Exact Collision Checking of Robot Paths

Fabian Schwarzer* Mitul Saha* Jean-Claude Latombe*

Abstract

This paper describes a new efficient collision checker to test single straight-line segments in c-space, sequences of such segments, or more complex paths. This checker is particularly suited for probabilistic roadmap (PRM) planners applied to manipulator arms and multi-robot systems. Such planners spend most of their time checking local paths between randomly sampled configurations for collision. While commonly used approaches test intermediate configurations on a segment at a prespecified resolution, the checker presented in this paper is exact, i.e., it cannot fail to find an existing collision, even when some robot links and obstacles are very thin. Its efficiency relies on its core algorithm, which dynamically adjusts the required resolution by relating the distances between objects in the workspace to the maximum lengths of the paths traced out by points on these objects. The checker's efficiency is further increased by several additional techniques presented in this paper, which adequately approximate distances between objects and lengths of travelled paths in workspace, and order collision tests to reveal collisions as early as possible. The new checker has been extensively tested, first on segments randomly generated in c-space, next as part of an existing PRM planner, and finally as part of a path smoother/optimizer. These experiments show that the checker is faster than a resolution-based approach (with suitable resolution), with the enormous advantage that it never returns an incorrect answer. The checker also admits a number of straightforward extensions. For example, it can monitor a minimum workspace distance between each robot link and other objects (e.g., obstacles, links of other robots).

1 Introduction

Collision checking is a fundamental operation in robot motion planning, graphic animation and physical simulation [5, 12, 18]. While *static* collision checking amounts to testing a single configuration for spatial overlaps, *dynamic* collision checking requires showing that all configurations on a continuous path in c-space are collision-free. Three major families of methods can be used for dynamic collision checking: feature-tracking, bounding-volume, and swept-volume methods.

*Dept. of Computer Science, Stanford University, Stanford, CA 94305. E-mail: schwarzf@stanford.edu, mitul@stanford.edu, latombe@cs.stanford.edu

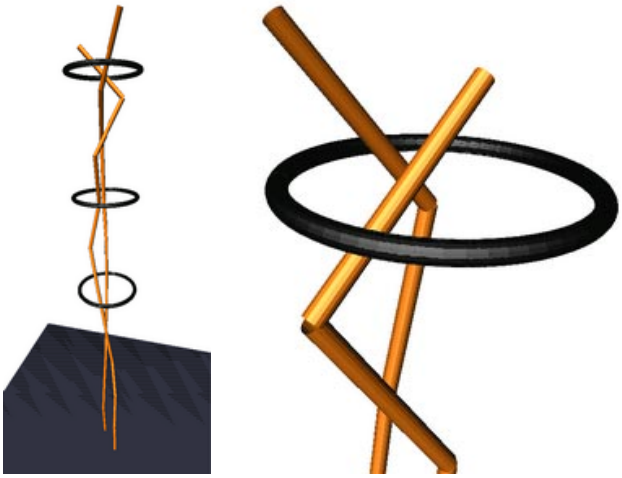


Figure 1: Collisions are easily missed in this example with two skinny 20-DOF arms (320 triangles each) which have to retract from three fixed thin tori (6300 triangles each).

- *Feature-tracking* methods rely on the following coherence assumption: the pair of closest features between two objects in relative motion changes only at discrete points of time and, when it changes, computing the new pair from the old one can be done efficiently [3, 7, 16, 17, 19]. However, to be practical, these methods require each object to be made of few convex components. Furthermore, they test a path by small increments, from one end to the other, which is not always the fastest way to detect if the path collides. The coherence assumption is particularly problematic for links of a kinematic chain and may require tiny steps, especially for the links closer to the end of the chain.

- *Bounding-volume* (BV) methods precompute, for each object, a hierarchy of BVs (e.g., spheres, boxes) that approximate the geometry of the object at varying resolutions [10, 14, 21, 23]. BVs then speed up collision checking by making it possible to quickly discard large portions of objects that cannot possibly collide. These methods have been successfully applied to objects with surfaces described by several 100,000 triangles, and more. But they are fundamentally static methods. To test a path for collision, the common approach is to test intermediate configurations along the path until a collision is found or any two successive configurations are less than some prespecified ε apart (the configura-

tions are usually obtained by recursively bisecting the path). In the second case, the path is declared collision-free (despite the fact that this answer has a slight chance of being incorrect). Choosing ε is difficult, especially in scenarios with articulated arms and/or multiple robots. If ε is small, collision checking is inefficient because many configurations on the path will be tested. If ε is large, the risk of missing collisions is significant.

- *Swept-volume and space-time volume intersection* methods consist of computing the volumes swept by the objects in the workspace, possibly with a time dimension added, and testing these volumes for overlap [6, 9]. However, exact computation of such volumes is time consuming, especially, when objects undergo rotations and are geometrically complex. Moreover, the overlap test can no longer be speeded up by exploiting precomputed data structures (such as BV hierarchies). Another problem is that, unless either the time dimension is added or relative motions are considered explicitly, swept volumes for pairs of moving objects may overlap although the objects themselves do not collide.

Hence, dynamic collision checking remains a major bottleneck in many applications. In particular, probabilistic roadmap (PRM) planners heavily rely on the availability of efficient dynamic checkers to test "local paths" between randomly sampled configurations for collision [1, 2, 4, 11, 13, 22]. Most such planners use a static BV method to test intermediate configurations along the local paths at some resolution ε. Choosing ε involves several trial-and-error experiments, which must be repeated for each new type of robot and environment. Large values of ε are acceptable when both robots and obstacles are fat. But when objects are thin, collisions are easy to miss. An example is shown in Fig. 1, which contains two skinny linkages, with 20 revolute joints each, and thin obstacles, all with significant geometric complexity. In this example, even small changes in the joint angles can make a linkage jump over an obstacle or over the other linkage. The value of ε needed to reliably detect such collisions must be very small, resulting in inefficient PRM planners. Further examples illustrating this problem are shown in Fig. 2 and Fig. 6.

This paper describes a new efficient dynamic collision checker to test single straight-line segments in c-space, sequences of such segments, or more complex paths. This checker is particularly suited for PRM planners applied to manipulator arms and multi-robot systems. It is exact in the sense that it always returns the correct answer. In particular, it cannot fail to find an existing collision, even when some robot links and obstacles are very thin. Its exactness 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. While the basic idea of adjusting the resolution has been described before (e.g., in [2, 5]), our approach extends it in several ways by techniques for adequately approximating distances between complex objects and lengths of paths traced out in workspace, and for ordering collision tests (along single segments or entire paths) to reveal collisions as quickly as possible.

