Collision and proximity avoidance for robust behaviour of real-time robot applications

Alexandru Dumitrache, Theodor Borangiu
University Politehnica of Bucharest, Romania
Centre for Research & Training in Robotics and CIM (CIMR)
{alex, borangiu}@cimr.pub.ro

***Abstract*—This paper proposes a set of techniques for predictive collision avoidance which ensure robust operation of robot applications. Implementation issues for applying the techniques to current state-of-art robots are presented, including integration with manual and automatic operation modes.**

***Index Terms*—Collision detection, robust operation, accidental damage prevention, robot motion prediction**

I. Introduction

Collision detection, also known as interference detection or contact determination, is a fundamental tool in computer games and animation, physical simulation robotics and computer-aided design and manufacturing (CAD/CAM). In these applications, solid bodies are not allowed to penetrate each other; here, collision detection automatically reports a geometric contact when it is about to occur or has already occured.

While higher-end physics simulations may consider deformable objects, collision detection in robotics and CAD/CAM can be successfully implemented using rigid bodies (of course, unless the robots manipulate deformable parts).

Solids involved in collision detection can be geometrical primitives (box, sphere, cylinder), polygonal meshes, algebraic surfaces or splines.

This problem has been extensively studied in the literature, and many algorithms are available for collision detection: there are general algorithms for dealing with arbitrary polygonal meshes with no particular structure (also called *polygon soups*), and particular algorithms, which exploit properties such as convexity or temporal coherence for faster queries.

Usually, the general notion of collision detection encompasses the following elements:

- *proximity detection*, which refers to the minimum distance between two solids
- *collision detection*, which detects whether two solid bodies touch each other
- *collision response*, which computes the changes in motion of the solid bodies after a collision

In 3D computer aided design (CAD), the following queries are usually performed:

- *clash* (intersection) detection: detect whether two bodies intersect each other
- *tolerance verification*: detecting whether two objects are closer than a given tolerance
- *distance computation*: computing the minimum distance between two objects

Traditionally collision detection (CD) was discrete: it tested overlapping between two static instances of moving objects; however, CD routines might ignore collisions between two fast moving objects (e.g., they may not notice a bullet passing through a narrow wall). In contrast, continuous collision detection techniques (CCD) are guaranteed to find the collision which has occured between two given static instances of the 3D scene, although they require more processing power than CD. Recent research efforts concentrate on optimizing continuous collision detection, and also on applying it to deformable objects, which is useful for more realistic simulations.

Two older surveys are avaliable in [1] and [2]; however, they only present non-continuous collision detection methods.

II. Case study: Reverse Engineering Platform

Collision avoidance techniques were studied in the context of a reverse engineering platform, which integrates:

- a short range laser scanning sensor
- a 6-DOF vertical robot arm, which sweeps the laser sensor around the scanned part
- a rotary table (1-DOF), which holds the scanned part
- a 4-axis CNC, able to create 3D parts from raw stock

The scanned part can be placed on the rotary table; therefore the scanning system is a redundant mechanism with 7 degrees of freedom (DOF). The extra DOF is used for optimal planning the motion of the rotary table, satisfying the following criteria:

- ensure a smooth motion of the rotary table, in order to allow the part to be placed without requiring any fixture

