DISASSEMBLY PATH PLANNING FOR OBJECTS WITH ARTICULATED PARTS

Juan Cortés, Thierry Siméon

LAAS-CNRS
7 Av. du Colonel-Roche, 31077 Toulouse, France

Abstract: Sampling-based path planning algorithms are powerful tools for computing 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.

Keywords: Path planning, Algorithms, Manipulation tasks,
Robotic manipulators, Assembly robots.

1. INTRODUCTION

This paper addresses the problem of automatically computing motions to disassemble objects. The problem can be formulated as a general path planning problem (Latombe, 1991; LaValle, 2006) (see Section 2). Indeed, path planning concepts and algorithms have been applied to solve different instances of the (dis)assembly planning problem (Lozano-Pérez and Wilson, 1993; Chang and Li, 1995; Halperin *et al.*, 2000; Sundaram *et al.*, 2001; Ferré and Laumond, 2004). The instance treated in this paper considers two objects, with the particularity that both objects may have multiple articulated parts. Figure 1 illustrates a simple two-dimensional example.

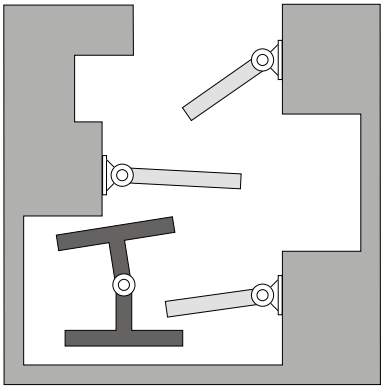

Fig. 1. Disassembly path planning problem for two objects with articulated parts. The problem consists in finding a path to extract the small (dark) object from the big one.

The algorithm presented in this paper is a variant of a sampling-based path planning method: the RRT algorithm introduced by LaValle (1998). Section 3 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 Manhat-

tan paths over these two sets of parameters that change alternatively. The ML-RRT algorithm is explained in Section 4. 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. First, the computing time is notably reduced (see results in Section 5.1). And second, but not less important, 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 5.2: the simulation of molecular interactions. Besides this application in structural bioinformatics, the ML-RRT algorithm is applicable to more classic problems involving part disassembly, such as Product Lifecycle Management (PLM) problems (Ferré and Laumond, 2004). 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 5.2). Other possible future extensions of the algorithm are outlined in Section 6.

2. 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* $\boldsymbol{C}$ (Lozano-Pérez, 1983; Latombe, 1991; LaValle, 2006). 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 $\mathbf{q}_{init}$ and a goal configuration $\mathbf{q}_{goal}$ (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: $\mathbf{q} = \{\mathbf{q}^M, \mathbf{q}^{Jm}, \mathbf{q}^{Js}\}$, where $\mathbf{q}^M$ contains parameters defining the position and the orientation of the mobile reference frame, and $\mathbf{q}^{Jm}$ and $\mathbf{q}^{Js}$ 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, $\mathbf{q}^M$. 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: the active parameters $\mathbf{q}^{act} = \mathbf{q}^M$ and the passive parameters $\mathbf{q}^{pas} = \{\mathbf{q}^{Jm}, \mathbf{q}^{Js}\}$. This partition induces the corresponding sub-manifolds in the configuration-space: $\boldsymbol{C} = \boldsymbol{C}^{act} \times \boldsymbol{C}^{pas}$. The terms active and passive have been chosen in relation to how the algorithm described in Section 4 acts on them.

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

3. THE BASIC RRT ALGORITHM

The basic principle of the RRT algorithm (LaValle, 1998) is to incrementally grow a random tree τ rooted at the initial configuration $\mathbf{q}_{init}$ to explore the reachable configuration-space and find a feasible path connecting $\mathbf{q}_{init}$ to a goal configuration $\mathbf{q}_{goal}$. Figure 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 $\mathbf{q}_{rand} \in \boldsymbol{C}$. 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 $\mathbf{q}_{near}$ in the tree to the sample $\mathbf{q}_{rand}$ is selected and an attempt is made to expand $\mathbf{q}_{near}$ in the direction of $\mathbf{q}_{rand}$. For holonomic systems, the expansion procedure can be simply performed by moving on the straight-line segment between $\mathbf{q}_{near}$ and $\mathbf{q}_{rand}$. If the expansion succeeds, a new node $\mathbf{q}_{new}$ and a feasible local path from $\mathbf{q}_{near}$ 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.

