Real-time Swept Volume and Distance Computation for Self Collision Detection

Holger Täubig, Berthold Bäuml, and Udo Frese

***Abstract*— We present a real-time self collision detection algorithm applicable for industrial and humanoid robots. The algorithm is based on computing the swept volumes of all bodies and checking them pairwise for collisions. The algorithm operates on joint angle intervals. Such, it does not only test a single or N intermediate configurations but assures safety of a whole movement. Key idea of the new swept volume computation is representing volumes as convex hulls extended by a buffer radius, so called sphere swept convex hulls (SSCH). This leads to tight and compact bounding volumes. The operation set available to model the different joints is strictly conservative and allows for a trade-off between accuracy and computation time. During a configurable timespan the algorithm updates a table of pairwise distances and thus can guarantee hard real-time. It is applied on DLR's humanoid Justin in a sports robotic scenario, where also accuracy and computational performance is evaluated (0.4ms, INTEL T2500@2GHz).**

I. Introduction

Collision avoidance is a crucial task for humanoid and industrial robots. Preferably this task should be separated from cognitive modules such as path planning. Even though planning and executing a safe path is obviously related, a separate safety module that avoids collisions independently and is able to override the commands of all other modules provides essential advantages: all other modules can use simpler collision models or even apply unsafe collision avoidance strategies and developers of that modules can perform practical tests in early development stages without concerns. This architecture enhances safety and follows the established practice to concentrate safety concerns in one place. Following that idea our aim is a self-contained, reactive collision detection module. The module runs cyclically, taking the robot's state (joint angles and velocities but no planned trajectories) as input. It computes whether braking must be triggered or not because it is safe to wait one more cycle and then decide again whether to brake.

The task is particularly difficult at high speeds, where the safety module must handle significant braking distances geometrically precisely with limited computation time. We faced such a situation in our project B-Catch, where DLR's humanoid robot Justin catches two simultaneously thrown balls [1][2]. Thus we developed a strictly conservative continuous collision detection algorithm, presented in this paper, which handles large braking distances, safeguards the whole braking movement, and executes in real-time (0.4ms). It first computes the swept volumes of all bodies, i.e., the volume the body touches within its movement (Fig. 1). Then all pairs of swept volumes are checked for collision. The algorithm requires a kinematic model of the robot defining joint-frames and a geometrical model with the robot's rigid bodies represented in one joint-frame. It operates as follows:

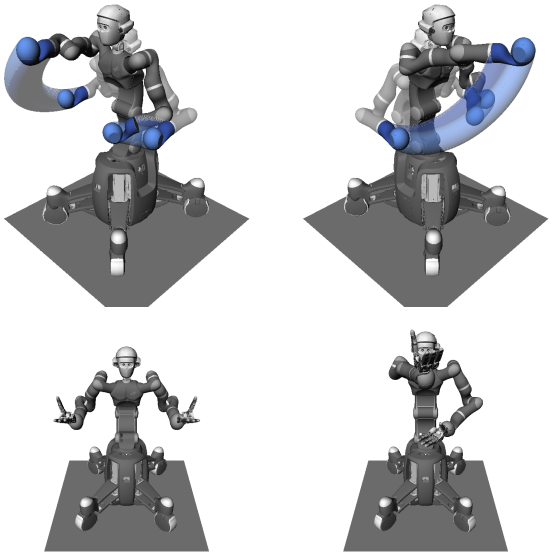


Fig. 1. Swept volumes of a large robot movement from the start to the stop configuration shown below; The movement is an illustration, a real braking trajectory is shorter. We use DLR's humanoid robot Justin for experiments and application. The hands are omitted here for better visibility.

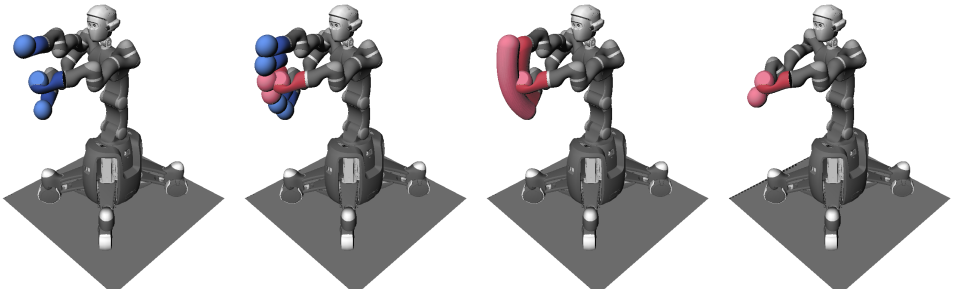
1) **Compute all joint intervals** $\mathcal{Q} = [q^0; q^1]$, such that when the robot starts braking the next cycle, it will stop within this interval. The intervals are based on joint angles, joint angle velocities, latency, and worst-case deceleration, as well as joint angle uncertainties.
2) **Compute swept volumes** $\mathcal{V}_k^i$ **of all bodies** B_i **in all joints** J_k **from the body down to the robot base by successively including the sweeping effect of one joint** J_k **after the other.** The swept volumes are represented in coordinates of the corresponding joint-frame C_k of joint J_k.
3) **For each body pair** (B_i, B_j)
   Compute the distance of $\mathcal{V}_k^i$ **and** $\mathcal{V}_k^j$ **in the first common joint-frame** C_k on the sequences of joints from B_i and B_j down to the robot base.
4) **Stop robot if any of the distances from 3) is zero.**[1]

[1]Alternatively, less or equal a configured *safety distance*.

Research supported by the DFG under Grant No. FR2620/1-1 (BCATCH).

H. Täubig and U. Frese are with Safe & Secure Cognitive Systems, Deutsches Forschungszentrum für künstliche Intelligenz, Bremen, Germany holger.taeubig@dfki.de, udo.frese@dfki.de

B. Bäuml is with DLR Institute of Robotics and Mechatronics, Münchnerstr. 20, 82234 Wessling, Germany berthold.baeuml@dlr.de



(a) (b) (c) (d)

Fig. 2. Different ways to check a motion for collisions: **(a)** Test start and stop configuration. **(b)** Test several intermediate configurations. **(c)** Test swept volume (as our algorithm does). **(d)** Determine the first configuration at which both bodies collide.

This paper contributes a strictly conservative and geometrically precise collision detection algorithm based on swept volume computation using an efficient, effective, and numerically stable representation. It is simple enough for a safety-related module and runs in strict real-time with a configurable computation time.

The paper is organized as follows: after related work in Sec. II, we derive the algorithm in Sec. III and extend it to real-time in Sec. IV. Thereafter, we present our application in Sec. V and evaluate accuracy and performance in Sec. VI.

