

‘Teleportation’-Based Motion Planner for Design Error Analysis

Jesse C. Himmelstein, *Member, IEEE*, Etienne Ferre, Jean-Paul Laumond, *Fellow, IEEE*

Abstract—Probabilistic path planning techniques have proven to be vital for finding and validating solutions for difficult industrial assembly tasks. Nevertheless, the failure of a path planner to find a solution to a task does not suggest how to correct the error. We suggest a methodology to identify possible bottlenecks and present an algorithm to analyze the extent to which the design must be modified in order for the task to complete successfully. We validate our algorithm on two industrial problems involving design errors, and explain how to interpret the results in order to improve the design.

I. INTRODUCTION

AS the power of computers have grown, so too have the detail and complexity of the tasks put before them. Complex CAD/CAM models represent vast mechanical assemblies, encompassing detail at multiple orders of magnitude. Since computers are drawn on to visualize and manipulate this data, the drive is for them to be used for the majority of analysis as well. If design errors can be detected early along the product development process, they can be dealt with more efficiently. Our algorithm is meant to help in that process.

A. Motivation

In the domain of Product Lifecycle Management (PLM), path planning is typically used to study the feasibility of assembly or disassembly tasks. Such studies determine if a part will be mountable during the assembly process, or demountable for maintenance, and if so suggests how to proceed. Probabilistic path planning methods have enjoyed enormous success in solving these types of problems [1].

If a sufficiently sophisticated path planning algorithm is unable to propose a means to disassemble a mechanical part, then it is likely there is no such solution. Given the low cost of such detection compared to dealing with a physical scale model, it makes sense to use path planning in order to detect design errors as early and as regularly as possible.

Once an assembly task has been flagged at problematic, the nature of the problem is still unclear. Should the part itself be moved or redesigned, or is there another part of the assembly that should be examined? In the labyrinthine models that PLM designers often work with, the solution is anything but obvious.

In order to analyze such cases, we propose to allow

Jesse Himmelstein is a PhD student at LAAS- CNRS under a joint grant with Kineo CAM, 31670 Labège FRANCE (phone: +33 (0)5 61 00 90 60; fax: +33 (0)5 61 00 90 61; e-mail: jh@kineocam.com).

Etienne Ferré is with Kineo CAM (e-mail: ef@kineocam.com).

Jean-Paul Laumond is with LAAS-CNRS, 31077 Toulouse FRANCE (e-mail: jpl@laas.fr).

certain parts, called *active obstacles*, to move within specified boundaries. Such movement can open up passages that may have been inadvertently closed during the design process, and allow for a solution to be found. By analyzing the extent and direction of movement needed to find a solution, the user can gain valuable knowledge about the design error as well as possible ways to address it.

B. Related Work

As far as we have been able to determine, design error analysis has not been studied to a great degree by the scientific community. The closest related fields are probably assembly sequencing and geometric tolerancing, which are briefly introduced here.

Assembly sequencing addresses the problem of constructing a product out of its parts. Typically, a sequence specifies a series of assembly tasks for the parts, and possibly indicates which tasks can be carried out in parallel. This problem was most actively studied in the 1980s and early 1990s [2-7], although recent work also exists which uses probabilistic path planning in order to find sequences [8].

Geometric tolerancing attempts to account for inevitable production errors by assigning acceptable margins of error to features of mechanical parts in the design stage. Tolerances can be assigned for feature placement, size, as well as shape [9-13].

However, we are trying to address problems in design rather than in production. To do so, we draw on path planning strategies where obstacles are allowed to move within the scene [14]. In particular, Cortés *et al* have suggested a variant of the Rapidly-exploring Random Tree (RRT) [15] path planning algorithm aimed at molecular disassembly problems [16]. They model the extraction of a small ligand from a large protein. Their approach treats atoms as spheres and molecular bond torsions as degrees of freedom. A non-collision constraint models the van der Waals force. Parts of the protein, called side-chains, can be pivoted around an axis by the moving ligand.

In their algorithm, called ML-RTT, the movement of the ligand and the side-chains are decoupled. At each iteration, the planner first attempts to move the ligand. If the ligand motion is blocked by one or more side chains, then these side-chains positions are resampled and the planner tries to reconnect the new position to the current one. Paths resulting from the ML-RTT show the ligand “pushing open” the side-chains blocking its path much like barn doors.

