Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments

Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe

Abstract— **Static collision checking amounts to testing a given configuration of objects for overlaps. In contrast, the goal of dynamic checking is to determine whether all configurations along a continuous path are collision-free. While there exist effective methods for static collision detection, dynamic checking still lacks methods that are both reliable and efficient. A common approach is to sample paths at some fixed, prespecified resolution and statically test each sampled configuration. But this approach is not guaranteed to detect collision whenever one occurs, and trying to increase its reliability by refining the sampling resolution along the entire path results in slow checking. This paper introduces a new method for testing straight path segments in c-space or collections of such segments, that is both reliable and efficient. This method locally adjusts the sampling resolution by comparing lower bounds on distances between objects in relative motion with higher bounds on lengths of curves traced by points of these moving objects. Several additional techniques and heuristics increase the checker's efficiency in scenarios with many moving objects (e.g., articulated arms and/or multiple robots) and high geometric complexity. The new method is general, but particularly well suited for use in probabilistic roadmap (PRM) planners. Extensive tests, in particular on randomly generated path segments and on multi-segment paths produced by PRM planners, show that the new method compares favorably with a fixed-resolution approach at "suitable" resolution, with the enormous advantage that it never fails to detect collision.**

Keywords: Collision checking, motion planning, distance computation, probabilistic roadmaps, robotics

I. INTRODUCTION

A. Dynamic collision checking

COLLISION checking is a fundamental operation in robot motion planning, graphic animation, and physical simulation [1], [2], [3]. While *static* checking amounts to testing a single configuration of objects for overlaps, *dynamic* checking requires determining whether *all* configurations on a continuous path are collision-free.

Four major families of methods have been proposed for dynamic collision checking:

- *Feature-tracking* methods track pertinent features (vertices, edges, faces) of two objects – usually, the pair of closest features – to determine if the objects remain separated along a path. They rely on the following *coherence assumption*: the pertinent features change relatively rarely and, when they do change, the new ones can be computed efficiently from the old ones [4], [5], [6], [7], [8]. These methods also require each object to be made of few convex components and to have relatively small geometric complexity. These assumptions are poorly verified by kinematic chains (e.g., robot arms) in practical environments. Then, paths must be tested by tiny increments, to avoid missing collisions, especially collisions involving links at the end of the chains.
- *Bounding-volume hierarchy* (BVH) methods pre-compute, for each object (robot link, obstacle), a hierarchy of BVs (e.g., spheres, boxes) that approximates the geometry of the object at successive levels of detail [9], [10], [11], [12], [13], [14]. To check two objects for collision, their BVHs are searched from the top down, making it possible to quickly discard large subsets of the objects contained in disjoint BVs. Such methods have been applied to complex objects with surfaces described by several 100,000 triangles, and more [9], [12]. But they are fundamentally static methods. To test a path, the common approach is to check intermediate configurations spaced along the path at a prespecified resolution. If all these configurations are found collision-free, then the path is declared collision-free, but this answer may not be correct.
- *Swept-volume intersection* methods compute the volumes swept out by the objects and test these volumes for overlap [15], [16]. However, exact computation of swept volumes is expensive, especially when objects undergo rotations and have complex geometry. Moreover, the overlap test can no longer be speeded up by using pre-computed data structures, such as BVHs. Another difficulty is that swept volumes for pairs of moving objects may overlap even when the objects do not collide. Hence, when multiple objects move relative to each other, one must either consider the relative motions for all pairs, or compute and test volumes swept out in 4D space-time. Both ways yield costly computations.
- *Trajectory parameterization* methods express the geometry of the objects along the tested path by algebraic polynomials in a single variable t [17], [18]. These polynomials are then used to construct a collision condition on t. In principle, finding the values of t verifying this condition gives the exact intervals of collision along a given path. But, in general, the polynomials have

The authors are with the Robotics Laboratory, Department of Computer Science, Stanford University, CA 94305 USA. (e-mails: schwarzf@robotics.stanford.edu, mitul@robotics.stanford.edu, latombe@cs.stanford.edu)

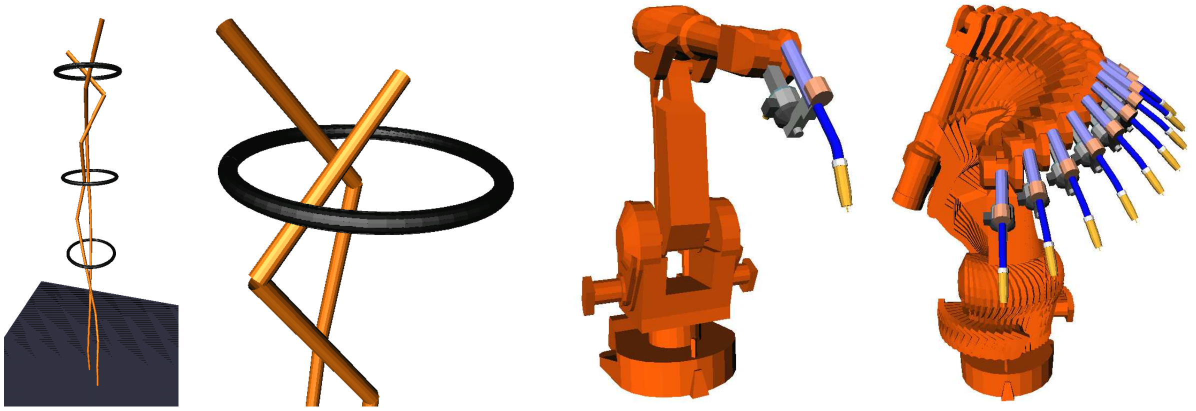


Fig. 1. *a*: Two skinny 20-DOF arms (320 triangles each) in environment with three fixed thin tori (6,000 triangles each), and detail showing arms in top ring. *b*: IRB 2400 robot with thin arc welding gun (3,500 triangles) and snapshots along a straight path segment in c-space

high degrees, so that solving the condition for t can be prohibitively expensive and furthermore pose numerical problems.

B. Fixed-resolution dynamic checking

Hence, while there exist effective methods for static collision checking (e.g., BVH methods), dynamic checking remains a practical bottleneck in many applications. In particular, probabilistic roadmap (PRM) planners rely on the availability of efficient checkers to test straight path segments (called local paths) between randomly sampled configurations (called milestones) [19], [20], [21], [22], [23], [24]. Most PRM planners use a static BVH method to test intermediate configurations spaced along each local path at some given resolution ε (in the c-space metric). These configurations are usually obtained by recursively bisecting the local path, until either a collision is found, or any two successive configurations are closer apart than ε [24], [25].

Choosing ε requires a delicate compromise between efficiency and reliability. This is especially true in scenarios with articulated arms and/or multiple robots. Rather large values of ε, which reduce the number of static checks, may be acceptable when robot links and obstacles are fat. But when these objects are thin or have sharp edges, the checker will then easily miss collisions. Trying to increase reliability by reducing ε results in slow checking of path segments. In the example of Figure 1*a*, which contains two long skinny serial linkages and three thin obstacles, tiny changes in joint angles can make the linkages jump over obstacles and/or each other. In Figure 1*b*, a small rotation of the robot's base may cause the welding gun to pass through an obstacle. In both examples, ε must be set very small for collisions to be reliably detected, yielding a slow dynamic checker.

One way to address this difficulty is to "grow" object models [26], by pre-computing the Minkowski sums of the original models and a sphere of radius ϱ. The value of ϱ is chosen such that, if the grown models do not overlap at the intermediate configurations, then the original models are guaranteed to be collision-free between these configurations. However, this further complicates the choice of ε. A large ϱ allows a large ε, but it also increases the chances that a path segment is incorrectly found to collide. To reduce the number of false collisions, while rarely missing true collisions, one must choose both ϱ and ε small, which yields again an inefficient checker. The checker proposed in [26] addresses this difficulty by pre-computing several grown copies of each model, each with a different value of ϱ. Then, at each step of a segment test, the checker switches to the largest collision-free grown model and adjusts the next step size accordingly. However, this approach is expensive (in both memory and time) and requires difficult tuning.

C. Adaptive bisection of paths

This paper introduces a new dynamic collision checking method that avoids these difficulties by locally adjusting the value of ε. By comparing lower bounds on distances between objects in relative motions with higher bounds on lengths of curves traced by points of these moving objects, this method automatically decides whether a path segment between two collision-free configurations needs to be bisected further. It not only frees the user from choosing ε; it also guarantees that no collision will ever be missed. Our checker is applicable to straight path segments in c-space or collections of such segments (e.g., multi-segment paths). It is particularly suited for scenarios with manipulator arms and/or multiple robots, where it can detect collision between robot links and obstacles, as well as between links of the same or different robots.

The basic idea of relating distances to path lengths has been suggested before (e.g., [1], [20]), though rarely implemented. We exploit this idea further by separately adjusting the bisection resolution for different pairs of objects. Indeed, very few pairs of objects (link-obstacle or link-link) require bisecting a path down to the same resolution. Exploiting this fact leads to testing a rapidly decreasing number of object pairs at each new level of bisection.

We also give a number of new techniques and heuristics that make the approach more efficient, especially when it is used in PRM planners and/or applied to geometrically complex environments. One is a BVH algorithm that computes non-trivial lower bounds on distances between pairs of objects almost as efficiently as if it was only testing the objects for collision. Another technique bounds link motions: given a straight path of a robot in c-space and a link of this robot, it quickly computes an upper bound on the lengths of the curves traced in workspace by all points in this link.

We have extensively tested our dynamic checker on randomly generated path segments and multi-segment paths produced by randomized planners. These tests show that it compares favorably to a fixed-resolution checker at "suitable" resolution, with the enormous advantage that it never fails to detect collision.

D. Paper organization

Section II describes our dynamic collision checking method based on adaptive bisection. Sections III and IV respectively present the techniques used to compute lower bounds on distances between objects and higher bounds on lengths of curves traced out by points of moving objects. Section V discusses experimental results on various examples that were obtained with the implemented new adaptive checker and a fixed-resolution checker, for comparison. Section VI summarizes our work, describes its limitations, and points to possible future work.

Throughout this paper, we assume that the workspace is the Euclidean space $\mathbf{R}^3$. Distances between objects and lengths of curves in workspace are all measured using the Euclidean metric.

II. Adaptive Dynamic Collision Checking

In this section we present our adaptive bisection algorithm for dynamic collision checking. We begin by establishing a basic result on which the algorithm is based.

A. Basic result

We consider the robot(s) and all obstacles as a collection of rigid objects $\mathcal{A}_1, \ldots, \mathcal{A}_n$ whose placements in workspace are uniquely determined by a vector $\boldsymbol{q} = (q_1, \ldots, q_d)$, the configuration of the system. All objects are allowed to move, provided that paths in c-space are piecewise linear.

Let $\mathcal{A}_i(\boldsymbol{q})$ denote object $\mathcal{A}_i$ at configuration $\boldsymbol{q}$. We define $\eta_{ij}(\boldsymbol{q})$ to be any non-trivial lower bound on the Euclidean distance between $\mathcal{A}_i(\boldsymbol{q})$ and $\mathcal{A}_j(\boldsymbol{q})$, *i.e.*, $\eta_{ij}(\boldsymbol{q}) = 0$ if and only if $\mathcal{A}_i(\boldsymbol{q})$ and $\mathcal{A}_j(\boldsymbol{q})$ overlap.

