Robotic Jigsaw: A Non-Holonomic Cutting Robot and Path Planning Algorithm

Haisen Zhao[1,4], Yash Talwekar[2], Wenqing Lan[1], Chetan Sharma[3], Daniela Rus[3], Adriana Schulz[1], Jeffrey I Lipton[1,2,*]

***Abstract*— Bladed tools such as jigsaws are common tools for wood workers on job-sites and in workshops, but do not currently have sufficient autonomous hardware or path planning algorithms to enable automation. Here we present a system of an autonomous robot and a path planning algorithm for automating jigsaw operations. The robot can drill holes, insert the jigsaw, and cut plywood. Our algorithm converts complex shapes into paths for the jigsaw, drill holes, and traversal movements for the robot. The algorithm decomposes input shapes into cuttable sections and determines possible locations for drilling entry holes for inserting the blade. We cast the drill hole problem as a set coverage problem with a trade-off between number of holes and cutting distance. We characterize the algorithm on a series of shapes and determined the algorithm found valid solutions. We executed an example on the robot to demonstrate the end-to-end system.**

I. Introduction

Many professions such as carpenters can see significant improvements by small amounts of automation. A wide variety of assembly tasks have been automated using mobile robots, but many cutting tasks have been understudied [1], [2], [3], [4], [5]. By automating low level cutting tasks, we can increase worker safety, accessibility, and worker productivity. We have developed a solution for scalable blade-based fabrication tools such as the jigsaw. It combines an algorithm for automatically planning cuts with bladed tools and drills and a mobile robot capable of executing the commands across arbitrary large surfaces. Together the algorithm and hardware demonstrate that we can automate many of the tasks currently done with automatic bladed tools.

Currently deployed solutions, such as stationary CNC machines and newer hand held CNC routers, rely on rotary tools as the basis for fabrication [6]. These tools are holonomic, and can be plunged into a surface and move in arbitrary directions. However, these tools have significant limitations, for instance they cannot leave a sharp corner. Carpenters by contrast rely heavily on linear blade tools such as hand saws, jigsaws, band saws, scroll saws and reciprocating saws to make most of their cuts. Without automating these bladed tools, workshop and jobsite applications will be limited.

Past researches into jobsite and workshop automation for carpentry have developed systems for automating lumber cutting using miter saws [7] and simple path planning algorithms for 2D cuts using linear blade tools [8]. This past work on linear bladed tools developed the foundational theory for modeling and path planning for such non-holonomic tools that we are building upon, but was limited by a lack of inter-tool interaction. It did not consider the synergies of bladed tools with drills. It assumed a single-entry point for a cut, and therefore could only cut the exteriors of shapes and required torturous movements between sections.

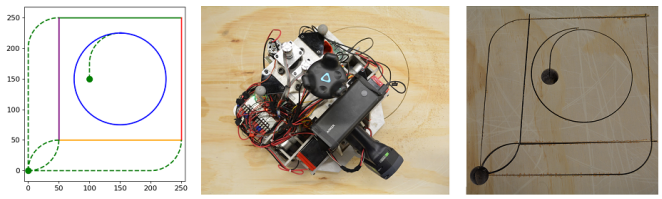

Fig. 1. A shape being planned (left), a circle being cut (middle) and the result of fabricating (right).

Our work extends the past research and focuses on how a drill and bladed tool work in tandem to produce more complex cuts. Drills fundamentally change the constraints on bladed tools, by allowing for the insertion of the blade into holes. This makes interior and nested cuts possible. It introduces a trade-off between the number of insertion points and the total cutting distance, which are two key considerations related to fabrication efficiency. We developed a new algorithm for planning under these new conditions and found it able to generate solutions for almost all planned cuts. We tested this algorithm against designs that would have caused previous planning algorithms to fail including designs used to make flat-pack furniture.

To prove the utility of the algorithm we built a robot capable of executing its solutions. It can drill pilot holes and insert and drive jigsaws over work surfaces. We used a scalable positioning system for localization that could be deployed to worksites. By successfully following commands from the algorithm to fabricate parts, we demonstrate the potential of the algorithm as well as the concept of scalable blade based robotic fabrication.

In this paper we:

- Developed the first path planning algorithm using bladed tools and drills for nested cuts
- Tested the algorithm on a library of shapes
- Developed a custom robot to implement and test the algorithm

[1]Computer Science And Engineering, University of Washington, Seattle WA, USA
[2]Mechanical Engineering, University of Washington, Seattle WA, USA
[3]CSAIL MIT, Cambridge MA, USA
[4]Computer Science And Engineering, Shandong University, Qingdao Shandong, China
*jilipton at uw.edu

II. Background and Related Work

A. Robot Carpentry

Many efforts in automating carpentry have relied on robotic arms or gantries. This has included prefab housing [9], post processing [10] and even bringing robots to job sites [11]. Efforts to find utility in bringing CNC machines to job sites have been limited by the difficulty in transporting machines and the inherent scale limitations on parts these machines have. A workpiece must fit inside gantry based CNC machines and making a larger CNC machine can increase cost and complexity in non-linear ways since the maximum deflection of a beam is proportional to the length of the span to the fourth power [12]. Deployable robotic arm based solutions have relied on moving robotic arms between stationary points [13], [14], [15].

More recent researches on using mobile robotics for onsite efforts have ranged from automating assembly [1] to lumber cutting [7]. By building a mobile robot around the tool we can decouple the robot size from the final part size. The next generation of mobile fabrication tools such as Shaper tools and Handibot allow a small CNC system to move over a much larger surface and cut features [6]. However, if you look around a workshop, almost no woodworker relies primarily on rotary tools like Dremel tools for most of their cuts. Instead, you will see a panoply of bladed tools.

