A GENERAL FRAMEWORK FOR A SAMPLING-BASED PLANNING APPROACH TO DISASSEMBLY SEQUENCE PLANNING

A Thesis

by

TIMOTHY EBINGER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Nancy M. Amato
Committee Members,	Dylan Shell
	Dezhen Song
	Suman Chakravorty
Head of Department,	Dilma Da Silva

December 2018

Major Subject: Computer Engineering

ABSTRACT

We present a framework for disassembly sequence planning. This framework is versatile, allowing different types of search schemes (exhaustive vs. preemptive), various part separation techniques, and the ability to group parts into subassemblies to improve solution efficiency and parallelism. This enables a truly hierarchical approach to disassembly sequence planning. Several implementations of the framework are in place, and perform better than existing state-of-the-art methods. Not only does the framework identify the necessary components for a successful disassembly planner, it will help in the analysis of problems and what kind of solvers are most effective in different scenarios.

The framework also supports the ability to plan for a manipulator while planning the disassembly sequence itself. Currently, this means planning for a simple manipulator arm to carry out the disassembly plan generated by the framework. This is done by first placing and validating the manipulator's end effector along all of the motions of each subassembly removed from the assembly, and then finding valid placements for the rest of the manipulator. The motions of the manipulator itself are left for future work, though valid placements for the manipulator are still found as a final step.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Nancy M. Amato [advisor] as well as Dylan Shell and Dezhen Song of the Department of Computer Science and Engineering and Suman Chakravorty of the Department of Aerospace Engineering.

Much of the supporting code base used was written by various members of Parasol laboratories. Dr. Shawna Thomas provided tremendous theoretical and analytical help every step of the way. Read Sandström provided support for any code issues, and helped ensure the code base used was up to date and ready to be integrated for every purpose for which it was needed. The published portion of this work [1] was done with equal contributions by Sascha Kaden from the University of Chemnitz, and by the student.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a Diversity fellowship from the Office of Graduate and Professional Studies of Texas A&M University and research funding from Professor Nancy M. Amato.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Disassembly Sequence Planning (DSP) identifies physically viable plans to disassemble an assembly of parts. DSP is often used in end-of-life product design to verify the future ability to disassemble the product for recycling or repairs [3]. DSP can also be applied to Assembly Sequence Planning (ASP) using the concept of assembly-by-disassembly [4] where disassembly sequences can be reversed to assemble the product. This field is crucial towards a more automated product design and manufacturing process. This problem is broken down into a search in disassembly sequence space, which is the space of all discrete assembly states, where each state represents a unique list of remaining/removed parts.

There are many different problems that can be addressed by an effective disassembly planner, particularly one that is incorporating motion planning as developed in this work. When trying to design a new product, the complexity of motions required to assemble/disassemble the 3D model can be taken into account to assess the ease of manufacture or recyclability for the product. It could also be that there is a common point of failure on a system, and so to change out the broken part can be a common occurrence. In this case, even from the initial design stage the designer can take into account accessibility of the part and can save a lot of effort and cost down the road by addressing this ahead of time. Finally, pick-and-place tasks using manipulators (where a set of objects with much clutter present must be moved elsewhere) often struggle with figuring out which objects to move, and in which order to do so. Modeling that problem as a disassembly problem will automatically identify this, and can give the motions that the manipulator can follow in order to carry out the motion plan itself.

Motion Planning is the task of finding a valid, collision-free motion of a robot (car, drone, vacuum, manipulator arm, etc.) from one position, the exact specification of which is called a configuration, to another specified configuration. This problem is extremely challenging, especially as robots typically have high degrees of freedom (DoF) which exponentially increases the difficulty of problems as the number of degrees increase. Configuration space [5] is the continuous space

in which plans are desired, as it spans the dimensionality of all DoF of a robot (which can be an assembly in this context). To help avoid this issue, Sampling-Based Motion Planning (SBMP) [6] [7] is founded on the idea of addressing extremely challenging aspects of these problems by utilizing *points* (samples) in a high-dimensional space instead of entire continuous subsets of the space.

Utilizing samples reduces the complexity of extremely hard problems, like those involving complex environments or robots with high DoF. Typically randomness is used to generate sample configurations in configuration space, which is the space of all possible configurations of a certain robot, whether valid or invalid. The space is navigated using points by attempting to connect configurations using some *local planning* technique, which in the most basic case will fit a straight line between two configurations and use a fine-enough sample resolution between them to verify that there is a valid path between the two configurations. A roadmap is a graph where nodes represent valid samples (configurations) of a robot, which is an assembly is this context. Each edge in a roadmap represents a valid local plan between two configurations. Once the starting point and the goal point can be connected to the roadmap, the problem is solved and the configuration space path is returned as the result.

In this research we propose a framework for solving disassembly sequence planning problems in a general way. A key point is that the contribution of this is more than just a solution to a subset or type of disassembly problems. Identifying an effective framework, and the key modules that a framework for solving these problems needs, provides a strategy for developing algorithms in the future that can be adapted to specific problems or types of problems, providing the highest level of generalization possible in a solution. Not only are motions for the parts found, but investigations into having a manipulator and its end effector are performed and analyzed, which use this framework. This means the framework is able to be used as an end-to-end solution for manipulator planning problems, like the aforementioned pick-and-place tasks, among other problems that currently prove difficult to solve.

The framework separates how we are solving problems into three distinct hierarchical cate-

gories:

1. The disassembly sequence
2. The manipulation plan
3. The motion plan

This creates a natural hierarchy that is analogous to Task and Motion Planning frameworks, where a higher order task (comparable to a disassembly sequence) is solved by breaking it up into motion planning problem(s), the solution of which represent completion of a task. Our DSP framework's goal is to find a disassembly sequence—the highest level of abstraction of the problem—by breaking the problem into sub-problems which require motion planning to solve. The actual sub-problems that are attempted are strategically chosen by the implementation of the framework being used. Manipulation planning could be considered alongside the motion planning aspect, as it is simply a more constrained and difficult motion planning problem. However, the motion planning done for sequence generation and that done for manipulator motion can also be performed disjointly from one another, depending on how the framework is employed.

There are cases where the motion plan is trivial in comparison to the information found in the sequence. This would occur in cases of large assemblies that can be simply disassembled, but have a very specific order with which disassembly can occur, like if parts are blocking the motion of other parts. The inverse can also be true, where the sequence planned for is trivial in comparison to the motions that are required. This occurs when we look at assemblies where few parts need to be removed, or those that need complex removal paths. In the general case, however, both the sequence and the motion plan provide the most informational picture of the problem being solved. Both in conjunction provide a maximally useful solution of the problem, at both a high level and a lower level of data. The manipulation planning aspect is also important and is identified as a separate aspect because in many ways it can be considered its own problem. A solver is made far more powerful and realistic by considering a manipulator in every step of the solution, but to strictly find part motions or sequences, it is not required.

