2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Adaptive Collision Checking for Continuous Robot Motions within Motion Constraints

Jinsung Kwon and Oussama Khatib

Abstract— **This paper presents an adaptive algorithm for checking collisions over any continuous robot motion set when tasks or constraints are given. As robots have begun to operate in human environments, which are unstructured and dynamically changing, the need for on-line robot planning and control strategies has increased. In implementing an on-line system, a fast and reliable collision checking method for continuous paths is a critical element. However, since external objects move unexpectedly, collision checking along the continuous path of a robot's motion suffers from increased uncertainty. Furthermore, computing the desired motion path or trajectory of a complex robotic task is very complex and slow. Therefore, we have developed a new collision checking strategy that can be applied to many types of motions that satisfy many given constraints. Our algorithm defines the applicable robot motions in a constraint-based manner, which is suitable for the multiple-task motion of a complex robot. This method can check the collision for the entire motion by finding the worst case with a small amount of computation, so that we can use the method for on-line applications. Moreover, our algorithm has a feature of adaptive resolution, which provides advantages in dynamically changing environments. The proposed method has been tested on high d.o.f. robots and the experimental results show that the method is suitable for on-line applications of multiple-tasks.**

I. INTRODUCTION

This paper presents an adaptive method for checking collisions over any continuous motion within provided motion constraints. This method is applicable to every robot motion that is restricted by multiple motion constraints. Since with this new method the continuous task trajectory of the conventional motion planning problem can be defined as one of the constraints, the method can also be applied to the conventional problems of trajectory tracking motion as well as the expanded cases of continuous range of motion, which includes the start configuration and the goal configuration. The range of motion is defined as the union of all configurations that satisfy the given motion constraints. Thus, this method provides a way to formulate robot motions in a general form.

As robots begin to operate in dynamic environments, where configurations of external objects change unexpectedly, collision checking along the continuous path of the robot's motion suffers from increased uncertainty. Since in dynamically changing situations the motion path keeps being modified, the robot's path tracking produces larger errors than in pre-defined and closed work spaces. Furthermore, it can be very complex and expensive to compute the desired motion path or trajectory of a complex robotic task. For example, without simulating and recording the given motions beforehand, it is not possible to compute the desired joint trajectory between two configurations of a complex robot performing multiple tasks. As a result, the strategy of detecting collisions over a continuous motion path in real time can only be applied to very restricted situations in which simple robots are used. Therefore, we have developed a new collision checking strategy which can be applied to more types of motions that satisfy many constraints.

Since our method defines the applicable motion in a constraint-based manner, not in a trajectory-based manner, a low-level motion controller has more flexibility in executing the task motion. Thus, the actual robot is able to react to sudden events in real time by running an additional reactive task within the boundaries of the previously provided constraints, since the method allows every motion that complies with the original constraints. This is a very useful feature for improving the safety of a robot system in dynamic environments. Furthermore, the definition of constraint-based motion is more suitable to the multiple-task framework that is frequently used for high degree of freedom (d.o.f.) robots.

One of frequently used approaches is the sample-based method, which samples configurations along a trajectory and checks collisions on the discrete samples with static collision-check algorithms such as [1], [2], [3], [4]. However, the sample-based methods are not able to guarantee safety over the continuous motions between the sampled configurations.

Thus, other algorithms have been developed to detect collisions along continuous motions, such as swept-volume methods [5], [6], [7], trajectory parameterization methods [8], [9], feature-tracking methods [10], [11], [12], and dynamic-envelope method [13]. However, those methods are applicable only to simple geometries or limited types of trajectories.

The collision checking method proposed in this paper is suitable to implementing on-line motion planning frameworks such as the elastic strip framework [14]. The elastic strip framework is an on-line motion modification strategy that can maintain a feasible motion in unpredictably changing situations. In this framework, a fast and reliable collision-checking method for continuous motions connecting discrete configurations is a critical element in implementing real-time systems. Our collision checking method is suitable to such real-time applications since the geometric or kinematic complexity does not significantly affect the load of computation, and the method can promptly check the collision for the entire target motion by finding the worst case configuration

Jinsung Kwon and Oussama Khatib are with Artificial Intelligence Laboratory, Stanford University jinskwon@stanford.edu, ok@cs.stanford.edu

with a small amount of computation. The elastic strip framework uses more numbers of robot configurations in a narrow space than in a wide space in order to increase the resolution of the collision checking method. By this way, the planning system can adapt the total amount of computation loads to the complexity of the environment.

The purpose of this paper is to present a collision-checking method that is able to test continuous motions. In order to detect a collision, the collision-checking method uses a proximity function that computes the minimum distance between the discrete configurations of two objects. However, this paper does not aim to improve the performance of a proximity function, but to present a new strategy to use the result of a proximity function. Hence, the collision-checking algorithm is able to adopt any kinds of existing proximity functions.

II. ALGORITHM

A. Collision Checking on a Motion Set

The most important issue with regard to on-line collision checking of a moving object is how to efficiently detect collisions over continuous motions. However, it is not a simple task to detect collisions over an infinite number of motions without false-negative (false non-collision) results and with a performance of real-time speed.

