for evaluating the Triangle assembly in Fig. 2.13d. The graphs in Fig. 4.4 show a more noticeable lead of ML-RRT algorithm compared to the Easy assembly (Fig. 4.1). Unlike the Tight assembly, the Triangle assembly does not have an overly fine geometry (parts are not wedged as much), and because of that, the parts have more space to move. Sampling-based algorithms require fewer iterations to complete the task. The Matrix-double algorithm confirmed its slower solving due to the higher number of considered parts (parts and their pairs). The average times (from Tab. 4.1) are comparably similar to the Easy assembly and, in the case of Matrix-based algorithm, also to Tight assembly (all three consist of four mobile parts).



(a) : Triangle assembly for all algorithms **(b) :** Triangle assembly zoomed on first 1 second

Figure 4.4: Evaluation of the Triangle assembly

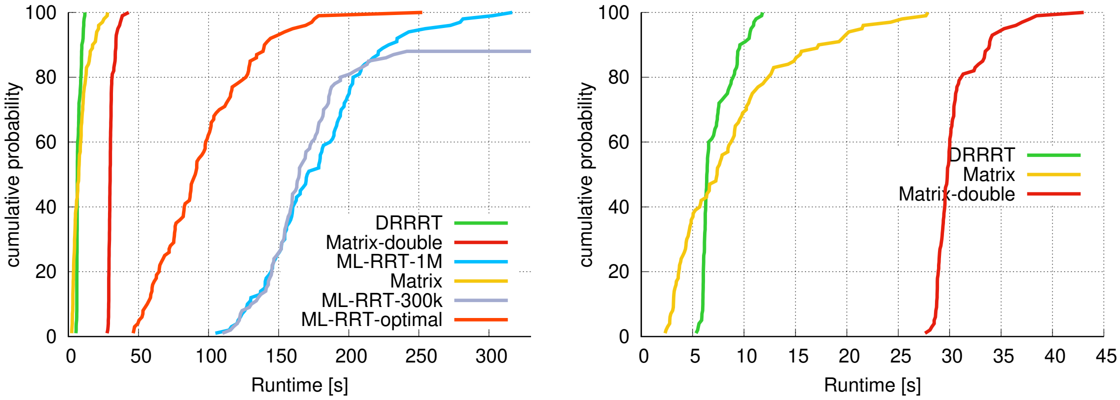
4.4.5 Hexa 7 assembly

The next experiment was done for the hexagonal seven-part puzzle shown in Fig. 2.13e. The optimal disassembly sequence is salmon, brown, yellow, green, grey, blue, and red. Firstly, the algorithms were executed for assembly with less space for manipulating the parts (black rectangle in Fig. 4.6). Considering less space was the reason why they were not able to disassemble the puzzle in most cases. After extending the workspace, the algorithms were tested again.

The distributions are shown in Fig. 4.5. The Matrix-based algorithm (single) is considerably faster than the Matrix-double modification depicted in Fig. 4.5b. The ML-RRT algorithm is tested thoroughly. The version ML-RRT-1M means that up to 10^6 attempts to sample the active sub-manifold for each part were made when considering randomly chosen sequence (specifically salmon, blue, green, yellow, red, brown and grey). This modification was successful in all tests. The modification ML-RRT-300k, which used only up to $300 \cdot 10^3$ iterations did not manage to disassemble 10% of tests, which is depicted in Fig. 4.5a at the end of ML-RRT-300k distribution function as the straight grey line. The reason is that after the first part (salmon), which was easily extracted, the second part (blue) had to be removed together with other (passive) parts. The limited mobility of the part led to a higher number of required iterations. Comparison can be given to the course of a version of ML-RRT called ML-RRT-optimal, which gets as input the optimal disassembly sequence of parts. The difference between these two courses is approximately 50 s. The last algorithm tested is the DRRRT algorithm, which has a better course than the Matrix-based algorithm in half of the cases and four times lower standard deviation (shown in Tab. 4.1).

4.4.6 Well assembly

Another experiment created for testing is the Well assembly shown in Fig. 2.13f. The ML-RRT algorithm was tested for several disassembly sequences of parts. The parts are numbered salmon — 1, blue — 2, green — 3 and yellow — 4. In the evaluation in Fig. 4.7 the first graph shows a comparison of all algorithms, the second shows a zoomed scope of the first 20 s and the third one the time range of the first 5 s. The fourth graph compares ML-RRT algorithm for various disassembly sequences. There is a significant difference in the performance of individual ML-RRT tests. The fastest appeared to be ML-RRT-4321, with the worst disassembly time of less than one second. Both ML-RRT-3421 and ML-RRT-3124 are the only other sequences



(a) : Hexa 7 assembly for all algorithms

(b) : Detail on Hexa 7 assembly

Figure 4.5: Evaluation of the Hexa 7 assembly

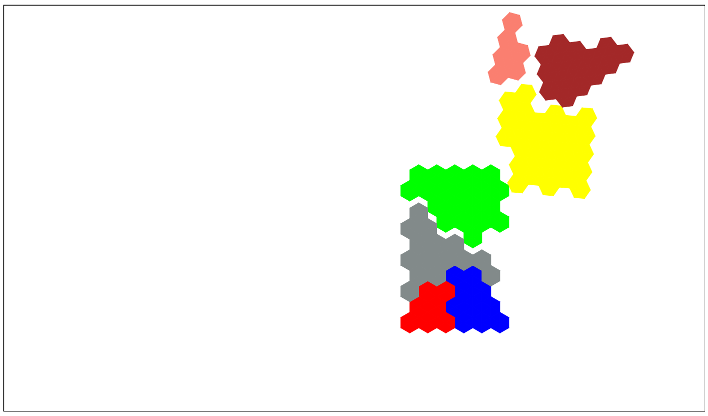
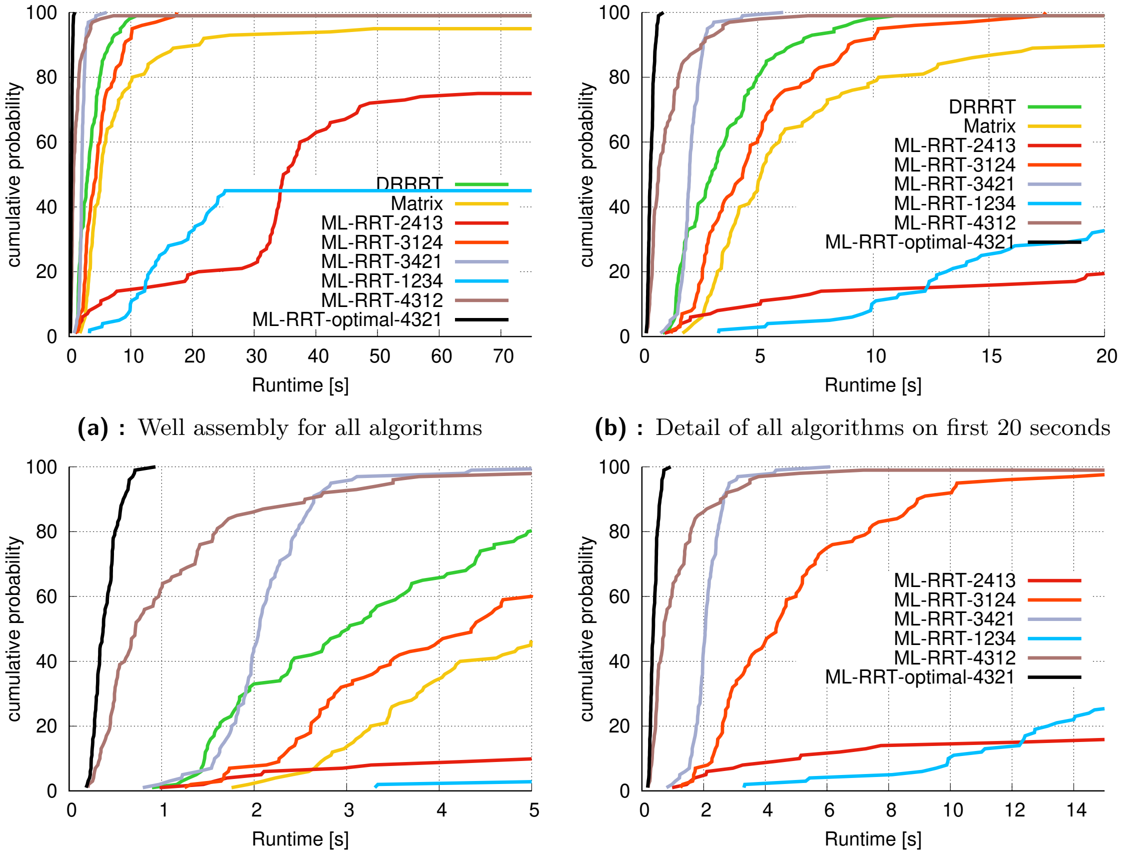