By establishing, implementing, and demonstrating several different strategies using this framework, and one day validating the plans made on a manipulator arm, this research pushes the field towards a more automatable future with proven desirable results. This framework lays down a foundation for how to solve these problems in the future, and introduces a plug-and-play concept, where changing out any given module can have major effects on the overall exploration of the disassembly sequence space. Finally, it is beyond the scope of this work, but this framework will also allow a very high-level abstraction that will potentially have tremendous effects on the adaptability of robots to tasks they were not specifically designed for. For instance, by allowing a meta-algorithm access to the underlying module selection, we can create a self-redesigning algorithm that learns how to solve difficult problems in ways that are increasingly optimal (and might not even be obvious to a human operator, as solutions to motion planning problems often are not). This framework provides the flexibility to solve problems in this way, and makes clear the different modules that could be chosen and rearranged.

1.1 Contributions

1. A framework for autonomously solving disassembly sequence problems.
2. Implementations of the framework which serve as novel DSP methods.
   (a) A method to find any valid solution as quickly as possible (Preemptive DFS).
   (b) A method to find *all* solutions that are possible (Full BFS).
   (c) A method employing A* search in disassembly sequence space to find an optimal solution as quickly as possible. While this method is implemented, it has not been studied much yet, which is left for future work.
3. Generation of placements for a manipulator arm along with motions for its end effector to carry out the disassembly plan in simulation.

Some of the results in this work have been published in the 2018 IEEE International Conference on Robotics (ICRA) [1] and are used with permission. This paper presented the general framework

for our method, along with the Preemptive DFS and Full BFS implementations of it. This work also included verification and comparison to other existing methods. These items are © 2018 IEEE. Note that some excerpts from that work have been used in this thesis verbatim.

1.2 Outline

This thesis will first highlight related work, including both in motion planning and in disassembly sequence planning. Next the approach taken by the framework will be identified, providing a detailing of the framework, as well as the algorithms that were developed using it by choosing various modules. Then the work not yet published will be discussed in talking about manipulator planning and a very brief description of A* using this framework. Finally, results and the overall outcomes of this research will be discussed.

2. RELATED WORK

This chapter discusses work that has been done by others related to the field of disassembly planning. Broadly speaking, this includes motion-based planning, optimal planning, and other frameworks that have been developed. This is a continuously growing field, with a wealth of previous research devoted to it. [1]

Both assembly and disassembly planning have been studied extensively over the past few decades. Planners exist which solve some of the problems we are trying to solve, or be used in different ways. For instance, many that use motion planning only attempt to remove a single part, or attempt to remove single parts until disassembly is achieved. These methods can handle a lot of problems, but it is easy to find assemblies that will be unsolvable using these methods. These are discussed in section 2.1.

Other methods have approached the problem quite differently. Some use concepts from the field of artificial intelligence, namely learning-based algorithms. Some commonly touched on are genetic algorithms and branch and bound techniques for forming sequences. However these typically are purely for the sequence generation step, given the relationships of parts (like which parts block or are near other parts). Further, it is common in the methods that the goal is to find an optimal sequence; which is limiting in scope for what our framework is intending. Our framework can be easily configured to find optimal sequences, use learning-based methods, have different sequence-searching strategies, or even different ways to verify/modify existing sequences. Learning-based and optimal methods are discussed further in section 2.2.

Beyond this, some frameworks have been investigated, but they have used some fundamentally different approaches which can limit their generality. Some have different goals in general, like providing automated analysis tools for the design phase of new products. This is also something our framework is meant to assist in, but not what we designed the framework around entirely. To the best of our knowledge, no other solvers exist that can solve such a diverse set of problems

[1] This chapter contains previously published material from [1].

like ours can. On top of that, we have seen no other frameworks that can be used to design such a diverse set of solvers. The solvers can vary based on the problem to be solved and the type of solution that is wanted: whether an optimal solution is needed, any solution will suffice, or a general investigation on the different solutions possible is desired. Other assembly-related frameworks are discussed in section 2.3

Overall, other methods fall into two categories: those that would fit under our framework as implementations of it, or other frameworks that are approaching the problem from a different aspect. Many that fall in the first category, particularly the learning-based methods, would not even be full implementations of the framework, but rather only one module of it: the disassembly sequence search. Other frameworks are limited in scope as compared to ours; they are either limited to optimal sequences or have to perform similar computations whether after optimal or non-optimal sequences.

2.1 Motion Planning Methods for DSP

Many DSP methods focus on removing a single part, as needed in product repair applications. Targetless RRT (T-RRT) [8] iteratively removes exterior parts using an approach similar to mating vectors until the target part is removed. The work in [9] emphasizes a cost optimization in the context of T-RRT planning. D-plan [10] uses a retraction method to generate samples in narrow passages, (a common scenario in DSP where parts are in close proximity) and constrained motion interpolation to connect samples together. As these methods focus on single part extraction, they do not efficiently extend to full disassembly problems, particularly those that require coordinated part motions.

Often methods struggle when there are subassemblies that must be moved together to find a solution. Most methods either ignore planning subassemblies or handle them implicitly by allowing the movement of one part to produce a responsive movement in another. For example in [11], Thanh Le et al. develops an iterative planner (I-ML-RRT) based on ML-RRT [12] that classifies parts as either *active* and *passive* during a separation attempt. Explicit RRT planning is only done for the active part, but if it is found to collide with any passive parts, then a subset of the passive

parts will be perturbed in an effort to clear removal paths. Even though only single parts are ever moved, the allowed perturbation of other parts upon collisions helps to enable this planner to solve some problems requiring semi-coordinated part motions, though solving times are longer proportionally to how much clearance each part has to move individually. This method (and thus also the less general single-part strategies discussed in the last paragraph) is an approach that could be implemented using our framework, which we did in order to compare this method against our own novel implementations of our framework. This is discussed further in section 4.1.

2.2 Learning-based Methods and Optimality

Many learning-based methods have been applied to DSP including petri nets [13, 14], genetic algorithms [15, 16, 17], particle swarm optimization [18], and reinforcement learning [19]. These are typically not evaluated on realistic assemblies, do not consider subassemblies, and tend to focus on evaluating the feasibility of applying these techniques to DSP. Optimal planning, where the shortest sequence possible is required, has been studied in [13, 19, 20, 21]. These investigations could all be applied into implementations of our framework, which could then boost performance of methods using them. Simply put, these are limited in scope to what our framework accomplishes, and optimal methods or methods that use learning-based components could be developed using the prior research from this existing research, which would be excellent to consider for future work.

In recent work [22], Costa et al. describe a system which operates as a plug-in for Solidworks, a popular 3D modeling software. Their goal is to find an optimal sequence, and visualizing that that sequence as an exploded view of the assembly's parts in Solidworks. Evaluation is somewhat difficult for this method as only a few assemblies were experimented on. Some of these experiments still took well over a day of computation time in order to solve, as well. This method appears to be comparable to our Full BFS technique, but employing biasing techniques in the sequence space to find only an optimal solution, while saving some search time by not investigating all of the non-optimal solutions. This method would also fit under our framework and could be reworked as an implementation of it.