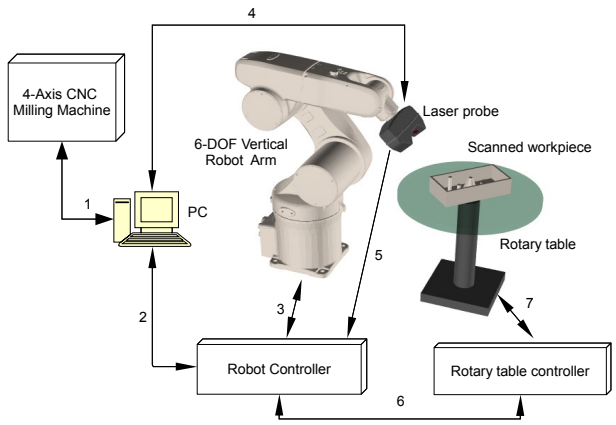

Fig. 1. Overview of the reverse engineering platform

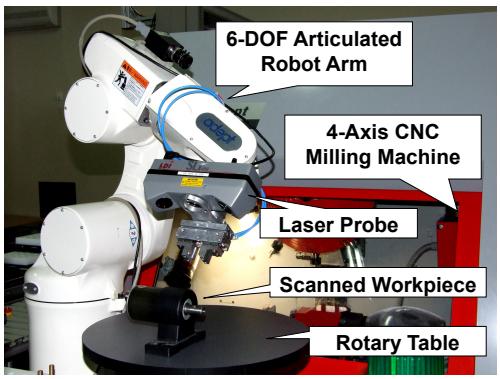

Fig. 2. The 3D scanning system

- avoid singular configurations in the robot arm, which would cause high joint speeds, low accuracy in scanning and unexpected end-effector movements
- avoid collisions between the following items:
  - laser probe with robot arm itself (Figure 3)
  - robot arm and laser probe, with rotary table
  - robot arm and laser probe, with the scanned part
  - robot arm and CNC milling machine
- minimize scanning time

In this scenario, collision detection is required at the motion planning stage, in order to make sure the robot motion, which is computed automatically by a planner, does not collide with any of the objects in the robot environment.

A real-time collision detection can be also employed during the use of the laser scanning system. In this way, the collision detection engine acts as an extra layer of protection, preventing accidental damage from erroneus user input or even from a programming mistake in the scanning strategy.

Finally, the CAM software component, which converts a 3D surface model into a set of CNC milling toolpaths, has to compute collision-free safe toolpaths, without the risk of damaging the milling cutter or the clamping fixtures. Since NC paths are generated as a series of 2D steps, collision avoidance is integrated into the toolpath generator without explicitely making queries of 3D rigid body intersections. An algorithm for generating collision-free milling paths with tool engagement control for arbitrarily complex raw stock and part geometry was proposed in [3].

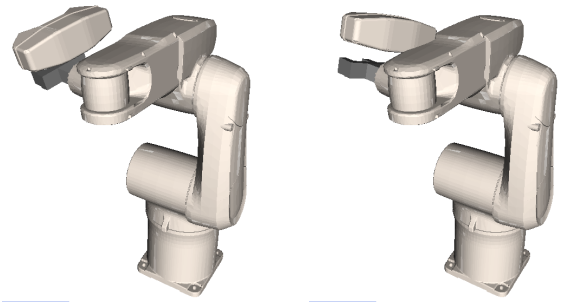

Fig. 3. Possible collision between laser sensor and robot body

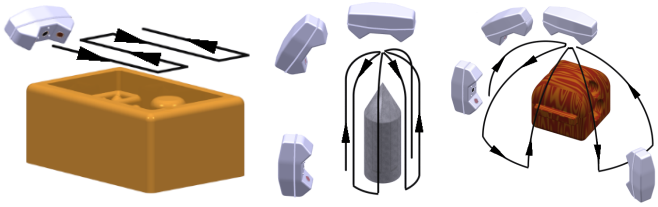

Fig. 4. Predefined scanning patterns: grid, cylindrical and spherical

III. Collision detection for robust operation

The 3D scanning module can be operated in two modes:

- From the manual control pendant (MCP) of the robot
- In automatic mode, where scanning trajectories are generated by the control software

These two modes are also employed in most robotic tasks.

In manual mode, the robot is usually moving at low speeds and the user is assumed to be careful not to cause collisions. However, a robust user interface shouldn't rely on correct user input; it should not allow the user to produce damage to the system no matter what the user input might be.