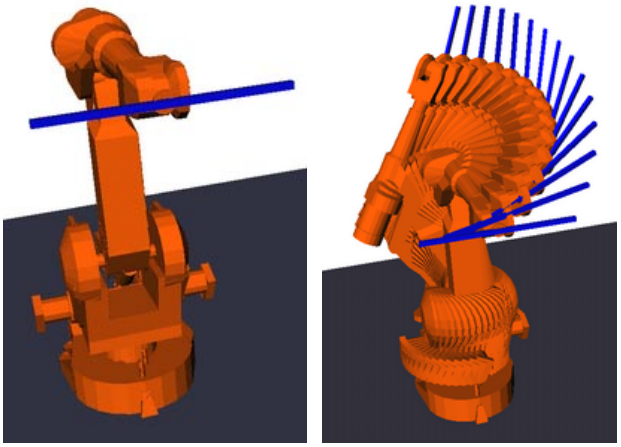


Figure 2: IRB 2400 robot carrying a thin rod.

Applications that will benefit from our new segment checker, both in terms of speed and accuracy, are e.g., PRM planners and path smoothing algorithms. We have extensively tested the checker, first on segments randomly generated in c-space for multiple robots in different environments, next as part of an existing PRM planner [22], and finally as part of a path smoother/optimizer. These experiments show that our checker is faster than previous resolution-based checkers (with suitable resolution ε), with the enormous advantage that it never returns an incorrect answer. Our techniques also admit a number of straightforward extensions. For example, they can easily be adapted to monitor a minimum workspace distance between each robot link and other objects (e.g., obstacles, links of other robots).

The rest of the paper is organized as follows. Section 2 introduces the problem of checking straight line segments in c-space, points out shortcomings of the commonly used approach and presents the idea of using distance information for collision checking. Section 3 generalizes this idea and develops our new adaptive segment checking algorithm. Section 4 reports on our experiments with a lazy PRM planner and a randomized path smoothing algorithm. Section 5 briefly explains how the basic algorithm can be extended to check for minimum clearances along entire segments. Section 6 concludes the paper and points to possible future work.

2 Checking segments in c-space

Our problem can be stated as follows: given two configurations $\boldsymbol{q}$ and $\boldsymbol{q}'$ and the straight line segment between

them

$$[\boldsymbol{q}, \boldsymbol{q}'] = \{\boldsymbol{q}(t) \mid \boldsymbol{q}(t) \equiv t\boldsymbol{q} + (1-t)\boldsymbol{q}', t \in [0,1]\}$$

show that all configurations on this segment are collision-free. A commonly used approach is to test only a finite number of equally spaced intermediate configurations $\boldsymbol{q}_i = \boldsymbol{q}(t_i)$ on this segment for collision. For a given resolution $\varepsilon > 0$, the intermediate configurations are chosen such that each neighboring pair of them is closer than ε according to some given metric $d(\cdot,\cdot)$ in c-space, i.e., $d(\boldsymbol{q}_i, \boldsymbol{q}_{i+1}) < \varepsilon$. The entire segment is then declared to be collision-free if all intermediate configurations have been found to be free.

In the case of a colliding segment, heuristic ordering of the tests of the intermediate configuration can decrease the expected time for finding the collision. For example, the lazy PRM planner in [22] assumes that the intermediate configurations on a segment have different probabilities of being in collision. Given that the endpoints of the segment are free, the midpoint $\boldsymbol{q}(\frac{1}{2})$ has a high prior collision probability and it is thus tested first. Once the midpoint has been found to be free, the segment can be broken into two sub-segments and now their midpoints $\boldsymbol{q}(\frac{1}{4})$ and $\boldsymbol{q}(\frac{3}{4})$ have high collision probabilities and will be tested next. The general strategy corresponds to a breadth-first recursion which terminates when a collision is found or the length of the smallest sub-segment is less than ε (otherwise it would not terminate on collision-free segments).

However, choosing a proper c-space resolution ε is difficult and has to be done for each configuration space anew. For example, in a centralized planner for multiple robots, ε depends not only on the types of the individual robots but also on their number, unless a c-space metric is used that is independent of the dimensionality. In particular, when a Euclidean metric is used, ε has to be decreased with increasing dimensionality. Choosing ε very small will result in many intermediate configurations and thus make the collision checker unnecessarily slow. Choosing it too large might result in missing collisions. Since collision checking is still the major computational bottleneck in PRM planners, we would like to keep the number of discrete intermediate configurations as small as possible while still guaranteeing not to miss any collisions.

The key difficulty here is that even small motions in c-space may cause large motions in workspace. This effect is especially distinctive for manipulator arms where the joints close to the base affect many links along the kinematic chain. We would therefore like to bound the motions of the links in workspace. In fact, any given metric in c-space can be related to distance in workspace [2]: for any robot, a constant $\rho > 0$ can be determined such that no point on the robot traces a curve in workspace that is longer than $\rho\, d(\boldsymbol{q}, \boldsymbol{q}')$ when we interpolate on a straight line from $\boldsymbol{q}$ to $\boldsymbol{q}'$. Note that although the connection in c-space is a straight line, the curves traced by points on the robot are usually non-linear. Further note that ρ depends on both the robot type and the used c-space metric. In Section 3.3, we describe in detail how ρ can be derived in practice.

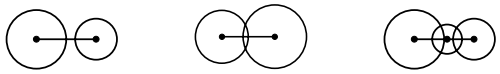


Figure 3: Uncovered and covered (safe) segments.

Assume for the moment that we have a single robot and its ρ-value, and we are only interested in detecting collisions of the robot with static obstacles in the environment. Further assume that we have a function $\eta(\cdot)$ that maps any configuration $\boldsymbol{q}$ to the Euclidean distance (in the workspace) between the robot placed according to $\boldsymbol{q}$ and the obstacles. Then, the entire segment $[\boldsymbol{q}, \boldsymbol{q}']$ is free if:

$$\rho\, d(\boldsymbol{q}, \boldsymbol{q}') < \eta(\boldsymbol{q}) + \eta(\boldsymbol{q}') \qquad (1)$$

To prove this, assume that the robot comes in contact with an obstacle at some configuration $\boldsymbol{q}(t_c)$ on the segment. A point on the robot that hits an obstacle at t_c would have to move at least $\eta(\boldsymbol{q})$ to reach the obstacle at the configuration $\boldsymbol{q}(t_c)$ and then again at least $\eta(\boldsymbol{q}')$ to move from $\boldsymbol{q}(t_c)$ to its position at $\boldsymbol{q}'$. However, this contradicts the assumption that no point on the robot traces a curve longer than $\rho\, d(\boldsymbol{q}, \boldsymbol{q}')$. Note that the opposite is not true: criterion (1) can be violated for a segment that is collision-free. In fact, $\rho\, d(\boldsymbol{q}, \boldsymbol{q}')$ can exceed $\eta(\boldsymbol{q}) + \eta(\boldsymbol{q}')$ without introducing a collision. If criterion (1) is violated, we bisect the segment and compute $\eta(\boldsymbol{q}(\frac{1}{2}))$. If this yields a collision, we stop and report the collision. Otherwise, we test the criterion (1) recursively for the segments $[\boldsymbol{q}, \boldsymbol{q}(\frac{1}{2})]$ and $[\boldsymbol{q}(\frac{1}{2}), \boldsymbol{q}']$. Eventually, we will either find a colliding configuration or all sub-segments will satisfy the collision-free criterion.

