Disassembly Path Planning for Complex Articulated Objects

Juan Cortés, Léonard Jaillet, Thierry Siméon

***Abstract*—Sampling-based path planning algorithms are powerful tools for computing constrained disassembly motions. This paper presents a variant of the RRT algorithm particularly devised for the disassembly of objects with articulated parts. Configuration parameters generally play two different roles in this type of problems: some of them are essential for the disassembly task, while others only need to move if they hinder the progress of the disassembly process. The proposed method is based on such a partition of the configuration parameters. Results show a remarkable performance improvement compared to standard path planning techniques. The paper also shows practical applications of the presented algorithm in robotics and structural bioinformatics.**

***Index Terms*—Path planning algorithms, Disassembly paths, Articulated mechanisms, Robotic manipulation, Molecular interactions.**

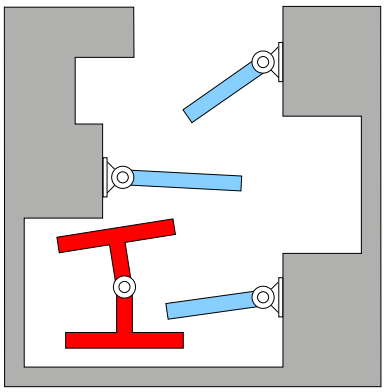

Fig. 1. Disassembly path planning problem for two objects with articulated parts. The problem consists in finding a path to extract the small (red/dark) object from the big one.

I. Introduction

THIS paper[1] addresses the problem of automatically computing motions to disassemble objects. The problem can be formulated as a general path planning problem [2], [3] (see Section III). Indeed, path planning concepts and algorithms have been applied to solve different instances of the (dis)assembly planning problem (see Section II). The instance treated in this paper considers two objects, with the particularity that both objects may have multiple articulated parts. Fig. 1 illustrates a simple two-dimensional example.

The algorithm presented in this paper is a variant of a sampling-based path planning method: the RRT algorithm introduced in [4]. Section IV reminds the principle of this method. Sampling-based path planners are efficient, general and easy-to-implement methods. The RRT algorithm has been widely studied and applied to different types of problems in the last years (see `http://msl.cs.uiuc.edu/rrt/` for a general survey). The particularity of the proposed variant is to introduce two types of configuration parameters, labeled as *active* and *passive*, and to generate their motion in a decoupled manner. We call this variant Manhattan-like RRT (ML-RRT) because the computed paths look like Manhattan paths over these two sets of parameters that change alternatively. The ML-RRT algorithm is explained in Section V. The partition of the configuration parameters into active and passive corresponds to their role in the disassembly problem. Active parameters are essential for the disassembly task, while passive parameters only need to move if they hinder the progress of the disassembly process. The ML-RRT algorithm presents two main advantages with respect to the basic RRT: (1) The computing time is notably reduced (see results in Section VI). (2) The passive parts that have to move for finding a solution path are automatically identified. Thus, the planner is able to handle models involving hundreds of potential degrees of freedom, avoiding user intervention to select the important ones. This feature is particularly interesting for one of the applications commented in Section VII: the simulation of molecular interactions. Besides this application in structural bioinformatics, the ML-RRT algorithm is applicable to more classic domains involving part disassembly, such as Product Lifecycle Management (PLM) [5]. Moreover, the algorithm can be easily extended for integrating the constraints imposed by the handling device (e.g. a robotic manipulator) that carries out the disassembly task (see Section VII). Other possible extensions are outlined in Section VIII.

The authors are with the LAAS-CNRS, Université de Toulouse, France.

[1]A preliminary version of this paper appeared in [1].

II. Related Work

Assembly and disassembly planning are important problems in manufacturing engineering. Many techniques have been developed in this field for automatically generating (dis)assembly plans that optimize time, cost, etc. [6], [7]. Most of these techniques are based on *relation graph models* of the assembly or *precedence graphs* and use graph theory and AI algorithms for computing disassembly sequences. Geometric reasoning approaches have been proposed for reducing the combinatorial complexity of the problem, as well as the amount of information that has to be provided by the user. Wilson's pioneering work on geometric reasoning about mechanical assembly [8] introduces the *directional blocking graph* (DBG), which identifies which parts collide given an instantaneous displacement in a given direction, and the

non-directed blocking graph (NDBG), which represents how parts are constraining each other, based on a partition of the space of allowed motions and on the associated DBGs. Many subsequent (dis)assembly sequencing methods have used these two concepts. The approach presented in [9] generalizes the solutions in [8] to arbitrary motions between parts. The method involves constructing configuration space diagrams that explicitly represent interferences between pairs of parts for every relative motion. A similar approach is developed in [10], based on the concept of *motion space*, which is an extension of the notion of configuration space, and represents possible motions of sub-assemblies. More recently, disassembly sequencing planners have been proposed based on randomized path planning algorithms. Contrary to the aforementioned planners, these methods are not complete, but they are able to treat complex models thanks to their computational efficiency. The technique presented in [11] constructs a disassembly tree rooted at the starting (assembled) configuration using a randomized diffusion strategy. The sampling of movement directions is biased using geometric information (e.g. face normals) for improving performance.

