[15] A. W. Mahoney and J. J. Abbott, “Generating rotating magnetic fields with a single permanent magnet for propulsion of untethered magnetic devices in a lumen,” *IEEE Trans. Robot.*, vol. 30, no. 2, pp. 411–420, Apr. 2014.
[16] A. W. Mahoney and J. J. Abbott, “5-DOF manipulation of a magnetic capsule in fluid using a single permanent magnet: Proof-of-concept for stomach endoscopy,” in *Proc. Hamlyn Symp. Med. Robot.*, 2013, pp. 114–115.
[17] A. J. Petruska and J. J. Abbott, “Omnimagnet: An omnidirectional electromagnet for controlled dipole-field generation,” *IEEE Trans. Magn.*, vol. 50, no. 7, p. 8400810, Jul. 2014.
[18] A. J. Petruska and J. J. Abbott, “Optimal permanent-magnet geometries for dipole field approximation,” *IEEE Trans. Magn.*, vol. 49, no. 2, pp. 811–819, Feb. 2013.
[19] D. J. Griffiths, *Introduction to Electrodynamics*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1999.
[20] M. Gauthier and E. Piat, “Control of a particular micro-macro positioning system applied to cell micromanipulation,” *IEEE Trans. Autom. Sci. Eng.*, vol. 3, no. 3, pp. 264–271, Jul. 2006.
[21] C. Pawashe, S. Floyd, and M. Sitti, “Modeling and experimental characterization of an untethered magnetic micro-robot,” *Int. J. Robot. Res.*, vol. 28, no. 8, pp. 1077–1094, 2009.
[22] L. Zhang, J. Abbott, L. Dong, B. Kratochvil, D. Bell, and B. Nelson, “Artificial bacterial flagella: Fabrication and magnetic control,” *Appl. Phys. Lett.*, vol. 94, pp. 064107-1–064107-3, 2009.
[23] A. W. Mahoney, S. E. Wright, and J. J. Abbott, “Managing the attractive magnetic force between an untethered magnetically actuated tool and a rotating permanent magnet,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5346–5351.

Constraint-Based Prioritized Trajectory Planning for Multibody Systems

Yuichi Tazaki and Tatsuya Suzuki

***Abstract*—This paper presents a trajectory-planning method for multibody systems. Trajectory planning of a multibody system is formulated as a constraint-solving problem on a set of variables expressing the motion of the multibody system over a finite-time interval. Constraints express the dynamics of rigid bodies, kinematic conditions of joints, various range limitations, as well as achievement of tasks, and they can be assigned different priority levels. The prioritized constraint-solving problem is then treated under the framework of lexicographical goal programming, where the local optimality of the problem is characterized in terms of Pareto efficiency condition. Based on this observation, an algorithm that iteratively updates the variables toward a locally optimal solution is derived. The proposed method is evaluated in simulation examples.**

***Index Terms*—Constraint solving, multibody systems, priority, trajectory planning.**

Manuscript received May 26, 2013; revised March 30, 2014; accepted April 19, 2014. Date of publication May 23, 2014; date of current version September 30, 2014. This paper was recommended for publication by Associate Editor N. Mansard and Editor A. Kheddar upon evaluation of the reviewers’ comments.

The authors are with the Graduate School of Engineering, Nagoya University, Nagoya 464-8601, Japan (e-mail: tazaki@nuem.nagoya-u.ac.jp; t_suzuki@nuem.nagoya-u.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org

Digital Object Identifier 10.1109/TRO.2014.2320794

I. Introduction

Trajectory planning of robotic systems with many degrees of freedom poses some technical challenges. First, one must find a trajectory that achieves given tasks, while fulfilling kinematic and physical constraints in a high-dimensional state space. Trajectory planning is often treated as an optimization problem (see [1]–[7]). Here, a set of physical variables that express a robotic trajectory is optimized to fulfill a series of constraints that originates from law of physics and various kinematic relationships between objects. A wide range of optimization techniques have been investigated, including the Newton method [3], the shooting method [2], [4], the covariance matrix adaptation [5], and the SQP [7].

The second difficulty is the multiobjective nature that most real-world trajectory planning problems possess. The task-space control framework is capable of handling different priorities of tasks (see [8]–[13]). In this framework, a unique local coordinate frame called a task space is defined for each task and tasks with lower priorities are treated in the null-space of the task spaces of those with higher priorities. This framework is extended to inequality tasks in [14]. Although it is quite useful for synthesizing a set of feedback controllers with multiple priority levels, it is not directly applicable to trajectory planning. When one considers trajectory planning, task priorities should be considered in the space of state trajectories rather than in the space of states. However, trajectory planning is a nonlinear problem in much higher dimension space. This makes some computational techniques that have been used in task-space control not directly applicable. These include the computation of the pseudoinverse matrix of constraint Jacobian (see [8]), the computation of the basis of constraint null-space using singular value decomposition (see [12]), and sequence of quadratic programs (see [14]).

