each represents the command of one user grasping one object. These records along with other sensory data, like vision and tactile data, can then be used as input data to a learning system that can learn how to grasp objects from actual grasping experiments. Moussa [13], [14] tested such a grasping system in a simulated environment using an object set that included 28 everyday objects with very promising results. But, this work did not include any user-specific grasping configurations or preferences.

Our next step is to enhance this natural language interface by adding a learning component. Using the data collected, we plan to explore finding patterns of commands that could be used to predict the user’s next command and the type of object grasped, thus leading to faster interaction between the user and the robot. Our view is that this could eventually lead to meta-grasping commands like “grasp this object like you grasp a comb.”

ACKNOWLEDGMENT

The authors would like to thank the participants who volunteered for the study.

REFERENCES


Disassembly Path Planning for Complex Articulated Objects

Juan Cortés, Léonard Jaillet, and Thierry Siméon

Abstract—Sampling-based path planning algorithms are powerful tools for computing constrained disassembly motions. This paper presents a variant of the Rapidly-exploring Random Tree (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 as compared to standard path planning techniques. The paper also shows practical applications of the presented algorithm in robotics and structural bioinformatics.

Index Terms—Articulated mechanisms, disassembly paths, molecular interactions, path planning algorithms, robotic manipulation.

I. INTRODUCTION

This paper1 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

1. A preliminary version of this paper is appeared in [1].
the particularity that both objects may have multiple articulated parts. Fig. 1 illustrates a simple 2-D example.

The algorithm presented in this paper is a variant of a sampling-based path planning method: the Rapidly-exploring Random Tree (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 few years (see http://mal.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 process. 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) and 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 (DoFs), 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.

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 nondirectional 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 mentioned before 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 $C$ [2], [3], [13]. A configuration $q$ is a minimal set of parameters defining the location of the mobile system in the world, and $C$ is the set of all the configurations. Given the assembled configuration $q_{\text{init}}$ and a goal configuration $q_{\text{goal}}$ (any disassembled configuration), the problem consists in finding a feasible collision-free path in $C$ that connects both the 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 DoFs associated with the articulated parts in both objects. Thus, the configuration vector is given by: $q = \{q^{\text{init}}, q^{\text{move}}, q^{\text{joint}}\}$, where $q^{\text{init}}$ contains parameters defining the position and the orientation of the mobile reference frame, and $q^{\text{move}}$ and $q^{\text{joint}}$ 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 $q^{\text{move}}$. 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: \( q = \{ q^{\text{act}}, q^{\text{pas}} \} \), with \( q^{\text{act}} = \mathbb{Q}^l \), representing the active parameters and \( q^{\text{pas}} = \{ q^{\text{tr}}, q^{\text{R}} \} \), representing 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 submanifolds in the configuration space: \( C = C^{\text{act}} \times C^{\text{pas}} \).

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

### IV. Basic RRT Algorithm

The basic principle of the RRT algorithm [4] is to incrementally grow a random tree \( \tau \) rooted at the initial configuration \( q^{\text{init}} \) in order to explore the reachable configuration space and to find a feasible path connecting \( q^{\text{init}} \) to a goal configuration \( q^{\text{goal}} \). Fig. 2 illustrates the process, and Algorithm 1 gives the pseudocode for the RRT construction. At each iteration, the tree is expanded toward a randomly sampled configuration \( q^{\text{rand}} \in C \). This random sample is used simultaneously to 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 \( q^{\text{near}} \) in the tree to the sample \( q^{\text{rand}} \) is selected, and an attempt is made to expand \( q^{\text{near}} \) in the direction of \( q^{\text{near}} \). For kinematically unconstrained systems, the expansion procedure can be simply performed by moving on the straight line segment between \( q^{\text{near}} \) and \( q^{\text{rand}} \). If the expansion succeeds, a new node \( q^{\text{new}} \) and a feasible local path from \( q^{\text{near}} \) 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, the 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 \( \text{SampleConf} \)) is normally made using a uniform random distribution in the configuration space \( C \). However, more sophisticated sampling strategies (e.g., [15] and [16]) may improve the performance of the RRT algorithm. Another technical point concerns the function \( \text{BestNeighbor} \). The basic RRT algorithm selects \( q^{\text{near}} \) as the nearest node to \( q^{\text{rand}} \) using an Euclidean metric\(^2\) in \( C \). 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 \( \text{BestNeighbor} \): 1) a node is no longer selected after its expansion fails a given number of consecutive times \( l \) and 2) \( q^{\text{near}} \) is selected at random among the \( k \) nearest neighbors.\(^3\) The efficiency of these two modifications has been shown in related works [18]–[20]. One can also choose a more or less greedy strategy for the expansion procedure (function \( \text{Expand} \) 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.

