The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Decentralized Planning for Dynamic Motion Generation of Multi-link Robotic Systems

Yuichi Tazaki, Hisashi Sugiura, Herbert Janssen and Christian Goerick

Abstract— **This paper presents a decentralized planning method for generating dynamic whole body motions of multi-link robots including humanoids. First, a robotic system will be modeled as a general multi-body dynamical system. The planning problem of a multi-body system will then be formulated as a constraint resolution problem. The problem will be solved by means of an extended Gauss-Seidel method, which is capable of handling multiple constraint groups with different priorities. The method will be demonstrated in whole-body motion generation tasks of a humanoid, both in numerical simulations and in experiments using a real humanoid robot.**

I. INTRODUCTION

Complex multi-link robots such as humanoid robots have a potential for performing multiple tasks simultaneously under various constraint conditions by making use of its large degrees of freedom. However, to design a controller that exploits this feature is extremely challenging. To date, several methods have been proposed: Kuffner et al [1] proposed a method that consists of two phases, in which a statically stable and collision free trajectory is generated using a randomized planner [2] in the first phase, and it is shaped to be dynamically consistent using a filtering module in the second phase. Two-phase approaches has also been taken by Yamane et al [3] for creating computer animation of human figures doing manipulation tasks, and by Yoshida et al [5], Harada et al [4] for dynamic motion planning of humanoid robots.

On the other hand, Sentis and Khatib [6] proposed a framework based on task prioritization. They first categorize the objectives of humanoid motion planning into three levels: constraints, movements in operational spaces, and postures. Each objective is expressed by means of a Jacobian matrix, which decomposes a whole configuration space into a task-space and a null-space, which are orthogonal to each other. Motions of lower priority are then projected onto the null space of higher priority objectives, so that they will never interfere with the motions of higher priority.

Although the existing methods have been successful to a certain extent, they seem to have limitations as well: First, the task-prioritization-based method is purely reactive; it makes no prediction into the future. There certainly are a class of tasks that requires planning; such tasks include generation of gait patterns, minimization of long-term energy consumption and many others. On the other hand, randomized search methods seem to suffer from a fundamental difficulty in sampling from the set of feasible configurations of the robot. In many cases, the volume of feasible configuration space is much smaller than that of the whole configuration space, making the sampling of feasible configurations inefficient. Currently, this problem is handled by introducing a low-dimensional parametrization of the feasible region, which is highly problem-dependent.

Motivated by the above backgrounds, this paper presents a decentralized planning method for robots with high degrees of freedom that is based on constrained optimization. The key idea underlying our method is that planning process should make use of gradient-based techniques as much as possible, and randomized techniques should be used to assist the gradient-based method to avoid local optimal solutions. In the proposed method, the robot is modeled as a collection of rigid bodies connected by holonomic constraints. Unlike conventional techniques that define decision variables in joint coordinate space, all state variables of the multi-body system (position and velocity of rigid bodies, constraint forces and so forth) are directly used for planning. The motion generation problem is then formulated as a large-scale constrained optimization problem. A feasible gradient descent direction subject to the constraints is obtained by computing Lagrange multipliers using an extended Gauss-Seidel method. Making use of the sparsity of the graph structure, the computational complexity of one iteration of the Gauss-Seidel method is linear with respect to the number of rigid bodies and that of joints. One advantage of this graph-based formulation is that it explicitly computes the constraint forces and thus it is especially suitable for tasks that involve interactions between different parts of the body such as dynamic balancing. This feature will be demonstrated in the body moment generation task of a humanoid robot, in which the robot is required to generate a desired counter moment on its body by swinging its arms. Moreover, the proposed method is applicable not only to humanoids but in principle to any type of multi-linked robotic systems.

The rest of this paper is organized as follows: Section II gives a brief explanation to the mathematical model of multi-body systems. Next, in Section III, the decentralized planning technique is explained in detail. In Section IV, numerical simulations and demonstrations using a real humanoid robot are shown. Finally, Section V summarizes this paper with comments on further extensions.

II. MULTI-BODY SYSTEMS

In this section, we introduce a multi-body system, which is composed of a collection of multiple rigid bodies connected

Y. Tazaki is with the Department of Mechanical Science and Engineering, Nagoya University, Nagoya, Japan. tazaki@nuem.nagoya-u.ac.jp
H. Sugiura, H. Janssen and C. Goerick are with Honda Research Institute Europe, Offenbach, Germany.