Figure 4.6: Unsuccessful disassembly of hexagonal assembly, green part is blocked

with a 100% success rate. The ML-RRT-4312 had one unsuccessful case, the ML-RRT-2413 had 75% success rate and the ML-RRT-1234 had just 45% success rate.

This outcome showed the importance of disassembly sequence planning and its impact on the probability of finding the solution, the iteration complexity of the problem, and the resulting time complexity. Note that the algorithms were stopped because the maximal number of iterations was reached (10^6), not because of the time complexity.

In the evaluation of the DRRRT algorithm, one test was not successful, and the overall performance was better than the Matrix-based algorithm and ML-RRT-3124. The Matrix-based algorithm appeared to be relatively fast compared to the ML-RRT-3124. The Well assembly is a non-monotone assembly because the yellow part has to be partly relocated to release the green part. It is also a 2-disassembly, especially for the Matrix-based algorithm. If supposing the disassembly sequence 3421 (heuristically computed by Matrix-based algorithm and supposing the yellow part is already relocated to the left), each part (excluding the last one) needs two movements for successful extraction. For example, the green part needs to be moved right and then upwards. Unfortunately, it is not possible to make two motions of the same part subsequently. The following motion can be suggested in the next iteration of the algorithm in another proposed disassembly sequence. The 95% success rate is acceptable, considering that Matrix-based algorithm was not designed for non-monotone and m-disassembly problems.



(a) : Well assembly for all algorithms

(b) : Detail of all algorithms on first 20 seconds

(c) : Detail of all algorithms on first 5 seconds

(d) : Only ML-RRT algorithm evaluation

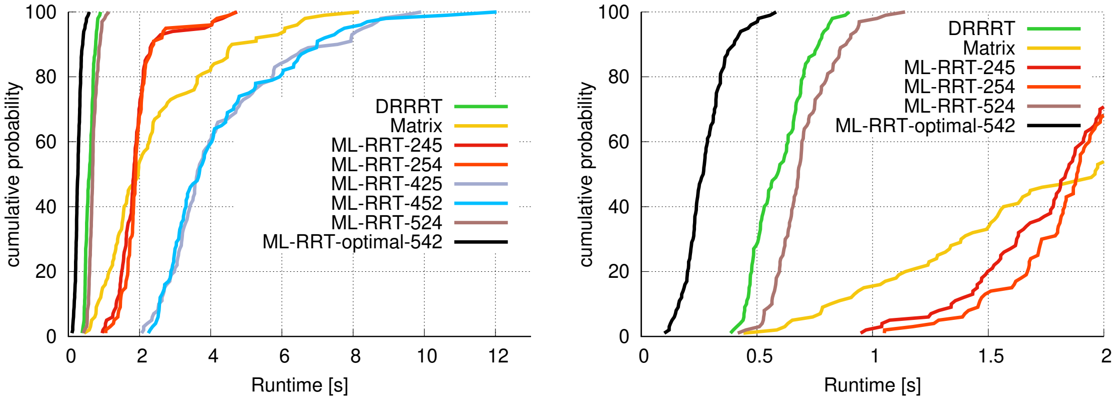
Figure 4.7: Evaluation of the Well assembly

4.4.7 Screw assembly

The Screw assembly, depicted in Fig. 2.13g is a puzzle, where each part except two static parts (blue and green) has to be extracted using at least two straight-line moves (2-disassembly). The parts are numbered as 1 — blue, 2 — salmon, 3 — green, 4 — yellow, 5 — red. The evaluation is shown in Fig. 4.8. The algorithms were tested similarly to the Well assembly, where various sequences of parts for ML-RRT were tested separately. The results of the distribution function and in Tab. 4.2 show that the difference between ML-RRT performance depending on the disassembly sequence, is enormous. The DRRRT and the Matrix-based algorithm have a 100% success rate, but DRRRT with a four-time better average time. The Matrix-based algorithm had a better performance than ML-RRT-425 and ML-RRT-452 and also in 50% of the tests better than ML-RRT-245 and ML-RRT-254. The DRRRT is the second best algorithm with similar performance to ML-RRT-524. It is possible that the heuristic, which selects parts for the DRRRT, chose a similar disassembly sequence to 524.

4.4.8 Bolt assembly

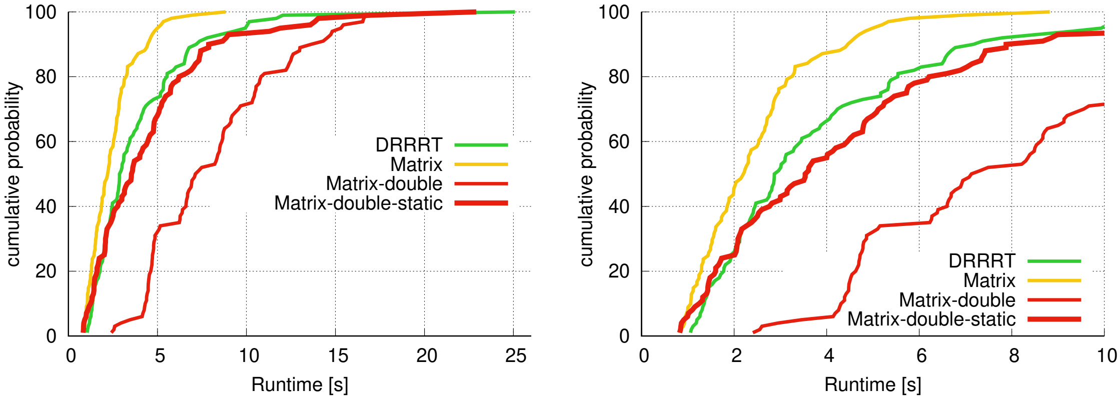
The Bolt assembly is classified as a non-monotone puzzle because the green part in Fig. 2.13h must be partially moved to extract the salmon part and then entirely removed. There are two possibilities to evaluate the puzzle. The first is that all parts can move, and the second is that the salmon part is considered a static obstacle (can not be relocated). In the second case, the assembly is non-linear because blue and red parts have to be extracted simultaneously. Note



