A Hybrid Assembly Task Planning System: Where Motion Planning Helps Symbolic Planning Find Good Solutions For Real-World Applications

Frederik W. Heger and **Sanjiv Singh**
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

Assembly has a natural step-by-step structure that makes it a good candidate for symbolic planning approaches. At the same time, for robots to successfully perform assembly tasks in challenging environments, they require motion planning capabilities to efficiently navigate. We present a hybrid approach to assembly planning where a symbolic planner constrains the search for a good assembly plan to valid sequences that satisfy all structural constraints while a motion planner helps with the evaluation of actions in the context of the real-world scenario. The result is a planning system that guarantees nominal feasibility of any plan it generates. In addition, as things will inevitably go wrong during execution, our framework is able to seamlessly repair and re-plan assembly sequences as necessary. The planner has been successfully applied to robotic assembly scenarios both in simulation and with real robots.

Introduction

People's first reaction to the term "robotic assembly" is usually a mental picture of an industrial assembly line where stationary robots perform repetitive tasks at high speeds and with high precision. That is not the kind of "assembly" our work is about. Instead, we consider mobile manipulators retrieving components from a storage location, transporting them through their environment and assembling them into a large structure. We are developing a framework for planning assembly tasks that, given a desired goal structure, automatically decomposes the task and commands robots to execute them. Our system seamlessly reacts to execution-time failures – which inevitably will occur – by repairing and re-planning the task as necessary.

Assembly has a natural step-by-step structure where a sequence of actions transforms an initial (disassembled) state into a desired final (assembled) state. For assembly tasks performed by real robots and in non-trivial environments, symbolic plan solutions at the level of intermediate structure configurations are insufficient to generate desirable plans. To overcome this limitation, the planner requires a way to reason about physical feasibility of assembly steps it considers. Motion planning techniques are uniquely suited to provide such information, but they require the large assembly problem to be broken down into smaller sub-problems to be able to find solutions.

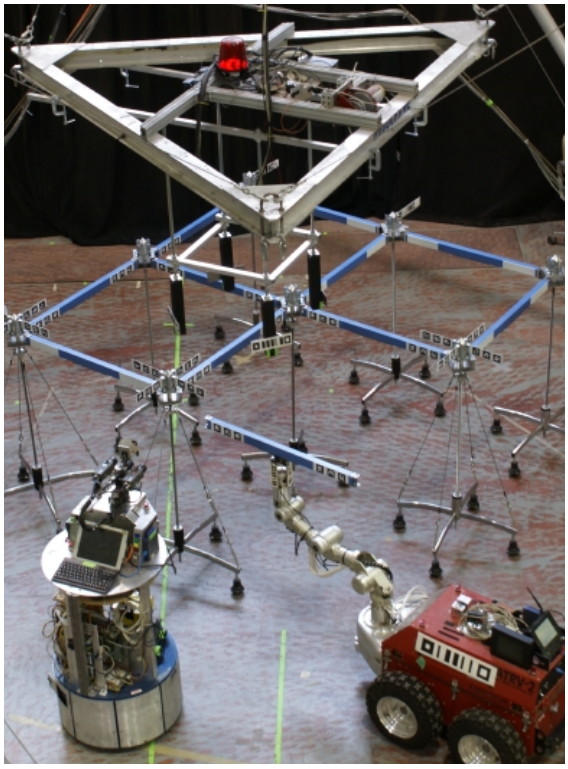

Figure 1: A nearly completed lattice of 21 components. Structural (internal components before external components) and environmental (the workspace is tight – can the robots get to where they need to be to perform their tasks?) constraints need to be considered by the planner.

This paper describes a hybrid planning system that combines a graph-based symbolic representation of an assembly problem with continuous (motion planning) reasoning about physical feasibility during planning. We consider assembly problems of lattices of beams and nodes (see Fig. 1) that have to be assembled by mobile manipulation robots in constrained environments. Given a structure to be built in an environment, the planner automatically generates an assembly sequence that can be directly executed by the robots. The planner ensures that a nominally feasible plan for the entire assembly exists before any robot starts executing tasks. If execution-time failures occur – as they likely will even

for perfect plans – plan repairs and re-plans require minimal "physical backtracking" by the robots in the environment.

Vision and Motivation

We envision an assembly planning system where the user or operator only has to supply a desired goal structure to be assembled and a place where the final assembly is to be located within the environment, and an automatic planner takes care of the rest. In cases where full autonomy is not desirable, an operator could be informed of key steps along the assembly task and help make decisions, or he could be involved more actively by assisting the robots (on request or proactively) to resolve and avoid problems.

Such a system will be of great interest to NASA in the context of establishing planetary outposts or habitats where human presence (especially during early phases of construction) is prohibitively expensive, not to mention dangerous for the workers. If robots can be sent ahead to prepare the necessary infrastructure with little supervision or help from remote humans, astronauts can focus on aspects of the mission they are better suited for, such as science experiments.