TABLE I
LIST OF VARIABLES

States of rigid body i at time t:

$p_{i,t} \in \mathbb{R}^3$	: position	$q_{i,t} \in \mathcal{Q}$	: orientation
$v_{i,t} \in \mathbb{R}^3$	: velocity	$\omega_{i,t} \in \mathbb{R}^3$	: angular velocity
$f_{i,t} \in \mathbb{R}^3$	: force	$\tau_{i,t} \in \mathbb{R}^3$	: moment

States of joint (i, j) at time t:

$\theta_{i,j,t} \in \mathbb{R}$	: joint angle
$f_{i,j,t} \in \mathbb{R}^3$	: constraint force
$\tau_{i,j,t} \in \mathbb{R}^3$	: constraint moment

together by holonomic constraints. Table I lists all the variables use for modeling the time-evolution of a multi-body system. The symbol $\mathcal{Q}$ denotes the set of unit quaternions. All the above variables are expressed with respect to the global coordinate frame. The kinematics and dynamics of the i-th rigid body are described by the following rules:

$$p_{i,t+1} = p_{i,t} + h\, v_{i,t}, \tag{1}$$
$$q_{i,t+1} = q(h\, \omega_{i,t})\, q_{i,t}, \tag{2}$$
$$m_i\, v_{i,t+1} = m_i\, v_{i,t} + h\, f_{i,t}, \tag{3}$$
$$I_{i,t+1}\, \omega_{i,t+1} = I_{i,t}\, \omega_{i,t} + h\, \tau_{i,t}. \tag{4}$$

Here, $h \in \mathbb{R}$ denotes the step size of the Euler stepping. The function $q(\omega)$ returns a quaternion representing a rotation along the vector $\omega/\|\omega\|$ with the rotation angle $\|\omega\|$. See Appendix for a concrete definition. Moreover, m_i and $I_{i,t}$ denote the mass and the inertia matrix of the i-th rigid body, respectively. All variables related to rigid bodies are expressed with respect to the global coordinate frame.

Now, let us denote by Θ a set of pairs of indices indicating which pair of rigid bodies are connected by a joint. Here we assume for any $(i, j) \in \Theta$, $i < j$. Let $p^{\mathrm{J}}_{i,j}$, $q^{\mathrm{J}}_{i,j}$ ($p^{\mathrm{J}}_{j,i}$, $q^{\mathrm{J}}_{j,i}$) be the displacement and the orientation of the joint expressed in the local coordinate frame of the i-th (j-th) rigid body. Then the holonomic constraint expressing the joint $(i, j) \in \Theta$ is expressed as follows:

$$\begin{aligned} p_{i,t} + q_{i,t}\, p^{\mathrm{J}}_{i,j} &= p_{j,t} + q_{j,t}\, p^{\mathrm{J}}_{j,i}, \\ (q_{i,t}\, q^{\mathrm{J}}_{i,j})^{-1} (q_{j,t}\, q^{\mathrm{J}}_{j,i}) &= q(e^{\mathrm{z}}\, \theta_{i,j,t}) \end{aligned} \tag{5}$$

where $e^{\mathrm{z}} = [0, 0, 1]^{\mathrm{T}}$ is a unit vector which determines the joint axis direction. Here we assume that joints are of revolutive type, but other various joint types including prismatic joints and spherical joints can be expressed in similar forms.

On the other hand, the total force(moment) applied to the i-th rigid body is the sum of all constraint forces(moments) and external forces(moments), thus the following hold:

$$f_{i,t} = f^{\mathrm{ext}}_{i,t} + \sum_{j\,:\,(i,j)\in\Theta} q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t} - \sum_{j\,:\,(j,i)\in\Theta} q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t}, \tag{6}$$

$$\begin{aligned} \tau_{i,t} = \tau^{\mathrm{ext}}_{i,t} &+ \sum_{j\,:\,(i,j)\in\Theta} (p^{\mathrm{J}}_{i,j,t} \times (q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t}) + q^{\mathrm{J}}_{i,j,t}\, \tau_{i,j,t}) \\ &- \sum_{j\,:\,(j,i)\in\Theta} (p^{\mathrm{J}}_{i,j,t} \times (q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t}) + q^{\mathrm{J}}_{i,j,t}\, \tau_{i,j,t}) \end{aligned} \tag{7}$$

where $p^{\mathrm{J}}_{i,j,t} = q_{i,t}\, p^{\mathrm{J}}_{i,j}$ and $q^{\mathrm{J}}_{i,j,t} = q_{i,t}\, q^{\mathrm{J}}_{i,j}$. The symbols f^{ext}_i and τ^{ext}_i denote the external force and moment; the sum of forces acting on the rigid body except constraint forces, which usually includes the gravity force.