(a) : Screw assembly for all algorithms

(b) : Detail of all algorithms on first 2 seconds

Figure 4.8: Evaluation of the Screw assembly



(a) : Comparison of all algorithms on Bolt assembly also with and without the salmon part as static

(b) : Detail on all algorithms

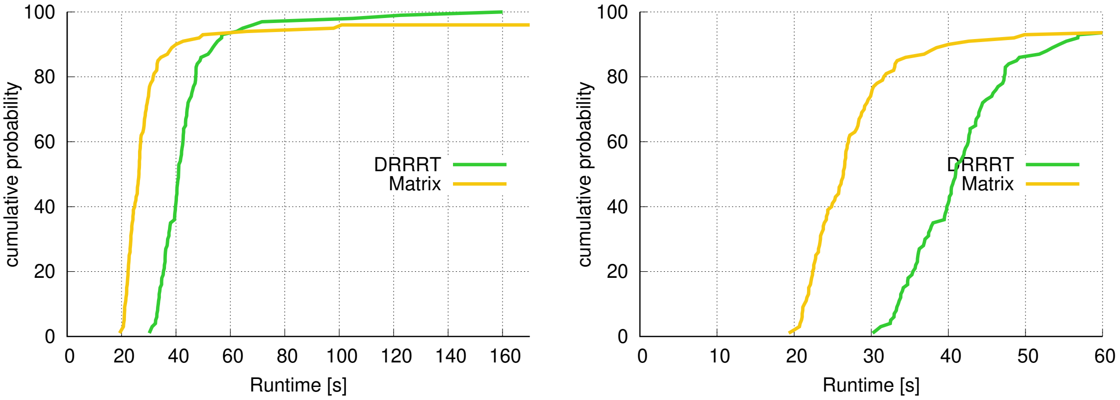
Figure 4.9: Evaluation of the Bolt assembly

that the assembly is still sequential because the disassembly operations are still two-handed. The evaluation of the first case was attempted for all proposed algorithms, as shown in Fig. 4.9, but only DRRRT and Matrix-based algorithms were able to solve the puzzle. ML-RRT did not manage to disassemble any sequence of parts. The Matrix-based had a 100% success rate, and the DRRRT did not manage to finish 13% of tests as is visible in Tab. 4.2. The reason is that the green part must be relocated very precisely to move the salmon part.

The second case (the salmon part is static) was possible to evaluate only for the Matrix-double algorithm (shown in Fig. 4.9 as Matrix-double-static). Interestingly, the performance is better than the scenario with all part mobile and considered doubles. The reason is that in the first case, the number of parts is $\binom{3}{2} + 3 = 6$ and in the second case $\binom{2}{2} + 2 = 3$. So the first case planning requires about two times more time to complete the task. From the average times for Matrix-double-static and Matrix-double in Tab. 4.2, the previously mentioned comparison corresponds to the measured results.

4.4.9 20 parts assembly

The 20 parts assembly in Fig. 2.13i was possible to evaluate only for Matrix-based algorithm and the DRRRT. After 25 minutes of disassembling, the ML-RRT managed to extract only



(a) : Comparison of Matrix-based and DRRRT algorithm on 20 parts assembly

(b) : Detail on first minute for 20 parts assembly

Figure 4.10: Evaluation of the 20 parts assembly

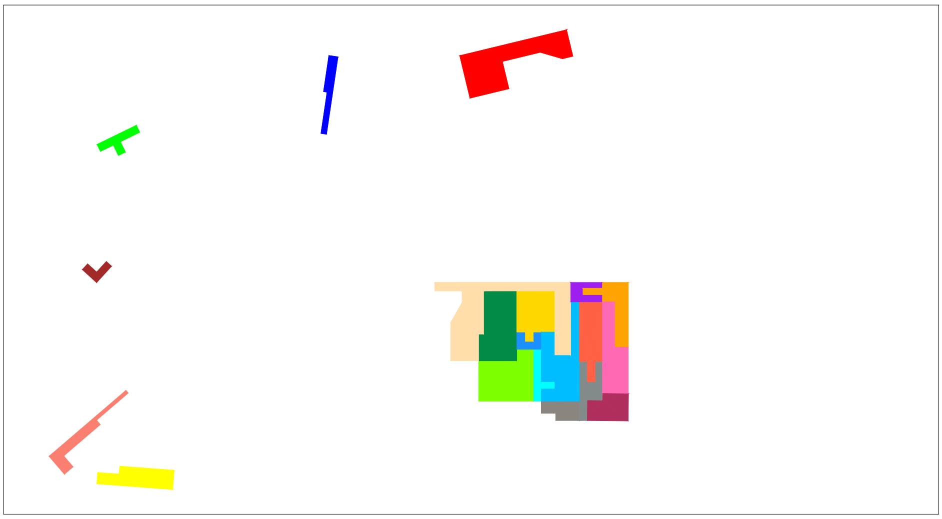
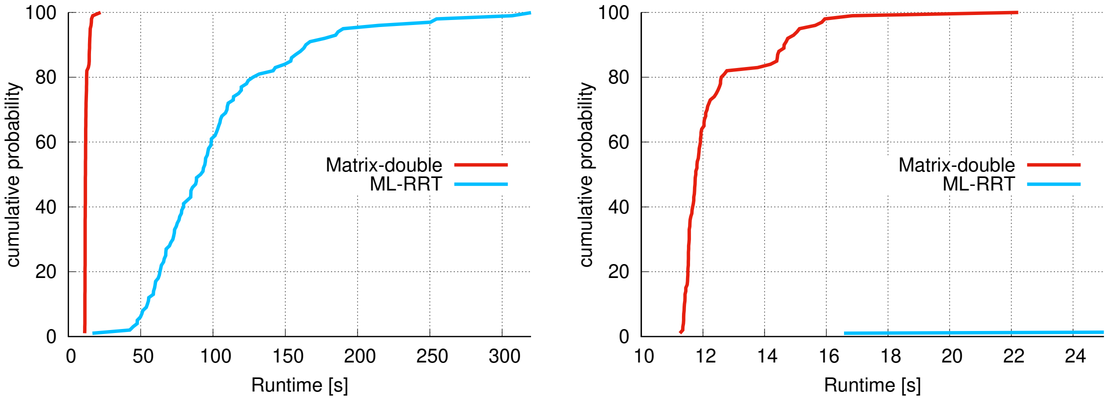


Figure 4.11: Unfinished solving the 20 parts assembly by the ML-RRT algorithm

six parts, as shown in Fig. 4.11. The DRRRT was not as fast as the Matrix-based algorithm, as shown in Fig. 4.10. Despite the reasonably fast runtime, the Matrix-based algorithm did not manage to disassemble the puzzle in 4 cases. It is not a creditable result considering that this type of assembly is considerably easier for straight-line parts extraction, not for sampling-based planning in the configuration space. The reason might be that there was not enough free space to put the disassembled parts or to manipulate the parts currently disassembled.