In automatic mode, the robot moves along parameterized scanning patterns (Figure 4), which also depend either on the user input, who may specify approximate part size, number of passes or scanning speed, or may rely on autodetection- However, users may make mistakes, and autodetection may produce incorrect results. It is difficult to predict when a certain combination of scanning parameters will lead to a collision or not; therefore, even after checking that all the parameters retrieved from user input are in the correct range, there is still possible that the scanning parameters may make the robot collide with the nearby equipment.

The robot controller supports very limited collision detection primitives, and only checks the end-effector point against maximum 4 obstacles, which can have one of the following shapes: axis-aligned box (AABB), cylinder, sphere or frustum [4]. These obstacles can only be specified in a robot configuration program, and cannot be changed while a user application is running on a robot.

These limitations make it very difficult to define obstacles which enclose tightly the robot, sensor and table in order to provide robust operation with respect to collisions.

A. *Collision detection during manual operation*

The proposed protection scheme is to have a dedicated task for realtime collision checking during robot operation. The protection task runs on the PC, continuously monitoring the robot position and velocity.

In manual mode, no user program is allowed to move the robot or change its speed, due to internal protection mechanisms implemented in the robot controller.

When the robot is heading to a colliding situation, the only actions that could be taken from a user program are:

- Give visual feedback on the MCP
- Give audible feedback to the user
- Assert the emergency stop signal (only in extreme cases)

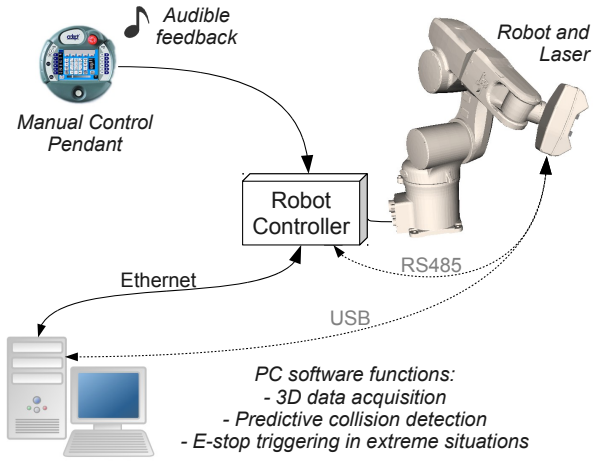

Fig. 5. Protection scheme for collision-free manual mode operation

An overview of the protection scheme is given in Figure 5.

The first option is useless if the user is not looking at the MCP, which is the most likely situation. The second option uses the control pendant's internal speaker to emit warning beeps, which change in intensity as the robot approaches an obstacle. If a collision is imminent, the only action allowed by the robot controller is to assert the emergency stop signal.

A more useful approach would have been to automatically reduce the robot speed when a collision is imminent; however, in order to implement this behaviour, safety mechanisms from the robot controller would have to be bypassed.

Warnings should be given only when two possibly colliding bodies become closer. When the user moves the robot away from the colliding situation, no warning should be given.

To implement this, the protection module should know also in which direction the robot is moving. User programs do not have access to the buttons pressed on the control pendant; this information is accessible only inside the MCP. However, in manual mode it is relatively easy to predict the robot motion with good accuracy, since the robot can be moved in a single direction at a time. The possible directions are:

- Cartesian translation (any direction in 3D space)
- End-effector rotation (around any fixed axis)
- Joint motion (rotate only one robot joint at a time)

Therefore, a predictive collision detection mechanism can be implemented. The predictor has to detect the type of motion:

- A joint interpolated motion
- A Cartesian motion (translation and/or rotation)

A model for describing and predicting joint-interpolated motions is given by:

$$J_i^{(t+1)} = J_i^{(t)} + s\ \delta j_i, \quad i = \overline{1,n} \tag{1}$$

where the robot has n independent joints, $J_i^{(t)}$ is the absolute position of i^th joint at time t, δj_i are weights which represent the relative speed of the i^th joint and s is the speed factor. This model represents general joint interpolated motions; however, in manual mode, only one joint is moving at a time. However, due to vibrations, all the joints will appear to be moving, but only one of them with a significant amount (let's say joint number k). Mathematically, $|\delta j_k| >> |\delta j_i|, i \neq k$ for a fixed k. However, this does not have any significant negative effects in the prediction process.