Another way to analyze collisions is quantify the penetration between overlapping objects. The most common

measure is called Penetration Depth (PD) and relates the minimum distance in translation necessary to separate two objects [17]. For complex 3D objects, PD may be estimated based on distance fields that deform along with the geometry [18]. Recently, PD has been generalized to include rotations [19].

Penetration distance is generic to any type of geometry and trivial to parameterize. However, even with the fastest methods available to us today [19, 20], calculating it explicitly would overly burden the collision detection routines, which already take over 80% of computation time in typical path planning examples. From our practical experience with PLM cases, collision detection times must be on the order of 1 ms, and penetration distance calculations are currently unable to achieve this level of performance.

C. Contributions

We present a method for using an RRT path planner to suggest corrections for design errors in mechanical assemblies. If a designer is unable to disassemble a part, he or she can designate parts of the assembly that might be in the way as active obstacles, and attribute a play value to each. The algorithm will then analyze the problem and suggest solutions that include the active obstacle movement as well as that of the initial part.

Our algorithmic problem statement is the following: given a starting and ending configuration for a robot, and a list of active as well as passive obstacles, find a path that brings the robot from one configuration to the other without collision with any obstacles, but letting the active obstacles move up to their given limits (Fig. 1).

Our approach to this problem avoids the expensive calculation of PD and permits solution paths that would never be found by the ML-RRT alone.

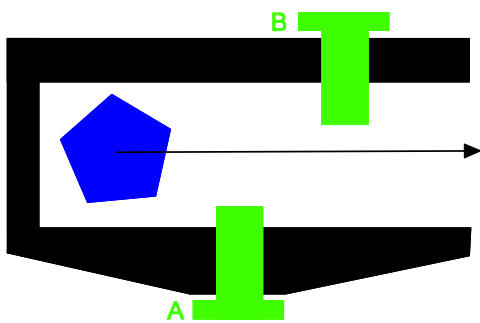


Fig. 1 Example disassembly problem with active obstacles. Extracting the blue pentagon from the assembly involves passing by two screws, A and B. There is no solution to this problem if the obstacles are not allowed to move. In order to apply our algorithm, we assign a starting and ending position to the pentagon, and specify that the two screws are active obstacles. The rest of the scene (the black objects) is considered as passive obstacles. Through our algorithm, we can determine that it is necessary to move screw B in order to extract the pentagon, but not screw A. Additionally, we can estimate the necessary movement of screw B.

D. Outline

The rest of this paper is organized as follows: Section II introduces how the RRT can be augmented to allow for penetration with active obstacles. This algorithm is analyzed and enhanced in Section III. We analyze experimental results in Section IV, and conclude with ideas for future work in Section V.

II. IMPLICITLY CONTROLLED PENETRATION

A. Penetration Distance and Movement Equivalent

Penetration distance is defined as a minimum translation distance needed to move one object out of collision with another. This implies that for a colliding position (x_0, y_0, z_0) there is a non-colliding position (x_1, y_1, z_1) so that the Euclidian distance between the two positions is equal to the penetration distance.

With this in mind, it is easy to see that given a colliding position and a distance p , if we are able to find a non-colliding position so that the distance between the two is less than p , the penetration distance must be less than or equal to p . Although the exact penetration distance is still unknown, it is at least bounded (Fig. 2).

The central idea of our approach is to fix p from the outset, and search for non-colliding positions that respect this distance in order to control the penetration. We call this distance the *play*. Each active obstacle is assigned a play, which determines the limits to which it can move throughout the path planning process.

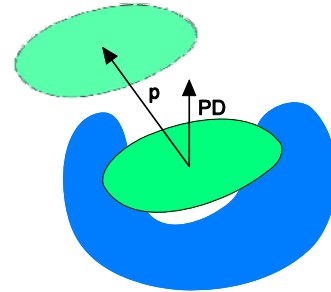


Fig. 2 Bounding PD by finding a non-colliding position. Given an ellipse in collision with an obstacle, there one or more PD vectors that provide the exact PD. Given a non-colliding position of the ellipse at a translation distance of p , however, we can conclude that $p \geq PD$. We call p the *play* of the obstacle.

B. Augmenting the Kinematic Chain

A robot can be modeled as a hierarchy of joints, also called a device or a kinematic chain. Joints can be of a variety of different types, such as rotations, translations, and combinations of these. Each joint has a number of degrees-of-freedom (DOF). Position calculation is done through kinematic relations, where each joint position is relative to its parent.

The first step of our process is to augment the original device with extra joints representing the active obstacles. To do so, we place a dummy joint called an anchor at the root of

the hierarchy. An anchor joint has no DOFs, but is used as a placeholder to contain any number of child joints. We then place the original root joint as the first child of the anchor joint.