Bladed tools defining feature is the use of a single-sided blade that either reciprocates or is part of a continuously moving band to cut curves [16], [17]. Aside from having a length, the blades also have a finite width called the kerf. This allows them to cut extremely thick parts more efficiently than rotary tools and allows the production of angled instead of curved corners. The cost of these advantages is the difficulty in planning.

B. Non-Holonomic Path Planning

Past efforts at path planning for bladed carpentry tools established that when a blade is in a work piece it can be thought of as a modified simple car [8]. Simple cars can move forward and backward, but have a finite turning radius [18]. This makes bladed tools non-holonomic, significantly complicating the planning.

While past research of algorithms for simple cars can aid in planning the unique needs and constraints on the bladed tool problem set it apart from standard non-holonomic path planning problems. Most past efforts focused on solving obstacle avoidance [19], [20], [21]. By contrast cutting a shape can be viewed as a dense version of a the traveling salesman problem (TSP) [22]. Those that do study TSP for non-holonomic focus on way point navigation [23], [24], [25]. Previous solutions targeting bladed tool cutting explored this space, but failed to account for the interaction of the bladed tools with drills for insertion. Our work here expands upon that previous work to develop a model to account for insertion points.

III. Path Planning Algorithm

A. Constraints on Bladed Tools

Bladed tools in work-pieces have a state consisting of a position and a heading defining a full state of $[X, Y, \theta]$ like a simple car (Figure 3A) [8]. Unlike simple cars, bladed tools can only move backward along paths they have already moved forwards along. Tools such as jigsaws also have multiple solutions for path following: they can cut by moving along a prescribed curve with the blade tangent to the curve at each point, or they can cut a shape by using the kerf of the blade to stop normal to the curve at a finite series of points (Figure 3D). This Kerf cutting technique is less desirable but can provide a rough approximation that can be cleaned in post processing.

When a bladed tool is in a hole in the work-piece, it can be viewed as a unicycle [18], able to turn arbitrarily but still having an orientation (Figure 3B). To insert a blade into a hole, the hole must have a diameter larger than the length of the blade. When using a Jigsaw, the hole must also be smaller than the width of the base plate plate 3B to ensure there is sufficient down force. From a single hole, the blade can make multiple entries into the work-piece (Figure 3C). The blade can move along several trajectories from the same entry point (Figure 3D). Ideally entries are either from the same point or spaced approximately 3 kerf widths apart on the circumference of the hole, however this spacing is a soft constraint, because the tools can make closer cuts, but would simply have a harder time making its transition into the work-piece. Therefor we did not consider it in the algorithm.

To make our algorithm work with a robot that moves along the surface of a work-piece, we have to enforce a constraint not typically found in carpentered items. This is a non-fall out constraint. Typically when cutting in a work-piece, if a closed loop is cut from a shape, that region falls away and becomes a hole. Since our robot drives along the surface of the work-piece and is not grounded separately from the work-piece, we cannot allow the material to fall away. With these constraints in mind, we can design our path planning algorithm.

B. Algorithm Overview

Our algorithm's input is a multiply connected shape $\mathscr{L}$ defined by a set of simple closed loops, $C_0, C_1, ..., C_n$, in which $C_1, ..., C_n$ are interior to C_0 and have disjoint interiors. Each loop is represented by a piece-wise linear function of lines between points of adaptive distance as shown in Figure 2. Output is a cutting path which takes a set of selected drill holes as starting points, with a trade-off between the number of drill holes and the cutting distance.

We first section the input loops into Maximal Cut Sections (MCS) based on the turning radius R_{min} of the blade (Section III-C). An MCS is a loop section which can be cut in a continuous manner (Section III-D). Second, we compute Entry Regions (ERs) for each MCS (Section III-E), where an ER indicates practicable locations to place drill holes as starting points to cut that MCS. Third, we determine

