The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Integrating Geometric Constraints into Reactive Leg Motion Generation

Fumio Kanehiro, Mitsuharu Morisawa, Wael Suleiman, Kenji Kaneko and Eiichi Yoshida

Abstract— **This paper proposes a reactive leg motion generation method which integrates geometric constraints into its generation process. In order to react given instructions instantaneously or to keep balance against external disturbances, feasible steps must be generated automatically in real-time for safety. In many cases this feasibility has been realized by using predefined steps or admissible stepping regions. However, these predefinitions are often too conservative or valid only in limited situations. The proposed method considers geometric constraints in addition to joint limits during its generation process and it can utilize the ability of the robot to a maximum extent. It can generate feasible walking pattern in real-time by modifying the swing leg motion and the next landing position at each control cycle. The proposed method is validated by experiments using a humanoid robot HRP-2.**

I. Introduction

Motion patterns have to respect not only constraints to achieve given tasks but also constraints such as joint angle/velocity limits and self-collision avoidance to be feasible. Especially when patterns are generated online to react given instructions instantaneously or to keep balance against external disturbances, feasible patterns must be generated within a short control period automatically. In order to generate feasible walking patterns online, a few predefined footprints or small landing regions have been often used. However, these step limitations are valid only in limited situations and often limit the robot's ability. Therefore, this paper proposes a feasible leg motion generation method which integrates geometric constraints into its generation process and utilize the robot's ability at maximum. Even when ill-placed footprints like shown in Fig.1(left) are given, the proposed method generates a feasible motion shown in Fig.1(right).

Motion planning methods to generate feasible motions have been well studied to generate motions for manipulators. Since humanoid robots have bodies which consists of several manipulators, we can use similar techniques but we need to consider issues peculiar to humanoid robots such as collisions between legs. Several online pattern generators which can generate safe patterns have been already proposed.

When a task is given as a set of constraints for some parts of the robot body such as an end-effector, joint angles which respect those constraints can be computed by solving inverse kinematics. [1] and [2] propose methods to add linear equality constraints to avoid self-collision which are defined based on distances between parts of the robot body. But those equality constraints are too strict and may narrow solution space. Constraints based on distances should be described as inequality constraints.

The authors are with Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology(AIST), Tsukuba Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki, 305-8568 Japan {f-kanehiro, m.morisawa, wael.suleiman, k.kaneko, e.yoshida}@aist.go.jp

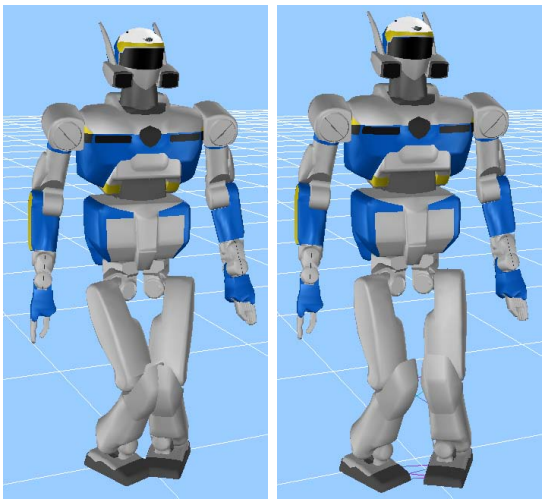

Fig. 1. Example postures generated without(left) and with(right) considering geometric constraints

[3] proposes a pattern generator which can stop walking safely when self-collisions are expected. This method uses a fast pattern generator[4] and checks self-collision very rapidly by tracking the minimum distances between shapes approximated by convex hulls using V-Clip[5]. If self-collision is expected in a newly generated pattern, a prepared pattern to stop walking is connected to the current pattern. This method can stop the robot safely but we want to continue the motion without stopping.

Some footstep planners such as [6] are using discrete footprints to grow search trees. Usually the number of those footprints is rather small and it limits the robot's walking ability. [7] propose a method to investigate a feasible stepping region offline and move the next landing position inside of the region if it is specified outside. Since the shape of this region depends on various conditions, for instance the waist height, the shape can be used only in limited situations.

This paper proposes a feasible leg motion generation method which can be used online. To make it possible to use 'online', the method must be able to generate appropriate leg motions within a control period even if the next landing position is changed every control cycle. In order to generate feasible motions, the proposed method considers geometric constraints in its pattern generation process. More precisely, geometric constraints are described as linear equality and inequality constraints and joint angles are computed under

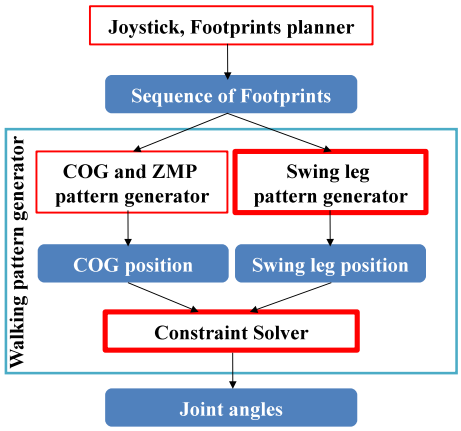

Fig. 2. Overall structure of walking pattern generator

those constraints by a constraint solver. And the method also provides a swing leg trajectory generator which considers feet placements to avoid the constraint solver falling in local minima. Moreover, a stiffness varying constraint and a landing position modification function is used to broaden the solution space.

This paper is organized as follows. In Section II the overall structure of the pattern generator is shown. In Section III the constraint solver to find feasible joint velocities is explained. In Section IV a swing leg trajectory generator is introduced to avoid the constraint solver falling in local minima. In Section V, landing positions are modified to improve the possibilities of feasible motions being obtained. We validate the proposed method by experiments in Section VI and summarize and conclude the paper in Section VII.