Inequality (1) can be illustrated schematically by drawing a segment of length proportional to $\rho\, d(\boldsymbol{q}, \boldsymbol{q}')$ and centering discs with radii proportional to the workspace distances at the endpoints of the segment and at additional bisection points (see Fig. 3). The motion along the segment is collision-free when the segment is covered by such discs.

While the above approach gives an exact algorithm for dynamic collision checking, a number of issues would make it rather inefficient in practice. The next section discusses them and proposes solutions.

3 Generalized adaptive bisection method

In the following, we consider the robot (or the robots) and all obstacles as a single collection of moving and static rigid bodies $\mathcal{A}_1, \ldots, \mathcal{A}_n$. This allows us to consider each pair $(\mathcal{A}_i, \mathcal{A}_j)$ independently of all other pairs. For each body $\mathcal{A}_i$, let $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ denote an upper bound on the length of the curve segment traced by any point on $\mathcal{A}_i$ when the configuration of the system is linearly

interpolated from some configuration $\boldsymbol{q}_a$ to another configuration $\boldsymbol{q}_b$. Clearly, $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) = 0$ if $\mathcal{A}_i$ is an obstacle. Given the discussion in the previous section, we could define $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) = \rho\ d(\boldsymbol{q}_a, \boldsymbol{q}_b)$ for each moving body $\mathcal{A}_i$. However, in the case of manipulator arms, links closer to the base typically move less than links closer to the end-effector, and thus using the same ρ for all links would give poor bounds for some links. Instead, we could determine an individual ρ_i for each link and use $\rho_i d(\boldsymbol{q}_a, \boldsymbol{q}_b)$, but this may still result in a rather loose bound, depending on the chosen metric $d(\cdot, \cdot)$. Section 3.3 describes in detail how a better bound $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ can be derived in practice.

Given $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ for all rigid bodies in the system, we can now derive from inequality (1) a similar certificate for an arbitrary pair $(\mathcal{A}_i, \mathcal{A}_j)$:

$$\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) < \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b) \qquad (2)$$

Here, $\eta_{ij}(\boldsymbol{q}_a)$ and $\eta_{ij}(\boldsymbol{q}_b)$ are the workspace distances (or lower bounds for them) between bodies $\mathcal{A}_i$ and $\mathcal{A}_j$ at configurations $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, respectively.

First, notice that (2) includes the case in which one of the two bodies is a static obstacle. To see this, assume that $\mathcal{A}_j$ is static and thus set $\lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b) = 0$. In fact, in this case, the entire argumentation from (1) applies, when we notice that $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ corresponds to $\rho\ d(\boldsymbol{q}_a, \boldsymbol{q}_b)$.

Now let both $\mathcal{A}_i$ and $\mathcal{A}_j$ move, but consider the motion of $\mathcal{A}_i$ in $\mathcal{A}_j$'s local frame. This lets us treat $\mathcal{A}_j$ as a static object. Replacing $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$ by an upper bound on the lengths of the curves traced by the points on $\mathcal{A}_i$ in $\mathcal{A}_j$'s frame, denoted by $\lambda_{ij}(\boldsymbol{q}_a, \boldsymbol{q}_b)$, leads us back to the above case where $\mathcal{A}_j$ is a static obstacle.

However, computing the bounds $\lambda_{ij}(\boldsymbol{q}_a, \boldsymbol{q}_b)$ explicitly for each pair of bodies may be expensive. Instead, we use the fact that $L_{ij}(\boldsymbol{q}_a, \boldsymbol{q}_b) \leq L_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + L_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$, where L denotes the exact curve lengths. (The proof directly follows by integration after using the triangle inequality for the absolute and relative velocities). We can thus use $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j(\boldsymbol{q}_a, \boldsymbol{q}_b)$ instead of $L_{ij}(\boldsymbol{q}_a, \boldsymbol{q}_b)$ in (2). In some cases, (2) may be rather conservative (e.g., consider two synchronously moving bodies).

3.1 Core algorithm

The new segment checking algorithm uses the certificate (2) to decide whether a segment is collision-free or whether it has to be further examined.

It is important to note that (2) applies to a single pair of bodies. We can therefore check each pair of objects independently of the other pairs. For example, links closer to the base of a manipulator usually require fewer bisections of the segment than links closer to the tip. Furthermore, the pairs can be processed in an order that depends on their relative distances and motions and thus decreases the expected time to find possible collisions. For example, pairs whose bodies are close and move a lot will be examined before pairs of slowly moving, far-apart bodies. To realize this, our algorithm maintains a priority queue of *pair-specific (sub-)segments*.

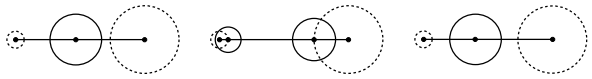


Figure 4: Segment covering strategies.

Each such entry in the queue is of the form $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$. It indicates that the pair $(\mathcal{A}_i, \mathcal{A}_j)$ remains to be tested for collisions along $[\boldsymbol{q}_a, \boldsymbol{q}_b]$. (Section 3.4 describes how we assign priorities to these entries.) Initially, the queue is filled with entries $[\boldsymbol{q}, \boldsymbol{q}']_{ij}$ spanning the entire given segment for each pair of rigid objects which we would like to check. The algorithm then processes the queue by removing the first element, say $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$, and tests if it satisfies (2). If successful, which means that $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$ is covered (see Fig. 3), it continues with the next element in the queue. Otherwise, it checks if the indexed pair $(\mathcal{A}_i, \mathcal{A}_j)$ collides at $\boldsymbol{q}_{mid} = (\boldsymbol{q}_a + \boldsymbol{q}_b)/2$. If this is the case, the whole procedure can stop and report the configuration and the colliding pair of objects. If $\mathcal{A}_i$ and $\mathcal{A}_j$ do not collide at $\boldsymbol{q}_{mid}$, two new entries $[\boldsymbol{q}_a, \boldsymbol{q}_{mid}]_{ij}$ and $[\boldsymbol{q}_{mid}, \boldsymbol{q}_b]_{ij}$, are inserted into the queue and the algorithm continues with the first element in the queue. When the queue is empty, the input segment $[\boldsymbol{q}, \boldsymbol{q}']$ can be reported to be collision-free.