4.4.10 Jingjang assembly

The Jingjang assembly is shown in Fig. 2.13j. This puzzle is non-linear because the pair of black and white parts has to be removed together. The evaluation of the algorithm is depicted in Fig. 4.12. This puzzle was evaluated for ML-RRT and Matrix-double algorithm. The ML-RRT solved the puzzle by tiny repeating movements of each active part and relocating the blocking parts until the active part was finally released. ML-RRT is not a fast method to solve the puzzle, which is evident from the extreme time complexity compared to the Matrix-double.



(a) : Comparison of Matrix-double to ML-RRT algorithms on Jingjang assembly

(b) : Detail on Jingjang evaluation

Figure 4.12: Evaluation of the Jingjang assembly

There is an interesting situation visible in Fig. 4.12b between 13 and 14 second. The Matrix-double has a strange plateau in the distribution function. At this time, minimum of tests have ended. The reason is that the faster tests were solved with the first disassembly sequence in less than 12 seconds. The plateau is at the time where a new disassembly sequence is computed, and the next part of the curve are the tests solved by the algorithm in the second disassembly sequence.

4.4.11 Elevator assembly

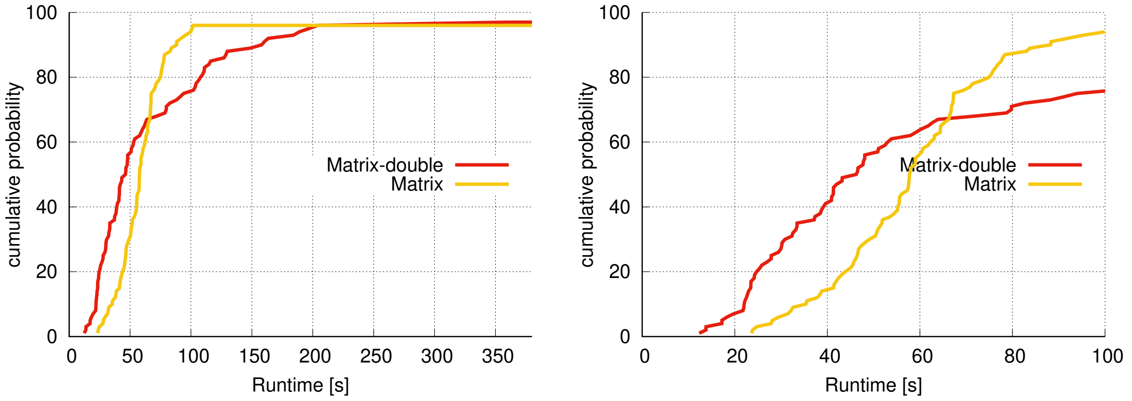
This puzzle (Fig. 2.13k) was only supposed to prove that the Matrix-based algorithm does not work well for m-disassembly and non-monotone puzzles because the disassembly plan of the Elevator assembly requires many partial relocations of parts. Note that the salmon part is static. The test evaluation is made in Fig. 4.13. The test showed a 96% success rate and an average disassembly time is 58 seconds, which is a great performance in comparison to other difficult assemblies, such as the 20 parts assembly. The Matrix-double algorithm performed better in almost 70% of tests, but the remaining 30% were by contrast much worse than the single part variant. The optimal disassembly plan of the assembly to comprehend the complexity of the puzzle is depicted in Fig. A.1.

4.4.12 Parallel assembly

The Parallel assembly depicted in Fig. 2.13l was disassembled only by the ML-RRT algorithm because all others were not able to find a solution. In the Parallel assembly, three parts (salmon, green and blue) have to be moved simultaneously, each part in a different direction. That means that it is a non-sequential puzzle (actually four-handed). The DRRRT and the Matrix-based algorithm are not designed for such type of scenarios. The evaluation is depicted in Fig. 4.14a and statistical evaluation in Tab. 4.2. Similarly, as the Jingjang assembly, the Parallel assembly is solved by ML-RRT by tiny relocations of parts leading to extraction of the active part.

4.4.13 Lock assembly

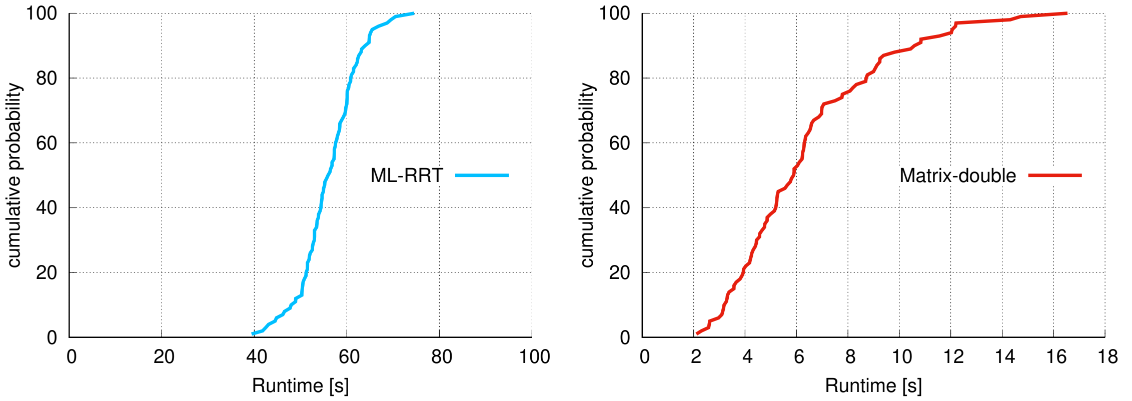
This assembly, shown in Fig. 2.13m, has a static blue part and two pairs of parts (salmon — yellow and green — red). The parts from the pair have to be disassembled simultaneously for