II. Structure of Pattern Generator

Fig.2 shows the overall structure of the walking pattern generator.

The pattern generator takes a sequence of footprints as an input. These footprints are generated by operating input devices such as joysticks, footstep planners such as [6] or the pattern generator itself to keep balance against external disturbances. Those footprints are passed to a COG(Center Of Gravity) pattern generator and a swing leg trajectory generator. The COG pattern generator is a core of the walking pattern generator and we are using the algorithm which can generate a COG trajectory and a ZMP(Zero Moment Point)[8] trajectory simultaneously and accept immediate modifications of footprints[9]. A constraint solver takes a COG position and a swing leg position as input and computes joint angles which realize those positions(In this work the waist position is constrained instead of COG position. The waist position is computed by adding a constant vector to the COG position. The constant vector is the offset vector between the waist and COG in the initial posture.). Usually, a conventional inverse kinematics solver is used as this constraint solver and it is one of the reasons for infeasible pattern being generated. Another reason is that usually a swing leg trajectory generator connects footprints

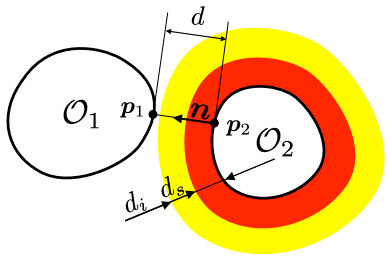

Fig. 3. Collision avoidance: Faverjon and Tournassoud's method

by straight lines in the horizontal plane. This trajectory may cause cross-legged situations. Therefore, we generates feasible patterns by combining a constraint solver which considers geometric constraints and a swing leg trajectory generator which considers feet placements.

III. Constraint Solver

The constraint solver described in this section is specialized to the leg motion generation, but its framework is same with the method we proposed in [10] and can be used to generate other motions.

Joint angles which respect a waist position/orientation and a swing foot position/orientation can be computed by using joint velocities $\dot{\boldsymbol{q}}$ obtained by solving the following optimization problem.

$$\begin{aligned} & \min_{\dot{\boldsymbol{q}}} \|\dot{\boldsymbol{q}}\|^2 \\ \text{subject to} \quad & \boldsymbol{J}_{waist}\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}_{waist} \\ & \boldsymbol{J}_{foot}\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}_{foot} \end{aligned} \tag{1}$$

where $\boldsymbol{J}_{waist}$ and $\boldsymbol{J}_{foot}$ are Jacobian matrices of the waist and the swing foot with respect to the support foot respectively. $\dot{\boldsymbol{x}}_{waist}$ and $\dot{\boldsymbol{x}}_{foot}$ are velocities of 6×1 required to reach reference position/orientation of the waist and the swing foot respectively.

By adding geometric constraints to this problem, feasible joint velocities are computed and by updating joint angles using the joint velocities at each control cycle, feasible patterns are generated.

A. Constraint for collision avoidance

In order to avoid self-collision, *velocity damper* proposed by Faverjon et al. is used. *velocity damper* is a constraint based on a distance between two objects and defined as follows[11].

$$\dot{d} \geq -\xi \frac{d - d_s}{d_i - d_s} \quad \text{if} \quad d < d_i \tag{2}$$

where d is the minimum distance between two objects. ξ is a positive coefficient to adjust the convergence speed. d_i and d_s are distances where *velocity damper* is activated and the minimum distance maintained between objects respectively(see Fig.3). This constraint prevents objects from coming closer too fast and as the result distance between them never be smaller than d_s.

Eq.(2) can be rewritten using $\dot{\boldsymbol{q}}$ as follows.

$$\boldsymbol{n}^T(\boldsymbol{J}_{\boldsymbol{p}_1} - \boldsymbol{J}_{\boldsymbol{p}_2})\dot{\boldsymbol{q}} \geq -\xi\frac{d - d_s}{d_i - d_s} \tag{3}$$

where $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are the closest points on objects and $\boldsymbol{n} = (\boldsymbol{p_1} - \boldsymbol{p_2})/\|\boldsymbol{p_1} - \boldsymbol{p_2}\|$ is a normal vector. Note that in order to get continuous velocities using this constraint, $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ have to move continuously on surfaces of objects. In other words, objects must be strictly convex objects.

B. Constraint for joint angle/velocity

A constraint to respect joint angle and velocity limits with continuous joint velocities can be also defined using *velocity damper* as follows.

$$\dot{q}_j^{max}(q_j) \geq \dot{q}_j \geq \dot{q}_j^{min}(q_j), \text{ for } j \in \{1, ..., n\} \tag{4}$$

where n is the number of joints. $\dot{q}_j^{max}(q_j)$ and $\dot{q}_j^{min}(q_j)$ are determined as follows.

$$\dot{q}_j^{max}(q_j) = \begin{cases} \xi_v \dfrac{(q_j^+ - q_j) - q_s}{q_i - q_s} & \text{if } q_j^+ - q_j \leq q_i \\ \dot{q}_j^+ & \text{otherwise} \end{cases} \tag{5}$$

$$\dot{q}_j^{min}(q_j) = \begin{cases} -\xi_v \dfrac{(q_j - q_j^-) - q_s}{q_i - q_s} & \text{if } q_j - q_j^- \leq q_i \\ \dot{q}_j^- & \text{otherwise} \end{cases} \tag{6}$$

where q_j^+ and q_j^- are physical limits of joint angles, $\dot{q}_j^+$ and $\dot{q}_j^-$ are joint velocity limits, ξ_v, q_i and q_s are parameters which corresponds to ξ, d_i and d_s in Eq.(2) respectively.

