Collision-Aware Assembly Planning

Tesca Fitzgerald Andrew Price Laura Strickland Zhefan Ye

***Abstract*— We present a collision-aware approach to assembly planning using a robotic manipulator arm equipped with a gripping endeffector tool. Planning of the manipulator's motion is accomplished using RRT-Connect. Computing the sequence in which components in the assembly should be placed in the assembly takes the size and shape of the gripper holding each of the parts to be placed in the assembly into account. By looking at how each part must be placed with respect to other parts in the assembly, as well as where the gripper may come into collision with the parts that will have already been placed up to that point, our planning system generates the sequence in which the parts should be assembled that best avoids collision between the gripper and the parts already in the assembly. Results are presented from a simulated manipulator arm that uses this approach to successfully plan and execute the assembly of a simple LEGO model.**

I. Motivation

"A good assembly process plan can increase the efficiency and quality, and decrease the cost and time of the whole product manufacturing process." [1]. The ability to produce inexpensive, high-quality products is a major feature of the modern developed economies (see [2] for a listing of per-capita GDP). Bearing these concerns in mind, it is easy to see how important efficient, automated production processes can be to the wealth of modern nations.

As this is a research frontier closely tied to the bottom lines of both corporations and governments, much work has been done to generate assembly sequences consumable by production machines from human-generated design documents. This is frequently accomplished by generating a precedence graph showing the high-level dependencies between parts, then using low-level skill primitives to actually achieve the assembly design. One area that we feel we may improve upon is augmenting this graph to show relationships imposed not by the interactions between the parts themselves, but between the parts and the tool(s) used to assemble them. Thus, we hope to show that certain high-level assembly tasks may be inadmissable or require action modification to successfully achieve the desired design.

In this paper, we discuss our approach to assembly planning, which combines traditional manipulator motion planning with collision-aware part sequence planning. Plans are generated that not only avoid manipulator self-collisions and collisions between the manipulator and the assembly-in-progress, but that also take into account that the endeffector gripper holding the next part to be placed may not collide with parts already in place. We have implemented this system in simulation to assemble a LEGO model as shown in Figure 1. While the LEGO model used in testing is a simple model requiring only placement of parts onto a plane parallel to the horizontal ground plane, and while model requires no two-handed assembly steps, we believe this is an excellent starting point for further collision-aware assembly planning development.

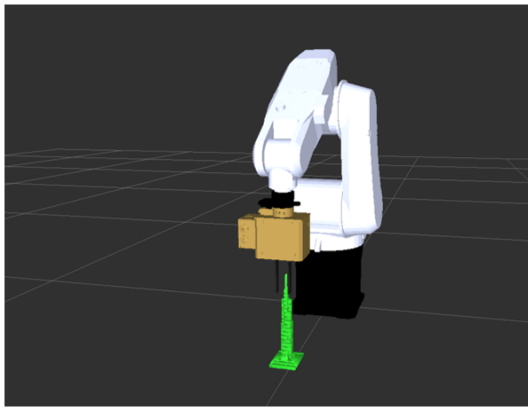


Fig. 1: KUKA KR5sixxR650WP arm (white) and completed LEGO brick model (green)

II. Related Work

As assembly planning is a vital step in many manufacturing processes, much work has already been done on planning for assembly and grasp planning. Common methods for planning sequences of part placement in assembly tasks include Assembly Sequence Planning (ASP), graph- and tree-search approaches, stochastic optimization methods, and partial-ordered planning approaches.

A. Assembly Sequence Planning

ASP is the selection of an ordered sequence of actions that successfully assemble a specific product. Common topics in the literature surrounding ASP include constraints and geometry concerns related to assembly sequencing and tool usage, optimization, and generalization.

1) Geometric Concerns in Assembly Planning: Graphical methods are often used to represent assemblies and how they may be assembled. Both [3] and [4] discuss the use of Non-Directional Blocking Graphs (NDBG) in ASP. The NDBG of an assembly represents the assembly's internal structure and describes possible interactions between parts [3]. An NDBG may be used to estimate assembly complexity based on the constraints at each step and generate assembly sequences [3]. In [3], Wilson, et al. discuss how the complexity of an assembly sequence may be measured and classified. Plans for some special cases of assemblies, such as sequences consisting entirely of one-step translations, may be computed in polynomial time [3] [4].

