Accurate Distance Tracking for Optimization-based Whole-Body Humanoid Motion Planning (Extended Abstract)*

Youngeun Lee[1], Abderrahmane Kheddar[2], and Young J. Kim[1],

***Abstract*— This paper presents a method to evaluate a distance function in semi-infinite programming (SIP) framework for optimization-based, whole-body humanoid motion planning. In our approach, the distance between pairs of robot's link and obstacles is evaluated along the joint's trajectory within successive time intervals. In these intervals, we compute a minimum of the continuous distance (MCD) to avoid collision. 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 use adaptive subdivision (AS) to compute the upper and lower bounds of the MCD. We have implemented our distance evaluation methods, and have experimentally validated the proposed method to effectively and accurately find the MCDs to generate a collision-free motion for the HRP-2 humanoid robot.**

I. Introduction

Motion planning, coupled with multi-contact control, for complicated robotic system such as humanoids is challenging problems and active areas of research in robotics. Recently, there has been a renewed interest for optimization-based motion planning techniques in the motion planning community, as 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, typically inherent to humanoid robots.

The optimization-based motion planning finds the best joint trajectories that minimize the cost function and satisfy the constraints in a continuous time domain. We use B-splines to convert the joint trajectories to a finite number parameters. This parameterization reduces the infinite programming problem into a semi-infinite programming (SIP). Moreover, the constraints must hold and the optimization cost function should be evaluated all along the time interval.

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

$$\delta(\mathcal{A}(t), \mathcal{B}(t)) - \epsilon \geq 0 \;\; \forall t \in [t_0, t_1] \tag{1}$$

where $\delta(\cdot)$ is a distance function and $\mathcal{A}$ and $\mathcal{B}$ are moving objects such as robot's links or obstacles. $\epsilon \geq 0$ is a user-defined security margin, and t_0 and t_1 denotes the starting and ending times of a motion, respectively. In our framework, we assume that the pairs to be checked are known a priori. The motion can be set as a constant or a variable i.e. part of the SIP parameters.

The moving objects should satisfy the non-penetration constraint the entire time interval. However, tracking the distance for moving objects over a continuous time interval is very difficult problem because it is necessary to track the whole time interval, not just some time instance, and to consider both the positive (i.e. Euclidean) and negative (i.e. penetration depth) distance. Note that sampling the distance over the time domain may violate the non-penetration constraint between sampled time-intervals and therefore collisions can be missed.

Main Results: In this paper, in order to accurately impose the non-penetration constraints into the SIP framework, we evaluate a distance function, namely the minimum of continuous distance (MCD). The MCD is the minimum value of the distance function in continuous time domain. Our algorithm computes the upper and lower bounds of the distance based on the amount of motion that a robot can make during the time interval, abandons the time intervals that cannot realize the MCD, updates the bounds for the remaining time intervals, and then adaptively subdivides the remaining intervals until the distance result is within the user-defined error bound. We integrate our algorithm into a optimization-based motion planner to perform challenging whole-body motion planning with the HRP-2 humanoid robot, while avoiding collision with obstacles and robot itself. Our algorithm takes tens to hundreds of milliseconds for robots and obstacles consisting of tens of thousands of triangles.

II. Background and Problem formulation

A. SIP-based Framework

At each iteration of the optimization process, the solver suggests a solution that is used to evaluate the constraints together with the cost function. The calculation results are fed back to the solver to compute a new solution as illustrated in Fig. 1.

* A full version of this extended abstract was submitted to IEEE Transactions on Robotics [1].

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

[2]Abderrahmane Kheddar is with CNRS-AIST Joint Robotics Laboratory (JRL), UMI3218/RL, Tsukuba, Japan and also with the CNRS-UM LIRMM, Interactive Digital Human, Montpellier, France.