Each $\mathcal{A}_i$ is a set of points. Each point traces a distinct curve segment in workspace when the configuration of the system is linearly interpolated between configurations $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. These curves are usually *not* straight lines. We define $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ to be an upper bound on the lengths of the curves traced by all points in $\mathcal{A}_i$, such that $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) = 0$ whenever $\mathcal{A}_i$ stays fixed during the interpolation (for example, if A_i is a fixed obstacle).

We now state a sufficient condition for two objects $\mathcal{A}_i$ and $\mathcal{A}_j$ not to collide along a straight path segment in c-space:

Lemma 1: Two objects $\mathcal{A}_i$ and $\mathcal{A}_j$ do not collide at any configuration $\boldsymbol{q}$ on the straight path segment joining $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ in c-space, if:

$$\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) < \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b). \qquad (1)$$

Proof: Assume that Inequality (1) is verified. If $\mathcal{A}_i$ and $\mathcal{A}_j$ collide, then a point $\boldsymbol{p}_i$ of $\mathcal{A}_i$ must coincide with a point $\boldsymbol{p}_j$ of $\mathcal{A}_j$ at some intermediate configuration $\boldsymbol{q}_c$ along the straight path segment between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. Let $\ell_i(\boldsymbol{q}, \boldsymbol{q}')$ be the length traced by $\boldsymbol{p}_i$ between any two configurations $\boldsymbol{q}$ and $\boldsymbol{q}'$ in this segment. Define $\ell_j(\boldsymbol{q}, \boldsymbol{q}')$ in the same way for point $\boldsymbol{p}_j$. For $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ to coincide at $\boldsymbol{q}_c$, we must have:

$$\ell_i(\boldsymbol{q}_a, \boldsymbol{q}_c) + \ell_j(\boldsymbol{q}_a, \boldsymbol{q}_c) \geq \eta_{ij}(\boldsymbol{q}_a),$$

$$\ell_i(\boldsymbol{q}_c, \boldsymbol{q}_b) + \ell_j(\boldsymbol{q}_c, \boldsymbol{q}_b) \geq \eta_{ij}(\boldsymbol{q}_b).$$

Since $\ell_i(\boldsymbol{q}_a, \boldsymbol{q}_c) + \ell_i(\boldsymbol{q}_c, \boldsymbol{q}_b) = \ell_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ and $\ell_j(\boldsymbol{q}_a, \boldsymbol{q}_c) + \ell_j(\boldsymbol{q}_c, \boldsymbol{q}_b) = \ell_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$, summing the previous two relations yields:

$$\ell_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \ell_j(\boldsymbol{q}_a, \boldsymbol{q}_b) \geq \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b).$$

Using $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) \geq \ell_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ and $\lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) \geq \ell_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$, we get:

$$\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) \geq \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b)$$

which contradicts our initial hypothesis that Inequality (1) is verified. So, $\mathcal{A}_i$ and $\mathcal{A}_j$ do not collide. ■

The reverse of Lemma 1 is not true: $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$ may exceed $\eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b)$ without introducing a collision.

B. Adaptive bisection algorithm

Our algorithm uses Lemma 1 to decide whether the straight path segment between two collision-free configurations $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ must be bisected: if Inequality (1) is verified for all pairs of objects A_i and A_j, then the segment is collision-free; otherwise, it must be bisected. Lemma 1 guarantees that no collision can be missed.

However, considering all pairs of objects simultaneously may yield a large amount of unnecessary work. Indeed, pairs of objects that are well-separated and undergo small displacements require fewer bisections than pairs that are closer and/or undergo greater displacements. So, taking advantage of the fact that Inequality (1) applies to an individual pair of objects, we check this condition for each pair, independent of the other pairs, and discard those pairs which verify the inequality. In this way, as a path segment gets bisected at a finer resolution, the number of remaining object pairs tends to drop quickly. Moreover, we can perform the tests in an order that speeds up the discovery of a collision, when there is one.

Figure 2 shows our algorithm, ADAPTIVE-BISECTION, to check a path segment between two collision-free configurations $\boldsymbol{q}$ and $\boldsymbol{q}'$. It maintains a priority queue Q of elements of the form $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$. The presence of $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$ in Q indicates that the objects $\mathcal{A}_i$ and $\mathcal{A}_j$ still need to be tested for collisions between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. At Step 1, Q is filled with the elements $[\boldsymbol{q}, \boldsymbol{q}']_{ij}$, for all object pairs that need to be tested. Then, at each loop of Step 2, the first element, say $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$, is removed from Q. If this element satisfies Inequality (1), then $\mathcal{A}_i$ and $\mathcal{A}_j$ cannot possibly collide between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, and the algorithm continues with the next element in the queue. Else it computes $\eta_{ij}(\boldsymbol{q}_{mid})$, where $\boldsymbol{q}_{mid}$ is the mid-configuration between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. If this computation reveals a collision —

Algorithm ADAPTIVE-BISECTION($\boldsymbol{q}, \boldsymbol{q}'$)

1. Initialize priority queue Q with $[\boldsymbol{q}, \boldsymbol{q}']_{ij}$ for all pairs of objects $(\mathcal{A}_i, \mathcal{A}_j)$ that need to be tested.
2. While Q is not empty do
   - 2.1 $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij} \leftarrow$ remove-first(Q)
   - 2.2 If $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) \geq \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b)$ then
     - 2.2.1 $\boldsymbol{q}_{mid} \leftarrow (\boldsymbol{q}_a + \boldsymbol{q}_b)/2$
     - 2.2.2 If $\eta_{ij}(\boldsymbol{q}_{mid}) = 0$ then return *collision*
     - 2.2.3 Else insert $[\boldsymbol{q}_a, \boldsymbol{q}_{mid}]_{ij}$ and $[\boldsymbol{q}_{mid}, \boldsymbol{q}_b]_{ij}$ into Q
3. Return *no collision*

Fig. 2. Adaptive bisection algorithm

that is, if $\eta_{ij}(\boldsymbol{q}_{mid}) = 0$ — then the algorithm reports the collision and halts. Otherwise, two new elements, $[\boldsymbol{q}_a, \boldsymbol{q}_{mid}]_{ij}$ and $[\boldsymbol{q}_{mid}, \boldsymbol{q}_b]_{ij}$, are inserted into Q. When Q is empty, the path segment between $\boldsymbol{q}$ and $\boldsymbol{q}'$ is reported free of collision.

In Sections III and IV, we will describe the techniques used in our implementation of ADAPTIVE-BISECTION to compute bounds on distances and curve lengths.

C. Ordering of the priority queue

If the path segment tested by ADAPTIVE-BISECTION is collision-free, then the ordering of Q has no impact on the running time of the algorithm, since all elements in Q will eventually have to be processed. But, for a colliding path segment, an appropriate ordering can lead to finding a collision quicker. This can be important in many applications, like PRM planning, where a large fraction of candidate paths are colliding.

In [24], [25], it was shown that in practice, the prior probability of a path segment to be colliding increases sharply with its length in c-space. This result justifies bisecting a segment into two sub-segments of equal lengths. The planners in [24], [27] exploit this result further to test multi-segment paths, by maintaining a priority queue of (sub-)segments sorted by decreasing lengths and treating the longest (sub-)segment first.

ADAPTIVE-BISECTION takes also advantage of the computed bounds on both distances between objects and lengths of traced curves. Intuitively, two objects are more likely to collide when they are closer to each other at one or both segment endpoints and/or the points in these objects trace longer curves. This intuition is directly related to Inequality (1) and leads us to sort the entries $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$ in the priority queue by decreasing values of the difference: $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) - \eta_{ij}(\boldsymbol{q}_a) - \eta_{ij}(\boldsymbol{q}_b)$. Our tests show that, on average, this ordering leads to finding a collision faster than sorting Q by decreasing lengths of segments.

D. Checking multi-segment paths

ADAPTIVE-BISECTION can be extended to concurrently test multiple segments forming a continuous, multi-segment path. This is simply done by filling Q, at Step 1, with the elements $[\boldsymbol{q}, \boldsymbol{q}']_{ij}$, for all segments $[\boldsymbol{q}, \boldsymbol{q}']$ and all object pairs that need to be tested. The same heuristic ordering of Q as above can be used. In general, it will lead to discovering a colliding segment before having spent much time testing collision-free segments.

However, a slightly more useful implementation in practice is to maintain several priority queues: one for the path and one for each segment. The priority queues for the individual segments are maintained as above. The queue for the path contains one entry per segment that has not yet been shown to be collision-free. The entries of the path queue are sorted by decreasing values of the differences $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) - \eta_{ij}(\boldsymbol{q}_a) - \eta_{ij}(\boldsymbol{q}_b)$, each computed for the first element of the corresponding segment's queue. Intuitively, this ordering corresponds to placing the segment that is the most likely to collide on top of the path queue. The advantage of using several queues is the following: if the path is eventually found to collide, we can then cache the priority queues of the segments that have not yet been found to collide or not to collide. In this way, if these segments are later parts of other paths, the collision-checking work already done can be retrieved.

The same extension actually holds for an arbitrary collection of segments.

E. Covering strategies

Inequality (1) can be illustrated by the following diagram: draw a line segment of length $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$ and two circles of radii $\eta_{ij}(\boldsymbol{q}_a)$ and $\eta_{ij}(\boldsymbol{q}_b)$ centered at the endpoints of this segment. See Figure 3a. (The segment in this figure is *not* the path segment in c-space.) We call the circles the *covering circles* of $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. If they cover the entire segment, as is the case in Figure 3a, then $\mathcal{A}_i$ and $\mathcal{A}_j$ do not collide along the straight path joining $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ in c-space. Otherwise, and if there is no collision along the path, ADAPTIVE-BISECTION will produce intermediate configurations $\boldsymbol{q}_{mid}$ whose covering circles will eventually complete the coverage of the entire line segment, as depicted in Figure 3b.

The above diagram, though only illustrative (as it is embedded neither in c-space, nor in workspace), suggests decomposition/covering strategies of the path between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ other than bisecting at the midpoint. For example, we could compute the two configurations where the covering circles of $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ intersect the segment (Figure 3c), and check whether their covering circles complete the covering of the segment. This strategy would require inserting at most one new entry into the priority queue at each iteration, instead of two. But half the coverage by the new circles would be wasted to cover parts of the segment that were covered by previous circles. This could nevertheless be a useful strategy if one wanted to check a path by small increments from end to the other, as is the case, for example, in haptic interaction with virtual worlds.

Another strategy would be to place a configuration at the middle point of the uncovered section (assuming it can be easily computed), as illustrated in Figure 3d. In term of coverage, this would be slightly better than the bisection strategy of our algorithm. However, our experimental tests indicate

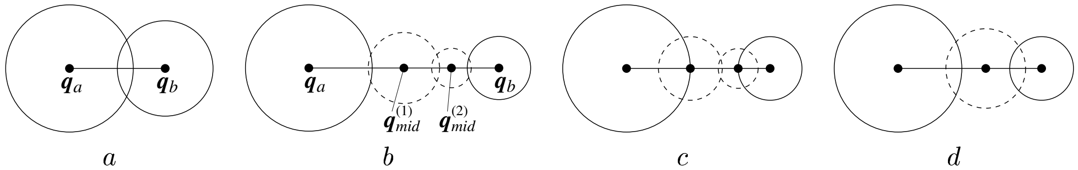


Fig. 3. Different covering strategies (see text)

that the potential gains are small. Moreover, this strategy produces different intermediate configurations for different pairs of objects. This is a drawback with articulated linkages, since it then requires computing the forward kinematics of these linkages at many more configurations. Instead, with the strategy of Figure 3*b*, the results of the forward kinematics computation done to determine the placement of a link at some configuration can be cached and later re-used to retrieve or compute the placement of another link in the same linkage at the same configuration.