The objective of our algorithm is to detect a collision between obstacles, $\{O_1, ..., O_m\}$, and all configurations of a motion set, $S(q_i, q_{i+1})$, whenever there exist overlaps between them. In our algorithm, a motion set is regarded as a set of continuous motions that include and connect two discrete configurations. Thus, the algorithm should return true only when all configurations in the motion set do not have any overlap with the obstacles, and it should return false when it cannot confirm that there is no overlap with the current resolution. With this approach, the collision-checking algorithm should be able to efficiently decide whether it is free of collision when a robot moves in a motion set.

A motion set is defined by a set of constraints that are parameterized by the start and goal configurations. Thus, we can adapt the motion constraints and the corresponding motion set to changing environments by updating the start and goal configurations. As a result, any continuous configurations in the current motion set satisfy all the constraints related to the current situation. Let $C_k(q_i, q_{i+1}, q) \geq 0$, $k = 1, \cdots, m$ be m constraints that are defined by the given start and goal configurations q_i, q_{i+1}, and restrict the range of valid configuration q. Therefore, when q_i and q_{i+1} are given, a motion set is defined as

$$S(q_i, q_{i+1}) = \{\ q \mid C_k(q_i, q_{i+1}, q) \geq 0\ ,\ k = 1, \cdots, m\} \tag{1}$$

Since the motion set includes the start and goal configurations q_i, q_{i+1}, the constraints C_k, for every k, must satisfy

$$C_k(q_i, q_{i+1}, q_i) \geq 0\ ,\quad C_k(q_i, q_{i+1}, q_{i+1}) \geq 0 \tag{2}$$

Our approach to detecting collisions over the infinite number of configurations included in a motion set is to define a common collision-test method for a configuration and find the worst-case configuration with respect to the test method. The collision-testing method for a configuration that we use in our algorithm is to find at least one collision-free path that begins at the start configuration, ends at the goal configuration, and passes through the tested configuration. If a valid path is found, the collision-free path guarantees that the tested configuration is also free of collision as shown in Figure 1.

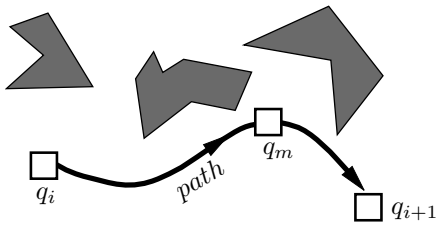

Fig. 1: Basic method for checking collision on an intermediate configuration: An intermediate configuration, q_m, is collision free if there exists at least one collision-free path which contains q_m.

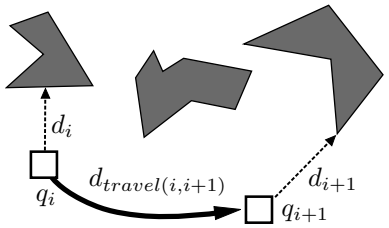

Fig. 2: Basic method for checking collision on a path: A path is collision free if the travel distance, $d_{travel(i,i+1)}$, is shorter than the sum of the two distances from the both ends' configurations to the obstacles such as $d_{travel(i,i+1)} < d_i + d_{i+1}$.

The method for testing collisions on a path we use is to compute the travel distance along the path and compare it with the sum of the distances from the configurations of the both ends to the closest obstacles, as introduced by Schwarzer, et al. [15]. If the travel distance is less than the sum of the obstacle distances, it is guaranteed that the path is free of collision. As shown in Figure 2, let d_i be the minimum distance from a robot body at a configuration q_i to the surrounding obstacles, and let $d_{travel(i,i+1)}$ be the travel distance along a test path that connects q_i and q_{i+1}. Then every configuration along the path is free of collision if

$$d_{travel(i,i+1)} < d_i + d_{i+1} \tag{3}$$

According to this test condition (3), a configuration q_m between q_i and q_{i+1} has a lower possibility of colliding with the obstacles if the test path which passes q_m has a shorter travel distance, $d_{travel(i,i+1)}$. For example, in the case of translation-only motion in the two-dimensional space, the straight line has the shortest travel distance and the path along that line has a higher possibility of satisfying the test

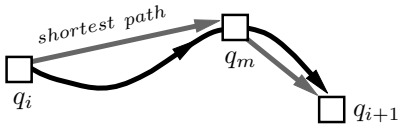

Fig. 3: Selecting the best *test path* for an intermediate configuration: Since a path which has the shorter travel distance can more easily pass the path collision test of condition (3), it is recommended to select a path which is as short as possible. For example, a straight line has the shortest travel distance when only translating motion is allowed.

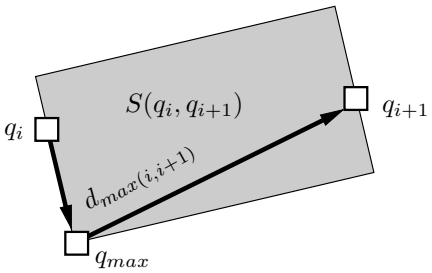

Fig. 4: Collision checking on the entire configurations in a motion set: Since d_i and d_{i+1} are common for every intermediate configuration, $q_m \in S(q_i, q_{i+1})$, in travel-distance computation, the collision checking on the entire intermediate configurations amounts to finding the extreme-case configuration, q_{max}, which has the test path of the longest travel distance. Therefore, $d_{travel(i,i+1)}(q_{max}) \equiv d_{max(i,i+1)} \geq d_{travel(i,i+1)}(q_m)$.

condition (3) and being confirmed as a collision-free path than any other possible path passing through the intermediate test points, as shown in Figure 3. Therefore, the first step is to determine the type of test path that has a small travel distance for each tested configuration.