II. Related Work

Collision detection in 3D has been studied in different contexts. A thorough overview of the methods from computer graphics, esp. 3D computer games, can, e.g., be found in the textbook [3], a particularly impressive system in [4]. Textbook methods often employ (a hierarchy of) geometrical primitives to represent or bound objects. Examples are axis aligned or oriented bounding boxes (AABB/OBB), spheres, sphere swept line segments (capsules), or sphere swept rectangles. By contrast, we represent volumes as sphere swept convex hulls (SSCH) of points, i.e., the convex hull of a finite set of points extended into all directions by the same buffer-radius. This representation can bound both angled and round objects with few points. Compared to the textbook methods it is more flexible, needs fewer bodies, and thus avoids the complex handling of a hierarchy as in [5][6][3, §6][4]. A hierarchy would be particularly complex here since our swept volumes are recomputed every cycle, not static as in [5] and the hierarchy would have to be updated. Instead we propose a distance updating scheme (Sec. IV) that is simpler and limits computation time to a given budget for real-time operation.

Unlike the textbook representation as a *union* of spheres, [7] and [8] propose to use the *convex hull* of spheres. This so-called s-tope is more general than our approach, as it allows different radii for different spheres. It more effectively represents conical objects but is much more difficult to handle. By contrast, in our representation computation of, e.g., a swept volume is just a simple formula (Sec. III-F).

In the context of collision avoidance for robotic applications range from pure distance computation [9], to path-planning of safe trajectories [10][11], and potential fields [12], where obstacles create a repulsive force. Approaches handling the robot motion can be discrete or continuous (cf. Fig. 2). Discrete approaches check for collisions at some intermediate configurations, potentially missing collisions in between ("tunneling problem") [3, §2.4.3]. Continuous collision detection approaches solve the tunneling problem and check the whole trajectory. Some, like ours, compute whether a collision occurs, e.g., by checking swept volumes, some determine the first colliding configuration [4].

Many systems, e.g., [7], [8], [11], and ours, use the GJK-algorithm [9] as a building block for computing the distance between two convex polyhedra given as arrays of points. Several optimizations have been proposed [13] which however rely on precomputation and do not pay off when the volumes are computed dynamically.

Swept volumes can be easily computed as voxel-grids [10]. This is precise and useful, e.g., to determine a machine's workspace, but too slow for real-time operation.

Our method[2] is the 3D generalization of a 2D collision detection algorithm we presented in [14][15]. There we used comparable techniques to compute dynamic safety zones that depend on the current motion of a vehicle. The focus laid on the safety properties of the algorithm, its implementation, and the braking model. We applied formal verification to achieve a certificate of compliance with IEC61508 SIL3 by the German TÜV. As a general result we showed that the techniques also applied here are appropriate to give safety guarantees in a conservative and strict mathematical sense.

III. Algorithm

We will now introduce the algorithm of swept volume computation and collision detection. We start introducing the algorithm using abstract sets and transformations, in particular the notion of the swept volume of a body in a frame. A major principle is the Minkowski idea of lifting operations to sets by taking the set of results for all (combinations of) inputs. Afterwards we present our concrete representation of volumes (Sec. III-D) and the set of operations each handling a single joint (Sec. III-E).

Beforehand some notation: Let B_i be a body, C_i a reference frame attached to B_i, J_i the joint with which B_i and C_i move, and q_i the corresponding entry of the joint-angle vector. The letter V denotes a volume, i.e., a subset of coordinate-vectors $\subset \mathbb{R}^3$ referring to a reference frame C_i, and $T_{j\leftarrow i}$

[2] A patent is pending at the German patent office (#102009006256.4-32).

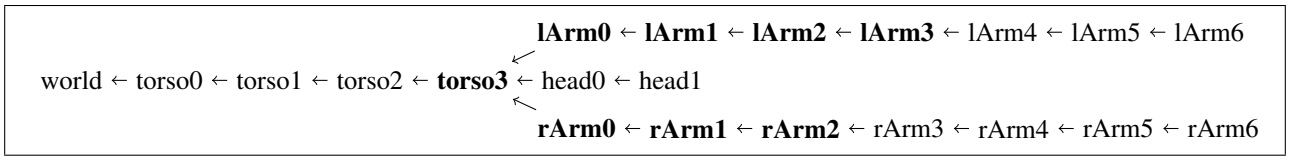


Fig. 3. Kinematic tree of Justin (20 joints) as an example. A node corresponds to a joint frame C_i, an edge $C_j \leftarrow C_i$ corresponds to the joint J_i. The relative pose of two bodies is affected by all joints on the path between the bodies' joint frames (shown for lArm3 and rArm2). The least-common-ancestor (lca) of both is the node (torso3) on that path closest to the world frame. The lca plays an important role in our algorithm, because the swept volumes of two bodies in their lca-frame are checked for collision. Therefor, a body B_i's swept volume is computed successively from C_i down to the world frame.

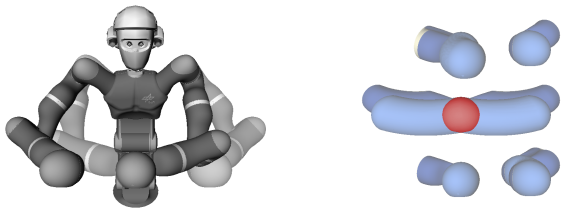


Fig. 4. A parallel motion of both arms (start and stop configuration left) and the swept volumes of the last links in the lArm6, torso3, and rArm6 frame (right, top to bottom; cf. kinematic tree in Fig. 3). In any reference frame, if the bodies collide, the swept volumes intersect. Conversely, if for a frame C_k the swept volumes of B_i and B_j intersect (e.g. in the torso-frame), both bodies touch the same point, relative to C_k, but not necessarily in the same moment. Potentially this is a false-alarm. However, if $k = i$ (resp. $k = j$) the swept volume of B_i (resp. B_j) is the body itself so an intersection of the swept volumes implies a collision (not the case here), without false-alarms.

the transformation matrix mapping C_i-coordinates to C_j-coordinates (at a single point in time). As usual, motion is described by time-varying transformations between frames. Finally, calligraphy letters ($\mathscr{V}, \mathscr{T}, \mathscr{Q}$) distinguish quantities of a whole motion from quantities of a single point in time.

A. Swept Volume of a Body in a Frame

Obviously, for a given point in time, two moving bodies B_i and B_j collide if they have a point in common; this holds regardless which reference frame is chosen, even if the reference frame moves. However, when the bodies' swept volumes are considered, the reference frame matters: A swept volume is only defined relative to a reference frame, i.e., it considers the relative motion between a body and a frame, e.g., another body (Fig. 4).