(a) : Elevator assembly for the Matrix-based algorithm

(b) : Detail of Elevator assembly for the Matrix-based algorithm

Figure 4.13: Evaluation of the Elevator assembly



(a) : Parallel assembly for the ML-RRT algorithm

(b) : Lock assembly evaluation

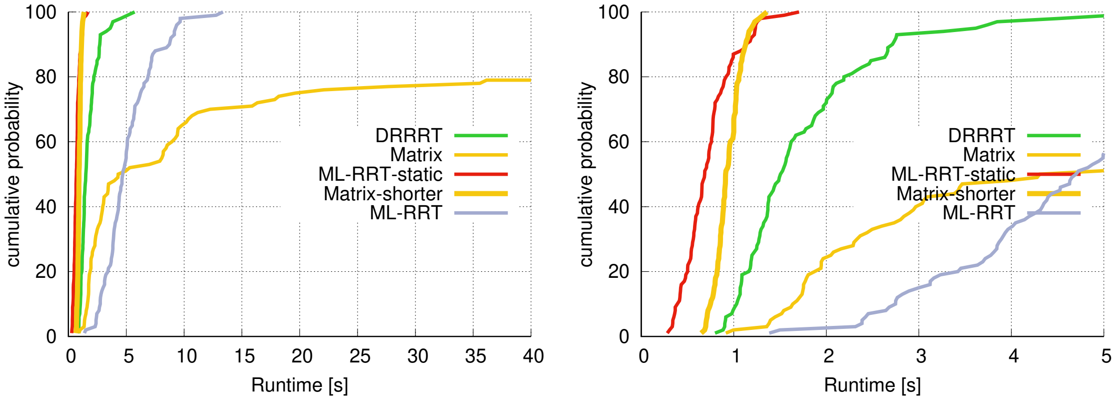
Figure 4.14: Evaluation of the Parallel and the Lock assembly

successful disassembling (it is a non-linear puzzle). The only algorithm able to disassemble the Lock assembly is Matrix-double algorithm. The ML-RRT was also attempted but was not successful even after extending the maximal number of iterations to $5 \cdot 10^6$. The evaluation is in Fig. 4.14b and in Tab. 4.2.

4.4.14 Bugtrap assembly

In the next experiment, the Bugtrap assembly in Fig. 2.13n was evaluated for the DRRRT, the ML-RRT and the Matrix-based algorithm. The Matrix-based algorithm can solve the puzzle not by the first phase of the movement (straight line locomotion along the disassembly direction) but by the second phase (the RRT-planning). The part to be removed is still moved along the disassembly direction, but this movement is stopped after a shorter distance d_{max} than during the evaluation of other algorithms. The distance d_{max} is critical to adjust. It can be seen from the evaluation in Fig. 4.15, where there is a course for d_{max} and shorter d_{max} (Matrix-shorter), that considering shorter d_{max} makes the first phase faster and the second phase may begin sooner. Also without shortening, the solution is not guaranteed (the 79% success rate of the Matrix-based algorithm).

The Bugtrap assembly was considered with and without the static blue part. In the



(a) : Bugtrap assembly for all algorithms **(b) :** Detail on the Bugtrap assembly evaluation

Figure 4.15: Evaluation of the Bugtrap assembly

Fig. 4.15, ML-RRT-static (blue static) achieved better performance than the DRRRT (also blue part static). The ML-RRT for a mobile blue part has worse performance than with a static blue part. The ML-RRT with static blue is better than other algorithms because when the part collides with the static one, the static does not have to be (can not be) relocated, and the planning of the active part can continue. When all parts are mobile, the blue part must be relocated after each collision with the active part until collision is detected. These relocations caused greater computational time demand. However, it did not change the success rate of the scenario.

4.4.15 Rotation assembly

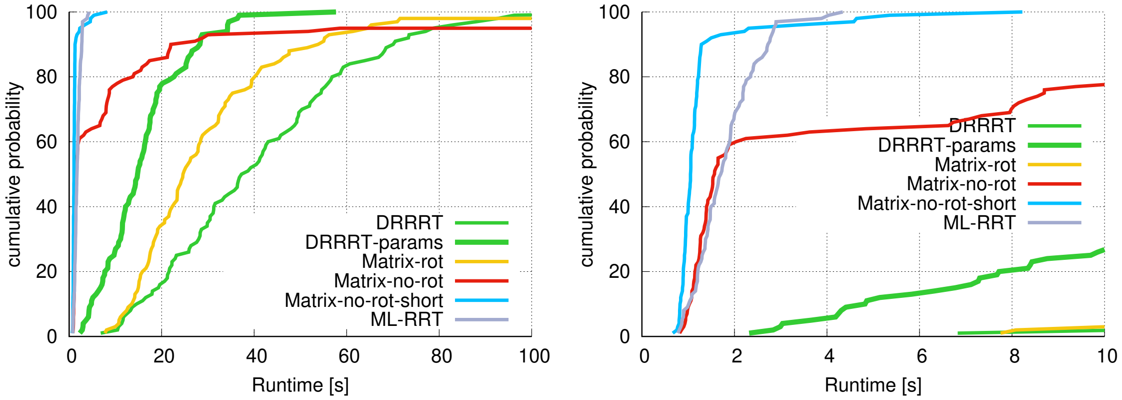
The Rotation assembly (Fig. 2.13o) was created to test rotations as the disassembly directions in the Matrix-based algorithm. As is visible from Fig. 4.16, the Matrix-based algorithm with considered rotations (Matrix-rot) is not as fast and not as successful as the Matrix-based algorithm without considered rotations and shorter distance during the first phase of the part movement (Matrix-no-rot-short). The DRRRT algorithm was first tested for the same settings as other experiments (DRRRT). After adjusting the parameters (greater radius of the DRRRT region and density of points in the DRRRT region), the performance improved (DRRRT-params). ML-RRT showed similar performance to Matrix-no-rot in about 60% of cases.

4.4.16 Hexa 10 assembly

This assembly (Fig. 2.13p) is an extension to the previously mentioned puzzle Hexa 7. It was not evaluated for the ML-RRT because, in the evaluation of the Hexa 7 assembly (Fig. 4.5), the performance was the worst even for the optimal disassembly sequence of parts. As well as at the Hexa 7 assembly, the DRRRT performed better than the Matrix-based algorithm.

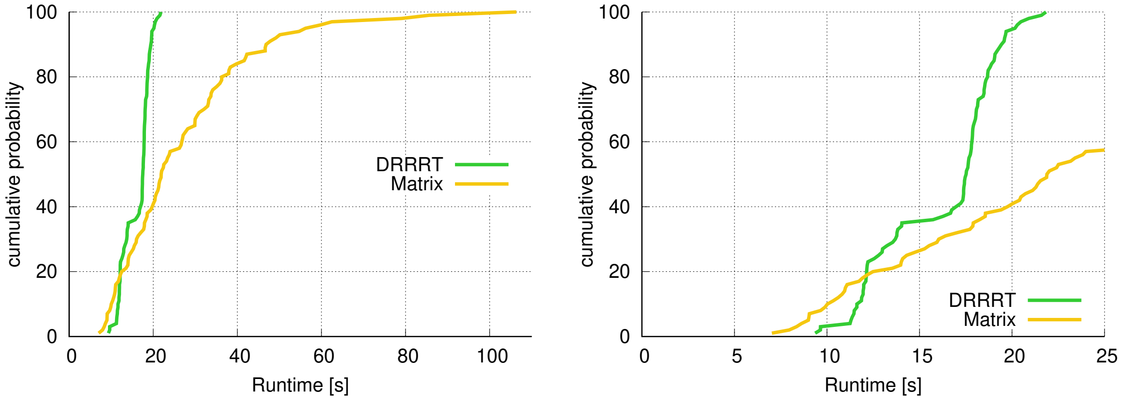
4.4.17 Chessboard assembly

This puzzle (Fig. 2.13q) was designed to test algorithms on an assembly with many parts and test the sampling of the high-dimensional space for the ML-RRT. The DRRRT and the Matrix-based algorithm had very similar performance, depicted in Fig. 4.18. ML-RRT had significantly worse average time than both of the previously mentioned algorithms. The



