2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 7-12, 2012. Vilamoura, Algarve, Portugal

Accurate Evaluation of a Distance Function for Optimization-based Motion Planning

Youngeun Lee[1], Sébastien Lengagne[2], Abderrahmane Kheddar[3], Young J. Kim[1]

***Abstract*— We propose three novel methods to evaluate a distance function for robotic motion planning based on semi-infinite programming (SIP) framework; these methods include golden section search (GSS), conservative advancement (CA) and a hybrid of GSS and CA. The distance function can have a positive and a negative value, each of which corresponds to the Euclidean distance and penetration depth, respectively. In our approach, each robot's link is approximated and bounded by a capsule shape, and the distance between some selected link pairs is continuously evaluated along the joint's trajectory, provided by the SIP solver, and the global minimum distance is found. This distance is fed into the SIP solver, which subsequently suggests a new trajectory. This process is iterated until no negative distance is found anywhere in the links of the robot. We have implemented the three distance evaluation methods, and experimentally validated that the proposed methods effectively and accurately find the global minimum distances to generate a self-collision-free motion for the HRP-2 humanoid robot. Moreover, we demonstrate that the hybrid method outperforms other two methods in terms of computational speed and reliability.**

I. Introduction

Motion planning is a central problem in robotics and computer animation. Many techniques exist to plan a motion for complicated robotic systems and animated figures including criticality-based, sampling-based, potential fields, and decomposition methods, etc [1]. Recently, however, there has been a renewed interest for optimization-based motion planning techniques in the planning community; advances in the field of optimization make it possible for researchers to use powerful generic solvers for large-scale non-linear problems, for instance, such as motion planning for a humanoid robot. Moreover, optimization-based techniques can be efficiently coupled with sampling-based techniques as a local steering method, that take into account diverse constraints such as collision, joint state, torques limits, dynamics, tasks, equilibrium, those typically inherent to humanoid robots.

Generating optimal time-space trajectories is known as semi-infinite programming (SIP) since the trajectories are written in terms of a finite number of parameters (which is also part of the optimization variables). Moreover, the constraints must hold and the optimization cost function should be evaluated all along the time interval where the motion is defined [2][3].

In a SIP formulation of robotic motion planning, the non-penetration constraint (or collision avoidance) can be written as [2], [3]:

$$\delta(C_i(t), C_j(t)) - \epsilon \geq 0 \ \ \forall t \in [0, T_f] \tag{1}$$

where C_i and C_j are robot's two moving links i and j respectively. In our framework, we assume that the values of indices i, j defining all the pairs to be checked are known a priori (see e.g. in [4]). Moreover, $\epsilon \geq 0$ is a user-defined security margin, and T_f denotes the ending time of a motion that can be set as a constant or a variable i.e. part of the SIP parameters.

In the approaches proposed by [2] and more recently by [5], a grid is defined to sample the optimization problem at a set of time-samples $[t_0, t_1, \cdots, t_{T_f}]$. Then all the constraints including the non-penetration constraints are evaluated only at these time-samples. These approaches assume that if a constraint holds at both t_k and t_{k+1}, then it holds $\forall t \in \ [t_k, t_{k+1}]$. However, there is no guarantee that this assumption is indeed valid, and therefore collisions can be missed.

Main Results: In this paper, in order to accurately impose the non-penetration constraints on the SIP framework, we propose three novel algorithms to evaluate a distance function for humanoid motion planning including golden section search, conservative advancement and their hybrid. Our algorithms perform proximity computation between pairs of capsule shapes that tightly approximate robot's links; capsule shapes have been popularly adopted as bounding volumes (BV) for collision detection and avoidance [6], [7], [8], [9], mainly because the signed distance between a pair of capsules can be computed analytically and rapidly. We integrated our algorithm into the optimization constraint solver, which lacks in the latest optimization-based planning work [3], and perform various simulations and experiments on challenging motion planning cases with the HRP-2 humanoid robot to validate our approach. Unlike earlier discrete time-sampling approaches, our methods provide error-bounded results in terms of distance calculation and show a good runtime behavior.