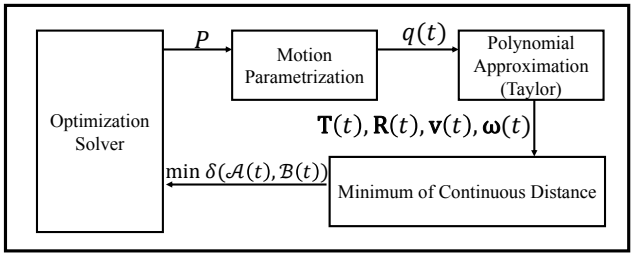

Fig. 1. Optimization-based motion planning using the MCD.

The trajectories $q(t)$ are represented as B-Splines, hence defined with a fixed number of control points (knots). These knots are the optimization variables together with the time interval of motion.

Our aim is to add constraints of type Eq. 1 in the optimization software. However, to the best of our knowledge, there is no optimization solver than can handle a varying number of non-penetration constraints. Thus, in our work, we focus on finding only the MCD and reformulate the non-penetration constraints as follow:

$$\min \delta(\mathcal{A}(t), \mathcal{B}(t)) - \epsilon \geq 0 \qquad \forall t \in [t_0, t_1] \tag{2}$$

B. Problem Statement

For two given models $\mathcal{A}$ and $\mathcal{B}$, our goal is to compute the continuous distance $\min_t \delta(\mathcal{A}(t), \mathcal{B}(t))$ over $t \in [t_0, t_1]$ when $\mathcal{A}(t)$ continuously and rigidly moves relative to $\mathcal{B}(t)$. The rigid relative motion combines translations and rotations, $\mathbf{T}(t)$ and $\mathbf{R}(t)$, and their associated linear and angular velocities are $\mathbf{v}(t)$ and $\omega(t)$, respectively. We further assume that $\mathbf{T}(t)$, $\mathbf{R}(t)$, $\mathbf{v}(t)$, and $\omega(t)$ are represented as fifth-order polynomials. A relative motion assumes that the object $\mathcal{B}$ is stationary while the object $\mathcal{A}$ is mobile (or vice-versa). It is obvious that the distance function $\delta(\cdot)$ is defined as the Euclidean distance when the models are separated. Further, we need to consider not only a positive distance but also a negative distance (i.e. signed distance) when $\mathcal{A}$ and $\mathcal{B}$ overlap. We define the negative distance as a negative value of the penetration depth (PD).

$$\delta(\mathcal{A}(t), \mathcal{B}(t)) = \begin{cases} \text{Euclidean distance} & \text{if } \mathcal{A} \cap \mathcal{B} = \emptyset \\ -\text{Penetration depth} & \text{if } \mathcal{A} \cap \mathcal{B} \neq \emptyset \end{cases} \tag{3}$$

III. Adaptive Subdivision

A. Adaptive Subdivision Algorithm

We use adaptive subdivision (AS) to compute the MCD. The concept of AS is very simple. We recursively subdivide the time interval $[t_0, t_1]$ to get the MCD. If an interval during the recursive subdivision is guaranteed to contain the MCD, we subdivide it further into two subintervals and check to see which sub-interval has the MCD.

The main function of adaptive subdivision is to test whether or not an interval includes the MCD. To do that, we define three distance bounds: the global upper bound UB, the local upper bounds UB_i, and the local lower bounds LB_i. UB is the upper bound of the MCD for the entire time interval $[t_0, t_1]$. UB_i and LB_i is the upper and lower bound of the minimum distance $\min_t \delta(\mathcal{A}(t), \mathcal{B}(t))$ over $t \in [t_0^i, t_1^i] \subset [t_0, t_1]$, respectively. We can determine whether an sub-interval includes the MCD using UB and LB^i as follows:

- $UB - LB^i < 0$: There is no MCD in the sub-interval. We discard the sub-interval.
- $UB - LB^i > \epsilon$: Since the MCD can be in the sub-interval, we subdivide the sub-interval for further examination. ϵ is the user-defined distance error bound.
- $0 \leq UB - LB^i \leq \epsilon$: In this case, all of the distance values in the sub-interval are greater than UB or are only ϵ away from UB. The minimum distance in the sub-interval is not updated to UB. We do not need to update UB to LB^i since LB^i is within ϵ from UB, so the minimum distance value in the sub-interval is LB^i and no more subdivision is done for the sub-interval.