### V. 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, ML-RRT, computes the motion of the parts associated with the 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 \( \text{SampleConf} \) receives as argument the list of active parts \( L^{\text{act}} \), and it only samples the associated parameters \( q^{\text{act}} \). Thus, this function generates a configuration \( q^{\text{rand}} \) in a submanifold of the configuration space involving the active parameters \( C^{\text{act}} \). The function \( \text{BestNeighbor} \) selects the node to be expanded \( q^{\text{near}} \) using a distance metric in \( C^{\text{act}} \). Note that the function \( \text{BestNeighbor} \) also integrates the basic improvements mentioned in Section IV. Then, \( \text{Expand} \) performs the expansion of the selected configuration by only changing the active

\(^2\)We use a weighted metric for translation and rotation components, with 3-D 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 \( n_{\text{node}} \) is the current number of nodes in the tree. These values have been empirically determined.
parameters. A greedy strategy is used. The returned configuration \( q_{\text{new}} \) corresponds to the last valid point computed along the straight line segment from \( q_{\text{near}} \) toward \{\( q_{\text{rand, act}} \), \( q_{\text{pas, near}} \}\). If the expansion is not negligible, a new node and a new edge are added to the tree. The function \( \text{Expand} \) also analyzes the collision pairs yielding the stop of the expansion process. If active parts in \( L_{\text{act}} \) collide with potentially mobile passive parts in \( L_{\text{pas}} \), the list of the involved passive parts \( L_{\text{col, pas}} \) is returned. This information is used in the second stage of the algorithm, which generates the motion of passive parts. The function \( \text{PerturbConf} \) generates a configuration \( q_{\text{pas, rand}} \) by randomly sampling the value of the passive parameters associated with \( L_{\text{col, pas}} \) in a ball around their configuration in \( q_{\text{near}} \). Note that, if the previous call to \( \text{Expand} \) has been successful, \( q_{\text{near}} \) has been updated in order to contain the new configuration of the active parameters. An attempt is then made to further expand \( q_{\text{near}} \) toward \{\( q_{\text{act, near}} \), \( q_{\text{pas, rand}} \}\). Only the parts in \( L_{\text{pas}} \setminus L_{\text{col}} \) move during this tree expansion. The function \( \text{Expand} \) returns a list \( L_{\text{pas}} \) if the expansion is stopped by a collision involving passive parts. If this list contains new passive parts (in relation to \( L_{\text{pas}} \)), 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 postprocessing 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 \textit{Move3D} [22]. An empirical performance analysis has been carried out applying both algorithms to several 2-D and 3-D 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 (see Fig. 3). The other two examples, \( 3D\)-simple and \( 3D\)-hard, correspond to similar variants of the 3-D problem illustrated in Fig. 5. In all the 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 ten 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, \( N_{\text{act, samp}} \) 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 1000 nodes, and yet, better covers the search space.

![Fig. 3. Projection of search trees for problem 2D-hard obtained. (a) With the basic RRT algorithm. (b) With the ML-RRT algorithm.](image-url)

<table>
<thead>
<tr>
<th>( N_{\text{node}} )</th>
<th>( N_{\text{act}} )</th>
<th>( N_{\text{sample}} )</th>
<th>( N_{\text{coll}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2D)-simple</td>
<td>7</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>( 2D)-hard</td>
<td>5047</td>
<td>1102</td>
<td>5054</td>
</tr>
<tr>
<td>( 3D)-simple</td>
<td>32348</td>
<td>1207</td>
<td>1194</td>
</tr>
<tr>
<td>( 3D)-hard</td>
<td>44742</td>
<td>5061</td>
<td>11353</td>
</tr>
</tbody>
</table>

**TABLE I**

**NUMERICAL RESULTS**

---

\( N_{\text{DOPF}} \) represents the number of DOPF configurations used in the planning process.
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 [see Fig. 4(a)] with many small motions needed to circumvent the mobile parts of the fixed object. The ML-RRT produces a much more natural-looking path in which the mobile object progresses toward its goal while “pushing” the passive parts.

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 nonredundant 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 subchains, called the active and passive sub-chains. The passive subchain is a nonredundant mechanism, with a finite number of possible configurations corresponding to a given configuration of the active subchain. RLG combines random sampling techniques with simple geometrical operations aiming to generate configurations of the active subchain into the reachable workspace of the passive subchain, whose configuration is then obtained by IK.