As a more down-to-earth example, imagine moving into and furnishing an apartment with your favorite assemble-yourself furniture – or rather, have robots assemble all the furniture according to your floor plan of where the bed/desk/shelf/... should go for you while you are exploring your new neighborhood.

State of the Art

Assembly is a challenging and difficult task (even for humans) with such complicating factors as tight tolerances, heavy and/or large parts to manipulate and more. By definition, robots will have to operate close to other objects, and they have to be able to sense, navigate, manipulate all at the same time as they are constructing a growing obstacle in their already constrained workspace. Intertwined with the necessity to reason about motion through the environment, there is inherent structure to an assembly where internal components must be installed before the outer layer is complete. To our knowledge, there is currently no system that is able to automatically generate assembly plans for mobile manipulators assembling large structures in realistic environments. While various parts of the overall problem have been addressed by previous work, a comprehensive solution has not yet been developed.

We have worked on robotic assembly scenarios for several years. This paper extends our previous work (Heger 2008) with a more efficient implementation to handle structures of up to 21 components (instead of 8 previously), a task executive that takes generated plans and directly turns them into task trees for robots to execute, and plan repair and re-planning capabilities throughout all levels of the planning hierarchy to allow the system to effectively react to and resolve execution-time exceptions. The planner is tied to the simulated or real-robot scenario using a task sequencing executive that also provides capabilities for sliding autonomy interaction with a user. This work is motivated by earlier experiments with real robots (Sellner et al. 2006) where a rather simple assembly plan was scripted by hand – a task that is not feasible for larger and larger structures.

Contributions

The work presented in this paper makes the following key contributions toward the goal of a more general and useful assembly planning system:

1. An expressive representation of assemblies that decomposes the problem into sub-problems that can be attacked using existing planning methods.
2. A method to automatically generate plans for mobile manipulators to assemble a desired structure.
3. Seamless repair and re-planning of assembly sequences in response to execution-time failures and exceptions.

Related Work

Prior work relevant to assembly planning falls into two main categories: approaches that treat assembly as a sequencing problem on an abstract symbolic level, and ones that consider fine-grained motions of the robots involved. We see both as integral aspects of a larger problem that cannot be solved well with either one method alone.

Symbolic Planning Symbolic methods abstract problems into simple operators with preconditions and effects. A sequence of operators that transform given initial conditions into desired final configurations describes a plan for the scenario (Fahlman 1973; Younes and Simmons 2003). Such approaches efficiently exploit the step-by-step nature of many problems by abstracting away difficult to compute constraints into simple heuristics. This abstraction, however, limits the reasoning about the real world to queries that have to be answered by an outside process (e.g., an oracle). Infeasibility of a plan often cannot be detected until the robots – during execution – come to a dead end because a workspace constraint was unknown during planning. Plan verification systems can help reduce such problems by verifying a symbolic plan step by step either after it is completed or while it is being planned (Kaufman et al. 1996).

Motion Planning Other applicable work focuses on the motion planning aspects of the problem (Bozma and Koditschek 2001; Karagöz, Bozma, and Koditschek 2004; Lengyel et al. 1990), but plans produced by those approaches do not always satisfy critical structural constraints imposed by the structure to be assembled. Koditschek et al.'s navigation functions, for example, assumes that all states between the initial and final configurations are valid (including partially assembled components), which is not the case in assembly scenarios. There is no guarantee that extrema in the navigation functions where assembly roles change coincide with valid assembly states where such a change is allowed. Klavins describes self-assembly using graph grammars to encode local interactions agents may engage in. The resulting global process results in an organization behavior that brings individual self-moving parts into an assembled configuration (Klavins 2006).

Since assembly scenarios have a distinct underlying step-by-step structure, pure motion planning approaches do not produce the results we are looking for. Stilman et al.'s navigation among movable obstacles (Stilman and Kuffner 2004) plans first in an abstract graph of configuration space segments and then uses motion planning techniques to evaluate paths suggested by graph edges. Manipulation planning is faced with similar challenges to assembly planning at a finer level of detail (e.g., dextrous motions to grasp and re-grasp components (Nielsen and Kavraki 2000; Gravot, Alami, and Siméon 2002)). An assembly plan sets up manipulation planning problems for each step in the plan.

Assembly Planning Due to its step-by-step nature, traditional assembly planning is a prime setting for symbolic planning approaches. Homem de Mello developed a representation for describing mechanical assembly sequences based on AND/OR graphs (Homem de Mello and Sanderson 1988; Homem de Mello 1989) similar to our representation. Using this graph structure, he presented a complete and correct algorithm for generating assembly sequences of a desired configuration by planning the disassembly of the goal structure (Homem de Mello and Sanderson 1991). Existing approaches are limited to highly structured environments (e.g., work cells, (Homem de Mello and Sanderson 1988; Kaufman et al. 1996)) and are usually concerned with assembly feasibility and serviceability of parts in assemblies. The focus is on optimal plans to maximize efficiency of the assembly/production process. Once a plan has been found, it will be executed thousands of times without variation.