In physics simulation, which is a typical application of multi-body systems, we calculate the time-evolution of rigid bodies in a single time step; that is, given states (postures and velocities) of rigid bodies at time t, we calculate constraint forces with which the state of rigid bodies at time $t + 1$, determined by (1)-(4) and (6)(7), satisfy (5). On the other hand, in the planning method described in the next section, the goal is to compute a sequence of postures, velocities and constraint forces in a prediction horizon with multiple time steps satisfying (1)-(7) with some additional constraints encapsulating joint limits and desired values.

III. DECENTRALIZED PLANNING

A. Overview of the proposed method

In this section, the proposed decentralized planning framework will be explained in detail. At first, we discuss a general constraint resolution problem. The core of solving this problem is the computation of Lagrange multipliers. For this purpose, we employ the projected Gauss-Seidel method. Next, a planning method for robotic systems will be derived from this general framework. From a conceptual point of view, the proposed method is decentralized in the sense that there is no single, high-level decision making element in the planner. Instead, at first a task objective will be imposed by means of a constraint on a task-relevant variable. Then this constraint information will in some sense be propagated to other variables by the Gauss-Seidel iteration. After a certain number of iterations, a motion that achieves the objective making use of all the variables in the kinematic graph will be produced. In this sense, each variable in the graph can be seen as a single agent and the whole graph can be seen as a multi-agent system, in which the agents negotiate each other through Gauss-Seidel iterations.

B. General constrained optimization problems

In this subsection, we formulate a general constraint resolution problem. Let $x \in \mathbb{R}^n$ and let $c : \mathbb{R}^n \mapsto \mathbb{R}^m$ be a smooth function. Moreover, let $y = c(x)$. The variable y is constrained in the following manner:

$$\begin{aligned} y_i &= 0 && \text{if } 1 \le i \le n_{\mathrm{eq}}, \\ \underline{y}_i &\le y_i \le \overline{y}_i && \text{otherwise.} \end{aligned} \tag{8}$$

The first n_{eq} elements of y are subject to equality constraints, and other elements are subject to range constraints. This defines a constraint manifold in $\mathbb{R}^n$; $\mathcal{M} := \{x \,|\, y = c(x) \text{ satisfies (8)}\}$. Our purpose is to find a value of x that lies in $\mathcal{M}$. To this aim, we first choose an initial value of x;

x^0. Next, we generate a sequence of points that converges to the constraint manifold $\mathcal{M}$ by the following formula: $x^{\tau+1} = x^\tau + J(x^\tau)^{\mathrm{T}}\lambda^\tau$. Here, the subscript τ is the index of points in the sequence. Moreover, $J(x)$ is the Jacobian matrix of $c(x)$; $J(x) = \partial c/\partial x(x)$ and $\lambda^\tau \in \mathbb{R}^m$ is the Lagrange multiplier. The multiplier λ^τ should be chosen in such a way that x moves towards the constraint manifold at every iteration. Such a multiplier is obtained in the following way: Let x be a point in the sequence and let λ be a multiplier (subscripts are omitted). The change of x in one step is given by $\delta x = J(x)^{\mathrm{T}}\lambda$. Moreover, by linear approximation, the change of y is expressed as $\delta y \approx J(x)\delta x = J(x)J(x)^{\mathrm{T}}\lambda$. Here we impose the following constraints on δy and λ: for $1 \le i \le n_{\mathrm{eq}}$, $\delta y_i = -\mu y_i$, and for $n_{\mathrm{eq}} < i \le m$,

$$\begin{cases} \delta y_i \ge -\mu(y_i - \underline{y}_i),\ \lambda_i \ge 0,\ (\delta y_i + \mu(y_i - \underline{y}_i))\lambda_i = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } y_i < \underline{y_i}, \\ \delta y_i \le -\mu(y_i - \overline{y}_i),\ \lambda_i \le 0,\ (\delta y_i + \mu(y_i - \overline{y}_i))\lambda_i = 0 \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } y_i > \overline{y_i}, \\ \lambda_i = 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{cases} \tag{9}$$

Here $0 < \mu < 1$. Notice that with a λ satisfying these conditions, the amount of constraint violation will be reduced with the rate $(1-\mu)$. Thus the sequence will exponentially converge to the constraint manifold $\mathcal{M}$. The rate of convergence is determined by the constant μ. The problem of finding λ satisfying the conditions given in (9) is a class of so-called *linear complementarity problems* (LCP in short). Today, the projected Gauss-Seidel method is known as a powerful iterative method for solving large-scale LCP. The reader is referred to [7] for theoretical background of LCP and introduction of projected Gauss-Seidel method. In the following, we present a specialized projected Gauss-Seidel method for constraint resolution problems.