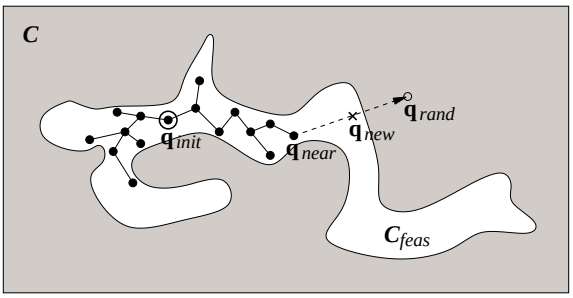

Fig. 2. Illustration of one expansion step of a search tree using an RRT-based algorithm. The tree tends to cover $\boldsymbol{C}_{feas}$: the feasible subset of the search-space $\boldsymbol{C}$.

Algorithm 1: Construct_RRT

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

Different strategies can be adopted for the design of path planners based on the RRT algorithm (LaValle and Kuffner, 2001). The configuration sampling achieved by the function `SampleConf` is normally made using a uniform random distribution in the configuration-space $\boldsymbol{C}$. However, Yershova *et al.* (2005) have shown that sampling in a dynamic domain that envelopes the search tree may improve the performance of the RRT algorithm. Another technical point concerns the function `BestNeighbor`. The basic RRT algorithm selects $\mathbf{q}_{near}$ as the nearest node to $\mathbf{q}_{rand}$ using an Euclidean metric in $\boldsymbol{C}$. Such a metric distance is very simple and easy to compute. However, since it does not consider motion constraints (kinematic constraints, obstacles, ...), it may lead to an undesired behavior of the planner, by repeatedly selecting "blocked" nodes for futile expansion. To avoid this problem, two modifications have been introduced in `BestNeighbor`: (1) A node is no longer selected after its expansion fails a given number of consecutive times l. (2) $\mathbf{q}_{near}$ is selected at random among the k nearest neighbors. The efficiency of these two modifications has been shown in related works (Cheng and LaValle, 2001), (Urmson and Simmons, 2003), (Cortés *et al.*, 2007). In our implementation, l is a constant with default value equal to 10, and k is computed as the nearest integer greater than or equal to $n_{nodes}/100$, where n_{nodes} is the current number of nodes in the tree. 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. The implementation used in this work applies a more greedy variant, 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 (LaValle and Kuffner, 2001), which are the type of systems considered in this paper.

4. 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 2. 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 motion).

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 only samples the associated parameters $\mathbf{q}^{act}$. Thus, this function generates a configuration $\mathbf{q}^{act}_{rand}$ in a sub-manifold of the configuration-space involving the active parameters, $\boldsymbol{C}^{act}$. The function `BestNeighbor` selects the node to be expanded $\mathbf{q}_{near}$ using a distance metric in $\boldsymbol{C}^{act}$. Note that the function `BestNeighbor` also integrates the basic improvements mentioned in Section 3. Then, `Expand` performs the expansion of the selected configuration by only changing the active parameters. A greedy strategy is used. The returned configuration $\mathbf{q}_{new}$ corresponds to the last valid point in the straight-line segment from $\mathbf{q}_{near}$ toward $\{\mathbf{q}^{act}_{rand}, \mathbf{q}^{pas}_{near}\}$. 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^{col}_{pas} 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 $\mathbf{q}^{pas}_{rand}$ by randomly sampling the value of the passive parameters associated with L^{col}_{pas} in a ball around their configuration in $\mathbf{q}_{near}$.

Algorithm 2: Construct_ML-RRT

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

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 generate a new node by expanding $\mathbf{q}_{near}$ toward $\{\mathbf{q}_{near}^{act}, \mathbf{q}_{rand}^{pas}\}$. Only the parts in L_{pas}^{col} move during this tree expansion. Like for the active parameters, a list $L_{pas}^{col'}$ is returned by the function `Expand` when 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.