The next step is to create joints for each active obstacle. Since we are only interested in translational PD, each active obstacle joint contains three translation DOFs that can offset the active obstacle from its initial position.

The magnitude of this offset is limited to be less than or equal to its play. That is, if the three DOFs are labeled x , y , and z , and the play is p , then

$$\|(x_1, y_1, z_1) - (x_0, y_0, z_0)\| \leq p \quad (1)$$

In other words, the movement of the center of an active obstacle is constrained to translation within a sphere of radius p around its initial position (Fig. 3).

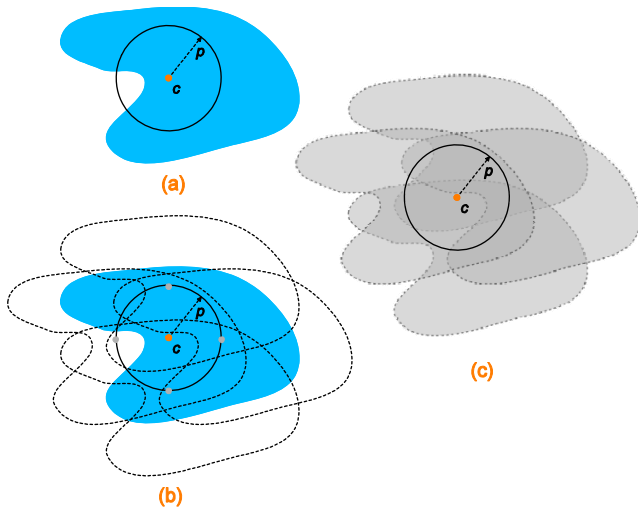


Fig. 3 The movement of the center of an active obstacle (the blue object) is limited to a sphere of size p around its initial position, c . The object and the sphere are shown in (a), and several possible positions are outlined in (b). The same is shown in (c), with semi-transparency.

C. Path Planning Algorithm

A basic RRT planner iteration can be decomposed as follows:

1. *Shoot*- Choose a random configuration in Configuration Space (CS).
2. *Pick*- Select the closest node to the shot configuration, using some distance metric.
3. *Extend*- Create a direct path between the picked node and the shot configuration, or to a configuration along the vector between the two.

With the augmented device in place, the only modification necessary in order for the basic RRT algorithm to function is in the *Shoot* stage. Active obstacle joint DOF values must fall within a sphere scaled to the play, so that the joint respects the distance constraint.

As shown in [21], if a solution to a path planning problem exists, the probability that the RRT algorithm will find it in a finite amount of time equals 1. Furthermore, the distribution of RRT vertices converges to the sampling distribution as

the algorithm progresses. Since the non-colliding active obstacle positions, and therefore the minimum penetration vectors, are chosen through the same process, this planner is probabilistically complete.

III. RRT ENHANCEMENTS

Theoretically, the changes discussed in Section II are enough for an RRT to solve path planning problems with active obstacles. Its performance, however, leaves much to be desired. By analyzing its behavior, we have developed several enhancements to overcome difficulties associated with our approach.

Our method involves creating an extra joint for each active obstacle in the scene. With three DOFs per joint, this can quickly multiply the dimensionality of the configuration space, invoking the dreaded “curse of dimensionality”. Thus, adding DOFs both introduces new solutions, and makes existing solutions harder to find.

In general, we have found that the best way to combat the curse of dimensionality brought on by the active obstacle joints is to treat them in different and more efficient manner, better adapted to their particularities.

A. Uniform Shooter

The first step of the RRT iteration, *Shoot*, involves choosing a configuration randomly within the configuration space. The simplest way to do is choose uniformly within the entire space. But for DOFs with no limits, it is difficult to randomly choose a value between negative and positive infinity. Instead, an enlarged local neighborhood around the existing roadmap can be used to approximate the known configuration space for non-limited DOFs.

This approach is valid for limited DOFs as well, as long as the enlarged local neighborhood is allowed to grow quickly enough. In such a case, the neighborhood has soon expanded to include both limits, assuming that no constraints prevented it from finding valid configurations near those points.

For active obstacle DOFs, there is no advantage in restricting the configuration choice to a local neighborhood around previous nodes. Additionally, since they cannot collide with the rest of the scene, there is no reason to prevent them from exploring all possible positions. In fact, a typical situation involves pushing an active obstacle DOF to its extreme value, a position more easily attainable through uniform sampling than local.

