Proceedings of ASME 2019 Dynamic Systems and Control Conference
DSCC 2019
October 8-11, 2019, Park City, UT, USA

DSCC2019-9173

ASSEMBLY PLANNING USING A TWO-ARM SYSTEM FOR POLYGONAL FURNITURE

Seth T. Payne[a], C. Fletcher Garrison IV[a], Steven E. Markham, Tucker Hermans[b,*] and Kam K. Leang[a,†]
[a]Design, Automation, Robotics, and Control (DARC) Lab, Dept. of Mechanical Engineering
[b]Learning Lab for Manipulation Autonomy (LL4MA), School of Computing
Robotics Center, University of Utah
Salt Lake City, Utah, 84112

ABSTRACT

This paper focuses on the assembly planning process for constructing polygonal furniture (such as cabinets, speakers, bookshelves, etc.) using robotic arms and manipulators. An algorithm is described that utilizes easily-implemented and generally-accepted motion planning algorithms to take advantage of the polygonal nature of the furniture, which reduces the complexity of the assembly planner. In particular, the algorithm disassembles a given CAD model in simulation to find a valid assembly order and disassembly path, then implements that assembly order with two robotic arms, using the disassembly path as the finishing path of the part into the assembly. Additionally, it finds a collision-free plan developed for each of the arms in the correct assembly order with the final result being the assembly of the model.

INTRODUCTION

Robotics-based automation of the assembly process for various products including furniture has attracted significant attention for improving throughput and lowering production costs [1]. For this reason, the field of Assembly Planning (AP) has received substantial attention in recent years [2]. With furniture such as cabinets, speakers, bookshelves, or boxes, an assembly planner can take advantage of the polygonal nature of furniture, for example, to reduce the complexity of the assembly. Described herein is an easily-implemented algorithm that enables the assembly of polygonal furniture when given the computer-aided design (CAD) file of the desired furniture. Robotics-based construction of polygonal furniture is common [3, 4], likewise for assembling printed circuit boards [5], in the automotive manufacturing sector [6], and many other manufacturing environments [7]. This algorithm is well suited for different manufacturing processes which require assembly planning to produce fully autonomous robotic assembly systems.

The field of Assembly Sequence Planning (ASP) is a subdivision of AP and significant research has been done to improve current assembly sequence planners. Work by Belhadj et al. [8], Morato et al. [9], and others [10, 11] focus on finding solutions for this assembly sequence problem. Other researchers such as Dogar et al. [12, 13] assume that the sequence of assembly has been performed previously and therefore focus on the challenges faced in the physical assembly of the product. Knepper et al. [4] propose a solution which combines these two areas of research and offer one process that answers both issues. The algorithm proposed in this paper fits into this last category in that it both discovers an assembly sequence and also an assembly plan which can be implemented by robotic arms. At the same time, it uses easily-implemented and generally-accepted algorithms to accomplish this task, which decreases the complexity of implementing this algorithm in a real-world scenario.

The proposed approach involves two main steps: (1) discovery of a viable assembly order and (2) assembly of the work piece. The contribution of the technique proposed is twofold:

*Email: thermans@cs.utah.edu;
†Email: kam.k.leang@utah.edu.

Copyright © 2019 by ASME

1. uses the disassembly path in the assembly path, which follows the assembly by the disassembly philosophy [14, 15] and
2. offers a novel, "start-to-finish" assembly algorithm which employs easily-implemented and generally-accepted motion planning techniques.

In particular, the algorithm finds a path to remove each part from the assembly, called the disassembly path, which is then used to "reassemble" the product as a part of the robot's assembly path. Concerning contribution number two, the proposed algorithm uses an iterative-deepening depth-first search (IDDFS) [16] to search through possible assembly sequences and uses a rapidly-exploring random tree (RRT) [17, 18] to assess the plausibility of the motion of parts and robotic arms through the workspace. Both of these methods of search are widely accepted and generally used [16–23].

The proposed algorithm is designed for polygonal furniture, but may be expanded to other polygonal assemblies. Furthermore, the algorithm was designed such that parts would be cut from a flat piece of material, such as Medium Density Fiberboard (MDF), then moved to an assembly area, where two robots could assemble the pieces into their proper positions, based on the provided CAD model.

This algorithm was tested using the UR5 robot arm in the 3D simulation software, VREP, but it can be adapted for use with any robot arm in any simulation software or other environment.

RELATED WORK

Several assembly planners have been created for different tasks. Wang et al. [24] applied ideas from the food-tracking abilities of ant colonies. In this work, Wang et al. present a method of assembly planning that mimics the ability of the colony to find food (the assembly plan), then find the shortest path to that food. Using the positive feedback process, distributed computation, and the greedy constructive heuristic search, Wang et al. present a method to finding assembly sequences that are "optimal or near-optimal".