C. Stiffness varying constraint

Even though humanoid robots are highly redundant system, the number of residual degrees of freedom is not so large if trajectories of some parts of the robot body are specified by a pattern generator. In the case of walking pattern generation, it is not possible to avoid collisions between legs if trajectories of the waist and feet are specified. However in the case of walking pattern generation, the swing leg need not to track the specified trajectory strictly while it is in the air. We can relax constraints by rewriting the optimization problem(1) as follows.

$$\begin{aligned} &\min_{\tilde{\dot{\boldsymbol{q}}}} \tilde{\dot{\boldsymbol{q}}}^T \boldsymbol{W} \tilde{\dot{\boldsymbol{q}}} \\ \text{subject to} \quad &\tilde{\boldsymbol{J}}_{waist}\tilde{\dot{\boldsymbol{q}}} = \dot{\boldsymbol{x}}_{waist} \\ &\tilde{\boldsymbol{J}}_{foot}\tilde{\dot{\boldsymbol{q}}} = \dot{\boldsymbol{x}}_{foot} \end{aligned} \tag{7}$$

where $\tilde{\dot{\boldsymbol{q}}}, \boldsymbol{W}, \tilde{\boldsymbol{J}}_{waist}$ and $\tilde{\boldsymbol{J}}_{foot}$ are given by

$$\begin{aligned} \tilde{\dot{\boldsymbol{q}}} &= [\dot{\boldsymbol{q}}^T \ \boldsymbol{e}^T]^T, \\ \boldsymbol{W} &= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{W}_e \end{bmatrix}, \\ \tilde{\boldsymbol{J}}_{waist} &= \begin{bmatrix} \boldsymbol{J}_{waist} & \boldsymbol{S}_{waist} \end{bmatrix}, \\ \tilde{\boldsymbol{J}}_{foot} &= \begin{bmatrix} \boldsymbol{J}_{foot} & \boldsymbol{S}_{foot} \end{bmatrix}. \end{aligned} \tag{8}$$

$\boldsymbol{e}$ is a new variable which contains errors of relaxed constraints. Its dimension n_e is the number of relaxed constraints. $\boldsymbol{W}_e$ is a diagonal weighting matrix of $n_e \times n_e$ which adjust stiffness of relaxed constraints. If we choose large value, the constraint becomes hard and if we do small, it becomes soft. In the case of the swing leg position constraint, the swing leg can deviate from the reference trajectory while it is in the air but it has to come back on it until the landing time. So the stiffness is set to large in the beginning and the end of single support phase to keep the landing position, and to small in the middle to avoid collisions. $\boldsymbol{S}_{waist}$ and $\boldsymbol{S}_{foot}$ are selection matrices of $6 \times n_e$ to pick up constraints which are relaxed.

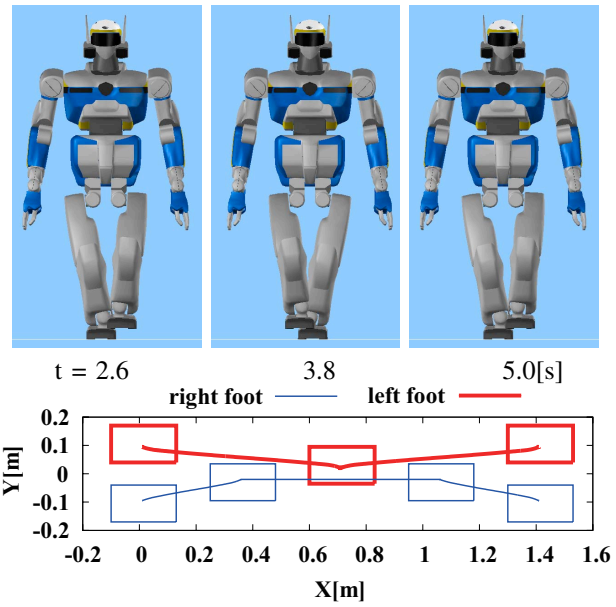

Fig. 4. Foot trajectories and postures generated by our conventional pattern generator. There are collisions between feet.

Another way to realize collision avoidance between legs would be to use a prioritization scheme. It is expected that the swing leg can avoid collision by putting a higher priority to a collision avoidance task and a lower one to a swing leg motion task. Although a problem solver for prioritized equality and inequality tasks already exists[12], its computational cost is high since it needs to solve one QP problem at each priority level. Since lower computational cost is preferable for the reactive motion generation, we choose the proposed method.

D. Example: Cross-legged walking

In order to confirm that feasible patterns can be generated using the proposed method, a walking pattern is generated from footprints that causes collisions between feet. The constraint solver is implemented using uQuadProg[13], the modified function of QuadProg++[14] and the target robot is HRP-2[15].

