KCD: a Collision Detector for Path Planning in Factory Models

Carl van Geem, Thierry Simeon

KCD: a Collision Detector for Path Planning in Factory Models.

C. Van Geem*, T. Siméon
LAAS-CNRS, Toulouse, France

Abstract

A critical operation in robot path planning is the verification for collision between the robot and the obstacles in its environment. In this contribution we present a new collision detector KCD dedicated to path planning for mechanical devices and robots in industrial-size CAD models. We describe the architecture of KCD and the way in which it is applied in the path planning tool Move3D. We report on experimental results showing its performance.

1 Introduction

A collision detection algorithm takes as input a soup of possibly moving objects and determines whether or not any of these objects clash. The algorithm may return a boolean result, the minimal distance between the nearest pair of objects, all pairs of colliding objects, or an estimation of penetration in case of collision. Applications may have to feed the algorithm with a particular format or may need particular feedback. Preconditions on the representation of objects can be exploited by an algorithm (e.g. if all objects are convex) resulting in gain of efficiency over loss of generality.

In this paper we will apply collision detection in the frame of path planning with the probabilistic roadmap planning approach (PRM, see [13], [2]). This method constructs a roadmap (or graph) of randomly generated collision-free robot configurations (nodes) connected by collision-free local paths (edges). A particular path planning problem is then reduced to a graph search. Our goal is to plan motions for mechanical devices (cranes, carts, trucks,...) and robots in complex CAD models of industrial sites (object representation by solids, CSG, and possibly non-convex polyhedra), where most objects do not move. This path planning problem is application oriented and the main topic of the European project *Motion for Logistics* (MOLOG).

In Section 2 we present a brief overview of the use of collision detectors in path planning and CAD modelling. Section 3 presents the main contributions of this paper, which are the development of the collision detector KCD - dedicated to path planning for articulated devices in complex CAD models - and the way in which collision detection is used for path planning purposes in the platform Move3D (see [23]). KCD allows efficient planning of guaranteed collision-free paths. We report experimental results in Section 4, before concluding.

*supported by the MOLOG project (Esprit IV, LTR 28226), now at KINEO Computer Aided Motion.

2 Related Work

Extensive and recent surveys on collision detection are published (e.g. [17], [12], [10]). Software of several collision detectors are freely available for research purposes. In this section we concentrate on the observations we made while reading about and experimenting with collision detection algorithms in the frame of path planning applications in industrial models.

Collision Detection Usually the input of a collision detector is a soup of objects, all objects can and do move a priori. Some CDs allow the user to define the pairs of objects that have to be tested or ignored; e.g. collision pairs in I_Collide ([5]), V_Collide ([11]).

Computation often falls apart in three stages: *pre-selection* of possibly colliding objects, *focussing* on entities of the selected objects more likely to collide, and *precise computation* of a collision or of a witness of disjointness. Pre-selection and entity focussing are usually based on computations with bounding volumes (BVs) that approximate the real objects.

The efficiency of a CD depends on the representation of the objects: the convexity in [5] allows efficient entity focussing. The choice of adequate BVs depends on the original objects: spherical shells ([16]) and pie-slices ([1]) are well-suited BVs for tesselated surfaces whereas oriented bounding boxes (OBBs, [8]) and k-DOPs ([15]) do better for rather spherical objects. Also, when objects move, their BVs must move or must be recomputed.

Different collision detectors use different representations and algorithms at the stage of precise

computation and return different kinds of information. V_Collide and RAPID ([8]) test intersection between two triangles, hence only find collisions between triangularized surfaces. SOLID ([24]) uses an iterative method to find a witness of disjointness or a pair of nearest points of two convex solids. V-Clip ([18]) computes the distance between two non-intersecting convex polyhedra or a penetration measure in case of collision.

Path Planning Applications Several collision detection algorithms were used for path planning (e.g. RAPID in [14], I_Collide in [22], V_Collide in [19], SOLID in [3]). The choice of a particular CD algorithm is often based upon the feed-back given by the CD and needed for the planning algorithm. In [26] V-Clip is used because it returns information on penetration. Usually, whereas the planning problem is difficult to solve for a planning algorithm, the environments have simple geometry.

CAD Applications. Some contributions to the field of CD are applied to industrial scenes. These include research aimed at the introduction of Virtual Reality techniques for car industry ([27], [28]) and cooperation with Boeing ([9], [15]). There exist also some path planning applications in industrial environments ([21], [25]). In all these applications unimportant cavities are omitted, models are simplified, the number of calls to the CD minimized.