Morato et al. [9] use assembly geometry and part proximity to define part clusters within the assembly, then apply a motion planning algorithm to determine which part clusters can be removed from the assembly without collision with other parts. Specifically, Morato et al. use a variation of RRT to assess the feasibility of removing each part. The work presented in this paper builds on that of Morato et al. by using the path found while assessing assembly sequence feasibility as a forward path for the part during assembly. This is referred to as the disassembly path. This paper also uses motion planning to generate plans for robot arms to assemble the work piece in a simulated industrial environment, where possible collision with robotic arms and parts must be addressed. The work presented by Morato et al. assumes that each part or part subassembly is a free-flying object, thus grasping and robotic arm movement is not considered. Therefore, this paper addresses a more direct connection between ASP and AP, using knowledge gained during ASP to decrease the computation time of the assembly plan, while also including a more complete presentation of the entire problem of AP, from individual parts to assembled product.

Knepper et al. [4] use a geometric planner to create a sensible assembly of the parts given only the parts as a guide. This geometric planner aligns holes in each part with holes in other parts, finding which plan uses all of the holes to connect parts together in the end assembly. The planning space is searched depth first through all plausible mating of the pieces until a final form is reached. This is then translated into a large set of small manipulation actions by a second planner. The paper also discusses several interchangeable tool sets for performing all operations. Finally, an algorithm is developed and presented for stringing the entire plan together and determining success or failure.

Knepper et al. [11] use this same method to create several possible assembly orders from one set of parts. The algorithm proposed in this paper addresses assemblies of parts where the method of attachment can be arbitrary, meaning the parts can be assembled using glue, screws, nails, or other methods, and thus is not restricted by using dowels to assemble the part.

Research performed by Dogar et al. [13] proposed modeling the assembly problem in much the same way humans do. During operation, the robots must decide whether it is more advantageous to maintain a grasp and transfer the entire work piece to the next operation, or to let go and grasp at a new location advantageous to the next step. This problem can be formulated as a constraint satisfaction problem (CSP) [25] where they take as inputs, the set of state variables, the domain of the variables and the set of constraints. Using this process Dogar et al. propose an algorithm for planning the sequences of grasps and carries necessary to complete the assembly. Additionally, this paper outlines previous work done to determine how to place delicate members such as pegs and nails, which could be implemented into the algorithm proposed in this paper to secure the pieces assembled by the robots.

Wan et al. [10] focus mostly on 3D manipulations and algorithms for obtaining the best motion plan to fit all components together, taking into account the different grasping situations depending on the component. The work also presents a method to check the stability of the assembly as parts are placed within it.

Dogar et al. also cover various stages of assembly [12], starting with moving a part to a different area of the manufacturing floor, then aligning several pieces and using fasteners to attach the parts. Specifically, Dogar et al. used perception to coordinate the movements of several robots and the position of the parts to be assembled. This paper discusses multi-scale perception, or planning on large scales to move the part from location to location, then on a smaller scale to align the parts for fastening.

Copyright © 2019 by ASME

This paper addresses a“start-to-finish” algorithm which can be used as a structure for implementation of much of the work proposed above. However, the algorithm proposes a novel method of finding a path for each part into the assembly by using the disassembly path as part of the assembly path, which moves the parts away from the assembly using the geometric center. This method helps prevent using high-cost, precise movements that could be necessary for placing a part in an exact pose without collision with other parts in the assembly. Other parts of the algorithm proposed here use standard methods in series with one another, which methods have been proven effective in many other applications [16–23].

TECHNICAL APPROACH

The goal of this algorithm is to allow a system of two robotic manipulators to assemble a work piece based on a blueprint provided by the user. The plan should be generated and returned to the user in a reasonable amount of time. If the algorithm cannot generate a plan it should return “No Plan Possible”. The first step in this process is to consider the simulated assembly environment used build the work piece, which is described next.

Assembly Environment

The first consideration is the limitation of the simulation. The simulation used to test the algorithm does not include the physics of movement of the pieces. This means that the robotic arm can manipulate any piece without regard to dropping, slipping, or shifting work pieces. Each robot is equipped with a vacuum gripper to hold each piece, which means the work piece is assumed to be non-porous. This algorithm also does not consider the weight of the piece to be moved and if the arm used in the simulation is physically capable to manipulate the piece.

Now that the limitations of the simulation are defined, the assembly area must be defined as well. This algorithm was implemented in a large, open area in which the robots can move without fear of collision to random objects, which is a similar environment to the industrial robots that use fences surrounding their workspace. Furthermore, it is important to assume that the individual pieces needed for assembly are readily accessible to the robots. This could be accomplished by a specific layout of all the pieces on a table or a conveyor belt system that brings each piece into reach of the arm when requested. In the described workspace, a central point in this area was selected to act as the global coordinate system origin. From here, a reference to x = 0 m, y = 0 m, z = 0 m or (0, 0, 0) m will refer to this point.

With the workspace clearly established, the next step is to establish a pose for the desired assembly. For convenience, the assembly was centered at (0, 0, 0) m. From this point, the robotic arms should be positioned in the workspace. Placing the arms in the correct locations is a balance between several factors. The most important factor is accessibility to any part in the assembly, however, if the arms are placed too close, they are more likely to collide with one another while moving through the area. This implies that the arms chosen for the assembly task will be a major factor in choosing placement. For this implementation, the arms chosen were the Universal Robots UR 5 model. These arms were chosen for the open source nature of the arm and their long reach of 850 mm, offering potential to assemble larger pieces of furniture. They were placed at (0.5, 0, 0.25) m and (-0.5, 0, 0.25) m. The inverse kinematics were generated by Ryan Keating at Johns Hopkins University [26].