Organization: The rest of the paper is organized as follows. We briefly survey the works relevant to ours in Sec. II, and provide the background information about our optimization-based planner as well as formulate our problem in Sec. III. In Sections IV, V, and VI, we propose our three

[1]Youngeun Lee and Young J. Kim are with the Department of Computer Science and Engineering at Ewha Womans University in Seoul, Korea `youngeunlee@ewhain.net, kimy@ewha.ac.kr`

[2]Sébastien Lengagne is with the Karlsruhe Institute of Technology, Institute for Anthropomatics, Humanoids and Intelligence Systems Lab, Adenauerring 2, 76131 Karlsruhe, Germany `lengagne@kit.edu`

[3]Abderrahmane Kheddar is with CNRS-AIST Joint Robotics Laboratory (JRL), UMI3218/CRT, Tsukuba, Japan and also with the CNRS-UM2 LIRMM, Interactive Digital Human group, Montpellier, France. `kheddar@ieee.org`

novel distance calculations techniques, and show experimental results in Section VII. We conclude our work and explain the possible future work in Section VIII.

II. Previous Work

In this section, we briefly survey the works relevant to proximity computation and our optimization-based motion planning for humanoids.

A. Proximity Computation

In robotics, CAD and computer graphics, many algorithms exist to rapidly evaluate a distance function depending on the underlying metric. A complete coverage on this topic is beyond the scope of this paper, and we survey only a few representative algorithms in the field and refer the readers to see [10] for more details.

Possibly, computing the Euclidean distance (also known as the separation distance) between a pair of separating bodies is the most well studied proximity problem in the literature. In particular, a simplex-based approach like GJK [11] or feature-tracking algorithm like Lin/Canny algorithm [12] are most well known algorithms to compute distance for convex polytopes and many variations exist depending on the type of objects and applications. For more general, polygon-soup models, bounding volume hierarchy (BVH)-based techniques are prevalent, and the algorithm like PQP [13] is widely used in the field.

Since the Euclidean distance is undefined for colliding objects, the notion of negative distance has been introduced. Among others, penetration depth (PD) is the most well known measure to quantify the amount of overlap or to represent the negative distance, which is often defined as a minimum translation to separate one object from another colliding object [14]. For convex polytopes, algorithms such as DEEP [15] and EPA [16] are most well-known in the community, and a few algorithms exists to compute the PD for non-convex polyhedra [17] or polygon soups [18].

Related to proximity computation, in the literature, several techniques have been proposed to avoid collision misses over a continuous motion: these collision checkers are often called continuous collision detection (CCD) or four dimensional collision checks [19][20][8]. In this paper, we have the similar spirit as CCD in a sense that we want to impose the non-penetration constraints over a continuous time-interval. However, the main departure from the existing CCD works is that we do so by evaluating a distance function and formulate it as a SIP.

B. Optimization-based Motion Planning for Humanoids

To formulate motion planning as an optimization problem, we need to define a cost function that evaluates the quality of the motion we want to obtain; e.g. minimizing the speed of realization, smoothness of the motion, energy consumption, resemblance to a reference motion, or any combination of these. We then define the constraints under which the motion is to be generated; e.g. the robot joints state and torques limits, equilibrium, collision avoidances. These constraints can be written as equality or inequality constraints, or even be part of the cost function. The cost function and the constraints along with the governing models of the robot are provided to the optimization solver; the models that govern the motion of a robot include its geometry, kinematics and dynamics that contain the joints and their derivatives and subsequently the optimization variables (i.e. trajectory parameters). Some work have considered formulating the problem together with the solver [21]; others considered using off-the shelf solvers and concentrate mainly on the formulation [2], [3], [5].

III. Background and Problem Formulation

A. SIP-based Framework

We formulate motion planning as a SIP as described in [3]. At each iteration of the optimization process, the solver proposes a solution in terms of P, the control knots of the B-spline corresponding to each joint's trajectory $q(t)$. Using the Taylor model [3], the motion of each robot's link (that is, translation $\mathbf{T}(t)$, rotation $\mathbf{R}(t)$, translational $\mathbf{v}(t)$ and angular $\omega(t)$ velocities[1]) is represented as time-dependent polynomials. For a given t, the state of any link is determined and the minimal distance can be computed between robot's link pairs. The minimal distance are fed back to the solver to compute a new solution (Fig. 1).