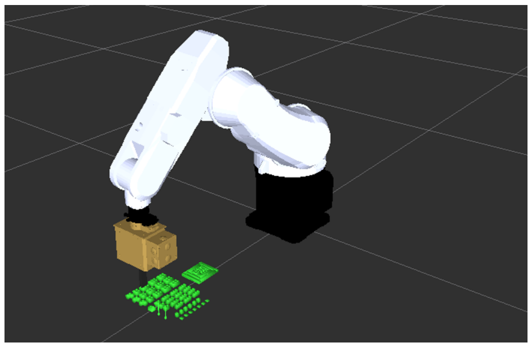
Geometric constraints on tool usage must often be taken into consideration. In [5], Wilson presents a framework for general assembly procedures to determining whether the use of some tool is feasible for some assembly step. This framework considers the space needed for the tool to be used, collision avoidance, and how the tool interacts with a part [5].

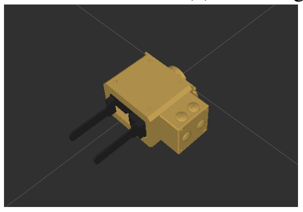
2) Automated Assembly Planning: Wang, et al. discuss several approaches to Automated Assembly Planning [1]. Automated assembly planning can be approached with two primary methods. Exact, or enumerative methods utilize tree- or graph-search to choose the best sequence of assembly steps. While exact methods can find the optimal assembly plan for an assembly process, this method does not scale well for large, complex assemblies with many components [1]. Heuristic methods do not guarantee an optimal sequence, but are much more efficient. Yet heuristic methods are prone to getting stuck in local optima, requiring algorithms that allow escape from local optima, such as stochastic optimization methods [1].

3) Stochastic Optimization: Various stochastic optimization methods may be used to prevent heuristic assembly sequence planning methods from getting stuck in local minima. Simulated annealing, such as presented in [6], is often used. Hong and Cho define an energy function used in the simulated annealing process based on the assembly cost and assembly constraints associated with a proposed assembly sequence [6]. The assembly sequence for which the cost is minimized is considered optimal for the assembly.

Ant Colony Optimization (ACO) is a stochastic optimization method used in the planning of assembly sequences based on the behavior of ant colonies [7]. In nature, ants deposit pheremones along their paths of travel, such as to denote a path that leads to a food source. When an ant finds a pheremone trail, it decides whether to follow the trail, leaving its own pheremone trail wherever it goes. If it follows the trail it found, that path's scent is strengthened, as is the likelihood of other ants to follow it. Similarly, in ACO, "ants" are placed at random locations on a graph comprised of possible disassembly steps for the finished product [7]. The ants build disassembly sequences from the graph, which may or may not be geometrically feasible. A pheremone matrix is constructed to track the frequency with which each edge of the graph is visited. The "best" sequence has the highest pheremone count [7].

4) Partial-Ordered Planning: Work has also been done to generalize plans by using partial-order planning. In [8], plans for assembly are generated from the disassembly sequence of a CAD model, and a plan is chosen based on user criteria. These plans are broken down into generalized steps, such as the translation of the endeffector, the rotation of a part, or the use of a sensor or tool. These generalized steps are then defined as lists of commands for a specific model of robot that will execute the assembly task [8]. The robot, equipped with these definitions, may use the generalized plan to execute the assembly task, given that all necessary constraints are met [8].


(a) Planning Workspace


(b) SCHUNK PG70 gripper

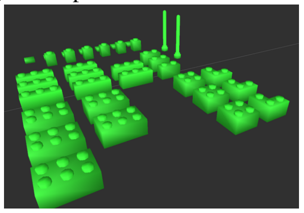

(c) LEGO bricks

Fig. 2: Overview of the workspace, detailed view of gripper and Lego bricks.

III. Collision-Aware Planning

While completing an assembly task, it is vital that a robot arm avoid collisions with itself and with other objects in the world. Our method plans collision-free arm trajectories and a sequence for part assembly that avoids collisions between the gripper and already-assembled parts.

A. Collision Checking

Motion planning is crucial to enable robots to move safely in real world environments while avoiding obstacles. It relies on accurate and fast collision checking to know whether poses of the robot in the world are in collision or not. Since motion planners often have to deal with partial or noisy information about the environment in which they operate, we used the Flexible Collision Library (FCL)[1] to check for collisions, as it is well-suited to address these limitations. FCL integrates several techniques, such as octree representation, for fast and accurate collision checking and proximity computation. It is based on hierarchical representations and is designed to perform multiple proximity queries on different model representations. FCL can perform collision checking for rigid objects, point clouds, deformable objects, and articulated objects [9]. Those features allow us to detect collisions for the robot arm (an articulated object) and rigid objects.