Interpreting the Blueprint of the Assembly

The next piece in the process is the input of a blueprint. The assembly planner was implemented in Python 2.7 using the numpy-stl 2.8.0 library. This format was chosen for ease of use with a wide variety of systems. The system will prompt the user for a folder containing any number of STL files describing the part. These parts should be generated by exporting an assembly from any commercial CAD program. Exporting from an assembly places all files in the correct position and orientation for the final configuration so the planner can parse the data. The STL files are then parsed by the algorithm and information about the center of mass, point cloud data, etc. are stored as objects in Python.

Disassembly Planner

Once the blueprint has been uploaded, the algorithm begins by planning a disassembly of the blueprint. The STL objects are passed as a list into a modified version of IDDFS to determine the correct order to take apart the assembly. In this setup, the states are defined as the individual parts present in the assembly after any given action. Actions are defined as part numbers to be removed. Because the goal is defined as removing all pieces from the assembly, the initial and maximum depth of the search are set to the number of pieces that must be removed from the assembly, or the total number of parts. This step reduces computation by excluding iterative searching for goals at smaller depths, where no goals exist. This knowledge of the depth of the goal increases the speed of the IDDFS.

The complexity of an IDDFS is $\mathcal{O}(b^d)$ [22], where b is the branching factor of the tree and d is the depth of the tree. The average branching factor used for b is the number of non-root nodes divided by the number of non-leaf nodes. For now, assume that any action, or removing any part from the assembly, is possible. For this case of disassembly of furniture, where the number of possible actions and states are limited by the number

Copyright © 2019 by ASME

of total parts, b is found using the equation,

$$b = \frac{\sum_{i=1}^{n} \frac{n!}{(i-1)!}}{1 + \sum_{i=1}^{n-1} \frac{n!}{i!}}, \tag{1}$$

where the numerator is the number of non-root nodes and the denominator is the number of non-leaf nodes in the search tree of an assembly with n parts. This equation can be simplified to

$$b = \frac{\sum_{i=0}^{n-1} \frac{1}{i!}}{-1 + \sum_{i=0}^{n-1} \frac{1}{i!}}, \tag{2}$$

where, as the number of parts approaches infinity, the summation in the numerator and the denominator can be substituted by the Taylor series approximations for the exponential function.Thus, the branching factor of the disassembly search tree converges to

$$b = \frac{e}{e-1} \tag{3}$$

as the number of parts approaches infinity. Because the number of nodes increases as a function of $n!$, the branching factor rapidly converges to the value in Eq. (3). Hence the branching factor is bounded, making the depth of the tree, d, or the number of parts that can be removed from the assembly, the main contributing factor when assessing the complexity of the IDDFS. In summary, the complexity of the IDDFS can be simplified to $\mathcal{O}_{IDDFS}(b^n)$, where b is the bounded branching factor seen in Eq. (3) and n is the number of parts in the assembly.

To determine whether a part can be removed it must pass through two checkpoints, contact checking and possibility of removal. This contact checking determines if, after the selected part is removed from the assembly, all other parts in the assembly are in contact with at least one other part. To do this, the algorithm takes the entire assembly and examines the point cloud data to check for collision with other parts. Because the parts are all polygonal, the vertex of the one part needs to lie inside the bounded plane made by three vertexes on another part. The algorithm stores a list of which parts in the assembly are in contact and uses this to check if the part can be removed and still leave all remaining in contact with another part. When there are two parts or less remaining in the assembly, this contact checking is ignored.

The complexity of this contact checking for an assembly with n parts is $\mathcal{O}_{con}(n\,log(n))$ because the algorithm must search for connection between part 1 to parts 2 through n, but part 2 must only check for a connection between parts 3 through n and so on.

The algorithm has now determined that removing the part leaves all other parts in contact with at least one other part, but the algorithm must decide if is possible to physically remove the part without collisions with other parts. To determine this, the algorithm uses an RRT planner with the selected piece as the moving object and the rest of the assembly as obstacles. The piece is given a goal location away from the assembly and a collision checker is utilized at each time step to determine if the piece can be removed. The complexity of the RRT algorithm for an assembly with n parts is $\mathcal{O}_{dis}(n)$ because this RRT planner must be performed once per part to be removed from the assembly. Research about the complexity of the RRT algorithm with n vertexes was performed by Svenstrup et al. [27], but this complexity is not based on the number of parts in the assembly, so it is not included here.

The goal for that piece is computed by finding the geometric centroid of the assembly and of the specific part using

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i V_i}{\sum_{i=1}^{n} V_i}; \quad \bar{y} = \frac{\sum_{i=1}^{n} y_i V_i}{\sum_{i=1}^{n} V_i}; \quad \bar{z} = \frac{\sum_{i=1}^{n} z_i V_i}{\sum_{i=1}^{n} V_i}, \tag{4}$$

where V_i is the volume of the ith part, n is the number of parts in the assembly, and x_i, y_i, and z_i are the Cartesian coordinates of the centroid of each part.