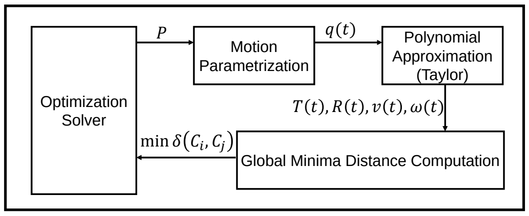
Optimization Solver
P
Motion Parametrization
$q(t)$
Polynomial Approximation (Taylor)
$T(t), R(t), v(t), \omega(t)$
$\min \delta(C_i, C_j)$
Global Minima Distance Computation

Fig. 1. Integrating the distance constraints into the optimization solver.

However, in [3], non-penetration constraint was not considered for a self-collision free motion. Our goal in the paper is to add the non-penetration constraints (1) to the above optimization pipeline to enable collision avoidance. More precisely, for given joint trajectories $\mathbf{q}(t)$ and the time interval T_f, we evaluate the minimal distance between pairs of robot's links and feed it back to the solver.

B. Problem Statement

Now we define our problem more precisely. For a given pair i, j of robot links and their respective capsule approximations $C_i(t), C_j(t)$, we want to compute the global minimum distance $\min_t \delta(C_i(t), C_j(t))$ over $t \in [t_k, t_{k+1}]$ when $C_i(t)$ moves relatively to $C_j(t)$ with $\mathbf{T}(t)$, $\mathbf{R}(t)$, $\mathbf{v}(t)$ and $\omega(t)$. Note that each element of $\mathbf{T}(t)$, $\mathbf{R}(t)$, $\mathbf{v}(t)$ and $\omega(t)$ is given as a fifth order polynomial based on [2], [3]. Further, we need to consider not only a positive distance, i.e. $\delta(C_i, C_j) > 0$ when C_i, C_j are separated, but also a negative

[1]We often omit the time parameter t from a time-dependent expression as long as its meaning is obvious in the context; e.g. $\delta(C_i, C_j)$ instead of $\delta(C_i(t), C_j(t))$.

distance, $\delta(C_i, C_j) \leq 0$ when C_i, C_j overlap, so that the optimization solver eventually realizes the non-penetration constraint (i.e. $\delta(C_i, C_j) > 0, \forall i, j$).

Since a capsule C_i is defined by its medial line l_i and a radius r_i, the distance between two capsules C_i and C_j can be computed as (also Fig.2):

$$\delta(C_i, C_j) = \delta(l_i, l_j) - (r_i + r_j) \tag{2}$$

It is obvious that (2) corresponds to the Euclidean distance for two capsules C_i, C_j when $C_i \cap C_j = \emptyset$. Otherwise, (2) is their penetration depth (PD) [14], i.e. a minimum translation to separate C_i from C_j or vice versa. Note that the maximum amount of PD according to (2) is $r_i + r_j$. Thus, a lower bound of our distance function is $-(r_i + r_j)$.

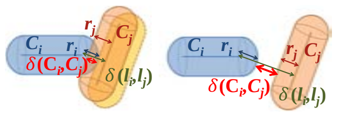

Fig. 2. The distance $\delta(C_i, C_j)$ between two capsules C_i, C_j. (Left) $\delta(C_i, C_j)$ is negative when $C_i \cap C_j \neq \emptyset$ and its amount corresponds to penetration depth. (Right) $\delta(C_i, C_j)$ is the separation distance when $C_i \cap C_j = \emptyset$.

IV. Golden Section Search

Now we first explain a simple approach to compute the minimum distance between two moving capsules over time. One can easily evaluate (2) at any given time $t \in [0, T_f]$, but (2) may not be differentiable everywhere. The Golden Section Search (GSS) is a well-known technique for finding the extremal value of a function $f(x)$ without calculating its derivative or without using other information about $f(x)$ [22], [23]. We can use GSS to find the minimum of (2).