Formally, let $V_i^i \subset \mathbb{R}^3$ be the volume of body B_i in frame C_i. Then $T_{k\leftarrow i}$ is applied to V_i^i by transforming all points.

$$V_k^i = T_{k\leftarrow i} \cdot V_i^i = \left\{ T_{k\leftarrow i} \cdot p \middle| p \in V_i^i \right\} \tag{1}$$

This equation describes a static situation, i.e., at a single point in time. Bodies collide in all or no frames. Now, let $\mathscr{T}_{k\leftarrow i}$ be the set of $T_{k\leftarrow i}$ during the considered motion, then the swept volume of B_i in C_k is

$$\mathscr{V}_k^i = \bigcup_{T \in \mathscr{T}_{k\leftarrow i}} T \cdot V_i^i = \left\{ T \cdot p \middle| T \in \mathscr{T}_{k\leftarrow i}, p \in V_i^i \right\}. \tag{2}$$

If two swept volumes $\mathscr{V}_k^i$ and $\mathscr{V}_k^j$ intersect, the implications depend on the relation of

$$\mathscr{T}_{j\leftarrow i} \subseteq \mathscr{T}_{j\leftarrow k} \cdot \mathscr{T}_{k\leftarrow i}. \tag{3}$$

If both sides are equal, an intersection implies that the bodies B_i and B_j collide by the following implications

$$\mathscr{V}_k^i \cap \mathscr{V}_k^j \neq \emptyset \tag{4}$$

$$\Rightarrow \exists T_i \in \mathscr{T}_{k\leftarrow i}, T_j \in \mathscr{T}_{k\leftarrow j} \,:\, T_i \cdot V_i^i \cap T_j \cdot V_j^j \neq \emptyset \tag{5}$$

$$\Rightarrow \exists T_i \in \mathscr{T}_{k\leftarrow i}, T_j' \in \mathscr{T}_{j\leftarrow k} \,:\, T_i \cdot V_i^i \cap T_j'^{-1} \cdot V_j^j \neq \emptyset \tag{6}$$

$$\Rightarrow \exists T \in \mathscr{T}_{j\leftarrow i} \,:\, T \cdot V_i^i \cap V_j^j \neq \emptyset, \quad \text{as } T = T_j' T_i \in \mathscr{T}_{j\leftarrow i}. \tag{7}$$

As $\mathscr{T}$ describes motion as a function of time, (3) becomes an equality only if C_k is fixed relative to C_i or C_j, e.g., $k = i$ or $k = j$; then only one of the factors in $T_{j\leftarrow k} T_{k\leftarrow i}$ moves and the other one is static. So, ideally, to give the tightest bound possible, the algorithm should compute the swept volume of one body in the other bodies' frame and intersect both. We will however come back to that later.

B. Kinematic Tree

For a robot the motion of bodies is caused by joints which rotate (or more generally move) adjacent bodies relative to each other by a joint angle q. All joints together establish the robot's kinematic tree (Fig. 3). The kinematic-tree defines all transformations between frames at some point of time as a function of the joint-angle vector q, and the set of all transformations during a motion as a function of the set of joint angles $\mathscr{Q}$ during the motion.

$$T_{j\leftarrow i} := T_{j\leftarrow i}(q) \tag{8}$$

$$\mathscr{T}_{j\leftarrow i} := \left\{ T_{j\leftarrow i}(q) \middle| q \in \mathscr{Q} \right\} = \mathscr{T}_{j\leftarrow i}(\mathscr{Q}) \tag{9}$$

For two adjacent frames C_i and C_j the transformation $T_{j\leftarrow i}$ along the corresponding edge $C_j \leftarrow C_i$ in the kinematic tree is the movement of a single joint J_i and depends only on the joint angle entry q_i, i.e., $T_{j\leftarrow i}(q) = T_{j\leftarrow i}(q_i)$. Thus, for two arbitrary frames C_i and C_j, the transformation $T_{j\leftarrow i}$ is given by successively applying all transformations along the unique path $k_1 = i, \ldots, k_m = j$

$$T_{j\leftarrow i}(q) = \prod_{l=m-1}^{1} T_{k_{l+1}\leftarrow k_l}(q) \tag{10}$$

and depends only on the joint angles along this path (Fig. 3).

C. Separation of the Effect of Different Joints

A major principle of our algorithm is a Minkowski-view on configuration sets. Considering configurations and volumes this means applying every configuration on every point of the volume (see (2)). Considering the combination of joints J_i and J_j it means combining every configuration

$q_i \in \mathcal{Q}_i$ with every configuration $q_j \in \mathcal{Q}_j$. So, while the real set of configurations is a (braking) trajectory $\mathcal{Q} = \{q(t)|t\}$, our algorithm bounds it by a cross product of intervals

$$\mathcal{Q} := \bigotimes_i [q_i^0; q_i^1] \supset \{q(t)|t\}, \tag{11}$$

chosen to contain the considered braking motion. This introduces considerable bounding error (Fig. 5), but is conservative and essential for the algorithm as it allows to decouple joints and to apply their sweeping effect successively. Formally, now (3) becomes an equality, whenever C_k is on the path $C_i - C_j$ in the kinematic tree, because then the paths $C_i - C_k$ and $C_k - C_j$ affect different joints and are independent in $\mathcal{Q}$ by (11). Hence, for two bodies B_i and B_j we can choose any frame on the connecting path to compute and intersect swept volumes in, without introducing additional bounding error beyond (11). For our algorithm we chose the lca-frame (see Fig. 3) to intersect the swept volumes of two bodies, because then, the swept volumes for each body B_i can be successively computed from the bodies' frame C_i down to the world frame (Fig. 3). With $(k_1 = i, \ldots, k_m = \text{world})$ denoting the path from C_i down to the world frame this is done by

$$\mathcal{V}^i_{k_{j+1}} = \mathcal{T}_{k_{j+1} \leftarrow k_j}(\mathcal{Q}_{k_j}) \cdot \mathcal{V}^i_{k_j}, \quad j = 1, \ldots, m-1. \tag{12}$$

The importance of intersecting swept volumes in the lca-frame not in the world frame is shown in the supplementary video: A fast hip (torso0) rotation produces large swept volumes $\mathcal{V}^{\text{lArm6}}_{\text{world}}$ and $\mathcal{V}^{\text{rArm6}}_{\text{world}}$ for the hands. Still they do not collide because the hip rotation affects both hands the same way, so $\mathcal{V}^{\text{lArm6}}_{\text{torso3}}$ and $\mathcal{V}^{\text{rArm6}}_{\text{torso3}}$ do not intersect.