The algorithm then finds a unit vector from the assembly centroid to the part centroid. Using the largest dimension of the assembly as a scalar distance multiplier, the algorithm sets the goal position of this part to be that distance away along the unit vector, making the goal a specified distance away from the centroid of the assembly. If the resulting z-dimension is below the floor, i.e. less than zero, the absolute value of the z-dimension is used for the goal. This process is shown in Alg. 1 and in Alg. 2 and has a linear complexity of $\mathcal{O}_{goal}(n)$ because the computation of the centroid of the part and assembly, as well as the goal location are computed once per part in the assembly. If the RRT is unable to find a way to remove a part to the computed goal, the whole check fails and the next action must be selected in the IDDFS. An example of this process could entail something like an internal shelf that provides no support, but could not be removed until the higher shelves or the top are first moved out of the way.

The result of this step is an order in which the work piece can be safely disassembled and a motion path provided by the RRT planner for how to remove each part. With this information, the algorithm reverses the order of disassembly and the part motion paths, creating a forward path to assembly referred to as the disassembly path. The initial state of each piece then becomes the goal location for the next planning segment, the arm planner.

Motion Planning of Each Arm

To plan the motion of each arm, the algorithm uses another implementation of the RRT algorithm. Because it already knows the path to remove parts without collision, which is a reversed version of the path to move the parts into their final poses in the

Copyright © 2019 by ASME

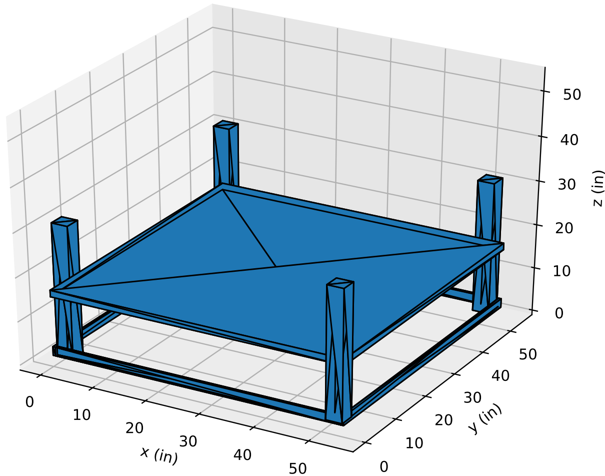

FIGURE 1. Invalid movement found in removing the table top from the table assembly. The collision checker rejects this movement because table top collides with the table legs.

Algorithm 1 Remove Part from Assembly

1: **procedure** REMOVE PART($part$, $parts_{remaining}$, $parts_{all}$)
2: $distance$ = max($dimensions_{assembly}$)
3: $start$ = $position_{part}$
4: $goal$ = generateGoal($part$, $parts_{all}$, $distance$)
5: $plan$ = RRTConnect($start$, $goal$, $parts_{remain}$)
6: **if** $plan$ is not “Empty” **then**
7: “Plan found”
8: **else**
9: “No plan found”
10: **end if**
11: return $plan$
12: **end procedure**

---

assembly, the algorithm only needs to plan from where the parts are sitting in the workspace to where the disassembly path begins. The complexity of this RRT planner is the linear, $\mathcal{O}_{arm}(n)$, as explained previously. The starting poses of all the parts must be passed to the algorithm, which can then plan a path from this pose to the beginning of the disassembly path. Figure 2 shows how these two paths, the disassembly path and the arm plan, work together to create the motion from the starting pose to the final pose in the assembly.

Even though no collisions were found when removing the work piece, collisions can still occur between the robotic arms and the assembly, so a collision checker must be used in this step in the algorithm. The robotic arms must be added to the environment using 3D models. As a first step, Arm 1 will be selected as the movable, assembly arm. It will pick up the first part and

---

Algorithm 2 Generate Goal for Part

1: **procedure** GENERATE GOAL($part$, $parts_{all}$, $distance$)
2: **for** pt in $parts_{all}$ **do**
3: $sum \mathrel{+}= volume_{pt} \times centroid_{pt}$
4: $volume_{total} \mathrel{+}= volume_{pt}$
5: **end for**
6: $centroid_{allparts} = sum/volume_{total}$
7: $vector_{centroid} = centroid_{part} - centroid_{allparts}$
8: $direction = \text{normalize}(vector_{centroid})$
9: $goalvect = direction \times distance$
10: $goal = centroid_{part} + goalvect$
11: **if** $zposition_{goal} < 0$ **then**
12: $zposition_{goal} = |\, zposition_{goal} \,|$
13: **end if**
14: return $goal$
15: **end procedure**

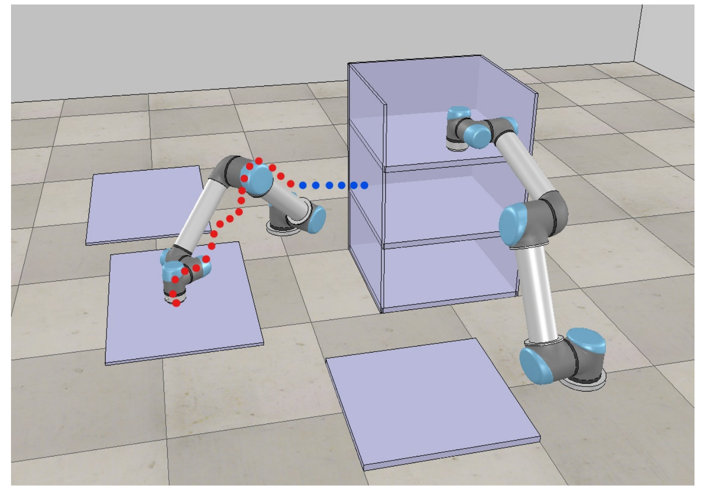

FIGURE 2. The arm motion planner uses the disassembly path found from a previous section of the algorithm as part of the full path of each part. The red path shown is a representation of a path found by the arm planner and the blue path represents a disassembly path found by the disassembly planner. The transparent part on the left side of the assembly shows the goal position of the part in the assembly.