(a) : Rotation assembly for all algorithms **(b) :** Detail on the Rotation assembly evaluation

Figure 4.16: Evaluation of the Rotation assembly



(a) : Hexa 10 assembly for all algorithms **(b) :** Detail on Hexa 10 evaluation

Figure 4.17: Evaluation of the Hexa 10 assembly

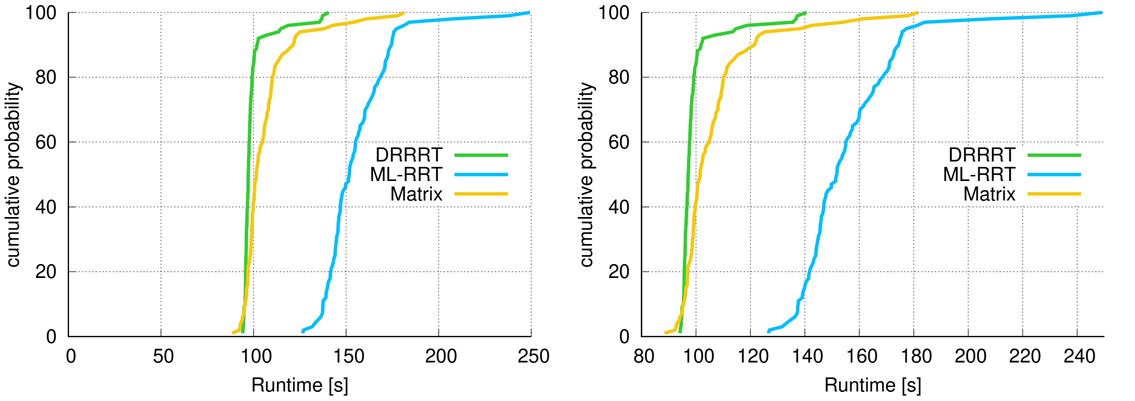
reason can be the non-optimality of the disassembly sequence or large configuration space (64 parts × 3 DOFs = 192 dimensions).

4.5 Comparison of the experiments

This section gives the overall results of all experiments and concludes the algorithm evaluation. It also compares the performance features of the ML-RRT algorithm in [9] and the evaluated results in this thesis.

For small (few parts) and coherent (parts far from each other) assemblies, the ML-RRT algorithm was the fastest. When the number of parts increased, the DRRRT began to be the best choice. Moreover, when the parts were more close together, the DRRRT or the Matrix-based algorithm had similar performance.

The ML-RRT algorithm does not have a tool to compute disassembly sequences. If the optimal sequence is selected, the ML-RRT is faster than other proposed algorithms (supposing the non-coherent puzzles). In comparison to the publication [9], where the ML-RRT is from, better results were achieved for the Well assembly (in the article called 2D Narrow puzzle), even when the salmon part is not considered as static (the 2D Narrow has three mobile and one static part). For the other tested puzzles, the authors confess that ML-RRT is not suitable



(a) : Chessboard assembly for all algorithms **(b) :** Detail on the Chessboard assembly evaluation

Figure 4.18: Evaluation of the Chessboard assembly

for coherent puzzles. They used the scaling factor for the parts as a trick to simplify the scenarios by increasing the space among the parts (in their case, the Diagonal Star Puzzle).

The algorithms DRRRT and Matrix-based can solve coherent (fine) puzzles, such as 20 parts assembly or hexagonal assemblies. Note that these assemblies are as fine as possible (they were designed to be tight and coherent). Only when they were created, the parts had to be scaled not to touch each other (collision detection detects touching parts as colliding). However, a significantly lower scaling factor was used. The authors used 0.98, 0.95 and 0.9, but in this thesis, 0.998 was used. This clearly explains the slow performance of the ML-RRT because disassembly planning for coherent assembly is more computationally demanding than for non-coherent assemblies.

The Matrix-based algorithm showed that it can solve non-monotone assemblies, such as the Bolt assembly or the challenging Elevator assembly. The Matrix-double modification was not as useful as supposed. Due to a higher number of parts (in the case of assemblies, such as 20 parts assembly and Chessboard assembly, the doubles were not even tested), the computational time for disassembly planning highly increases. When considering rotations as disassembly directions, the performance was significantly longer in comparison to Matrix-based algorithm without rotations and shortened first phase straight-line movement.

The DRRRT algorithm did well on all proposed assemblies (except non-monotone and non-linear), mainly coherent puzzles, such as hexagonal assemblies. The Reeb graph skeleton helped navigating the almost touching parts to their final positions. The worst performance was for the Rotation assembly, where the parameters of the DRRRT had to be changed to improve the performance. The performance was the worst, because the parts had to be specifically rotated to fit in the small hole in the surrounding box. There were not enough points generated nearby this narrow passage to successfully extract the part.

Puzzle	Method	Runtime [s]	Success rate [%]
Easy	DRRRT	0.68 | 0.1	100.0
	Matrix	0.61 | 0.21	100.0
	ML-RRT	0.3 | 0.09	100.0
Rectangle	DRRRT	1.61 | 0.21	100.0
	Matrix	1.5 | 0.25	100.0
	ML-RRT	2.0 | 0.69	100.0
Tight	DRRRT	1.13 | 0.14	100.0
	Matrix	0.67 | 0.19	100.0
	Matrix-double	2.91 | 1.47	100.0
	ML-RRT	0.85 | 0.14	100.0
Triangle	DRRRT	0.67 | 0.06	100.0
	Matrix	0.82 | 0.32	100.0
	Matrix-double	2.7 | 0.52	100.0
	ML-RRT	0.2 | 0.05	100.0
Hexa 7	DRRRT	7.27 | 1.64	100.0
	Matrix	8.78 | 5.98	100.0
	Matrix-doubles	30.56 | 2.48	100.0
	ML-RRT-1M	178.56 | 41.27	100.0
	ML-RRT-300k	163.57 | 26.11	88.0
	ML-RRT-optimal	96.48 | 35.59	100.0
Well	DRRRT	3.52 | 2.12	99.0
	Matrix	7.38 | 7.39	95.0
	ML-RRT-1234	15.25 | 6.08	45.0
	ML-RRT-2413	29.5 | 14.82	75.0
	ML-RRT-3124	5.13 | 3.27	100.0
	ML-RRT-3421	2.15 | 0.65	100.0
	ML-RRT-4312	1.13 | 1.09	99.0
	ML-RRT-4321	0.4 | 0.14	100.0

Table 4.1: Statistical evaluation of benchmarks. Runtime is shown in format avg | std. dev made out of 100 trials.