5. RESULTS

This section presents two types of results. First, an empirical performance analysis is presented based on several academic examples. Then, two practical applications are illustrated: robotic manipulation and the simulation of molecular interactions.

5.1 Empirical performance analysis

The basic RRT algorithm and the ML-RRT variant have been implemented into the motion planning software *Move3D* (Siméon *et al.*, 2001). An empirical performance analysis has been carried out applying both algorithms to several two-dimensional and three-dimensional academic examples. The first example, stb-2D-S, corresponds to the problem illustrated in Figure 1. The second example, stb-2D-L, is a more difficult version with a longer static object containing six mobile sticks (see Figure 3). The other two examples, stb-3D-S and stb-3D-L, correspond to the two versions of a three-dimensional disassembly problem illustrated in Figure 5. All tests have been performed on an Apple iBook with a 1.2 GHz PowerPC G4 processor.

Table 1. RRT results.

	stb-2D-S	stb-2D-L	stb-3D-S	stb-3D-L
T(sec.)	751.7	$\rightarrow \infty$	10.1	$\rightarrow \infty$
N_{nodes}	5047	$\rightarrow \infty$	1102	$\rightarrow \infty$

Table 2. ML-RRT results.

	stb-2D-S	stb-2D-L	stb-3D-S	stb-3D-L
T(sec.)	8.0	13.5	2.7	11.0
N_{nodes}	856	1189	757	1142

Table 1 and Table 2 display the computing time and the number of nodes in the search trees generated for solving the four problems with RRT and ML-RRT respectively. Numerical results have been averaged over 10 runs. 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. Figure 3 shows a projection of the search trees on the coordinates of the center of mass of the mobile object for example stb-2D-L. The tree computed with the basic RRT algorithm contains 10000 nodes but all are concentrated in a small region of

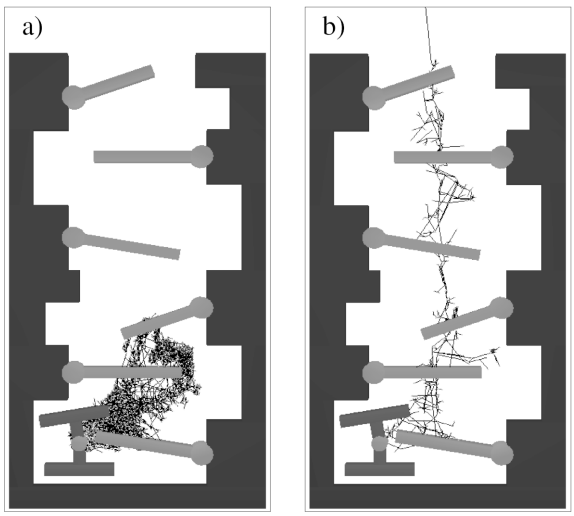

Fig. 3. Projection of search trees for problem stb-2D-L obtained with the basic RRT algorithm (a) and with the ML-RRT algorithm (b).

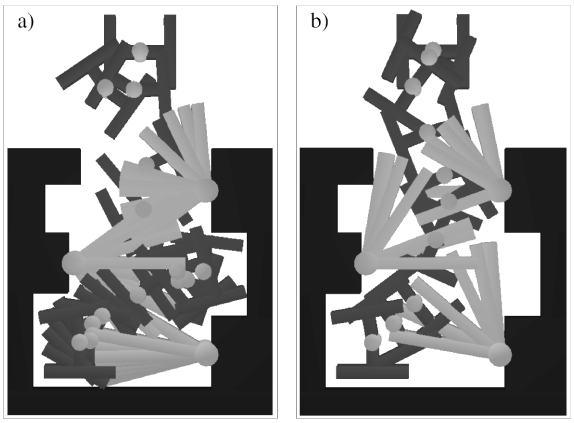

Fig. 4. Trace of solution paths for problem stb-2D-S obtained with the basic RRT algorithm (a) and with the ML-RRT algorithm (b).

the search-space around the initial configuration. The tree obtained with the ML-RRT algorithm contains less than 1000 nodes and yet better covers the search-space.