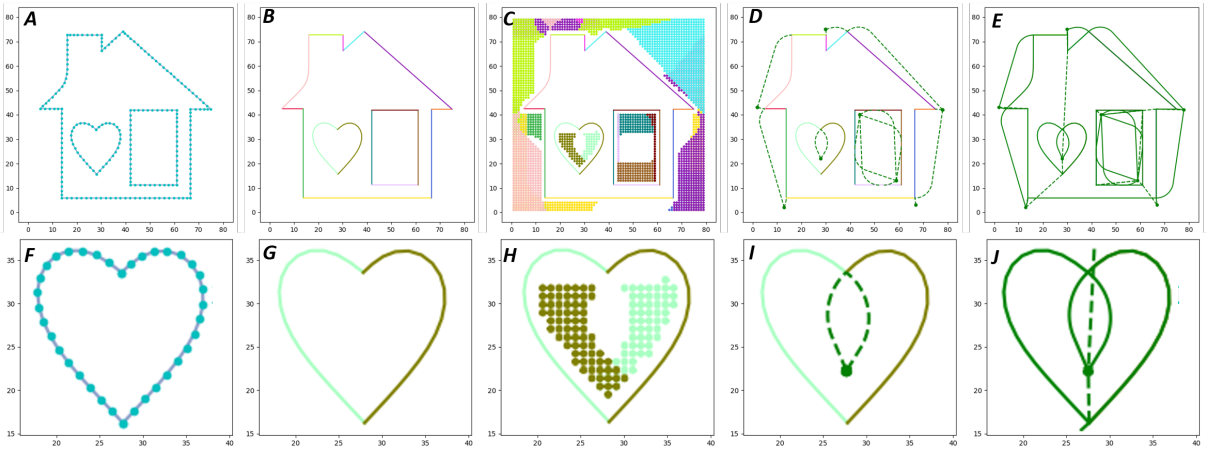

Fig. 2. Processing a given shape into a cuttable trajectory. The top row shows the overall shape being processed. The bottom row shows a representative close-up being processed. The target curve A, F is decomposed into a set of Maximal Cut Sections (MCS) shown in B, G, in which each MCS is indicated with different colors. For every MCS, we compute its Entry Region (ER) where locates the drill holes taken as the starting point to cut that MCS. In C and H, the Entry Region is rendering with the same color as its corresponding MCS. Entry Regions of different MCS can be overlapping shown in C. Based on MCS's ER, a simulated annealing algorithm is applied to extract the drill holes in D, I. Then the cut trajectories are ordered together and connected with a greedy TSP algorithm shown in E, J. Note that we use a relative larger $R_{min} = 7$ inch in this example for better visualization.

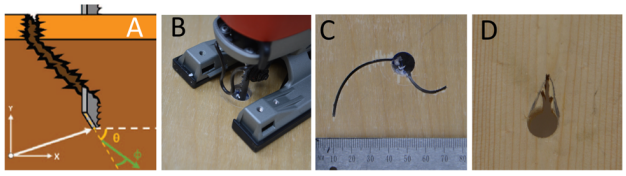

Fig. 3. Jigsaws in the material have a modified simple car model (A). Jigsaws can be inserted in holes and are free to rotate (B). From a single hole blades can cut at multiple headings and radii (C) and can use the kerf to cut normal to paths (D)

the locations of drill holes to provide coverage for the set of MCS (Section III-F). Finally, we generate trajectories for each MCS and connect them with a standard traveling-salesman based method. The pseudo-code of our algorithm can be found in Algorithm 1. In the following sections, we will describe this in detail.

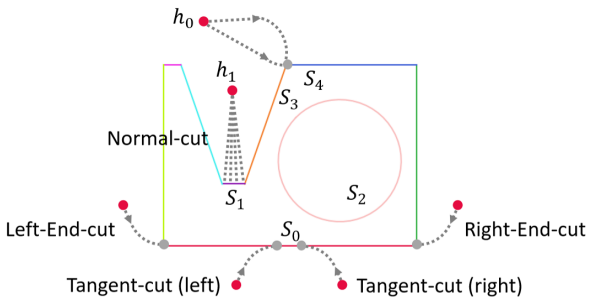

Fig. 4. The various MCS types and different cut strategies. The points on the target curve are decomposed into closed Maximal Cut Section (S_2) and open Maximal Cut Section (such as S_0, S_1, S_3, S_4). *Right-End-Cut*, *Left-End-Cut* and *Tangent-Cut* are demonstrated to generate cut-able trajectory of S_0. *Normal-cut* is demonstrated with S_1. The red and grey dots indicate drill holes and entry points separately. A drill hole h_0 is shared by multiple MCSes, S_3, and S_4.

C. Making Sections

We use a curvature test to decompose the input loops of $\mathscr{L}$ into a set of Maximal Cut Sections (MCS). At each point v_i along a loop, we calculate the curvature in terms of the change in tangents divided by the distance between the edge centers [26]:

$$\kappa_i = \frac{|t_i - t_{i-1}|}{|[(v_i + v_{i+1})/2] - [(v_{i-1} + v_i)/2]|}$$

where v_{i-1} is the preceding point of v_i, v_{i+1} is the subsequent point and t_i is the middle point between v_i and v_{i+1}. If the curvature of a point is bigger than the maximal turning curvature defined as $1/R_{min}$, it is taken as a breaking point. Each point on the input loops is then decomposed into sections with these breaking points. These sections are taken as Maximal Cut Sections (MCS). This enforces the minimum turning radius constraint of the bladed tool. An MCS with multiple points can have the blade move along the trajectory. On the other hand, an MCS with a single point cannot be cut along by the blade due to the turning limit, but we can use the kerf of the blade and stop with the blade normal to the curve at that point. A special case is that when one point breaks the curve into two MCSes of multiple points, we consider this point as an endpoint for its neighboring MCSes and don't mark it as a separate MCS of a single point. MCSes can be closed or open curves. In Figure 4, S_2 is closed MCS and S_0, S_1, S_3, S_4 are open MCSes.

D. Cut Strategies

An MCS can be cut with four different strategies, *Right-End-Cut*, *Left-End-Cut*, *Tangent-Cut*, and *Normal-Cut*. The *Right-End-Cut* (*Left-End-Cut*) strategy cuts an MCS starting from the right (left) end and then traverse the section to the left (right) end. The *Tangent-Cut* strategy cuts the whole section with two internal points (p_i, p_j) as entry points

Algorithm 1: Process Shape $\mathscr{L}$ with given R_{min}

```
MCSes ← MakeSections(ℒ, R_min);
foreach MCS do
    ER ← GetEntryRegion(MCS) ;
end
holes ← SelectHoleLocations(ERs) ;
foreach MCS do
    Trajs ← Trajs ⋃ GenerateTrajectory(holes) ;
end
return ConnectTrajectory(Trajs);
```

where p_i is right-tangent-accessible and p_j is left-tangent-accessible. The set of points to the right of p_i, including p_i and the set of points to the left of p_j, including p_j, have an intersection with at least 1 point. For those MCSes with a single point, p_i, p_j are the same point. The *Normal-cut* strategy is to move the blade to the points on the section then stopped. As shown in Figure 4 each cut strategy starts from a drill hole and ends on the MCS. With the exception of *Normal-Cut*, they start with a straight segment and an arc segment and enter the MCS tangent to a point on the MCS, then cut along that MCS. *Normal-Cut* follows the same cut trajectory pattern but moves normal to a point on the MCS. Note that a single drill hole can be taken as the same starting point for multiple cut strategies.