B. Distance Metric

The second technique that we propose is to modify the distance metric used by the *Pick* step of the RRT algorithm. The distance metric plays an important role, since the choice of which node will be extended towards the shot configuration can seriously impact performance. Any interpolation between a picked node and a shot configuration can lead to a collision, and by picking nodes that are “closer” to the shot configurations, this possibility can be minimized.

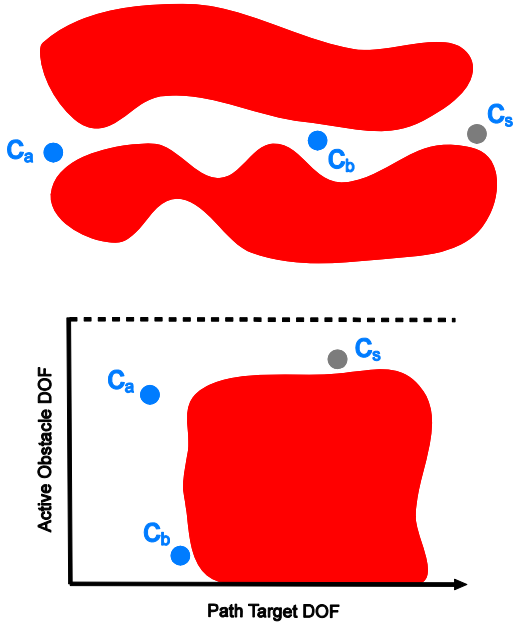


Fig. 4 Role of the distance metric. In cases where the active obstacles do not play a large role, such as a narrow passage example (top), then using their positions to pick the closest node leads to bad choices. The shot position c_s could be considered closer to c_a than c_b if one or more active obstacle positions happened to be closer, although only the robot position matters. In an opposite example (bottom), only basing a choice on the robot DOF could lead the planner to pick c_b instead of c_a , although the latter would be more practical in order to follow the passage opened up by the active obstacle.

Although many distance metrics can be used, a common choice for robotic systems is a scaled DOF difference. With this metric, each DOF d is associated with a weight w . The distance between two configurations c^0 and c^1 is thus:

$$\sqrt{\sum_d [w_d(c_d^1 - c_d^0)]^2} \quad (2)$$

One advantage of this technique is that DOFs can be given higher or lower weight in accordance with their “importance,” whether geometrical, mechanical, or otherwise defined by the application.

By default, active obstacle DOFs had the same weight as other translation DOFs in the device. But when the number of active obstacle DOFs becomes larger than that of the robot, this leads to the planner is taking more account of the obstacles than of the object we are trying to plan for. In such a case, the planner will pick nodes for which the robot is arbitrarily far away from the shot configuration, only because the obstacles are closer (Fig. 4).

The previous scenario suggests that active obstacle DOF weights should be reduced as much as possible. Nevertheless, setting them to zero, while an improvement over equal weighting, is actually counterproductive. Since active obstacle positions are shot randomly, it is worthwhile to save those lead to roadmap growth. In typical examples, the obstacles are “pushed” out of the way by the robot. Once out of the way, they can be left there to allow further maneuvering.

Therefore a balance must be struck. In practice, we’ve found that very low but positive weights work best, several orders of magnitude less than the normal robot DOF weights.

C. Teleportation

The third enhancement that we can bring to the RRT algorithm relates to the interpolation between successive configurations. As the output of an RRT is only a sequence of configurations, the interpolation provides object positions between configurations.

The interpolation used for robot movement depends on the context, but it is safe to assume that almost all are continuous, in that there are no sudden “jumps” from one position to another. By default, the active obstacle movement inherits this behavior, smoothly going from one configuration to the next. For example, a simple linear interpolation from c^0 to c^1 with the variable x in range $[0, 1]$ can be expressed as follows:

$$h_{linear}(c^0, c^1, x) = c^0 + x(c^1 - c^0) \quad (3)$$

Since we are not attempting to simulate physical movement of the active obstacles, continuous interpolation leads to overly conservative behavior. As described in Section II.A, only a single active obstacle position is necessary to validate the PD constraint. Therefore, interpolating the active obstacle positions, although giving valid results, provides no more information than simply choosing a single position (i.e. the position being interpolated to). As Fig. 5 illustrates, interpolating the active obstacle DOFs linearly often creates more possibilities for collision rather than reducing them.