3 KCD for PRM in CAD Models.

Previously we used several of the above mentionned collision detection packages in the motion planning platform Move3D [23], but none of them are designed for the particular application in path planning for devices in CAD models. Therefore we decided to develop our own collision detector. KCD can be seen as a hybrid collision checker that contains pre-selection and entity focussing techniques as in V_Collide for handling complex scenes, and precise computation techniques as in SOLID allowing to process composite models made up of possibly concave polyhedra together with volumic primitives (spheres, tubes, cones,...). Furthermore, when no collision is found, KCD can return a distance estimate to the nearest obstacle. In this section we first indicate the particularities that motivate the architecture, after which follows its description.

KCD and CAD. Collision detection becomes more time consuming in CAD models due to their high geometrical complexity. Hence we aim for a reduction of the size of the data structures proper to the collision checker without loss of exactness and solvability of the path planning problem. The limitation of the size of the data structures also leads to a faster initialization phase when charging a model.

We observed in experiments with V_Collide that the notion of *object* is important. In a first stage we defined a V_Collide object for each of the polyhedra in the scene. This resulted in too many small axis aligned bounding boxes (AABBs) and most computation time was spent in the pre-selection stage. The OBB-trees on the set of the triangularized facets had an uneefficient height. Grouping the triangles of several polyhedra in one V_Collide object increases performance. However, the notion of object in a CAD system usually groups primitives with the same semantics (e.g. the piping part of the environment) rather than the primitives of one geometrical object, potentially affecting the efficacy of the OBB-trees built on top of the grouped primitives. In the initialization phase of KCD we try to group the primitives automatically in a geometrically meaningful way.

Primitives are simple solids in many CAD systems. Therefore KCD allows a mixture of solids and convex facets. Solids need not to be approximated by polyhedra since in the third stage we apply the GJK algorithm.

The CAD models used in industrial applications may be recycled models, designed for other purposes than path planning. Often objects are more detailed than necessary for path planning. KCD filters away unnecessary details in a conservative way.

KCD and Path Planning. The pre-selection stage of KCD exploits the fact that in CAD path planning applications most objects of the model are fixed. KCD pre-processes the static part of the model at the initialization phase of the internal data structures and thus avoids the general notion of collision pairs at the basis of the *nbody* mechanism in V_Collide for the preselection of possibly colliding object pairs.

Probabilistic Roadmap planners make extensive use of collision detection for node and edge verification. When the CD only returns a boolean answer, edge verification is done by a high number of calls to a static collision detector for configurations along a chosen path between two nodes. In this case some collisions may not be detected when occuring between two consecutive tested collision-free configurations. The path planner can benefit of supplementary information like a distance estimate to the nearest obstacle or BV. It allows fewer calls to the CD and guarantees collision-freeness of the paths along the edges of the

roadmap. Moreover, path planning applications may require to take into account a user-defined security distance; an extra reason to require distance estimation.

4 KCD Architecture.

In order to treat the static part of the model seperately, we introduce the notion of object type: *static obstacle* and *moving object* (for each link of a device as well as for a freight). KCD tests each moving object for collision with the obstacles, as well as pairs of moving objects. KCD only allows to define the pairs of moving objects to be tested, a necessary feature for articulated devices with adjacent always intersecting links in their geometrical model.

The choice of the bounding volumes (AABBs and OBBs) is also motivated by the observation that since most objects do not move, their bounding volumes must never be re-computed. Furthermore, the placement of OBBs around movable objects can be expressed with the same transformation matrix as for the movable object.

Initialization for pre-selection We first structurize the static objects. We construct a hierarchy of AABBs and group the objects automatically according to their placement in workspace. We put an AABB around each of the primitives. Pairwize overlapping AABBs belonging to the same CAD-object are grouped in an AABB on the next level in the hierarchy and define a geometrically meaningful *KCD object*. For the higher levels of the hierarchy we repeat the grouping of pairwize overlapping AABBs of any of the AABBs in the previous level. The AABB hierarchy is a tree with nodes having multiple child nodes. An AABB is placed around each movable object as well. The AABBs are used in the pre-selection stage.

Initialization for entity focussing We use binary OBB-trees around the KCD-obstacles and around each of the moving objects. The construction of these trees is done in two steps. Firstly an OBB-tree is constructed on each polyhedron and an OBB is places around each solid. Secondly an OBB-tree is constructed on top of these for each KCD-obstacle. The OBB-trees are binary trees constructed in a top-down fashion, starting of with one large OBB around the whole collection of geometrical data and creating two smaller OBBs around half of the geometrical data used for the construction of their parent box. The first kind of OBB-tree takes as input the convex facets of the polyhedron and the triangles computed by a triangulation algorithm applied to its non-convex facets. The second kind of OBB-tree takes as input the OBBs around the solids in the KCD-obstacle and the OBB in the root of the OBB-tree of the polyhedra in the KCD-obstacle. The resulting OBB-tree on the KCD-obstacle is connected to the OBBs and OBB-trees at its leafs. Each movable object is treated in the same way as a KCD-obstacle for the OBB-tree construction.