Different cut strategies are not equivalent to each other. We set three levels of priority. Level one is *Right-End-Cut* and *Left-End-Cut*, level two is *Tangent-Cut*, and level three is *Normal-Cut*. We prefer lower level solutions, because they provide the highest quality and fastest cuts. Cutting using *Tangent-Cut* is preferred over *Normal-Cut* because it cuts along the curve rather than leave defects from using the kerf. Thus, once we find a valid solution, we will pick that solution for the given MCS and stop computing the rest of the cut strategies. If we end up computing the Normal-Cut and no solution is found for one point on the MCS, it means our machine cannot cut the MCS. A specific cut strategy is valid iff there exists a non-empty Entry Region. Next section will detail the Entry Region detection process for each cut strategy.

E. Entry Region Detection

This step aims to detect the region where holes can be drilled for a cut strategy used on an MCS. This region is called the Entry Region. The detailed detection process of three types of cut strategies is demonstrated in Fig. 5.

For each cut strategy, we first back-trace two quarter circles given the entering direction. Each quarter circle has radius of R_{min} to limit our search space. For P_0 and P_3 in figure 5, either side of the cut-in direction is a cuttable region, because the circles do not intersect the shape $\mathcal{L}$. For $\{P1, P2\}$, we can only cut from one of the circles. Next, we shoot rays tangent to the circle to find their intersections against the shape $\mathcal{L}$ or the bounding box of the shape. We select the closest intersection to avoid the ray cutting through

Algorithm 2: Drill Hole Selection

```
Data: V_init, T_max, T_min and r
T ← T_max; V ← V_init;
C, H ← EvaluateCost(V);
V_best ← V; C_best ← C; H_best ← H;
while T > T_min do
    V_new ← Modify (V);
    C_new, H_new ← EvaluateCost(V_new);
    if exp(−(C_new − C)/T) > rand(0, 1) then
        V ← V_new; C ← C_new;
        if C_new < C_best then
            V_best ← V_new; C_best ← C_new;
            H_best ← H_new;
        end
    end
    T ← T ∗ r;
end
return V_best;
```

the shape. Next, we connect the intersections to form an Entry Region as shown in Fig. 5 A. If the connection segment cuts the shape, we move the boundary segment inward until no intersection found between it and the shape.

We discritize the Entry Region curve with a grid size equal to the drill diameter to generate drill hole candidates. The output is depicted by the red dots in Fig. 5 B. To prevent the drill from drilling through the shape's boundary, we eliminate drill candidates that are within the distance of drill radius around the shape.

During the Entry Region detecting process, we have four rules to limit our search space: 1) We use a squared bounding box to constrain the shape. 2) When computing Tangent-Cut, we only choose the first valid solution to minimize computation time; 3) We define a maximal leading distance D_{max} and only consider drill points within D_{max} from the entering location; 4) We trace rays on quarter circles of R_{min}, so that we have the largest valid area to find drill points. These rules can be modified to increase the search for valid drill holes.

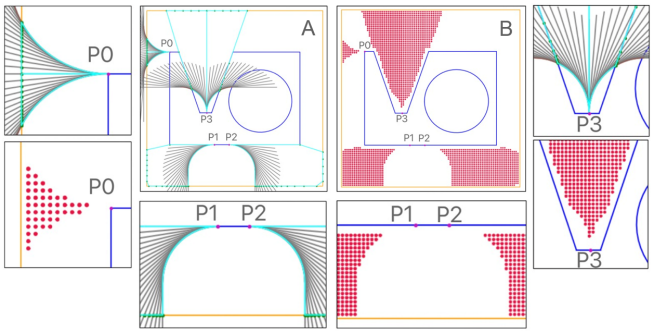

Fig. 5. Showing Entry Region detection for each cut strategy, P_0 (End-Cut), $\{P_1, P_2\}$ (Tangent-Cut), and P_3 (Normal-Cut). The cyan regions in figure A represent the Entry Regions found for each cut. The areas of red dots in figure B represent the drill locations sampled with the size of drill diameter within the according to Entry Regions in Figure A.

F. Drill Hole Selection

Once the shape has cut strategies with associated Entry Regions generated for each MCS, our next step is to determine a specific cut strategy and select a hole point for each MCS. The main consideration includes two terms, the number of holes and the total leading path distance. Our basic idea is to apply a simulated annealing algorithm, outlined in Algorithm 2.

We use an m-dimensional integral vector $V = V_i, \in [1, m]$ to indicate which cut strategy is chosen for MCS S_i. The algorithm has three input parameters T_{max}, T_{min}, and r, which indicate the starting temperature, ending temperature, and cooling rate, respectively. In our experiment, we set T_{max} as 10^3, T_{min} as 0.1 and r as 0.96. We first initialize V with a vector of zeros. During each iteration, we apply a Modify function to get a new vector V_{new}. In the Modify function, for each $j \in [1, m]$, we randomly select the cut strategy of the j_{th} Maximal Cut Section.