Taking this analysis into account, we’ve adapted a hybrid interpolation, in which the robot DOFs are interpolated normally (e.g. with h_{linear}) but the active obstacle DOFs are interpolated with the following function:

$$h_{jump}(c^0, c^1, x) = \begin{cases} c^0, & x = 0 \\ c^1, & x > 0 \end{cases} \quad (4)$$

Through its discontinuity, this interpolation method leads to much faster exploration, as well as introducing certain maneuvers that could not arise in with a continuous method, such as the active obstacle “jumping over” the robot. Such behavior is counterintuitive from a path-planning perspective, but is allowed by our PD constraint.

D. Manhattan-like RRT

The ML-RRT uses a Manhattan-like method to connect a node to a shot configuration in the *Extend* step of an RRT iteration [16]. The step is broken up into two parts: first the robot motion is dealt with, followed by the obstacle motion. With a few adjustments, we’ve adopted the ML-RRT to the active obstacle context.

1) Adapted Algorithm

In the first part, the motion of the robot is tested progressively along the interpolation between the node and the shot configuration. If no collision was detected, then a path segment from the node and the shot is added to the roadmap. If a collision was found, then the added segment

only reaches until the last non-colliding configuration tested.

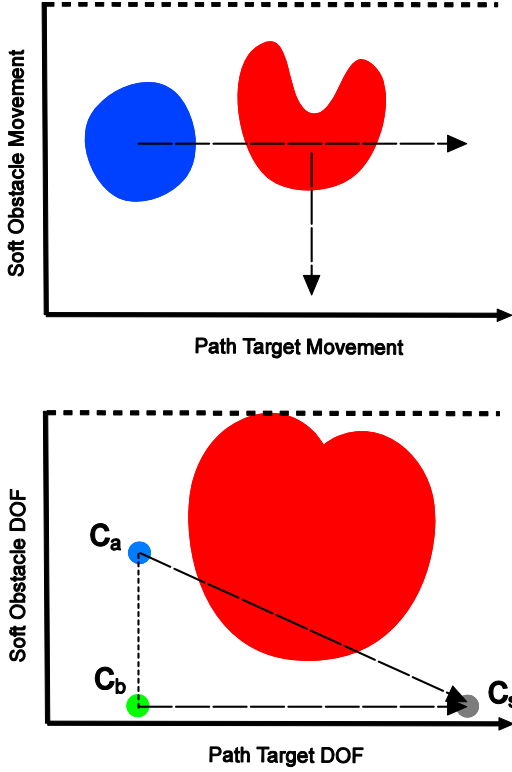


Fig. 5 Discontinuous interpolations give better performance. Translating the blue object to the right along a single axis leads to a collision with the red obstacle, even if the obstacle itself linearly interpolated towards the bottom (above). The same situation can be illustrated in configuration space (below) in which a linear interpolation from c_a to c_s leads to collision with the obstacle. By adapting a hybrid interpolation in which the active obstacle instantaneously jumps to c_b while the robot is interpolated smoothly, this collision can be avoided.

If no collision occurred in the first part, then the second part is skipped. Otherwise, if the collision occurred only with active obstacles, then the planner attempts to add a path segment connecting the previously added node to a new configuration in which only those active obstacles are in movement (Fig. 6).

The algorithm can be succinctly described using a function m that combines two configurations into one, based on the membership of each DOF. If a DOF belongs to an object provided in the set L , then receives its value from the second configuration. Otherwise, it takes its value from the first.

$$m(c^0, c^1, L) = \left\{ (d_0, \dots, d_n) \mid d_i = \begin{cases} c_d^1, & d \in L \\ c_d^0, & d \notin L \end{cases} \right\} \quad (5)$$

Let the function $expand(from, to)$ progressively test an interpolation for collision, and return the last non-colliding configuration found as well and the set of colliding obstacles (L_C). Let the set SO contain all the active obstacles, and PT the robot. Now the Manhattan extend step can be written as Algorithm 1.

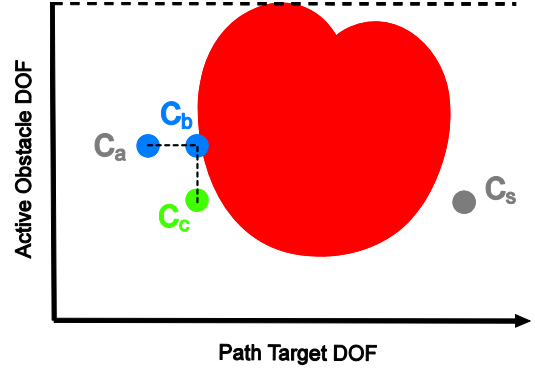


Fig. 6 Manhattan-like RRT. The extension of c_a to c_s is broken up into two components. First, the robot DOF is interpolated until a collision is detected, and a node posed at that point, c_b . Next, the movement of all colliding active obstacles is interpolated, leading to the segment from c_b to c_c .