2.3 Frameworks Related to Assembly Planning

There have been some investigations into frameworks for Assembly Sequence Planning (ASP). These typically are trying to identify the aspects needed by a planner in general, instead of also considering building a versatile, functional planner with the framework. Further, other aspects are different which can negatively impact the generality of the frameworks, like only considering planning for design applications or needing to completely build data structures like a nondirectional blocking graph (NDBG) in order to create a sequence at all. It is worth noting that ASP and DSP are very similar problems, and problem instances where sequences from one would not be able to be used for the other are not covered in this work, these cases are generally if part breakages are allowed and is left for future work.

In the work [23] by Demoly et al., they describe a framework for Assembly Oriented Design (AOD) which is looking at using *aspects* of ASP in order to help guide the design of products. This is relevant to what we are studying in our framework, though our framework has a more general takeaway. As such, AOD is an aspect for which our framework is also designed to be used, but without being limited to that. Further, our framework is designed around the sequencing aspect with a specific motion planning emphasis.

Da Xu et al. in [24] identify a system for assembly planning with a similar goal as ours, but has some key differences. Namely, they look at the ASP problem disjointly from the motion planning problem, while we are planning our disassembly sequence by using motion plans as the disassembly graph is built. They first form a sequence, and then plan an actual path that could be used to carry out the sequence. There is also more of an emphasis on refining and optimizing existing ASP plans than on generating them using novel methods.

A work [25] by Maropoulos et al. identifies a framework for metrology process models to be used in assembly planning. This focuses on metrology and its place in assembly planning, so the emphasis of this framework is on creating those process models that could be used in assembly planning. Finally, the work [26] by Halperin et al. is trying to solve almost the same problem we are, in a similarly generic way, but has a lot of key differences in the approach. First, configuration

space is not used; instead motion space is used to construct a NDBG which can be used itself to create assembly sequences. The NDBG after being built would have its parallel in our method as the disassembly graph, after being exhaustively generated as in our Full BFS method. This is inherently going to be a slow process, which would end up impacting the solvability of difficult problems like some that we have studied. Further, not relying on concepts like configuration space mean abandoning a lot of tools and research that have been developed for that field, specifically sampling-based motion planning, which our method relies upon heavily and benefits from.

3. APPROACH OF THE DSP FRAMEWORK

In this chapter, we first discuss a higher-level understanding of the modules we have identified to be included in a DSP framework. Then we go module-by-module and explain the three major modules of the framework, and what might be chosen for each of them. Finally, manipulator planning is discussed, and what must be considered in the framework when a plan is also desired for one. [1]

3.1 Overview

A reliable framework had not before been established for solving DSP problems, despite there being canonical frameworks in related fields, like that in SBMP. In SBMP, the framework helps establish the modularity of the system, and so the intention of the DSP framework is to do the same. There are three key modules to the framework, and changing the selection of any of them will alter the underlying strategy for exploring the disassembly sequence space (which is the space of all sequences and sub-sequences to be considered when performing the disassembly). The key modules are as follows:

1. Disassembly Graph search strategy.
2. Which subassembly (whether single- or multi-part) to attempt to remove.
3. The expansion strategy for the subassembly chosen (can be any motion planning strategy, or can be a simple translation as in [27], with or without rotations, etc.).

Some important concepts to cover are the data structures involved. The fundamental one for SBMP is the roadmap. This is a standard graph where the nodes represent configurations in configuration space, and edges represent *valid, verified* edges between configurations. In disassembly planning, we use another level on top of this of abstraction, in something called a Disassembly

[1] This chapter contains previously published material from [1].

Graph (DG). The DG nodes represent the states of the removal sequence (whether partially disassembled or completely), and edges between them represent *valid, verified* removals that can be done simultaneously, if the edge represents more than one part being moved. In summary, the roadmap keeps track of valid points in configuration space and how to get between them for a robot, while the DG keeps track of the higher-level sequence of the assembly in disassembly sequence space. Each node in the DG is linked to one or more nodes in the roadmap.

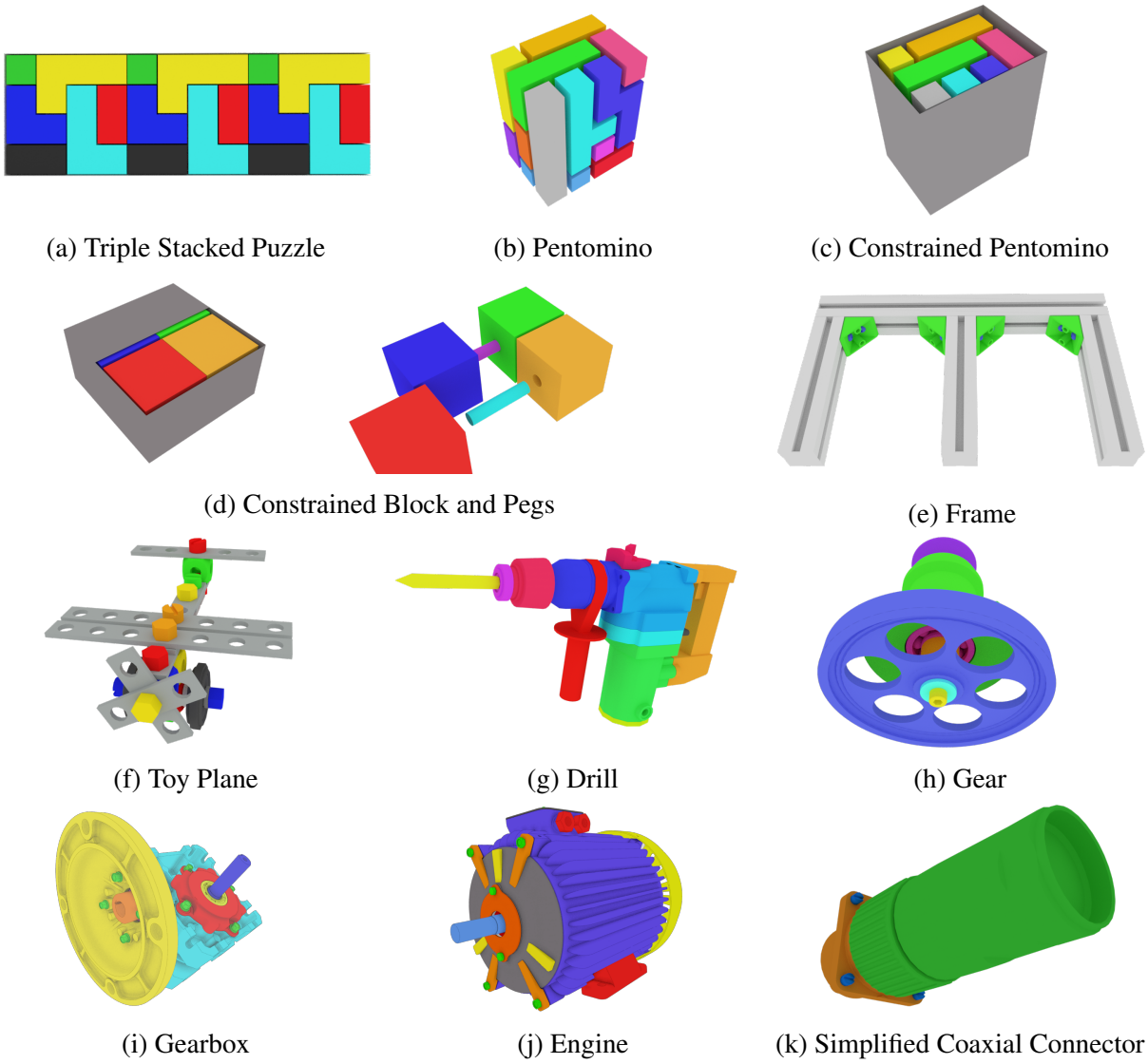
The choices made in the framework will decide the search strategy through the DG. This will be highlighted by explaining the two contrasting implementations of the framework from my existing work on this topic. The two implementations are called Preemptive Depth-First Search (DFS) and Exhaustive Breadth-First Search (BFS). Preemptive DFS search is meant to find solutions as quickly as possible. It does this by going as “deep" (towards nodes with the most disassembly progress made so far) as it can, in a greedy manner into the disassembly graph (DG). Exhaustive BFS is meant to investigate *every* path through the DG, which can be an incredible number of paths for even small assemblies. The DG has a factorial relationship to the number of parts of the assembly, which means the space grows at an impossible rate for assemblies with more than a few parts.