Given a bracket $[a, b, c]$ where $a < b < c$, $f(b) < f(a)$, $f(b) < f(c)$, GSS computes a new narrower bracket containing the minimum. At each iteration, GSS chooses a new point d using the golden ratio $R = \frac{3-\sqrt{5}}{2}$ and compare $f(d)$ against $f(b)$ to get a new bracket containing the minimum value. We choose either b or d as a new middle point of the bracket, that has a smaller functional value:

1) If $|b-a| \geq |c-b|$, d should be between a and b and set $d = b - R(b-a)$.
   - If $f(d) \geq f(b)$, the new bracket is $[d, b, c]$
   - If $f(d) < f(b)$, the new bracket is $[a, d, b]$
2) If $|b-a| < |c-b|$, then d should be between b and c such that $d = b + R(c-b)$.
   - If $f(d) \geq f(b)$, the new bracket is $[a, b, d]$.
   - If $f(d) < f(b)$, the new bracket is $[b, d, c]$.

GSS iterates this process until $|c-a| < \tau(|b| - |d|)$ where τ is the required precision for the result, and finally returns $f(b)$ as a minimum.

In our minimum distance problem, a, c and b respectively correspond to the time instances t_k, t_{k+1} and their midpoint. Since we cannot guarantee that $\delta(C_i, C_j)$ at b is less than those at a, c, we return the minimum of $\delta(C_i(a), C_j(a))$, $\delta(C_i(b), C_j(b))$ and $\delta(C_i(c), C_j(c))$ as the global minimum distance.

Note that the GSS may miss the global minimum distance when $\delta(C_i, C_j)$ has more than two local minima over $[t_k, t_{k+1}]$. Hence, GSS can be reliably used only when the (distance) function is known to have one extremum for the given interval.

V. Search based on Conservative Advancement

Now, we present a method to search for the global minimum distance based on conservative advancement (CA).

A. CA for Capsules

The CA is a simple method to calculate the first time of contact t_{toc} between two moving convex objects C_i, C_j; i.e. $\delta(C_i(t_{toc}), C_j(t_{toc})) = 0$, assuming that w.l.o.g. $t_{toc} \in [0, 1]$ (i.e. the normalized time interval). In more detail, in CA, we compute an upper bound on the time Δt in which an object safely moves without creating any collision [24][25]:

$$\Delta t \leq \frac{\delta(C_i, C_j)}{\mu} \tag{3}$$

where μ is an upper bound on the relative motion between two objects along the direction of $\delta(C_i, C_j)$ at the beginning of CA (i.e. the closest direction $\mathbf{n}$). Then, Δt is recalculated after advancing the objects, and iterated until $\delta(C_i, C_j)$ becomes sufficiently small. Then, $t_{toc} = \sum \Delta t$ [8].

Since there are many efficient methods known to compute $\delta(C_i, C_j)$ for convex objects such as [11], [12], the main computational issue in CA boils down to computing μ tightly. We use a variant of [8] to compute μ for two capsules C_i, C_j as follows (w.l.o.g. C_i is movable and C_j is fixed):

$$\mu = \max_t |\mathbf{v}(t) \cdot \mathbf{n}| + \max_t \|\omega(t) \times \mathbf{n}\| (r + \frac{\max_{l,t} \|\mathbf{c}_l \times \omega(t)\|}{\min_t \|\omega(t)\|}) \tag{4}$$

where C_i is composed of a medial line $\overline{\mathbf{c}_1 \mathbf{c}_2}$ and a radius r, and $\mathbf{v}$, ω are the linear and angular velocities of C_i. The first term $|\mathbf{v}(t) \cdot \mathbf{n}|$ in (4) is a polynomial of degree 5, represented by a B-spline. We use the control points of the B-spline trajectory to maximize $|\mathbf{v}(t) \cdot \mathbf{n}|$ like [3]. The other terms $\|\omega(t) \times \mathbf{n}\|$, $\|\mathbf{c}_l \times \omega(t)\|$, $\|\omega(t)\|$ are polynomials of degree 10, and we truncate any polynomials greater than degree 5, and maximize(or minimize) the rest of polynomials in the same manner.

B. Minimum Finding using CA

One can generalize the CA technique to find the time instance t such that $\delta(C_i(t), C_j(t)) = d$; note that the time of contact t_{toc} is a special case of this, $\delta(C_i(t_{toc}), C_j(t_{toc})) = 0$. In order to do this, the CA iteration in (4) can be transformed to:

$$\Delta t \leq \frac{\delta(C_i, C_j) - d}{\mu} \tag{5}$$

Note that the right hand side of the inequality of the above equation is always non-negative, since $\delta(C_i, C_j) - d \geq 0$ as long as d is less than the distance between C_i and C_j at the beginning of iterations; i.e. $d \leq \delta(C_i(0), C_j(0))$. (5) is still valid when $d < 0$ for capsules (i.e. penetration depth), as proven in Lemma 1.