Therefore, the model parameters are $[\delta j_i],\ i = \overline{1,n}$. These parameters remain constant throughout the motion and can be identified by nonlinear minimization. In contrast, s may change freely throughout the motion due to acceleration.

Cartesian motions are described by linear interpolation in X, Y and Z, and spherical linear interpolations (slerp) in orientation. Therefore, the rotation axis remains constant throughout the Cartesian motion. A model for predicting linear motions in Cartesian space, where the end-effector is allowed to change its orientation, is:

$$\begin{aligned} X^{(t+1)} &= X^{(t)} + s\ \delta x \\ Y^{(t+1)} &= Y^{(t)} + s\ \delta y \\ Z^{(t+1)} &= Z^{(t)} + s\ \delta z \\ \theta^{(t+1)} &= \theta^{(t)} + s\ \delta\theta \\ R^{(t)} &= \mathcal{R}_{[rx,ry,rz]}(\theta^{(t)}) \cdot R^{(0)} \end{aligned} \tag{2}$$

where (X, Y, Z, R) is the Cartesian end-effector position and orientation (R is a 3×3 rotation matrix), $R^{(0)}$ is the initial end-effector orientation (at $t = 0$), and $[rx, ry, rz]$ is the rotation axis throughout the motion, which is constant.

The model parameters, which remain constant throughout the motion, are $[\delta x, \delta y, \delta z, \delta\theta, rx, ry, rz]$.

The decision for the motion type (Joint or Cartesian) is taken by trying to fit both models and select the one which gives lower residuals.

Transformations between Cartesian and Joint spaces are given by direct and inverse kinematics functions.

B. Collision detection during automatic operation

Even if the trajectory planner is programmed to generate collision-free paths, nobody can be sure that there are no programming mistakes in the robot application software. In semi-automatic modes, where the robot moves between points taught by the user, trajectory planning is performed solely on the robot controller, which does not check for collisions; however, collision checking can be done before sending a motion instruction to the robot. Of course, this assumes the robot program runs from the PC terminal.

The collision detection mechanism described in this section is designed to be as general as possible, in order to be useful regardless of the particular robot application. The implementation is a watchdog task, which analyzes the subsequent motion transparently, while the program is running. If a collision becomes imminent, the following actions can be taken:

- User feedback (visual or auditive)
- Gradually reduce monitor speed[1](this can be performed even while another program is running)
- Trigger the emergency stop (only in extreme cases)

[1]Monitor speed is a global setting of the robot, regardless of program speed

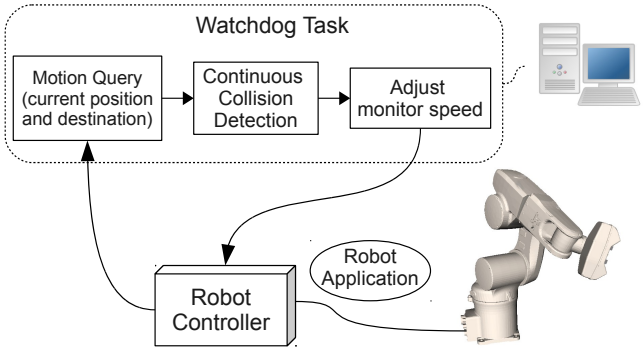

Fig. 6. Protection scheme for collision-free automatic operation

In automatic operation, only one program is normally allowed to *move* the robot. However, there may be additional program tasks which can *watch* the robot motion, i.e. read the current position within a loop, and also retrieve the destination of the current motion. Therefore, the watchdog task knows in advance the robot trajectory, and no prediction is necessary.

A schematic view of the protection scheme for automatic operation is given in Figure 6.

Experimental results: Robot position query can be performed from PC terminal every 32 milliseconds via Ethernet. A major cycle of the robot controller has 16 milliseconds.