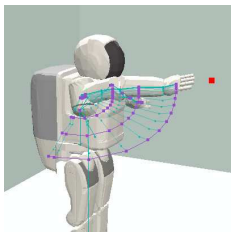
Algorithm **compute_multiplier**
Inputs
x constrained variable
λ^0 initial value of Lagrange multiplier
Outputs
δx change of constrained variable
λ Lagrange multiplier
Initialization
$y := c(x), \quad \lambda := \lambda^0, \ \delta x := J(x)^{\mathrm{T}}\lambda.$
for $i = 1$ **to** m
  if $i \le n_{\mathrm{eq}}$
    $r_i := -\mu y_i$
  else
    $r_i := -\mu(y_i - \underline{y}_i)$ **if** $y_i < \underline{y}_i$
    $r_i := -\mu(y_i - \overline{y}_i)$ **if** $y_i > \overline{y}_i$
    $r_i := 0$ **otherwise**
  end
  $A_{i,i} :=$ i-th diagonal element of $J(x)J(x)^{\mathrm{T}}$
end
$r := r - J(x)\delta x$
Iteration
loop
  if convergence condition is satisfied, terminate.
  for $i = 1$ **to** m
    $\hat{\lambda}_i := \lambda_i + r_i/A_{i,i}$
    if $i > n_{\mathrm{eq}}$
      $\hat{\lambda}_i := \max(0, \hat{\lambda}_i)$ **if** $y_i < \underline{y}_i$
      $\hat{\lambda}_i := \min(0, \hat{\lambda}_i)$ **if** $y_i > \overline{y}_i$
    end
    $\delta\lambda_i := \hat{\lambda}_i - \lambda_i$
    $\lambda_i := \hat{\lambda}_i$
    $\delta\delta x := J(x)_i^{\mathrm{row}\,\mathrm{T}}\delta\lambda_i$
    $\delta x := \delta x + \delta\delta x$
    $r := r - J(x)\delta\delta x$
  end
end

In this algorithm, at first, the Lagrange multiplier λ is initialized as λ^0, and then the constraint residual r is calculated. In each iteration, for each i-th element starting from $i = 1$ up to m, a new value of λ_i, $\hat{\lambda}_i$, is calculated using the residual r_i and the i-th diagonal element of $A = J(x)J(x)^{\mathrm{T}}$, which can be precomputed. If the i-th constraint is an range constraint, then $\hat{\lambda}_i$ is projected to 0 according to the state of constraint violation (whether the variable is hitting the upper limit or the lower limit) and the sign of $\hat{\lambda}_i$. Since A is symmetric positive definite, this projection ensures that the i-th linear complementarity condition is preserved. Finally, the change of variable δx and the residual r is updated accordingly. The symbol $J(x)_i^{\mathrm{row}}$ denotes the i-th row vector of $J(x)$.

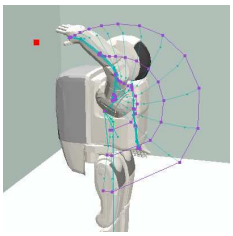
Using the above algorithm as a sub-routine, the next algorithm generates a sequence of points that converges to the constraint manifold $\mathcal{M}$.

Algorithm **execute_planning**
Inputs
x^0 initial value of constrained variable
Outputs
$\{x^\tau\}$ sequence of constrained variable
$\lambda := \mathbf{0}$
for $\tau = 0, 1, \ldots$
  if convergence condition is satisfied, terminate.
  $(\delta x^\tau, \lambda) :=$ **compute_multiplier**$(x^\tau, k\lambda)$
  $x^{\tau+1} := x^\tau + \delta x^\tau$
end

Remark: Each time **compute_multiplier** is called in **execute_planning**, the previous value of the Lagrange multiplier multiplied by a constant k is passed as an initial value. This is because in most case the value of Lagrange multiplier changes continuously, meaning that the value computed in the previous iteration serves as a good initial guess for the next iteration. The constant k is normally chosen from $[0, 1]$. However, setting k as 1 sometimes causes a long-period oscillatory behavior. A good compromise between speed and stability should be found by tuning.