B. Construction Element Sequence Planner

1) Motivation: The first step of this planning process uses classical planning to derive an total-order plan from the

[1]http://wiki.ros.org/fcl

partial-order plan provided by the model building instructions. The provided instructions consist of a series of steps, each of which contains brick-placing actions that should be performed within that step, but without a specified order. While a human may easily use these instructions to construct a model, this is in part due to our ability to manipulate small objects dexterously. A robot, however, may not be able to manipulate pieces with this same ability, and is thus faced with a set of restrictions, such as grasping each object such that it can be placed in the goal location without the gripper hitting already-placed objects. As a result, the total-order plan provides the robot with two benefits:

1) Provides a fully ordered list of brick-placing actions
2) Provides a grip position for each action such that the item can be placed without the robot's gripper hitting already-placed objects in the process

2) Method: Since a partial-order plan is provided in the instructions, the fully-ordered planning occurs to order actions in each step of the provided plan as follows.

```
function OrderActions($ordered$, $unordered$)
    if $Size(unordered) = 0$ then
        return $ordered$
    end if
    for $a$ in $unordered$ do
        $grasps \leftarrow GetGraspPositions(a)$
        for $g$ in $grasps$ do
            $lGrasp \leftarrow g[0]$
            $rGrasp \leftarrow g[1]$
            for $block$ in $ordered$ do
                if $Contains(block, lGrasp, rGrasp)$ then
                    $collision \leftarrow True$
                end if
            end for
            if $collision = False$ then
                $action \leftarrow [a, g]$
                $ordered \leftarrow ordered + action$
                $unordered \leftarrow unordered - a$
                return $OrderActions(ordered, unordered)$
            end if
        end for
    end for
end function
```

For each step, the above function is called, which tests that each object can be placed at one of the specified grasp positions. If an object can be grasped, it is placed into the ordered plan and the next object is then tested.

The function $Contains(brick, leftGrasp, rightGrasp)$ detects whether grasping the brick at either of the given grasp points will interfere with any of the already-placed bricks. Specifically, it checks that each point does not touch any object at the same height. Objects that have been placed at a lower height cannot collide with the robot's gripper, and thus do not interfere with any grasp position.

Initially, the planner was written to reorder a series of actions if a situation were to arise in which a block is difficult to place, and cannot be placed using any of the available

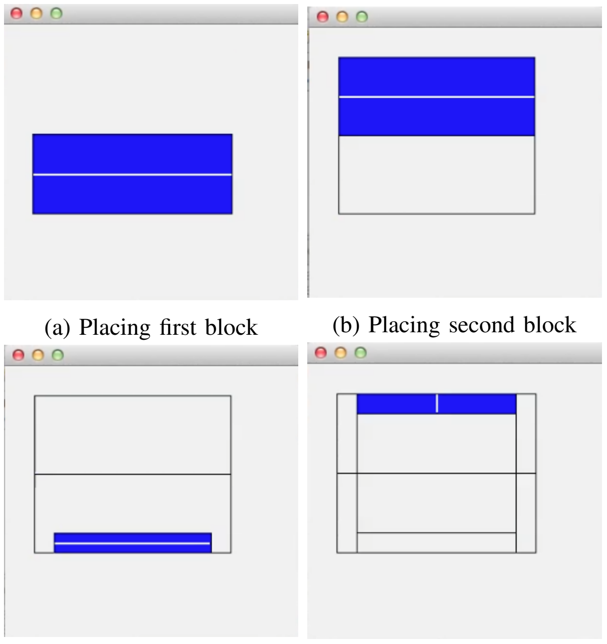

(a) Placing first block (b) Placing second block (c) Stacking block vertically (d) Alternate grasp