F. Bounding the running time of ADAPTIVE-BISECTION

Ignoring floating-point arithmetics issues, ADAPTIVE-BISECTION always gives a correct answer in a finite amount of time. However, this time is not bounded in the worst case, as one can easily create examples where the algorithm would bisect an arbitrary number of times. Very bad cases are unlikely in practice, but they may eventually occur when several thousands of path segments are tested, as is often the case in PRM planning.

There are several ways to deal with this issue. One is to switch to another dynamic collision-checking method when a (sub-)segment shorter than some threshold requires being bisected further. Then, the potential collision is already well localized, so that only restricted subsets of the objects (hence, small number of triangles) need to be considered. Thus, a method like the swept-volume method (see Section I-A) could be applicable.

Another way is to modify slightly the definition of the lower bound $\eta_{ij}(\boldsymbol{q})$ on the distance between $\mathcal{A}_i$ and $\mathcal{A}_j$ at configuration $\boldsymbol{q}$. In the new definition, $\eta_{ij}(\boldsymbol{q}) = 0$ whenever the actual distance between the objects is less than some small predefined δ, and $\eta_{ij}(\boldsymbol{q}) \geq \delta$ otherwise. Any $\delta > 0$ results in bounding the running time of ADAPTIVE-BISECTION. Then, the algorithm is slightly conservative: while it still cannot miss a collision, it may incorrectly return that a path segment is colliding when two objects come closer than δ apart. Note that choosing δ is very different from setting the resolution ε of a fixed-resolution checker. A fixed-resolution checker always breaks a path segment into sub-segments of length ε to determine that the segment is collision-free. In contrast, in most cases, our checker stops bisecting before any $\eta_{ij}(\boldsymbol{q})$ gets smaller than δ. Setting δ is also different from growing the objects by some ϱ as suggested at the end of Section I-B. Indeed, ϱ must be chosen large enough to prevent any collision from happening along sub-segments of length ε. So, ϱ is directly related to the length of the maximal displacement of points in the moving objects, and can be quite large.

In our current implementation, we bound the running time of ADAPTIVE-BISECTION by setting a threshold δ as described above. In our experiments, we found that δ can be chosen very small without affecting significantly the average running time of ADAPTIVE-BISECTION. This is due to the fact that cases where this threshold is actually needed do not happen too often in practice.

III. Computing Lower Bounds on Distances Between Objects

ADAPTIVE-BISECTION requires that non-trivial lower bounds on distances between objects be efficiently computed. Tighter bounds may yield fewer bisections, but are usually more expensive to compute. Here, we describe a greedy algorithm, based on bounding volume hierarchies (BVHs), that computes lower bounds on distances at about the same cost as if it was testing the objects for collision.

A. Distance computation using BVHs

Bounding volume hierarchies (BVHs) are widely used to check collision and/or compute distances between objects of high geometric complexity. An object's BVH is an approximately balanced binary tree whose leaves are triangles of the object surface and intermediate nodes are BVs, each bounding the triangles below it. It is pre-computed using techniques such as those described in [9], [12], [13]. Various types of BVs have been proposed, including spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB), and rectangle swept spheres (RSS) [3].

To determine if two objects collide at a given configuration, their hierarchies are searched from the top down. Each step of a search path consists of testing whether two BVs or two triangles, one from each hierarchy, are separated. A collision is reported whenever two triangles are found to intersect. A search path is terminated as soon as two tested BVs are separated, because disjoint BVs cannot contain intersecting triangles. In the worst case, if each object has $O(n)$ triangles, the algorithm takes $O(n^2)$ time. But, in practice, the average running time is usually sub-linear, as many search paths are terminated early.

The same algorithm can be used to compute the exact distance between two objects, with few changes to maintain the closest distance d found so far. Initially, d is set to a very large number. A search path is terminated whenever the distance between two tested BVs is greater than d. If a search path reaches a pair of triangles and the distance between these triangles is less than d, then d is set to this new distance. The algorithm terminates as soon as $d = 0$ (the objects are colliding) or when all search paths have been terminated. In both cases, the final value of d is the distance between the two objects.

BVH methods are widely used, because they have been found more efficient in practice than other techniques, especially for complex objects. Nevertheless, they take much more time to compute distances than to check collision. As any two tested BVs are more likely to be disjoint than separated by at least $d > 0$, search paths to compute distances are longer.

Two approaches have been proposed to speed up distance computation:

- One approach is to prune search paths more effectively, either by re-using data computed at a previous configuration, or by better selecting which search path to explore next [12], [28]. For example, *triangle caching* initializes d to the distance between the two closest triangles at the previous configuration. But it is effective only if the coherence assumption is verified. *Priority directed search* schedules the pending tests of BVs and triangles into a priority queue sorted by distances. It does not rely on any coherence assumption.
- The other approach computes an *approximate* distance with a *guaranteed* bound on the relative error [12], [13]. The algorithm in [13] computes a lower bound d' on the distance d between two objects, such that $(d-d')/d \leq \gamma$, where $0 < \gamma < 1$ is an input constant. It initializes d' to a large value (as in the exact case). When it finds that two triangles are closer than d' apart, it resets d' to be $1-\gamma$ times the distance between them. The final value of d' verifies $(1-\gamma)d \leq d' \leq d$. The algorithm becomes faster as γ is increased, but it is still slower than pure collision checking. The algorithm in [12] computes a higher bound on the distance.

In fact, the speed of distance computation turns out to be crucial for the overall efficiency of the new adaptive dynamic collision checking approach. That is, while ADAPTIVE-BISECTION may perform fewer bisections to test a path than a fixed-resolution checker with small ε, it could nevertheless take longer to run because standard methods for distance computation are much slower than pure collision checking. The next section therefore proposes a new algorithm, called GREEDY-DIST, that computes lower distance bounds almost as fast as a typical BVH algorithm checks for collision.

B. Greedy distance computation algorithm

To compute a lower bound on the distance between two objects, our implementation of ADAPTIVE-BISECTION calls GREEDY-DIST(B_α, B_β), where GREEDY-DIST is the algorithm shown in Figure 4 and B_α and B_β identify the root BVs of the hierarchies representing these objects. The call returns a positive lower bound on the distance, if the objects are separated, and 0 otherwise.

GREEDY-DIST works like a classical BVH collision checker [12]. It follows the same search paths, tests the same pairs of BVs and triangles, and terminates each search path when a pair of tested BVs has null intersection. But, instead of testing if two BVs or triangles intersect, it computes the distance between them and eventually returns the smallest distance found. It is faster than an approximate distance computation algorithm because it skips the additional search needed to verify that the relative error is smaller than a given γ. Though GREEDY-DIST offers no guarantee on the relative error, our tests show that it returns a good approximation on average.

Algorithm GREEDY-DIST(B_i, B_j)

1. $d \leftarrow distance(B_i, B_j)$
2. If B_i and B_j are both triangles then return d
3. If $d > 0$ then return d
4. If B_i is bigger than B_j then switch B_i and B_j
5. Set B_{j1} and B_{j2} to the two children of B_j in the BVH
6. $\alpha \leftarrow$ GREEDY-DIST(B_i, B_{j1})
7. If $\alpha > 0$ then
   7.1 $\beta \leftarrow$ GREEDY-DIST(B_i, B_{j2})
   7.2 If $\beta > 0$ then return $\min\{\alpha, \beta\}$
8. Return 0

Fig. 4. Greedy distance computation algorithm

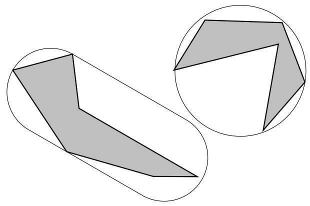


Fig. 5. A bad case for GREEDY-DIST: two distant objects are bounded by disjoint RSSs that are very close

GREEDY-DIST is independent of the choice of BV, as long as the distance between pairs of BVs can be computed efficiently. In our implementation, we use RSSs. An RSS is defined as the Minkowski sum of a rectangle and a sphere [12]. The RSS of an object is created by first estimating the two principal directions spanned by the object [9]. A rectangle R is then constructed along these directions to enclose the projection of the object onto the plane defined by these directions. The RSS is the Minkowski sum of R and the sphere whose radius is half the length of the interval spanned by the object along the direction perpendicular to R. In comparison, the OBB of the object is the cross-product of R by this interval. It is often possible to improve the quality of the RSS fit by shrinking R (each side can be reduced by at most twice the radius). Like OBBs, RSSs provide reasonably tight fits for objects of various shapes. But distance computation between RSSs is faster than between OBBs [12].

To bound the running time of ADAPTIVE-BISECTION using the parameter δ (see Section II-F), we modify the algorithm of Figure 4 by replacing the zeros in step 3, 7, and 7.2 by δ. Then, GREEDY-DIST returns a positive lower bound whenever the distance is greater than δ, and 0 otherwise.

C. Experimental analysis

As Figure 5 illustrates, in a bad case, the relative error on a bound computed by GREEDY-DIST can be arbitrarily close to 1 (when $\delta = 0$). Our experiments, however, show that the average bounds returned by GREEDY-DIST are quite good and

Fig.	# queries	COLL-CHECKER	GREEDY-DIST	EXACT-DIST	APPROX-DIST	μ	μ'
1a	40,000	1.7 / 0.1	2.1 / 0.1	859 / 141	3.7 / 0.1	0.99	0.53
11a	54,000	47 / 0.6	54 / 0.8	946 / 78	105 / 2.0	0.82	0.64
11b	42,000	26 / 1.0	25 / 0.8	827 / 67	58 / 1.9	0.81	0.61
12a	14,000	14 / 0.4	14 / 0.3	1,444 / 268	70 / 2.2	0.64	0.60
12b	22,000	42 / 0.9	46 / 1.0	819 / 125	129 / 5.4	0.51	0.58
12c	18,000	25 / 1.3	26 / 1.4	1,248 / 128	111 / 3.2	0.57	0.62
12d	21,000	2.6 / 0.1	3.9 / 0.2	703 / 170	74 / 16	0.58	0.58

TABLE I

COMPARISON OF COLL-CHECKER, GREEDY-DIST, EXACT-DIST AND APPROX-DIST (SEE TEXT)

compare favorably to those computed (at a greater cost) by an approximate distance computation algorithm.