By decoupling the movement of the robot from the active obstacles, the active obstacles positions are not modified as long as they do not collide with the robot. When they do, they tend to move only to allow the robot to pass, giving the impression that the robot is physically pushing them. Once an active obstacle is moved out of the way, it tends to stay in that position, which is advantageous to the planner.

2) Combining with a Discontinuous Interpolation

Since the ML-RRT extension step breaks up the interpolation into two parts, it reduces the effect of a discontinuous interpolation as described in Section III.C. As the active obstacle DOFs are not touched by the first segment, the “jump” has no effect on it. The second segment will have the discontinuity, but not use it to further explore the space.

```

1. procedure extend(node, shot)
2.    $c^m \leftarrow m(\text{node}, \text{shot}, PT)$ 
3.    $(c^{pt}, L_C) \leftarrow \text{expand}(\text{node}, c^m)$ 
4.   if  $c^{pt} \neq \text{node}$  then
5.     addEdge(node,  $c^{pt}$ )
6.     node  $\leftarrow c^{pt}$ 
7.   end if
8.   if  $L_C \neq \emptyset$  and  $L_C \subseteq SO$  then
9.      $c^m \leftarrow m(\text{node}, \text{shot}, L_C)$ 
10.     $(c^{so}, L_C) \leftarrow \text{expand}(\text{node}, c^m)$ 
11.    if  $c^{so} \neq \text{node}$  then
12.      addEdge(node,  $c^{so}$ )
13.    end if
14.  end if
15. end procedure

```

Algorithm 1 Manhattan-like RRT for active obstacles. The traditional *Extend* step is broken into two parts. In lines 2-7, only the robot is moved along the interpolation. If no collision was detected, or if a collision with non-active obstacles occurred, then only a single edge is added. Otherwise, an additional edge is added in which only the active obstacles are moved (lines 8-14).

We can improve efficiency by having the two work in concert. It is necessary to modify the second edge that the ML-RRT attempts to add so that the robot DOFs vary along with those of the colliding active obstacles (Fig. 7). The only adjustment to the previous algorithm is on line 9.

$$9. \quad c^m \leftarrow m(\text{node}, \text{shot}, L_c \cup \text{PT})$$

Algorithm 2 Change to ML-RRT to better include the discontinuous interpolation.

The combination of the ML-RRT and the discontinuous interpolation then retains the interests of both, as only the active obstacles in collision are moved, but the space is explored rapidly without enforcing a “physically realistic” maneuver.

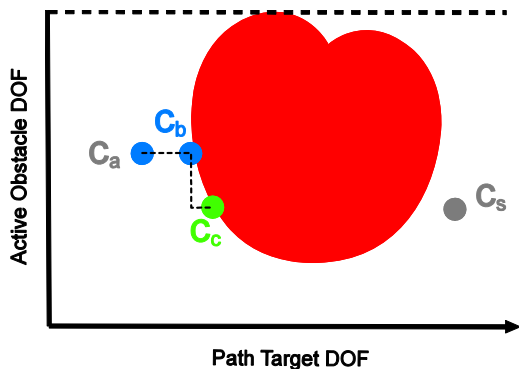


Fig. 7 Manhattan-like RRT combined with discontinuous interpolation. By including the robot in the objects interpolated for the second segment that the ML-RRT attempts to add, the configuration c_c stays much tighter to the obstacle. In this example, with only one active obstacle, the second segment tested by the planner is from c_b to c_s , leading to the node c_c where it collides with the obstacle.

IV. EXPERIMENTAL RESULTS

We tested our approach on two real industrial assembly cases. The first involves a car windshield wiper, for which there is no solution for extracting a wiper from its holder. Through specifying a suspect part as an active obstacle and running our algorithm, a disassembly path is found (Fig. 8). Now that the offending part has been identified, we can move on to characterizing the design error and suggest ways to correct it.

To do so, we start with a high play for the active obstacle, solve for a path, and then reduce the play and solve again. This results in a series of increasingly difficult set of path planning problems. For the wiper, we were unable to find a solution at less than 4.2mm. Therefore, we can conclude that this is close to the minimum play necessary to let the wiper pass.