LEMMA 1 *Given two capsules, C_i, C_j, $d \leq \delta(C_i(\Delta t), C_j(\Delta t))$ when Δt is given as (3).*

Moreover, for a given $d \leq \delta(C_i(0), C_j(0))$, we can also test whether $\forall t, d < \min_t \delta(C_i(t), C_j(t))$ by checking whether t exists such that $\delta(C_i(t), C_j(t)) = d$ for the given interval. The non-existence of such t implies that $\forall t, d < \min_t \delta(C_i(t), C_j(t))$. We use this fact to find the global minimum distance $\min \delta(C_i, C_j)$.

Our method to compute the global minimum distance is essentially a bisection method that recursively bisects the codomain of the distance function (e.g. the y-axis in Fig. 3). We first set up the initial interval I_0 along y-axis as $I_0 = [d_0, d_1]$ where $d_0 = -(r_i + r_j)$ and $d_1 = \min(\delta(C_i(t_k), C_j(t_k)), \delta(C_i(t_{k+1}), C_j(t_{k+1})))$. The lower bound d_0 of I_0 is the negative value of the maximum amount of PD between two capsules, according to (2). Then, we choose the mid point d_2 of d_0 and d_1 (i.e. $d_2 = \frac{d_0+d_1}{2}$) and test if $d_2 > \min \delta(C_i, C_j)$. This test is equivalent to finding the time $\exists t_2 \in [t_k, t_{k+1}]$ such that $d_2 = \delta(C_i(t_2), C_j(t_2))$. Then,

- If such t_2 exists, since the global minimum distance is between d_0 and d_2, we get the new interval $[d_0, d_2]$, and $\delta(C_i, C_j) > d_2$ for $t \in [t_k, t_2)$. Then, take d_3 as a mid point of d_0 and d_2 for the next iteration.
- If t_2 does not exist, the global minimum distance is between d_1 and d_2, and the new interval is $[d_1, d_2]$. Then, take d_3 as a mid point d_1 and d_2 for the next iteration.

We iterate the above process until the bisected time interval becomes less than some threshold τ. An example of this process is given in Fig. 3.

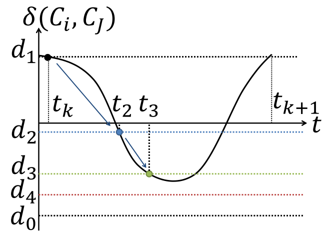

Fig. 3. Finding the global minimum distance by CA. The initial interval is $[d_0, d_1]$. The mid point d_2 is selected, and since $\exists t_2, d_2 = \delta(C_i(t_2), C_j(t_2))$ by using CA, the next interval is chosen as $[d_0, d_2]$. Same for the next mid point d_3, and thus the new interval is $[d_0, d_3]$. For the next mid point d_4, since $\forall t, d_4 < \delta(C_i(t), C_j(t))$, the new interval is $[d_4, d_3]$. This process is repeated.

Our CA-based method guarantees the resulting interval on the codomain (i.e. y-axis) always contains the global minimum distance since $\delta(C_i, C_j)$ is continuous, and the absolute error decreases in half at each iteration; for the nth iteration, the error is less than $\frac{|d_1-d_0|}{2^n}$. Even though our CA-based minimum-finding method guarantees a bounded error, which was impossible for GSS, the CA-based method can be slower than GSS. The main reason is that the motion bound μ can be relatively much greater than the distance δ in (2), as the iteration progresses. In detail, μ becomes loose due to the high order of polynomials involved in the underlying motion. In order to address the issue of slow convergence, in the next section, we combine the advantages of GSS and CA to provide a good performance in terms of speed and accuracy.

VI. Hybrid Method

The basic idea of our hybrid method is that we use CA to narrow down the interval that contains the global minimum distance, and then use GSS to quickly search for the minimum within that interval.