We tested GREEDY-DIST (with $\delta = 0$) and three similar BVH algorithms: COLL-CHECKER for pure collision checking, EXACT-DIST for exact distance computation, and APPROX-DIST for approximate distance computation with relative error $\gamma = 0.5$. APPROX-DIST uses the approach of [13], hence returns a value between $0.5d$ and d, where d is the exact distance. We use RSSs in EXACT-DIST and APPROX-DIST, as in GREEDY-DIST, but OBBs in COLL-CHECKER. Indeed, while being closely related to RSSs, OBBs are slightly faster to test for overlap. Finally, we "tuned" EXACT-DIST using both priority directed search and triangle caching (see Section III-A). To make the best use of triangle caching, we initialize the distance d (used for pruning the search) with the exact distance (computed separately). This leads EXACT-DIST to terminate a search path as soon as a BV pair is found to be further apart than the exact distance. In practice, such perfect initialization is impossible, and could only be approached in cases where the coherence assumption is verified extremely well.

We compared the performance of the four algorithms as follows. For each of the seven environments shown in Figures $1a$, $11a$-b and $12a$-d, we generated 1,000 random configurations of the robot(s) and, at each configuration, we tested all robot links against all fixed obstacles. The results are summarized in Table I. Column 1 identifies the environment and column 2 indicates the total number of object pairs examined by each of the four algorithms. Columns 3-6 give the average numbers of pairs of BVs/triangles tested per query by each of the four algorithms. As expected, the numbers for COLL-CHECK and GREEDY-DIST are about the same. The small differences result from the fact that one uses OBBs and the other RSSs. Since computing the distance between two RSSs is only a small factor slower than testing two OBBs for overlap, GREEDY-DIST is almost as fast as COLL-CHECK. In contrast, EXACT-DIST examines many more pairs of BVs and triangles, despite the perfect initialization of triangle caching. APPROX-DIST also examines significantly more BVs and triangles than GREEDY-DIST; its running time is greater in the same ratio.

The last two columns of Table I measure the quality of the bounds. Column 7 gives the average ratio μ of the bound returned by GREEDY-DIST by the exact distance, in each environment, computed only for the collision-free pairs of objects. The values of μ are excellent for the first three environments in which obstacles are tightly bounded by RSSs, and they remain greater than 0.5 in the other four environments, where objects have diverse shapes. The last column gives the average ratio μ' computed for the bounds returned by APPROX-DIST. In the first three environments, μ' is smaller than μ; in the last four, the two factors are similar, meaning that GREEDY-DIST returns on average similar bounds, at smaller computational cost.

For ADAPTIVE-BISECTION, the average performance of the algorithm computing distance bounds matters more than its worst-case performance. Indeed, if GREEDY-DIST returns a bad lower bound, then ADAPTIVE-BISECTION may have to bisect the segment once more and call GREEDY-DIST again at a new configuration. The probability of encountering several bad cases in a row is small. We have tested ADAPTIVE-BISECTION with GREEDY-DIST, EXACT-DIST, and APPROX-DIST (with different values of γ). The best results were obtained with GREEDY-DIST.

IV. BOUNDING MOTIONS IN WORKSPACE

ADAPTIVE-BISECTION also requires computing an upper bound $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ on the lengths of the curve segments traced by all points of a moving object $\mathcal{A}_i$, when the system is linearly interpolated between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$. Tight bounds could be computed by numeric integration, but this may be quite slow in practice. Moreover, the cost of the computation would increase with the distance between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$.

In this section, we derive the expression of an upper bound $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ that is fast to evaluate, and is valid for kinematic chains with revolute and prismatic joints. We first establish this expression on a simple example, then in the general case. Next, we propose a technique to compute constant factors appearing in this expression. Finally, we experimentally evaluate the quality of the bound.

Throughout this section, we adopt simplifying conventions that do not restrict the generality of our results. We assume a one-to-one correspondence between the joint parameters of the robot(s) and moving obstacles and the configuration parameters. We define these parameters such that: (1) a rotation of any revolute joint by some angle ω (in radians) produces a variation of ω of the corresponding configuration parameter, and (2) a translation of any prismatic joint by some vector $\boldsymbol{v}$ results in a variation of $||\boldsymbol{v}||$ of the corresponding parameter.

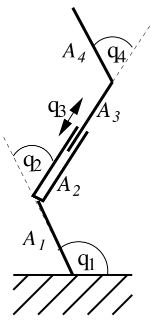


Fig. 6. Planar linkage with three revolute joints and one prismatic joint

A. Example

We first establish an expression of $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ for the planar linkage of Figure 6. This linkage has three revolute joints corresponding to joint parameters q_1, q_2, and q_4, and one prismatic joint corresponding to q_3. Each of the links $\mathcal{A}_1, \ldots, \mathcal{A}_4$ has length L and zero width. Each revolute joint can perform a single full rotation, hence we define q_1, q_2, and q_3 to vary, each, between 0 and 2π. When prismatic joint 3 is completely retracted, the distance between revolute joints 2 and 4 (equivalently, between the base of $\mathcal{A}_2$ and the tip of $\mathcal{A}_3$) is L. When joint 3 is maximally extended, this distance is $L + D$. So, we define q_3 to vary between 0 and D.

For this linkage, we can write the bounds λ_i ($i = 1, \ldots, 4$) as follows:

$$
\begin{aligned}
\lambda_1(\boldsymbol{q}_a, \boldsymbol{q}_b) &= L\,|q_{a,1} - q_{b,1}| \\
\lambda_2(\boldsymbol{q}_a, \boldsymbol{q}_b) &= 2L\,|q_{a,1} - q_{b,1}| + L\,|q_{a,2} - q_{b,2}| \\
\lambda_3(\boldsymbol{q}_a, \boldsymbol{q}_b) &= (2L + D)\,|q_{a,1} - q_{b,1}| \\
&+ (L + D)\,|q_{a,2} - q_{b,2}| + |q_{a,3} - q_{b,3}| \\
\lambda_4(\boldsymbol{q}_a, \boldsymbol{q}_b) &= (3L + D)\,|q_{a,1} - q_{b,1}| \\
&+ (2L + D)\,|q_{a,2} - q_{b,2}| + |q_{a,3} - q_{b,3}| \\
&+ L\,|q_{a,4} - q_{b,4}|
\end{aligned}
$$

The bound $\lambda_1(\boldsymbol{q}_a, \boldsymbol{q}_b)$ is established by considering the point in $\mathcal{A}_1$ that is the furthest away from the center of rotation of joint 1. While the second term of $\lambda_2(\boldsymbol{q}_a, \boldsymbol{q}_b)$ is established in a similar way, its first term is derived by considering the maximal distance between a point of $\mathcal{A}_2$ and the center of rotation of joint 1. This distance is achieved by the tip of $\mathcal{A}_2$ when $\mathcal{A}_1$ and $\mathcal{A}_2$ are aligned. Clearly, no point of $\mathcal{A}_2$ can move by a larger amount than the sum of the two terms defining $\lambda_2(\boldsymbol{q}_a, \boldsymbol{q}_b)$. The other two bounds are generated in a similar way.

We can write each of the bounds above in the form:

$$\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) = \sum_{k=1}^{i} R_k^i \; |q_{b,k} - q_{a,k}|$$

where $i = 1, \ldots, 4$. If k is a prismatic joint, then $R_k^i = 1$; otherwise, R_k^i is is an upper bound on the distances between the points of $\mathcal{A}_i$ and the center of rotation of joint k. So, in the above expression, we have:

$$
\begin{array}{llll}
R_1^1 = L & & & \\
R_1^2 = 2L, & R_2^2 = L & & \\
R_1^3 = 2L + D, & R_2^3 = L + D, & R_3^3 = 1 & \\
R_1^4 = 3L + D, & R_2^4 = 2L + D, & R_3^4 = 1, & R_4^4 = L
\end{array}
$$

B. Upper bound in general case

In the following, we focus our attention on the moving object $\mathcal{A}_i$. Without loss of generality, we let $q_1, \ldots, q_i$ denote the configuration parameters that influence $\mathcal{A}_i$'s placement.

Lemma 2: When the system configuration is linearly interpolated between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, an upper bound on the lengths of the curves traced by the points of object $\mathcal{A}_i$ is:

$$\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) = \sum_{k=1}^{i} R_k^i \; |q_{b,k} - q_{a,k}| \tag{2}$$

where $q_1, \ldots, q_i$ are the configuration parameters that influence the placement of $\mathcal{A}_i$, and $R_k^i > 0$ is a constant factor defined as follows:
- if k is a prismatic joint, then $R_k^i = 1$,
- otherwise, R_k^i is an upper bound on the distances between the points of $\mathcal{A}_i$ and the axis of rotation of joint k.

Proof: Let $\boldsymbol{p}$ be an arbitrary point of $\mathcal{A}_i$. The straight path segment in c-space between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ can be parameterized by $\boldsymbol{q}(t) = (1-t)\boldsymbol{q}_a + t\boldsymbol{q}_b$, with $t \in [0, 1]$. Let $\mathcal{A}_i(t)$ and $\boldsymbol{p}(t)$ denote the placement of $\mathcal{A}_i$ and $\boldsymbol{p}$ at configuration $\boldsymbol{q}(t)$.

The length $L_{\boldsymbol{p}}$ of the curve traced by $\boldsymbol{p}(t)$ when t varies from 0 to 1 is:

$$L_{\boldsymbol{p}} = \int_0^1 \|\dot{\boldsymbol{p}}(t)\| dt \tag{3}$$

where:

$$\dot{\boldsymbol{p}}(t) = \sum_{k=1}^{i} \frac{\partial \boldsymbol{p}}{\partial q_k} \dot{q}_k(t).$$

We can bound the modulus of $\dot{\boldsymbol{p}}(t)$ by applying the triangle inequality:

$$0 \;\leq\; \|\dot{\boldsymbol{p}}(t)\| \;\leq\; \sum_{k=1}^{i} \left\| \frac{\partial \boldsymbol{p}}{\partial q_k} \right\| |\dot{q}_k(t)| \,. \tag{4}$$

Along the straight path between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, we have:

$$\dot{q}_k(t) = q_{b,k} - q_{a,k}. \tag{5}$$

Moreover, by definition of the configuration parameters and the constants R_k^i, we have:

$$\left\| \frac{\partial \boldsymbol{p}}{\partial q_k} \right\| \leq R_k^i. \tag{6}$$

By first plugging (4) into (3) and then using relations (6) and (5), we get:

$$
\begin{aligned}
L_{\boldsymbol{p}} &\leq \int_0^1 \sum_{k=1}^{i} \left\| \frac{\partial \boldsymbol{p}}{\partial q_k} \right\| |\dot{q}_k(t)|\, dt \\
&\leq \sum_{k=1}^{i} R_k^i \int_0^1 |\dot{q}_k(t)|\, dt \\
&= \sum_{k=1}^{i} R_k^i \; |q_{b,k} - q_{a,k}| \,.
\end{aligned} \tag{7}
$$

Algorithm COMPUTE-SPHERE(i, k)

1. If $i = k$ then $S(i, k+1) \leftarrow$ ENCLOSING-SPHERE$(\mathcal{A}_i)$
2. Else $S(i, k+1) \leftarrow$ COMPUTE-SPHERE$(i, k+1)$
3. If joint k is prismatic then
   Sweep $S(i, k+1)$ along the full translational range of joint k and construct the sphere $S(i, k)$ that tightly encloses the swept volume.
4. Else if joint k is revolute then
   Sweep $S(i, k+1)$ around the axis of joint k by performing a full 2π rotation and construct the sphere $S(i, k)$ that tightly encloses the swept volume.
5. Return $S(i, k)$

Fig. 7. Computation of sphere $S(i, k)$

Since we made no assumption on the location of $\boldsymbol{p}$ in $\mathcal{A}_i$, this bound holds for all points of $\mathcal{A}_i$. ■

Note that the bound $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ defined by Equation (2) is null if and only if $\boldsymbol{q}_a = \boldsymbol{q}_b$. In general, we use Equation (2) to compute both $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ and $\lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$ in (1). However, for two links i and j ($j < i$) on the same kinematic chain, we can derive tighter bounds by considering motions in the local frame of link j. Thus, $\lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) = 0$ and for $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$, we simply sum over $k = j+1, \ldots, i$ instead of $k = 1, \ldots, i$ in (2).

