Efficient Collision Detection for Path Planning for Industrial Robots

László Zahorán
EPIC Center of Excellence in Production Informatics and Control,
Inst. Comp. Sci. & Control, Hun. Acad. Sci., and
Dept. Measurement and Information Systems, Budapest Univ. Technology and Economics
laszlo.zahoran@sztaki.mta.hu

András Kovács
EPIC Center of Excellence in Production Informatics and Control,
Inst. Comp. Sci. & Control, Hun. Acad. Sci.
andras.kovacs@sztaki.mta.hu

ABSTRACT

Efficient collision detection is crucial for the success of automated process planning and path planning for robotic manipulation and assembly. Yet, collision detection for articulated industrial robots holds various challenges. This paper gives an overview of these challenges, and presents an efficient implementation of collision detection techniques for such robots. The applicability of the developed techniques to support path planning in an industrial test case is also illustrated.

1. INTRODUCTION

A crucial requirement towards automated process planning and path planning methods for robotic operations is that they must guarantee the geometric feasibility of the computed plans, by ensuring that no collisions occur during the movement of the robot and the manipulated objects. At the same time, collision detection is typically the computationally most challenging sub-problem of path planning [1, 3]. Particular challenges in collision detection for industrial robots lie in the following:

- Collision detection methods must be able to find *all potential collisions* of every moving and static object in the work cell, including the robot, the gripper, the workpiece, the fixture, as well as all static elements of the cell.
- The above objects are all characterized by *complex free-form geometries*.
- While the *continuous motion* of the robot must be checked for collisions, nearly all approaches in computational geometry focus on checking static configurations. To overcome this discrepancy, continuous collision detection must be reduced to an appropriately defined series of queries on static configurations.
- The kinematic chain of typical articulated industrial robots consists of 6 or more robots links. The motion of these links can be characterized in the *joint configuration space* of the robot, i.e., by the vector of joint angles. At the same time, effective tasks must be planned and collisions must be detected in the *Cartesian space*. Hence, the *mapping* between the two representations must be maintained at all times. For this purpose, *forward kinematic transformation* calculates the position of the robot links and the grasped objects in the Cartesian space from the joint angles, whereas *inverse kinematics* search for the joint angles that realize a given position in the Cartesian space. Yet, for many kinematic structures, the inverse kinematic calculation is a challenging computational problem with non-unique solutions.
- Certain types of contact between objects are allowed, e.g., between neighboring robot links or between the gripper and the workpiece. Moreover, the allowed types of contact may vary over time, e.g., a workpiece can touch the fixture when the robot inserts the workpiece

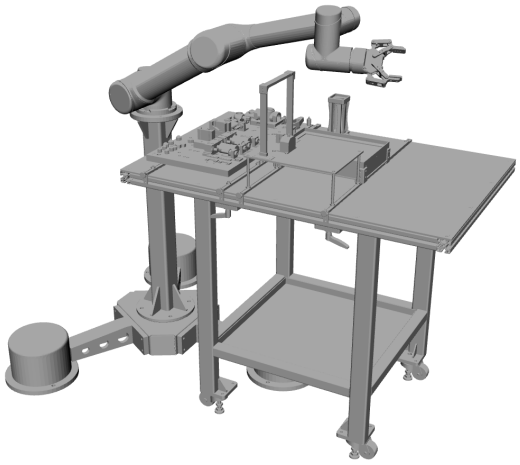

Figure 1: Work cell with a UR5 robot and a Robotiq gripper.

DOI: https://doi.org/10.26493/978-961-7055-82-5.19-22

into the fixture, but the same contact is forbidden during other motions. Hence, *collision rules* must be maintained dynamically.

- The configuration of the robot and the work cell may vary over time, e.g., when the robot grasps or releases the workpiece. These configuration changes must be managed, and pre-computation techniques must be handled with care.

- Since process planning and path planning methods rely on iteratively checking a vast number of candidate robot motions, the *computational efficiency* of collision detection is crucial.

Various generic-purpose libraries are available today for collision detection, such as the Proximity Query Package (PQP) [2] or the Flexible Collision Library (FCL) [4]. These libraries offer collision and distance queries for static configurations of free-form 3D solid objects (although FCL handles some restricted forms of continuous collision queries as well). Hence, they must be extended substantially to respond to the above challenges.