The assembly maintainability study [12] is a variant of the (dis)assembly problem. Given an assembled system, maintainability studies are conducted to determine if it is possible to remove a particular part, and if so, to obtain the disassembly path. Normally, such studies involve only one mobile part, and therefore, "standard" path planning algorithms could be applied. However, the workspace is usually extremely constrained in this context, and problem-specific algorithms are required for efficiently computing disassembly paths. A fast and effective algorithm for this kind of problems is presented in [5]. The method is based on an iterative RRT-like algorithm that reconstructs some parts of the search tree while progressively increasing the size of the objects.

All the methods above address (dis)assembly problems involving rigid objects. The method we present in this paper is well suited for assembly maintainability studies in which the disassembled objects have articulated parts.

III. Problem Formulation

The disassembly path planning problem can be formulated as a general path planning problem for a system with multiple mobile objects, using the notion of *configuration space* $\boldsymbol{C}$ [13], [2], [3]. A *configuration* $\mathbf{q}$ is a minimal set of parameters defining the location of the mobile system in the world, and $\boldsymbol{C}$ is the set of all the configurations. Given the assembled configuration $\mathbf{q}_{\text{init}}$ and a goal configuration $\mathbf{q}_{\text{goal}}$ (any disassembled configuration) the problem consists in finding a feasible collision-free path in $\boldsymbol{C}$ that connects both configurations.

The instance studied in this paper considers two objects with possibly multiple articulated parts. Considering that the spatial location of one of the objects is fixed, then, the configuration parameters are those defining the pose of the reference frame attached to the other (mobile) object plus the degrees of freedom associated with the articulated parts in both objects. Thus, the configuration vector is given by: $\mathbf{q} = \{\mathbf{q}^{\text{M}},\mathbf{q}^{\text{Jm}},\mathbf{q}^{\text{Js}}\}$, where $\mathbf{q}^{\text{M}}$ contains parameters defining the position and the orientation of the mobile reference frame, and $\mathbf{q}^{\text{Jm}}$ and $\mathbf{q}^{\text{Js}}$ represent the joint variables of the articulated parts in the mobile object and the static object respectively.

In general, the most significant parameters for the disassembly of articulated objects are those concerning the pose of the mobile object, $\mathbf{q}^{\text{M}}$. The parameters associated with the articulated parts are relatively less important, since they only need to move if they hinder the progress of the mobile object toward the disassembled configuration. Therefore, configuration parameters can be separated into two sets: $\mathbf{q} = \{\mathbf{q}^{\text{act}},\mathbf{q}^{\text{pas}}\}$, with $\mathbf{q}^{\text{act}} = \mathbf{q}^{\text{M}}$ representing the active parameters, and $\mathbf{q}^{\text{pas}} = \{\mathbf{q}^{\text{Jm}},\mathbf{q}^{\text{Js}}\}$ the passive parameters. The terms active and passive have been chosen in relation to how the algorithm described in Section V acts on them. This partition induces the corresponding sub-manifolds in the configuration space: $\boldsymbol{C} = \boldsymbol{C}^{\text{act}} \times \boldsymbol{C}^{\text{pas}}$.

Although the above described partition can be generally adopted, any other partition can be defined by the user. The mobile parts are separated into two lists L_{act} and L_{pas} containing the active and the passive parts respectively. For a given partition, $\mathbf{q}^{\text{act}}$ is the set of configuration parameters associated with the parts in L_{act} and $\mathbf{q}^{\text{pas}}$ is is the set associated with L_{pas}.