Details about example assemblies, like number of parts and DOFs for each, are shown in Table 3.1. The examples themselves are shown in Figure 3.1.

3.2 Disassembly Graph Search Strategy

This is the order in which to visit nodes (or the order in which to de-queue them) during the search. Preemptive DFS will always use the last-created node, which will be the node with the most amount of progress towards the solution. Exhaustive BFS has to keep track of every single DG node that is created, and keep them queued up until everything has been visited in order to guarantee that the space has been exhaustively searched. In reality, the order of the nodes in Exhaustive BFS does not matter, since all of them will be visited, and the order of visitation will not affect any outcome in that situation.

There could also be other uses of this module. For the future work planned in A* search, a



(a) Triple Stacked Puzzle (b) Pentomino (c) Constrained Pentomino

(d) Constrained Block and Pegs (e) Frame

(f) Toy Plane (g) Drill (h) Gear

(i) Gearbox (j) Engine (k) Simplified Coaxial Connector

Figure 3.1: Environments studied: Triple Stacked Puzzle (a) with a surrounding obstacle box and a opening on the right side, Pentomino both without (b) and with (c) a constraining box, Constrained Block and Peg Assembly (d), Frame with angle brackets, slot nuts and screws (e), Toy Plane built from Baufix [2] parts (f), Drilling machine with inner (hidden) parts (g), a Gear (h), a Gearbox (i), an Engine (j), and a Simplified Coaxial Connector (k) requiring simultaneous rotation and translation. Details about each environment are shown in Table 3.1. This figure was previously published in [1].

Environment Name	Number of Parts	Motions Considered	Total DOF	Source
Triple Stacked Puzzle	18	2D Translational/Rotational	36/54	[11]
Pentomino	12	3D Translational	36	[28]
Constrained Pentomino	12	3D Translational	36	[28]
Constrained Block and Pegs	6	3D Translational	18	[30]
Frame	24	3D Translational	72	[29]
Toy Plane	32	3D Translational	96	[2]
Drill	22	3D Translational	66	[29]
Gear	9	3D Translational	27	[29]
Gearbox	30	3D Translational	90	[30]
Engine	57	3D Translational	171	[30]
Simplified Coaxial Connector	6	3D Rotational	36	[30]

Table 3.1: Details of each environment from Figure 3.1 studied.

priority queue of DG nodes will be kept. The queue will be sorted based on both a metric (the edge weights of DG being used) summed with a heuristic value of the estimated cost to the goal. This means the next node chosen has the best shot of being on an optimal path that the search knows of at that time. A* is guaranteed optimal given that the heuristic used never overestimates the remaining cost to the goal state, so it will pull out the best path that Exhaustive BFS would find, but would only find the one path and would do so in a much faster way in general.

3.3 Subassembly Selection Strategy

Once a node has been selected, the next stage of the framework is to choose how an expansion should be attempted from it. There will be a list of remaining parts that are to be disassembled (if multi-part subassemblies have been removed, any that haven't been separated will be in their own subset) and a part or set of parts must be chosen, or a way to order the parts with which to expand. Preemptive DFS will first try all single parts that are available, and only if all of those fail will it move on to multi-part subassemblies. This is because attempting single parts is faster (and in the general case more likely to work) than moving multiple parts in formation. Also, if a subassembly is removed, it must still be returned to in order to fully disassemble it, so it is often not any more efficient than single part removals. Exhaustive BFS has another trivial choice for this module since it is exhaustive, but it also attempts single parts first, and then attempts multiple-part subassemblies

after. This order doesn't matter because all of these will be attempted, no matter the order or which ones fail to be removed. This is enforced in order to guarantee exhaustive searching of the DG.

The subassembly candidate generation strategy that has been used is not novel and is intended to be a piece of this module that is changed depending on the implementation. Generally speaking, the method considers parts to be candidates for a subassembly if, during some simple translating motions, they collide with each other and do not collide with obstacles.

3.4 Expansion Strategy

The final step in a node is to actually attempt a motion plan to see if the subassembly chosen is separable from the assembly in this state. Simply stated, given a state and a set of part(s) to move, this module determines how to move it, and also determines when a node should stop expanding (whether to switch methods, ask for a different subassembly, or move on to a new state). To realize this, any motion planning method can be used. So far SBMP has been used to attempt very fast and simple translation motions, and also the method of rapidly-exploring random trees (RRT) has been applied as a more general and robust method to attempt a subassembly separation (but is also slower to plan with). Preemptive DFS uses the simple translation technique as much as possible. This is an extremely fast way to attempt a removal, and is purely deterministic. It is a good first-pass for a state, and at the very least allows all of the more simple parts of an assembly to be removed before trying something more expensive on parts that might be simple. I-ML-RRT[12] does not take advantage of this, and when parts can be removed along a straight line, yet are tight fitting in the assembly, the penalty of not considering translations in this manner is seen by significantly worse planning times. Preemptive DFS only attempts RRT if the simple mating approach fails on *all* single- and multi-part subassemblies in a given state. Once progress is made from a node, the visitation of that node ends and the new state that has been generated will be expanded from. Other potentially valid outgoing edges from this DG node will not be investigated for this method.

Exhaustive BFS will just change the order in which RRT is attempted. Since all subassemblies must be exhaustively tried, if the simple translation fails for any given subassembly, RRT is attempted immediately to see with a higher degree of confidence whether it can be removed from

the current state at all.

3.5 Manipulator Planning

3.5.1 End Effector Placement

On top of what is in our prior publication [1], is the simulation of a manipulator arm which can carry out the plan generated by the method. This essentially involves a post-processing step to the disassembly plan that is generated by the above methods. In the work thus far, the planner assumes parts can move freely and does not have the motions for how a manipulator or its end effector might reach a part. The main premise is to generate a plan using the implemented framework, and then generating the configurations for the end effector such that it can carry out the motions. In the cases where it cannot, ideally a re-planning step will occur, which will involve generating a new motion plan for the part, or perhaps rearranging the order in which parts will be attempted. However, this is currently not attempted and is left for future investigation on this topic.