move it into place. Once this has been accomplished, Arm 1 will switch into a “hold” mode where it remains stationary to secure the work piece while Arm 2 is put into “assemble” mode and places the second part into position. Once Arm 2 has placed its part it switches into “hold” mode while Arm 1 reverts to “assemble” mode and acquires the next piece. This process continues until the entire assembly is completed. During each assembly phase for each arm the RRT utilizes a collision checker to determine if its movement to each step is allowed. This checker compares the point cloud data of the arm in question with every

Copyright © 2019 by ASME

other object in the workspace. If a collision is detected the state is rejected and planning continues. Pseudocode for this part of the algorithm can be read in Alg. 3 and Alg. 4.

Algorithm 3 Assemble Parts in the Given Order

1: **procedure** ASSEMBLE PARTS($order_{assembly}$)
2: **for** *part* in $order_{assembly}$ **do**
3: $result$ = placePart($part$, $location_{part}$, $goal_{part}$)
4: **if** *result* is "Order Failed" **then**
5: return "Assembly Order Failed"
6: **else**
7: continue
8: **end if**
9: **end for**
10: return "Assembly Order Succeeded"
11: **end procedure**

Algorithm 4 Place Part in Assembly

1: **procedure** PLACE PART($part$, $location$, $goal$)
2: $plan$ = RRT($part$, $location$, $goal$)
3: **if** *plan* is "Empty" **then**
4: return "Order Failed"
5: **else**
6: return "Part Placed"
7: **end if**
8: **end procedure**

One issue that may arise with this process is that the arm in "hold" mode may make it impossible to place the desired piece for the arm in "assemble" mode. If it is determined that there is no collision-free path to assemble the selected part, the algorithm knows this must be due to a collision with the arm in "hold" mode because the algorithm has already found a collision-free plan for assembling each piece into the assembly. For this reason, the collision cannot be caused by two parts colliding with each other; it must be from collision with the arm. If this is the case, the arm in "hold" mode is passed to a new planner that will determine a plausible way to grasp the assembly in a different location to allow the arm in "assemble" mode to continue its work. This grasp planner is not implemented in the current algorithm and will be left for future work.

Once the assembly RRT has completed, the algorithm will output a complete plan comprising of a list of states for each arm to construct the desired blueprint. A diagram of the full assembly algorithm can be seen in Fig. 3.

One important simplification to this process is the absence of fasteners. It is currently assumed that once a piece is in the correct location it will stay fixed. This is obviously not true for real-world application, but this problem could be solved by adding a

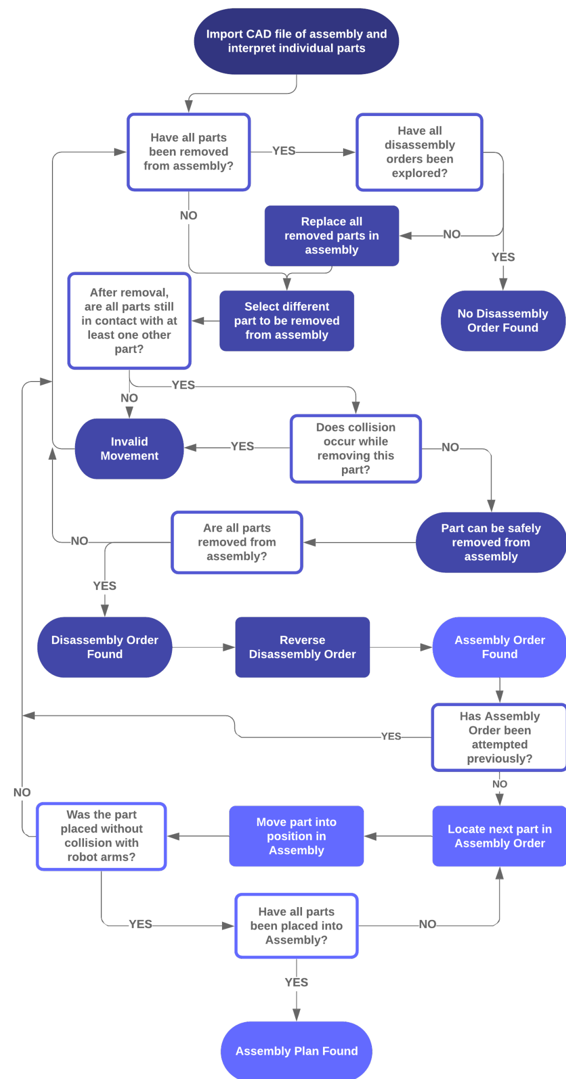

FIGURE 3. Flow chart of the assembly algorithm discussed in this paper. The algorithm starts with a blueprint or CAD file of the piece to be assembled, and outputs the assembly plan or that no assembly plan is possible.

third arm whose task is to move across joints and add fasteners such as nails, screws or glue in between each phase of switching between "hold" and "assemble" modes for the arms.