Fig.4 shows a walking pattern generated without using the proposed method. The upper row shows postures of the robot at the moments when the swing leg passes side of the support leg. The lower row shows footprints and trajectories of feet. Since the lateral distances between footprints are set

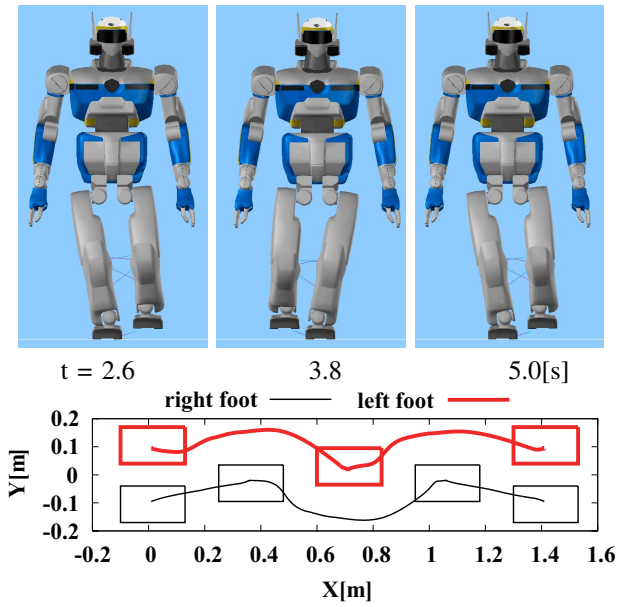

Fig. 5. Foot trajectories and postures generated using the constraint solver which considers geometric constraints. The swing leg is pushed outside to avoid collision.

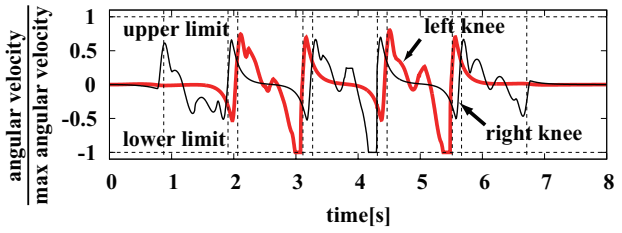

Fig. 6. Ratios of knee joint velocities against those limits during cross-legged walking. Velocities never exceed limits.

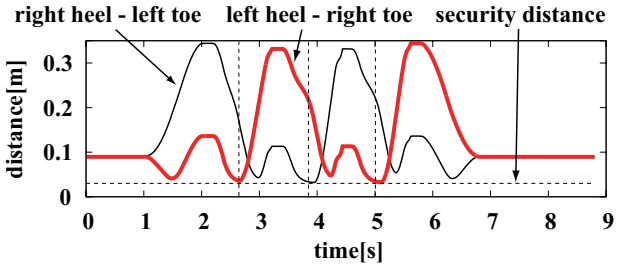

Fig. 7. Distances between feet while walking. Distances are always bigger than the security distance. Vertical lines shows moments when snapshots in Fig.5 were taken.

to small and our conventional swing leg trajectory generator tries to connect footprints by straight lines, feet collide. (Feet trajectories shown in Fig.4 are not straight. Because foot rotations such as heel contact and toe takeoff changes its shape slightly.)

In order to compute distances for Eq.(2), geometries of the robot are required. As mentioned above, those geometries must be strictly convex to obtain continuous joint velocities. Therefore, we can not use triangle meshes which describe geometries of HRP-2. We need to approximate them by strictly convex shapes such as spheres or STP-BV[16]. We proposed a collision avoidance method which can use triangle meshes as geometries[17], but it is not suitable for online pattern generation since it requires to solve many constraints. Therefore, we use spheres to approximate geometries. We approximate shank links by three spheres for each and foot links by two spheres. We didn't approximate thigh links since there is no collision between them due to large space realized by cantilever structure of hip joints and joint limits.

The proposed method is used to generate 12 leg joints motions. A swing leg position in the horizontal plane is constrained by stiffness varying constraints. As the result, the dimension of $\tilde{\boldsymbol{q}}$ is 14. $\boldsymbol{e}, \boldsymbol{S}_{waist}$ and $\boldsymbol{S}_{foot}$ are given by

$$\boldsymbol{e} = \begin{bmatrix} e_x \\ e_y \end{bmatrix}, \ \boldsymbol{S}_{waist} = \boldsymbol{0}, \ \boldsymbol{S}_{foot} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ & \boldsymbol{0} \end{bmatrix}. \quad (9)$$

The stiffness values in $\boldsymbol{W}_e$ are set to 10^5 in the beginning and the end of single support phases and to 10^3 in the middle. These values are chosen empirically so that the displacement of the swing leg becomes negligible at the landing position and it doesn't become too large even in the middle of the single support phase. 12 equality constraints are used for the waist position/orientation and feet position/orientation(including two stiffness varying constraints). 24 inequality constraints are used for joint angle/velocity limits. 8 inequality constraints are used for self-collision avoidance(8 is the maximum number and it changes according to distances.). A leg motion is generated under these 44 constraints.

The upper row of Fig.5 show postures which correspond to postures in the upper row of Fig.4. The lower row shows generated feet trajectories and we can see the swing leg is pushed outside to avoid collisions.

Fig.6 shows ratios of knee joint velocities against those limits. Vertical dotted lines indicates boundaries between single support and double support phases. In the latter half of the single support phase, large joint velocities are required to stretch knees but they are limited under those limits.

Fig.7 shows distances between spheres which approximate feet geometries. Vertical lines shows moments when snapshots in Fig.5 were taken. In this case we chose 0.03[m], 0.2[m] and 1.5[m/s] for d_s, d_i and ξ in Eq.(3) respectively and can confirm distances are always larger than 0.03[m].