D. Sphere Swept Convex Hull Representation

We represent all volumes in terms of sphere swept convex hulls (SSCH) of a finite set of points $P = \{[p_l]_{l=1}^n\}$ defined as

$$\mathcal{V}(r; P) = \operatorname{conv} P + \left\{ b \in \mathbb{R}^3 \;\middle|\; |b| \le r \right\}, \tag{13}$$

where $\operatorname{conv} P$ is the convex hull of a point-set P. So each volume is the Minkowski-sum of a convex polyhedron given by a set of points, and a ball of radius r. Both, robot bodies and swept volumes are represented this way.

From a modeling point of view this representation is very flexible. It bounds both edged and round bodies tightly with few points. E.g., the 26 bodies of Justin's collision model (20 joints) only contain 80 points and 26 radii. A single bodie's representation uses only 3.1 points on average, less than 4 for a tetrahedron, the simplest polyhedral volume. Nevertheless, the collision model is tight and conservative (cf. Fig. 6 and video)[3]. Non-convex bodies have to be split into convex subparts which are treated as separate bodies, otherwise the algorithm safeguards their convex hull.

Furthermore, the representation is numerically good-natured as it does not use connectivity information, e.g., triangle meshes. This avoids typical problems of computational geometry involving degenerate triangles. Here, computation is simply computing the radius and the generating points.

[3]The joints torso1←torso2 and torso2←torso3 are coupled leading to 19 independent DOF. The hand volume bounds different hand configurations.

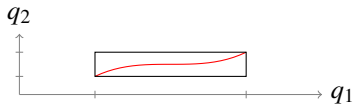


Fig. 5. Bounding the considered configurations $(q_1(t), q_2(t))$, e.g., a braking trajectory, by a cross-product of intervals. This approximation is over-conservative, as, e.g., the lower/right and upper/left corner of the rectangle are safeguarded by the algorithm but never reached by the robot.

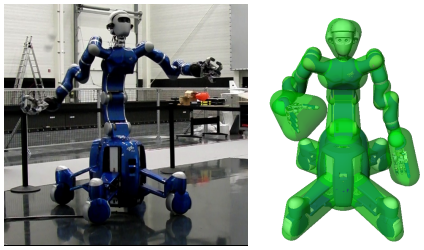


Fig. 6. DLR's mobile humanoid Justin and its collision model.

The GJK algorithm can easily be adopted to SSCHs. Originally it computes a lower bound of the distance between two convex hulls, given as a finite set of points P_i, P_j.

$$\begin{aligned} &GJK(P_i; P_j) \le d(\operatorname{conv} P_i, \operatorname{conv} P_j), \\ &\text{with } d(X,Y) = \min\left\{ |x-y| \;\middle|\; x \in X, y \in Y \right\} \end{aligned} \tag{14}$$

For details of this computation see the original paper [9]. A lower bound on the distance of SSCHs is given by the distance of the convex hulls itself minus both extension radii

$$GJK_r(r_i; P_i; r_j; P_j) := GJK(P_i; P_j) - r_i - r_j \tag{15}$$

$$\le d\left(V\left(r_i; P_i\right), V\left(r_j; P_j\right)\right), \tag{16}$$

because Minkowski addition of 0-centered balls with radius r_i and r_j in (13) reduces the distance by r_i and r_j.

E. Operations in the SSCH Representation

Each operation $\mathcal{V} \mapsto \mathcal{T}_{j \leftarrow i}(\mathcal{Q}_i) \cdot \mathcal{V}$ in (12) maps the effect of one joint J_i for a set of configurations $\mathcal{Q}_i$ while switching the frame from C_i to C_j. We will now propose concrete formulas that implement these operations for different kinds of joints by taking a volume $V\left(r; [p_l]_{l=1}^n\right)$ as input and computing a volume $V\left(r'; [p'_l]_{l=1}^n\right) \supset \mathcal{T}_{j \leftarrow i}(\mathcal{Q}_i) \cdot V\left(r; [p_l]_{l=1}^n\right)$ as output. Further, we present alternative operations to trade-off between accuracy and computation time.

Beforehand, we need some properties of the representation $V\left(r; [p_l]_{l=1}^n\right)$ allowing to implement operations on volumes by operations on their generating points.

$$V\left(r_1; V\left(r_2; [p_l]_{l=1}^n\right)\right) = V\left(r_1 + r_2; [p_l]_{l=1}^n\right) \tag{17}$$

$$T(q) \cdot V\left(r; [p_l]_{l=1}^n\right) = V\left(r; T(q) \cdot [p_l]_{l=1}^n\right) \tag{18}$$

This means, extending a volume by radius r_1 and r_2 one after another is equivalent to extending by radius $r_1 + r_2$ and a rigid body transformation is applied to a volume by transforming all generating points.

The general fact that the union of convex hulls is contained in the convex hull of the union

$$\operatorname{conv} X \cup \operatorname{conv} Y \subset \operatorname{conv}(X \cup Y) \tag{19}$$

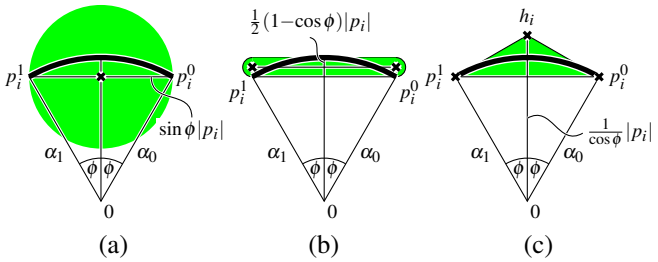


Fig. 7. Bounds of a circular arc by an SSCH with 1, 2, and 3 points.

establishes a unification rule for volumes

$$\bigcup_l V\left(r_l;P_l\right) \subset V\left(\max_l r_l;\bigcup_l P_l\right) \tag{20}$$

$$\mathscr{T}(\mathscr{Q})\cdot V\left(r;[p_l]_{l=1}^n\right) = \bigcup_{q\in\mathscr{Q}} V\left(r;T(q)\cdot[p_l]_{l=1}^n\right) \tag{21}$$

$$\subset V\left(r;\textstyle\bigcup_{l=1}^n \mathscr{T}(\mathscr{Q})\cdot\{p_l\}\right), \tag{22}$$

which shows the key safety property: the convex hull of the motion $\mathscr{T}(\mathscr{Q})\cdot\{p_l\}$ of the generating points (extended by r) bounds the swept volume $\mathscr{T}(\mathscr{Q})\cdot V\left(r;[p_l]_{l=1}^n\right)$ for that motion.