IMPLEMENTATION OF ALGORITHM

To prove the validity of the algorithm presented in this paper, the process was tested using simple geometric CAD models, such as a simple table and box. Using these models as test work pieces, the algorithm could be shown to be working properly or

Copyright © 2019 by ASME

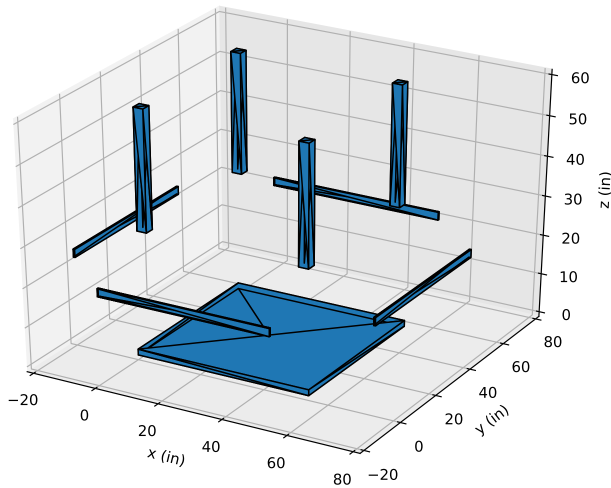

FIGURE 4. A representation of the positions used as goals in the disassembly algorithm. If the part can move to this goal without colliding with other parts of the assembly, then the algorithm knows the part can be removed from the assembly.

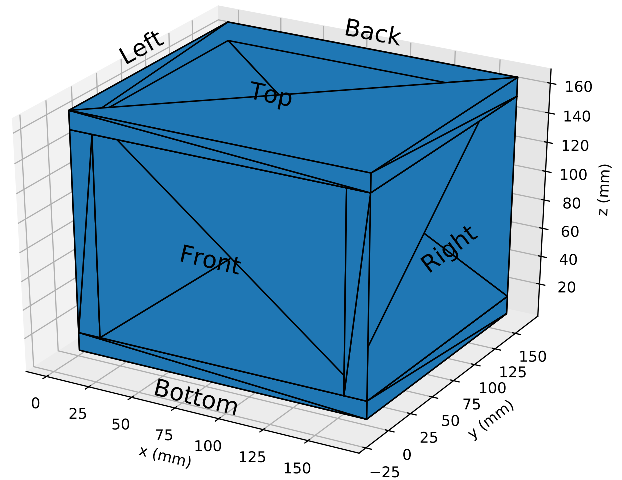

FIGURE 5. This simple box was used to test the success of the method described. The assembly order as found by the algorithm is as follows: Bottom, Left, Front, Back, Right, Top.

not. As stated previously, the tests happened in simulation where there would be space provided for all components. The components were also assumed to be in locations where the arms could easily reach them, and that the weight of each component was negligible such that any part could be lifted by the robotic arm using a suction cup gripper.

The first phase of the algorithm, disassembling the assembly, was the most important as it provided the crucial information of the assembly order. As stated previously, this was done in Python using .stl files and assemblies exported from CAD modeling software (SolidWorks 2017). When disassembling the box, one assumption made was that it did not matter which side was the bottom as the box could be physically assembled independent of the orientation. With the table, however, this is not the case. In the case of two robot arms performing the assembly task, the table top cannot be placed in the assembly without the table legs falling over. Because of this, the table's assembly orientation was with the table top on the ground. Using IDDFS, the assembly was disassembled as explained previously and then plotted to show the final disassembly in Fig. 4. The assembly order planned by the algorithm can be seen in Fig. 5 and 6.

The second phase was the arm planner, which put the pieces into the assembly. A composite of primitive shapes [16] was used to check for collisions with the other arms and parts held by each arm. Using this collision checker, the algorithm found paths to get the parts into their locations in the assembly. Because the disassembly planner assured no collision occurred while removing the parts from the assembly, no collision checker was necessary to check for collision between the part in motion and the other

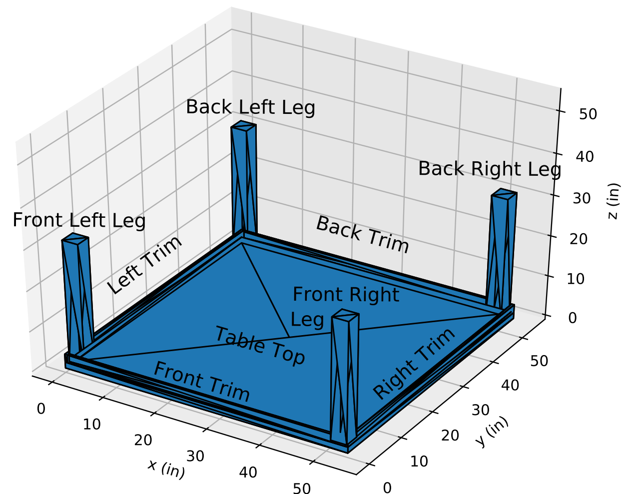

FIGURE 6. This table was used to test the success of this method. The assembly order as found by the algorithm is as follows: Table Top, Back Left Leg, Front Left Leg, Back Right Leg, Front Right Leg, Left Trim, Right Trim, Back Trim, Front Trim.

parts in the assembly. Once the first part was set into the assembly, the "place and hold" system described previously was implemented to keep the assembly stable while a new part was moved into the assembly.