IV. The Basic RRT Algorithm

The basic principle of the RRT algorithm [4] is to incrementally grow a random tree τ rooted at the initial configuration $\mathbf{q}_{\text{init}}$ in order to explore the reachable configuration space and to find a feasible path connecting $\mathbf{q}_{\text{init}}$ to a goal configuration $\mathbf{q}_{\text{goal}}$. Fig. 2 illustrates the process and Algorithm 1 gives the pseudo-code for the RRT construction. At each iteration, the tree is expanded toward a randomly sampled configuration $\mathbf{q}_{\text{rand}} \in \boldsymbol{C}$. This random sample is used to simultaneously determine the tree node to be expanded and the direction in which it is expanded. Given a distance metric in the configuration space, the nearest node $\mathbf{q}_{\text{near}}$ in the tree to the sample $\mathbf{q}_{\text{rand}}$ is selected and an attempt is made to expand $\mathbf{q}_{\text{near}}$ in the direction of $\mathbf{q}_{\text{rand}}$. For kinematically unconstrained systems, the expansion procedure can be simply performed by moving on the straight-line segment between $\mathbf{q}_{\text{near}}$ and $\mathbf{q}_{\text{rand}}$. If the expansion succeeds, a new node $\mathbf{q}_{\text{new}}$ and a feasible local path from $\mathbf{q}_{\text{near}}$ are generated. The key idea of this expansion strategy is to bias the exploration toward unexplored regions of the space. Hence, the probability that a node will be chosen for an expansion is proportional to the volume of its Voronoi region (i.e. the set of points closer to this node than to any other node). Therefore, RRT expansion is biased toward large Voronoi regions enabling rapid exploration before uniformly covering the space.

Different strategies can be adopted for the design of RRT-based path planners [14]. Configuration sampling (function `SampleConf`) is normally made using a uniform random distribution in the configuration space $\boldsymbol{C}$. However, more sophisticated sampling strategies (e.g. [15], [16]) may improve the performance of the RRT algorithm. Another technical point concerns the function `BestNeighbor`. The basic RRT

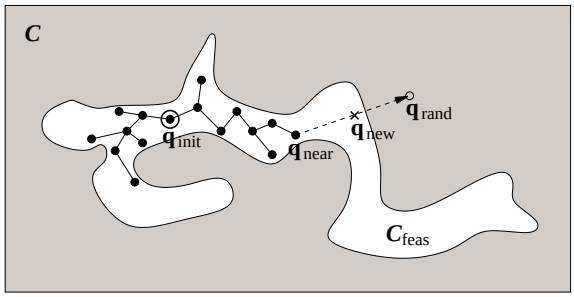

Fig. 2. Illustration of one expansion step of an RRT search tree. The tree tends to cover $\boldsymbol{C}_{\text{feas}}$: the feasible subset of the configuration space $\boldsymbol{C}$.

Algorithm 1: Construct_RRT

input : the configuration space $\boldsymbol{C}$;
the root $\mathbf{q}_{\text{init}}$ and the goal $\mathbf{q}_{\text{goal}}$;
output : the tree τ;
begin
  $\tau \leftarrow$ `InitTree`($\mathbf{q}_{\text{init}}$);
  while not `StopCondition`(τ, $\mathbf{q}_{\text{goal}}$) **do**
    $\mathbf{q}_{\text{rand}} \leftarrow$ `SampleConf`($\boldsymbol{C}$);
    $\mathbf{q}_{\text{near}} \leftarrow$ `BestNeighbor`(τ, $\mathbf{q}_{\text{rand}}$);
    $\mathbf{q}_{\text{new}} \leftarrow$ `Expand`($\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{rand}}$);
    if not `TooSimilar`($\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$) **then**
      `AddNewNode`(τ, $\mathbf{q}_{\text{new}}$);
      `AddNewEdge`(τ, $\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$);
end

algorithm selects $\mathbf{q}_{\text{near}}$ as the nearest node to $\mathbf{q}_{\text{rand}}$ using an Euclidean metric[2] in $\boldsymbol{C}$. Such a metric distance is very simple and easy to compute. However, since it does not consider motion constraints (e.g. obstacles, kinematic constraints), it may lead to a poor performance of the planner, by repeatedly selecting "exhausted" nodes for futile expansion. To avoid this problem, two modifications can be introduced in `BestNeighbor`: (1) A node is no longer selected after its expansion fails a given number of consecutive times l. (2) $\mathbf{q}_{\text{near}}$ is selected at random among the k nearest neighbors[3]. The efficiency of these two modifications has been shown in related works [18], [19], [20]. One can also choose a more or less greedy strategy for the expansion procedure (function `Expand` in Algorithm 1). In the basic RRT algorithm, a single expansion step of fixed distance is performed. Here we use the RRT-Connect variant [14], which iterates the expansion step while feasibility constraints are satisfied. This variant is in general more efficient than the single-step version for systems without differential constraints, which are the type of systems considered in this paper.

[2]We use a weighted metric for translation and rotation components, with 3D rotations represented by Euler angles. Note however that the use of unit quaternions will be more appropriate [17].