However, when network delays occur, the PC would not have the possibility to slow down the robot quickly enough. Therefore, a protection mechanism has been employed: if the time since last message received from the watchdog is higher than normal, the monitor speed on robot decreases gradually, even until a full stop if network delays are very high. However, the method is not yet reliable for high speed operations.

IV. Collision detection for path planning

A heuristic planning algorithm for the robot arm and rotary table was presented in [5]. The 7-DOF mechanism (robot + rotary table) is redundant; this redundancy can be exploited to satisfy additional constraints. The angle of the rotary table, θ_R, may be chosen freely; once this angle is fixed, the remaining 6-DOF can be uniquely chosen from the possible inverse kinematics solutions.

The algorithm provides support for additional constraints specified as real-valued functions f_i, with $0 \leq f \leq 1$, which evaluate any static robot pose, with the following meaning:

- $f_i = 0$: the constraint is not satisfied
- $f_i = 1$: the constraint is fully satisfied
- $0 < f_i < 1$: the constraint is only partly satisfied

If there are many constraints, their functions can be multiplied, resulting a metric for evaluating any static robot configuration, with the same interpretation of values:

$$f = \prod_{i=1}^{m} f_i \tag{3}$$

where m is the number of constraints.

If only one constraint is not satisfied ($f_i = 0$), the specific robot configuration is avoided, since $f = 0$.

A set of constraints which keeps the robot away from its joint limits, ensuring a natural configuration, is:

$$f_j^*(\theta_j) = \left(\sin\left(\frac{\theta_j - \theta_j^{\min}}{\theta_j^{\max} - \theta_j^{\min}} \cdot \pi \right) \right)^{\gamma_j} \tag{4}$$

where $1 \leq j \leq 6$ is the joint number.

Static constraints can be represented graphically as graylevel configuration maps. An example is given in Figure 7 and 8, where the two dimensions of the map are the rotary angle θ_R and the discrete time t. The planner attempts to find a path from a start configuration (leftmost column in the map) to a final configuration (rightmost column in the map) which, while obeying all the constraints at least partly, has to be as smooth at possible and also have to obey the maximum angular speed and acceleration values for the rotary table. The robot motion is not constrained explicitely, but high speeds in robot motions can be avoided by adding static constraints which do not allow the robot to reach singular configurations.

Dynamic constraints for the rotary table are specified using scalar weights for angular speed and acceleration, k_a and k_ω. The heuristic algorithm, called *Ray Shooting*, attempts to try various constant-acceleration paths (rays) and selecting the one which has the lowest cost, at every time step. This strategy ensures low variations in the angular speed of the rotary table, which allows scanned parts to sit on the table without any additional fixture. The planning algorithm also attempts to reduce scanning time, energy consumption and increased scanning accuracy by avoiding near-singular configurations.

A constraint suitable for avoiding collision detection depends on the *minimal distance* between two rigid bodies, d_{min}. If d_{min} is less than a threshold d_{min}^{low}, the constraint is not satisfied, and this configuration is forbidden (the planner will never drive the robot through this configuration). If d_{min} is higher than d_{min}^{high}, the constraint is fully satisfied:

$$f_C(d_{min}) = \begin{cases} 0, d_{min} < d_{min}^{low} \\ sin\left(\frac{d_{min} - d_{min}^{low}}{d_{min}^{high} - d_{min}^{low}} \cdot \pi \right)^{\gamma_C} \\ 1, d_{min} > d_{min}^{high} \end{cases} \tag{5}$$