During implementation in the VREP environment, the algorithm successfully followed the disassembly path, which always placed the part in its correct position in the assembly.

Copyright © 2019 by ASME

RESULTS AND ANALYSIS

Overall, the two phases in the algorithm worked well together. The total complexity of the algorithm can be found by summing the complexities of the IDDFS, contact checking, disassembly RRT planner, goal generation, and RRT arm planner, which yields

$$\mathcal{O}(b^n) \approx \mathcal{O}_{IDDFS}(b^n) + \mathcal{O}_{con}(n\ log(n)) + \mathcal{O}_{dis}(n) + \mathcal{O}_{goal}(n) + \mathcal{O}_{arm}(n) \quad (5)$$

making the IDDFS the most complex part of the algorithm in terms of number of parts. However, as explained previously, the constant b is bounded by Eq. (3). By using the preliminary disassembly steps to quickly verify whether the object can be taken apart, the algorithm uses this knowledge to avoid more precise movements to place the part in the exact orientation without collision with the adjoining pieces.

The algorithm presented has room for improvement. Working with simply shaped parts reduces the possible issues that may appear from more complicated parts or assemblies. Some examples are: CAD models that must be rotated to place components (i.e. the table in Fig.6), models with many pieces inside an outer structure, and parts that must be slid sideways into place. This algorithm is currently limited to moving only one part at a time, so more complex movements, such as multiple pieces needing to be placed at once, are not possible.

A number of steps can be taken to increase the performance of the RRT used to find paths in both the disassembly and assembly phases of this algorithm. The RRT's implemented in this research are complete, as they will find a path between the initial and goal poses if a path is possible. However, they do not find an optimal solution in terms of path length. Utilization of RRT* [21] or RRT# [19] can provide an asymptotically optimal solution for the path of the robot arms. One other motion planning algorithm, called Lazy-RRG* [20], implements a "lazy collision checker" which does not check for collision until after a path to the goal is found, which decreases computation time. These algorithms can be implemented in the disassembly planner to remove parts from the assembly, as well as in the arm motion planner to connect parts from their initial positions to the end of the disassembly path. These algorithms mentioned will find an asymptotically optimal disassembly path in less time [19–21].

Another limitation of the algorithm addressed in this paper occurs when a robot arm or part is found to intersect with another arm. If this case is encountered, the algorithm would need to replan its motion, or possibly return to the disassembly planner to receive a new assembly order.

CONCLUSIONS AND FUTURE WORK

Through the assembly by disassembly philosophy, the algorithm presented in this paper finds an appropriate assembly order, then assembles the polygonal furniture using two robotic arms. The graph searching method of IDDFS is used to find an appropriate disassembly order. If the algorithm can remove a piece from the assembly, the part must not collide with other parts in the assembly and each part in the assembly must remain in contact with at least one other part. After the part is successfully removed from the assembly, the algorithm saves the disassembly path as illustrated by the blue path found in Fig. 2 for use in the assembly process. Once the assembly order is found, an RRT is employed to find a collision free path to the end of the disassembly path, which is then followed to position the part in the assembly. This assembly path is illustrated by the red path found in Fig. 2. Success of this portion of the algorithm is determined by its ability to assemble the part without collision. If collision does occur, the algorithm returns to the disassembly order portion to find a new disassembly order, then re-enters the assembly planning portion.

The success of this algorithm is in its ability to break down the assembly task into two parts. The first part is to find an order in which to assemble the parts. The second part of the algorithm is to plan the motion of the two arms.

Overall, the algorithm described in this paper is effective in finding an assembly order and planning the path to place each part in the assembly. This paper offers a novel method to finding a path to place each part in the assembly, using the disassembly path as a reversed, finishing assembly path.

One direction for future work is to implement a more accurate collision checker for the movement of the robotic arms. The primitive shapes used in this implementation can be conservative in calculating collision, so other methods might more accurately predict collision. These improved methods of collision detection could increase the ability to plan the motion of the arms, which would increase the overall performance of the algorithm.

As another area of future work, the researcher could add a third arm to insert fasteners into the joints of the work pieces. This would allow the process to become fully automated, rather than relying on human interaction to fix the pieces together. This objective could be reached by adding this fastener planner as an intermediate step between placing a part in the assembly, then moving to place the next part in the assembly. This fastener planner would create a more comprehensive assembly plan, which would remove simplifying assumptions made in this work.

REFERENCES

[1] Frohm, J., Lindström, V., Winroth, M. and Stahre, J., *The Industry's View on Automation in Manufacturing*, IFAC Proceedings, vol 39(4), pp. 453-458, 2006.

[2] Gupta, S., "Overview of Assembly Modeling, Planning,

Copyright © 2019 by ASME

and Instruction Generation Research at the Advanced Manufacturing Lab," Advanced Manufacturing Lab, University of Maryland, College Park, MD 2012.