The second case considers an automobile exhaust system. The design flaw in this case is the placement of a bracket that closes the already narrow passage (Fig. 9). Analysis with our algorithm reveals that a play value of 10mm lets the object pass.

To obtain performance results, we ran our algorithm 10 times at different play values and took the mean, median, and standard deviation. The results are listed in Table I.

TABLE I
PERFORMANCE RESULTS

Model	Play (mm)	Mean (s)	Median (s)	Std. Dev. (s)
Wiper	10	55.57	47.75	27.03
Wiper	7	544.38	600.57	328.59
Exhaust	15	975.17	935.26	735.84
Exhaust	10	1973.71	634.95	935.26

It is clear that the lower the play, the more difficult the problem becomes. This can explain the large differences in performance with the identical scenario at different levels of play. The trend is evident despite the large standard deviations, which are a general problem with probabilistic path planners [22], and make precise comparison difficult.

V. CONCLUSION AND FUTURE WORK

We have introduced an algorithm to help mechanical designers identify and treat flaws in complex CAD assemblies. It requires only minor changes to a basic RRT path planner and applies to any type of geometry, making it an ideal candidate for inclusion in industrial analysis software.

Alternative techniques for confronting our problem statement include explicit PD calculation, and using the ML-RRT by itself. Compared to the first, our approach is both faster and places less restrictions on the geometry being tested (many CAD models are disconnected and malformed triangle soups). On the other hand, it only provides an upper bound on the PD, rather than an exact answer. As compared to the ML-RRT taken alone, our approach permits a larger range of solutions (through teleportation) and finds them faster (through the adjusted distance metric). The discontinuous interpolations would be inappropriate, however, if only physically possible movements for the obstacles are desired.

In terms of completeness, our approach benefits from the guaranteed convergence of the RRT to find a solution if it exists. Also like the RRT, it is incapable of proving the absence of a solution.

An interesting topic for future work concerns how best to handle multiple active obstacles that are very close to one another. Such a case multiplies the difficulty of the problem by forcing the planner to deal with several moving obstacles at once instead of in series. It would be interesting to have the planner intelligently group the active obstacles based on their relative positions in order to manipulate several at once.

Another idea, and a subject of current work, is shift the task of moving the active obstacles onto the collision checker itself. This could help performance as well as make it more easily applicable to other path planning methods, such as PRM.

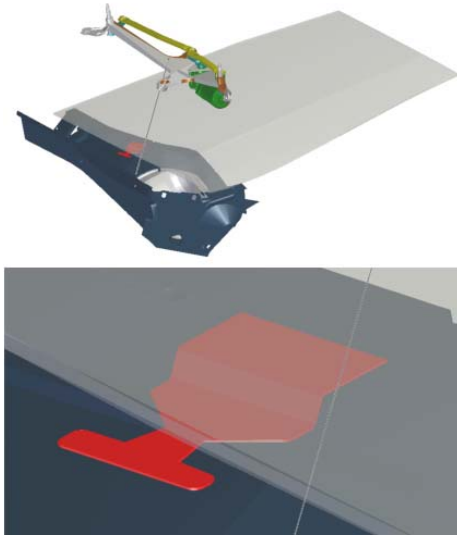


Fig. 8 Windshield wiper test case. The goal is to assemble the wiper into its holder (top). The close-up (bottom) reveals the red plate that prevents this task from being carried out. This design error can be fixed with the help of our algorithm, once the red part is made into an active obstacle.

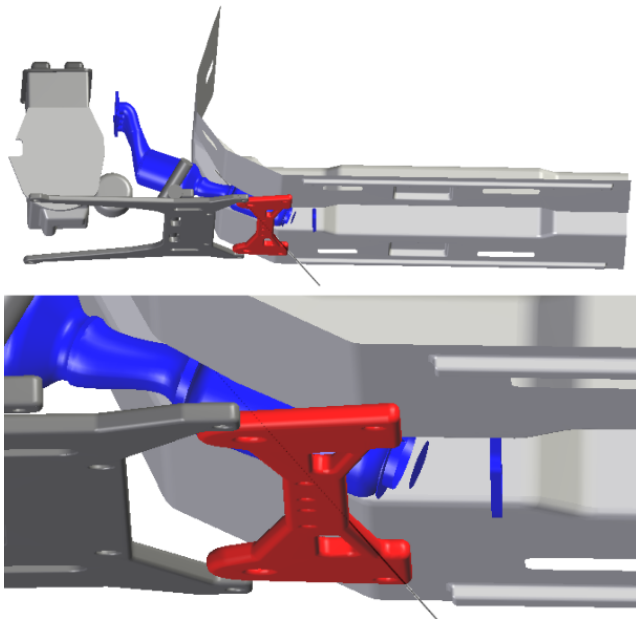


Fig. 9 Exhaust test case. Extracting the blue exhaust system (top) is prevented by the red part, shown in the close-up (bottom). By declaring the red part an active obstacle, we can evaluate the necessary movement to correct the design error.