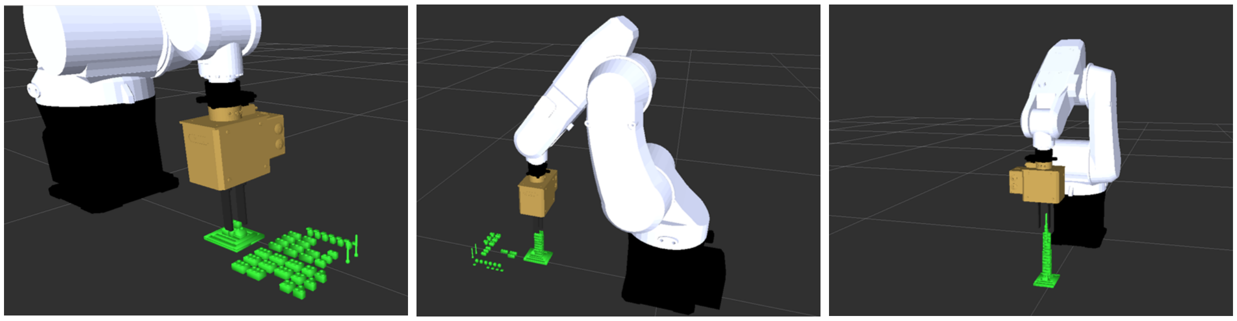
Fig. 3: Overhead View of Planner's Model Construction

grasp points. By reordering the set of actions in that step, the difficult-to-place block could be placed first, with the remaining actions following. However, several steps in the provided instructions that cannot be completed using a two-point grasp without colliding with an already-placed block. Thus, we defined three possible grasp configurations: a two-point grasp which contacts the block at both the left and right sides along the same x-axis; a two-point grasp which contacts the block at both the far and near sides along the same y-axis; and a one-point grasp which contacts the block at the upward-facing surface. The last grasp would require a more specialized tool to drop the block into place from overhead, similar to how we might slide and press a block into a goal location that is difficult to reach. As a result, the planner only uses this third grasp if neither of the other grasps are possible without colliding with the model.

3) Classical Planning Results: The result of this classical planner is a fully-ordered set of actions written in the same format as the original model building instructions, but with a number at the end of each action statement to indicate the index of the grasp type. This resulting plan is then provided to the motion planner, which plans to move the specified object to the planned goal location using the specified grasp position.

Figure 3 depicts a top-down view of four stages along the classical planner's model construction. Figure 3a illustrates the first block placement, which is a large LEGO plate highlighted in blue. The white line represents the grasp configuration, where the two end-points of the line are the gripper's points of contact with the LEGO piece.

Figure 3b illustrates the second block placement, also highlighted in blue. The previously-placed object is also



(a) Building state 1 (b) Building state 2 (c) Building state 3

Fig. 4: Sample images from different building states. Figure 4c shows the completed state.

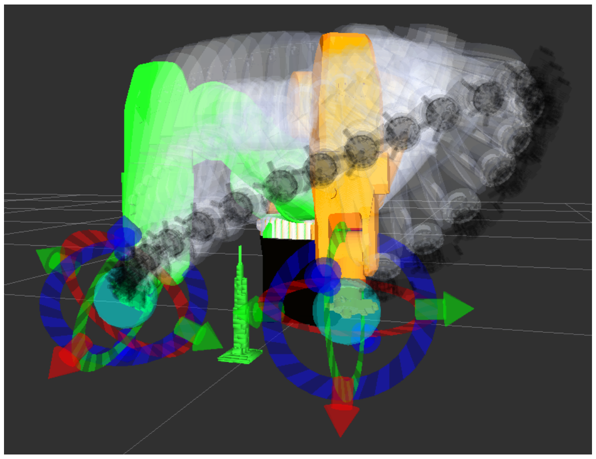


Fig. 5: MoveIt planner leveraging OMPL's RRT-Connect implementation to avoid the completed tower.

outlined in this figure, but is no longer highlighted. The third building stage, Figure 3c, is an example of the first stacked LEGO piece, which is placed vertically over a previously-placed piece. Figure 3d shows a later building stage, in which the highlighted piece cannot be grasped left-to-right without colliding with the two previously placed parts; as a result, a top-bottom grasp is used instead, as illustrated by the white line.

C. *Motion Planning*

We used the Open Motion Planning Library (OMPL) [10] to plan the trajectory for assembly. OMPL is a sampling-based motion planner, and it contains implementations of many state-of-the-art planning algorithms, such as Probablistic Road Method (PRM) [11], Rapidly-expanding Random Trees (RRT) [12], RRT-Connect [13], and Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) [14]. For our project, given a start pose A, a goal pose B, and a URDF description of a robot arm, OMPL is able to find a motion plan using RRT-Connect, a bi-directional RRT method, to guide the robot arm to move from A to B with a valid trajectory.