This paper extends our previous work on graph-based assembly representations (Heger 2008) in several ways. Under the hood, we have improved the implementation of our representation to be able to handle larger structures (21 components, up from 8), and we have implemented exception handling capabilities throughout the framework that allow plan repair and re-planning as necessary in response to execution-time failures. To visualize the planner's output and exercise the system, we developed a simulation environment driven by a task executive that directly converts generated plans into task trees for (real or simulated) robots.

Robotic Assembly In addition to our own work in multi-robot assembly (Sellner et al. 2006) we are aware of one other group where real robots cooperate to assemble a (simple) structure (Stroupe et al. 2005). Both efforts thus far focus on the execution part of the problem and operate according to a simple script written by hand that is followed by the robots. The planning system we describe here will replace manual scripting of assembly actions with automatic tasking of assembly robots based on a high-level goal specification of the structure to be assembled.

Approach and Implementation

Our assembly planning framework consists of four main components (see Fig. 2): the assembly planner itself that generates the assembly sequence, an executive responsible for parameterizing robot behaviors, real or simulated robots to execute those behaviors and exception handling mechanism to recover from execution-time failures.

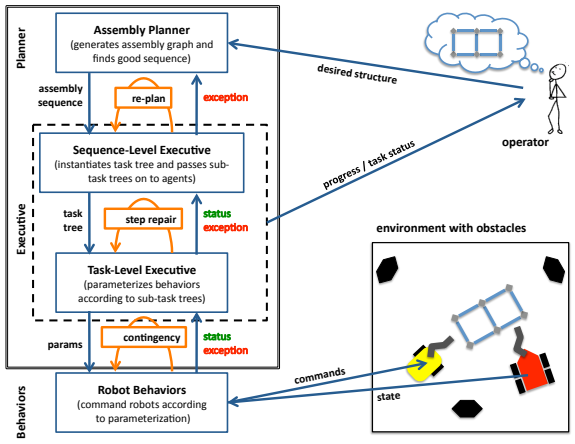

Figure 2: High-level overview of the entire system. This paper covers the parts enclosed by a solid line. The planner generates an assembly sequence based on high-level input form the operator. The executive decomposes the task tree into sub-tasks and parameterizes robot behaviors. Execution-time exceptions trigger recovery actions throughout the hierarchy.

Assembly Planner

The assembly planner is the workhorse of the system. Given a desired goal structure, an environment, available robots and a final pose for the assembly, its goal is to produce a sequence of assembly steps (i.e., *AssembleComponent 1* $\rightarrow$ *AssembleComponent 2* $\rightarrow$ ... $\rightarrow$ *AssembleComponent N*) that are nominally feasible (assuming some characteristic information about the robots involved and no significant deviations from the plan during execution). Obviously, nominal feasibility is no guarantee for success, but it is usually a good start, and fewer plan repairs are necessary.

The underlying representation of the assembly planning problem (Heger 2008) is a directed graph with unique assembly states (intermediate configurations of the structure as it is being assembled) at its vertices and the plans necessary to add a new component to the growing partial structure on its edges. Edges are weighted by the goodness of their associated plans (see below), and vertices are scored on the structural properties of the partial assembly they represent. Assembly plans are traces through this graph from an initial configuration (usually the completely disassembled state) to a final configuration (usually the fully assembled state). The chosen graph edges contain all the parameters necessary to enable the executive to create the corresponding task tree. The planner searches the assembly graph using A* to find the assembly sequence to publish to the executive.

Assembly Graph The basic version of the assembly graph contains vertices for each structure state between completely disassembled and fully assembled and edges with the appropriate component being added to transition from one state to the next (Fig. 3 (left), see (Heger 2008) for more detail). The graph is constructed goal-to-start by beginning with the fully assembled structure at vertex 0 and then considering