To simplify the presentation, we assume that each joint J_i defines an auxiliary frame $C_{i'}$, such that C_i and $C_{i'}$ coincide except for the joint-angle dependent motion, e.g., a pure rotation or translation, and $C_{i'}$ is fixed relative to the previous joint frame C_j defining the joints relative pose. $T_{j\leftarrow i'}$ in (12) is then handled by (18). For $\mathscr{T}_{i'\leftarrow i}(\mathscr{Q}_i)$ we will now derive concrete formulas based on (22). This means bounding the motion $\mathscr{T}(\mathscr{Q})\cdot\{p_l\}$ of a single generating point by an SSCH.

F. Revolute Joints

Revolute joints provide single axis rotation. The center of that rotation is the origin of both C_i and $C_{i'}$ and let a be the axis in C_i and $C_{i'}$. The configuration of a revolute joint is an angle α bounded by $\mathscr{Q}_i=[\alpha_0,\alpha_1]$ according to (11). We assume $|\alpha_1-\alpha_0|<\pi$. Hence, every point of the input volume moves on a circular arc within angle α_0 and α_1 around axis a. The arc (Fig. 7a) covered by one generating point p_l within $p_l^0=T_{i'\leftarrow i}(\alpha_0)p_l$ and $p_l^1=T_{i'\leftarrow i}(\alpha_1)p_l$ is contained in a disc, or actually in a ball, which is given by $V\left(\sin|\phi||p_l|;\frac{1}{2}(p_l^0+p_l^1)\right)$ with $\phi=\frac{\alpha_1-\alpha_0}{2}$. This bound uses one radius and one point. Having this bound for every generating point p_l according to (22) the overall effect of the joint is

$$\mathscr{T}_{i'\leftarrow i}(\mathscr{Q}_i)\cdot V\left(r;[p_l]_{l=1}^n\right)\subset V\left(r+\sin|\phi|\max_l|p_l|;[\tfrac{1}{2}(p_l^0+p_l^1)]_{l=1}^n\right) \tag{23}$$

Practically, operations compute a radius and a list of points:

$$\mathrm{Circ}_1\left(\alpha_0,\alpha_1,\left(r;[p_l]_{l=1}^n\right)\right)=\left(r+\sin|\phi|\max_l|p_l|;[\tfrac{1}{2}(p_l^0+p_l^1)]_{l=1}^n\right),$$
$$\text{with } \phi=\tfrac{\alpha_1-\alpha_0}{2}. \tag{24}$$

A tighter bound of a circular arc is obtained by using two points (Fig. 7b), formally $V\left(f|p_l|;p_l^0+fp_l^{1/2},p_l^1+fp_l^{1/2}\right)$ with $f=\frac{1-\cos\phi}{2}$ and $p_l^{1/2}=T_{i'\leftarrow i}(\alpha_0+\phi)p_l$. While this operation is more precise it doubles the number of points in the result:

$$\mathrm{Circ}_2\left(\alpha_0,\alpha_1,\left(r;[p_l]_{l=1}^n\right)\right)=\left(r+f\max_l|p_l|;[p_l^0+fp_l^{1/2},p_l^1+fp_l^{1/2}]_{l=1}^n\right)$$
$$\text{with } \phi=\tfrac{\alpha_1-\alpha_0}{2},\ f=\tfrac{1-\cos\phi}{2},\ p_l^{1/2}=T_{i'\leftarrow i}(\alpha_0+\phi)p_l \tag{25}$$

This happens for every joint along a kinematic chain where Circ_2 is used. Thus, choosing between Circ_1 and Circ_2 during configuration of the kinematic tree is a major aspect that allows to trade-off between precision and computation time. Nevertheless, all of the operations are purely conservative.

The choice of operations influences the modeling of motion due to the joints and not the modeling of the bodies themselves. In general Circ_1 is better for upper joints and Circ_2 for lower joints, because the latter have a larger lever and inertia resulting in a longer braking distance which must be bounded more tightly. For Justin we use Circ_2 up to (incl.) l/rArm2, i.e., the third arm joint and Circ_1 above that. This creates 928 points in 114 swept volumes for the 26 bodies.

A circular arc can also be bounded by a triangle (Fig. 7c) or even more generally by a $(S+2)$-gon by

$$\mathrm{Circ}_{S+2}\left(\alpha_0,\alpha_1,\left(r;[p_l]_{l=1}^n\right)\right)=\left(r;[p_l^0,p_l^1,[h_l^s]_{s=0}^{S-1}]_{l=1}^n\right) \tag{26}$$
$$\text{with } h_l^s=\tfrac{1}{\cos\triangle\alpha}T_{i'\leftarrow i}(\alpha_0+(2s+1)\triangle\alpha)p_l \text{ and } \triangle\alpha=\tfrac{\alpha_1-\alpha_0}{2S}$$

This operation uses no radius. It can arbitrarily precisely bound the *convex hull* of the swept volume. We used it in [14] for computing safety zones of a vehicle, where the factor in the number of points does not multiply over several joints.

G. Other Joints

Prismatic joints provide single axis sliding. Let a be the sliding direction in C_i and $C_{i'}$. The configuration interval $\mathscr{Q}_i=[d_0,d_1]$ bounds the translation of the input volume along a. Every generating point moves along a line from $p_l^0=T(d_0)p_l$ to $p_l^1=T(d_1)p_l$. This line can be bounded by one point in the middle and a radius of half its length $\left(V\left(\frac{1}{2}|d_1-d_0|;\frac{1}{2}(p_l^0+p_l^1)\right)\right)$, or by its two endpoints and no radius $\left(V\left(0;p_l^0,p_l^1\right)\right)$ [3, §9.5.7]. Again, the latter bound doubles the number of points:

$$\mathrm{Trans}_1\left(d_0,d_1,\left(r;[p_l]_{l=1}^n\right)\right)=\left(r+\tfrac{1}{2}|d_1-d_0|;[\tfrac{1}{2}(p_l^0+p_l^1)]_{l=1}^n\right) \tag{27}$$

$$\mathrm{Trans}_2\left(d_0,d_1,\left(r;[p_l]_{l=1}^n\right)\right)=\left(r;[p_l^0,p_l^1]_{l=1}^n\right) \tag{28}$$

Cylindrical joints can be composed of two joints. *Ball joints* could be built of three revolute joints but then singularities would occur. Thus, an extension of the circular arc approximation in Fig. 7 to patches of spheres should be prefered. *Mobile platforms* can be modeled as a rotation around the instantaneous center of rotation as in our previous 2D work [14]. Thinking of a joint more abstractly as a description of a relative movement of connected bodies, bodies moving on a *Bezier curve* are remarkable. As a Bezier curve is fully contained in the convex hull of its control points, an operation computing the control points of all of the volumes generating points would be a proper swept volume computation even for this more complex joint behaviour.