[3] Helping robots put it all together, (2019). Photo credit: Dominick Reuter [image] Available at: http://news.mit.edu/2015/assembly-algorithm-for-autonomous-robots-0527 [Accessed 25 Mar. 2019].

[4] Knepper, R. Layton, T., Romanishin, J. and Rus, D., "IkeaBot: An autonomous multi-robot coordinated furniture assembly system," *IEEE Conference Publication*. (2013)

[5] Robot pcb, (2019). [image] Available at: https://commons.wikimedia.org/wiki/File:Robot˙pcb.jpg [Accessed 3 July 2019]. Image modified to fit page.

[6] KUKA Industrial Robots IR, (2019). [image] Available at: https://commons.wikimedia.org/wiki/File:KUKA˙Industrial˙Robots˙IR.jpg [Accessed 3 July 2019].

[7] Acemoglu, D., and Restrepo, P., "Artificial Intelligence, Automation and Work," *SSRN Electronic Journal*, 2018. Available: 10.2139/ssrn.3098384.

[8] Belhadj, I., Trigui, M., and Benamara, A. "Subassembly generation algorithm from a CAD model," *The International Journal of Advanced Manufacturing Technology*, vol. 87, no. 9-12, pp. 2829-2840, 2016. Available: 10.1007/s00170-016-8637-x.

[9] Morato, C., Kaipa, K., and Gupta, S., "Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters," *Computer-Aided Design*, vol. 45, no. 11, pp. 1349-1364, 2013. Available: 10.1016/j.cad.2013.06.005.

[10] Wan, W., Harada, K. and Nagata, K., "Assembly sequence planning for motion planning," *Assembly Automation*, 38(2), pp. 195-206, 2016.

[11] Knepper, R., Ahuja, D., Lalonde, G. and Rus, D., "Distributed Assembly with AND/OR Graphs," *Workshop on AI Robotics International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[12] Dogar, M., Knepper, R., Spielberg, A., Shoi, C., Christensen, H. and Rus, D., "Multi-scale assembly with robot teams," *The International Journal of Robotics Research*, vol. 34(13) pp. 1645-1659, 2015.

[13] Dogar, M., Spielberg, A., Baker, S. and Rus, D., "Multi-robot grasp planning for sequential assembly operations," *Autonomous Robots*, 2018.

[14] De Fazio, T., and Whitney, D., "Simplified generation of all mechanical assembly sequences," *IEEE Journal on Robotics and Automation*, vol. 3, no. 6, pp. 640-658, 1987. Available: 10.1109/jra.1987.1087132.

[15] Homem de Mello, L., and Sanderson, A., "AND/OR graph representation of assembly plans," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 188-199, 1990. Available: 10.1109/70.54734.

[16] LaValle, S., *Planning Algorithms*, New York (NY): Cambridge University Press, 2014, pp. 39, 82-122.

[17] LaValle, S. and Kuffner, J. Jr. "Rapidly-exploring random trees: Progress and prospects," In *Algorithmic and computational robotics: new directions: the fourth Workshop on the Algorithmic Foundations of Robotics*, AK Peters, Ltd., 2001, pp. 293.

[18] Kuffner, J. Jr. and LaValle, S., "RRT-Connect: An Efficient Approach to Single-Query Path Planning," *IEEE International Conference on Robotics and Automation, 2000. Proceedings., ICRA '00, 2000.* vol. 2, 2000.

[19] Arslan, O., and Tsiotras, P., "Use of relaxation methods in sampling-based algorithms for optimal motion planning," *IEEE International Conference on Robotics and Automation, Proceedings*, 2013, pp. 2421-2428.

[20] Hauser, K., "Lazy collision checking in asymptotically-optimal motion planning" *IEEE International Conference on Robotics and Automation, Proceedings*, 2015, pp. 2951-2957.

[21] Karaman, S., and Frazzoli, E., "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30(7), pp. 846-894, 2011. Available: 10.1177/0278364911406761.

[22] Korf, R., "Depth-first iterative-deepening: an optimal admissible tree search," *Artificial Intelligence*, vol. 27(1), pp. 97-109, 1985.

[23] Lim, K., Seng, K., Yeong, L., Ang, L. and Ch'ng, S. "Uninformed pathfinding: A new approach," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2722-2730, 2015. Available: 10.1016/j.eswa.2014.10.046.

[24] Wang, J., Liu, J., and Zhong, Y., "A novel ant colony algorithm for assembly sequence planning," *The International Journal of Advanced Manufacturing Technology*, vol. 25, no. 11-12, pp. 1137-1143, 2004. Available: 10.1007/s00170-003-1952-z.

[25] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P., "Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, 58(1) pp. 161-205.

[26] Keating, R., "UR5 Inverse Kinematics," *Slideshare.net*. https://www.slideshare.net/RyanKeating13/ur5-ik [Accessed 13 Dec. 2018].

[27] Svenstrup, M., Bak, T., and Andersen, H. J., "Minimising computational complexity of the RRT algorithm a practical approach," *IEEE International Conference on Robotics and Automation*, 2011, pp. 5602-5607.

Copyright © 2019 by ASME