Instead of adding a single bisection configuration $\boldsymbol{q}_{mid}$ exactly at the midpoint of the (sub-)segment (Fig. 4(a)), one could come up with different strategies to refine a yet uncovered (sub-)segment. For example, we could add two new configurations and place them as shown by the solid discs in Fig. 4(b). This would require re-inserting only one new sub-segment into the priority queue, instead of two. However, half of the new distance information would be wasted in redundantly covering part of the segment. This remark yields another strategy, bisecting in the middle of the uncovered section of the segment (Fig. 4(c)). However, in practice we could not notice a significant difference between approaches (a) and (c). Furthermore, (c) has the drawback of bisecting at different points for different pairs of bodies. In fact, we favor (a) because it allows for implementing an additional caching and indexing mechanism for rigid body transforms at the inserted configurations (see Section 4).

The bisection depth for an entry $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$ will depend on the minimum distance η_{min} that $\mathcal{A}_i$ and $\mathcal{A}_j$ assume over all configurations on the sub-segment $[\boldsymbol{q}_a, \boldsymbol{q}_b]$. Using a pure collision-check to evaluate the bisection point $\boldsymbol{q}_{mid}$ may therefore lead to a deep recursion, if the two bodies happen to come extremely close during the motion without colliding. One way to deal with this issue is to introduce a constant minimum (workspace) clearance δ and replace the collision check of $\boldsymbol{q}_{mid}$ by a test if the distance between $\mathcal{A}_i$ and $\mathcal{A}_j$ (or an upper bound for it) is smaller than δ. If one of these tests succeeds, we can report a "pseudo-collision".

3.2 Greedy distance computation

One tenet of our approach is the availability of an efficient algorithm to compute non-trivial lower distance bounds (or exact distances) between pairs of complex polyhedral bodies. Tighter bounds allow segment to be covered with less bisection, but they are also more expensive to compute.

Bounding volume (BV) hierarchies are commonly used for checking collision and/or computing distances between bodies (see Section 1). However, computing exact distances has proven significantly more expensive that pure collision checking. The efficiency gained by our adaptive bisection in using distances can easily be dwarfed by the higher cost of computing exact distances. Some BV algorithms are able to compute approximate distances much faster than exact distances (e.g., [14, 21]), but still slower than they can check collision. Furthermore, their efficiency depends on the allowed relative or absolute error, a parameter that may be difficult to set.

Instead, we settled for a new algorithm that greedily computes lower distance bounds between general non-convex polyhedra, while checking collision or minimal separation δ (as indicated in Section 3.1). This algorithm, called GREEDY-DIST, is also based on BV hierarchies. While being barely more expensive than a similar pure collision-detection algorithm, in practice it returns good lower bounds most of the time. It 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 RSS [14].

We assume that each BV hierarchy is an approximately balanced binary tree of BVs, with each leaf being a triangle of the surface of the polyhedral body represented by the hierarchy. To compute a lower bound on the distance between two bodies, we call GREEDY-DIST(B_i, B_j), where B_i and B_j denote the BVs at the roots of the hierarchies representing the two bodies. The algorithm is as follows, where $\delta \geq 0$ is the required minimum separation between the two bodies and B_{j1} and B_{j2} are the two children of B_j in the BV hierarchy:

Algorithm GREEDY-DIST(B_i, B_j)
  $d \leftarrow$ distance(B_i, B_j)
  if B_i and B_j are both triangles then return d
  if $d > \delta$ then return d
  if B_i is bigger than B_j then switch B_i and B_j
  $\alpha \leftarrow$ GREEDY-DIST(B_i, B_{j1})
  if $\alpha > \delta$ then
    $\beta \leftarrow$ GREEDY-DIST(B_i, B_{j2})
    if $\beta > \delta$ then return $\min\{\alpha, \beta\}$
  return 0

If the original call GREEDY-DIST(B_i, B_j) returns 0, then the distance between the two bodies is less than or equal to δ, otherwise it is greater than δ and the returned value is a lower bound.

	#BV/Tri pairs				[%]	
Fig.#	Coll	New	Dist	0.5-Dist	μ	μ'
1	2.1/0.03	2.1/0.03	822/226	8.0/0.17	78	59
6(a)	11/0.12	13/0.15	502/51	29/0.66	91	64
6(b)	4.8/0.17	4.7/0.14	466/47	23/0.78	88	63
7(a)	14/0.52	14/0.45	2342/458	69/2.4	65	60
7(b)	73/2.1	85/2.2	1363/216	174/6.6	52	57
7(c)	44/0.52	52/0.66	1845/215	172/4.2	58	59

Table 1: Comparison of GREEDY-DIST with collision checking and exact and approximate distance computation algorithms.

This algorithm is much more efficient than exact distance computation because it terminates as soon as a separation greater than δ has been shown, which happens for many configurations. In fact, when δ is set to 0, the algorithm visits exactly the same pairs of BVs and triangles as a standard recursive collision checker using the same BV type would do. But it additionally returns a non-zero lower bound on the distance if the two bodies do not intersect. The experiments reported below show that in practice this bound is quite good.

Table 1 compares the performance of GREEDY-DIST with $\delta = 0$ to algorithms as described in [14] for pure collision checking, exact distance computation and approximate distance computation (with 0.5 relative error in this case). Since the original approximate distance algorithm in [14] reports only an upper approximation, we slightly modified it along the lines of [21] to compute a lower approximation instead. The number returned by the algorithm is then guaranteed to be between $0.5d$ and d, where d is the exact distance.

Columns 2 through 5 of Table 1 give average numbers of pairs of BVs and triangles tested by each of the four implemented algorithms. These numbers were generated as follows, for the six examples shown in Fig. 1, 6 and 7. In each example, we generated 1,000 random configurations of the robot(s) and at each configuration all robot links were tested against all fixed obstacles. The numbers reported in Table 1 are averages over all configurations and all tested pairs. They show that the performance of GREEDY-DIST is similar to the pure collision checker (note that, in this case, computing the distance between two triangles or two RSSs costs only a small constant factor more than checking whether they intersect). But the performance of GREEDY-DIST is much better, by large factors, than the algorithm performing exact distance computation. It is also significantly better than the algorithm computing an 0.5-approximate distance. The reason is that, independent of the selected relative error, the approximate-distance algorithm recurses down to the leaf level of the BV hierarchies at least once, while GREEDY-DIST often terminates the search at higher levels.

The last two columns of Table 1 compare the lower distance bounds returned by GREEDY-DIST with the distances returned by the other two algorithms. The column labelled by μ describes the lower bounds by their average fractions in % of the exact distances. Here, the

averages were computed only for the collision-free configurations. Indeed, with $\delta = 0$, when there is a collision, GREEDY-DIST always returns the exact distance. All the factors in column μ are larger than 50%; half of them are significantly larger. The column labelled μ' gives the distances returned by the approximate-distance algorithm by their average fractions in % of the exact distances. In most cases, the μ' factor is smaller than the μ factor, indicating that GREEDY-DIST usually provides tighter bounds than the approximate-distance algorithm with 0.5 relative error, at smaller computational cost.