First of all, to accelerate the CA-based method presented in the previous section, we employ a dual-advancement scheme similar to [26] using both forward and backward advancements. Whereas the forward advancement does as described in Sec. V, the backward advancement starts reversely from t_{k+1} toward t_k. More precisely, the backward advancement looks for the nearest $t \in [t_k, t_{k+1}]$ such that $\delta(C_i, C_j) = d_2$ from t_{k+1}. (2) can be similarly used here to find t, except that the new time is obtained as $t = t_{k+1} - \sum \Delta t$ in contrast to $t = t_k + \sum \Delta t$ for forward advancement.

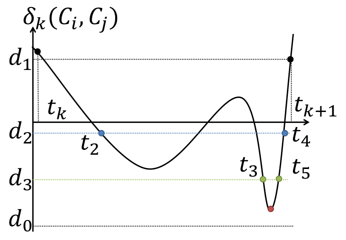

Fig. 4. The forward advancement first returns t_2 and then t_3 as a result. The backward one returns t_4 and t_5. Then, the global minimum distance is located between t_3 and t_5, and the GSS searches the minimum in $[t_3, t_5]$.

For instance, as shown in Fig. 4, the initial interval $[d_0, d_1]$ is the same for both forward and backward advancements. Then, the time interval $[t_3, t_5]$ is provided to the GSS solver, and use the same technique presented in Sec. IV to find the minimum distance.

In our hybrid method, we still need to address when we need to switch from the CA-based method to the GSS-based method. Ideally, as soon as the CA-based method isolates a single minimum, the GSS-based method can take it over to find the global minimum distance. A practical implementation idea is that whenever the size of distance interval is less than some threshold (i.e. $[d_i, d_{i+1}] < \epsilon$), one can switch from CA to GSS. Even though this method does not guarantee that the found result is a true minimum, but the error is always bounded by ϵ.

VII. Experimental Results

We implemented our distance calculation algorithms using C++ program language (cmake 2.8.0 compiler) and integrated it into our optimization software under Kubuntu 10.04

LTS 64bits operating system equipped with an Intel Core i7 2.67GHz CPU and 3GB main memory. We use the IPOPT package to solve the nonlinear optimization problem and the HSL archive package as a thirty party code for IPOPT. We also utilize the Eigen template for linear algebra computation and FADBAD++ to enable automatic differentiation.

We have tested our optimization-based motion planning algorithm on an HRP-2 humanoid robot using three sets of benchmarks. In the first benchmark (Fig. 5), the robot is trying to cross its arms twice while avoiding self-collisions. The second benchmark (Fig. 6) shows a case where the robot is twisting its upper body while avoiding the collision between arms and legs. In the last benchmark (Fig. 7), the robot attempts to pick up an object behind it without lifting off its feet.

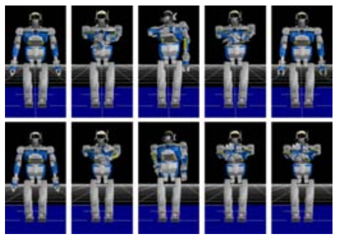
Fig. 5. Benchmark 1: hand-crossing. A collision occurs between the hands. (Top row) a motion without non-penetration constraint. (Bottom row) a self-collision-free motion using our hybrid method.

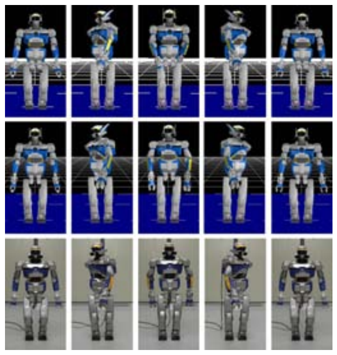
Fig. 6. Benchmark 2: twisting. (Top row) a motion without non-penetration constraint. Collisions occur between the hands and bodies. (Middle row) a self-collision-free motion using our hybrid method. (Bottom row) realization on a physical HRP-2 robot.

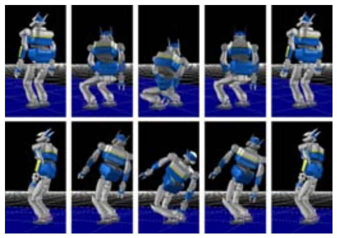
Fig. 7. Benchmark 3: picking. (Top row) a motion without non-penetration constraint. Collisions occur between the hands and legs. (Bottom row) a self-collision-free motion using our hybrid method.