(a) target in front



(b) target behind

Fig. 1. Target reaching task

C. Constraints with priorities

Robotic planning is in general multi-objective. In multi-objective planning, it is not always possible to accomplish all objectives at the same time. One practical solution is to introduce priorities; if not all objective are achievable, those with lower priorities are neglected or only partially achieved. We will show in this subsection that a slight modification to the conventional projected Gauss-Seidel method enables us to handle multiple constraint groups with different priorities. Let us consider L different constraint groups, in which groups with larger indices are assigned higher priorities. The basic idea is that when we calculate the Lagrange multiplier of the l-th constraint group, we take into account the effect of the multipliers of lower priorities (1st up to $(l-1)$-th constraint groups), while ignoring those of higher priority groups ($(l+1)$-th up to L-th). The actual modification needed to implement this prioritization is quite simple; we split the whole Gauss-Seidel loop into multiple loops according to the constraint groups, and execute these loops in the ascending order with respect to the constraint priority. The modified algorithm is shown below:

Algorithm **execute_planning_with_priority**
$\lambda_l := \mathbf{0}$ **for** $l \in [1, L]$
for $\tau = 0, 1, \ldots$
  if convergence condition is satisfied, terminate.
  $\hat{x}^\tau := x^\tau$
  for $l = 1$ **to** L
    $(\delta x_l^\tau, \lambda_l) :=$ **compute_multiplier_group**$(l, \hat{x}^\tau, k\lambda_l)$
    $\hat{x}^\tau := \hat{x}^\tau + \delta x_l^\tau$
  end
  $x^{\tau+1} := \hat{x}^\tau$
end

The algorithm **compute_multiplier_group** is almost the same as **compute_multiplier** except that **compute_multiplier_group** only treats a constraint group with an index specified by the input l.

D. Constraint resolution for robotic planning

In this subsection, we will specialize the general constraint resolution framework described in the previous subsections for robotic planning problems. All variables listed in Table I in Section II are directly used as planning variables. First, we will formulate the kinematic and physical laws described in Section II as constraints. The constraint variable and its amount of change for the position update law (1) are written as

$$\begin{aligned} y^p_{i,t} &= p_{i,t+1} - p_{i,t} - h\, v_{i,t}, \\ \delta y^p_{i,t} &= \delta p_{i,t+1} - \delta p_{i,t} - h\, \delta v_{i,t}. \end{aligned} \tag{10}$$

For the orientation update law (2), we cannot simply define the change of orientation in the same domain as orientation itself, since orientation is defined in the domain of unit quaternions. Instead, we express the change of orientation by means of a vector in $\mathbb{R}^3$. Let $\Omega_{i,t} \in \mathbb{R}^3$ be a rotation vector expressing the change of $q_{i,t}$. The orientation after rotation is given by $q(\Omega_{i,t})q_{i,t}$. Using this formulation, a constraint expressing the orientation update law is derived as

$$\begin{aligned} y^q_{i,t} &= q(h\,\omega_{i,t})\, q_{i,t}\, \Omega(q_{i,t}^{-1}\, q(h\,\omega_{i,t})^{-1}\, q_{i,t+1}), \\ \delta y^q_{i,t} &= \Omega_{i,t+1} - q(h\,\omega_{i,t})\, \Omega_{i,t} - h\, \delta\omega_{i,t}. \end{aligned} \tag{11}$$

The proof is omitted. See Appendix for the definition of the function $\Omega(\cdot)$. For the velocity update law (3), we have

$$\begin{aligned} y^v_{i,t} &= m_i\, v_{i,t+1} - m_i\, v_{i,t} - h\, f_{i,t}, \\ \delta y^v_{i,t} &= m_i\, \delta v_{i,t+1} - m_i\, \delta v_{i,t} - h\, \delta f_{i,t}. \end{aligned} \tag{12}$$

For the angular velocity update law (4), although the inertia matrix $I_{i,t}$ is a function of the orientation $q_{i,t}$, it is difficult to take this dependency into account. Here, we simply ignore this dependency and regard (4) as a constraint on angular velocities and moments:

$$\begin{aligned} y^\omega_{i,t} &= I_{i,t+1}\, \omega_{i,t+1} - I_{i,t}\, \omega_{i,t} - h\, \tau_{i,t}, \\ \delta y^\omega_{i,t} &= I_{i,t+1}\, \delta\omega_{i,t+1} - I_{i,t}\, \delta\omega_{i,t} - h\, \delta\tau_{i,t}. \end{aligned} \tag{13}$$