IV. Swing Leg Trajectory Generation

The proposed method in the previous section is a local method which finds joint velocities only using local(current) information. In order to generate feasible patterns using this method, it is required that input footprints are well designed and a feasible motion can be obtained by applying comparatively small changes. For instance, if smaller lateral distances between feet are used in the previous example to cross legs more deeply, the proposed method will fail. Because the swing leg will try to pass the wrong side(outside) of the support leg and it won't be able to reach the landing

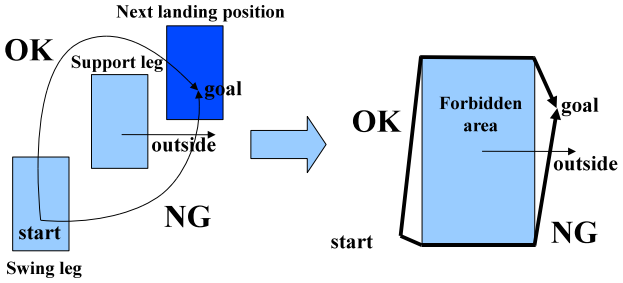

Fig. 8. Two candidates of a swing leg trajectory. The shortest path which doesn't cross with the outside vector is used to generate a swing leg trajectory.

position due to collisions with the support leg. This is an essential issue of local methods.

In the case of walking pattern generation, we can use the next landing position as future information. If a swing leg trajectory generator can provide a collision-free trajectory, the constraint solver will not fall in such kind of local minima. Of course, such a collision-free swing leg trajectory can be planned using global planning methods such as RRT[18]. However, these methods are too heavy to compute leg motion at each control cycle. Therefore the swing leg trajectory generator only considers collisions between feet and we assume that residual collisions are avoided by the constraint solver.

Fig.8(left) shows two candidates of the swing leg trajectory. If the swing leg tries to pass outside of the support leg, collisions between legs will prevent the swing leg from reaching the landing position. Therefore the trajectory which does not cross with a half line which starts from the center of the support leg and directs outside is selected. A region where the center of the swing leg should not pass is computed as Minkowski sum of the support leg polygon and a polygon which moved to the origin after inverting the swing leg polygon. Using this Minkowski sum, two shortest paths from the current position to the landing position are computed and then the appropriate one is chosen.

If the swing leg track the chosen path, its velocity becomes discontinuous when it passes corners and when shape of the shortest path changes due to the next landing position modification. In order to prevent these discontinuous cases, the swing leg trajectory is generated by the following algorithm.

Algorithm 1 calcSwingLegPosition($path, t_{remain}, x, v, a$)

1: **if** $t_{remain} > t_{offset}$ **then**
2: $target \leftarrow$Divide($path, \Delta t/(t_{remain} - t_{offset})$)
3: HoffArbib($t_{offset}, target, x, v, a$)
4: **else**
5: $target \leftarrow$Divide($path$, 1.0)
6: HoffArbib($t_{remain}, target, x, v, a$)
7: **end if**

Arguments $path$ and t_{remain} are the chosen path and remaining time of the single support phase respectively. x, v and a are current position, velocity and acceleration of the swing leg. The function Divide($path, ratio$) computes a position which divide $path$ internally by $ratio$. Δt is a control period. Using this algorithm, $target$ moves on the shortest path with a constant speed while the next landing position is not changed and reaches the landing position t_{offset} earlier than the landing time. The swing leg pursues $target$ with time delay t_{offset} using the minimum jerk trajectory generation method proposed by Hoff and Arbib[19] shown in algorithm(2). In algorithm(2), x, v and a are current position, velocity and acceleration respectively. $goal$ and t_{remain} are the goal position and remaining time to reach the goal respectively. Using this method, the swing leg moves with continuous acceleration even if the next landing position is changed at each control cycle.

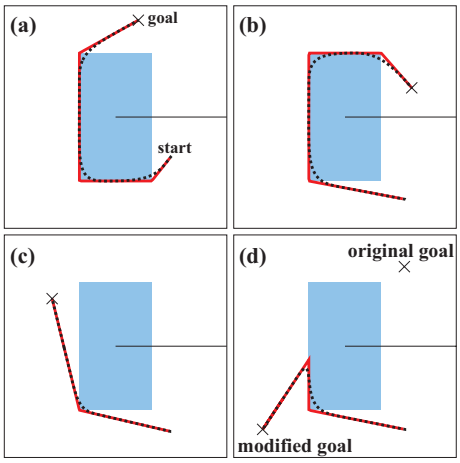

Fig. 9. Examples of swing leg trajectories. (a),(b) and (c) are cases landing positions are not changed. (d) is a case it is changed.

Algorithm 2 HoffArbib($t_{remain}, goal, x, v, a$)

1: $\dot{a} \leftarrow -9a/t_{remain} - 36v/t_{remain}^2 + 60(goal - x)/t_{remain}^3$
2: $a \leftarrow a + \dot{a}\Delta t$
3: $v \leftarrow v + a\Delta t$
4: $x \leftarrow x + v\Delta t$

Fig.9 shows examples of the shortest paths and generated swing leg trajectories. Gray boxes are regions where the swing leg should not pass and half lines show outside direction. Cross marks indicate landing positions. Thick solid lines and dotted lines are the shortest paths and swing leg trajectories respectively. Example(a), (b) and (c) are cases landing positions are not changed. Example(d) is a case the landing position is changed. Generated swing leg trajectories intrude forbidden regions slightly and it means that collisions may happen. We admit these intrusions assuming these collision will be solved by the constraint solver.

V. Landing Position Modification

So far we assumed that landing positions are always possible under geometric constraints. In reality, however, some of them are impossible to reach. In order to make it possible to generate feasible patterns from these footprints,

we add the landing position modification function to our method. Both of the swing leg trajectory generator and the constraint solver adjust the next landing position at each control cycle.