As is often the case with such comparison, there are subtleties that we must be aware of. Several types of BVs (e.g., spheres, AABBs, OBBs, RSSs, k-DOPs) have been proposed and discussed in the literature [18]. The BVs used by GREEDY-DIST, RSSs, provide reasonably tight approximations for a wide range of bodies found in robotic scenarios and allow fast distance computation. However, a distance query on two RSSs is slightly more expensive than an intersection query on two OBBs (the close relatives of RSSs). The implementation that was used for the pure collision checker uses OBBs. Since OBBs and RSSs are geometrically related, in most cases OBB and RSS hierarchies are very similar. This explains why the results in columns 2 and 3 of Table 1 are almost the same (hence, comparable). On the other hand, in our experiments, we found that performing an intersection query on two OBBs is, on average, only less than twice faster than computing the distance between two RSSs.

3.3 Bounding motions in workspace

As the configuration of the system changes by linear interpolation along a (sub-)segment $[\boldsymbol{q}_a, \boldsymbol{q}_b]$, every point of each moving body $\mathcal{A}_i$ traces a specific curve segment in 3-space. Our algorithm needs to efficiently compute a good upper bound $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ on the lengths of all curve segments traced by points of $\mathcal{A}_i$.

It can be shown that one of the vertices of a convex bounding polyhedron for $\mathcal{A}_i$ traces a curve segment that is not shorter than the curve segments traced by any point on $\mathcal{A}_i$. Our problem can thus be reduced to computing bounds for the motion of a small number of points. Assume that the motion of one such point is given by $p(t)$ for $t \in [0, 1]$. The exact length of the curve segment traced by $p(t)$ is given by $L = \int_0^1 \|\dot{p}(t)\| dt$. For general linkages, this integral cannot be solved analytically, and numerical evaluation with the required precision can be costly. Obviously, the straightforward approach of sampling the curve traced by $p(t)$ and approximating it with a sequence of straight line segments yields an underestimate of the length, while we need an upper bound to guarantee that no collisions will be missed.

Most practical manipulators contain either revolute or prismatic joints and a few of them contain both types. For robots with only prismatic joints, traced curves are straight segments whose exact lengths are easy to compute.

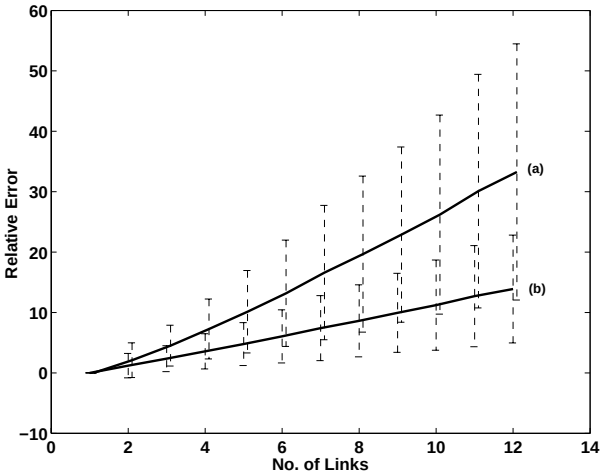


Figure 5: Relative errors and standard deviations of the two simple curve length bounds for different vertices on a planar linkage. (a) $\rho_i\ d(\boldsymbol{q}_a, \boldsymbol{q}_b)$, (b) $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$

To illustrate how we deal with revolute joints consider the following simplified example. Let the robot be a planar linkage, consisting of m straight line segments connected by m revolute joints with angles $\varphi_1, \ldots, \varphi_m$. Link 1 is directly connected to the environment and controlled by angle φ_1, link 2 is connected to the endpoint of link 1 and affected by angles φ_1 and φ_2, etc. Let each link have unit length and assume a simple correspondence between joint angles and c-space parameters, i.e. $\boldsymbol{q} = (q_1, \ldots, q_m)$ and $\varphi_i = q_i$.

A simple upper bound of the length of the curve traced by the endpoint of link i is:

$$\rho_i\ d(\boldsymbol{q}_a, \boldsymbol{q}_b) = \sum_{j=1}^{i} (i - j + 1) d(\boldsymbol{q}_a, \boldsymbol{q}_b)$$

where $d(\cdot, \cdot)$ is the c-space distance defined by the L^∞-norm and $\rho_i = \sum_{j=1}^{i} (i - j + 1)$ is the radius of the disc in which the endpoint of link i can move.

However, this bound can be improved by using the actual parameter differences and we obtain the bound

$$\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b) = \sum_{j=1}^{i} (i - j + 1) |q_j^{(a)} - q_j^{(b)}|$$

Fig. 5 shows the relative errors and standard deviations of the above two bounds for the endpoints of planar linkages of different lengths. The plots were obtained by sampling 1,000 random segments for linkages of lengths between one and twelve and comparing the upper bounds to the result of a numerical integration with high precision. Although $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ overestimates the curve length for the endpoint after 6 links by a factor of about 6 on average, this estimate will be cut in half by each bisection step and we therefore expect it to be reduced rather quickly. Furthermore observe that the error is smaller for links closer to the base. In our experiments in Section 4 $\lambda_i(\boldsymbol{q}_a, \boldsymbol{q}_b)$ appeared to be frequently dominated

by large workspace distances $\eta_{ij}(\boldsymbol{q}_a)$ and $\eta_{ij}(\boldsymbol{q}_b)$ even for kinematic chains of length up to 10 links.

The above discussion can be easily extended to robots with both prismatic and revolute joints, and also to more realistic robots with polyhedral links but we omit the details here.

3.4 Ordering of the priority queue

We now discuss the ordering of the entries $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$ in the priority queue used by the core algorithm presented in Section 3.1. For a collision-free segment, the ordering has no impact on the efficiency of the check, since all entries will be eventually processed. However, for a colliding segment, an appropriate ordering of the entries can be very effective to discover a collision quicker. In PRM planning, a significant amount of time is "wasted" in showing that candidate segments between sampled milestones are actually colliding.

In [8, 22], it was shown that in practice the prior probability of a segment to be colliding increases with its length in c-space. The planners in [20, 22] exploit this result to test multi-segment paths, by maintaining a priority queue of (sub-)segments sorted by decreasing length.

Here, we can take advantage that we also know bounds on workspace distances and on lengths of traced curves for each (sub-)segment. Intuitively, two bodies are more likely to collide if they are closer at one or both segment endpoints and/or the points in these bodies trace longer curves. This intuition is directly related to inequality (2). This leads us to proceed as follows. For each entry $[\boldsymbol{q}_a, \boldsymbol{q}_b]_{ij}$ in the priority queue we compute the length of the subsegment that is not covered (in the sense defined by the schema in Fig. 3). This is done by subtracting the left-hand side of (2) from its right-and side. We call the result the *non-covered length* of the segment between $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ for the bodies $\mathcal{A}_i$ and $\mathcal{A}_j$. If this length is negative or null, then the two bodies are guaranteed not to collide along the segment, hence the entry is not inserted in the queue. All other entries are sorted by decreasing non-covered length.