We recursively and selectively subdivide the time intervals using these relationships. This method guarantees that the distance error is always bounded by ϵ.

B. Bound Calculation

Now we present how to calculate UB, UB^i, and LB^i to determine whether or not a sub-interval contains a solution for the MCD.

Since the MCD in $[t_0, t_1]$ is less than or equal to the minimum distance defined over some sub-interval $[t_0^i, t_1^i]$, UB can be set to the minimum of UB^i as in Eq. (4).

$$UB = \min_i UB^i, \tag{4}$$

where UB^i is an upper bound of the minimum distance value in the sub-interval $[t_0^i, t_1^i]$. The minimum distance over some time interval i is less than or equal to the distances at the end points of the interval, $\delta(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i))$ and $\delta(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i))$, and the local upper bound UB^i can be defined as the minimum of the distances at the endpoints as shown in Eq. (5).

$$UB^i = \min\{\delta(\mathcal{A}(t_0^i), \mathcal{B}(t_0^i)), \delta(\mathcal{A}(t_1^i), \mathcal{B}(t_1^i))\} \tag{5}$$

By definition, the local lower bound LB^i is less than any distance in the sub-interval $[t_0^i, t_1^i]$. In order to compute a non-trivial and tight LB^i, however, we compute a motion bound μ that is the largest displacement of an object during a time interval. Using μ, we estimate the closest $\mathcal{A}$ can be to $\mathcal{B}$ during the time interval, and the minimum distance at the time instance is LB^i, since $\mathcal{A}$ cannot move farther than μ. If we compute LB^i under the assumption that $\delta(\cdot)$ has only one minimal value in the sub-interval as the blue line illustrated in Fig. 2, LB^i is the minimum distance among the trajectories that has the same motion bound μ.

However, in reality, there may exist multiple extremal values in the sub-interval like the black line in Fig. 2. In this case, the time interval between the two extremal values can be ignored while calculating μ since the motion bound

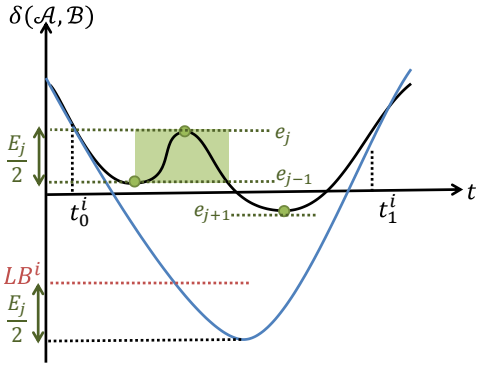

Fig. 2. Local lower bound computation using extremal values.

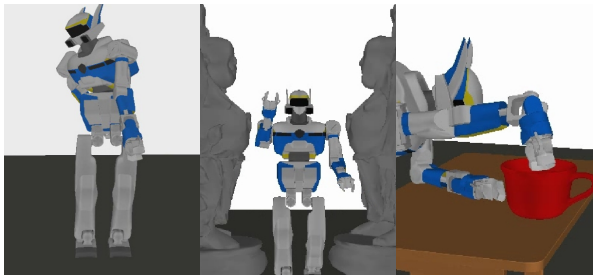

Fig. 3. HRP-2 robot motion benchmarks.

calculation deals only with the object displacement. For instance, in Fig. 2, the time interval corresponding to the green box can be safely removed to compute the minimum distance.