IV. Real-time Algorithm

The algorithm introduced so far is not strictly real-time, because the GJK algorithm takes an unknown number of iterations. By contrast, the swept volume computation needs a considerable but constant time. The algorithm in this section makes the distance computation also real-time.

A. Updating Pairwise Distance Bounds

The idea is to maintain a lower bound $D_{i,j}$ of the distance for every pair of bodies (B_i, B_j). In every cycle these bounds are updated according to the change in the swept volumes. Formally, this has quadratic computation time but in fact is very fast. Afterwards a configurable time budget is spent on improving bounds by GJK_r iterations. This is possible, as GJK_r needs an unknown number of iterations to converge but every iteration computes an increasingly tighter lower bound[4]. The following sub-algorithm replaces step 3 in Section I:

3a) **Compute changes Δ_k^i for swept volumes $\mathcal{V}_k^i$** (cf. (30)).

3b) **Update bounds $D_{i,j}$ for distance of $\mathcal{V}_k^i$ and $\mathcal{V}_k^j$** (cf. (31)). Here and below k is the lca-frame of i and j.

3c) **While time available and $D_{i,j} = 0$ for some i, j**
Update $D_{i,j}$ by GJK_r iteration on $\mathcal{V}_k^i$ and $\mathcal{V}_k^j$.

3d) **While time available**
Update $D_{i,j}$ by one GJK_r iteration on $\mathcal{V}_k^i$ and $\mathcal{V}_k^j$ with (i, j) going round-robin.

First, the algorithm reduces computation time as many body pairs are distant and need only be recalculated infrequently. For instance, imagine two bodies B_i and B_j moving around with 1cm/cycle at 1m distance but without actually coming closer. Then the distance bound $D_{i,j}$ reduces by 2cm each cycle and a GJK_r iteration is needed every 50 cycles. Second, the algorithm is adaptive: It tries as long as possible to verify that $D_{i,j} > 0 \; \forall i, j$ and the motion is safe. If it does not succeed braking is triggered, which may be a false-alarm. If the robot moves slower the algorithm has more cycles to verify $D_{i,j} > 0$ and may approve the motion.

Step 3d) makes sense in a real-time system, where it does not help if the computation time is *sometimes* lower. So the algorithm uses its remaining computation budget to "prepare for the future" by improving distance bounds. This postpones the need for a future GJK_r iteration and helps avoiding unnecessary braking.

In practice, we implemented "time available" by counting floating point operations (FLOPS) in GJK_r and configuring a budget for that. One could also use actual time, however causing mildly nondeterministic behavior.

B. Change of Swept Volumes and Their Distances

The change Δ_k^i of a swept volume since the last cycle is the maximum distance any point of that volume has changed, i.e., every point of the new swept volume $\mathcal{V}_k^{i\prime} \subset V\left(r'; [p_l']_{l=1}^n\right)$ must be within $\leq \Delta_k^i$ of the previous swept volume $\mathcal{V}_k^i \subset V\left(r; [p_l]_{l=1}^n\right)$:

$$V\left(r'; [p_l']_{l=1}^n\right) \subset V\left(r; [p_l]_{l=1}^n\right) + \left\{b \;\middle|\; |b| \leq \Delta_k^i\right\}, \tag{29}$$

$$\text{achieved by } \Delta_k^i = \max\left(\max_l |p_l - p_l'| + r' - r,\, 0\right) \tag{30}$$

Using that, a lower bound for the distance of a body pair is

$$D'_{i,j} \geq \max(D_{i,j} - \Delta_k^i - \Delta_k^j, 0), \tag{31}$$

with k the lca-frame of i and j.

[4]The lower bound can actually decrease after a GJK_r iteration. But then the previous one is still valid and kept.

V. Integration

A. Visualization

The proposed algorithm operates on volumes represented by a simple point array without connectivity information such as triangles. This avoids the well known problems with degenerate triangles. Nevertheless, for visualization $V\left(r; [p_l]_{l=1}^n\right)$ must be converted into a triangle-mesh. One possibility is to run a convex hull algorithm on $[p_l]_{l=1}^n$ and convert each vertex resp. edge to an r-sphere resp. r-cylinder. However, this is rather complicated. Instead, we take a spherical grid of unit vectors n_s and compute the support point in direction n_s. These support points

$$x_s = r\, n_s + \operatorname*{arg\,max}_{p \in \{[p_l]_{l=1}^n\}} p \cdot n_s \tag{32}$$

form a regular polyhedron approximating $V\left(r; [p_l]_{l=1}^n\right)$. It is not conservative but used for visualization and for computing the volume of $V\left(r; [p_l]_{l=1}^n\right)$ needed below.

B. Model Construction

It is time-consuming and error-prone to define a complex model such as the one for the humanoid Justin (Fig. 6) manually. Instead it was generated semi-automatically from an existing CAD model. We have manually divided the robot into bodies and decided how many points to use for each body. The points p_i and the radius r of each body $V\left(r; [p_l]_{l=1}^n\right)$ are then automatically fitted to contain all points $[p_l^*]_{l=1}^{n_*}$ from the CAD model in the lowest possible volume. As the fitted volume is convex, this automatically includes the convex hull of $[p_l^*]_{l=1}^{n_*}$. So it suffices to simply extract all vertices from the CAD model as $[p_l^*]_{l=1}^{n_*}$ and ignore edges and facets.

The optimization is a non-linear programming problem with the volume (computed from (32)) as cost function, constrained by

$$r \geq d\left(\{p_j^*\}, V\left(0; [p_l]_{l=1}^n\right)\right) \quad \forall j \tag{33}$$

$$\Leftrightarrow r \geq \max_j d\left(\{p_j^*\}, V\left(0; [p_l]_{l=1}^n\right)\right). \tag{34}$$

First, we obtain an initial guess by randomly choosing points from the input model and computing the radius by (34). This is repeated (1000×) and the smallest volume chosen. The randomization helps finding the *global* minimum.

We then follow a typical step-wise-linearization approach. In each iteration both the cost function and the constraints are linearized by taking numerical derivatives with a given step size. Then a linear-programming sub-algorithm (`lp_solve` library [16]) determines the linearized optimum within the step size region. This optimum is however not used directly but as direction for a non-linear line search (Brent's method). In the line-search the p_l are chosen according to the search direction while r is determined from (34) to fulfill all constraints. The change obtained by the line-search is then used as step-size in the next iteration.

The crucial point in this optimization is that each linearized iteration considers all constraints (33) simultaneously. Initially, we had tried an unconstraint optimization of the p_l with r solved by (34) as in the line search. This

approach converged poorly, because then only the current maximum constraint in (34) is visible to the optimizer.