[3]In our implementation, l is a constant with default value equal to 10, and k is computed as $n_{\text{node}}/100$ rounded to the nearest upper integer, where n_{node} is the current number of nodes in the tree. These values have been empirically determined.

Algorithm 2: Construct_ML-RRT

input : the configuration space $\boldsymbol{C}$;
the root $\mathbf{q}_{\text{init}}$ and the goal $\mathbf{q}_{\text{goal}}$;
the partition $\{L_{\text{act}}, L_{\text{pas}}\}$;
output : the tree τ;
begin
  $\tau \leftarrow$ `InitTree`($\mathbf{q}_{\text{init}}$);
  while not `StopCondition`(τ, $\mathbf{q}_{\text{goal}}$) **do**
    $\mathbf{q}_{\text{rand}}^{\text{act}} \leftarrow$ `SampleConf`($\boldsymbol{C}$, L_{act});
    $\mathbf{q}_{\text{near}} \leftarrow$ `BestNeighbor`(τ, $\mathbf{q}_{\text{rand}}^{\text{act}}$, L_{act});
    ($\mathbf{q}_{\text{new}}$, $L_{\text{pas}}^{\text{col}}$) $\leftarrow$ `Expand`($\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{rand}}^{\text{act}}$);
    if not `TooSimilar`($\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$) **then**
      `AddNewNode`(τ, $\mathbf{q}_{\text{new}}$);
      `AddNewEdge`(τ, $\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$);
      $\mathbf{q}_{\text{near}} \leftarrow \mathbf{q}_{\text{new}}$;
    while $L_{pas}^{col} \neq \emptyset$ **do**
      $\mathbf{q}_{\text{rand}}^{\text{pas}} \leftarrow$ `PerturbConf`($\boldsymbol{C}$, $\mathbf{q}_{\text{near}}$, $L_{\text{pas}}^{\text{col}}$);
      ($\mathbf{q}_{\text{new}}$, $L_{\text{pas}}^{\text{col'}}$) $\leftarrow$ `Expand`($\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{rand}}^{\text{pas}}$);
      if not `TooSimilar`($\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$) **then**
        `AddNewNode`(τ, $\mathbf{q}_{\text{new}}$);
        `AddNewEdge`(τ, $\mathbf{q}_{\text{near}}$, $\mathbf{q}_{\text{new}}$);
        $\mathbf{q}_{\text{near}} \leftarrow \mathbf{q}_{\text{new}}$;
      $L_{\text{pas}}^{\text{col}} \leftarrow L_{\text{pas}}^{\text{col'}} \setminus L_{\text{pas}}^{\text{col}}$;
end

V. The ML-RRT Variant

This section presents a variant of the RRT algorithm that considers the active/passive partition of the configuration parameters introduced in Section III. The algorithm, called Manhattan-like RRT (ML-RRT), computes the motion of the parts associated with both parameter types in a decoupled manner. Active parameters are directly handled by the planner, while passive parameters are treated only when required to expand the tree. Indeed, passive parts only move if they hinder the motion of other mobile parts (active parts or other passive parts involved in the expansion).

The ML-RRT algorithm is schematized in Algorithm 2. At each iteration, the motion of active parts is computed first. The function `SampleConf` receives as argument the list of active parts L_{act} and it only samples the associated parameters $\mathbf{q}^{\text{act}}$. Thus, this function generates a configuration $\mathbf{q}_{\text{rand}}^{\text{act}}$ in a sub-manifold of the configuration space involving the active parameters, $\boldsymbol{C}^{\text{act}}$. The function `BestNeighbor` selects the node to be expanded $\mathbf{q}_{\text{near}}$ using a distance metric in $\boldsymbol{C}^{\text{act}}$. Note that the function `BestNeighbor` also integrates the basic improvements mentioned in Section IV. Then, `Expand` performs the expansion of the selected configuration by only changing the active parameters. A greedy strategy is used. The returned configuration $\mathbf{q}_{\text{new}}$ corresponds to the last valid point computed along the straight-line segment from $\mathbf{q}_{\text{near}}$ toward $\{\mathbf{q}_{\text{rand}}^{\text{act}}, \mathbf{q}_{\text{near}}^{\text{pas}}\}$. If the expansion is not negligible, a new node and a new edge are added to the tree. The function `Expand` also analyzes the collision pairs yielding the stop of the expansion process. If active parts in L_{act} collide with potentially mobile passive parts in L_{pas}, the list of the involved passive parts $L_{\text{pas}}^{\text{col}}$ is returned. This information is used in the second stage of