V only encodes the selection of cut strategy along with its Entry Region for each MCS. We still need a follow-up operation to select holes from each Entry Region. This is processed in the EvaluateCost function, which determines the selected holes H to minimize the number of holes N and the total leading path distance D (the total trajectory length for the robot to move from the starting holes to the entry points). A cost C is defined to evaluate H :

$$C = w \cdot (N/N_m) + (1 - w) \cdot (D/D_m)$$

where the weight $w \in [0.0, 1.0]$ is chosen to trade-off between N and D. N_m and D_m is used to normalize N and D , $N_m = 2N_{MCS}$, $D_m = N_{MCS} \cdot BB(\mathscr{L})$ in which N_{MCS} indicates the number of MCS before grouping one point Cut-Normal MCSes and $BB(\mathscr{L})$ is the bounding box size of the input shape $\mathscr{L}$.

As a specific set cover problem, we apply a greedy-driven iterative process to select holes from each MCS with the determined cut strategy for each MCS. We first get a union region U by overlapping all candidate Entry Regions of V, shown in C and H of Figure 2. Each point in U is associated with a set of MCS which can be reached from that point, and the corresponding leading path distance. During each iteration, search for a point in U that covers most non-hole-assigned MCS and uses the shortest leading path. The searching metric is $w \cdot ((N_m - \widetilde{N})/N_m) + (1-w) \cdot (\widetilde{D}/D_m)$, where $\widetilde{N}$ is the number of randomly selected subset of associated MCS, and $\widetilde{D}$ is the corresponding leading path distance. Such iteration ends when all MCS have been assigned with a hole. Note that our algorithm does not explore all subset of the associated MCS of a point in U, the maximal subset we explored is 20 in our experiments.

G. Integration of the trajectories into a path

The final step of our algorithm is to generate a single path. With the holes and cut strategy for each MCS, connecting these trajectories is a standard traveling-salesman problem so we apply a TSP-based method to minimize the path length used to connect trajectories. Each connection trajectory starts at a point in the MCS at the end of the cut strategy trajectory and ends at a drill hole. There are two strategies for connections: 1) the trajectory of V_i with the opposite direction without withdrawing the blade; 2) a straight segment with withdrawing the blade. The more efficient option is chosen which can be evaluated with the parameters in Section VI.

IV. Path Planning Results and Discussion

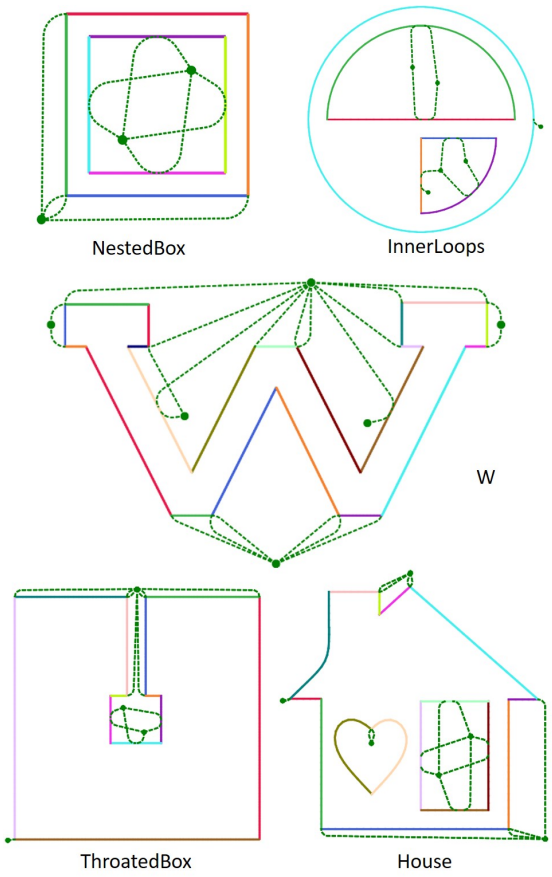

Fig. 6. To test our algorithm, we run our algorithm on several shapes with different complexity. A "NestedBox" shape is decomposed into 8 open MCS and one hole is used to cut the exterior square and two holes is for the internal square. Six holes are selected for the "InnerLoops" shape. We could find the holes used for the "W" are heavily shared by multiple open MCS. The " ThroatedBox" shape is produced with 4 holes, which is taken as a failure case in [8]. All of these shapes are processed with the same parameters listed in Section VI.

A. Algorithm Validation

To validate our algorithm, we tested our method on a variety of shapes shown in Figure 6. Our algorithm processed 5 shapes, including a nested square shape where a smaller square is placed in a big one (NestedBox), a nested circle shape where two curved sections are placed in a big circle (InnerLoop), a square connected to a smaller inner square, a "W" shape bounded with multiple segments (W), and a complex "house" shape with a "heart" shape and a square in the interior. For each input, our algorithm was able to section the input shape into Maximal Cut Sections, compute their Entry Regions (see Table I). The Nested Box demonstrates

our ability to plan for embedded loops. We were able to decompose the squares into the 4 sides automatically and find the appropriate cut strategies for Exterior paths, we were able to find the edge based entrances, and for the interior paths we detected the need to use the *Tangent-cut* strategy. The interior loops structure shows that we can handle closed loop shapes on the exterior as well as interior curved sections. The throated box demonstrates the algorithm's ability to handle complex exteriors which create pseudo-pockets which the previous path planning algorithm could not handle. Finally, the House demonstrates the ability to handle multiple interior curves in the same part. For all of the shapes, we searched for the minimum number of holes by setting $w = 0.99$ that could allow us to cut for the part and report them in Table I. Even for the complex House shape, we found only 6 holes were needed to make the shape. For the nested box, we found that only 3 holes were needed to the structure.