The entire computation during the run-time amounts to the computation load for searching for the worst-case configuration in the motion set. After determining the type of test path, we need to formularize the function to compute the travel distance in order to find the worst-case configuration, q_{max}, that has the longest travel distance $d_{travel,max}$ along the test path. In this case, the efficiency of finding the worst-case configuration is important for the overall speed of the algorithm. The final collision-test is done using the inequality:

$$d_{max(i,i+1)} < d_i + d_{i+1} \tag{4}$$

It is worth noting that the collision-checking method can be applied to each of the robot's rigid bodies independently. If it is independently proved that every rigid body is free of collision within its corresponding motion set, the whole robot body is free of collision. Therefore, we don't need to consider the joint constraints during the collision-checking process. This fact enables us to choose a simple test path, with which it is easy to compute the travel distance. The only conditions to consider in choosing a test path are that the path should include q_i and q_{i+1}, and it should be included in the motion set, $S(q_i, q_{i+1})$.

B. Adapting a Resolution to the Width of Free Space

In our collision-detection algorithm, the resolution is adaptively determined in accordance with the free-space width near the motion set that is currently tested. The resolution of the algorithm means the level of free-space detail inspected by the collision tests. A motion planner that uses our collision checking algorithm can increase the resolution by adding more number of nodes to capture the collision-free motion in the tested region.

The result of the collision-test algorithm agrees more with the actual collision status as the resolution becomes higher. Since the algorithm does not produce false-negative results, all configurations of a motion set are free of collision when the algorithm returns true. However, when the return value is false, the result can be interpreted as an indication that the algorithm is not able to guarantee no collision in the motion set since the resolution is not fine enough. Therefore, a motion planner can inspect the motion-set more specifically by inserting more nodes into the current motion set. If any of the new nodes have a robot configuration that collides with obstacles, the motion set is confirmed to have collisions since an actual collision is found. Or if no collision is found along the continuous motion in the motion set, the entire motion set is considered free of collision. Otherwise, whether there are collisions or not has not yet been confirmed and the motion planner needs to determine whether to increase the resolution further or stop the free-space inspection at this level of resolution.

When the resolution is refined, the current motion set, $S(q_i, q_{i+1})$ is sub-divided into two smaller motion sets of $S(q_i, q_j)$ and $S(q_j, q_{i+1})$. The new configuration q_j, which divides the original motion set into those smaller motion sets, should be contained by the original motion-set such as

$$q_j \in S(q_i, q_{i+1}) \tag{5}$$

Once the motion-set is divided, the original collision-checking condition (4) is replaced by the two separate conditions:

$$\begin{aligned} d_{max(i,j)} &< d_i + d_j \\ d_{max(j,i+1)} &< d_j + d_{i+1} \end{aligned} \tag{6}$$

If a motion planner chooses to conserve the range of the initial motion-set after a sub-division, which is

$$S(q_i, q_j) \cup S(q_j, q_{i+1}) = S(q_i, q_{i+1}) \tag{7}$$

, then the maximum travel distances of the sub-divided motion-sets have a positive lower-bound d_{bound}. Let k be the number of sub-divisions, S_k be a k-times sub-divided motion-set, $V(S_k)$ be the volume of a motion-set S_k, and $d_{max,k}$ be the maximum travel distance of S_k, then

$$V(S_k) \to 0,\ d_{max,k} \to d_{bound} > 0 \quad as\ k \to \infty \tag{8}$$

Otherwise, if a motion planner chooses to make the maximum travel distance converge to 0 as the resolution

keeps being refined, the condition (7) needs to be relaxed as

$$S(q_i, q_j) \cup S(q_j, q_{i+1}) \subseteq S(q_i, q_{i+1}) \tag{9}$$

In this case, if obstacles approach to the motion path and the free-space becomes narrow, then the overall motion-set range is reduced by the sub-divisions. As a result, the allowed robot-motion range shrinks and converses to a collision-avoidance motion.

If the collision checking of the original motion set, $S(q_i, q_{i+1})$, returns the value "false" and the newly added configuration q_j is found to collide with obstacles, we have a case of an actual collision after increasing the resolution. Or if all the collision checks on both of the smaller sets return the value "true", the motion set is confirmed to be free of collision at the higher resolution.

The feature of adaptive resolution provides advantages when it is used in dynamically changing environments. When the surrounding area becomes narrower, a motion planner can increase the resolution and inspect collisions of motion sets with higher accuracy so that the method does not fail to find a collision-free path. On other hand, a motion planner can increase the speed of the inspection by reducing the resolution in a wider region. By using this approach, a motion planner can concentrate the computation resource more on narrower regions than on wider regions so that the algorithm can increase its accuracy as well as the efficiency in changing environments.