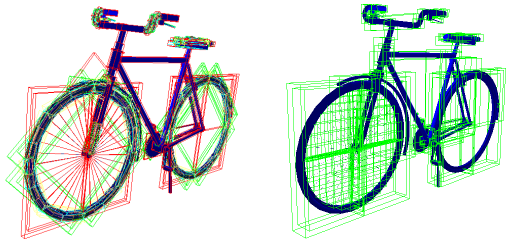

Figure 1: object OBB-tree, AABBs of its primitives.

Precise computation As in SOLID, we use the enhanced GJK method ([7], [4]) on the lowest level: simple solids can be treated as such (see [6]). Since we treat convex facets at the lowest level, we can put an OBB-leaf around a convex facet rather than only a triangle (as required by [8]).

Processing queries It is up to the user to decide whether KCD reports on collision by true or false without further information, or it returns a distance estimate in case of non-collision for each of the movable objects. At each of the three stages faster procedures return only boolean values. The distance estimate is the smallest value computed at one of the levels to decide for non-collision, and therefore is only a conservative estimate. The distance-mode also allows the user to set a safety distance.

The overlap test for AABBs is applied to the AABB hierarchy starting at the root of the tree. When all child boxes intersect an AABB of a movable object, testing is stopped at that level of the tree and the parent AABB is stored. This mechanism results in a collection of AABBs for each of the movable objects. Each AABB contains obstacles that potentially collide. If already at this stage no collision occurs, and if the user asked for a distance estimate, the smallest distance between the AABBs of the movable objects and the visited AABBs of the static environment is returned.

The overlap test for two OBBs limits computations more than the corresponding function in RAPID. The version returning a boolean answer tests at most 15

cases in its quest for a separating axis. A minor modification to this algorithm allows to compute the distance between the boxes. The overlap test is called recursively on the OBB-trees of objects in the AABBs selected at the first stage. Again, if already at this stage no collision occurs, and if the user asked for a distance estimate, the smallest distance between the OBBs of the movable objects and the OBBs of the static environment is returned.

At the lowest level the GJK algorithm is called on the geometrical elements englobed by the leafs of the OBB-trees. As in SOLID, two different algorithms deliver two different types of answers: a boolean reply or a distance. This test is called when in both of the OBB-trees a leaf is reached.

Omitting details of the scene Another useful feature of KCD is the possibility to filter out details of objects that are useless for path planning purposes. We call a bounding box *small* when its volume (or the surface of degenerate bounding box around a facet) is smaller than v^3 (or v^2) for a given value v. At the initialization of the internal data we stop the top-down construction as soon as the parent bounding box groups only small bounding boxes. When the bounding box of a primitive or a facet is small, we do not call GJK for that primitive or facet. Similar to the filtering technique at the initialization of the internal data structures, the user can demand to omit detailed checking during the queries. Upon request, KCD does not descend any further as soon as a small bounding box is treated, even if this internal data structure exists. The result returned by KCD is thus guaranteed more conservative than without filtering.

Checking path validity For a tested collision-free configuration along a given path, KCD returns a distance estimate for each movable object, hence for each link of an articulated robot. This allows to compute a collision-free interval in the joint-space of the robot, and to cover the path by collision-free intervals around tested configurations. Also, collisions along trajectories mostly occur somewhat further away from the collision-free initial and goal configurations. One expects that dichotomic sampling of a path finds a collision sooner than when sampling the path stepwize. Since most local paths are in collision, dichotomy improves performance of the path planner, as our experiments show. Finally, in the case of articulated devices that may self-collide (e.g. some robot arms, humanoids), it is wise to check for self-collision before starting the computation with the rest of the model.

5 Experiments in Move3D.

We used the platform *Move3D* (see [23]) in order to experiment with KCD and to compare with V_Collide.