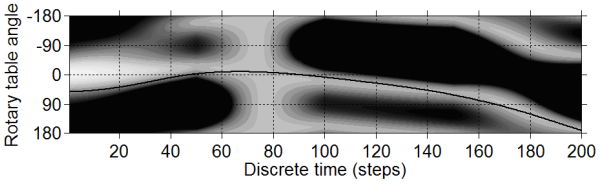

Fig. 7. Grayscale configuration map for static constraints

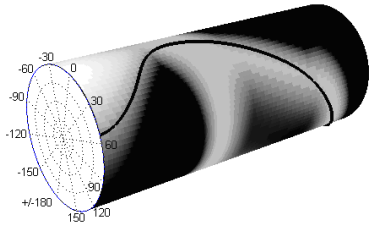

Fig. 8. Cylindrical view of the configuration map

The exponent γ_C controls the constraint intensity for d_{min} between $[d_{min}^{low}...d_{min}^{high}]$: higher values rejects values closer to d_{min}^{low}, while lower values are more permissive.

Of course, two successive bodies in the kinematic chain should not be tested for collisions, since they are always touching each other.

The parameters d_{min}^{low} and d_{min}^{high} can be chosen the same for every pair of possibly colliding bodies, or can be adjusted for each pair. For example, the distance between the laser sensor and 4^th robot link is by design 10 milimeters, so there's no point in setting a higher value for d_{min}^{high}. However, if one has to keep the distance between the laser sensor and the rotary table at least 20 mm, and preferably 50 mm, then d_{min}^{low} and d_{min}^{high} should be set to 20 and 50 respectively.

V. Implementation issues

The elements involved in collision detection are geometric models of their physical counterpart (Figure 9). For the robot arm, the meshes are imported from the CAD files. The rotary table has a very simple mechanical structure, and can be modelled as a stack of two cylinders. The laser sensor did not come with a 3D CAD file, but its shape was easily reconstructed from its dimensions, since it did not have a complex shape.

Other surrounding objects in the robot space can be digitized with the laser scanner itself. These objects include the CNC milling machine, the working table mounted near the robot arm and the mechanical parts placed on the table.

State-of-art collision detection algorithms are able to exploit the convexity of the geometries in order to reduce computation time and provide smooth real-time operation. Figure 10 shows the difference between the complete 3D robot model (left) and the one composed from the convex hulls of each segment (right). The number of triangles in each complete (and concave) mesh is 5000; the convex hulls have between 350 and 1200 faces. From a visual point of view, the meshes are not suitable from rendering, but would give satisfacatory results for collision detection even without the full meshes.

A *protective convex hull*, which completely encloses the underlying geometry and provides a *safety margin* around each link, was proposed in [6]. This also has the additional advantage that the triangle mesh can be simplified even more, for further speed increase.

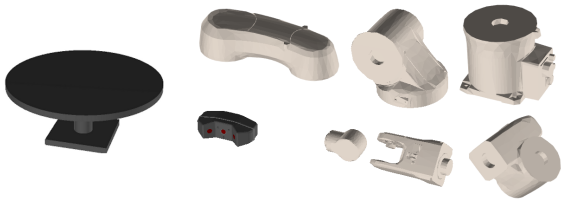

Fig. 9. Geometric models of the 3D scanning system components

Convex hull and mesh simplification were performed with the open source tool MeshLab [7].

Fast implementations use preliminary tests with simple geometric primitives, e.g. bounding spheres (which are easiest to check) and bounding boxes (which may approximate the true shape better). Bounding boxes can be aligned with the World axes (AABB[2]) or can have arbitrary orientation (OBB[3]), e.g. the one which minimizes the box volume. For articulated bodies, a hybrid approach can be used: the AABB is computed for each segment at the beginning of the simulation; then, while the articulated body moves, the AABBs are reoriented using the trasformation matrix for each link (Figure 10, right). In this way, only the box corners have to be transformed during the articulated motion. For pure AABB and OBB models, the bounding box would have to be computed at every step from a much larger data set, which would limit the usefulness of the bounding boxes.

Bounding boxes alone are not suitable for collision detection, since they may give many false positives. However, it is also possible to approximate a shape using hierarchies of AABBs, OBBs or bounding spheres.

A. Software implementations of collision detection

There are two main classes of pubicly available libraries which implement collision detection:

- rigid body dynamics simulators, used for computer games
- standalone libraries for collision and proximity queries.