Shape	S(inch)	#L	#MCS	#ER	#EH	#MH	#H	T_{ER}(s)	T_H(s)
House	70x70	3	16	20	2998	5878	6	8.4	1.25
NestedBox	16x16	2	8	12	552	1268	3	3.4	0.3
InnerLoops	72x72	3	6	6	1850	1931	6	2.8	0.1
W	80x40	1	21	30	3950	24953	6	10.3	113.1
ThroatedBox	70x70	1	12	19	1095	3274	4	4.68	0.2

TABLE I

Some statistics and running times for our algorithm. All running time is in seconds. For each input shape, we report its size (S) and the number of loops (#L). We test these shapes with $w = 0.99$ to mainly minimize the number of holes. The number of Entry Regions (#ER), and the final selected holes (#H) are reported in this table.

B. Algorithm Performance

Table I reports some statistics with the running time of Entry Region Detection (T_{ER}) and Drill Hole Selection (T_H). We could find that the running time of Entry Region detection will increase along with the number of Maximal Cut Sections (#MCS), the number of overall holes in Entry Regions (#EH) and the number of #EH multiplied by how many MCS can be reached from each hole location (#MH).clarify the definitation and update the caption. The drill hole selection was efficient for most shapes except the W shape which used roughly 2 minutes to extract holes. As we can see, #MH of the W shape is much bigger than others which indicates a lot of overlapping between Entry Regions of Maximal Cut Sections. If a hole point is associated with many MCS, there will be a large number of potential coverage sets to be explored and it will take a longer time for the greedy-driven iterative algorithm to extract the optimal hole point. As one future work, we may speed up this step with a heuristic-driven pruning operation before applying the iterative algorithm.

C. Optimization Trade-off

The drill hole selection step of our algorithm considers two objectives, minimizing the number of selected holes to cover the whole MCS and minimizing the total leading path length. We use a weighted cost to be the trade-off between the two considerations. In Figure 7, we show two cases which search for different trade-offs by setting different w. As we can see, both cases will use more holes with a shorter leading path when w is equal to 0.1. When w is set as 0.9, the optimization results show less number of holes and a longer leading path. These results prove that our algorithm supports the trade-off optimization.

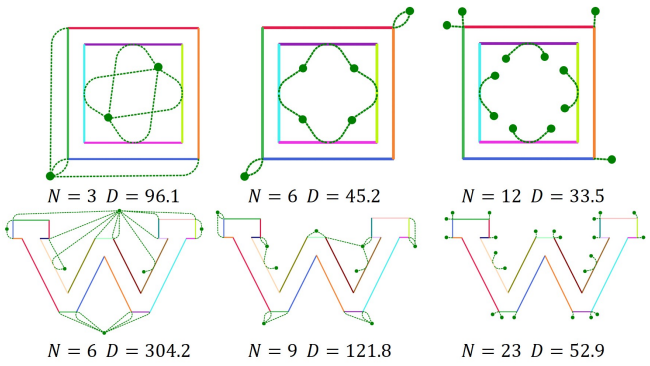

Fig. 7. Our algorithm supports different trade-offs between the number of holes N and the total leading path length D. This figure shows the planning results of the 'NestedBox" shape and the "W" shape with different w $(0.9, 0.5, 0.1$ from left to right column). The number of selected holes and the leading path length (in inch) are indicated in the figure.

D. Limitations and Failure case

Our algorithm will fail when we could not find any available Entry Region for a Maximal Cut Section. As shown in Figure 8, in the green square, the Maximal Cut Section between two upper leaves is not cuttable by our framework. Since the blade cannot cut in from either endpoint, we start to compute the Entry Region of the Tangent-Cut. Figure A shows the quarter circles of the Tangent-Cut at the middle point P_0. The two regions bounded by the circles and the shape have areas smaller than the circle produced by our drill, so we cannot find a valid drill location. We then move to compute the Normal-Cut for this MCS, but Figure B shows how Normal-Cut would fail on P_1. We can see that both quarter circles intersect the shape from the beginning, so we cannot find a valid Entry Region for point P_1. Thus, we cannot perform Normal-Cut on this MCS either.

One limitation of our algorithm is that we do not allow for transitions between ends of cuts that do not involve the removal of the blade or the reversal of the blade along a trajectory. All starting movements for a section are assumed to occur at an insertion site. In previous algorithms, Dubin paths [18] were used to connect the ends of cuts. This allowed for exterior square shapes to be cut from a single insertion without returning to the insertion point. As seen in the nested box example, we currently require 1-4 holes to make this cut and we must return to the insertion point after each cut. For some shapes, adding this capability in would reduce cut time. However there is generally a good reason why carpenters rarely do this maneuver, it generates a lot of wasted material relative to simply removing and reinserting the blade.

Another major limitation of our algorithm is the no-fall out constraint. While this is needed for a robot to drive on the surface of the part while cutting, applications where a jigsaw is independently supported, or uses on tools such as scroll saws do not have the need for this limitation. Extending this work to scroll saw planning should remove this constraint and take advantage of the planning freedom its removal would generate. This algorithm is also not suited for non-insertable blade tools such as band-saws. Our algorithm is sensitive to the resolution of processing. For instance, in the W shape, sections that should be cut from the edges are instead cut using normals. This is a result of the system being run at too large of spacing between points and the section being seen as blocked by the curves on the edge.