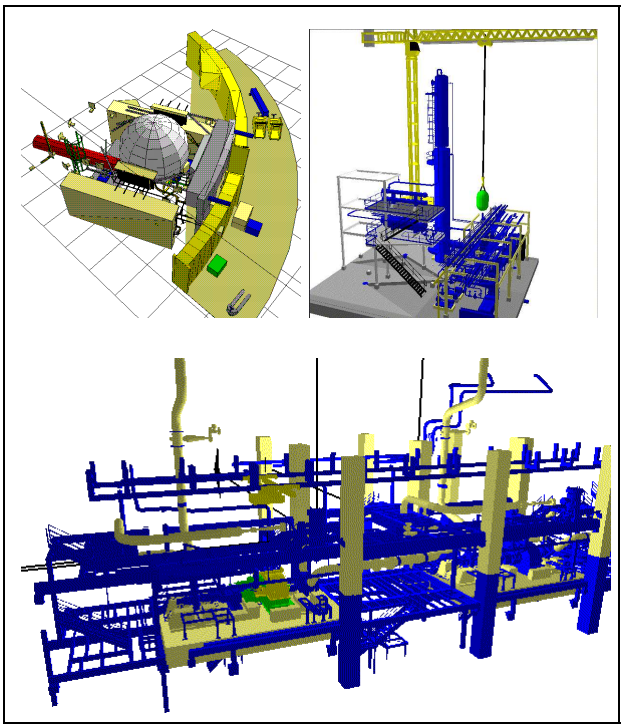

Figure 2: Scenes EDF, Stabilizer and TPA.

Models: We used the three scenes shown in Figures 2 for our experiments. The complexity of the different models is indicated in the table below. The first line is the total number of polyhedra and solids in the model, the second line is the total number of solids, the third line is the total number of polyhedra. The fourth line is the number of static polyhedra and solids in the model, that is the initial number of AABBs before grouping in KCD objects. The fifth line is the number of robot links, hence the number of movable KCD objects in the model. The sixth line is the number of CAD objects in the model, also the number of V_Collide objects (both static and movable, since V_Collide does not make the difference). The last line is the number of KCD static objects, after automatic AABB grouping. In the pre-selection stage these are the leafs of the AABB hierarchy. This is also the maximal number of OBB-trees on KCD obstacles tested in the focussing stage against the movable OBB-trees of line 5. In the precise computation phase, KCD will act on solids and the facets of the polyhedra, whereas

V_Collide acts on the triangulated facets of all primitives (each solid is then approximated by a polyhedron).

	Edf	Stab.	TPA
1. primitives	869	4829	7353
2. solids	0	3002	6213
3. polyhedra	869	1827	1140
4. static prims.	850	4650	7199
5. robot links	1	4	3
6. VCollide obj .	35	144	86
7. KCD obstacles	169	388	1307

Complexity of the models.

Results: The first table reflects the computation time needed in order to generate 5 nodes of a graph with the PRM-visibility method (see [20]). Grouping several polyhedra in one object improves performance of V_Collide due to a better pre-selection. Dichotomy for path verification results in less calls to the collision detector. The boolean static collision detecting by KCD is somewhat faster than V_Collide. When KCD returns a distance estimate instead of a boolean, the number of calls can be reduced further for path verification. This leads to a further gain of efficiency for path planning although the static collision test is more time consuming when a distance must be computed. Note also that the path verification is certain when using the distance information.

CD	CD-obj.	along path	answer	time (sec.)	calls to CD
VCollide	pol.	stepwise	bool	2900	14039
VCollide	obj.	stepwise	bool	118	14039
VCollide	obj.	dichotomy	bool	54	6093
KCD	obj.	dichotomy	bool	24	6093
KCD	obj.	dichotomy	dist.	11	1822

Usage for PRM (model: Stabilizer).

The next table compares the performance for static collision checking. Since a solid in the Stabilizer model are not longer approximated by a polyhedron and its OBB-tree but treated as a solid with just one OBB around it, the proper data structure of KCD is smaller than the one of V_Collide and a significant gain of performance can be observed.

Model	hits/ calls	VCollide	KCD (bool)	KCD (dist.)
EDF	4568/10000	3.85	2.78	3.90
Stabilizer	249/1000	4.87	1.59	2.27

Static collision checking.

The third table compares the consumption of time and space in the initialization phase. KCD invests more time in the initialization phase but uses less memory than V_Collide due to the different treatment of solids. When ignoring details, the initialization phase is executed faster and memory usage is further reduced.

	Stabilizer	TPA
VCollide	-/18sec./143Mb	-/20sec./148Mb
KCD	0.0/23sec./96Mb	0.0/27sec./78Mb
KCD	10.0/19sec./61Mb	10.0/24sec./54Mb
KCD	20.0/16sec./44Mb	20.0/21sec./39Mb

Initialization (size v of negligable detail/time/memory).

The following table shows a gain in performance of KCD when details of the scene (of volume smaller than v^3 or surface smaller than v^2) are ignored. Somewhat more collisions are found due to less precision, since we approximate details up to their somewhat larger bounding box.