This paper presents a library for efficient collision detection for industrial robots. The library is built on the top of the generic-purpose PQP collision detection engine, and extends it with various kinematic and geometric calculation methods to serve the needs of robotic process planning and path planning. On the top of these collision detection techniques, the library contains an implementation of the Rapidly-exploring Random Trees (RRT) single-query probabilistic path planning algorithm [3], as well as the Probabilistic Roadmaps (PRM) multi-query path planner [1], which use the continuous collision queries as a so-called local planner (i.e., checking the direct movement of the industrial robot between two configurations). The paper gives an overview of the implemented collision detection techniques and demonstrates their computational efficiency in industrial case studies.

2. COLLISION DETECTION FOR ARTICULATED INDUSTRIAL ROBOTS

As pointed out above, collision detection for industrial robots requires extending general-purpose collision libraries in two main directions: (1) robot motions specified in the joint configuration space must be mapped into the Cartesian space using forward kinematic calculations; and (2) continuous collision detection for the robot motion must be reduced to checking an appropriate series of static robot configurations.

A static configuration c of an m-axis industrial robot can be characterized by a vector of m joint angles in the form of $c = (\alpha_0, ..., \alpha_m) \in C$, where C is the configuration space of the robot, defined by its joint limits. As usual, we focus on linear movements in the joint configuration space. Accordingly, the movement between start configuration c_0 and end configuration c_1 is interpolated by $c(t) = c_0(1-t)+c_1t$, $t \in [0, 1]$. This movement is considered free of collisions if every static configuration $c(t)$ is collision-free for $t \in [0, 1]$.

Collision detection must capture every so-called collision object in the work cell, including the robot (with a separate collision object for each robot link), the gripper (with interchangeable geometric models corresponding to different degrees of opening), the workpieces, as well as all other objects in the cell. While the geometry of each collision object is characterized by a triangle mesh representation, given in an STL file, a configuration is described by a homogeneous transformation matrix for each collision object. This matrix can be computed by forward kinematics from the robot configuration.

Collision rules between pairs of collision objects define whether the contact of the two objects is considered as a collision or not. By default, the contact of the neighboring robot links, as well as the contact between the robot base and the static work cell elements are allowed. These default rules can be overridden dynamically depending on the task executed by the robot.

2.1 Sampling-based Collision Detection

Sampling-based collision detection is the most common approach in robotics to check robot movements. The continuous movement $c(t)$ is sampled by looking at a finite set of static configurations $c(t_i)$, with $i = 1, ..., n$. Since the motion is given in the joint configuration space, the sampling rate is controlled by angle δ that specifies the maximum distance between neighboring samples $c(t_i)$ and $c(t_{i+1})$, using the maximum norm over different joints. The movement is classified as collision-free if and only if every static sample is collision-free.

A critical issue is the choice of parameter δ: using a low value is computationally demanding, whereas increasing δ also increases the risk of missing a collision. The proper value must be determined for each application individually.

Since typical path planning algorithms cannot exploit any information on the location of collisions, the checking of continuous movements can be interrupted upon finding the first collision. This implies that the performance of the algorithm on colliding motions can be improved significantly by the proper ordering of the collision queries.

As a heuristic, the probability of collision rises with the distance from known collision-free configurations. Accordingly, the implemented algorithm checks the start and end configurations first, whereas in the iterative step, it bisects the previous motion sections until the distance decreases below the given threshold δ. Furthermore, when checking a given static configuration, collision queries for different pairs of collision objects are ordered by the probability of collision, estimated based on historic records. In contrast, all collision queries must be executed on collision-free movements.

2.2 Conservative Advancement

Instead of the above heuristic method for checking continuous movements, another approach that provides a formal guarantee of collision-free continuous movements is strongly preferred. Such an approach is the so-called Conservative Advancement (CA) method [6], which achieves this by using distance queries on static configurations. The approach exploits that if, in a collision-free configuration c_1, the distance between two collision objects is d, and the relative displacement of these objects in configuration c_2 compared

to $c1$ is at most d, then these object do not collide in c_2 either.