Besides the computational efficiency, the solution paths obtained with both algorithms are also qualitatively different. Figure 4 shows the trace of one of the paths obtained with the basic RRT and another obtained with ML-RRT. In the path displayed by Figure 4.a, obtained with the basic RRT, the mobile object circumvents the mobile parts of the fixed object. In the solution path obtained with ML-RRT, the mobile object "pushes" the passive parts. This type of solution path seems more natural for this kind of problem. 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 is performed in the composite configuration-space of all the parameters, so that simultaneous motions are obtained in the final path.

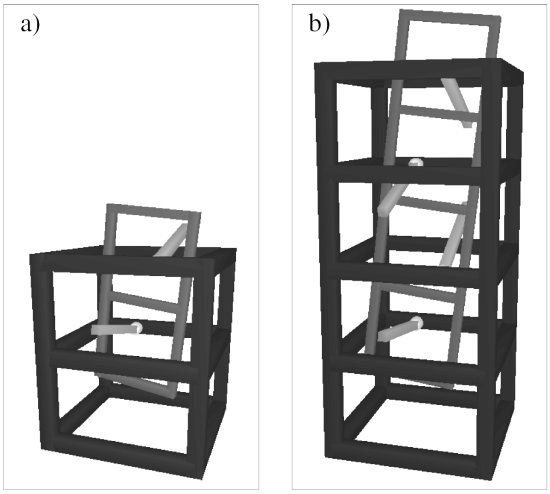

Fig. 5. Two variants of a three-dimensional academic example: a) stb-3D-S, b) stb-3D-L.

5.2 Practical applications

Robotic manipulation. The proposed algorithm can be easily extended to integrate a robot that manipulates the mobile object. For non-redundant manipulators, all the degrees of freedom are considered as passive parameters for the planner and their values are directly computed from the active parameters that define the location of the object using inverse kinematics. For redundant manipulators, the general approach for closed-chain path planning described by Cortés and Siméon (2005) can be used. Figure 6 shows an example in which a robotic manipulator 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.

Structural bioinformatics. A mechanistic representation of molecules permits to apply path planning algorithms for studying their interactions (Cortés *et al.*, 2005). An important problem in structural bioinformatics is to compute access (or exit) pathways of a ligand to the active site of a protein. Figure 7 illustrates this problem. Both, the protein and the ligand, can be modeled as articulated mechanisms, and then the problem can be formulated as a mechanical disassembly problem for two articulated objects. The difficulty is that the motion of many atoms may be involved in the disassembly. Indeed, if an a priori knowledge about the ligand passageway is not available, hundreds of potential degrees of freedom have to be considered. The ML-RRT algorithm performs very well when applied to this kind of difficult problems (Cortés *et al.*, 2007). Problems involving more than 300 potential degrees of freedom are solved in only a few seconds.

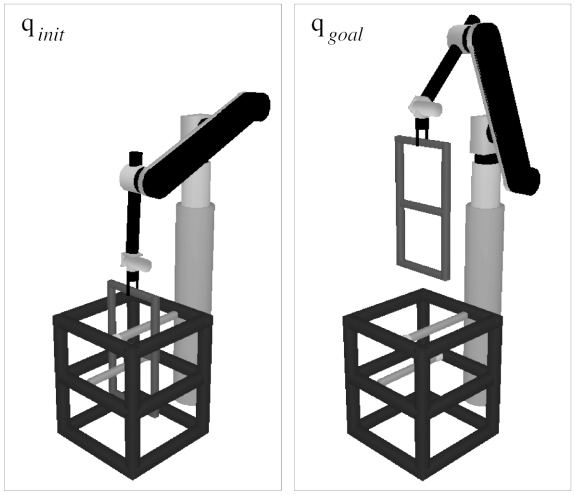

Fig. 6. The two objects of the example stb-3D-S are disassembled by a robotic manipulator.