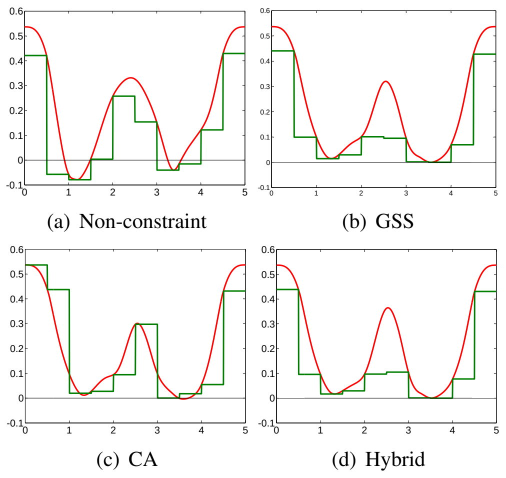

Fig. 8. These graphs shows the tracking result of distance between the hands of the robot in benchmark1. In the graph, the red lines show the exact distance tracked over the course of time interval, and the green lines mean the global minimum distance computed by each method for sub-intervals. Note that in (a), we manually calculate the minimum distance (the green line) by tracking the distance function.

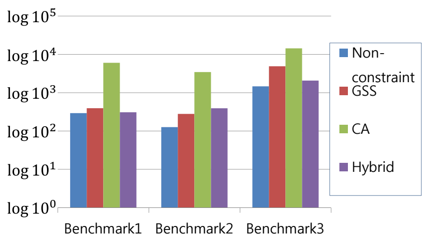

Fig. 9. Performance of our optimization-based motion planning algorithm in seconds using different distance computation algorithms. The non-constraint bar denotes the results of trajectory planning without taking into account the non-penetration constraint. The y-axis denotes the timing (second) on a logarithmic scale.

For GSS- and CA-based methods, we set the threshold τ to 10^{-3} to terminate the iteration. Fig. 8 shows the evolution of the distance (the red solid lines) between the right and left hands of the robot in the benchmark 1 as a result of optimization-based planning. In the figure, the green lines denote the results of minimum distance calculation for each time interval. In this benchmark, the distance results of GSS and hybrid method are nearly same. However, the results of CA are different from those of other methods and it cannot find the global minimum distance in this particular setting. A possible reason is that we may need a higher number of iterations to generate sufficiently accurate distance results, since the motion bound μ might be too loose for a motion containing high order polynomials; in this experiment, the maximum number of CA iterations that we tried was 500. In the hybrid method, we typically switch from CA-based to GSS-based after a single CA iteration.

Fig. 9 summarizes the timing performance of our optimization-based planner using the three distance calculation algorithms. In our planning algorithm, it is hard and may not be meaningful to extract only the distance computation time from the total planning time, since our distance computation algorithm is tightly coupled with the optimization solver and they affect the performance of each other. The CA-based method takes longer time than other methods. One interesting observation from our experiment is that GSS is slower than the hybrid method for benchmark 1 and benchmark 3, even though GSS is generally a *simpler* technique than CA. This is due to the fact that GSS requires a high number of iterations to find a solution comparable to CA's result for a complicated motion planning scenario involving many potential collisions. In case of benchmark 3, GSS is twice slower than the hybrid method. We summarize our experimental results as:

- The GSS method is a simpler technique than others and finds the global minimum distance similarly well like the hybrid method. But GSS has no guarantee as to the result.
- The planner based on the CA method is slowest and CA requires a high number of iterations to find a correct solution. However, it can provide a bounded error for the result.
- The planner based on the hybrid method is superior to other two in that it is faster and provides a bounded-error result.

VIII. Conclusions and Future Works

We present three algorithms, namely GSS, CA, and hybrid CA-GSS, to evaluate distance functions for a SIP-based motion planning algorithm and demonstrate its capability on the simulated and the real HRP-2 humanoid robot. Our results show that GSS works well in practice even though it cannot produce a guaranteed result. Theoretically, CA has a nice property of bounded-error output in terms of minimum distance calculation, but it may require many iterations to converge to the solution. The hybrid method takes the advantages of both GSS and CA, and performs faster and more reliably than each of the techniques alone.

In this work, we compute the global minimum distance only for self-collision cases, but there is no reason why this method is limited only to self-collision; inter-object collision can be handled similarly. Currently, our method is applicable only to a capsule-type geometry, but we would like to extend the method to a more general geometry, e.g. polyhedral models, for future work.