A. Modification by swing leg trajectory generator

If footprints are generated without considering foot shapes and there might be overlaps between footprints. Therefore, overlap between feet at the next landing position is checked before computing the swing leg reference position using the method described in Section IV. If it is expected, the overlap is eliminated by the minimum translation of the next landing position. Overlaps between feet are detected by checking whether the origin is in Minkowski sum of the support leg polygon and the inverted swing leg polygon or not. If the origin is in it, the minimum translation to solve the overlap is a vector from the origin to the point on the boundary of the Minkowski sum which is closest to the origin. The next landing position is updated using this vector.

B. Modification by constraint solver

Even if there is no overlap between footprints, some of landing positions are unreachable due to collisions between legs or other limits. When the next landing position is unreachable, some of reference positions generated by the swing leg trajectory generator are also unreachable. When an unreachable reference position is given while a constraint for the swing leg position is relaxed using Eq.(9), the computed position has displacement from the reference position. The amount of the displacement is obtained as components of $\boldsymbol{e}$ in Eq.(9). The next landing position $[p_x \ p_y]^T$ is updated using these components as follows.

$$p_x := p_x - e_x \Delta t \tag{10}$$

$$p_y := p_y - e_y \Delta t \tag{11}$$

where Δt is a control period. Repeating this procedure at each control cycle, the next landing position is moved to the reachable area.

C. Examples

1) Deeply cross-legged walking: Fig.10 shows an example of the landing position modification. The upper row shows footprints and feet trajectories. Dotted lines show trajectories generated by our conventional swing leg trajectory generator and solid lines do trajectories generated by new swing leg trajectory generator. When conventional one is used, the right foot tries to pass left side of the left leg in the second step. As the result, the right leg is stopped to avoid collisions and the pattern generation fails. When new one is used, it passes right side and the pattern generation continues. Rectangles drawn by dotted lines shows given footprints and the others are realized footprints. The second and third steps are placed outside of the support leg and these landing positions are not reachable, they are moved inside. The lower row of Fig.10 shows snapshots of the second step. Black rectangles drawn on the floor are given landing positions. In the beginning it is placed left side of the left foot. The right

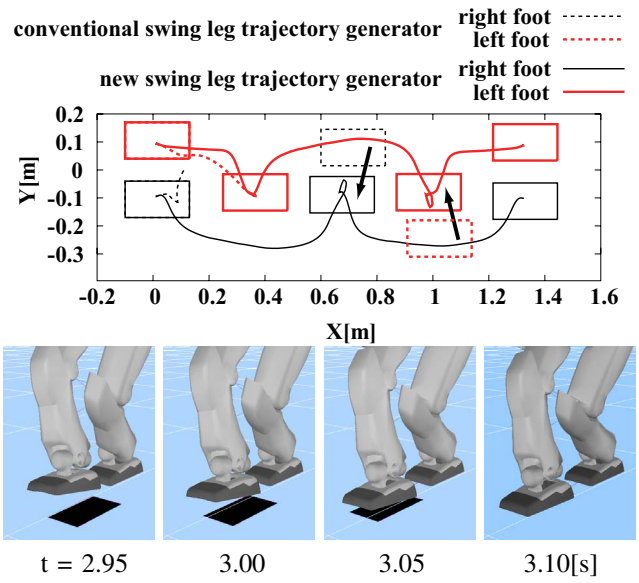

Fig. 10. Example of the next landing position modification. The second and third landing positions are modified since they are not reachable.

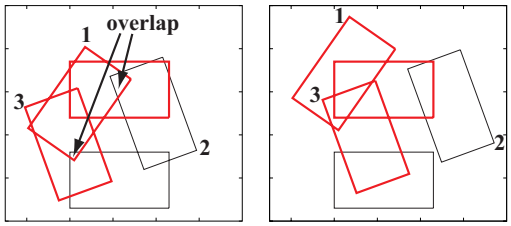

Fig. 11. Original(left) and modified(right) footprints of a wide step turning motion. Overlaps in the first and second steps are eliminated by the minimum translations.

leg tries to reach there but it is not possible because knees come closer. As the result, the landing position is moved right until the foot can reach the modified position.

2) Wide step turning: Fig.11 shows footprints of a wide step turning motion shown in Fig.1. This motion opens the left leg 55[deg] in the first step, close the right leg 55[deg] and align the left leg parallel. Fig.1 shows its second step. Fig.11(left) shows given footprints. There are overlaps in the first step and the second step. Fig.11(right) shows realized footprints and we can confirm those overlaps are solved.(There are gaps between feet because we are keeping 3[cm] margin between feet).

The upper row of Fig.12 shows the waist height during generating this motion and the lower row does distances between two pairs of spheres attached to knees. To generate this motion, we add a soft constraint for the waist height. $\boldsymbol{e}, \boldsymbol{S}_{waist}$ and $\boldsymbol{S}_{foot}$ are given by

$$\boldsymbol{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}, \ \boldsymbol{S}_{waist} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ & \boldsymbol{0} & \end{bmatrix}, \ \boldsymbol{S}_{foot} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ & \boldsymbol{0} & \end{bmatrix}. \tag{12}$$

A weight for e_z is set to 1.0.

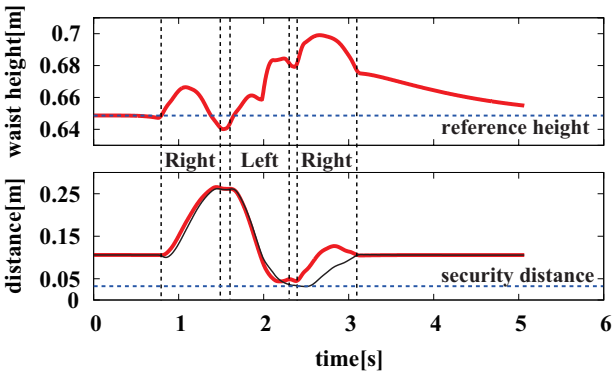

Fig. 12. Waist height and distances between two pairs of spheres attached to knees while turning. The waist is pushed up to avoid collisions between knees.