III. EXPERIMENTS

The collision checking algorithm has been developed for real-time applications in dynamic environments, for continuous motions of high d.o.f. complex robots, and for frequently changing, multiple tasks. Therefore, we have performed various experiments designed to test on-line updating performance, multiple task capability, and efficient and adaptive computation capability.

A. Performance in On-line Updating

Using a motion-modification approach in dynamically changing situations requires updating the path in a very short time period. Moreover, the computation time of the collision checking algorithm over the whole motion path is a critical part of the motion-modification process. Thus, we have measured the computation time of the algorithm for different types of manipulation robots when obstacles move around the robots at various distances.

For the first experiment, we used a 7 d.o.f. Kuka robot arm. The robot's task is to move from an initial posture to a goal posture. Three motion-sets are implemented to check collisions along the task motion; three spherical motion-ranges constrain the orientations of the upper arm, the lower arm, and the end-effector, respectively, as shown in Figure 7. The spherical motion-set defines boundaries of a direction $\vec{e}$ according to the global coordinates by using a margin angle ω, as in Figure 5. The maximum travel-angle in a spherical motion-set boundary is computed as shown in Figure 6.

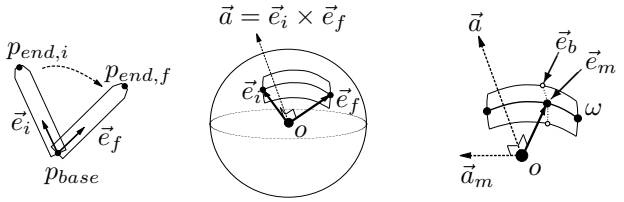

Fig. 5: A spherical motion-constraint, which defines a motion-set boundary of the direction of a body in global coordinates. A direction vector $\vec{e}$ is defined by the global positions of two points on the bodies as in $\vec{e} = \frac{p_{end} - p_{base}}{|p_{end} - p_{base}|}$. The initial direction $\vec{e}_i$ and the final direction $\vec{e}_f$ can be indicated on a unit sphere as shown on the middle. The motion-set boundary is also indicated on the unit sphere: the main axis $\vec{a}$ is the rotation axis of $\vec{e}_i$ and $\vec{e}_f$, defined as $\vec{a} = \vec{e}_i \times \vec{e}_f$. The boundary on the unit sphere is composed of the vectors of $\vec{e}_b$ as shown on the right. When $\vec{a}_m = \vec{e}_m \times \vec{a}$ for a vector $\vec{e}_m$ on the middle line, a boundary vector $\vec{e}_b$ can be constructed by rotating $\vec{e}_m$ along $\vec{a}_m$ by a predefined margin angle ω.

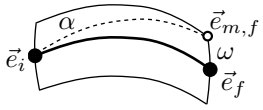

Fig. 6: The maximum travel-angle in a spherical motion-set. The travel angle is longest when the direction-point travels through one of points at the four corners. For instance, if $\vec{e}_{m,f}$ is the vector that is rotated from $\vec{e}_f$ by the margin angle ω, and α is the angle between $\vec{e}_i$ and $\vec{e}_{m,f}$ such as $\alpha = \angle(\vec{e}_i, \vec{e}_{m,f})$, then the maximum travel-angle is $(\alpha + \omega)$.

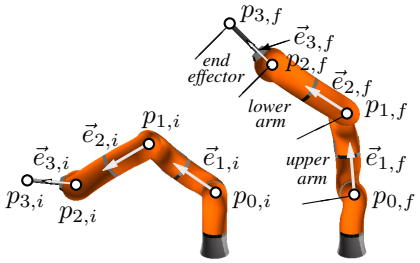

Fig. 7: Motion-sets placed on the Kuka arm. Three spherical motion-sets are configured on the 7 d.o.f. manipulator. Four positions of $p_0, \cdots, p_3$ on the robot body define the three unit-vectors of $\vec{e}_1, \cdots, \vec{e}_3$, each of which defines a motion-set on a unit sphere.

The maximum travel-angle $\theta_{max,ua}$ and maximum travel-distance $d_{max,ua}$ of the upper arm are computed as

$$\theta_{max,ua} = \alpha_{ua} + \omega_{ua} \tag{10}$$

$$d_{max,ua} = L_{max,ua}\theta_{max,ua} \tag{11}$$

where α_{ua} is the angle between the initial direction $\vec{e}_i$ and the intermediate direction $\vec{e}_{m,f}$ of the upper arm, as in Figure 6, and $L_{max,ua}$ is the longest length from the point p_0 to every point in the upper-arm body. Since the value of $L_{max,ua}$ is

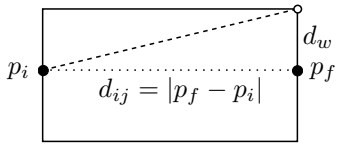

Fig. 8: A rectangular motion-constraint, which defines a boundary of the position of a body point. The boundary is defined by the positions of p_i and p_f, and a pre-defined margin width d_w. The maximum travel-distance in the motion-set is $\sqrt{d_{ij}^2 + d_w^2}$, where d_{ij} is the distance between p_i and p_f.