Given the extremal values in the sub-interval, we look for the maximal value e_j and then compare its minimal value e_{j-1} nearest from the left and e_{j+1} nearest from the right. If $e_{j-1} > e_{j+1}$ like in Fig. 2, $\mathcal{A}$ goes to the corresponding position from e_{j-1} to e_j and then comes back to e_{j-1}. In other words, $\mathcal{A}$ moves by $E_j = 2(e_j - e_{j-1})$. From μ, E_j does not correspond to the minimal value, but the maximal value. Thus, we can consider the motion bound of $\mathcal{A}$ to compute LB^i as $\mu - E_j$ instead of μ. Finally, we can get tighter like in Eq. (6) using E_j in the sub-interval.

$$LB^i = \min\{\delta(\mathcal{A}(t_0^i),\mathcal{B}(t_0^i)), \delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))\} - \frac{\mu - |\delta(\mathcal{A}(t_0^i),\mathcal{B}(t_0^i)) - \delta(\mathcal{A}(t_1^i),\mathcal{B}(t_1^i))| - \sum E_j}{2} \quad (6)$$

The distance function has the extremal values when the relative speed, Eq. (7),is equal to 0.

$$s_{rel}(t) = (\mathbf{p}_\mathcal{A} - \mathbf{p}_\mathcal{B}) \cdot (\dot{\mathbf{p}}_\mathcal{A}(t) - \dot{\mathbf{p}}_\mathcal{B}(t)) \quad (7)$$

However, it is equally difficult to keep track of $\mathbf{p}_\mathcal{A}$ and $\mathbf{p}_\mathcal{B}$ to calculate $\delta(\cdot)$ over a continuous time interval. So we set $\mathbf{p}_\mathcal{A}$ and $\mathbf{p}_\mathcal{B}$ as the closest point at t_0^i. Consequently, although $\delta(\cdot)$ may not have extremal values when $s_{rel} = 0$, we can still use LB^i in (6) as a lower bound, since E_j is always less than the difference between the extremals.

A motion bound μ is described as the maximum displacement of an object $\mathcal{A}$ for a given time, and it can be computed as follows:

$$\mu = \max_j \int_{t_0^i}^{t_1^i} \dot{\mathbf{p}}_j(t)dt \leq (\max|\mathbf{v}(t)| + \max|\omega(t)| \max_j |\mathbf{p}_j|)(t_1^i - t_0^i) \quad (8)$$

IV. Results and Discussions

We implemented our distance calculation algorithm using the C++ under a Kubuntu 10.04 LTS 64 bit operating system equipped with an Intel Core i7 2.67GHz CPU and 3GB of main memory. We have tested our MCD calculation method on an HRP-2 humanoid robot using various motion planning benchmarks. Our distance calculation algorithm is used to satisfy the non-penetration constraint in Eq. (1).

We have tested our distance calculation algorithm using three sets of benchmarks. In the first benchmark, hand-crossing, the robot tries to cross its arms twice while avoiding self-collisions (Fig. 3). The first benchmark, twisting, shows a case in which the robot twists its upper body while avoiding collisions between its arms and legs. The passing benchmark shows a case in which the robot is passing through Buddha statues while avoiding self-collisions and obstacle collisions. The robot in the last benchmark, putting, leans on a table and attempts to put an object inside a cup without creating a collision with the cup. Our algorithm takes 25.71∼125.39msec for robots and obstacles consisting of 100∼10K triangles.

V. Conclusion

We present an algorithm to evaluate distance functions for SIP-based motion planning algorithm and demonstrate its capability with the HRP-2 humanoid robot. Our algorithm can compute the MCD within a user-bounded error. However, the AS algorithm is still approximate when the robot circulates around obstacles, so AS may require a high number of subdivisions since the local bound might be loose due to the small difference between extremal values.

For future work, we would like to further improve the performance of our distance computation algorithm. We plan to work on a continuous and stable PD computation. Finally, we would like to extend the distance function to be able to handle the general penetration depth.

VI. Acknowledgements

This work was supported in part by NRF in Korea (grant number 2014K1A3A1A17073365) and MCST/KOCCA in the CT R&D program 2014 (grant number R2014060011).

References

[1] Y. Lee, A. Kheddar, and Y. J. Kim, "Continuous signed distance computation for capsule-shaped and polygonal robots," *submitted to IEEE Transactions on Robotics*.