Puzzle	Method	Runtime [s]	Success rate [%]
Screw	DRRRT	0.6 | 0.12	100.0
	Matrix	2.47 | 1.68	100.0
	ML-RRT-245	1.94 | 0.68	100.0
	ML-RRT-254	1.99 | 0.62	100.0
	ML-RRT-425	4.32 | 1.83	100.0
	ML-RRT-452	4.31 | 1.89	100.0
	ML-RRT-524	0.7 | 0.13	100.0
	ML-RRT-542	0.28 | 0.1	100.0
Bolt	DRRRT	3.96 | 3.25	100.0
	Matrix	2.46 | 1.4	100.0
	Matrix-double	8.17 | 3.86	100.0
	Matrix-double-stat	4.48 | 3.69	100.0
20 parts	DRRRT	44.55 | 17.25	100.0
	Matrix	28.79 | 12.38	96.0
Jingjang	ML-RRT	103.04 | 53.01	100.0
	Matrix-double	12.41 | 1.6	100.0
Elevator	Matrix	58.02 | 17.4	96.0
	Matrix-double	64.81 | 54.82	97.0
Parallel	ML-RRT	56.14 | 6.37	100.0
Lock	Matrix-double	6.38 | 2.92	100.0
Hexa 10	DRRRT	16.18 | 3.15	100.0
	Matrix	26.91 | 17.15	100.0
Bugtrap	DRRRT	1.78 | 0.87	100.0
	Matrix	6.35 | 7.22	79.0
	Matrix-shorter-dist	0.94 | 0.15	100.0
	ML-RRT	5.19 | 2.23	100.0
	ML-RRT-static	0.72 | 0.28	100.0
Rotation	DRRRT	41.64 | 22.71	100.0
	DRRRT-params	15.95 | 9.29	100.0
	Matrix-rot	28.41 | 14.56	98.0
	Matrix-no-rot	6.48 | 10.06	95.0
	Matrix-no-rot-short	1.27 | 1.01	100.0
	ML-RRT	1.81 | 0.69	100.0
Chessboard	DRRRT	99.49 | 8.62	100.0
	Matrix	106.6 | 15.73	100.0
	ML-RRT	155.51 | 18.85	100.0

Table 4.2: Statistical evaluation of benchmarks. Runtime is shown in format avg | std. dev made out of 100 trials.

Chapter 5

Conclusion

The main goal of this thesis was to investigate the problem of disassembly planning and disassembly path planning, provide a thorough description of the problem nature and implement a suitable path planners to complete the disassembly task. Two algorithms were designed and implemented from scratch, and one (the ML-RRT algorithm) was adopted from [9] to be a comparison to the designed ones.

The first proposed algorithm, the Matrix-based algorithm, uses a heuristic method, adopted from [10], to compute the disassembly sequence of parts and disassembly directions (can be translation or rotation) and after a unique two-phase movement moves the parts from the disassembly sequence. The move consists of shifting along the disassembly direction and sampling-based planning. The advantage is the fast escaping of parts from the narrow passages and the ability to solve non-monotone and coherent assemblies. Apart from the thesis assignment, the Matrix-based algorithm was extended to consider simultaneous two part extraction (along the same disassembly direction, so the problem is non-linear) as the Matrix-based double algorithm.

The second algorithm is based on a fast sampling-based planner called DRRRT [8]. This algorithm heuristically computes disassembly sequence the same way as the Matrix-based algorithm. Then a free space skeleton of a free assembly workspace called Reeb graph is built, and guided search is used for creating the path for the selected part in the disassembly sequence. This algorithm improves the performance of the Matrix-based algorithm for non-coherent problems and surprisingly also for coherent problems, such as hexagonal assemblies.

The comparison of the algorithms with each other and also with the ML-RRT was executed on a wide variety of assemblies, described in section 4.4 and 4.5. The assemblies were partly adopted from available literature, but also many benchmarks were own designed. All considered benchmarks are depicted in Fig. 2.13.

The ML-RRT was very slow for coherent assemblies, and sometimes it even was not able to disassemble them. This algorithm does not have a disassembly sequence planner, such as the matrix heuristic in the Matrix-based algorithm, so it failed when a non-optimal disassembly sequence was considered. It also failed for non-monotone and non-linear problems, such as Bolt assembly and Lock assembly. It struggled with the high-dimensional assemblies, such as the Chessboard assembly, where it was significantly slower than the other designed algorithms. However, for simpler assemblies, the ML-RRT was the fastest method. Moreover, it was the only algorithm able to solve a non-sequential problem, the Parallel assembly.

The possibilities of further improvement of the proposed algorithms are to extend the methods to disassemble 3D problems. Also, the extension for non-sequential and non-linear assemblies for any number of simultaneously moving parts would be a goal the future research may aim to. Next, a more advanced disassembly sequence planner to plan a move of subassemblies and also put forward more direction possibilities, will be the further research.

Bibliography

[1] M. M. L. Chang, S. K. Ong, and A. Y. Nee, “Approaches and challenges in product disassembly planning for sustainability,” *Procedia CIRP*, vol. 60, pp. 506–511, 2017.

[2] F. Tao, L. Bi, Y. Zuo, and A. Nee, “Partial/parallel disassembly sequence planning for complex products,” *Journal of Manufacturing Science and Engineering*, vol. 140, no. 1, 2018.

[3] M. Santochi, G. Dini, and F. Failli, “Computer aided disassembly planning: state of the art and perspectives,” *CIRP Annals*, vol. 51, no. 2, pp. 507–529, 2002.

[4] J. Cortés, L. Jaillet, and T. Siméon, “Molecular disassembly with RRT-like algorithms,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3301–3306, IEEE, 2007.

[5] J. Cortés, D. T. Le, R. Iehl, and T. Siméon, “Simulating ligand-induced conformational changes in proteins using a mechanical disassembly method,” *Physical Chemistry Chemical Physics*, vol. 12, no. 29, pp. 8268–8276, 2010.

[6] L. Kavraki, J.-C. Latombe, and R. H. Wilson, “On the complexity of assembly partitioning,” *Information Processing Letters*, vol. 48, no. 5, pp. 229–235, 1993.

[7] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[8] J. Denny, R. Sandström, A. Bregger, and N. M. Amato, “Dynamic region-biased rapidly-exploring random trees,” in *Algorithmic Foundations of Robotics XII*, pp. 640–655, Springer, 2020.

[9] D. T. Le, J. Cortés, and T. Siméon, “A path planning approach to (dis)assembly sequencing,” in *2009 IEEE International Conference on Automation Science and Engineering*, pp. 286–291, IEEE, 2009.

[10] S. Ghandi and E. Masehian, “A breakout local search (BLS) method for solving the assembly sequence planning problem,” *Engineering Applications of Artificial Intelligence*, vol. 39, pp. 245–266, 2015.

[11] I. Aguinaga, D. Borro, and L. Matey, “Path-planning techniques for the simulation of disassembly tasks,” *Assembly Automation*, 2007.

[12] S. Ghandi and E. Masehian, “Review and taxonomies of assembly and disassembly path planning problems and approaches,” *Computer-Aided Design*, vol. 67, pp. 58–86, 2015.