The above result in Lemma 2 can be extended to parameters $q_k(t)$ that are non-linear functions in the "pseudo-time" t, as long as $|\dot{q}_k(t)|$ can be bounded. An upper bound on $|\dot{q}_k(t)|$ then replaces $|q_{b,k} - q_{a,k}|$ in (7), hence in (2). This modification covers any linkage made of revolute and/or prismatic joints, including those with closed loops. (Note that parallelogram mechanisms, which involve only linear parametric changes, can be handled directly without this extension.) It also makes it possible to bound curve lengths when the path segment in c-space is not straight.

C. Computation of factors R_k^i

We now describe a general technique to compute the constant factors R_k^i in Equation (2) for robot arms in three-dimensional workspace, when k is a revolute joint. Recall that R_k^i must be an upper bound on the distances between the points of $\mathcal{A}_i$ and the axis of rotation of joint k.

Our technique computes a sphere bounding the volume swept out by each link when the configuration parameters vary over their full ranges (which we take to be at least 2π for revolute joints). Thus, the computed factors R_k^i are independent of any considered path segment in c-space, and are computed only once during pre-processing.

If $i = k$, then we compute the distances of all link vertices to the axis of rotation of joint i and return R_i^i to be the maximum of these distances.

For every $k < i$, we compute a sphere $S(i, k)$ of radius $R_k^i = r(i, k)$ centered at a point $\boldsymbol{c}(i, k)$ located in the axis of rotation of joint k, that is guaranteed to enclose link i for any values of the configuration parameters $q_k, \ldots, q_i$.

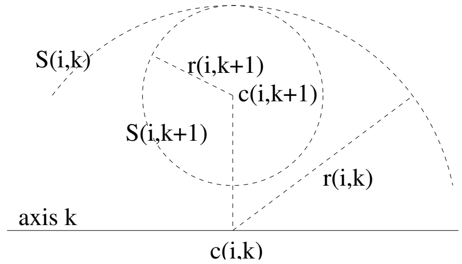


Fig. 8. Construction of $S(i, k)$ at Step 4 of algorithm COMPUTE-SPHERE

The spheres $S(i, k)$, $k \leq i$, are computed recursively by the algorithm of Figure 7. At Step 1, ENCLOSING-SPHERE$(\mathcal{A}_i)$ computes a tight enclosing sphere of $\mathcal{A}_i$ using a standard algorithm [29]. At Step 3, computing the sphere that tightly encloses the volume swept by moving $S(i, k+1)$ along the entire range of the prismatic joint k is straightforward. At Step 4, we proceed as follows: we compute the center $\boldsymbol{c}(i, k)$ of $S(i, k)$ as the projection of $\boldsymbol{c}(i, k+1)$ on the axis of rotation of joint k and the radius $r(i, k)$ of $S(i, k)$ as the sum of $r(i, k+1)$ and the distance between $\boldsymbol{c}(i, k+1)$ and $\boldsymbol{c}(i, k)$, as illustrated in Figure 8.

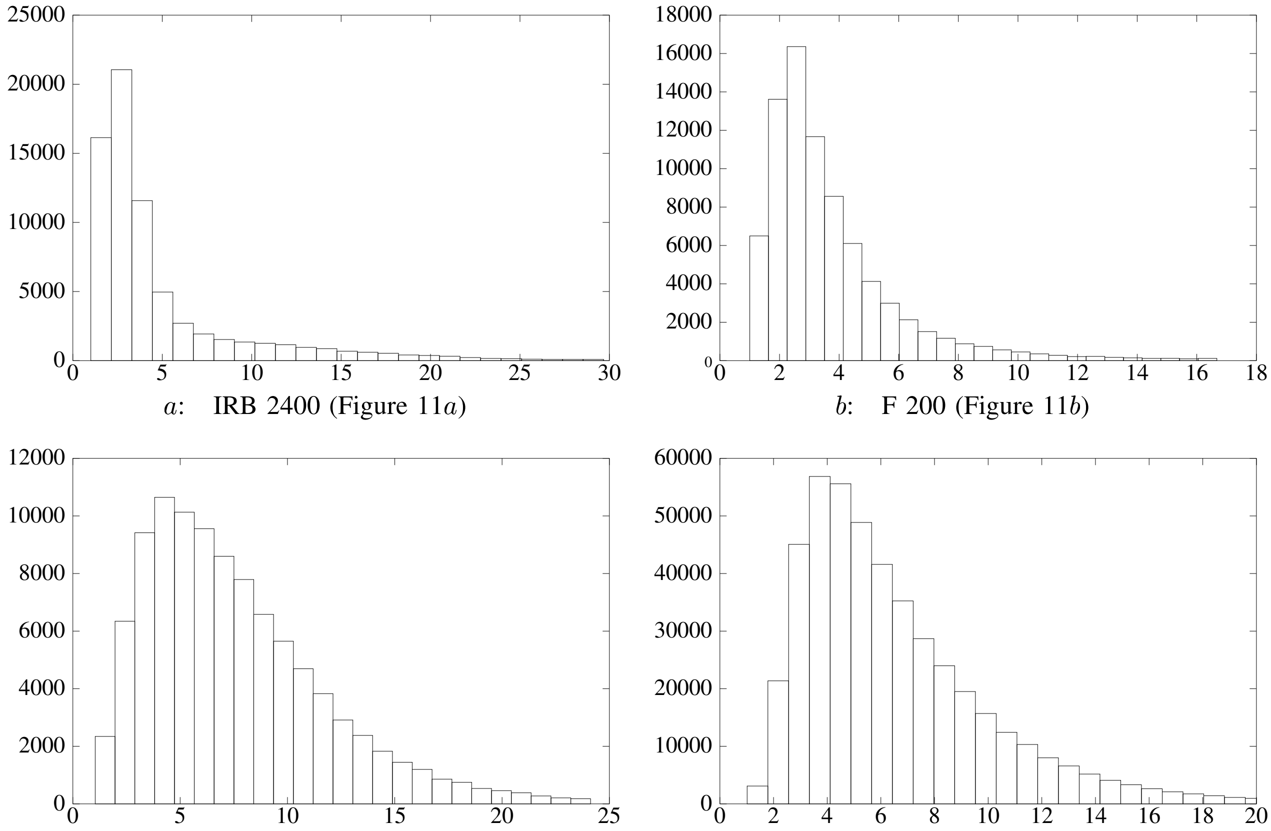
D. Experimental analysis

The bounds defined as above are clearly conservative, but they are also very fast to compute. To evaluate their average quality, we made the following experiments with the robots of Figures 1a, 11a, and 11b.

For each robot, we generated 10,000 segments $[\boldsymbol{q}, \boldsymbol{q}']$ by uniformly sampling two independent endpoints $\boldsymbol{q}$ and $\boldsymbol{q}'$. For each segment, we computed the bounds $\lambda_i(\boldsymbol{q}, \boldsymbol{q}')$ for all links, as well as lower bounds $\ell_i(\boldsymbol{q}, \boldsymbol{q}')$ obtained by integrating Equation (3) at a low resolution for some randomly chosen point on $\mathcal{A}_i$. We computed the ratio $\lambda_i(\boldsymbol{q}, \boldsymbol{q}')/\ell_i(\boldsymbol{q}, \boldsymbol{q}')$ for each link over all 10,000 segments. Note that this ratio is not bounded in the worst case, even if $\ell_i(\boldsymbol{q}, \boldsymbol{q}')$ was the exact length of the longest curve traced by a point of $\mathcal{A}_i$.

The histograms a, b, and c in Figure 9 present 99% of the results in order to crop outliers. They show that, most of the time, the bounds $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ are within a rather small factor from optimal. For example, for the IRB 2400 and F 200 robots, more than 80% of them are within factor 5 from the corresponding lower bounds $\ell_i(\boldsymbol{q}, \boldsymbol{q}')$. Even for the hyper-redundant arm (histogram c), the ratio is smaller than 15 most of the time.

The histogram d shows the distribution of the ratios $\lambda_i(\boldsymbol{q}, \boldsymbol{q}')/\ell_i(\boldsymbol{q}, \boldsymbol{q}')$ as above plus additional, slightly different, ratios $\lambda_{ij}(\boldsymbol{q}, \boldsymbol{q}')/\ell_{ij}(\boldsymbol{q}, \boldsymbol{q}')$ that account for self-collision testing, as follows. For two links $\mathcal{A}_i$ and $\mathcal{A}_j$ ($j < i$) of the same robot, we compute $\lambda_{ij}(\boldsymbol{q}, \boldsymbol{q}')$ to bound the lengths of curves traced by all points on $\mathcal{A}_i$ in $\mathcal{A}_j$'s local reference frame, as described in Section IV-B. Similarly, $\ell_{ij}(\boldsymbol{q}, \boldsymbol{q}')$ is a non-trivial lower bound on the length of curves traced by all points of $\mathcal{A}_i$ in $\mathcal{A}_j$'s local frame, obtained by integrating Equation (3), in $\mathcal{A}_j$'s frame, for some randomly chosen point on $\mathcal{A}_i$.



a: IRB 2400 (Figure 11*a*) *b*: F 200 (Figure 11*b*)

c: 20-DOF arm (Figure 1*a*) *d*: 20-DOF arm (inter-link tests)

Fig. 9. Histograms showing the distributions of the quality of the bound $\lambda_i(\boldsymbol{q}, \boldsymbol{q}')$ for three robots. The horizontal axis shows a higher bound on the ratio of $\lambda_i(\boldsymbol{q}, \boldsymbol{q}')$ by the length of the curve. The closer to 1 this ratio, the better the quality. Histogram *d* additionally includes slightly different ratios that account for self-collision testing (see text)

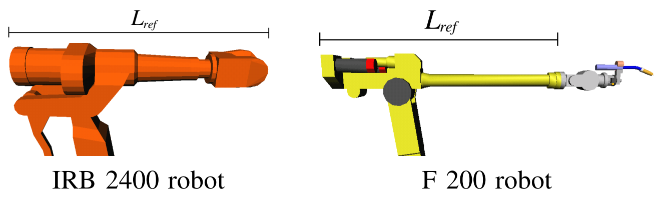


Fig. 10. Reference links of IRB 2400 and F 200 robots

Fig.	1a	11a	11b	11c	12a	12b	12c	12d
L_{ref}	0.5	1.77	1.65	0.5	1.16	0.64	1.59	1.32

TABLE II

L_{ref} (IN UNITS) IN DIFFERENT EXAMPLES.