This heuristic ordering can be extended from single segments and their sub-segments to multi-segment paths [22]. Instead of checking all segments on such a path in some fixed sequence, the checker can enter all of them in the same priority queue and thus switch dynamically among them, as dictated by the uncovered lengths. In most cases, if a segment collides, this collision will be detected before having spent much time toward the validation of collision-free segments, hence making it possible to reject the entire path quickly. Actually, a better implementation is to maintain two levels of priority queues, one priority queue for the path (with one entry per segment that has not yet been shown free of collision), and one priority queue for each segment. The priority queue for a segment is the same as above. The priority of a segment in the path priority queue is the uncovered length of the first entry of this segment's priority queue. With these two levels of queues, if the path is eventually found to collide, we can then cache the priority queues of the segments that have neither been shown to collide nor to be collision-free. So, if these segments are later parts of other paths, the prior collision-checking work does not have to be repeated.

4 Experiments

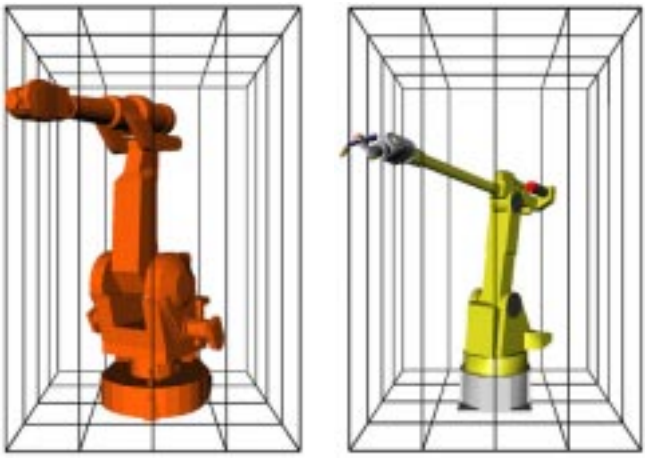
All experiments were performed on a 1GHz Pentium III PC with 1GB RAM, using a single processor. In our current implementation we recompute rigid-body transforms for both bodies of each tested pair whenever a (sub-)segment is bisected for this pair. In some examples presented in this section, more than 70% of the total running time is consumed by these partly redundant computations. Since we bisect the subsegments in the same uniform way for all pairs of bodies, identical transforms could be shared across pairs. We could also take further advantage of partly computed forward kinematics for manipulator arms. To this end, we are currently implementing a caching and indexing mechanism for rigid-body transforms that will allow for substantial reduction of running times in most examples.

4.1 Random segments

We first consider random segments to compare the performance of our new segment checker to a simple fixed-resolution approach that is used in most PRM planners. The simple approach bisects the segments and orders the (sub-)segment tests until their lengths in c-space become smaller than a fixed ε as described in [22].

Table 2 shows the results for 1,000 random colliding segments with collision-free endpoints. Since the segments are known to be colliding, we ran the simple checker with $\varepsilon = 0$ and our new checker with $\delta = 0$. (By setting $\varepsilon = 0$, we assure that the simple checker finds the collision.) The simple checker seems to perform better except for the example in Fig. 7(b). However, this is mainly due to redundant recomputations of transforms (a minor reason is that distance computation for a RSSs pair is slower than an intersection test for two OBBs). For example, in Fig. 7(c), there are many pairs of moving links and for each pair, the current implementation of the new segment checker recomputes transforms for both links of the pair. In Fig. 6, the redundancy is less obvious but can be explained by the fact that each wire of the cage is modeled as an individual obstacle. Thus, the transform for each robot link is recomputed when tested against a part of the cage. As noted before, we are currently implementing a caching mechanism that will eliminate the redundant recomputations of transforms. Another way to solve the problem in Fig. 6 would be to construct a BV hierarchy for the entire cage.

Table 3 compares the average times per segment for 1,000 random collision-free segments which were gener-



(a) (b)

Figure 6: Examples with thin obstacles (cages). (a) IRB 2400 robot (2,991 triangles), (b) Fanuc 200 robot (2,502 triangles) with arc welding gun.

Colliding segments		
Fig.	Simple ($\varepsilon = 0$)	New ($\delta = 0$)
6(a)	2.04	2.97
6(b)	2.18	3.38
7(a)	0.96	0.99
7(b)	2.2	1.3
7(c)	3.2	15

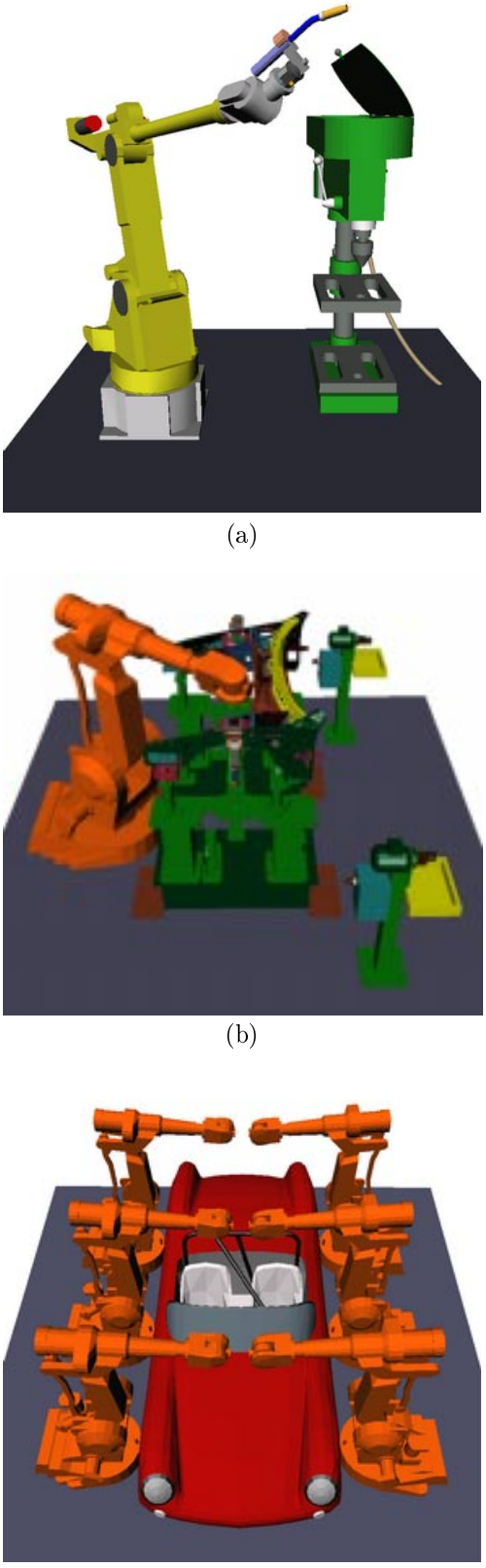
Table 2: Average times per segment (in millisec.) of simple and new collision checker for 1,000 random colliding segments (with free endpoints).