That is not to say that the disassembly sequence plan is uninformed about the end effector or the manipulator; when manipulator planning is being considered, an initial test before any planning for a given subassembly takes place, where it is ensured that the end effector can be placed somewhere on the subassembly being moved, given the configuration that is to be started from. If it fails, it is counted the same as a failure of an expansion approach normally is. This step helps filter out disassembly sequences where replanning (which will potentially be an expensive step) might have later been necessary.

The additional algorithms to the disassembly framework follow. The first step is placing the End Effector (EE) as shown by Algorithm 1. Then the full manipulator is placed after having the EE positioned as shown by Algorithm 2.

3.5.2 Manipulator Placement Using Reachable Volumes

The theory of Reachable Volumes (RV) describes a way to get random, valid samples for a manipulator arm given a placement of its end effector. This is why the first step for this was to place an end effector, and verify that it can follow along to remove each subassembly, and move in

Algorithm 1 End Effector Placement Framework

function PLACEENDEFFECTOR(DG, R)
    Input: A disassembly graph (DG) with complete disassembly sequence, and a roadmap of valid configurations (R).
    Output: A removal trajectory T with the manipulator validly placed along all path configurations.
    Initialize roadmap R_e and R_m as empty roadmaps, respectively representing the assembly and EE and the assembly and the manipulator.
    Initialize $path$ to be an empty path of configurations
    for each node n in the disassembly sequence **do**
        $removalSuccess \leftarrow false$
        while $removalSuccess$ is false **do**
            $removalCfgs \leftarrow n.removalCfgs$
            $lastCfg \leftarrow removalCfgs$.pop_back()
            $removalPath \leftarrow$ PlaceEEOnPath($n.removedParts$, $removalCfgs$)
            if removalPath has less than 2 cfgs **then**
                Report failure and exit
            end if
            if n is not the first node in the disassembly sequence **then**
                $lastRemovalCfg \leftarrow$ last configuration of previous removal
                $success \leftarrow$ AttemptEELocalPlan($lastRemovalCfg$, $removalPath.first$)
                if $success = false$ **then**
                    Return to PlaceEEOnPath(), starting from the next polygon to attempt
                end if
            end if
            Concatenate $removalPath$ onto the end of $path$
        end while
    end for
    return $path$
end function

the environment in such a way to get from removal to removal. Once this has been done, we have a set of end effector placements that we can then use for RV to follow along with, and perhaps re-plan as necessary.

There are some aspects of the actual manipulator placing that are to be left for future work, however. The most glaring of these is local planning. Currently, as the final post-processing step after the end effector's path has been found and validated, a sample is created for a manipulator position for all configurations along this path. The future work, then, is to attach the subassembly

Algorithm 2 PlaceEEOnPath Function

```
1: function PLACEEEONPATH(S, C)
2:     Input: The part(s) being removed (S) and the configurations on which to find an EE placement (C).
3:     Output: Configurations which now include the EE, which all include the same, valid placement for the EE.
4:     for each individual part P in S do
5:         for each of the N largest polygons polygon on P do
6:             path ← empty configuration path
7:             for each configuration cfg from C do
8:                 successful ← AttemptEEPlacement(polygon, cfg)
9:                 if successful = true then
10:                    if cfg is not the first configuration of C then
11:                        lastCfg ← last configuration with EE placed
12:                        successful ← AttemptLocalPlan(lastCfg, cfg)
13:                    end if
14:                end if
15:                if successful = false then
16:                    Break loop, reattempt from next polygon.
17:                end if
18:                path.push_back(cfg)
19:            end for
20:        end for
21:    end for
22:    return path (returns empty path if ran out of polygons/parts to attempt)
23: end function
```

to the manipulator between each of these configurations, and perform some kind of local plan. It is believed that a highly goal-directed RRT would perform very well and be a generic way to handle this problem.

3.6 A* Search in Disassembly Sequence Space

The framework has been shown to provide a diverse ability to implement search strategies. The strategies themselves have been proven to be more effective than the state-of-the-art DSP strategies that exist already, though more improvement and investigation into more intelligent search strategies is still warranted. The popular A* search strategy has been adopted for this purpose, and the framework has been used in a way to take advantage of this graph search strategy. Analysis of this

method, along with analysis of the best metrics and heuristics, is left for future work.

4. RESULTS

The results in this chapter are split into two major categories. This first does not consider planning for a manipulator, while the second set includes a manipulator in the planning process. [1]

4.1 DSP Without Manipulation Planning

The framework was designed and both Preemptive DFS and Exhasutive BFS were implemented for [1], and compared to I-ML-RRT[12]. The example assemblies used are in Figure 1, and the results of the tests are in Figure 2. Data is averaged over 10 runs with a timeout of 5000 seconds, except for the Gearbox and Engine examples which were given 10000 and 15000 seconds, respectively. Average times and path metrics are only reported for successful runs.

The Preemptive DFS method is the only one to always successfully solve each problem within the allotted time. It is also at least an order of magnitude faster than I-ML-RRT. The improved performance may be attributed to both the use of mating vectors and subassemblies. Mating vector exploration is faster than RRT exploration, even if the RRT exploration was constrained to mating vector directions. The more frequently mating vector separation is successful, the greater the improvement in running time. Recall that I-ML-RRT does not handle subassemblies and only allows for adjusting other parts if the single part being attempted causes collisions. This is a slow way to handle problems that require coordinated motion of parts, and means it cannot solve problems that truly require *simultaneous* movements of parts, like our Constrained Block and Pegs example where there are tight-fitting parts of the assembly (e.g., the rod between the two blocks, all surrounded by a snug container).

Preemptive DFS produces much shorter paths in terms of numbers of roadmap nodes due to its use of mating vectors in addition to RRT exploration for all problems. The number of roadmap nodes along the path indicates the complexity of movement required as nodes occur at direction changes. Here we see the impact of mating vector separation on path simplicity.

[1] This chapter contains previously published material from [1].

Model	Rotation	Success Rate (%)		Avg. Successful Time (sec)		Avg. Path Length (nodes)	
		Preemptive DFS	I-ML-RRT	Preemptive DFS	I-ML-RRT	Preemptive DFS	I-ML-RRT
Triple Stacked Puzzle 2D	No	100	100	0.30	29.27	36	80
Triple Stacked Puzzle 2D	Yes	100	100	0.31	40.27	36	73
Pentomino	No	100	100	0.05	1.67	24	36
Constrained Pentomino	No	100	100	0.11	29.57	24	97
Constrained Block and Pegs	No	100	0	19.57	-	26	-
Frame	No	100	0	23.27	-	48	-
Toy Plane	No	100	90	6.46	2609.14	64	112
Drill	No	100	0	25.81	-	44	-
Gear	No	100	0	208.70	-	18	-
Gearbox	No	100	0	5352.27	-	60	-
Engine	No	100	0	10638.08	-	114	-
Simplified Coaxial Connector	Yes	100	100	50.98	1007.31	12	19

Table 4.1: Comparison of Preemptive DFS to I-ML-RRT for the models in Fig. 3.1. Results are averaged over 10 runs, excluding failed runs. All methods are restricted to translation except for the Simplified Coaxial Connector (which requires rotation) and the rotation version of the Triple Stacked Puzzle 2D. This table was previously published in [1].