the algorithm, which generates the motion of passive parts. The function `PerturbConf` generates a configuration $\mathbf{q}_{rand}^{pas}$ by randomly sampling the value of the passive parameters associated with L_{pas}^{col} in a ball around their configuration in $\mathbf{q}_{near}$. Note that, if the previous call to `Expand` has been successful, $\mathbf{q}_{near}$ has been updated in order to contain the new configuration of the active parameters. An attempt is then made to further expand $\mathbf{q}_{near}$ toward $\{\mathbf{q}_{near}^{act},\mathbf{q}_{rand}^{pas}\}$. Only the parts in L_{pas}^{col} move during this tree expansion. The function `Expand` returns a list $L_{pas}^{col'}$ if the expansion is stopped by a collision involving passive parts. If this list contains new passive parts (in relation to L_{pas}^{col}), the process generating passive part motions is iterated. Such a possible cascade of passive part motions may be useful to solve problems where passive parts indirectly hinder the motion of the active ones because they block other passive parts.

Finally, note that although the active and passive parts move alternately in the path obtained by the ML-RRT algorithm, a randomized path smoothing post-processing[4] is performed in the composite configuration space of all the parameters, so that simultaneous motions are obtained in the final path.

VI. Empirical Performance Analysis

The basic RRT algorithm and the ML-RRT variant have been implemented into the path planning software *Move3D* [22]. An empirical performance analysis has been carried out applying both algorithms to several two-dimensional and three-dimensional academic examples. The first example, *2D-simple*, corresponds to the problem illustrated in Fig. 1. The second example, *2D-hard*, is a more difficult version involving a longer static object with six mobile sticks (Fig. 3). The other two examples, *3D-simple* and *3D-hard*, correspond to similar variants of the three-dimensional problem illustrated in Fig. 5. In all cases, the active parameters for the ML-RRT algorithm are those defining the location of the mobile object, while the parameters corresponding to all the articulated parts are passive. Tests have been performed on an Apple iBook with a 1.2 GHz PowerPC G4 processor. Numerical results have been averaged over 10 runs.

Table I displays the computing time (with standard deviation), and the number of nodes, samples and collision tests required for solving the four problems with RRT and ML-RRT. Note that, for ML-RRT, N_{samp}^{act} represents the number of samples for the active parameters. These results show that ML-RRT clearly outperforms the basic RRT, and that the performance gain increases with the complexity of the articulated objects. Note that the basic RRT is unable to solve the difficult versions of the problems in reasonable computing time, while the performance of ML-RRT is only slightly affected by the problem difficulty. Fig. 3 shows a projection of the search trees on the coordinates of the center of mass of the mobile object for example *2D-hard*. The tree computed with the basic RRT algorithm contains 10,000 nodes but all are concentrated in a small region of the search-space around the initial configuration. The tree obtained with the ML-RRT algorithm contains less than 1,000 nodes and yet better covers the search-space.

[4]We use the *probabilistic path shortening* method [21] for path smoothing.

TABLE I
Numerical results

		2D-simple	*2D-hard*	*3D-simple*	*3D-hard*
	N_{DOF}	7	10	9	11
RRT	T(sec.)	752 ± 526	$\rightarrow\infty$	10 ± 9	$\rightarrow\infty$
	N_{node}	5047	$\rightarrow\infty$	1102	$\rightarrow\infty$
	N_{samp}	32348	$\rightarrow\infty$	1207	$\rightarrow\infty$
	N_{coll}	44742	$\rightarrow\infty$	5054	$\rightarrow\infty$
ML-RRT	T(sec.)	8 ± 4	14 ± 6	3 ± 2	11 ± 8
	N_{node}	856	1189	757	1142
	N_{samp}^{act}	1698	2226	412	1194
	N_{coll}	5458	7650	5061	11353

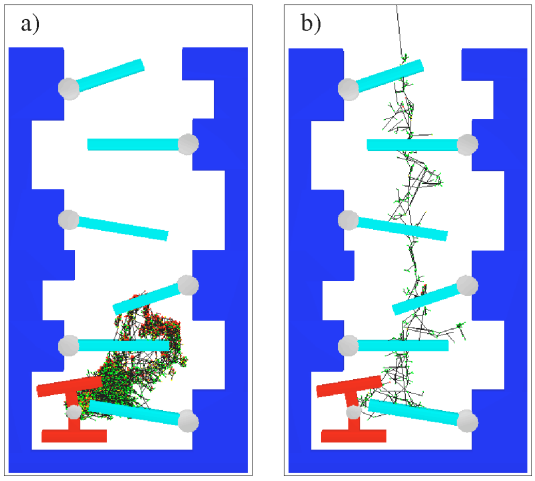

Fig. 3. Projection of search trees for problem *2D-hard* obtained with the basic RRT algorithm (a) and with the ML-RRT algorithm (b).