For joint constraints (5), we have

$$\begin{aligned} y^p_{i,j,t} &= p_{i,t} + p^{\mathrm{J}}_{i,j,t} - p_{j,t} - p^{\mathrm{J}}_{j,i,t}, \\ \delta y^p_{i,j,t} &= \delta p_{i,t} + \Omega_{i,t} \times p^{\mathrm{J}}_{i,j,t} - \delta p_{j,t} - \Omega_{j,t} \times p^{\mathrm{J}}_{j,i,t}, \end{aligned} \tag{14}$$

$$\begin{aligned} y^q_{i,j,t} &= (q^{\mathrm{J}}_{i,j,t})^{-1}\, (q^{\mathrm{J}}_{j,i,t})\, \Omega({q^{\mathrm{J}}_{j,i,t}}^{-1}\, q^{\mathrm{J}}_{i,j,t})\, q(e^{\mathrm{z}}\, \theta_{i,j,t}), \\ \delta y^q_{i,j,t} &= e^{\mathrm{z}}\, \delta\theta_{i,j,t} + {q^{\mathrm{J}}_{i,j,t}}^{-1}\, \Omega_{i,t} - {q^{\mathrm{J}}_{i,j,t}}^{-1}\, \Omega_{j,t}. \end{aligned} \tag{15}$$

These are derived based on a discussion similar to the case of (11). Finally, for force and moment constraints, we obtain

$$\begin{aligned} y^f_{i,t} &= f_{i,t} - f^{\mathrm{ext}}_{i,t} \\ &\quad - \sum_{j\,:\,(i,j)\in\Theta} q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t} + \sum_{j\,:\,(j,i)\in\Theta} q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t}, \\ \delta y^f_{i,t} &= \delta f_{i,t} - \sum_{j\,:\,(i,j)\in\Theta} q^{\mathrm{J}}_{i,j,t}\, \delta f_{i,j,t} + \sum_{j\,:\,(j,i)\in\Theta} q^{\mathrm{J}}_{i,j,t}\, \delta f_{i,j,t}, \end{aligned} \tag{16}$$

$$\begin{aligned} y^\tau_{i,t} &= \tau_{i,t} - \tau^{\mathrm{ext}}_{i,t} - \sum_{j\,:\,(i,j)\in\Theta} (p^{\mathrm{J}}_{i,j,t} \times q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t} + q^{\mathrm{J}}_{i,j,t}\, \tau_{i,j,t}) \\ &\quad + \sum_{j\,:\,(j,i)\in\Theta} (p^{\mathrm{J}}_{i,j,t} \times q^{\mathrm{J}}_{i,j,t}\, f_{i,j,t} + q^{\mathrm{J}}_{i,j,t}\, \tau_{i,j,t}), \\ \delta y^\tau_{i,t} &= \delta\tau_{i,t} - \sum_{j\,:\,(i,j)\in\Theta} (p^{\mathrm{J}}_{i,j,t} \times q^{\mathrm{J}}_{i,j,t}\, \delta f_{i,j,t} + q^{\mathrm{J}}_{i,j,t}\, \delta\tau_{i,j,t}) \\ &\quad + \sum_{j\,:\,(j,i)\in\Theta} (p^{\mathrm{J}}_{i,j,t} \times q^{\mathrm{J}}_{i,j,t}\, \delta f_{i,j,t} + q^{\mathrm{J}}_{i,j,t}\, \delta\tau_{i,j,t}). \end{aligned} \tag{17}$$

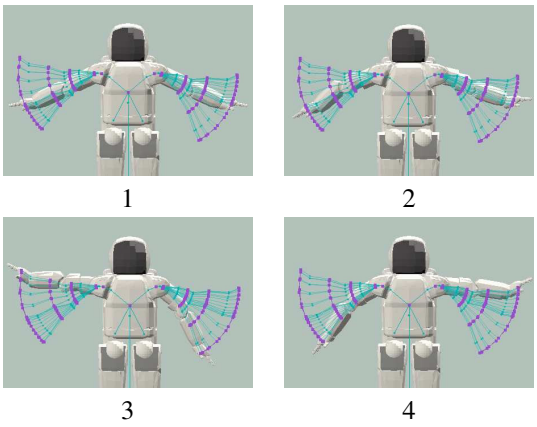


Fig. 2. Moment generation along x-axis

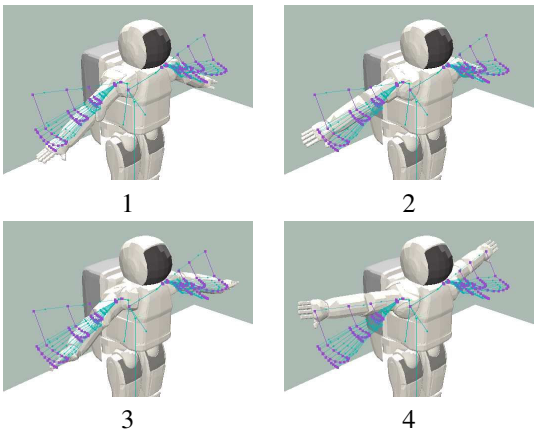
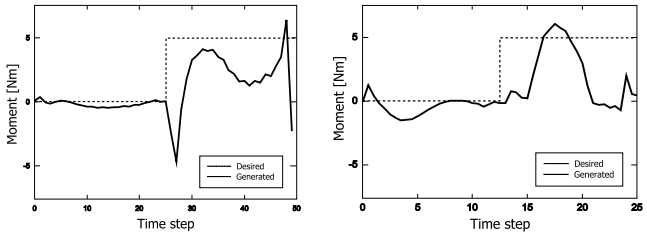