REFERENCES

[1] J.-P. Laumond, "Motion planning for PLM: state of the art and perspectives," *International Journal of Product Lifecycle Management*, vol. 1, pp. 129-142, 2006.
 [2] J.-C. Latombe and R. H. Wilson, "Assembly Sequencing with Toleranced Parts," in *Proceedings of ACM Symposium on Solid and*

Physical Modeling, 1995, pp. 83-94.
 [3] K. T. Seow and R. Devanathan, "A Temporal Framework for Assembly Sequence Representation and Analysis," *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 220-229, 1994.
 [4] R. H. Wilson and J.-C. Latombe, "Geometric Reasoning about Mechanical Assembly," *Artificial Intelligence*, vol. 71, pp. 371-396, 1995.
 [5] A. Delchambre, "A pragmatic approach to computer-aided assembly planning," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1990, pp. 1600-1605.
 [6] J. M. Milner, S. C. Graves, and D. E. Whitney, "Using simulated annealing to select least-cost assembly sequences," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1994, pp. 2058-2063.
 [7] T. Cao and A. C. Sanderson, "Task Decomposition And Analysis of Assembly Sequence Plans Using Petri Nets," in *Proceedings of Computer Integrated Manufacturing*, 1992, pp. 138-147.
 [8] S. Sundaram, I. Remmler, and N. M. Amato, "Disassembly Sequencing Using a Motion Planning Approach," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2001, pp. 1475-1480.
 [9] L. Joskowicz, E. Sacks, and V. Srinivasan, "Kinematic Tolerance Analysis," *Computer-Aided Design*, vol. 29, pp. 147-157, 1997.
 [10] C. K. Yap and E.-C. Chang, "Issues in the Metrology of Geometric Tolerancing," in *Algorithms for Robot Motion Planning and Manipulation*, J.-P. L. a. M. Overmars, Ed. Wellesley, Massachusetts, USA: A.K. Peters, 1997, pp. 393-400.
 [11] R. Jayaraman and V. Srinivasan, "Geometric Tolerancing: I. Virtual Boundary Requirements," *IBM Journal of Research and Development*, vol. 33, pp. 90-104, 1989.
 [12] M. Inui, M. Miura, and F. Kimura, "Analysis of Position Uncertainties of parts in an Assembly using Configuration Space in Octree Representation," in *Proceedings of ACM Symposium on Solid and Physical Modeling*, 1995, pp. 73-82.
 [13] V. Srinivasan and R. Jayaraman, "Geometric Tolerancing: II. Conditional Tolerances," *IBM Journal of Research and Development*, vol. 33, pp. 105-124, 1989.
 [14] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of Symposium on Computational Geometry*, 1988, pp. 279-288.
 [15] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Computer Science Dept., Iowa State University TR 98-11*, 1998.
 [16] J. Cortés, L. Jaillet, and T. Siméon, "Molecular Disassembly with RRT-like Algorithms," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 3301-3306.
 [17] S. A. Cameron and B. Culley, "Determining the Minimum Translational Distance between Two Convex Polyhedra," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1986, pp. 591-596.
 [18] S. Fisher and M. C. Lin, "Deformed distance fields for simulation of non-penetrating flexible bodies," in *Proceedings of Eurographic Workshop on Computer Animation and Simulation*, 2001, pp. 99-111.
 [19] L. Zhang, Y. J. Kim, and D. Manocha, "A Fast and Practical Algorithm for Generalized Penetration Depth Computation," in *Proceedings of Robotics: Science and Systems Conference (RSS)*, 2007.
 [20] Y. J. Kim, M. C. Lin, and D. Manocha, "Fast Penetration Depth Estimation Using Rasterization Hardware and Hierarchical Refinement," in *Proceedings of Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2002.
 [21] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 995-1001.
 [22] P. Isto, M. Mäntylä, and J. Tuominen, "On addressing the run-cost variance in randomized motion planners," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 2934-2939.