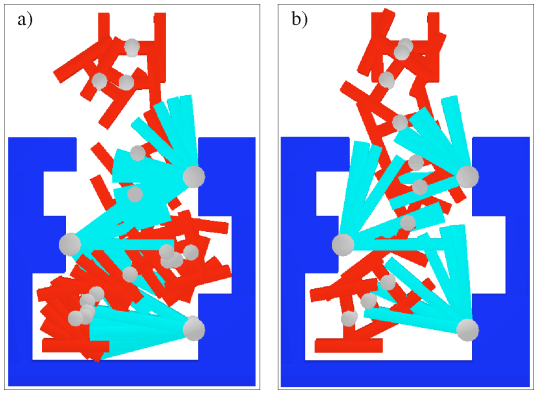

Fig. 4. Trace of solution paths for problem *2D-simple* obtained with the basic RRT algorithm (a) and with the ML-RRT algorithm (b).

Besides the computational efficiency, the solution paths obtained with the ML-RRT algorithm are also qualitatively different to those of the basic RRT. Fig. 4 shows the trace of a solution path obtained with each algorithm. The difficulty of the basic RRT in finding the solution is reflected by an ugly path (Fig. 4.a) with many small motions needed to circumvent the mobile parts of the fixed object. ML-RRT produces a much more natural-looking path in which the mobile object progresses toward its goal while "pushing" the passive parts.

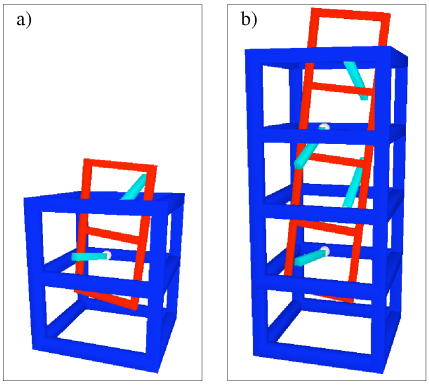

Fig. 5. Two variants of a three-dimensional academic example: a) *3D-simple*, b) *3D-hard*.

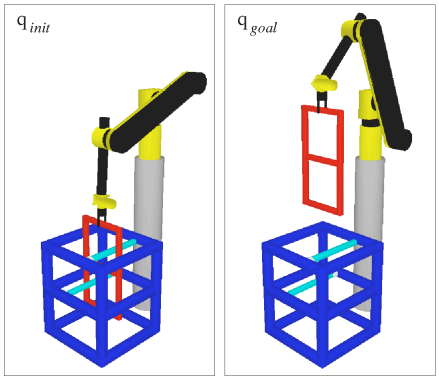

Fig. 6. The two objects of the example *3D-simple* are disassembled by a robotic manipulator.

VII. Practical Applications

A. Robotic Manipulation

The proposed algorithm can be extended to integrate a robot that manipulates the mobile object. The system composed by the robot grasping the mobile object at a given pose can be seen as a closed-chain mechanism, and possible motions take place in the self-motion manifold of this mechanism.

For non-redundant manipulators, only a finite number of configurations will match a given object pose and grasp position. These configurations can be directly obtained by inverse kinematics (IK). For ensuring the path continuity and avoiding singularities, only the IK solution corresponding to the same configuration type (e.g. elbow-up or elbow-down) as the expanded configuration should be considered.

The extension is more difficult for redundant manipulators, since the robot can grasp the object with an infinite number of configurations. The general approach for closed-chain path planning described in [23] can be applied to explore the self-motion manifold for these more difficult cases. This approach solves the configuration sampling problem using the *Random Loop Generator* (*RLG*) algorithm. The kinematic loop is broken into two open sub-chains, called the active and passive sub-chains. The passive sub-chain is a non-redundant mechanism, with a finite number of possible configurations corresponding to a given configuration of the active sub-chain. RLG combines random sampling techniques with simple geometrical operations aiming to generate configurations of the active sub-chain into the reachable workspace of the passive sub-chain, whose configuration is then obtained by IK.

Fig. 6 shows an example in which a non-redundant manipulator arm extracts an object from a box containing articulated parts. This problem has been solved with ML-RRT (extended to closed chains) in only 4 seconds.