Fig. 3. Moment generation along z-axis

Here, we again ignore the dependency of orientations on these constraints. Using the above relations, we can construct the function $c(x)$ and its Jacobian $J(x)$. Moreover, the Jacobian $J(x)$ is highly sparse; by an appropriate implementation, the computational complexity of a single iteration in **compute_multiplier**[_**group**] becomes proportional to the number of constraints. Furthermore, although not described here, various limits (joint movable ranges, velocity limits, torque limits) can be expressed as range constraints. Moreover, desired values of variables can be encapsulated as constraints as well. However, it is clear that if we treat desired value constraints equally with other constraints, it will lead to over-constrained situation and consequently no feasible solution will be produced (for an example, specifying an unreachable hand position may violate joint constraints). To cope with this problem, we assign lower priorities to the desired value constraints than those of other constraints. By doing this, the planner will produce a solution that tries to accomplish the desired values as much as possible, while fulfilling the kinematic / physical constraints precisely.



(a) x-moment (b) z-moment

Fig. 4. Desired and generated moments

IV. Experiments

A. Implementation of the method for humanoid upper body

We have implemented the proposed method to the upper body control of the humanoid robot ASIMO. The upper body of the robot is fixed to the global coordinate frame. Here, x-axis points to the front, y-axis upward and z-axis to the right with respect to the robot. Each hand is regarded as a single rigid body. Moreover, the head movement is not considered. Therefore, the whole upper body is modeled as a multi-body system composed of 11 rigid bodies and 10 hinge joints. In the following experiments, we set the step size h as 0.1[s] and the prediction step length N as 25. The planning method is implemented in C++ programming language and executed in a computing environment composed of 2.4GHz CPU and 2GB memory. The simulation environment is constructed using Springhead physics simulation library [8].

B. Target reaching task

We first consider a simple target reaching task, in which the robot moves its hand position towards a specified target position. To specify this objective, we impose a desired-position constraint on the right hand. Fig. 1(a)(b) show simulation results with two different target positions, one in front of the robot and one behind. Planned trajectories are visualized with small dots depicting the centers of mass of rigid bodies connected by solid lines. A relatively large dot depicts the target position. The robot in each figure is in the posture after executing the plan. Thanks to the prioritization mechanism, even when a target is specified in an unreachable position, the hand is moved to the nearest possible position to the target while not violating the kinematic constraints.

C. Body moment generation task

Next, we consider a body moment generation task, in which the robot should move its whole body to generate a specified amount of counter moment on its body in a specified direction. To implement this objective, we impose a desired-moment constraint on the robot's body during the latter half ($t \in [N/2, N)$) of the prediction horizon. Let us emphasize that this task is much more difficult than the target reaching task, because the rigid body to which a desired-moment constraint is imposed on (the body) and those whose motions actually generate a desired moment are different. This means that the effect of the desired-moment constraint should be transmitted from the center to the tip of the

kinematic tree, in order to generate an appropriate motion. Fig. 2 and Fig. 3 show the simulation results. We can observe that each motion is basically composed of three distinct phases; i) the robot moves its arms to a good starting posture (preparation), ii) the robot accelerates the arms in a certain direction (forward acceleration), and then iii) accelerates them in the opposite direction (backward acceleration). Note that the planner is not given any a priori knowledge about how it should swing the arms in order to generate a desired counter moment. Fig. 4 shows the comparison between the desired moment and the actually generated moment. Desired moments are drawn in dashed lines while genereted moments are drawn in solid lines.

Next we show the results obtained using real humanoid robot ASIMO (Fig. 5). In this case, the robot consecutively plans and executes moment generation motions for three different directions (x, z, and y axes). A key difference between the previous example is that the whole motion is required to be continuous; the robot should use the terminal state of the previous motion as the initial state of the next motion. To efficiently generate body-moments under this condition, the planner effectively makes use of the former half of the prediction horizon, on which no desired moment constraint is imposed, to generate a preparation motion for the next moment generation.