Figure 3 shows the results of the Full BFS method. The full search identified hundreds to thousands of collision-free disassembly solutions for both problems, each with unique disassembly sequences. This of course comes at the cost of much larger running times, which is exponential with the number of parts. Even for these small problems, there is a wide range of path lengths in the solution set. The Full BFS strategy finds a solution with shorter path length than either the Preemptive DFS or I-ML-RRT which both terminate after finding the first viable solution.

Not only does this strategy find shorter paths, it can probe the distribution of solution path lengths. The Single Stacked Puzzle is a simple problem where subassemblies are not required and parts may be be removed using only single translations toward the opening, favoring the mating vector approach. This is reflected in the path length distribution where it has a single peak at a relatively short path length. The Constrained Block and Pegs problem has a different distribution (multiple peaks, less heavily weighted toward shorter/mating vector paths). Unlike the prior problem, it requires subassemblies and more complex extraction paths (using RRT) to properly position the parts near the opening. The full details of the Preemptive DFS results are shown in Table 4.1. The full details of the Full BFS results are shown in Table 4.2.

Model	Time (sec)	# of Valid Solutions	Path Length Range (nodes)
Single Stacked Puzzle	79.81	5446	12 – 48
Constrained Block/Pegs	550.36	560	28 – 97

Table 4.2: Performance of Full BFS. The method finds many solutions that are possible in both cases. The wide range of path lengths are shown, as measured by the number of nodes (configurations) that are connected by straight lines in the path's configuration space. This table was previously published in [1].

4.2 DSP With Manipulation Planning

The test environments Pentomino, Simplified Coaxial Connector, and Constrained Block and Pegs from the last section have been the focus of validating the manipulation placement. In the first two, a full solution is found (meaning the DSP plan is formed, the end effector placed and all local plans verified, and then the manipulator can be placed along configurations). Figures 4.1, 4.2, 4.3, and 4.4 show a couple of simple manipulator placements that were found for these examples.

For the Constrained Block and Pegs environment, there is only an issue for having the end effector finding a viable path from the end of one removal to the start of the next. This is likely due to needing a more robust planning method for this step, though the solution and analysis of this problem is left for future work.