Value v	0	30	100	300
Collision	250	253	256	256
Time (sec.)	2.11	1.96	1.90	1.85

Initial volume (model: Stabilizer, KCD: 1000 calls).

The last table shows that KCD can take into account a security distance without loss of efficiency. With a higher safety distance more collisions are found in the same amount of time.

TPA	sd=0.0	sd=50.0	sd=100.0
	774/4.1	799/3.9	826/4.1

Safety distance (sd): number of hits/time in sec.

6 Conclusion.

We presented the new collision detector KCD - dedicated to path planning for articulated devices in complex CAD models. We described the way in which KCD is used for path planning purposes with the platform Move3D. Experiments show that KCD allows efficient planning of guaranteed collision-free paths. In future, it would be useful for path planners if KCD would return a penetration estimate in case of collision.

References

[1] G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell, and A. Tal. BOXTREE: A Hierarchical Representation for Surfaces in 3D. *Computer Graphics Forum*, 15(3), September 1996.

[2] J. Barraquand, L. Kavraki, J.-C. Latombe, T. Li, and P. Raghavan. A random sampling scheme for path planning. *The International Journal of Robotics Research*, 16(6):759–774, December 1997.

[3] V. Boor, M. H. Overmars, and A. F. Van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE Int. Conf. on Robotics and Automation*, 1999.

[4] S. Cameron. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In *IEEE Int. Conf. on Robotics and Automation*, 1997.

[5] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments. In *Proc. ACM Int. 3D Graphics Conf.*, pages 189–196, 1995.

[6] E. G. Gilbert and C.-P. Foo. Computing the distance between general convex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 6(1):291–302, June 1990.

[7] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[8] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *Proc. of ACM Siggraph '96*, 1996.

[9] M. Held, J. T. Klosowski, and J. S. B. Mitchell. Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs. In *7th Canadian Conference Computational Geometry*, 1995.

[10] P. M. Hubbard. *Collision Detection for Interactive Graphics Applications*. PhD thesis, Department of Computer Science, Brown University, October 1994.

[11] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated Collision Detection for VRML. In *Proc. of VRML '97*, 1997.

[12] P. Jiménez, F. Thomas, and C. Torras. Collision Detection Algorithms for Motion Planning. *Robot Motion Planning and Control, ed. J.P. Laumond*, Lecture Notes in Control and Information Sciences 229:305–343, 1998.

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

[14] L. E. Kavraki, F. Lamiraux, and C. Holleman. Towards Planning for Elastic Objects . In *Workshop on Algorithmic Foundations of Robotics*, pages 313–325, 1998.

[15] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), March 1998.

[16] S. Krishnan, A. Pattekar, M. C. Lin, and D. Manocha. Spherical Shells: A Higher Order Bounding Volume for Fast Proximity Queries. In *Workshop on Algorithmic Foundations of Robotics*, pages 177–190, 1998.

[17] M. C. Lin and S. Gottschalk. Collision Detection between Geometric Models: A Survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.

[18] B. Mirtich. V-Clip: Fast and Robust Polyhedral Collision Detection. Technical Report TR97-05, Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA, 1997.

[19] C. Nissoux. *Visibility and Probabilistic Methods for Motion Planning in Robotics*. PhD thesis, LAAS-CNRS/Paul Sabatier University Toulouse, 1999.

[20] C. Nissoux, T. Siméon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *IEEE Int. Conf. on Intelligent Robots and Systems*, 1999.

[21] L. Overgaard, H. G. Petersen, and J. W. Perram. Motion planning for an articulated robot: A multi-agent approach. In *6th. European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 171–182, 1994.

[22] T. Simeon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6):445–550, 2001 (shorter version also published in *Int. Conf. on Intelligent Robots and Systems*, 1999).

[23] T. Simeon, J.-P. Laumond, C. Van Geem, and J. Cortes. Computer aided motion: Move3d within molog. In *IEEE Int. Conf. on Robotics and Automation*, 2001.

[24] G. van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[25] C. Van Geem, T. Siméon, J.-P. Laumond, J.-L. Bouchet, and J.-F. Rit. Mobile analysis for feasibilities studies in cad models of industrial environments. In *IEEE Int. Conf. on Robotics and Automation*, 1999.

[26] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space. In *IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.

[27] G. Zachmann. Real-Time and Exact Collision Detection for Interactive Virtual Prototyping. In *ASME Design Engineering Technical Conferences*, 1997.

[28] G. Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. In *IEEE Virtual Reality Annual International Symposium*, 1998.