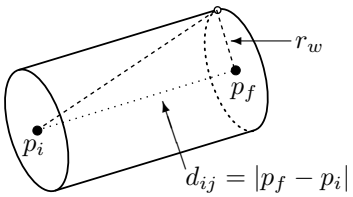

Fig. 9: A cylindrical motion-constraint, which defines a boundary of the 3D position of a body point. The 3D cylinder-shaped boundary is defined by the positions of p_i and p_f, and a pre-defined radius r_w. The maximum travel-distance in the motion-set is $\sqrt{d_{ij}^2 + r_w^2}$, where d_{ij} is the distance between p_i and p_f.

constant once the position of p_0 has been decided, $L_{max,ua}$ can be pre-computed before a run-time loop begins.

Similarly, the maximum travel-angle of the lower arm is

$$\theta_{max,la} = \alpha_{la} + \omega_{la} \tag{12}$$

And the maximum travel-distance of the lower arm is

$$d_{max,la} = d_{max,p1} + L_{max,la}\theta_{max,la} \tag{13}$$

where $d_{max,p1}$ is the maximum travel-distance of the point $p1$, computed as

$$d_{max,p1} = L_{p_0p_1}\theta_{max,ua} \tag{14}$$

and $L_{max,la}$ is the maximum length from the point p_1 to every point in the lower-arm body. $L_{p_0p_1}$ is the length between the point p_0 and the point p_1, which is also a constant value. Therefore,

$$d_{max,la} = L_{p_0p_1}\theta_{max,ua} + L_{max,la}\theta_{max,la} \tag{15}$$

For the end-effector part, the travel distance is computed similarly as

$$d_{max,ee} = L_{p_0p_1}\theta_{max,ua} + L_{p_1p_2}\theta_{max,la} + L_{max,ee}\theta_{max,ee} \tag{16}$$

The motions of rotating along the direction axes $\vec{e}_1, \vec{e}_2, \vec{e}_3$ are not considered in the computation of the maximum travel distances since those motions are less likely to cause a collision due to the cylindrical shapes of the robot's arm.

In the second experiment, we used a 9 d.o.f. Puma mobile robot model that has a 6 d.o.f. Puma-560 manipulator on a 3 d.o.f. mobile base. The robot's task is to move from an initial position to a goal position 1.6m away and reach a specific goal point with the tip of its end-effector. The task motion is monitored by five motion sets: a rectangular motion-set constraining the mobile base position, a cylindrical motion-set for the wrist position, and three spherical motion-sets for the orientations of the upper arm, the lower arm, and the end-effector, as shown in Figure 10. The collision checking algorithm applies separately to four different parts: the mobile base, the upper arm, the lower arm, and the end-effector.

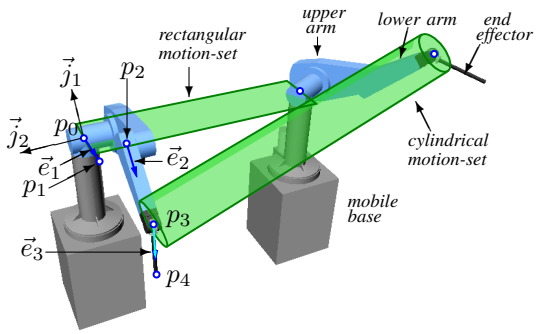

Fig. 10: Motion-sets placed on the Puma-mobile robot. A rectangular motion-set bounds the base position p_0 on a plane, and a cylindrical motion-set bounds the wrist position p_3. Three unit-vectors of $\vec{e}_1, \vec{e}_2, \vec{e}_3$ define spherical motion-sets, which bound the orientations of the upper arm, the lower arm, and the end-effector. Note that since the point p_1 is located on the crossed axis of $(\vec{j}_1 \times \vec{j}_2)$, where $\vec{j}_1$ and $\vec{j}_2$ are the axes of the first and second joints of the manipulator, p_1 is placed outside the second link body even though the point moves with the second link. Placing p_1 on $(\vec{j}_1 \times \vec{j}_2)$ eliminates the rotating motion of the upper arm along $\vec{e}_1$.

The maximum travel-distance of the mobile base is computed as

$$d_{max,base} = \sqrt{(p_{0,f} - p_{0,i})^2 + {d_{w0}}^2} \tag{17}$$

where d_{w0} is the pre-defined margin width of p_0. Similarly, the maximum-distance of the wrist point is computed as

$$d_{max,wrist} = \sqrt{(p_{3,f} - p_{3,i})^2 + {r_{w3}}^2} \tag{18}$$

where r_{w3} is the pre-defined margin radius of p_3. The maximum travel-distance of the upper arm is the sum of the translation motion of p_0 and the rotational motion of the upper arm, which is

$$d_{max,ua} = d_{max,base} + L_{max,ua}\theta_{max,ua} \tag{19}$$

where the definitions of $L_{max,ua}$ and $\theta_{max,ua}$ are same as in equation (11). The maximum travel-distances of the lower arm and the end-effector are computed from the translation motion of the wrist point p_3. Therefore, those distances are

$$d_{max,la} = d_{max,wrist} + L_{max,la}\theta_{max,la} \tag{20}$$

$$d_{max,ee} = d_{max,wrist} + L_{max,ee}\theta_{max,ee} \tag{21}$$

For those two types of the robots, we made an external obstacle model move around the robot motion paths at various distances and recorded the algorithm-computation time, proximity-computation time, number of nodes, number