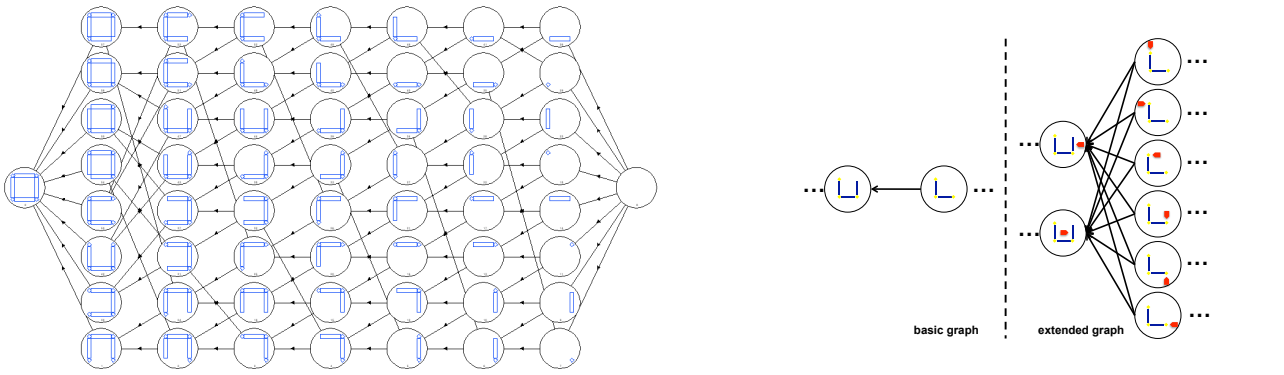

Figure 3: (left) Basic graph representation of a simple assembly problem (a one-square subset of the assembly shown in Fig. 1). Vertices mark structure configurations, edges indicate assembly steps to transition from one configuration to another (by adding a new component to the structure). (right) Graph expansion to include robot state in addition to structure state.

all valid components for removal (since we are planning for physically feasible assembly, only components external to the structure are considered – internal components are considered not reachable by robots approaching the structure from the outside). The graph generation implicitly assumes that removing a component from a structure state is simply the reverse of installing it in the previous state. This is a common assumption in assembly planning, and all assembly steps considered in this paper meet this requirement.

There is one significant limitation to the usefulness of this representation. With each structure state represented as a single vertex, it cannot capture the fact that the robot(s) performing the assembly may be positioned at different locations around the structure depending on how they transport/handle a component. The robot's task plan to assemble a component depends on which component was installed prior to the current one, and where the robot was located during that installation. Thus, in order to instantiate an assembly plan, the planner first needs to select a trace through the assembly graph to ground all necessary motions. This selection can only consider qualities of the intermediate assembly configurations (e.g., stability, approachability, etc.), but no measure of quality associated with the motion necessary during the sub-task (since the specific motion goals are not known until the sequence is established). Using a plan verification step to test potential candidate sequences, the planner attempts to generate the required motion parameters. As it encounters a step that cannot be parameterized (e.g., the environment does not allow for the required motion of the robot(s)), the corresponding assembly graph edge is marked impassable and the graph search is repeated.

At the cost of additional vertices and edges (the number of vertices increases 4-7x, the number of edges increases 8-10x, and the number of traces through the graph increases by three orders of magnitude for our sample structures), the extended assembly graph addresses the issue of single graph nodes in the base graph representing multiple and potentially incompatible states of the assembly and robots. Instead of vertices that mark only different structure configurations, this larger graph also considers the state of the manipulating robots relative to the structure (Fig. 3 (right)). Initially generating the extended graph is significantly slower than the base graph, but it enables us to meaningfully score assembly operations using motion planning considerations. Edges in the extended assembly graph can have a weight associated with them that indicates not only whether or not an assembly location is valid (which is all the base graph could guarantee), it also can make a statement about how good (e.g., in terms of distances driven by the assembling robot, tight areas to be maneuvered through, etc.) it is. While it takes more time up-front to construct the larger graph, the greater expressive power of this representation allows the planner to find better plans for larger structures. In addition, the assembly graph is static for a given structure (the graph structure does not change, only the weights of vertices and edges during the search step depends on the specific environment) and can be pre-computed and stored.

Graph Search and Plan Generation With the assembly problem represented in graph structure, finding an assembly plan becomes a graph search problem. Any trace through the graph from the vertex marking the fully disassembled structure to the vertex marking the completed assembly is a candidate plan that satisfies all assembly requirements at a symbolic level (i.e., all states are valid along the way, internal components are dealt with while they are reachable, etc.). The goal is to find a sequence that is also feasible (and desirable) for actual robots to execute.

When using the basic assembly graph, this symbolic feasibility is all that can be extracted from the graph, and the actual plan generation is an iterative process of trying to instantiate a candidate plan step by step using a motion planner (with a candidate sequence, the robot's positions are known based on incoming and outgoing edges at each vertex, this is information that was unavailable until the graph search was complete) and a re-search of the graph if the motion planner cannot find a valid instantiation of a particular edge (in that case, the corresponding edge is marked with impassable cost, and the graph search is repeated for another candidate plan). Note that this action does not consider alternate approach positions for the same robot/component pair, as such a change would invalidate the remainder of the assembly sequence under consideration. Using instead the extended assembly graph, such alternatives are encoded in more detail in the graph itself and considered during the graph search.

In addition, motion costs can be taken into account when searching the extended graph. The graph search directly returns a plan that is guaranteed to be at least nominally feasible without requiring iterative plan verification. Our system is not particularly tuned for efficiency. In the current implementation, approximately 70% of all edges are considered during the graph search, and each edge takes approximately 0.11-0.16 seconds to evaluate.