IV. Results

We tested the combination of the construction element sequence planner and OMPL to devise a possible construction sequence for a simple LEGO tower in simulation, demonstrating the viability of our method for performing simple assembly operations. A model of the KUKA KR5sixxR650WP arm equipped with a SCHUNK PG70 endeffector gripper was used in visualization of the plan implementation.

Figure 2a shows the simulated world, which includes the robot arm, the gripper and LEGO bricks.

Figure 2 shows the robot gripper and LEGO bricks in details. Figure 4 shows the sample images from different building states. Note that Figure 4c shows the final state of our LEGO building.

Potential directions for future work include testing additional assembly tasks in simulation, such as additional and more complex LEGO models. Beyond simulation testing, the presented method could be implemented on a robot arm to test real-world performance.

V. Conclusion

In this paper, we present a method of assembly planning for manipulator arms completing an assembly task. This method uses the possible grasps for each part to be assembled and the resulting size and shape of the gripper holding the part, as well as what other parts could already be in the assembly, to plan the order in which parts are assembled. The order is specifically chosen to avoid collisions between the gripper (holding the part to be placed) and already-assembled parts. This is complemented by the use of OMPL to plan the arm's trajectory and FCL to check for collisions.

The presented method was tested in simulation, with a simulated robot arm assembling a simple LEGO model. The part-sequence-planning algorithm was successful in determining the order of assembly, as the gripper did not collide with any of the pieces already in the assembly.

VI. Breakdown of Work

A. *Tesca Fitzgerald*

- Implemented classical planning algorithm
- Wrote object grasp visualizer

B. Andrew Price

- Created 3D models and visualizations
- Configured MoveIt motion planner for use with Kuka arm.

C. Laura Strickland

- Wrote file I/O for LDraw files
- Worked on implementation and integration of motion planning node

D. Zhefan Ye

- Worked on implementation and integration of motion planning node
- Created video and image media

VII. Appendix

A. Repositories

Source code for the work discussed herein and instructions for how the simulation and related code may be run is available in the following repositories:

- Repository containing motion planning for construction element sequencing and arm motion: `https://github.gatech.edu/CS7649-HW1-AwesomeGroup/everything_is_awesome`
- Integration of MoveIt! package into simulation: `https://github.gatech.edu/CS7649-HW1-AwesomeGroup/kuka_kr5sixx_moveit`
- Description of KR5sixxR650WP manipulator arm: `https://github.com/a-price/KR5sixxR650WP_description/tree/GTRI`
- Description of SCHUNK PG70 gripper: `https://github.com/a-price/schunkPG70_description/tree/GTRI`

B. Videos

A video showing the assembly sequence plan generated by the construction element sequence planner: `http://youtu.be/xmkYb9cUIzg` Another video shows the building process of LEGO Sears Tower: `http://youtu.be/FWt_a5A4tt4`

References

[1] Lihui Wang, Shadi Keshavarzmanesh, Hsi-Yung Feng, and RalphO. Buchal. Assembly process planning and its future in collaborative manufacturing: a review. *The International Journal of Advanced Manufacturing Technology*, 41(1-2):132–144, 2009.

[2] The World Bank. Gdp per capita, ppp (current international $). `http://data.worldbank.org/indicator/NY.GDP.PCAP.PP.CD`, 2014. Accessed 2014-10-30.

[3] Randall H. Wilson and Jean-Claude Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371 – 396, 1994.

[4] D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26(3-4):577–601, 2000.

[5] Randall H Wilson. A framework for geometric reasoning about tools in assembly. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1837–1844. IEEE, 1996.

[6] DS Hong and HS Cho. Generation of robotic assembly sequences using a simulated annealing. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 2, pages 1247–1252. IEEE, 1999.

[7] J.F. Wang, J.H. Liu, and Y.F. Zhong. A novel ant colony algorithm for assembly sequence planning. *The International Journal of Advanced Manufacturing Technology*, 25(11-12):1137–1143, 2005.

[8] U. Thomas and F.M. Wahl. A system for automatic planning, evaluation and execution of assembly sequences for industrial robots. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 3, pages 1458–1464 vol.3, 2001.

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

[10] Ioan A \cSucan, Mark Moll, and Lydia E Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012.

[11] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[12] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[13] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[14] Ioan Sucan and Lydia E Kavraki. A sampling-based tree planner for systems with complex dynamics. *Robotics, IEEE Transactions on*, 28(1):116–131, 2012.