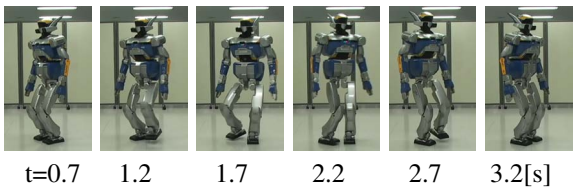

Fig. 14. Snapshots of experiments: wide step turning

During the latter half of the single support phase of the second step, distances between knees becomes small. The waist is pushed up by stretching knees in order to avoid collision between knees. We can see the waist height is also changed during the first step even though distances are big enough. This is because specified swing leg height is realized not only by bending the swing leg, but also stretching the support leg to minimize the objective function in the optimization problem(7).

VI. Experiments

In order to confirm feasible patterns are generated, we tested patterns generated from footprints shown in Fig.4 and Fig.11 on the real robot HRP-2. To generate these patterns, $\boldsymbol{e}$ in Eq.(12), the swing leg trajectory generator which considers feet placement and a landing position modification function are used. Fig.13 and Fig.14 show snapshots of experiments. The robot can walk stably using these patterns.

In the case of HRP-2, the control period Δt is 5[ms] and a balance controller and some other processing must be executed within the period in addition to the walking pattern generator. Therefore we can use 3[ms] at maximum to generate patterns. In order to keep real-time constraint, the worst case time must be smaller than 3[ms].

Fig.15 shows computational time to generate a cross-legged walking motion shown in Fig.13. It takes 0.3[ms] on an average and 0.7[ms] at maximum. The optimization problem(7) has 15 parameters and 44 constraints. The computational time is enough short even though the processor on HRP-2 is not so powerful, Pentium III 1.26[GHz]. We can apply our method to more redundant systems or add more constraints.

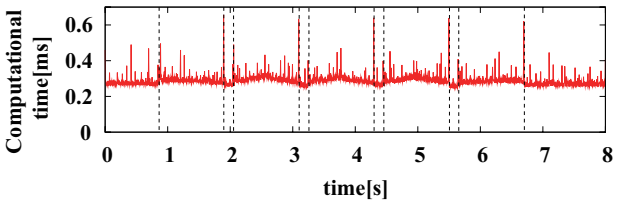

Fig. 15. Computational time to generate cross-legged walking. It is enough short to generate patterns online.

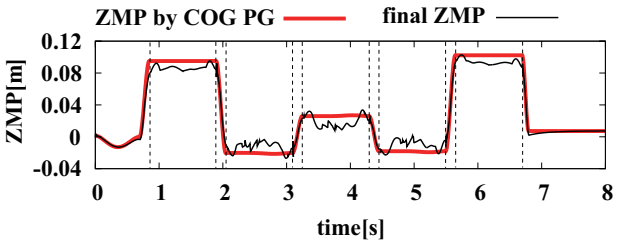

Fig. 16. ZMP trajectory generated by COG pattern generator and final ZMP trajectory. ZMP error is around 2[cm] and enough small.

Dynamic balance is an indispensable constraint for humanoid robots. But it is not considered in the proposed method. Because we assume that modifications to respect geometric constraints don't have so big effect on ZMP. Fig.16 shows lateral elements of ZMP trajectories for "Cross-legged walking". The thick line shows ZMP generated by the COG pattern generator and the thin one does ZMP computed by applying generated joint trajectories to the multi-body model. Differences between these two trajectories are around 2[cm] and it includes errors between the single mass point model and the multi-body model. We can say that the proposed method doesn't have big effect on ZMP at least in this case.

However, in the case of "Deeply cross-legged walking", the landing position are changed just before landing and modifications like this cause large ZMP fluctuation[20]. Fig.17 shows an ideal ZMP trajectory and a trajectory generated by the COG pattern generator. Since landing positions for the second and third steps are modified, unacceptable ZMP fluctuations are generated. To solve this issue, we have to improve both of the COG pattern generator and the landing position modification algorithm. This is one of our future works.

VII. Summary and Future Works

This paper proposed a reactive leg motion generation method which considers geometric constraints to generate feasible motions. This method consists of the constraint solver and the swing leg trajectory generator. The constraint solver describes geometric constraints as linear equality and inequality constraints and find joint velocities which respect all the given constraints by solving the optimization problem. The swing leg trajectory generator considers foot placements and gives better reference trajectory to the constraint solver. Moreover, we added a stiffness varying constraint and a landing position modification function to improve the possibilities of feasible solutions being obtained.

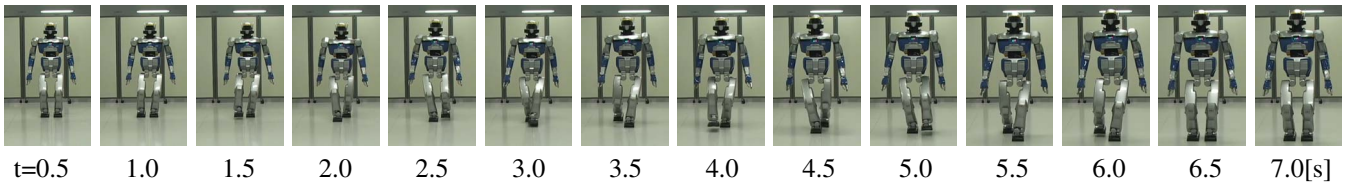

Fig. 13. Snapshots of experiments: cross-legged walking

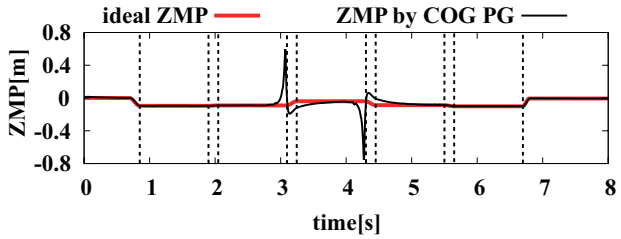

Fig. 17. ZMP trajectory generated by COG pattern generator and ideal ZMP trajectory. Big fluctuations are caused by landing position modifications just before landing.