Fig. 6 shows an example in which a nonredundant 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 s.

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
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 submanifolds associated with the active/passive parameter subsets.

**REFERENCES**


Sensor Planning for Mobile Robot Localization—A Hierarchical Approach Using a Bayesian Network and a Particle Filter

Hongjun Zhou and Shigeyuki Sakane

Abstract—In this paper, we propose a hierarchical approach to solving sensor planning for the global localization of a mobile robot. Our system consists of two subsystems: a lower layer and a higher layer. The lower layer uses a particle filter to evaluate the posterior probability of the localization. The particles converge into clusters, which are constructed using particle clustering and sensor planning to generate an optimal sensing action sequence for the localization. The higher layer uses a Bayesian network for probabilistic inference. This model allows us to add multiple types of sensor data arbitrarily. Thus, our use of a Bayesian network is more general and extensible than that of the traditional POMDP style model.

Index Terms—Bayesian network, hierarchical approach, localization, particle filter, sensor planning.

I. INTRODUCTION

In recent years, the global localization of mobile robots has been an active area of research and various methods have been developed. However, most of the studies have taken a passive approach [2]–[9] in the sense that the robot’s actions are random and are not planned explicitly for efficient localization. In this paper, we propose a hierarchical approach to solving sensor planning for global localization.

In the reports of [2] and [3], the robot environment was represented by an occupancy grid map (cell size: 0.1 m × 0.1 m), and the robot pose is updated after the robot’s movement and sensor activation based on a hidden Markov model. A Monte Carlo version of global localization (MCL) has been reported in [4] and [9]. The particle filter resamples and updates the belief of localization, and estimates the maximal posterior probability density for the localization. The advantage of MCL is that arbitrary distribution can be represented by a set of samples, and the computational complexity of the Bayes filter [4] is reduced. As reported in [2]–[4] and [9], the robot pose was represented by a small occupancy grid and high angle resolution. In [10] and [11], the environment was represented by a topological map, with corridors, T-junctions, and doorways decomposed into larger cells (cell size: 1 m × 1 m), and with each cell corresponding to a node of the topological map. The robot updates its pose based on the cells using the partially observable Markov decision process (POMDP). Since these approaches assume that it is not necessary to know the robot’s pose in detail, the pose of the robot is represented based on the larger cell and four compass directions. In the aforementioned approaches, since the robot moved randomly without sensor planning, the systems did not always assure efficient convergence of the global localization if the environment had locally similar sensing patterns.

The theory of POMDP is an extension of the Markov decision process (MDP). Using the viewpoint of a Bayesian network, we can put a POMDP into the domain of a hidden Markov model (HMM) using action control. POMDPs are well understood and widely applied to mobile robot applications. The observation node of a POMDP corresponds to the sensor data or its features, and the state node corresponds to the robot pose. However, in practice, the robot sensor information is not always one type, and a real and complex environment may need multisensor data for representation. How to integrate such multisensor data will be a challenge for the traditional POMDP style model.

A Bayesian network is an extension of the HMM and Kalman filter. It allows one state node to have multiple observation nodes, and one observation node corresponds to multiple state nodes. In our system, we have employed a Bayesian network to represent and integrate the sensor information, the robot pose, and the sensing actions. This model allows us to add multiple types of sensor data arbitrarily. Thus, our use of a Bayesian network is more general and extensible than that of the POMDP style model.

Active global localization is a research topic extending from passive global localization and addresses sensing action selection for efficient global localization. In the past decade, some active localization approaches [13]–[23] have been proposed. Jensfelt and Kristensen [13] used multiple hypothesis tracking based on Kalman filtering for active global localization. The environment of their system was represented by a topological graph. The robot pose hypotheses were tracked based on Kalman filtering and the robot sensing commands were generated by POMDP style planning. Tomatis et al. [19] also proposed integrating a Kalman-filter-based metric map and a POMDP-based topological map for robot localization and navigation. These systems [13], [19] are similar to our hierarchical system. We employ a particle filter to track the robot accurately. Our particle filter can deal with linear/nonlinear dynamics with arbitrary noise. A basic Kalman filter is limited to a linear assumption. An extended Kalman filter (EKF) and unscented Kalman filter (UKF) can deal with nonlinear dynamics, but need to assume the existence of Gaussian noise. We employ a particle filter only for robot localization in the lower layer, and it is not the core algorithm of our sensor planning for localization.

Basye et al. [20] and Cassandra et al. [21] have proposed a POMDP model for robot navigation and localization. As described in [10], [11],