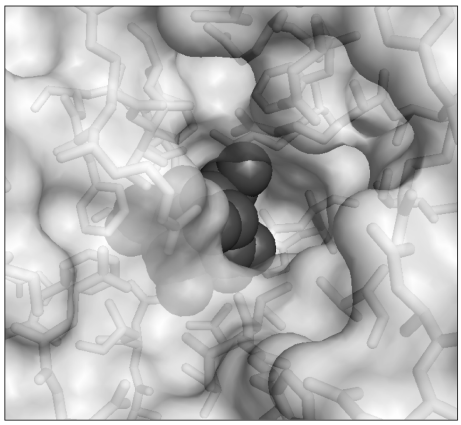

Fig. 7. Ligand in the active site of a protein. Both molecules can be modeled as articulated mechanisms.

6. 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 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 (Sundaram *et al.*, 2001). The main advantage of ML-RRT over other existing methods is a reduced computational cost thanks to the decoupled exploration of sub-manifolds in the configuration-space that correspond to different subsets of parameters.

REFERENCES

Chang, H. and T.-Y. Li (1995). Assembly maintainability study with motion planning. *Proc. IEEE Int. Conf. Robot. Automat.* pp. 1012–1019.

Cheng, P. and S.M. LaValle (2001). Reducing metric sensitivity in randomized trajectory design. *Proc. IEEE/RSJ Int. Conf. Intel. Rob. Sys.* pp. 43–48.

Cortés, J. and T. Siméon (2005). Sampling-based motion planning under kinematic loop-closure constraints. In: *Algorithmic Foundations of Roadmaps VI* (M. Erdmann, D. Hsu, M. Overmars and F. van der Stappen, Eds.). pp. 75–90. Springer-Verlag. Berlin.

Cortés, J., L. Jaillet and T. Siméon (2007). Molecular disassembly with RRT-like algorithms. *Proc. IEEE Int. Conf. Robot. Automat.* In press.

Cortés, J., T. Siméon, V. Ruiz-deAngulo, D. Guieysse, M. Remaud-Siméon and V. Tran (2005). A path planning approach for computing large-amplitude motions of flexible molecules. *Bioinformatics* **21**, 116–125.

Ferré, E. and J.-P. Laumond (2004). An iterative diffusion algorithm for part disassembly. *Proc. IEEE Int. Conf. Robot. Automat.* pp. 3149–3154.

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

Latombe, J.-C. (1991). *Robot Motion Planning.* Kluwer Academic Publishers. Boston.

LaValle, S.M. (1998). Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University.

LaValle, S.M. (2006). *Planning Algorithms.* Cambridge University Press. New York.

LaValle, S.M. and J.J. Kuffner (2001). Rapidly-exploring random trees : Progress and prospects. In: *Algorithmic and Computational Robotics: New Directions* (B.R. Donald, K.M. Lynch and D. Rus, Eds.). pp. 293–308. A.K. Peters. Boston.

Lozano-Pérez, T. (1983). Spatial planning: A configuration space approach. *IEEE Trans. Comput.* **32**, 108–120.

Lozano-Pérez, T. and R.H. Wilson (1993). Assembly sequencing for arbitrary motions. *Proc. IEEE Int. Conf. Robot. Automat.* pp. 527–532.

Siméon, T., J.-P. Laumond and F. Lamiraux (2001). Move3D: A generic platform for path planning. *Proc. IEEE Int. Symp. on Assembly & Task Planning* pp. 25–30.

Sundaram, S., I. Remmler and N.M. Amato (2001). Disassembly sequencing using a motion planning approach. *Proc. IEEE Int. Conf. Robot. Automat.* pp. 1475–1480.

Urmson, C. and R. Simmons (2003). Approaches for heuristically biasing RRT growth. *Proc. IEEE/RSJ Int. Conf. Intel. Rob. Sys.* pp. 1178–1183.

Yershova, A., L. Jaillet, T. Siméon and S.M. LaValle (2005). Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. *Proc. IEEE Int. Conf. Robot. Automat.* pp. 3867–3872.