VI. Performance Evaluation and Experiments

In the first experiment Justin moves its hands towards each other until the safety module triggers braking (see supplementary video). Then the commanded configuration for the position controller is fixed, so the robot brakes and returns to this configuration. For a single joint's state $(q,\dot{q})^T$ we determined an interval that safely bounds the reachable angle (step 1) by assuming a) a worst-case acceleration $a_{max} = 20\text{rad}/\text{s}^2$ during a latency $t_L = 10\text{ms}$ consisting of cycle time (10ms) and communication time (negligible) and b) braking of the joint thereafter with at least $a_B = 20\text{rad}/\text{s}^2$.

$$[q^0,q^1] = [\min(q,q^-,q^+),\max(q,q^-,q^+)] \tag{35}$$

$$q^\pm = q + \frac{\dot{q}+\dot{q}^\pm}{2}t_L + \frac{\dot{q}^\pm|\dot{q}^\pm|}{2a_B} \tag{36}$$

$$\dot{q}^\pm = \dot{q} \pm a_{max}t_L. \tag{37}$$

The results of this experiment (Fig. 8) demonstrate the safety of the approach and show occuring overestimations. Overestimation is not only caused by our algorithm but also by the safety margins in (35). A second experiment done in simulation using the same parameters (Fig. 9) further investigates that overestimation and shows properties of the real-time algorithm. The seven braking maneuvers that occured are listed in Table I. Computation time, in particular for the real-time algorithm is reported in Fig. 10.

The overestimation due to the latency t_L in (35) warrants discussion as it is present even at standstill (0.5cm in Fig. 8) and problematic for fine-manipulation. It is caused by a potential acceleration before the braking can be triggered at time t_L. This makes (35) rather conservative. If this needs to be avoided, the joint angle intervals could be made tighter by considering the actual robot dynamics, e.g., the current command, instead of a worst case acceleration.

Finally, the video shows that by using the lca-frame the algorithm avoids false-positive collision alerts. However, some motion constraints are not exploited and may cause false-positives, e.g., when one hand follows the other but moved by the wrist joints and not by some common joint.

VII. Conclusions and Future Work

We proposed a real-time continuous self collision detection algorithm based on swept volume computation in an application with a fast moving humanoid robot. It avoids collisions properly and achieves bounded computation times of 0.4ms (INTEL T2500@2GHz). Key to efficiency is a flexible but compact volume representation, the sphere swept convex hulls (SSCH). In particular, the feature of adding a buffer radius in all directions has also been exploited in modeling the robots kinematics to trade-off accuracy and computation time. The main contribution is a method to bound the braking motion strictly conservatively: First, by a cross-product of intervals in joint-angle space and second, by successively applying operators for each joint that compute the effect of one joint-angle interval onto a bodies swept volume. Thus, it combines safety and usability properties in a way that fulfilled all requirements on a self-contained collision avoidance module for fast robot motions.

In [14] we have already applied comparable techniques to 2D collision detection for vehicles. In the near future we are going to integrate both solutions which will yield a seamlessly integrated 2D/3D collision detection module. Further there is some potential for computational costs reduction which we currently explore: The change Δ in a swept volume could be bounded from the change in the joint-angle intervals, so the swept volume needs only be computed for a GJK_r update.

References

[1] B. Bäuml, F. Schmidt, T. Wimböck, O. Birbach, A. Dietrich, M. Fuchs, W. Friedl, U. Frese, C. Borst, M. Grebenstein, O. Eiberger, and G. Hirzinger, "Catching flying balls and preparing coffee: Mobile humanoid rollin'justin perfoms dynamic and sensitive tasks," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, 2011.

[2] B. Bäuml, O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger, "Catching flying balls with a mobile humanoid: System overview and design considerations," in *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, 2011, (submitted).

[3] C. Ericson, *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[4] X. Zhang, S. Redon, M. Lee, and Y. Kim, "Continuous collision detection for articulated models using Taylor models and temporal culling," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, vol. 26, no. 3, p. 15, 2007.

[5] M. Tang, Y. Kim, and D. Manocha, "C2A: Controlled conservative advancement for continuous collision detection of polygonal models," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan*, 2009, pp. 849–854.

[6] S. Redon, Y. Kim, M. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *Proc. of the 9th ACM Symposium on Solid Modeling and Applications*, 2004, pp. 145–156.

[7] E. J. Bernabeu, J. Tornero, and M. Tomizuka, "Collision prediction and avoidance amidst moving objects for trajectory planning applications," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[8] E. J. Bernabeu and J. Tornero, "Hough transform for distance computation and collision avoidance," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 3, 2002.

[9] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in 3D space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, 1988.

[10] J. Himmelstein, E. Ferré, and J.-P. Laumond, "Swept volume approximation of polygon soups," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA), Roma, Italy*, 2007, pp. 4854–4860.

[11] M. Gissler, C. Dornhege, B. Nebel, and M. Teschner, "Deformable proximity queries and their application in mobile manipulation planning," in *Proc. Symposium on Visual Computing ISVC, Las Vegas*, 2009, pp. 79–88.

[12] A. Dietrich, T. Wimböck, H. Täubig, A. Albu-Schäffer, and G. Hirzinger, "Extensions to reactive self-collision avoidance for torque and position controlled humanoids," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, 2011.

[13] C. Ong and E. Gilbert, "Fast versions of the Gilbert-Johnson-Keerthi distance algorithm: additional results and comparisons," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, 2001.

[14] H. Täubig, U. Frese, C. Hertzberg, C. Lüth, S. Mohr, E. Vorobev, and D. Walter, "Guaranteeing functional safety: Design for provability and computer-aided verification," *Autonomous Robots*, 2011, (submitted).

[15] D. Walter, H. Täubig, and C. Lüth, "Experiences in applying formal verification in robotics," in *SafeComp 2010 — 29th International Conference on Computer Safety, Reliability and Security*, ser. Lecture Notes in Computer Science. Springer, 2010, pp. 347– 360, 6351.

[16] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve reference guide," http://lpsolve.sourceforge.net/, 2010.