The key difficulty with applying CA to industrial robots is that robot movements are defined in the joint configuration space, whereas the distance queries compute the allowed displacement in the Cartesian space. To overcome this discrepancy, an upper estimation of the Cartesian displacement caused by any given joint motion is required. This paper compares two implementations of this upper bound: the original bound from [6] and an improved bound. The bounds use the following information:

- $\varphi = (\varphi_0, \varphi_1, ..., \varphi_m)$: for an m axis industrial robot, the difference of joint angles between the start and end configurations of the motion.
- d_i, a_i: Denavit-Hartenberg parameters of the ith robot link (joint offsets in z and y).
- l_i: length of ith link from the lower joint node to the farthest point of the robot link geometry.
- r_i: distance of upper and lower joints of the ith robot link.

Based on these input data, an upper bound of $\delta_{i,j} = (\sum_{x=i}^{j}((r_x + l_x)\sum_{y=i}^{x}(\varphi_y)))$ can be given on the relative displacement of the ith and jth element of the kinematic chain.

Again, the appropriate choice and ordering of distance queries is crucial for computational efficiency. For this purpose, the proposed algorithm maintains a queue of so-called CA tasks, each CA task consisting of a pair of collision objects, as well as a start and end configuration of the motion. At every point in time, the queue is ordered by the length of the motion and the historic likelihood of collision between the two objects. Initially, the queue contains one CA task for each relevant object pair with the original start and end configurations of the motion.

In each iterative step, the first CA task is taken from the queue, and the distance of the two collision objects is computed in the mid-point of the motion, i.e., configuration $c(\frac{1}{2})$. If the query returns with a positive distance, then configurations $c(\frac{1}{2}^-)$ and $c(\frac{1}{2}^+)$, i.e., the first and last proven collision-free configurations before and after the mid-point are determined according to the above bound. If these are different from the start and end configurations of the original CA task, then two new CA tasks corresponding to $[c(0), c(\frac{1}{2}^-)]$ and $[c(\frac{1}{2}^+), c(1)]$ are created and inserted into the queue. The process is terminated when a collision is encountered or the queue is empty, where the latter means that the original movement is proven to be collision-free.

The accuracy of the upper bounds on the displacement greatly influences the number of distance queries executed. The original bound of [6] uses the bound for most distant objects in the kinematic chain for every pair of collision objects. The proposed minor improvement is to apply the bound $\delta_{i,j}$ corresponding to the specific objects.

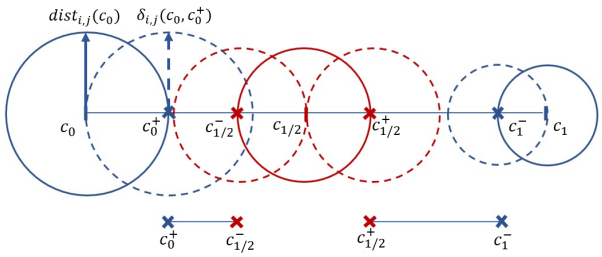

Figure 2: Conservative Advancement.

2.3 Comparison of the Two Approaches

The boolean collision queries used by the sampling-based approach are an order of magnitude faster than distance queries. CA can balance this difference by taking larger steps, and therefore executing less queries, especially in large open spaces. A crucial qualitative difference between the two approaches is that CA gives a formal guarantee of the geometrical feasibility of continuous robot movements. Moreover, CA can be naturally extended to maintain a specified safety distance between the objects in the work cell.

3. COMPUTATIONAL EXPERIMENTS

The presented algorithms were implemented in C# in MS Visual Studio. The solution contains separate projects for PQP (a native C++ project with C# wrapper), CollisionLibrary (a C# class library including UR5, UR10 robot models, RobotiQ and other grippers, implementations of collision detection, path planning, and path smoothing algorithms), CellVisualizer (a WPF application for the graphical animation of the work cell and the computed paths), as well as a console application for executing tests and measurements.

A reference solution was developed in the commercial robot simulation and off-line programming software called RoboDK [5] for verifying the correctness of the results and for comparing the computational performance to the state-of-the-art. RoboDK has a Python API to implement custom algorithms, and offers high-level functionality for simulating robot movements and collision detection. In the presented experiments, the `Move_JTest(configA, configB, samplFreq)` method of RoboDK was used, which implements a sampling-based approach for checking robot movements. It should be noted that this method finds every colliding object pair (although this information is not used later), whereas our implementation looks for the first collision only. Some difference in the performance of the two approaches may also stem from the difference of the publicly available UR5 robot geometry adopted in our implementation and the robot model applied in RoboDK. All experiments were performed on an Intel i5-4200M 2.5GHz dual-core CPU and 8GB RAM.