Task Executive

The executive is responsible for tasking robots to perform tasks that yield the desired structure assembly based on the plan generated. The higher level of the executive coordinates and monitors the overall assembly sequence (sequence-level executive), while the lower level (task-level executive) tasks all robots involved in particular assembly steps (Fig. 2).

Task Template For a given class of problems (assemblies of structures, in our case), the executive contains a task template that describes all possible task instantiations the planner can produce. The template establishes the link between the planner's output and the robots that will execute the plan by parameterizing and instantiating a generic task tree to represent the specific assembly plan.

Fig. 4 shows the task template for structure assembly tasks. At the highest level of abstraction, any assembly we consider is a series of *AssembleComponent* tasks, one for each component that is part of the structure. It is the planner's responsibility to generate a sequence that respects ordering constraints for certain structures (i.e., internal components before external components, etc.). At the next level of detail, each *AssembleComponent* task decomposes into four subtasks. The component needs to be retrieved from its storage location (*RetrieveComponent*), the partial structure may need to be braced and repositioned (*BraceStructure* and *RepositionStructure*, respectively), and, finally, the new component needs to be added to the growing structure (*InstallComponent*). At the lowest level, three behavior tasks accomplish each step. The robot will have to travel through

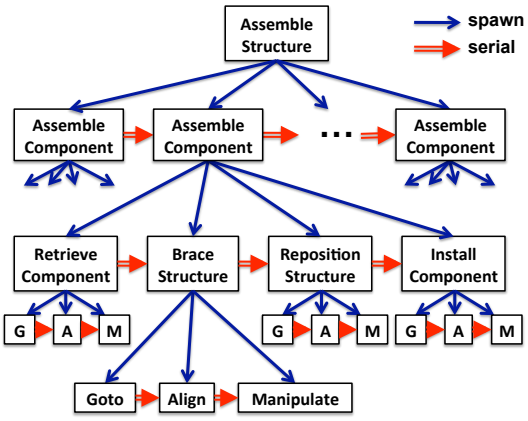

Figure 4: The structure assembly task template. This template is instantiated as a task tree for a specific structure assembly based on planner output.

the environment to the site where the task is to be carried out (*Goto*, *G*), it has to align itself once it gets there (*Align*, *A*) to ensure that all initial conditions for the actual task are met, and then the actual manipulation task can take place (*Manipulate*, *M*). Based on the planner's output, this template is instantiated for a specific task with parameters such as the component involved in each step, motion plans to guide the robot through the environment, connections to establish during manipulation, etc.

Behaviors As described above, all assemblies can be carried out using a concatenation of three (correctly parameterized by the planner according to the template) repeating behaviors: *Goto*, *Align* and *Manipulate*.

Goto's parameters are a series of waypoints the robot should follow to travel through the environment to its next task position. Alternatively, a motion corridor could be provided, within which the robot should travel to its task location. We assume the robots are equipped with the necessary sensors and capabilities for basic navigation.

Align is parameterized by the components involved in the subsequent installation, which define the set of visual markers that need to be visible to be tracked. The goal of this step is to remove any drift due to poor odometry during the previous waypoint following and to ensure the robot is in a proper position for the upcoming installation task.

Manipulate specifies the relative position of the new component relative to the already present structure and the connections to be established to complete the installation.

While we call the third behavior *Manipulate*, a sensing operation could easily be expressed in the same framework, where the range of motion of a manipulator is replaced by the field of view and range of a sensor. Additionally, if subtasks require coordination between multiple robots, appropriate parameters are added to the behavior tasks. Finally, note that the lowest-level tasks are still fairly abstract and require certain capabilities of the robots' behavioral controllers to be executed – the goal of the planner is to provide all parameters required by the behavioral controller so that no manual input or tuning is necessary beyond the implementation of the parameterizable behaviors.

Robots Our current implementation only considers a single mobile manipulator that performs the assembly and an above-the-workspace crane for structure repositioning (a subset of the robots we have available in our test bed, see Fig. 1). In the future, we intend to extend the approach to allow for cooperative strategies where an external sensing agent assists the manipulator with additional sensor information (as shown in Fig. 1), and to consider manipulation of single (large) components by two cooperating manipulators.

Exception Handling The executive is also in charge of responding to execution-time failures (such as robots stopping to avoid an impending collision, required markers not or no longer being in the field of view, and errors occurring during manipulation tasks). Parent tasks are responsible for handling exceptions thrown by their child tasks and to interact with the planner to obtain new parameters or new plans. The exception handling mechanisms built into all levels of the

executive trigger requests to repair individual assembly steps or re-plan entire assembly sequences as described below.

Plan Repair and Re-Planning

The generated assembly plan generated is guaranteed to be nominally feasible – within the assumptions made by the planner about robot shapes and sizes, as well as the robot's ability to move. Specifically, our implementation uses a motion planner that assumes holonomic robot motion. As the real robot (a skid-steer ATRV-2) attempts to follow the planned motions, or as other run-time exceptions occur (e.g., targets out of the field of view of the camera, some error during manipulation tasks, etc.), the system will have to be able to handle exceptions and react to them appropriately.