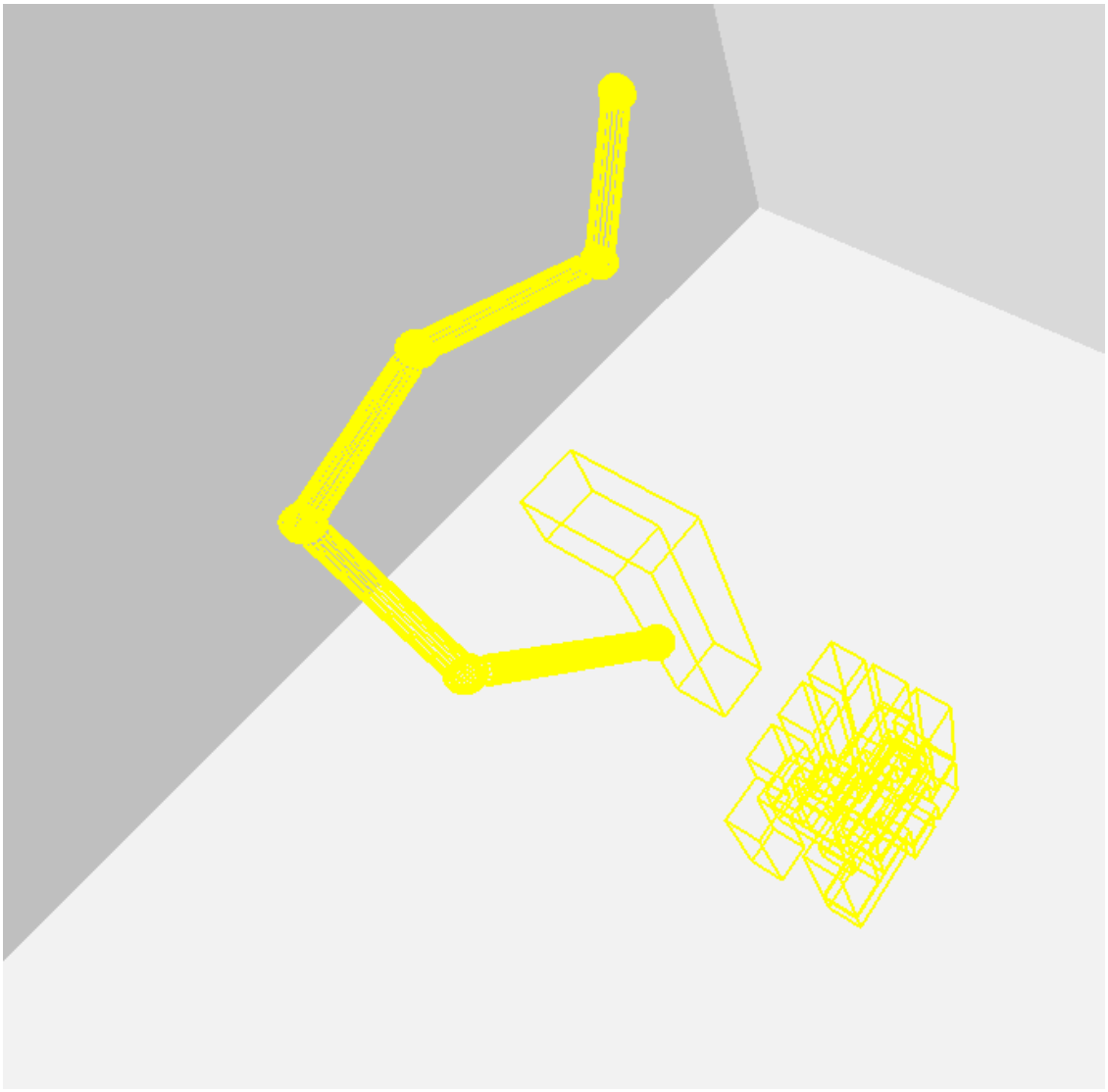


Figure 4.1: Manipulator placement on a part removal in Pentomino environment

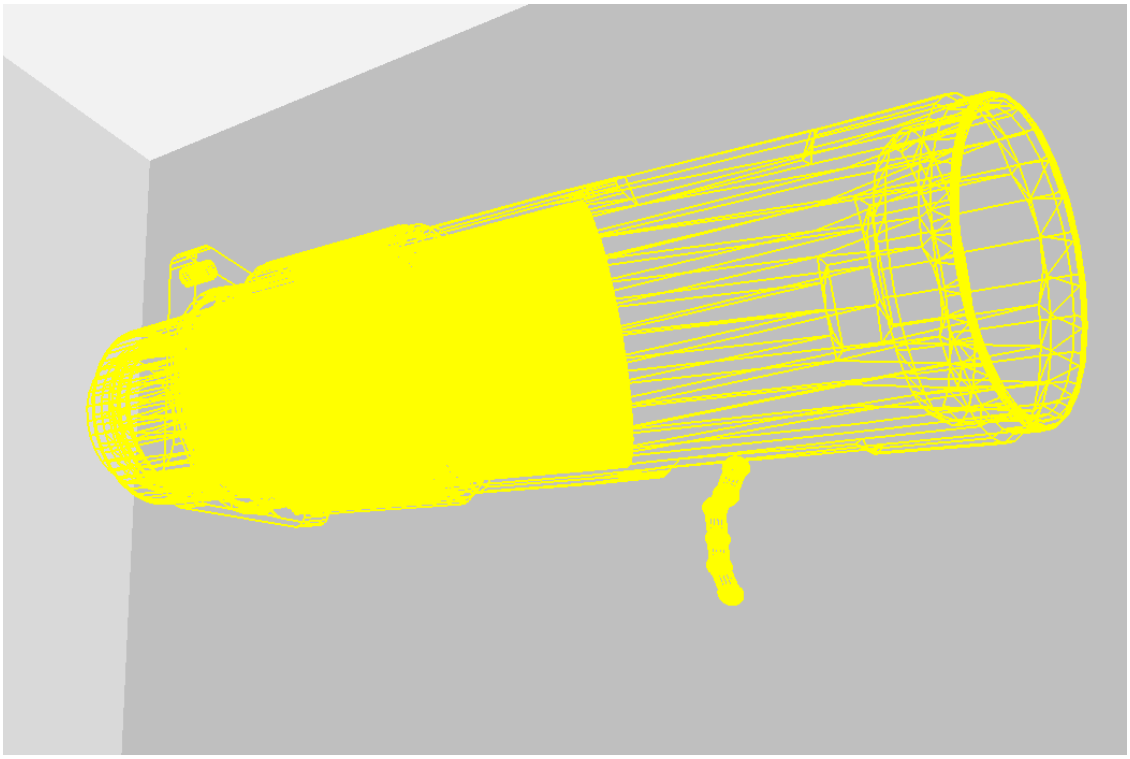


Figure 4.2: Manipulator placement on a part removal in Simplified Coaxial Connector environment

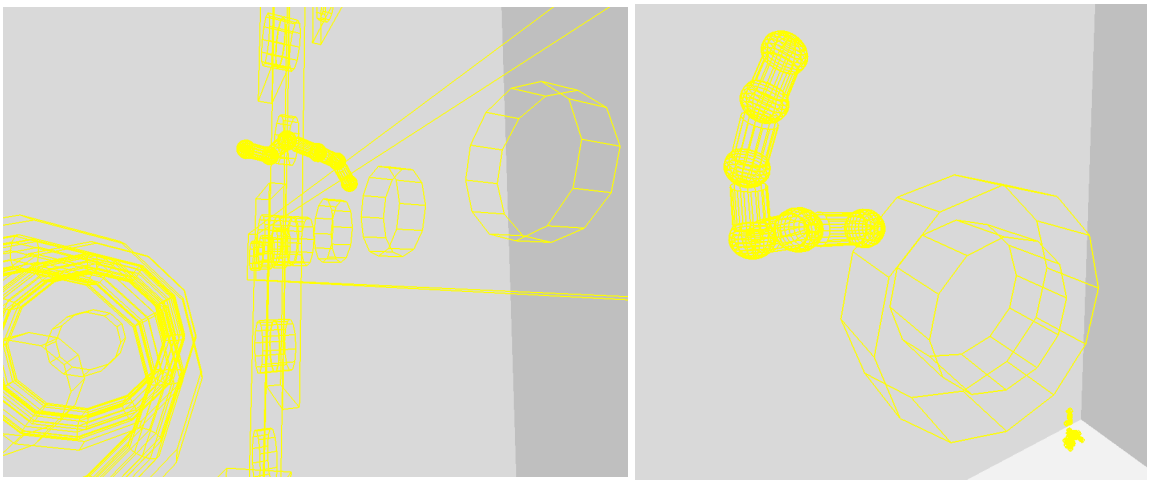


Figure 4.3: Manipulator placements on a part removals in Toy Plane environment

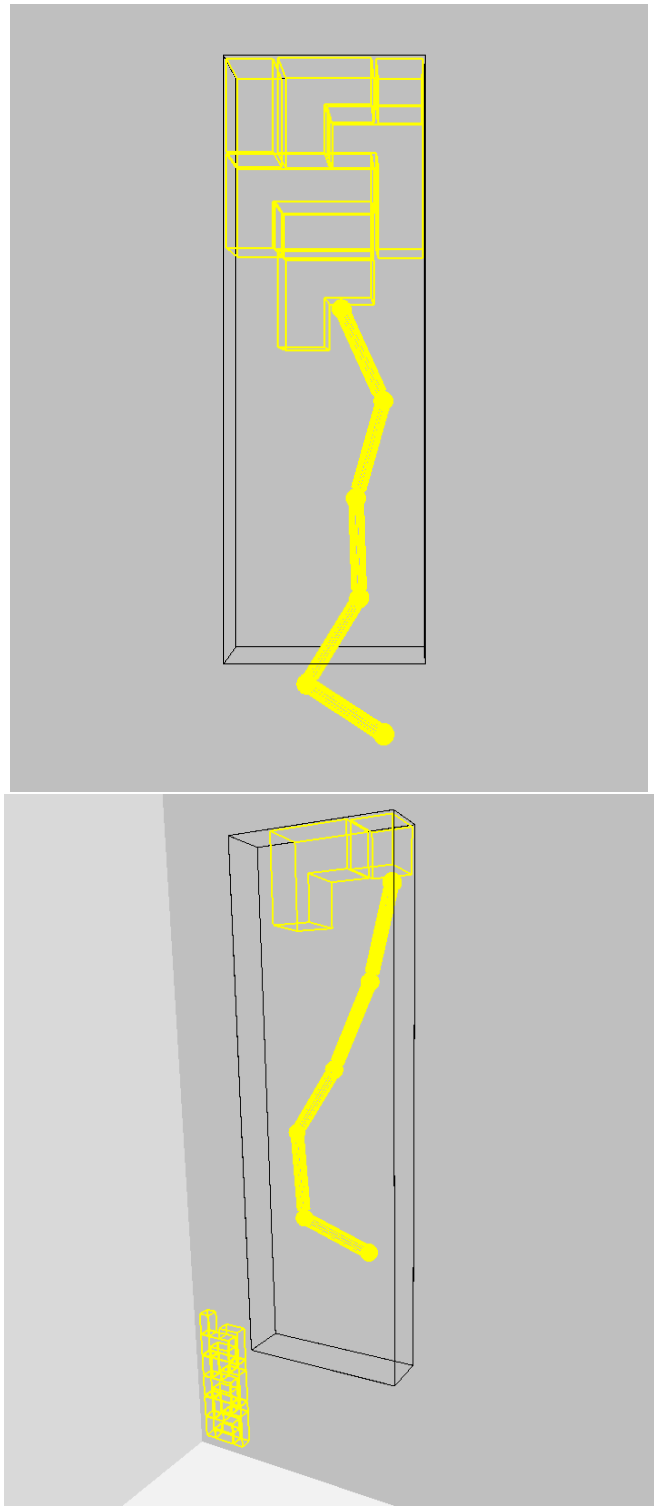


Figure 4.4: Manipulator placements on a part removals in Triple Stacked Puzzle environment. These are both at difficult stages of disassembly as a manipulator must reach deep inside the box in order to reach parts being removed.