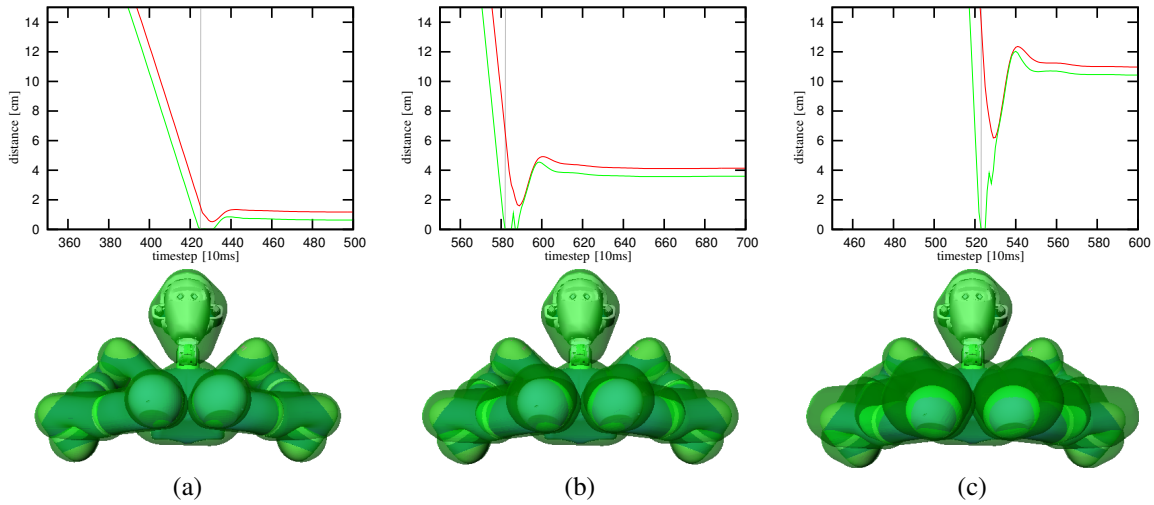


Fig. 8. Experiment of moving Justin's hands towards each other with **(a)** 0.22, **(b)** 0.65, and **(c)** 1.1m/s until the safety module brakes. **above:** distance in cm of Justin's hands (upper red line) and distance of their swept volumes (lower green line). The final distance is the same distance at which braking started (1.2, 4.1, resp. 11.0cm), because the robot returns to this configuration. The minimal distance during braking shows how much the safety module overestimates (0.5, 1.6, resp. 6.2cm at real braking distances of 0.7, 2.5, and 5.8cm). It contains overestimation of the swept volume and distance computation as well as safety margins in the interval computation (35), resp. in its parameters. The final difference of the two curves is caused by the latency t_L leading to $q^1 > q^0$ in (35) even for $\dot{q} = 0$. This safety distance (0.5cm) can be problematic for fine manipulation. **below:** colliding swept volumes at the time braking is triggered. The hands are omitted here for better visibility.

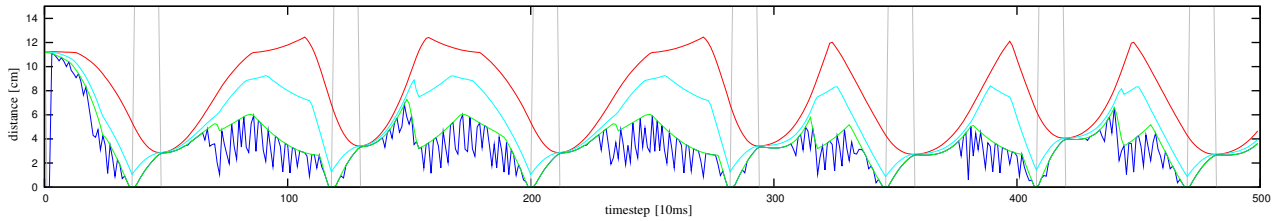


Fig. 9. Distance computation results in cm while repeatedly moving and stopping Justin in simulation ($\dot{q}$ up to 0.75π rad/s, time steps 10ms). **First**, compare the actual distance (top, red) with the swept volume distance (3rd, green, Sec. III). Their difference is the braking distance bounded by our algorithm. When the swept volume distance is zero (vertical line) braking is triggered, the red line showing the bounded braking distance in that moment. By comparing the actual distance here to the actual distance when stopped (next vertical line), the actual braking distance and the overestimation can be seen. A huge amount of the overestimation is caused by the latency t_L (avg. 1.7cm), not by the swept volume computation, as can be seen by the 2nd (cyan) line where $t_L = 0$. This effect is inherent to (35), but is increased by our swept volume computation as its overestimation grows with the length of the movement. **Second**, compare the plain algorithm (3rd, green) with the real-time algorithm (bottom, blue, Sec. IV, 55000 FLOPS). The values vary from cycle to cycle as the real-time algorithm only cares whether a distance is positive or zero. Typically a step down is a distance update by (31), a step up is an GJK_r iteration. Still the real-time algorithm succeeds in verifying safety, whenever the motion is safe, i.e., the swept volume distance (green) > 0.

bounded braking distance [cm]	5.8	7.5	5.8	7.5	4.6	6.3	4.6
actual braking distance [cm]	3.0	4.1	2.9	4.1	1.9	2.2	1.9
over-estimation [cm]	2.8	3.4	2.9	3.4	2.7	4.1	2.7
over-estimation factor	1.9	1.8	2.0	1.8	2.4	2.9	2.4
braking distance $t_L = 0$ [cm]	4.3	5.5	4.3	5.5	3.4	4.9	3.4
speed of approach (distance derivative) [cm/s]	40	65	40	65	34	49	34

TABLE I

THE SEVEN BRAKING MANEUVERS FROM FIG. 9, AVG. OVERESTIMATION 3.1CM.

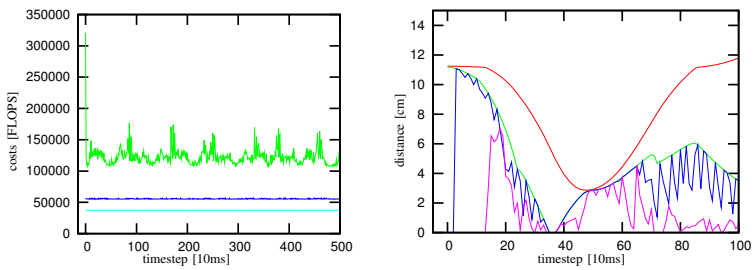


Fig. 10. **Left**: effort in FLOPS of the computations in Fig. 9; top to bottom: distance computation by plain (green) and real-time algorithm (blue), swept volume computation (cyan). The real-time algorithms effort (0.4ms on INTEL T2500@2GHz) is much more regular than the plain algorithm's (0.6ms to 1.3ms) and without a peak at the beginning. **Right**: first 100 cycles of Fig. 9, additionally with a computation budget of 30000 FLOPS (bottom, magenta). Plain and real-time algorithm behave different at the beginning. The plain algorithm initializes all distances by GJK_r iterations causing a peak in computation time. Both real-time variants spread this effort over, keeping the distance output at 0 (robot stopped) for the first 3 resp. 14 cycles. While with 55000 FLOPS the real-time algorithm behaves well, 30000 FLOPS are not enough to approve the robots motion, as the 30000 FLOPS variant (magenta) returns zero distance several times when the actual distance is positive. This would trigger unnecessary braking.