B. Structural Bioinformatics

The computational analysis of molecular interactions in biological systems is a key instrument for the understanding of life. In this framework, we address protein-ligand interactions [24]. Fig. 7 shows a protein model with a ligand located in its active site. Most of the computational approaches to this problem address a static view of the molecular recognition. However, several studies tend to show that the ligand access/exit to the protein active site can be very important for the understanding of the biological mechanism [25]. The difficulty is that computing the pathway of a ligand to go out from a deep active site to the surface of a protein (or vice versa) with "classic" molecular modeling methods [26] is too computationally expensive. For facing the complexity of computing molecular motions, molecules can be modeled as articulated mechanisms [27] and efficient path planning algorithms can be used to explore their conformational changes [28]. Considering such a mechanistic representation of molecules, the protein-ligand exit problem can be formulated as a mechanical disassembly problem for articulated objects and the ML-RRT algorithm can be applied for finding solution pathways. The difficulty comes from the complexity of the molecular model that contains hundreds of flexible side-chains possibly involved in the disassembly. Thus, if no prior knowledge on the ligand passageway is available, hundreds of degrees of freedom (DOF) have to be considered. The ML-RRT algorithm performs well when applied to this kind of difficult problems [20]. Problems involving hundreds of potential DOF are solved in very short computing time. For example, the protein in Fig. 7 contains 628 amino-acid residues. If all the protein side-chains are considered to be flexible, the model contains 1237 DOF. The ligand exit pathway in this example (computed in 160 seconds) is very constrained and requires an important motion of some side-chains. Among all the side-chains, ML-RRT determines that the motion of only 9 of them is required for "disassembling" the ligand, as illustrated in Fig. 8. Note the significant motion of the side-chain located at the middle-top of the image. This side-chain motion, which is known to be biologically important for the protein-ligand interaction, was automatically identified by the algorithm.

VIII. Conclusions and Future Work

The ML-RRT algorithm described in this paper is an efficient method for disassembly path planning of two objects with articulated parts. An interesting feature of the algorithm is its ability to treat problems with a high number of potentially mobile parts and to automatically identify the degrees of freedom that are important for the disassembly task. This feature has already been exploited in structural bioinformatics applications, and we think that it will be also very useful in

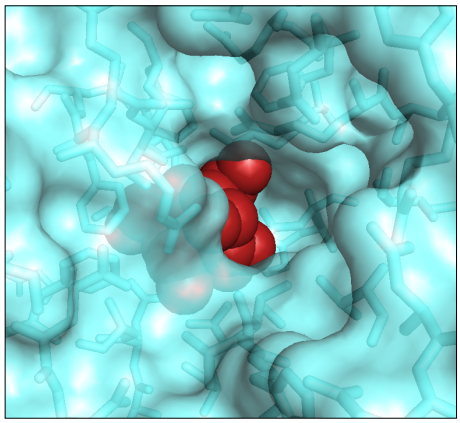

Fig. 7. Ligand in the active site of a protein. Both molecules can be modeled as articulated mechanisms.

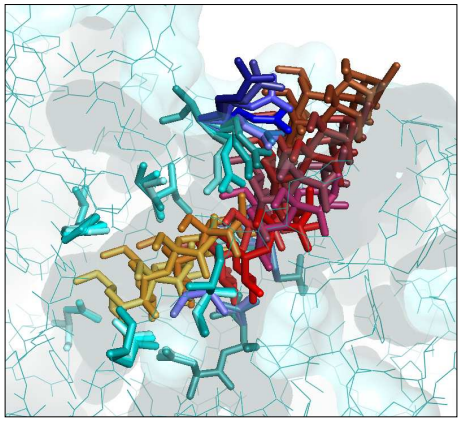

Fig. 8. Solution path for the molecular disassembly problem illustrated in Fig. 7. The image shows a transversal cut of the protein active site and the trace of the ligand path. The ligand and the 9 residues with moving side-chains are displayed in stick representation. The configurations of the ligand and the moving side-chains at different steps along the path are colored in red scale and blue scale respectively.

CAD/PLM problems.

The current version of ML-RRT is devised for solving problems in which passive articulated parts are "pushed" by the mobile object. A future extension of the algorithm will also consider "pulling" motions, which may be important in some classes of disassembly problems.

Another envisaged extension is to address disassembly planning problems for multiple (possibly articulated) objects. Disassembly sequences could be computed using an active/passive decomposition of the configuration parameters and applying the mechanism for motion propagation implemented in the ML-RRT algorithm. The active/passive roles could be assigned based on a (random) selection of objects being moved with priority. Sampling-based path planning algorithms have already been proposed for disassembly sequencing [11]. The main advantage of ML-RRT over other existing methods is a reduced computational cost thanks to the decoupled exploration of configuration space sub-manifolds associated with the active/passive parameter subsets.