Free segments						
\	Simple		New		New	
Fig.	ε	t [ms]	δ	t [ms]	δ	t [ms]
6(a)	0.0120	90	0.001	44	0.01	14
6(b)	0.0060	104	0.001	31	0.01	14
7(a)	0.0012	52	0.001	26	0.01	6.5
7(b)	0.0120	56	0.001	24	0.01	6.6
7(c)	0.0120	295	0.001	81	0.01	16

Table 3: Average times per segment (in millisec.) for simple and new collision checker for 1,000 random collision-free segments.

	SBL		**A-SBL**
Example	ε	t [sec]	t [sec]
Fig. 6(a)	0.0120	83	44
Fig. 6(b)	0.0060	17	4.80
Fig. 7(a)	0.0012	3.20	2.10
Fig. 7(b)	0.0120	1.20	0.81
Fig. 7(c)	0.0120	85	52

Table 4: Comparison of original SBL planner and A-SBL variant with new segment checker.



(a) (b) (c)

Figure 7: (a) Fanuc 200 with arc welding gun and a machine tool (obstacles: 34,171 triangles). (b) IRB 2400 robot in workshop (obstacles: 74,681 triangles). (c) Six IRB 2400 robots working on a car body (car: 19,668 triangles).

ated using the new checker with $\delta = 0.001$. (This value is three orders of magnitude smaller than the sizes of the robot links and corresponds to a millimeter for a robot of realistic size.) A different value of ε was determined experimentally for each of the examples in order to achieve a reasonable tradeoff between performance and accuracy of the simple collision checker (see Section 4.2). Both the simple checker and the new checker were then run on the generated segments. The results show that our new segment checker performs faster than the simple checker, although the abovementioned overhead of using RSSs and redundant computations of transforms applies here as well.

We generated another set of 1,000 random free segments using the new checker with $\delta = 0.01$ (which corresponds to one centimeter). While the performance of the simple checker is unaffected by the increased distance from the obstacles, the new checker (with $\delta = 0.01$) performed significantly better (see columns 6 and 7 in Table 3).

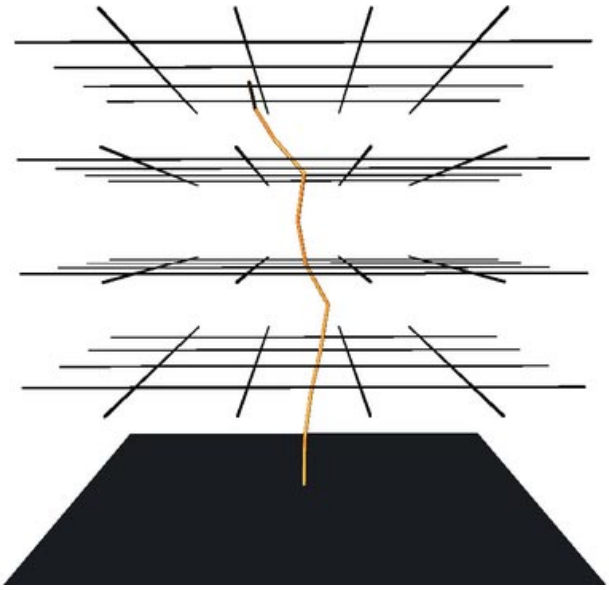


Figure 8: A skinny 20-DOF linkage amongst thin lattices. Another, similar version contains nine linkages, each threaded through a different set of holes of the grids.

4.2 Application in a PRM path planner

The PRM planner described in [22], called SBL, bisects each segment up to a given resolution ε (see Section 2). It assumes that the segment is collision-free if all intermediate points are collision-free. This approach does not prove that a segment is actually collision-free, but the error probability can be made arbitrarily small by decreasing ε. However, reducing ε also means that more bisection points have to be checked and this increases the planning time. It thus requires some trial-and-error experimentation to determine a reasonable value of ε and this has to be done for every environment anew.

We have replaced this discrete segment checking algorithm in the original SBL implementation by our new adaptive segment checker. We call the resulting planner "adaptive SBL" (A-SBL). This planner is guaranteed to return a collision-free path. To prevent deep recursion, as described in Section 3.1, we set the minimum workspace clearance to $\delta = 0.01$ which is two orders of magnitude smaller than the sizes of the links.

In Table 4 we compare the running times of SBL and A-SBL on examples in the five environments shown in Fig. 6 and 7. For each environment, a parameter ε was determined for SBL such that the planner returned a collision-free path for 10 successive runs on the respective problem. The values of ε are indicated in the table. The times reported in the table are averages over 10 runs of each planner. For these examples and values of ε, A-SBL is faster than the original version of SBL, with the enormous advantage that unlike most implemented PRM planners, it is absolutely guaranteed to return collision-free paths.

4.3 Application to path smoothing

We implemented a simple but effective randomized path smoother that works as follows. At each step, it samples a pair of configurations (not necessarily vertices) on the path and then tries to shortcut the path section between them by a single straight line segment. This is where our new collision checker is used. Sampling the endpoints of the potential shortcut uniformly from the entire path may lead to a high rejection rate, so we maintain a dynamic window from which the two new configurations are picked at each step. The smoother works well in practice and can be stopped at any time, either after a prespecified number of iterations or when the path length has converged. Since it is only an application example, we omit further details here.

With this simple path smoother, we were able to find quite good paths for a set of difficult examples (Figs. 1, 8), even with a remarkably simple planner. The starting configurations are given by the fully extended linkage(s) which are attached with one end to the base plate. The goal is to retract the linkage(s) downward out of the obstacles but no goal positions are given explicitly. All possible collisions are checked. These examples are not only difficult because they describe narrow channels in high-dimensional c-spaces but also because it is easy to miss collisions when checking segments at a fixed resolution. In fact, the c-spaces could be intuitively described as "high-dimensional swiss cheese". A segment checker based on discrete static collision checks would require extremely small step sizes in order not to miss a wall of the channel and thus waste most of its time on checking

	Random walk		Smoothing		σ
	# segs.	t [sec]	# steps	t [sec]	[%]
Fig. 1	85,784	898	10,000	735	7
Fig. 8	6,795	95	2,000	83	17
(*)	27,596	8,079	20,000	8,891	22

Table 5: Results for random walks and smoothing. Example (*) is similar to Fig. 8 but has nine linkages (180 DOFs).

those parts of the robot which are far apart from the obstacles and cannot collide.

The "planner" that we used here performs a random walk in c-space by changing a single DOF at each step. The reason why a simple random walk finds out of the channels in the above examples is that the expected radius of gyration (the radius of the smallest circumsphere) of a chain with n links is proportional to $\sqrt{n}$ [15].