V. Experimental Results

We have experimented with ADAPTIVE-BISECTION on numerous problems ranging from testing randomly generated segments, to testing local paths in multi- and single-query roadmaps, to optimizing jerky paths. In this section, we present a subset of our results.

A. Experimental setup

All the results below were obtained on a 1GHz Pentium III PC with 1GB RAM. Our implementation of ADAPTIVE-BISECTION computes lower bounds on distances between objects using the algorithm GREEDY-DIST of Section III-B and higher bounds on lengths of traced curves using Equation (2), with the factors R^i_k computed as described in Section IV-C. Our implementation of GREEDY-DIST uses functions from the PQP library[1] to construct RSS hierarchies and compute distances between pairs of RSSs. We tried several algorithms other than GREEDY-DIST to compute exact and approximate distances, but GREEDY-DIST gave the best results overall.

Since ADAPTIVE-BISECTION bisects (sub-)segments in the same way for all pairs of objects, we share results of forward kinematics computations across pairs, by caching the rigid-body transforms defining the positions and orientations of robot links at every configuration considered by the checker. In environments with manipulator arms, such caching avoids many redundant computations.

Pairs of objects that cannot possibly collide (due to constraints on their motion) are identified in a pre-processing phase by bounding the swept volumes of all objects by spheres (as in Section IV-C) and then testing these spheres for intersection. If two swept spheres are disjoint, then the underlying object pair is never inserted in the priority queue Q processed by ADAPTIVE-BISECTION. Likewise, pairs of static objects (e.g., robot bases or obstacles) are not tested for intersection.

We tested ADAPTIVE-BISECTION with various values of δ.

[1] `http://www.cs.unc.edu/~geom/SSV/`

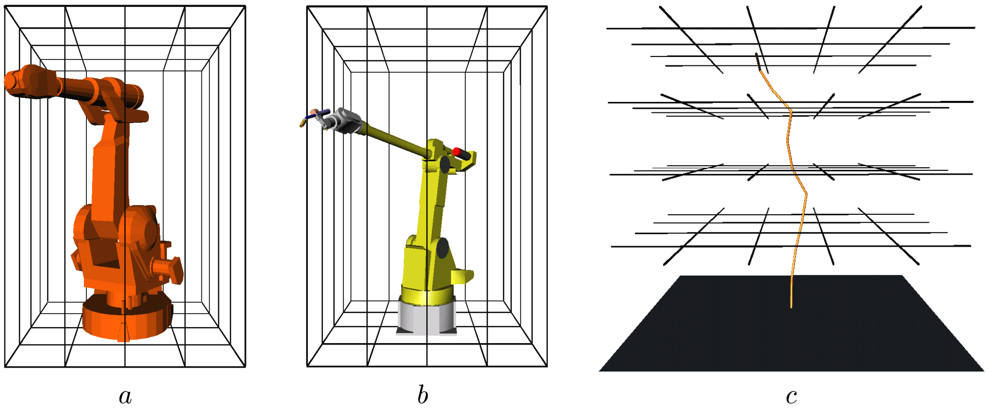


Fig. 11. Examples with thin obstacles. a: IRB 2400 robot (2,991 triangles), cage (432 triangles). b: F 200 robot with arc welding gun (2,502 triangles), cage (432 triangles). c: 20-DOF arm (320 triangles), obstacle lattice (384 triangles)

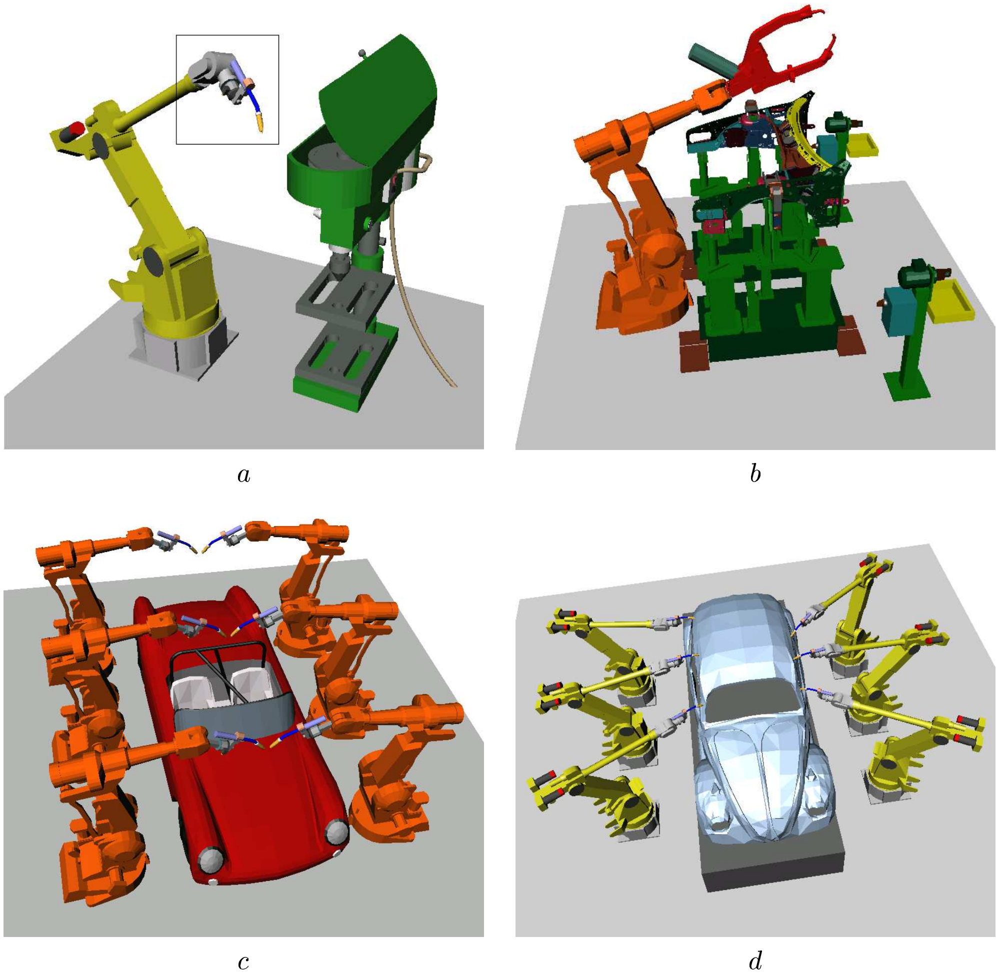


Fig. 12. a: F 200 robot with arc welding gun (2,502 triangles), machine tool (34,171 triangles). b: IRB 2400 robot (5,450 triangles) in workshop (74,681 triangles). c: Six IRB 2400 robots (6 × 3,594 triangles), car body (19,668 triangles). d: Six F 200 robots (6 × 2,502 triangles), car body (4,016 triangles)

Fig.	FIXED-RES-BISECTION $\varepsilon = 0$			ADAPTIVE-BISECTION $\delta = 0$			$\delta = 0.005$		
	t (ms)	BV	Tri	t (ms)	BV	Tri	t (ms)	BV	Tri
$11a$	0.44	375	3.1	0.33	197	3.0	0.29	169	2.6
$11b$	0.46	380	3.5	0.43	324	3.9	0.38	255	3.2
$12a$	0.67	522	18.7	0.40	501	18.0	0.37	420	21.8
$12b$	1.21	1,035	10.1	0.57	571	10.8	0.40	440	13.4
$12c$	3.11	2,866	13.1	2.44	686	12.1	2.23	610	12.7
$12d$	1.14	999	16.3	1.93	609	19.5	1.71	528	21.6

TABLE III

COMPARISON OF FIXED-RES-BISECTION WITH $\varepsilon = 0$ AND ADAPTIVE-BISECTION WITH $\delta = 0$ AND $\delta = 0.005$

Fig.	$N = 50$ ε	t (s)	$N = 100$ ε	t (s)
$11a$	0.00402	4.13	0.00232	5.15
$11b$	0.00078	12.26	0.00078	13.87
$12a$	0.00833	0.17	0.00833	0.17
$12b$	0.00162	7.52	0.00038	16.45
$12c$	0.00694	2.72	0.00279	3.82
$12d$	0.01200	0.67	0.00833	0.92

TABLE IV

AVERAGE RUNNING TIMES OF SBL ON FIVE DISTINCT PATH-PLANNING PROBLEMS FOR $N = 50$ AND $N = 100$ CONSECUTIVE CORRECT RUNS

δ	.01	.005	.001	.0005	.0001	.00005	.00001	.0
$11a$	5.59	5.04	5.60	7.95	11.17	13.38	17.07	24.85
$11b$	11.78	12.37	11.47	13.17	16.03	16.99	20.59	27.54
$12a$	0.16	0.22	0.42	0.49	1.32	1.87	3.48	21.48
$12b$	6.71	7.33	9.55	12.63	27.53	31.91	67.52	251.56
$12c$	2.00	1.59	1.80	1.76	2.95	2.89	3.83	5.54
$12d$	0.43	0.65	0.71	0.86	2.05	2.42	3.23	10.90

TABLE V

AVERAGE RUNNING TIMES OF A-SBL FOR SUCCESSIVE VALUES OF δ

To relate these values to the size of the robots and obstacles in each environment, we give the lengths L_{ref} of key robot links in Table II. For the linkages in Figures $1a$ and $11c$, L_{ref} is the length of any individual link. For the IRB 2400 and F 200 robots, it is the length of the robot's forearm as shown in Figure 10 (Different values for the same reference link across examples are due to different robot scaling factors.)

We compared ADAPTIVE-BISECTION to a classical fixed-resolution checker, which we call FIXED-RES-BISECTION. This checker follows exactly the bisection algorithm given in [24]: Given a straight path segment with collision-free endpoints, it bisects this segment and its sub-segments until either the mid-point of a (sub-)segment is found to collide, or all sub-segments are shorter than some given ε (in the Euclidean metric of c-space). Longer sub-segments are bisected before shorter ones, hence in a breadth-first ordering, in order to find collisions quicker (as longer sub-segments are more likely to collide). For each tested mid-point, FIXED-RES-BISECTION checks all pairs of objects for collision, except those identified as never colliding in the pre-processing phase. Like ADAPTIVE-BISECTION, FIXED-RES-BISECTION can be used to test individual segments or multi-segment paths.

FIXED-RES-BISECTION uses hierarchies of OBBs to test configurations. The construction of the OBB hierarchies and the collision test of two OBB hierarchies are carried out by functions of the PQP library.

B. Random colliding segments

In this series of experiments, we compared the performance of the adaptive and fixed-resolution checkers on *colliding* path segments. In each of the six environments of Figures $11a$-b and $12a$-d, we generated 1,000 colliding segments by sampling pairs of collision-free endpoints uniformly at random, and retaining those segments which ADAPTIVE-BISECTION (with $\delta = 0$) found to be colliding. We then tested each segment using: (1) FIXED-RES-BISECTION with $\varepsilon = 0$, (2) ADAPTIVE-BISECTION with $\delta = 0$, and (3) ADAPTIVE-BISECTION with $\delta = 0.005$. When a segment is known to be colliding, FIXED-RES-BISECTION with $\varepsilon = 0$ is guaranteed to detect a collision in finite time.