[13] P. Jiménez, “Survey on assembly sequencing: a combinatorial and geometrical perspective,” *Journal of Intelligent Manufacturing*, vol. 24, no. 2, pp. 235–250, 2013.

[14] C. Becker and A. Scholl, “A survey on problems and methods in generalized assembly line balancing,” *European journal of operational research*, vol. 168, no. 3, pp. 694–715, 2006.

[15] A. J. Lambert, “Disassembly sequencing: a survey,” *International Journal of Production Research*, vol. 41, no. 16, pp. 3721–3759, 2003.

[16] J. Cortés, L. Jaillet, and T. Siméon, “Disassembly path planning for complex articulated objects,” *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 475–481, 2008.

[17] H. Srinivasan, N. Shyamsundar, and R. Gadh, “A framework for virtual disassembly analysis,” *Journal of Intelligent Manufacturing*, vol. 8, no. 4, pp. 277–295, 1997.

[18] R. H. Wilson, “On geometric assembly planning.,” tech. rep., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1992.

[19] D. Coleman, “Lee’s O (n2 log n) visibility graph algorithm implementation and analysis,” 2012.

[20] M. Garber and M. C. Lin, “Constraint-based motion planning using voronoi diagrams,” in *Algorithmic Foundations of Robotics V*, pp. 541–558, Springer, 2004.

[21] O. Takahashi and R. J. Schilling, “Motion planning in a plane using generalized voronoi diagrams,” *IEEE Transactions on robotics and automation*, vol. 5, no. 2, pp. 143–150, 1989.

[22] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, “Robust on-line computation of Reeb graphs: simplicity and speed,” in *ACM SIGGRAPH 2007 papers*, pp. 58–es, ACM Trans. Graph, 2007.

[23] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[24] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[25] S. Gottschalk, M. C. Lin, and D. Manocha, “RAPID: Robust and accurate polygon interference detection,” 1997.

[26] S. Gottschalk, M. C. Lin, and D. Manocha, “OBBTree: A hierarchical structure for rapid interference detection,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, 1996.

[27] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, “Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. x–xvi, 2018.

[28] J. R. Shewchuk, “Triangle: Engineering a 2D quality mesh generator and delaunay triangulator,” in *Workshop on Applied Computational Geometry*, pp. 203–222, Springer, 1996.

[29] W. J. Schroeder, L. S. Avila, and W. Hoffman, "Visualizing with VTK: a tutorial," *IEEE Computer graphics and applications*, vol. 20, no. 5, pp. 20–27, 2000.

[30] E. Masehian and S. Ghandi, "ASPPR: A new assembly sequence and path planner/replanner for monotone and nonmonotone assembly planning," *Computer-Aided Design*, p. 102828, 2020.

[31] R. H. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994.

[32] A. Yershova and S. M. LaValle, "MPNN: Nearest neighbor library for motion planning," 2006.

[33] J. R. Shewchuk, "Triangle a two-dimensional quality mesh generator and delaunay triangulator." `http://www.cs.cmu.edu/~quake/triangle.html`. Accessed: 2021-04-20.

[34] S. Sutanthavibul, B. V. Smith, and P. King, "FIG: Facility for interactive generation of figures," 1985.

Appendix A

Disassembly plan of the Elevator assembly

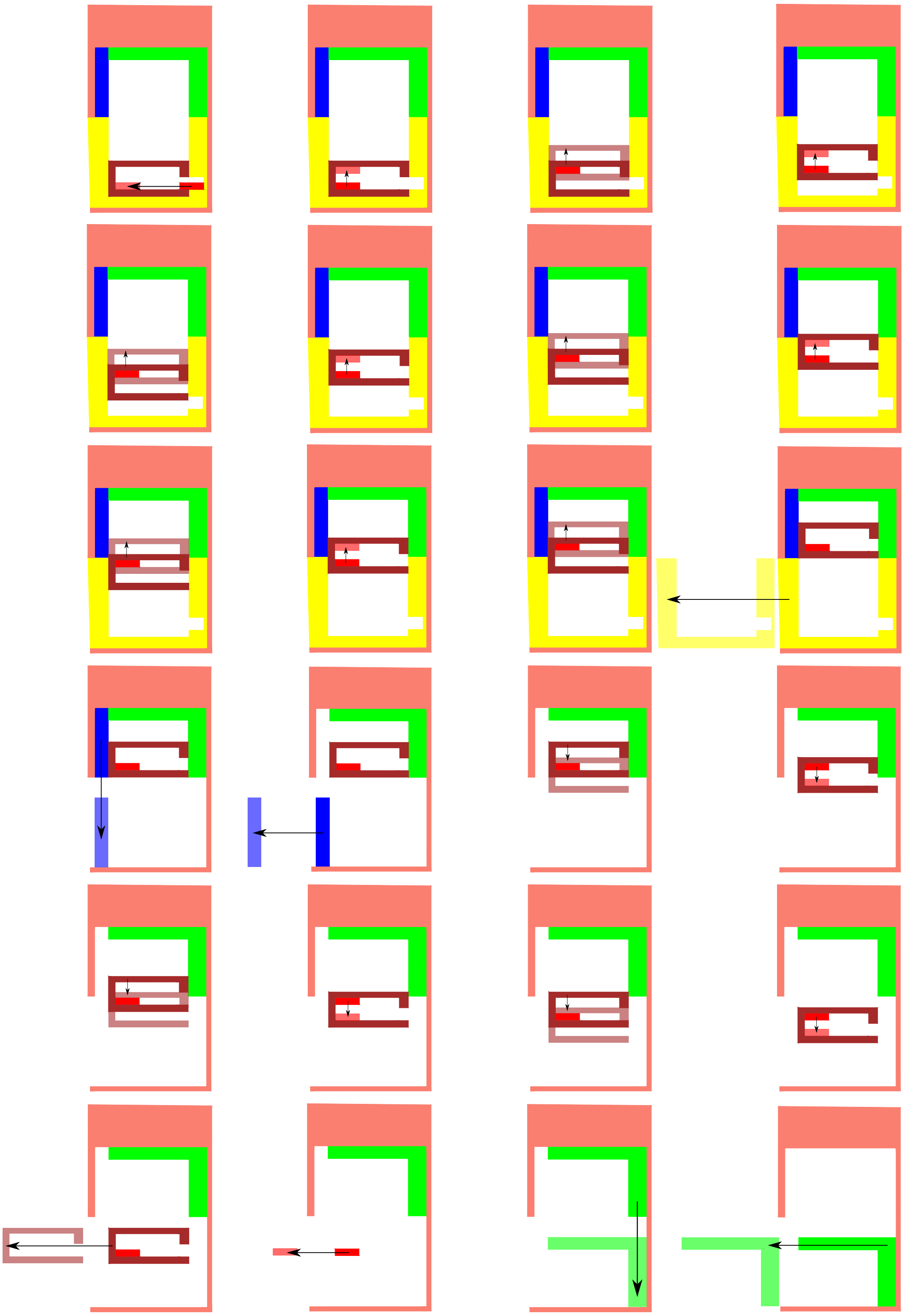


Figure A.1: Disassembly plan of the Elevator assembly. Read from left to right.