ACKNOWLEDGMENT

This work has been partially supported by the EC under Contract IST 045359 "PHRIENDS", the Région Midi-Pyrénées under projets ITAV "ALMA" and CTP "AMOBIO", and the ANR project "NanoBioMod".

REFERENCES

[1] J. Cortés and T. Siméon, "Disassembly path planning for objects with articulated parts," *Proc. IFAC Workshop on Intelligent Assembly and Disassembly*, pp. 34–39, 2007.
[2] J.-C. Latombe, *Robot Motion Planning*. Boston: Kluwer Academic Publishers, 1991.
[3] S. M. LaValle, *Planning Algorithms*. New York: Cambridge University Press, 2006.
[4] ——, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Computer Science Dept., Iowa State University, 1998.
[5] E. Ferré and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3149–3154, 2004.
[6] A. Bourjault, "Contribution à une approche méthodologique de l'assemblage automatisé : Elaboration automatique des séquences opératoires," Thèse d'Etat, Université de Franche-Comté, 1984.
[7] L. S. Homem de Mello and S. Lee, *Computer-Aided Mechanical Assembly Planning*. Boston: Kluwer Academic Publishers, 1991.
[8] R. H. Wilson, "On geometric assembly planning," Ph.D. dissertation, Stanford University, 1992.
[9] T. Lozano-Pérez and R. H. Wilson, "Assembly sequencing for arbitrary motions," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 527–532, 1993.
[10] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, pp. 577–601, 2000.
[11] S. Sundaram, I. Remmler, and N. M. Amato, "Disassembly sequencing using a motion planning approach," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 1475–1480, 2001.
[12] H. Chang and T.-Y. Li, "Assembly maintainability study with motion planning," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 1012–1019, 1995.
[13] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. 32, pp. 108–120, 1983.
[14] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees : Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. Donald, K. Lynch, and D. Rus, Eds. Boston: A.K. Peters, 2001, pp. 293–308.
[15] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-Domain RRTs: Efficient exploration by controlling the sampling domain," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3867–3872, 2005.
[16] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3307–3312, 2007.
[17] J. J. Kuffner, "Effective sampling and distance metrics for 3D rigid body path planning," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3993–3998, 2004.
[18] P. Cheng and S. M. LaValle, "Reducing metric sensitivity in randomized trajectory design," *Proc. IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, pp. 43–48, 2001.
[19] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," *Proc. IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, pp. 1178–1183, 2003.
[20] J. Cortés, L. Jaillet, and T. Siméon, "Molecular disassembly with RRT-like algorithms," *Proc. IEEE Int. Conf. Robot. Automat.*, pp. 3301–3306, 2007.
[21] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars, "Multi-level path planning for nonholonomic robots using semi-holonomic subsystems," *Int. J. Robot. Res.*, vol. 17(8), pp. 840–857, 1998.
[22] T. Siméon, J.-P. Laumond, and F. Lamiraux, "Move3D: A generic platform for path planning," *Proc. IEEE Int. Symp. on Assembly & Task Planning*, pp. 25–30, 2001.
[23] J. Cortés and T. Siméon, "Sampling-based motion planning under kinematic loop-closure constraints," in *Algorithmic Foundations of Robotics VI*, M. Erdmann, D. Hsu, M. Overmars, and F. van der Stappen, Eds. Berlin: Springer-Verlag, 2005, pp. 75–90.
[24] H.-J. Böhm and G. Schneider, *Protein-Ligand Interactions: From Molecular Recognition to Drug Design*. Weinheim: Wiley-VCH, 2003.
[25] S. K. Lüdemann, V. Lounnas, and R. C. Wade, "How do substrates enter and products exit the buried active site of cytochrome p450cam? 1. random expulsion molecular dynamics investigation of ligand access channels and mechanisms," *J. Mol. Biol.*, vol. 303(5), pp. 797–811, 2000.
[26] T. Schlick, *Molecular Modeling and Simulation - An Interdisciplinary Guide*. New York: Springer, 2002.
[27] D. Parsons and J. Canny, "Geometric problems in molecular biology and robotics," *Proc. Int. Conf. Intel. Sys. Mol. Biol.*, pp. 322–330, 1994.
[28] J. Cortés, T. Siméon, V. Ruiz, D. Guieysse, M. Remaud, and V. Tran, "A path planning approach for computing large-amplitude motions of flexible molecules," *Bioinformatics*, vol. 21, pp. 116–125, 2005.