Possible solutions include using a jigsaw with a smaller turning radius and using a drill with a smaller diameter. These will be left up for future discussion.

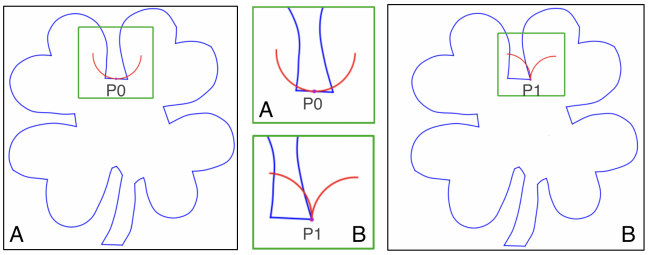

Fig. 8. Showing the failure case. The middle MCS between two upper leaves is not cuttable by our machine. Figure A shows the quarter circles of the Tangent-Cut at the middle point P_0. Due to the very limited region bounded by the circles and the shape, we cannot find a valid drill location that fits our drill diameter. Due to no solution found in Tangent-Cut, we compute Normal-Cut for each point on this MCS. Figure B shows the quarter circles of one point P_1 for Normal-Cut. Since both quarter circles cut the shape from the beginning, our jigsaw cannot reach P_1.

V. Design of Robot

Our robot was designed around a jigsaw drill pair. We used a Festool CARVEX PSBC 420 EB 561753 Cordless Jigsaw and a Milwaukee M12 12V 3/8-Inch Drill Driver (2407-20). Each was placed on a screw driven linear rail to allow them to be lifted up and driven down into the wood. The drill was placed in front of the jigsaw. The drives were placed 306mm apart and inline with the center of the jigsaw. This allows for the only constraint on turning radius to be the interaction of the saw with the wood.

To determine the force requirements on the drives we took the jigsaw equipped with a Bosch T101B 4-Inch 10-Tooth T-Shank blade and pulled them through 1/2inch of plywood via a spring scale to determine the force needed. With a cut speed of 30mm/s, 24.5N was needed, and with a cut speed of 42mm/s, 39.2N was needed. This provided us with a torque requirement for our system. The drive consists of Maxon DCX motors tuned for 15RPM at 2.6Nm. Each one is connected to the wheels through a belt transmission. The wheels are BaneBots Wheel, 2-7/8" diameter with a 50A durometer coating. We performed a tilt test and found the wheels had a coefficient of friction of 0.36 on wood and 0.28 on sawdust covered wood.

To control each power tool, we routed the tools' batteries through relays controlled by the robot's on-board computer. We used a Raspberry Pi 3B+ as the on-board computer and we drove the motors with a RoboClaw 2x7A Motor Controller. To power the system we had a separate USB power supply for the Raspberry Pi, two 12V 7Ah lead acid batteries to drive the motors and solenoids, and the batteries that came with the power tools. Since the robot has a cutting surface and a drill, we determined that we needed a hardware remote kill switch for safety. This was placed between the batteries and the control electronics. The resulting hardware can be seen in Figure 9.

We used an HTC Vive lighthouse based tracker on top of the robot centered over the jigsaw blade as the positioning system (See Figure 9. We used the PyOpenVR library [27] on a windows PC to interface with the tracker and obtain the state information, which we then pass through a Kalman Filter before sending it to the controller. When the tracker is stationary, we observed the noise to be normally distributed with standard deviations 0.1 mm, 0.1 mm and 0.01 rad for x, y and θ. However, we also observed that over time the tracking estimates drift by a maximum of 10 mm after some motion. We speculate the cause of this drift to be accumulation of integration errors in the in-built inertial sensors in the tracker. Further, reflective surfaces in the environment affect the tracking performance of the lighthouse system which leads to higher noise in some areas of the robot's working space. We wirelessly networked the pi to the PC and used ZeroMQ as the main message passing system. The pi therefore only needed to execute commands sent via ZeroMQ. We used the same model predictive controller as described in [8].

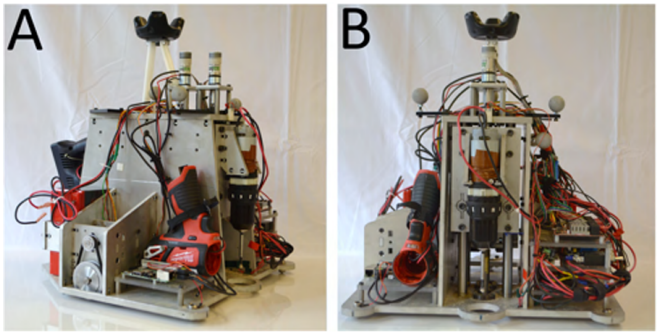

Fig. 9. Hardware for the Drill and Jigsaw Robot from the side (A) and front (B)

VI. Robot Characterization

To characterize the robot we first measured the minimum turning radius of the jigsaw when cutting plywood. We found the minimum turning radius was 15mm. Next we needed to characterize the movement accuracy. We drove the system without the controller active and found a drift of 12.5mm per meter off of straight lines at a speed of 40mm/s and a mean radial error of 18.3mm for a circle of radius 200mm.

With the MPC controller active we found that we reduced the error to 2 mm per meter off of straight lines and the mean radial error to 1.4 mm. The maximum speed we could achieve with the blade out is 125 mm/s and 25 mm/s when the blade is cutting. Inserting or removing the blade takes 23 s and drilling an insertion hole takes 15 s. Due to a limit on the force that the drill linear drive can exert only a small hole 6.35 mm in diameter can be placed by the robot. To determine the minimum drilling radius needed, we had to measure the position repeatability of the robot. So we repeatedly moved the robot to the same drill point and found we varied by 4.7mm in drill position. Given that as the variation of the center, and the blade having a length of 7 mm we need a hole drilled that is 25.4mm to ensure that the robot can insert safely.