V. Conclusion

In the current implementation, the convergence speed of the Gauss-Seidel iteration is not fast enough, making the method unable to be executed in real-time. In the future, further speed-up of the method, analytical study of computational complexity as well as comparison with other existing planning methods are required. One of the interesting extensions would be to incorporate contact constraints, which enables more complex and various motions such as walking, object manipulation and so forth. We believe such complex motions involving discontinuity and non-convexity can still be treated in a decentralized framework. However, the current gradient descent formalism is obviously not enough; we will need to incorporate more sophisticated mechanism for communication among planning agents in order to generate globally optimal motions.

VI. Acknowledgments

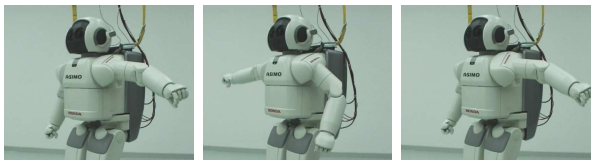
This research has been funded by Japan Society for the Promotion of Science.

Appendix

Quaternion is a convenient mathematical tool for representing rotation in 3D space. A unit quaternion q is a vector with four elements: $q = [w \;\; v^{\mathrm{T}}]^{\mathrm{T}} = [w \;\; [x \;\; y \;\; z]]^{\mathrm{T}}, \;\; \|q\| = 1$. A quaternion representing a rotation along a unit vector η with a rotation angle θ is given as

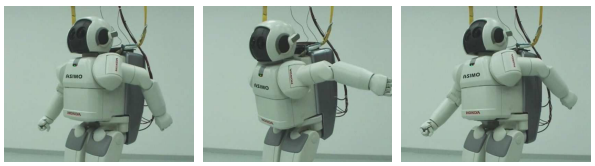
$$q(\eta, \theta) = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\eta \end{bmatrix}.$$

Further, we define $q(\omega) = q(\omega/\|\omega\|, \|\omega\|)$. Conversely, the function $\Omega(q)$ returns a vector that represents a rotation



(a) x-axis moment

(b) z-axis moment



(c) y-axis moment

Fig. 5. Motion replay on the real humanoid

equivalent to a given quaternion q; for $q = q(\eta, \theta)$, $\Omega(q) = \theta\eta$. A product of two quaternions is defined as

$$q_1 \cdot q_2 = \begin{bmatrix} s_1 s_2 - v_1^{\mathrm{T}} v_2 \\ s_1 v_2 + s_2 v_1 + v_1 \times v_2 \end{bmatrix},$$

and it represents a rotation equivalent to the composition of two rotations. The inverse of a quaternion $q = [w \;\; v^{\mathrm{T}}]^{\mathrm{T}}$ is given by $q^{-1} = [w \;\; -v^{\mathrm{T}}]^{\mathrm{T}}$. A rotation transformation of a vector $v \in \mathbb{R}^3$ is defined as

$$qv := \hat{v}, \;\; \begin{bmatrix} * \\ \hat{v} \end{bmatrix} = q \cdot \begin{bmatrix} * \\ v \end{bmatrix} q^{-1}$$

where $*$ indicates the value does not matter.

References

[1] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba and H. Inoue; Dynamically-stable Motion Planning for Humanoid Robots, Autonomous Robots, vol.12, no.1, pp.105-118, 2002.

[2] J. J. Kuffner and S.M. LaValle, RRT-connect: An efficient approach to single-query path planning, Proceedings of IEEE International Conference on Robotics and Automation, 2000.

[3] K. Yamane, J. Kuffner and J.K. Hodgins; Synthesizing Animations of Human Manipulation Tasks, ACM International Conference on Computer Graphics and Interactive Techniques (Siggraph 2004), pp.532-539, 2004.

[4] K. Harada, M. Morisawa, K. Miura, S. Nakaoka, K. Fujiwara, K. Kaneko and S. Kajita; Kinodynamic Gait Planning for Full-Body Humanoid Robots, IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France, Sept. 22-26, 2008.

[5] E. Yoshida, I. Belousov, C. Esteves and J.P. Laumond; Humanoid Motion Planning for Dynamic Tasks, IEEE-RAS International Conference on Humanoid Robots, Tsukuba, Japan, 2005.

[6] L. Sentis and O. Khatib; A Whole-Body Control Framework for Humanoids Operating in Human Environments, Proceedings of the IEEE International Conference in Robotics and Automation, Orlando, USA, May, 2006.

[7] R. W. Cottle, J. Pang, R. E. Stone; The Linear Complementarity Problem, Academic Press, 1992.

[8] Springhead physics simulation engine, `http://springhead.info`