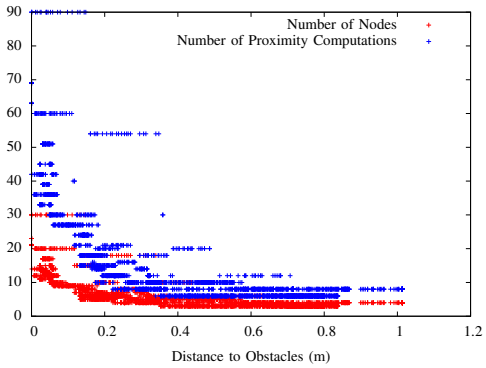

Fig. 11: The number of nodes that are created to check collisions of the Kuka robot motion with obstacles at different distances. The blue dots indicate the number of proximity-computation function calls at the corresponding distances. This result shows that the number of nodes is adjusted to adapt the computational amount to the environmental complexity.

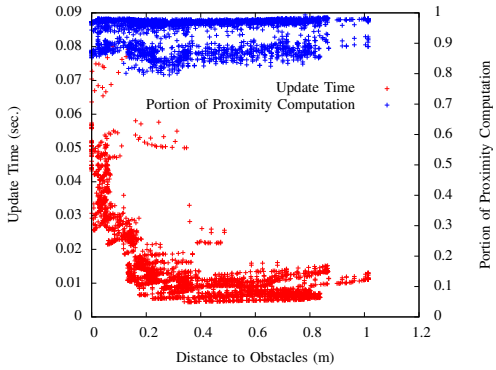

Fig. 12: Update time (Kuka robot) of the collision-checking algorithm at various distances from the obstacles. The blue dots indicate the portion of proximity-computation time with respect to the time of the overall collision-checking algorithm. This result shows that the overall computation time increases up to 0.08 seconds when the obstacle is very close to the robot models, and that more than 90% of the computation time is used for the proximity computation.

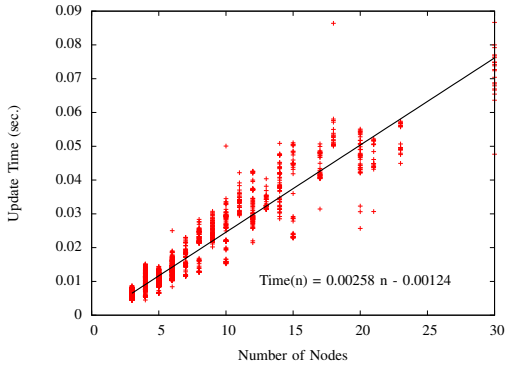

Fig. 13: Update time (Kuka robot) of the collision-checking algorithm for each different number of nodes. The plot shows that the update time is linearly related to the number of nodes. The data is fitted to the straight line whose slope is 0.00258. Thus, each node adds 2ms to the computation time.

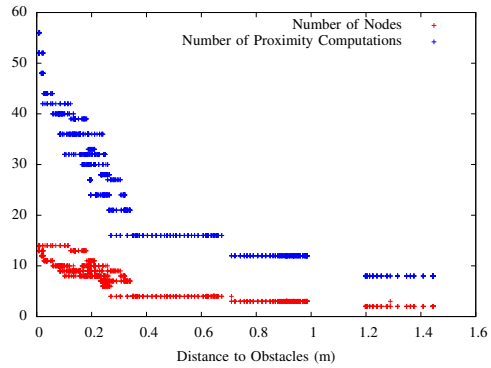

Fig. 14: Number of nodes that are created to check collisions of an 8 d.o.f. mobile-base Puma robot along the 1.6m long motion path. The number of nodes increases steeply when the distance to obstacles is less than 0.4m and reaches to the maximum number when the distance is less than 0.1m.

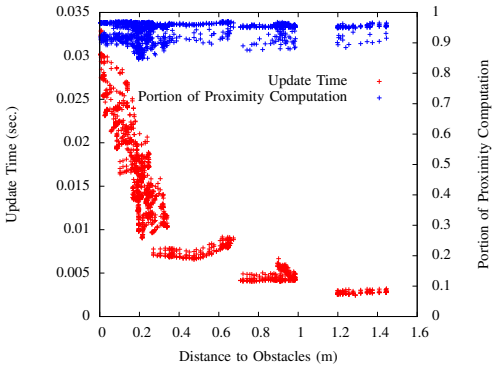

Fig. 15: Update time (Puma robot) of the collision-checking algorithm at various distances from the obstacles. The overall computation time increases up to 0.03 seconds when the node number increases to 14. As similar to the results of Kuka robot, the proximity computation consumes 90% of the overall computation time.

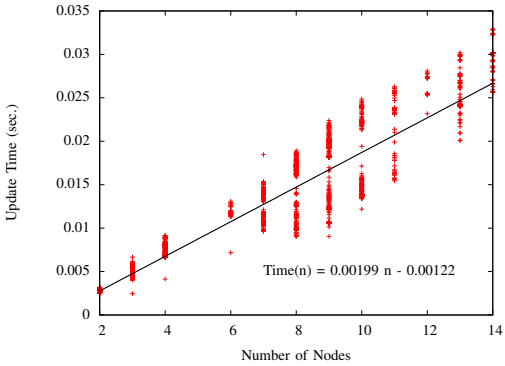

Fig. 16: Update time (Puma robot) of the collision-checking algorithm for each different number of nodes. The fitting line has a slope of 0.00199, which is similar to the result of Kuka robot.