[2]AABB - Axis Aligned Bounding Box
[3]OBB - Oriented Bounding Box

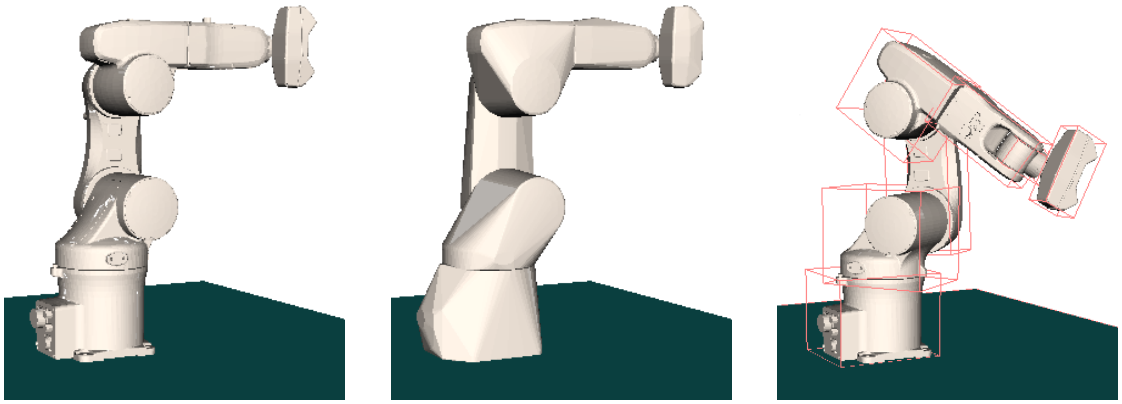

Fig. 10. 3D models for robot and laser probe: accurate (for rendering), convex and intermediate oriented boxes (for fast collision detection)

1) Rigid body dynamics simulation packages: There are numerous rigid body dynamics simulators which implement state-of-art collision detection algorithms. They may be used for collision queries alone; however, with a possible overhead.

Rigid body engines available under proprietary licenses include NVidia Physx (formerly known as AGEIA and Novodex), Intel HAVOK, Newton Game Dynamics and True Axis. There are also engines available under public licenses, such as BSD, ZLib and GPL, including Open Dynamics Engine (ODE), Bullet, JigLib and Tokamak. Most of the above engines can be wrapped in a unified abstraction system, like PAL, OPAL and GangstaWrapper. A comparative test among engines supported by PAL may be found in [8].

2) Libraries for discrete collision queries:

Limited to convex polyhedra:

- GJK - Gilbert, Johnson and Keerthi distance routine; runs in expected constant time [9]; used in Bullet Physics;
- I-COLLIDE: exact collision detection for large environments [10]; uses Lin-Canny Closest Features [11]
- SWIFT: supports also bodies composed of convex pieces. Queries: clash, distance and contact determination [12]

For non-convex polyhedra, also known as *polygon soup*:

- RAPID: uses OBBTree, a hierarchy of OBBs [13]
- PQP: intersection, distance and tolerance verification [14]
- V-COLLIDE: works with a large number of objects; uses 3 tests: *n-body*, hierarchical OBB and exact. [15]
- SWIFT++: for arbitrary polyhedral models. Queries: clash, tolerance, distance and contact determination [16]
- V-CLIP: Voronoi Clip algorithm; similar to Lin-Canny, with less complexity and improved robustness [17]
- OPCODE: memory-optimized AABB-tree [18]
- GIMPACT: implemented in ODE and Bullet.

3) Libraries for continuous collision detection (CCD):

- FAST: CCD for general, rigid polyhedra [19]
- CATCH: CCD for articulated models [20]; uses FAST for solid body CCD and SWIFT++ for distance queries

A proof-of-concept implementation of robot collision detection was implemented using Open Dynamics Engine as a wrapper for collision detection, which uses OPCODE and GIMPACT libraries. The simulation is based on the framework presented in [21], and it was implemented in Python, using PyODE for low-level ODE calls and cgkit (Python Computer Graphics Kit) for importing triangle meshes.