5. SUMMARY AND CONCLUSIONS

A framework for solving disassembly sequence planning problems has been described and analyzed. The framework is versatile in nature and breaks up what a DSP solver needs into individual modules. Contrasting implementations of the framework are demonstrated for evaluation against the current state-of-the-art methods for solving DSP problems. An implementation on a manipulator arm in simulation helps to provide a proof-of-concept about the usefulness of the framework and its potential for realistic applications of the problem.

The framework is fully designed and instantiations of it (the strategies discussed) have been implemented, verified, and applied to simulated problems. Data to evaluate performance, as well as another method to compare against, has been generated and analyzed. All of this was included in the previously-discussed ICRA 2018 publication [1]. Expanding on that we have used a manipulator arm and its end effector to make this more immediately useful to those needing to solve DSP problems in real scenarios. This is mostly done as a post-processing step, which shows the versatility of the framework and its outcomes. While there are still many open aspects to DSP, we believe that this framework is the best foundation for identifying and solving future problems of this kind.

REFERENCES

[1] T. Ebinger, S. Kaden, S. Thomas, R. Andre, N. M. Amato, and U. Thomas, "A general and flexible search framework for disassembly planning," in *International Conference on Robotics and Automation*, May 2018.

[2] "Baufix Wooden Toys." http://www.baufix.de/.

[3] Y. Tang, M. Zhou, E. Zussman, and R. Caudill, "Disassembly modeling, planning, and application," *Journal of Manufacturing Systems*, vol. 21, no. 3, pp. 200–217, 2002.

[4] L. S. Homem de Mello and A. C. Sanderson, "Automatic generation of mechanical assembly sequences," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1988.

[5] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, pp. 560–570, October 1979.

[6] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, pp. 378–400, May 2001.

[7] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, pp. 566–580, August 1996.

[8] I. Aguinaga, D. Borro, and L. Matey, "Parallel rrt-based path planning for selective disassembly planning," *The International Journal of Advanced Manufacturing Technology*, vol. 36, no. 11, pp. 1221–1233, 2008.

[9] J. Xiong, Y. Hu, B. Wu, and X. Duan, "Minimum-cost rapid-growing random trees for segmented assembly path planning," *The International Journal of Advanced Manufacturing Technology*, vol. 77, no. 5-8, pp. 1043–1055, 2015.

[10] L. Zhang, X. Huang, Y. J. Kim, and D. Manocha, "D-plan: Efficient collision-free path computation for part removal and disassembly," *Computer-Aided Design and Applications*, vol. 5, no. 6, pp. 774–786, 2008.

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

[12] J. Cortés, L. Jaillet, and T. Siméon, "Disassembly path planning for complex articulated objects," *IEEE Trans. Robot.*, vol. 24, no. 2, pp. 475–481, 2008.

[13] K. E. Moore, A. Güngör, and S. M. Gupta, "Petri net approach to disassembly process planning for products with complex and/or precedence relationships," *European Journal of Operational Research*, vol. 135, no. 2, pp. 428–449, 2001.

[14] Y. Tang, M. Zhou, and R. J. Caudill, "An integrated approach to disassembly planning and demanufacturing operation," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 773–784, 2001.

[15] L. M. Galantucci, G. Percoco, and R. Spina, "Assembly and disassembly planning by using fuzzy logic & genetic algorithms," *International Journal of Advanced Robotic Systems*, vol. 1, no. 2, p. 7, 2004.

[16] E. Kongar and S. M. Gupta, "Disassembly sequencing using genetic algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 30, no. 5, pp. 497–506, 2006.

[17] W. Hui, X. Dong, and D. Guanghong, "A genetic algorithm for product disassembly sequence planning," *Neurocomputing*, vol. 71, no. 13, pp. 2720–2726, 2008.

[18] Y.-J. Tseng, F.-Y. Yu, and F.-Y. Huang, "A green assembly sequence planning model with a closed-loop assembly and disassembly sequence planning using a particle swarm optimization method," *The International Journal of Advanced Manufacturing Technology*, vol. 57, no. 9, pp. 1183–1197, 2011.

[19] S. A. Reveliotis, "Uncertainty management in optimal disassembly planning through learning-based strategies," *Iie Transactions*, vol. 39, no. 6, pp. 645–658, 2007.

[20] S. Smith, G. Smith, and W.-H. Chen, "Disassembly sequence structure graphs: An optimal approach for multiple-target selective disassembly sequence planning," *Advanced engineering informatics*, vol. 26, no. 2, pp. 306–316, 2012.

[21] Y. Huang and C. G. Lee, "A framework of knowledge-based assembly planning," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 599–604, IEEE, 1991.

[22] C. M. Costa, G. Veiga, A. Sousa, L. Rocha, E. Oliveira, H. L. Cardoso, and U. Thomas, "Automatic generation of disassembly sequences and exploded views from solidworks symbolic geometric relationships," in *Autonomous Robot Systems and Competitions (ICARSC), 2018 IEEE International Conference on*, pp. 211–218, IEEE, 2018.

[23] F. Demoly, X.-T. Yan, B. Eynard, L. Rivest, and S. Gomes, "An assembly oriented design framework for product structure engineering and assembly sequence planning," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 33–46, 2011.

[24] L. Da Xu, C. Wang, Z. Bi, and J. Yu, "Autoassem: an automated assembly planning system for complex products," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 3, pp. 669–678, 2012.

[25] P. G. Maropoulos, Y. Guo, J. Jamshidi, and B. Cai, "Large volume metrology process models: A framework for integrating measurement with assembly planning," *CIRP Annals-Manufacturing Technology*, vol. 57, no. 1, pp. 477–480, 2008.

[26] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3-4, pp. 577–601, 2000.

[27] U. Thomas, M. Barrenscheen, and F. M. Wahl, "Efficient assembly sequence planning using stereographical projections of c-space obstacles," in *Assembly and Task Planning, 2003. Proceedings of the IEEE International Symposium on*, pp. 96–102, IEEE, 2003.

[28] S. Sundaram, I. Remmler, and N. Amato, "Disassembly sequencing using a motion planning approach," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp. 1475–1480, 2001.

[29] R. Andre and U. Thomas, "Anytime assembly sequence planning," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pp. 1–8, June 2016.

[30] "GrabCAD." http://www.grabcad.com/.