We consider three distinct levels of failure recovery (see Fig. 2). At the lowest level, as a first recovery attempt, each behavior should have simple contingency responses for things that are known to go wrong from time to time. Often "try again" is a valid recovery strategy. For such contingency recovery actions, the assembly planner never gets involved. However, after a number of contingency attempts fail, more work is required to continue on with the task.

Each failed assembly operation is associated with an edge in the assembly graph. As a first attempt of recovery at the level of the assembly planner, we consider the failed graph edge and attempt to repair this particular step (Fig. 5 (left)). Enforcing the same final condition as in the original plan and taking into account any new information available due to the failure, the planner checks to see if there are alternative parameterizations of the failed task that allow it to repair the plan and then continue on as originally planned. Depending on how far along the assembly step the error occurred, the planner may have different (or none at all) options available to repair a step. If a repair is possible, the affected assembly step is reparameterized and execution continues (until more errors require further repairs).

If no repair is possible (either because the task had already progressed too far to allow for alternative parameterizations while still enforcing the required final condition, or because there is no other way to perform this particular step at the current point in the overall sequence), the exception jumps up to a higher level in the executive, and the planner is queried for a new sequence from the current state of the assembly to the desired goal state. In this case, the offending graph edge is marked impassable, and a new graph search is run from the source state of the failed edge to the original target state (Fig. 5 (right)). Note however, that the failed assembly step left the robot somewhere along its task, possibly carrying a component that it is trying to install. Thus, the re-planned sequence needs to be prepended with some setup tasks that return the robots to a clean state from where to continue on with the new plan. In our case, carried components are returned to their storage location and the braced structure is released.

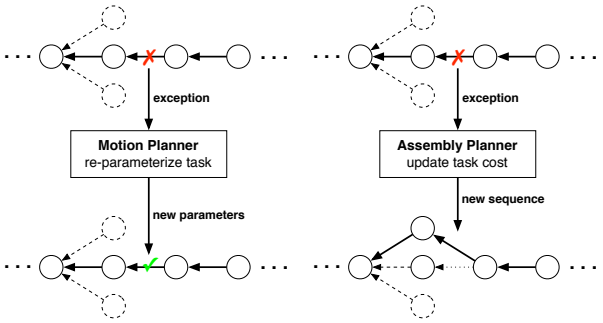

Figure 5: Plan repair (left) and re-planning (right) in response to execution-time failures.

The Planner at Work

In this section, we describe a number of scenarios for which we have used the assembly planning framework we described to generate plans for simulated assembly scenarios. We use a simulator/visualizer based on OpenRAVE (http://openrave.programmingvision.com/) to examine the output of the planner and to exercise the system's repair and re-planning capabilities. The executive that interacts with the planner and tasks the robots is implemented using the Task Description Language (TDL, (Simmons and Apfelbaum 1998)) and is the same for the simulated robot case as it will be once we move to planning for real robots.

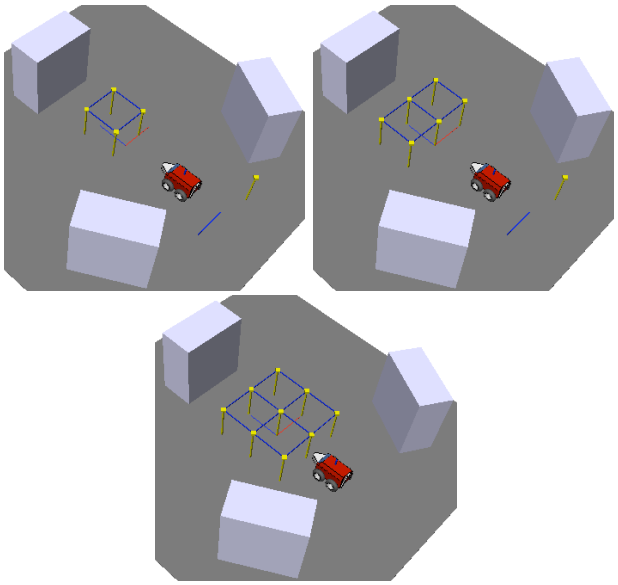

Figure 6: Three simulated assembly scenarios.

Scenario 1: Single Square

As the base case to demonstrate the planner's capabilities, we chose a scenario we had worked with previously with real robots assembling a square of four nodes and four beams (Fig. 6 (upper left)) according to a hand-written script (Heger et al. 2005; Sellner et al. 2006).

For this scenario, the base graph representation contains 58 vertices and 112 edges, and there are 512 potential candidate traces through the graph. Expanding the graph to include robot positions around the structure, the assembly graph grows to 246 vertices, 924 edges and $1.3 \cdot 10^5$ candidate assembly sequences.