VI. Conclusion

The paper presented practical considerations about implementing a predictive collision detection routine, which improves robustness of existing robotic applications. A watchdog task runs on a PC workstation, continuously monitoring the robot motion, while having also knowledge of geometries of robot arm and nearby equipment. The watchdog intercepts the robot motion without disturbing the application program, using a prediction scheme when necessary. Collision detection queries for the entire scene run in 10 miliseconds on a Core2Duo CPU, therefore the bottleneck is the communication layer between the PC and robot. Future work will improve robustness and generality for high-speed robot applications.

Acknowledgment

This work is funded by the National Council for Scientific University Research, in the framework of the National Plan for Research, Development and Innovation, grant 69/2007.

References

[1] M. C. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *In Proc. of IMA Conference on Mathematics of Surfaces*, 1998, pp. 37–56.

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

[3] A. Dumitrache, T. Borangiu, and A. Dogar, "Automatic generation of milling toolpaths with tool engagement control for complex part geometry," preprint, 2010.

[4] Adept Technology, Inc., "Adept SmartMotion Developer's Guide," 2004.

[5] T. Borangiu, A. Dogar, and A. Dumitrache, "A heuristic approach for constrained real time motion planning of a redundant 7-dof mechanism for 3d laser scanning," in *INCOM 2009 - 13th IFAC Symposium on Information Control Problems in Manufacturing*, Moscow, Russia.

[6] J. Kuffner, K. Nishiwaki, S. Kagami, K. N. S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Self-collision detection and prevention for humanoid robots," in *in Proceedings of the IEEE International Conference on Robotics and Automation*, 2002, pp. 2265–2270.

[7] P. Cignoni, M. Corsini, and G. Ranzuglia, "Meshlab: an open-source 3d mesh processing system," *ERCIM News*, no. 73, pp. 45–46, April 2008.

[8] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. New York, NY, USA: ACM, 2007, pp. 281–288.

[9] G. Van den Bergen, "A fast and robust GJK implementation for collision detection of convex objects," *J. Graph. Tools*, vol. 4, no. 2, pp. 7–25, 1999.

[10] 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 *I3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*. New York, NY, USA: ACM, 1995, pp. 189–ff.

[11] B. V. Mirtich, "Impulse-based dynamic simulation of rigid body systems," Ph.D. dissertation, 1996.

[12] S. A. Ehmann and M. C. Lin, "Accelerated proximity queries between convex polyhedra by multi-level voronoi marching," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000, pp. 2101–2106.

[13] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: a hierarchical structure for rapid interference detection," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1996, pp. 171–180.

[14] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Tech. Rep., 1999.

[15] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, and D. Manocha, "V-collide: accelerated collision detection for vrml," in *VRML '97: Proceedings of the second symposium on Virtual reality modeling language*. New York, NY, USA: ACM, 1997, pp. 117–ff.

[16] S. A. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition," in *in Computer Graphics Forum*, 2001, pp. 500–510.

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

[18] P. Terdiman, "Memory-optimized bounding-volume hierarchies," March 2001.

[19] X. Zhang, M. Lee, and Y. J. Kim, "Interactive continuous collision detection for non-convex polyhedra," *Vis. Comput.*, vol. 22, no. 9, pp. 749–760, 2006.

[20] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, "Continuous collision detection for articulated models using taylor models and temporal culling," *Proceedings of SIGGRAPH 2007*, vol. 26, no. 3, p. 15, 2007.

[21] T. Borangiu, A. Dogar, and A. Dumitrache, "Modelling and simulation of short range 3D triangulation-based laser scanning system," *Int. Journal of Computers, Communications and Control (IJCCC)*, vol. 3, no. Suppl. Issue - ICCCC'08, pp. 190–195, 2008.