The path smoother interpolates in the full-dimensional space and thus introduces simultaneous coordinated motions of all DOFs. Table 5 shows the obtained results for "planning" and smoothing, respectively. The number of tested segments include both free and colliding segments. The number of smoothing steps was given in advance and corresponds to the number of potential shortcuts tested by the smoother. The column σ shows the fraction in % of the path length after smoothing, respectively. The example (*) is similar to Fig. 8 but contains nine identical linkages, each threaded through a different set of grid holes.

In all of these examples, the computation of rigid-body transforms currently consumes about 60%-70% of the total running time. These examples would therefore benefit significantly from the method for caching and re-using transforms which we are currently implementing.

5 Extensions

The basic algorithm can be extended to allow for requiring a minimum clearance along an entire segment/path. In the form presented in Section 3, our algorithm reports a pseudo-collision if any two bodies are found to be less than δ apart at one of the interpolated configurations. However, this does not guarantee that they are at least δ apart during the entire motion. Inequality (2) rules out only true intersections along the segment. To guarantee that a pair $(\mathcal{A}_i, \mathcal{A}_j)$ is further than δ apart for all configurations on the segment, (2) can be restricted in the following way.

Suppose we dilate both bodies with a sphere of radius $r = \delta/2$ and thus obtain two 'hull' bodies $\mathcal{A}_i^r$ and $\mathcal{A}_j^r$. Proving the segment collision-free for the pair $(\mathcal{A}_i^r, \mathcal{A}_j^r)$ would certify the required separation of $(\mathcal{A}_i, \mathcal{A}_j)$. Each of the workspace distances of $(\mathcal{A}_i^r, \mathcal{A}_j^r)$ at $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$ would be smaller by δ than the corresponding distances $\eta_{ij}(\boldsymbol{q}_a)$ and $\eta_{ij}(\boldsymbol{q}_b)$ of the pair $(\mathcal{A}_i, \mathcal{A}_j)$, respectively. Instead of actually using the hulls we subtract 2δ from the right hand side of the original certificate (2). Furthermore, the curve length bounds have to be replaced by bounds derived for the hulls, $\lambda_i^r(\boldsymbol{q}_a, \boldsymbol{q}_b)$ and $\lambda_j^r(\boldsymbol{q}_a, \boldsymbol{q}_b)$, because points on the hulls may trace longer curves. Following Section 3.3, this is straight-forward and does not require actual computation of the hulls, either.

A more conservative variant of (2) which guarantees δ clearance of $(\mathcal{A}_i, \mathcal{A}_j)$ along the entire segment can thus be written as

$$\lambda_i^r(\boldsymbol{q}_a, \boldsymbol{q}_b) + \lambda_j^r(\boldsymbol{q}_a, \boldsymbol{q}_b) < \eta_{ij}(\boldsymbol{q}_a) + \eta_{ij}(\boldsymbol{q}_b) - 2\delta \quad (3)$$

Without going into further details, we note that we can even use a different minimum clearance for each pair of bodies, which may be important in certain practical situations.

A similar extension would be to consider $\delta < 0$ for allowing bounded penetration by less than $|\delta|$. However, in order to allow for bounded penetration along the entire segment, the function $\eta_{ij}(\cdot,\cdot)$ would have to be extended to compute negative penetration distances in order to verify the segment endpoints and the configurations generated by the bisection.

6 Conclusion and future work

We have presented a new exact collision checker for straight line segments and entire paths in c-space that is particularly suited for PRM planners applied to manipulator arms and multi-robot systems. Unlike the commonly used approach of checking discrete configurations up to a fixed resolution, it has the enormous advantage of never missing a collision. It also obviates the need for experimentally determining a reasonable resolution parameter.

Exactness 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. While this basic idea has been proposed before, we introduced several new techniques that make the method applicable to PRM planners and scenarios with realistic complexity. These techniques include a greedy distance computation algorithm that is as efficient as collision checking, a simple and efficient method for bounding lengths of paths traced out in workspace, and a scheme for ordering collision tests to reveal collisions as quickly as possible.

In our experiments, the new checker was faster than the fixed-resolution method with an appropriately set resolution. Beyond pure collision checking, new our algorithm can be easily extended to monitor individual minimum workspace clearances between each pair of objects.

We are currently implementing a caching and indexing mechanism for rigid-body transforms that will allow for substantial reduction of running times in many of the presented examples. Future work will include further optimization of the tradeoff between bisection and refinement of bounds on distances and curve lengths.

Acknowledgements: This work was partially funded by an NSF ITR grant, a gift from General Motors, and a grant from ABB.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3d workspaces. In *Proc. of the Workshop on Algorithmic Foundations of Robotics (WAFR'98)*, pages 155–168, March 1998.

[2] J. Barraquand, L. Kavraki, J. C. Latombe, T.Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *Inernational Journal of Robotics Research*, 16(6):759–774, 1996.

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

[4] R. Bohlin and L. Kavraki. Path planning using lazy PRM. In *Proc. of the Int. Conf. on Robot. & Autom. (ICRA)*, pages 521–528, 2000.

[5] S. Cameron. A study of the clash detection problem in robotics. In *Proc. IEEE Int. Conf. on Robotics & Automation*, volume 1, pages 488–493, 1985.

[6] S. A. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Trans. Robotics Automat.*, 6:291–302, June 1990.

[7] 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 *Sym. on Interactive 3D Graphics*, pages 189–196, 218, 1995.

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

[9] A. Foisy and V. Hayward. A safe swept volume method for collision detection. In *The Sixth International Symposium of Robotics Research*, pages 61–68, Pittsburgh (PE), Oct. 1993.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. *Comp. Graphics*, 30(Annual Conf. Series):171–180, 1996.

[11] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9(4&5):495–512, 1999.

[12] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. *Computers and Graphics*, 25(2):269–285, 2001.

[13] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[14] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *IEEE Conf. on Rob. and Auto.*, 2000.

[15] A. Leach. *Molecular Modelling: Principles and Applications*. Longman, Essex, England, 1996.

[16] M. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In J. P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation: WAFR 1996*, pages 129–142. A. K. Peters, 1996.

[17] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *IEEE Int. Conf. on Rob. and Auto.*, pages 1008–1014, 1991.

[18] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *IMA Conference on Mathematics of Surfaces*, volume 1, pages 602–608, San Diego (CA), 1998.

[19] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[20] Ch. Nielsen and L. E. Kavraki. A two-level fuzzy PRM for manipulation planning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Japan, 2000.

[21] S. Quinlan. Efficient distance computation between non-convex objects. In *IEEE Intern. Conf. on Rob. and Auto.*, pages 3324–3329, 1994.

[22] G. Sanchez and J. C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Int. Symposium on Robotics Research (ISRR'01)*, Lorne, Victoria, Australia, 2001.

[23] G. Van der Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphic Tools*, 2(4):1–13, 1997.