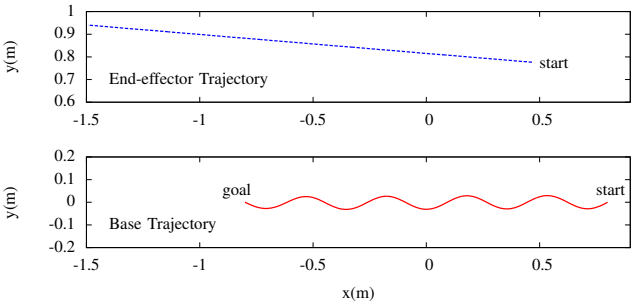

Fig. 17: The x and y positions of the base and the end-effector. The base position moves along a sinusoid path, and the end-effector moves toward the goal position along a straight line. Those two tasks were implemented with the whole body control framework and successfully run without any mutual interference.

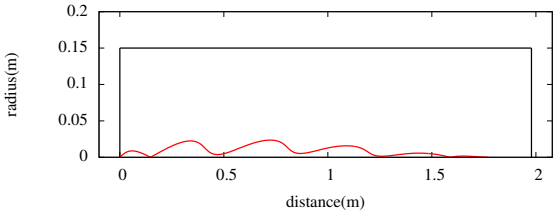

Fig. 18: Motion-set constraint evaluation of the cylinder model at the wrist position. In order to satisfy the constraint, the wrist position should stay inside the cylinder that connects the start wrist position and final wrist position as shown in Figure 10. The plot indicates the wrist positions by the distance from the initial wrist position along the center axis and the radius from the center axis.

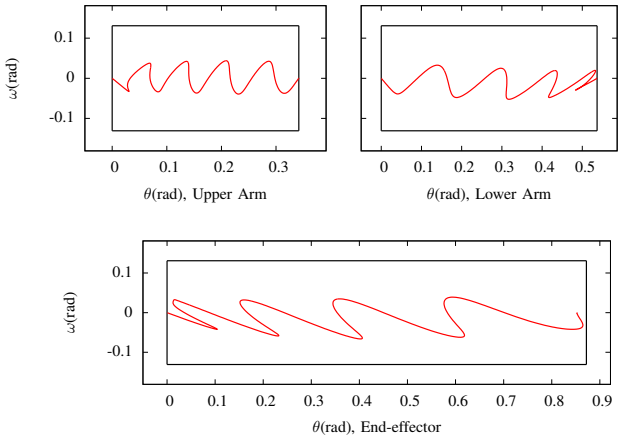

Fig. 19: Motion-set constraint evaluation of the sphere models for the orientations of the upper arm, the lower arm, and the end-effector. The sphere model indicates an orientation as a point on the surface of a unit sphere as shown in Figure 5. Thus, the relative position of the end point with respect to the base point is converted into a point on a unit sphere. The plot shows the trajectory of the points indicated by two angles on a unit sphere.

of proximity computations, and minimum distance from every node configuration to the obstacle. The performance was measured on a computer with an Intel Core i7 CPU (8 cores of 2.8GHz) and a Windows 7 operating system.

Figure 11, 14 show the number of nodes that were created for the recursive collision checking at various distances from the closest obstacle, and the number of proximity-computation function calls at those distances. Those plots show that the node number is the smallest when the obstacle is more than 1.0m away from the robot models and that it increases up to the maximum node number when the obstacle approaches closer than 0.1m. The number of proximity-computation function calls increases almost linearly with the number of nodes in both cases.

Figure 12, 15 show the update time of the collision-checking algorithm at various distances. The update time of the Kuka robot ranges between 0.003 and 0.08 seconds and time of the Puma robot ranges between 0.003 and 0.03 seconds, which is almost linearly dependent on the number of nodes, as shown in figure 13, 16. In both experiments, each node adds approximately 2ms to the update time. The result shows that our algorithm is suitable for constructing an on-line motion-modification system that updates on the order of 10Hz in speed.

In those previous figures, the results show the ability to adapt the computation load to the complexity of the environment. A motion planner can create more nodes at narrower areas in order to increase the resolution of the collision-checking method. On other hand, it can remove redundant nodes and minimizes the number of nodes at wider areas so that it can reduce computation load and speed up the collision checking process.

Figures 12 and 15 also indicate the portion of update time occupied by the proximity-computation function. In both cases, the proximity computation occupied almost 90% of the overall update time. The proximity computation was able to dominate in both examples because the complexity of the computation of the maximum travel-distance is merely $O(1)$. Therefore, in order to achieve the best performance, it is very important to use the fastest proximity-computation algorithm and optimize it in the implementation. In our current implementation, we use Quinlan's hierarchical sphere model [16] for the proximity computation.