Table III shows data gathered during these tests. For each checker, we indicate the average running time and the average number of BV and triangle pairs tested per segment. ADAPTIVE-BISECTION (with $\delta = 0$ and 0.005) tests significantly fewer pairs of BVs than FIXED-RES-BISECTION. This gain derives from both the use of Inequality (1), which results in bisecting path segments at a coarser average resolution, and the better ordering of the priority queue.

The most significant savings were achieved in the environment of Figure $12c$ where collisions between thin end-effectors are quite difficult to detect. The environment of Figure $12d$ is similar, but collisions between a robot and the car body are easier to detect in this case, which results in smaller differences between the checkers.

The times shown in the table indicate that, except for the environment of Figure $12d$, ADAPTIVE-BISECTION with $\delta = 0$ is faster on average than FIXED-RES-BISECTION, despite the fact that computing distances between RSSs is not as fast as testing OBBs for overlap.

The results with $\delta = 0.005$ show a small speed-up over those obtained with $\delta = 0$ because with $\delta > 0$, some segments are rejected as soon as a pair of objects is found to come closer

Fig.	FIXED-RES-BISECTION t (ms)	BV	Tri	ADAPTIVE-BISECTION t (ms)	BV	Tri
11b	14.6	15,060	19.8	11.3	9,239	200.8
12a	5.1	4,032	17.9	3.9	3,834	158.6

TABLE VI

COMPARISON OF FIXED-RES-BISECTION WITH $\varepsilon = 0.006$ AND ADAPTIVE-BISECTION WITH $\delta = 0.005$

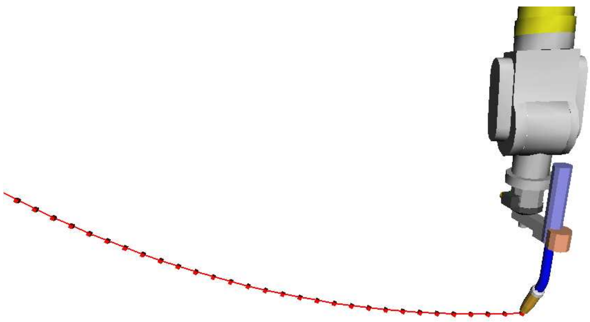


Fig. 13. Curve traced by the tip of the end-effector of the robot in Figure 11b for a typical path segment. The black dots along the curve correspond to the configurations tested by FIXED-RES-BISECTION, with $\varepsilon = 0.006$. Each dot is a cube whose side length is equal to 0.01 unit

than δ.

C. Random collision-free segments

The above experiment favors the fixed-resolution checker, since we could set $\varepsilon = 0$. When segments may not be colliding, this is not possible. To illustrate the respective effects of ε and δ, we used FIXED-RES-BISECTION and ADAPTIVE-BISECTION to test *collision-free* path segments. For the results presented below, we set $\varepsilon = 0.006$ and $\delta = 0.005$.

Like in the previous experiment, in each considered environment we generated 1,000 random collision-free segments, by picking end-points at random and using the adaptive checker (with $\delta = 0.005$) to test them. Table VI shows the same quantities as Table III measured in the environments of Figures 11b and 12a. The value of ε was chosen so that the fixed-resolution checker took only slightly longer than the adaptive checker.

Figure 13 shows, as a thin line, the curve traced by the tip of the end-effector when the robot of Figure 11b moves along a typical straight path segment in c-space. The dots along this curve correspond to the configurations tested by FIXED-RES-BISECTION with $\varepsilon = 0.006$. In reality, each dot is a cube whose side length is 0.01 units (in the workspace metric), hence twice the value selected for δ. To make FIXED-RES-BISECTION sample the curve at resolution δ, we would have to set ε so small that the running time of FIXED-RES-BISECTION would be considerably longer.

D. PRM planning

In this section we compare the dynamic and fixed-resolution checkers when they are integrated in the PRM planner described in [24], called SBL. We briefly recall how this planner works.

SBL constructs a roadmap made of two trees of milestones (collision-free configurations sampled at random) rooted at the two query configurations input by the user. At each iteration, the planner performs the following operations:

(1) It picks a new milestone m' at random in a neighborhood of an existing milestone m and installs m' as a child of m *without* testing the straight path segment between them for collision.

(2) It selects two milestones, one from each tree, and connects them by a candidate straight path segment. This connection creates a multi-segment path between the two query configurations, but only the vertices of this path (milestones) have so far been tested for collision.

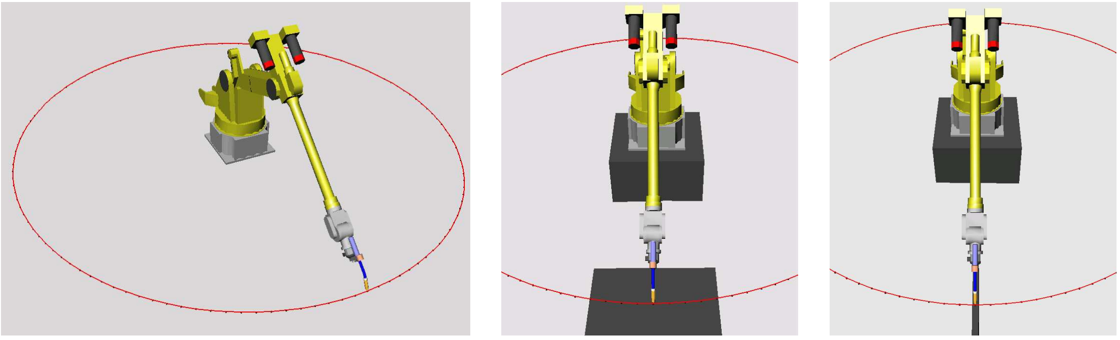
(3) It checks this entire path for collision. If no collision is detected, it returns the path. Otherwise it removes the segment found to be colliding from the roadmap, caches the collision-checking work done along other segments (to avoid repeating it if any of these segments is part of a candidate path at a later iteration), and proceeds to the next iteration.

Hence, SBL tests segments between milestones only when this is absolutely needed. This so-called "lazy collision checking" strategy has been shown to significantly reduce the amount of work spent on collision-free segments that are not on the final solution path.

The implementation of SBL described in [24] uses FIXED-RES-BISECTION to test multi-segment paths. Hence, it requires setting the value of ε for each new environment, and is never guaranteed to return a collision-free path. We have created a new version of SBL, called A-SBL, by replacing the fixed-resolution checker by ADAPTIVE-BISECTION. Like SBL, A-SBL also records the collision-checking work done along path segments that have not been fully tested (by maintaining a distinct priority queue for each segment, as described in Section II-D).

We ran SBL and A-SBL on several path-planning problems, each defined by a pair of query configurations in a given environment. For each problem, we performed several series of runs of SBL with decreasing values of ε. During each series, ε kept the same value, but a different seed of the random number generator was used at each run to construct a different roadmap, hence a different path. We tested each path returned by SBL using ADAPTIVE-BISECTION with $\delta = 0$. Whenever a collision was detected in a path, we terminated the series, cut the value of ε by 1.2, and started another series. Starting with a rather large value of ε, we retained the first series of 50 (respectively 100) consecutive runs of SBL that did not return a single colliding path.

Tables IV and V list results obtained for six planning problems in the environments of Figures 11a-b and 12a-d. For each problem, Table IV gives the first value of ε that produced N consecutive correct runs, for $N = 50$ and 100, and the average running time of SBL over these N runs. Note that ε is much smaller for $N = 100$ in all cases, except two (11b and 12a), meaning that the values of ε obtained for $N = 50$ are usually not sufficient to guarantee the reliability of SBL.



a b c

Fig. 14. Three bad-case examples for ADAPTIVE-BISECTION

Fig.	Random walk #segments	Random walk t (s)	Smoothing K	Smoothing t (s)	Smoothing σ
1a	40,439	82	1,000	152	0.06
11c [1 linkage]	4,208	5.9	500	16	0.11
11c [9 linkages]	47,744	823	6,000	5,952	0.25

TABLE VII

RESULTS FOR RANDOM WALKS AND PATH SMOOTHING

Table V lists the average running times (over 100 independent runs) of A-SBL for decreasing values of δ. As expected, the average running time of A-SBL increases when δ goes to 0. However, the increase is moderate until δ reaches very small values, and even then remains relatively small for most problems. Note that in two examples (11a and 12c), for relatively large δ, the running time even decreases slightly when the value of δ goes down. In these cases, planning may be slightly harder for larger values of δ as more collision-free paths are rejected by ADAPTIVE-BISECTION.

A-SBL has the considerable advantage over SBL that it is guaranteed to never return a colliding path. This is not the case of SBL, even for the values of ε allowing 100 consecutive correct runs. In most examples, selecting a value of N greater than 100 would have led to smaller values of ε.

E. Path Smoothing

We conducted the following experiment in the environment of Figure 1a. First, starting at a configuration where both linkages stand straight up through the three rings, we performed a random walk of collision-free steps until both linkages lie entirely below the rings, hence were retracted from the rings. ADAPTIVE-BISECTION was used to test each attempted step (straight segment) of the random walk. Next, we smoothed the obtained path by performing K times the following operations: Pick a point $\boldsymbol{q}$ at random in the path; let $\boldsymbol{q}_{le}$ ($\boldsymbol{q}_{ri}$) be the midpoint along the path between the first (last) configuration of the path and $\boldsymbol{q}$. If the straight line segment between $\boldsymbol{q}_{le}$ and $\boldsymbol{q}_{ri}$ tests collision-free, shortcut the path section between them by this segment, otherwise reset $\boldsymbol{q}_{le}$ ($\boldsymbol{q}_{ri}$) to the midpoint along the path between itself and $\boldsymbol{q}$, and recurse. Again, ADAPTIVE-BISECTION was used to test each attempted shortcut. We did the same experiment in the environment of Figure 11c, first with a single linkage, then with 9 identical linkages, each

ζ	δ	All links checked BV	All links checked Tri	Only end-effector checked BV	Only end-effector checked Tri
0.1	0.1	11,232	0	8,789	0
0.01	0.01	62,866	0	62,727	0
0.001	0.001	990,766	0	990,595	0
0.0001	0.0001	7.91e+06	0	7.91e+06	0
0.1	0.0001	258	0	127	0
0.01	0.0001	753,939	0	753,800	0
0.001	0.0001	1.12e+06	0	1.12e+06	0
0.0001	0.0001	7.91e+06	0	7.91e+06	0

TABLE VIII

NUMBER OF BV/TRIANGLE PAIRS TESTED IN FIGURE 14a

ζ	δ	All links checked BV	All links checked Tri	Only end-effector checked BV	Only end-effector checked Tri
0.1	0.1	1,421	105	1,397	105
0.01	0.01	5,664	101	5,646	101
0.001	0.001	73,018	122	72,998	122
0.0001	0.0001	578,220	3,358	578,200	3,358
0.1	0.0001	33	0	17	0
0.01	0.0001	23,801	0	23,783	0
0.001	0.0001	67,165	0	67,145	0
0.0001	0.0001	578,220	3,358	578,200	3,358

TABLE IX

NUMBER OF BV/TRIANGLE PAIRS TESTED IN FIGURE 14b

ζ	δ	All links checked BV	All links checked Tri	Only end-effector checked BV	Only end-effector checked Tri
0.1	0.1	653	5	634	5
0.01	0.01	3,059	51	3,045	51
0.001	0.001	27,685	87	27,671	87
0.0001	0.0001	202,870	2,153	202,856	2,153
0.1	0.0001	27	0	15	0
0.01	0.0001	315	0	301	0
0.001	0.0001	15,426	0	15,412	0
0.0001	0.0001	202,870	2,153	202,856	2,153

TABLE X

NUMBER OF BV/TRIANGLE PAIRS TESTED IN FIGURE 14c

threaded through a different set of grid holes in their initial configuration.