3.1 Experiment Design

The work cell used in the experiment contains a UR5 robot equipped with a Robotiq gripper, as well as a robot stand and a work table with fixtures for assembling a ball valve. The number of collision objects is 10, with 165 000 triangles altogether, resulting in 27 active collision rules. The experimental workspace has large collision-free spaces as ca. 80% of checked movements are collision-free. The rate of colliding and non-colliding movements greatly affects perfor-

mance, since the checking of colliding movements can be interrupted upon finding the first collision.

Computational experiments were performed on a set of 5000 continuous robot movements arising when building a PRM on the above work cell with 1000 random robot configurations and 5 neighbors per node. The average length of the robot movements was 45–50° in each robot joint.

Four different collision detection techniques were compared: sampling in RoboDK and in the proposed implementation, as well as CA in the proposed implementation with the original displacement bound of [6] and its improved version.

3.2 Experimental Results

The computational results are displayed in Table 1, which displays the key parameters, as well as the results achieved by the four algorithms. Both sampling-based approaches used a 1° sampling rate for the joint movements, without giving a formal guarantee of the geometrical feasibility of the checked motions or maintaining a safety distance. With this sampling rate, our implementation classified 2 out of 5000 colliding robot motions incorrectly as collision-free. A higher number of mistakes by RoboDK probably stems from the different geometrical models used.The efficient implementation resulted in a 23 times speedup compared to RoboDK.

In contrast, the two CA implementations both provided a guarantee of geometrical feasibility and could maintain a safety distance. At the same time, in order to facilitate a comparison between CA and sampling, a safety distance of 0 mm was used in the experiments. Moreover, allowing a relative tolerance of 3% in the PQP distance queries resulted in a considerable speedup of the algorithm, without any incorrect classifications on this test set. As a result, the two CA implementations returned correct and identical classifications. The improved displacement upper bound resulted in a 2.89 times speedup compared to the original upper bound, and computation times only 19% higher than for sampling. We regard this as a favorable tradeoff for the formal guarantee on the feasibility of the robot motions.

Table 1: Experimental Results

	Sampling (RoboDK)	Sampling (own)	CA (orig.)	CA (impr.)
Sampling	1°	1°	-	-
Safety dist.	-	-	0 mm	0 mm
Guarantee	-	-	✓	✓
Time [mm:ss]	38:08	01:41	05:47	02:00

4. CONCLUSIONS

The paper gave an overview of the computational challenges in continuous collision detection for articulated industrial robots, and presented alternative approaches to tackling this challenge. An efficient implementation of the sampling and the conservative advancement approaches was introduced, with various improvements compared to earlier algorithms in the literature. In computational experiments, the proposed sampling-based algorithm achieved a 23 times speedup compared to a similar algorithm of a commercial software, whereas an improved displacement bound for conservative advancement resulted in a nearly three times speedup w.r.t. using the earlier bound from the literature.

The presented collision detection library is a key component of a process planning and path planning toolbox for industrial robots under development. Future work will focus on the completion of the robotic path planning algorithms, especially PRM and RRT, on top of the presented collision detection library. We plan to apply this library to process planning in various industrial applications, including a camera-based robotic pick-and-place work cell and the assembly of electric components. A research challenge is the handling of constraints and performance measurements defined in the Cartesian task space, such as linear motions or Cartesian speed limits, while planning in the robot joint configuration space.

5. ACKNOWLEDGMENTS

This research has been supported by the ED_18-2-2018-0006 grant on “Research on prime exploitation of the potential provided by the industrial digitalisation” and the GINOP-2.3.2-15-2016-00002 grant on an “Industry 4.0 research and innovation center of excellence”. A. Kovács acknowledges the support of the János Bolyai Research Fellowship.

6. REFERENCES

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

[2] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 3719–3726, 2000.

[3] S. M. Lavalle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[4] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012.

[5] RoboDK. Simulation and OLP for robots, 2019. https://robodk.com/.

[6] F. Schwarzer, M. Saha, and J.-C. Latombe. *Exact Collision Checking of Robot Paths*, pages 25–41. Springer, 2004.