Since given footprints may be modified, the robot may reach an unexpected position. To solve this issue, the walking pattern generator need to inform the modified footprints to the footstep planner and the foot step planner update footprints as needed.

The proposed method can not guarantee that a feasible pattern is always generated. For instance, since both feet are fixed during the double support phase and the residual number of freedom becomes small, the solution space might be empty in some cases. To solve this issue, we need to relax some of hard constraints to obtain feasible motions. For example, it is possible to enlarge the solution space by relaxing a constraint for rotation of the waist link around a vertical axis (Readers can see the example in the accompanied video segment.). This is one of our future works.

ACKNOWLEDGMENTS

This work has been partly supported by JST-CNRS Strategic Japanese-French Cooperative Program "Robot motion planning and execution through online information structuring in real-world environment". The work by W. Suleiman was supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS).

REFERENCES

[1] H. Sugiura, M. Gienger, H. Janssen, and C. Goerick, "Real-Time Collision Avoidance with Whole Body Motion Control for Humanoid Robots," in *Proc. of International Conference on Intelligent Robots and Systems*, 2007, pp. 2053–2058.

[2] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar, "Real-Time (Self)-Collision Avoidance Task on a HRP-2 Humanoid Robot," in *Proc. of the 2008 IEEE International Conference on Robotics & Automation*, 2008, pp. 3200–3205.

[3] J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Self-Collision Detection and Prevention for Humanoid Robots," in *Proc. of the 2002 IEEE International Conference on Robotics & Automation*, 2002, pp. 2265–2270.

[4] K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Online Generation of Humanoid Walking Motion based on a Fast Generation Method of Motion Pattern that Follows Desired ZMP," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS'02)*, 2002, pp. 2684–2689.

[5] M.Mirtich, "VClip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177–208, 1998.

[6] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Online Footstep Planning for Humanoid Robots," in *Proc. of the 2003 IEEE International Conference on Robotics & Automation*, 2003, pp. 932–937.

[7] N. Perrin, O. Stasse, F. Lamiraux, P. Evrard, and A. Kheddar, "On the Problem of Online Footsteps Correction for Humanoid Robots," in *Proc. 27th Annual Conference of Robotics Society of Japan*, 2009, pp. RSJ2009AC3O1–04.

[8] M.Vukobratović and B.Borovac, "Zero-moment point –thirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004.

[9] M.Morisawa, K.Harada, S.Kajita, K.Kaneko, F.Kanehiro, K.Fujiwara, S.Nakaoka, and H.Hirukawa, "A Biped Pattern Generation Allowing Immediate Modification of Foot Placement in Real-time," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 581–586.

[10] F. Kanehiro, W. Suleiman, K. Miura, M. Morisawa, and E. Yoshida, "Feasible Pattern Generation Method for Humanoid Robots," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, 2009, pp. 542–548.

[11] B. Faverjon and P. Tournassoud, "A Local Based Approach for Path Planning of Manipulators With a High Number of Degrees of Freedom," in *Proc. of IEEE International Conference on Robotics and Automation*, 1987, pp. 1152–1159.

[12] O. Kanoun, F. Lamiraux, F. Kanehiro, E. Yoshida, J.-P. Laumond, and P.-B. Wieber, "Prioritizing Linear Equality and Inequality Systems: Application to Local Motion Planning for Redundant Robot," in *Proc. of the 2009 IEEE International Conference on Robotics & Automation*, 2009, pp. 2939–2944.

[13] "uQuadProg," http://www.lis.deis.unical.it/ furfaro/uQuadProg/uQuadProg.php.

[14] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming*, vol. 27, pp. 1–33, 1983.

[15] K. Kaneko, F. Kanehiro, S. Kajita, M. Hirata, K. Akachi, and T. Isozumi, "Humanoid Robot HRP-2," in *Proc. of the 2004 IEEE International Conference on Robotics & Automation*, 2004, pp. 1083–1090.

[16] M. Benallegue, A. Escande, S. Miossec, and A. Kheddar, "Fast C^1 Proximity Queries using Support Mapping of Sphere-Torus-Patches Bounding Volumes," in *IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, 12-17 May 2009, pp. 483–488.

[17] F. Kanehiro, F. Lamiraux, O. Kanoun, E. Yoshida, and J.-P. Laumond, "A Local Collision Avoidance Method for Non-strictly Convex Polyhedra," in *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.

[18] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 98-11, 1998.

[19] B. Hoff and M. A. Arbib, "Models of Trajectory Formation and Temporal Interaction of Reach and Grasp," *Journal of Motor Behavior*, vol. 25, no. 3, pp. 175–192, 1993.

[20] M. Morisawa, K. Harada, S. Kajita, K. Kaneko, J. Sola, E. Yoshida, N. Mansard, K. Yokoi, and J.-P. Laumond, "Reactive Stepping to Prevent Falling for Humanoids," in *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, 2009, pp. 528–534.