B. Multiple-Task Motions in the Safe Range

One of the advantages of the algorithm is that we can easily run a complex task motion without collision since the algorithm provides collision-free conditions as a set of motion ranges. Therefore, complex task motions such as multiple task motions or suddenly changing, reactive motions can be run quickly if those motions satisfy the collision-avoidance conditions of the motion-set. The next experiment is an example of multiple-task execution within the motion-set constraints. In the example, two different tasks are executed by the Puma mobile-base robot. The first task is for the base to follow a sinusoid path between the start and end positions; the second task is for the end-effector to move in a straight line until reaching the goal position. Those two different tasks can be executed by the high d.o.f. Puma robot

when the controller is implemented with the whole body control framework [17], which supports prioritized, multiple-task control in operation spaces. For collision checking, the same set of five motion sets (1 rectangle, 1 cylinder, 3 spheres) as in the previous experiments was used in this one.

The whole body controller computes the multiple-task torque with

$$\Gamma = \Gamma_{ee} + N_{ee}^T(\Gamma_{base} + N_{base}^T\Gamma_{posture}) \qquad (22)$$

where Γ_{ee} is the three-dimensional position-control torque at the end-effector, Γ_{base} is the two-dimensional position-control torque at the base, and $\Gamma_{posture}$ is the torque for a default posture. N_{ee} and N_{base} are the null-space matrices of the end-effector position and the base position, respectively, by which the priorities of multiple tasks are implemented.

Figure 17 indicates the two trajectories recoded during the multiple task execution. The two task conditions are satisfied until the robot reaches the final position and there are no significant interferences between those different task controls. The base position stays within the base motion-set constraint that has a rectangular shape.

Figure 18-19 show the evaluations of the motion-sets on the wrist position, the upper arm orientation, the lower arm orientation, and the end-effector orientation, respectively. Figure 18 shows the wrist position trajectory as distance from the initial position along the cylinder axis and radius from the axis. The other figures show the orientation trajectories with two angles. Those plots indicate that none of the five motion-set constraints has been violated. These results mean that the robot successfully finished the multiple tasks without any collision against external obstacles. As seen in this example, when proper motion sets are constructed and indicated as collision-free by the algorithm, a high d.o.f. robot can easily execute complex tasks and avoid collisions within the ranges of the motion set.

IV. CONCLUSIONS

Our adaptive collision-checking method has demonstrated fast collision detection over continuous motion ranges of high d.o.f. robots. The method has three advantages, which were verified by experimental results. First, since the amount of computation in run-time is very small, the method is suitable for on-line applications. Moreover, the algorithm defines the applicable motion in a constraint-based manner so that complex robots can simultaneously execute multiple tasks and avoid collisions. Furthermore, for computational efficiency the resolution of the collision checking is dynamically adapted to the free-space width near the motion. Therefore, the proposed method promises to become one of the essential elements in realizing on-line, robotic applications in dynamically changing environments, such as applications in which humans and robots cooperate to perform interactive tasks.

REFERENCES

[1] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Conference on Computer graphics and interactive techniques*, vol. 30, 1996, pp. 171–180.

[2] G. v. d. Bergen, "Efficient collision detection of complex deformable models using aabb trees," *Journal of Graphics Tools*, vol. 2, no. 4, pp. 1–13, 1997.

[3] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 4, no. 1, pp. 21–36, 1998.

[4] J. Li and J. Xiao, "Exact and efficient collision detection for a multi-section continuum manipulator," in *International Conference on Robotics and Automation*, 2012, pp. 4340–4346.

[5] S. Cameron, "Collision detection by four-dimensional intersection testing," *Robotics and Automation, IEEE Transactions on*, vol. 6, no. 3, pp. 291–302, 1990.

[6] A. Foisy and V. Hayward, "A safe swept volume method for collision detection," in *The Sixth International Symposium of Robotics Research*, 1993, pp. 61–68.

[7] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *Proceedings of the ninth ACM symposium on Solid modeling and applications*, 2004, pp. 145–156.

[8] J. Canny, "Collision detection for moving polyhedra," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 2, pp. 200–209, 1986.

[9] A. Schweikard, "Polynomial time collision detection for manipulator paths specified by joint motions," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 6, pp. 865–870, 1991.

[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 *Proceedings of the 1995 symposium on Interactive 3D graphics*, 1995, pp. 189–196. [Online]. Available: http://doi.acm.org/10.1145/199404.199437

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

[12] J. Basch, L. J. Guibas, and J. Hershberger, "Data structures for mobile data," in *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 1997, pp. 747–756.

[13] R. Vatcha and J. Xiao, "Perceiving guaranteed continuously collision-free robot trajectories in an unknown and unpredictable environment," in *International Conference on Intelligent Robots and Systems*, 2009, pp. 1433–1438.

[14] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, Dec. 2002.

[15] F. Schwarzer, M. Saha, and J.-C. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 338–353, 2005.

[16] S. Quinlan, "Efficient distance computation between non-convex objects," in *International Conference on Robotics and Automation*, 1994, pp. 3324–3329.

[17] O. Khatib, L. Sentis, and J. Park, "A unified framework for whole-body humanoid robot control with multiple constraints and contacts," in *European Robotics Symposium*, 2009, vol. 44, pp. 303–312.