VII. Fabrication Results and Discussion

To verify that the algorithm worked and was compatible with our hardware we planned to cut a square with a circle nested inside seen in Figure 1A. The Square had an external side length of 200mm and the circle was 150mm in diameter. First the robot drilled pilot holes and moved to a safe position. Next a human worker drilled out the holes to a diameter of 25.4mm. Finally the cut trajectories were executed open loop. We needed to execute open-loop because the tracker could not provide meaningful information because of the vibrations from the jigsaw. The end result can be found in Figure 1C. While the sides are not square and ideal for even a crude human carpenter, this is mostly due to errors in the tracking and the lack of control. The part demonstrates that the algorithm can make a viable plan that can be autonomously executed. Future improvements in tracking, and hardware will allow for higher quality results.

VIII. Conclusions

We have developed a new algorithm for path planning bladed tools and using multiple insertion points made by a drill. This algorithm extends the pre-existing literature to account for the interactions between bladed tools and drills to enable cutting nested shapes. To validate the algorithm we tested against a series of shapes and built a robot capable of automatically marking holes for drilling, inserting a jig saw, and driving the jigsaw to execute trajectories. This system enables scalable CNC manufacturing of complex shapes. This will enable scalable and deployable fabrication and help bring robotics out of the factory and into the workshop and jobsite.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation, grant numbers CCF-2017927, EEC-2035717, and 1644558 and by the NSF China (61772318).

References

[1] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *2013 IEEE International conference on robotics and automation*. IEEE, 2013, pp. 855–862.

[2] M. Dogar, A. Spielberg, S. Baker, and D. Rus, "Multi-robot grasp planning for sequential assembly operations," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 193–200.

[3] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction with quadrotor teams," *Autonomous Robots*, vol. 33, no. 3, pp. 323–336, 2012.

[4] M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus, "Towards coordinated precision assembly with robot teams," in *Experimental Robotics*. Springer, 2016, pp. 655–669.

[5] G. Reinhart and S. Zaidan, "A generic framework for workpiece-based programming of cooperating industrial robots," in *2009 International Conference on Mechatronics and Automation*. IEEE, 2009, pp. 37–42.

[6] A. Rivers, I. E. Moyer, and F. Durand, "Position-correcting tools for 2d digital fabrication," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 88, 2012.

[7] J. I. Lipton, A. Schulz, A. Spielberg, L. Trueba, W. Matusik, and D. Rus, "Robot assisted carpentry for mass customization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3540–3547.

[8] J. I. Lipton, Z. Manchester, and D. Rus, "Planning cuts for mobile robots with bladed tools," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 572–579.

[9] A. Thoma, A. Adel, M. Helmreich, T. Wehrle, F. Gramazio, and M. Kohler, "Robotic fabrication of bespoke timber frame modules," in *Robotic Fabrication in Architecture, Art and Design*. Springer, 2018, pp. 447–458.

[10] A. Amini, J. I. Lipton, and D. Rus, "Uncertainty aware texture classification and mapping using soft tactile sensors."

[11] V. Helm, S. Ercan, F. Gramazio, and M. Kohler, "Mobile robotic fabrication on construction sites: Dimrob," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4335–4341.

[12] Y. Altintas, *Manufacturing automation: metal cutting mechanics, machine tool vibrations, and CNC design*. Cambridge university press, 2012.

[13] R. Holmberg and O. Khatib, "Development and control of a holonomic mobile robot for mobile manipulation tasks," *The International Journal of Robotics Research*, vol. 19, no. 11, pp. 1066–1074, 2000.

[14] B. Hamner, S. Koterba, J. Shi, R. Simmons, and S. Singh, "An autonomous mobile manipulator for assembly tasks," *Autonomous Robots*, vol. 28, no. 1, pp. 131–149, 2010.

[15] ——, "Mobile robotic dynamic tracking for assembly tasks," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2489–2495.

[16] T. J. Ko and H. S. Kim, "Mechanistic cutting force model in band sawing," *International Journal of Machine Tools and Manufacture*, vol. 39, no. 8, pp. 1185–1197, 1999.

[17] B. Lehmann, "The cutting behavior of bandsaws," Ph.D. dissertation, University of British Columbia, 1993.

[18] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[19] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5041–5047.

[20] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[21] C.-b. Moon and W. Chung, "Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree," *IEEE Transactions on industrial electronics*, vol. 62, no. 2, pp. 1080–1090, 2014.

[22] G. Reinelt, "Tsplib—a traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.

[23] D. G. Macharet, A. A. Neto, V. F. da Camara Neto, and M. F. Campos, "Nonholonomic path planning optimization for dubins' vehicles," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4208–4213.

[24] D. G. Macharet and M. F. Campos, "A survey on routing problems and robotic systems," *Robotica*, vol. 36, no. 12, pp. 1781–1803, 2018.

[25] S. G. Manyam and S. Rathinam, "On tightly bounding the dubins traveling salesman's optimum," *Journal of Dynamic Systems, Measurement, and Control*, vol. 140, no. 7, p. 071013, 2018.

[26] M. P. Do Carmo, *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

[27] C. Bruns, ""pyopenvr," 2017.