Acknowledgment

This research was supported in part by IT R&D program of MKE/MCST/KOCCA (KI001818), NRF in Korea (NRF-2011-013-D00112), the RoboHow.Cog FP7 `www.robohow.eu`, Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (B), JSPS Grant-in-Aid for JSPS Fellows (P09809), and German Research Foundation (DFG: Deutsche Forschungsgemeinschaft).

References

[1] J.-C. Latombe, *Robot Motion Planning*. Springer, 1990.

[2] S. Miossec, K. Yokoi, and A. Kheddar, "Development of a software for motion optimization of robots - application to the kick motion of the hrp-2 robot," in *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on*, Dec. 2006, pp. 299–304.

[3] S. Lengagne, P. Mathieu, A. Kheddar, and E. Yoshida, "Generation of dynamic motions under continuous constraints: Efficient computation using b-splines and taylor polynomials," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[4] J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Self-collision detection and prevention for humanoid robots," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 2002, pp. 2265–2270.

[5] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, may 2011, pp. 3719 –3726.

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

[7] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Department of Computer Science, University of North Carolina, Tech. Rep. TR99-018, 1999.

[8] M. Tang, Y. J. Kim, and D. Manocha, "C^2A: Controlled conservative advancement for continuous collision detection of polygonal models," *Proceedings of International Conference on Robotics and Automation*, pp. 356–361, 2009.

[9] C. Lauterbach, Q. Mo, and D. Manocha, "gproximity: Hierarchical gpu-based operations for collision and distance queries," *Computer Graphics Forum*, vol. 29, no. 2, pp. 419–428, 2010.

[10] M. Lin and D. Manocha, "Collision and proximity queries," in *Handbook of Discrete and Computational Geometry*, 2003.

[11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between objects in three-dimensional space," *IEEE J. Robotics and Automation*, vol. vol RA-4, pp. 193–203, 1988.

[12] M. C. Lin, "Efficient collision detection for animation and robotics," Ph.D. dissertation, University of California, Berkeley, 1993.

[13] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Distance queries with rectangular swept sphere volumes," *Proc. of IEEE Int. Conference on Robotics and Automation*, 2000.

[14] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri, "Computing the intersection-depth of polyhedra," *Algorithmica*, vol. 9, pp. 518–533, 1993.

[15] Y. J. Kim, M. C. Lin, and D. Manocha, "DEEP: an incremental algorithm for penetration depth computation between convex polytopes," *Proc. of IEEE Conference on Robotics and Automation*, pp. 921–926, 2002.

[16] G. van den Bergen, "Proximity queries and penetration depth computation on 3d game objects," in *Game Developers Conference*, 2001.

[17] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha, "Fast penetration depth computation using rasterization hardware and hierarchical refinement," *Proc. of Workshop on Algorithmic Foundations of Robotics*, 2002.

[18] C. Je, M. Tang, Y. Lee, M. Lee, and Y. Kim, "PolyDepth: Real-time penetration depth computation using iterative contact-space projection," *ACM Transactions on Graphics*, vol. 31, no. 1, Jan 2012.

[19] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," in *Proc. of Eurographics Computer Graphics Forum*, 2002.

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

[21] S.-H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, "Newton-type algorithms for dynamics-based robot movement optimization," *IEEE Transactions on Robotics*, vol. 21, pp. 657–667, 2005. [Online]. Available: http://ieeexplore.ieee.org/iel5/8860/32079/01492481.pdf

[22] J. Kiefer, "Sequential minimax search for a maximum," *Proceedings of the American Mathematical Society*, vol. 4, no. 3, pp. 502–506, 1953.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.

[24] M. C. Lin, "Efficient collision detection for animation and robotics," Ph.D. dissertation, 1993, aAI9430587.

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

[26] M. Tang, Y. Kim, and D. Manocha, "CCQ: Efficient local planning using connection collision query," in *Algorithmic Foundations of Robotics IX*, ser. Springer Tracts in Advanced Robotics, D. Hsu, V. Isler, J.-C. Latombe, and M. Lin, Eds. Algorithmic Foundations of Robotics IX, 2011, vol. 68, pp. 229–247.