Clearly it would be impossible for an operator to truly evaluate that many sequences and choose the best one to encode in a static script. Fortunately that was not necessary, as the structure was simple enough and we had enough domain knowledge and experience with the robots to select a good sequence. Never the less, our assembly planner successfully generated assembly plans with different initial conditions in under 80 seconds. For comparison, the real robots take 30 minutes and more to assemble the square.

With the desired structures becoming larger, the size of the assembly graphs and with it the number of potential plans grows rapidly. As the structures become larger and more complicated, finding good plans out of the vast number of potential ones becomes increasingly difficult even for skilled operators, and an automated system like the planner described in this paper becomes highly desirable.

Scenario 2: Two Squares

Increasing the complexity of the goal structure slightly by adding a second square (Fig. 6 (upper right)), we also introduce an internal component that was not previously present. Here the planner has to discover that the center beam between the two squares needs to be installed before the outer components all are in place.

Still quite a simple structure, the two-square scenario requires a graph representation of 408 (2,117) vertices and 1,187 (10,388) edges for the base (extended) graph representations, respectively, with $1.2{\cdot}10^6$ ($3.5{\cdot}10^9$) potential candidate traces through the graphs. The growing number of edges is of concern here, since that is where the planner has to do its work when evaluating assembly steps (i.e., run a motion planner to evaluate feasibility and goodness). While the graphs are not exceptionally large (at least for the structures considered here), the graph search can potentially take quite some time to complete the evaluation. In our experiments, the graph search required an average of 998 seconds to evaluate 7224 graph edges. We argue that in the end, spending the time up-front during planning to make sure no robot moves until the plan is at least nominally feasible is worth the effort as otherwise very expensive physical backtracking is unavoidable.

Scenario 3: Four-Square Lattice

Continuing the expansion of the structure, we arrive at a four-square lattice (Fig. 6 (bottom)). That is the largest structure we can build with the hardware we have available in our test bed, and we are working toward being able to show this assembly using real robots. It is also the largest structure the current implementation of the assembly planner can handle to represent. With this planner and in simulation, we plan to extend the size of the planable structure to 100 components and beyond in the future.

This 21-component structure assembly is represented by graphs of 9,397 (65,566) vertices and 44,646 (460,511) edges for the base (extended) graph, with $1.1{\cdot}10^{13}$ ($6.5{\cdot}10^{17}$) potential ways to assemble the structure. This progression in graph sizes for still fairly simple structures clearly indicates that just continuing in this manner does not scale sufficiently to be able to plan for structures of 100+ components. In order to solve this problem, we are currently investigating approaches that avoid constructing the entire assembly graph explicitly but rather only construct as much graph as is needed to find a solution.

Real-Robot Scenario

We have also used the task parameterization through motion planning capabilities of our system for a space-themed assembly project as part of an STTR effort with Metrica Inc. For that project's scenario, two robots have to separately retrieve two modules from the environment and place them into a larger container. They then have to cooperatively move this container through the environment to some desired final location (Fig. 7).

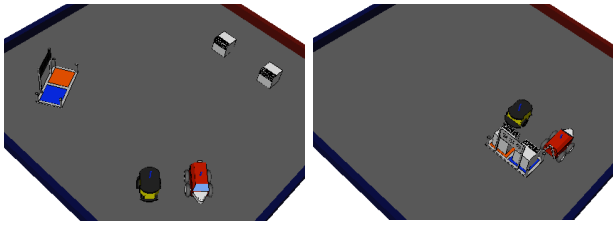

Figure 7: Modules have to be transported by robots and placed into a large container. The container then has to be moved cooperatively by the robots to a designated final location in the environment.

While the high-level plan in this case is given (NASA is patient enough and the problem is small enough to do it by hand), the instantiation of the specific motions through the environment to execute the plan are left to the planning system. In future generations of this work, one could imagine a system that, in the case of an execution-time failure, could pre-evaluate potential recovery strategies, score them for goodness and present alternatives to a remote operator to choose from in a sliding autonomy interaction scheme.

Discussion

Our work on the assembly planner to date has been focused on enabling it to plan for realistic structures like the ones we can build in our test bed and to provide recovery capabilities for execution-time failures that we must anticipate with any real-robot system. Given that the robots' speed of operation during assembly is fairly slow, optimizing for efficiency has been a lower priority than increasing system robustness.

While we are currently using quite simple motion planning strategies, our observations of the system in operation suggest that a lot of the time these approaches are sufficient. In the cases where they are not, robots are able to recognize that they are in an off-nominal state, safely stop and throw an exception requesting help (from an autonomous planner or a human operator). Autonomous recovery seems to be the place to introduce more sophisticated motion planning techniques. There they could leverage their greater capabilities without requiring additional computation throughout the system that may not be necessary for the robots to succeed.