The goal of these experiments was to demonstrate that ADAPTIVE-BISECTION enables the implementation of a brute-force planning scheme requiring many collision tests between very thin objects. These experiments would be quasi-impossible using FIXED-RES-BISECTION, as we would then have to chose ε extremely small.

Table VII shows data obtained for a typical run of each experiment. For the random walk, it gives the number of segments tested and the computation time of the walk. For path smoothing, it lists the value of K, the running time, and the ratio σ of the final length of the path by its initial length. Note that each of the K iterations usually results in many segment tests, due to the recursive attempts to shortcut path sections.

F. Bad-case scenarios

A very unfavorable case for ADAPTIVE-BISECTION is when, along a path segment in c-space, two or more objects in relative motion stay at very close distance from each other, without coming closer than the threshold δ. To study the behavior of ADAPTIVE-BISECTION in such cases, we constructed three examples shown in Figure 14. In all three examples, the robot (2,502 triangles) moves a full 2π rotation around its first joint, with all other joints remaining fixed. This motion defines a long straight segment in c-space. In Figure 14a, the robot is placed directly on the floor, a flat surface tessellated by 8,000 triangles. During the motion, the tip of the end-effector (1,080 triangles) traces a circle at a constant clearance above the floor that is slightly bigger than a given $\zeta > 0$. The example in Figure 14b is similar, but the robot is now placed on top of a box, and during the rotation the end-effector grazes the top face of another box, again with a constant clearance slightly bigger than $\zeta > 0$. Hence, the end-effector now comes close to an obstacle only during a fraction of the rotation. Each face of the box is modeled with 8,000 triangles. The example in Figure 14c is almost identical: the box is narrower, but its faces still contain 8,000 triangles, each.

Tables VIII–X show the numbers of pairs of BVs and triangles tested in each of the above three examples, for different combinations of values of δ and ζ. In each table the numbers are reported for all robot links and for the end-effector only.

The numbers of BV/triangle pairs tested by the checker depend on both ζ and δ. However, in each example, the tables show that links other than the end-effector barely influence these numbers. This is due to the fact that the checker tests each pair of objects independently of the others, starting with the object pairs that are the most likely to collide. Links other than the end-effector, which are rather far away from any obstacle, are quickly eliminated (as not colliding), if ever tested.

Another observation is that the checker focuses on the critical portion of the path segment. Indeed, the numbers of BV/triangle tests decrease significantly from the first example to the third, although the three path segments have equal length and the obstacle's top surface has the same number of triangles in each case. This comes from the fact that the use of Inequality (1) allows the checker to sample the path segment more coarsely when obstacles are further away from the end-effector.

As we expected, this experiment shows that the performance of ADAPTIVE-BISECTION degrades when two objects in relative motion remain very close apart and both δ and ζ approach zero. Nevertheless, even for very small values of δ and ζ, the checker remains reasonably efficient as long as the small clearance occurs over short path sections. In PRM path planning, where segment endpoints are selected at random, this condition is likely to be verified. But in some applications, such as machining and assembly, small clearances over long segments may occur more frequently. Then, methods based on swept-volume computation or trajectory parameterization might be more appropriate, possibly in combination with adaptive bisection.

VI. Conclusion

This paper describes a new dynamic collision checker to test straight line segments in c-space or collections of such segments, including continuous multi-segment paths. This checker is general, but particularly suited for PRM planners applied to manipulator arms and/or multi-robot systems. Its main advantage over the commonly used fixed-resolution approach is to never miss a collision. In addition, it eliminates the need for experimentally determining a suitable value of the c-space resolution parameter ε and its running time compares favorably to that of a fixed-resolution checker.

Reliability and efficiency are obtained by dynamically adjusting the local resolution at which configurations along a path are tested by relating the distances between objects in the workspace to the maximum lengths of the paths traced out by points on these objects. This idea is combined with other techniques, including:
- a greedy distance computation algorithm that is nearly as efficient as a pure collision checker,
- a fast technique for bounding lengths of paths traced out in workspace,
- a heuristics for ordering collision tests to reveal collisions as quickly as possible.

Relatively simple extensions are possible. For example, one could guarantee that no two objects come closer than some minimal distance $\varrho > 0$ along a tested path segment, by using the following variant of Inequality (1):

$$\lambda_i^r(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j^r(\boldsymbol{q}_a, \boldsymbol{q}_b) < \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b) - 2\varrho.$$

Here, $\lambda_i^r(\boldsymbol{q}_a, \boldsymbol{q}_b)$ (similarly $\lambda_j^r(\boldsymbol{q}_a, \boldsymbol{q}_b)$) bounds the motions of $\mathcal{A}_i^r$ which is defined as the Minkowski sum of $\mathcal{A}_i$ and a sphere of radius $r = \varrho/2$.

Dealing with non-linear path segments in c-space is also possible, as it only requires changing the upper bounds on lengths of traced curves. This can be done as discussed at the end of Section IV-B.

We believe that our checker could still gain in efficiency by computing tighter bounds on lengths of curves traced out by points of moving objects. For example, instead of computing factors R_k^i that are valid across the entire c-space, we could partition the c-space of each linkage into a coarse grid of cells, and compute smaller factors over each cell. Another way of improving the efficiency of the checker could be to insert elements based on pairs of bounding volumes in the priority queue, instead of elements based on pairs of objects. In this way, as a path segment gets bisected at a finer resolution, not only the number of remaining object pairs would drop, but in addition smaller and smaller subsets of the remaining objects would be considered.

The new checker has certain limitations, however. In some applications, one may want to determine the first collision configuration when moving from one end of a path segment to the other. Then, bisection is not the best approach. However, even in this case, a variant of Inequality (1) could be used to decompose the segment into a series of safe segments. Another limitation is that the checker loses efficiency when two moving objects come arbitrarily close and/or stay very close along a long section of a path segment. To bound the running time we have introduced a clearance parameter $\delta > 0$. Though δ can be set quite small in practice, it may be undesirable for some applications (e.g., mechanical assembly). An alternative discussed in Section II-F is to locally switch to another dynamic collision-checking method, for instance, a swept-volume or a trajectory parameterization method. Such a method is then made practical by the fact that the potential collision can only occur on restricted sections of the segment and between restricted portions of the objects.

ACKNOWLEDGMENT

This work was partially funded by a gift from General Motors and a grant from ABB.

REFERENCES

[1] S. Cameron, "A study of the clash detection problem in robotics," in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, 1985, pp. 488–493.

[2] P. Jiménez, F. Thomas, and C. Torras, "3D collision detection: A survey," *Computers and Graphics*, vol. 25, no. 2, pp. 269–285, 2001.

[3] M. C. Lin and S. Gottschalk, "Collision detection between geometric models: a survey," in *Proc. IMA Conf. on Mathematics of Surfaces*, vol. 1, San Diego (CA), 1998, pp. 602–608.

[4] J. Basch, L. Guibas, and J. Hershberger, "Data structures for mobile data," in *Proc. SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[5] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi, "I-COLLIDE: An interactive and exact collision detection system for large-scale environments," in *Proc. Sym. on Interactive 3D Graphics*, 1995, pp. 189–196, 218.

[6] M. Lin, D. Manocha, J. Cohen, and S. Gottschalk, "Collision detection: Algorithms and applications," in *Proc. Algorithms for Robotic Motion and Manipulation: WAFR*, J. P. Laumond and M. Overmars, Eds. A. K. Peters, 1996, pp. 129–142.

[7] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation," in *IEEE Int. Conf. on Robotics and Automation*, 1991, pp. 1008–1014.

[8] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Trans. on Graphics*, vol. 17, no. 3, pp. 177–208, July 1998.

[9] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," *Computer Graphics*, vol. 30, no. Annual Conf. Series, pp. 171–180, 1996.

[10] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Tr. on Graphics*, vol. 15, no. 3, pp. 179–210, 1996.

[11] J. T. Klosowski, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs," *IEEE Trans. Visual. Comput. Graphics*, vol. 4, no. 1, pp. 21–36, 1998.

[12] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," in *Proc. IEEE Conf. on Robotics and Automation*, 2000.

[13] S. Quinlan, "Efficient distance computation between non-convex objects," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1994, pp. 3324–3329.

[14] G. Van der Bergen, "Efficient collision detection of complex deformable models using AABB trees," *J. of Graphic Tools*, vol. 2, no. 4, pp. 1–13, 1997.

[15] S. A. Cameron, "Collision detection by four-dimensional intersection testing," *IEEE Trans. Automat. Contr.*, vol. 6, pp. 291–302, June 1990.

[16] A. Foisy and V. Hayward, "A safe swept volume method for collision detection," in *Proc. The Sixth Int. Symp. of Robotics Research*, Pittsburgh (PE), Oct. 1993, pp. 61–68.

[17] J. F. Canny, "Collision detection for moving polyhedra," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, no. 2, pp. 200–209, Mar. 1986.

[18] A. Schweikard, "Polynomial time collision detection for manipulator paths specified by joint motions," *IEEE Trans. Automat. Contr.*, vol. 7, no. 6, pp. 865–870, 1991.

[19] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Proc. Workshop on Algorithmic Foundations of Robotics*, Mar. 1998, pp. 155–168.

[20] J. Barraquand, L. Kavraki, J. C. Latombe, T. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for path planning," *Int. J. of Robotics Research*, vol. 16, no. 6, pp. 759–774, 1996.

[21] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *Proc. Int. Conf. on Robotics and Automation*, 2000, pp. 521–528.

[22] D. Hsu, J. C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *Int. J. of Computational Geometry and Applications*, vol. 9, no. 4-5, pp. 495–512, 1999.

[23] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Automat. Contr.*, vol. 12, no. 4, pp. 566–580, 1996.

[24] G. Sanchez and J. C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Proc. Int. Symp. on Robotics Research*, Lorne, Victoria, Australia, 2001.

[25] L. Dale, G. Song, and N. Amato, "Faster, more effective connection for probabilistic roadmaps," Department of Computer Science, Texas A&M University, Tech. Rep. TR00-005, 20, 2000.

[26] B. Baginski, "Efficient dynamic collision detection using expanded geometry models," in *Proc. IEEE/RSJ/GI Int. Conf. on Intelligent Robots and Systems*, 1997, pp. 1714–1719.

[27] C. Nielsen and L. E. Kavraki, "A two-level fuzzy PRM for manipulation planning," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Japan, 2000.

[28] T. Li and J. Chen, "Incremental 3D collision detection with hierarchical data structures," in *Proc. ACM Symp. on Virtual Reality Software and Technology*, Taipei, Taiwan, 1998, pp. 139–144.

[29] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New Results and New Trends in Computer Science*, ser. LNCS 555, H. Maurer, Ed. Springer Verlag, 1991, pp. 359–370.