A concern during re-planning is that robots have to "physically backtrack" to restore the system to a clean state in the

assembly graph before continuing toward their goal. These are in fact the situations where our system can benefit the most from assistance from an operator. While even sophisticated motion planners might have trouble in tight spaces or similarly challenging situations, a human seeing the bigger picture can often easily get the robots back on track.

Summary

We have presented a graph-based planner for assembly problems that combines symbolic graph-search over valid intermediate structure configurations with motion planning techniques to evaluate nominal feasibility of the considered actions. The result is an assembly plan that contains all parameters necessary to instantiate a generic task template into a task tree that can be executed directly by mobile assembly robots without additional operator intervention. Furthermore, as execution-time exceptions occur, the executive controlling the robots interacts with the planner to repair and replan the assembly sequence as necessary and appropriate.

This work advances the field of planning toward more useful robot operation that involve physical interaction with the real world. While traditional symbolic planners do not have a concept of physical feasibility and interaction with the environment, strictly motion-planning solutions for assembly tasks require a way to specify intermediate goals (simply specifying the initial, disassembled state and the desired, fully assembled state yields an intractable problem for all but the simplest structures). Our hybrid approach accomplishes just that. The graph structure supplies intermediate goals for a motion planner, and the motion planner, in return, provides meaningful information about the feasibility and goodness of actions in the workspace.

While we have made some progress toward a system as described in our vision at the beginning, many issues still remain. We are actively working toward scaling our approach to larger structures (on the order of 100 components) by using an incremental graph search strategy where the assembly graph is constructed only as required during the graph search. We are also focusing our attention on robust plan repair and re-planning capabilities, as those will be essential when planning for real-robot systems (we plan to integrate this work with the robots and assembly tasks shown in Fig. 1). Finally, cooperative strategies with multiple robots will have to be considered to advance beyond the single-robot-that-does-everything solution we present in this paper.

References

Bozma, H. I., and Koditschek, D. E. 2001. Assembly as a Noncooperative Game of its Pieces: Analysis of 1D Sphere Assemblies. *Robotica* 19:93–108.

Fahlman, S. E. 1973. A Planning System for Robot Construction Task. Technical Report AITR-283, MIT Artificial Intelligence Laboratory.

Gravot, F.; Alami, R.; and Siméon, T. 2002. Playing with Several Roadmaps to Solve Manipulation Problems. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

Heger, F. W.; Hiatt, L. M.; Sellner, B.; Simmons, R.; and Singh, S. 2005. Results in Sliding Autonomy for Multi-Robot Spatial Assembly. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*.

Heger, F. W. 2008. Generating Robust Assembly Plans in Constrained Environments. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*.

Homem de Mello, L. S., and Sanderson, A. C. 1988. Autommatic Generation of Mechanical Assembly Sequences. Technical Report CMU-RI-TR-88-19, Carnegie Mellon University, The Robotics Institute, Pittsburgh, PA.

Homem de Mello, L. S., and Sanderson, A. C. 1991. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Transactions on Robotics and Automation* 7(2):228–240.

Homem de Mello, L. S. 1989. *Task Sequence Planning for Robotic Assembly*. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA.

Karagöz, C. S.; Bozma, H. I.; and Koditschek, D. E. 2004. Feedback-Based Event-Driven Parts Moving. *IEEE Transactions on Robotics* 20(6):1012–1018.

Kaufman, S. G.; Wilson, R. H.; Jones, R. E.; Calton, T. L.; and Ames, A. L. 1996. The Archimedes 2 Mechanical Assembly Planning System. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 4, 3361–3368.

Klavins, E. 2006. Self-Assembly From the Point of View of its Pieces. In *Proceedings of the American Control Conference (ACC)*.

Lengyel, J.; Reichert, M.; Donald, B. R.; and Greenberg, D. P. 1990. Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. *Computer Graphics* 24(4):327–335.

Nielsen, C. L., and Kavraki, L. E. 2000. A Two Level Fuzzy PRM for Manipulation Planning. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

Sellner, B.; Heger, F. W.; Hiatt, L. M.; Simmons, R.; and Singh, S. 2006. Coordinated Multi-Agent Teams and Sliding Autonomy for Large-Scale Assembly. *Proceedings of the IEEE* 94(7).

Simmons, R., and Apfelbaum, D. 1998. A Task Description Language for Robot Control. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

Stilman, M., and Kuffner, J. J. 2004. Navigation Among Movable Obstacles: Real-Time Reasoning in Complex Environments. In *Proceedings of the International Conference on Humanoid Robotics (Humanoids)*.

Stroupe, A.; Huntsberger, T.; Okon, A.; and Aghazarian, H. 2005. Precision Manipulation With Cooperative Robots. In Parker, L.; Schneider, F.; and Schultz, A., eds., *Multi-Robot Systems: From Swarms to Intelligent Automata*. Springer.

Younes, H. L. S., and Simmons, R. 2003. VHPOP: Versatile Heuristic Partial Order Planner. *Journal of Artificial Intelligence Research* 20:405–430.