Based on the above background, this paper proposes a trajectory planning method for robotic systems that are represented as multibody systems. The multibody representation enables the expression robots with various morphologies and workspaces with different settings in a uniform manner. A trajectory-planning problem of a multibody system is formulated as a constraint satisfaction problem with multiple priority levels, in which the kinematics and the dynamics of the multibody system, as well as the achievement of tasks are expressed as a set of constraint conditions. It is shown that the problem can be viewed as a class of goal programming problem (see [18]) and its local optimality condition is given as a special form of Pareto-efficiency condition. Based on this observation, an algorithm that iteratively updates a set of decision variables toward a prioritized Pareto-efficient point is proposed. The proposed algorithm requires no expensive computation other than the solution of quadratic programs, and therefore, it is suitable for trajectory planning of robotic systems with many degrees of freedom. The basic concept of the proposed method has been presented in the authors’ previous publications (see [16] and [17]). In this paper, the theoretical foundation of the proposed method is strengthened by revealing the connection between lexicographical optimality and a special type of Pareto-efficiency (see Section II). The formulation of multibody trajectory planning is presented considering both sparse and dense parameterizations (see Section III). Moreover, the computational performance of the proposed method is compared with conventional multiobjective optimization problems that are computed by a generic solver (see Section IV). Concluding remarks are made in Section V.

Notation: A sequence of integers from $i_1, i_1 + 1, \ldots, i_2$ is written as $[i_1 : i_2]$. Moreover, the vertical concatenation of vectors, $[v_1^\mathsf{T}\ v_2^\mathsf{T}]^\mathsf{T}$ is written as $[v_1 ; v_2]$. For a vector c and an index set $\mathcal{I}$, $c_\mathcal{I}$ denotes the subvector of c with the components indexed by $\mathcal{I}$.

II. Constraint Error Minimization Problem Based on Goal Programming

A. Constraint-Error Minimization and Goal Programming

The trajectory-planning problem takes the form of a constraint satisfaction problem for a set of physical variables that constitute a multibody system. Constraints that express the kinematics and the dynamics of the system are equality constraints, while constraints that express collision avoidance and movable ranges are in inequalities. Let us define the decision variable vector $z \in \mathbb{R}^{n_z}$ as the concatenation of all variables that are involved in a trajectory-planning problem. Moreover, let all constraints aggregate into the following form:

$$c^{\mathrm{l}} \le c(z) \le c^{\mathrm{u}}. \tag{1}$$

Here, $c : \mathbb{R}^{n_z} \mapsto \mathbb{R}^{n_c}$, and n_c denotes the number of constraints. The *constraint function* $c(z)$ must be C^1 continuous, since the algorithm presented later makes use of the Jacobian matrix of $c(z)$. Moreover, c^{l} and c^{u} denote the lower and upper bounds on $c(z)$. Setting $c_i^{\mathrm{l}} = c_i^{\mathrm{u}}$ imposes an equality constraint on the ith component of $c(z)$. The ith component is *active*, if either $c_i(z) \le c_i^{\mathrm{l}}$ or $c_i(z) \ge c_i^{\mathrm{u}}$ holds.

There may be an overconstrained situation, in which not all constraints are simultaneously satisfiable. In such a case, one should introduce priorities into constraints; constraints with higher priorities are strictly fulfilled, while the errors of lower priority constraints are minimized using the remaining degrees of freedom. High-priority constraints typically include the kinematics and dynamics constraints, which are critical to the realizability of the trajectories, and lower priority constraints are task constraints. Needless to say, tasks may have different priority levels as well.

Goal programming (GP) is one of the frameworks in the multiobjective programming literature that can handle multiple objectives with different priorities (see [18]). Among several formulations that have been investigated in the context of GP, the weighted goal programming (WGP) and the lexicographical goal programming (LGP) are particularly well known. WGP, as its name suggests, minimizes the weighted sum of the objective functions

$$\min_z \sum_{l=0}^{L} w_l\, \phi_l(z). \tag{2}$$

Here, $\phi_l(z)$ ($l \in [0 : L]$) are the objective functions that return nonnegative scalar values, and $w_l > 0$ are the weighting coefficients. Objective functions with larger weights are selectively reduced; therefore, priorities can be indirectly reflected in the combination of weights. As the number of objective functions increases, however, the weight adjustment to realize an intended balance in the objective values becomes increasingly difficult.

On the other hand, LGP is formulated as follows:

$$\begin{aligned} \phi_0^* &= \min_z\ \phi_0(z), \\ \phi_1^* &= \min_z\ \phi_1(z) \text{ subject to } \phi_0(z) = \phi_0^*, \\ &\vdots \\ \phi_L^* &= \min_z\ \phi_L(z) \text{ sub.to } \phi_l(z) = \phi_l^*\ \forall l \in [0 : L-1]. \end{aligned} \tag{3}$$

In LGP, $l \in [0 : L]$ represents the priority of objective functions. The objective function ϕ_0 has the highest priority, while ϕ_L has the lowest (be aware that smaller l indicates higher priority). First, $\phi_0(z)$ is minimized and its minimum value is denoted by ϕ_0^*. Next, $\phi_1(z)$ is minimized subject to the constraint that the minimum value of $\phi_0(z)$ is maintained. More generally, the lth objective function is minimized subject to the constraint that all the previously minimized objective values are maintained. LGP has an important property that higher priority objectives are unaffected by the lower ones; that is, the minimum objective value is determined regardless of whether the lower priority objectives exist or not. This is a great advantage over WGP, which requires the try-and-error procedure for weight tuning.

Consider solving the above LGP-type problem by an iterative procedure. Let $\mathcal{D}^{\mathrm{lex}}(z)$ denote the set of admissible update directions at the point z, defined as follows:

$$\begin{aligned} &\mathcal{D}^{\mathrm{lex}}\langle \phi_0, \phi_1, \ldots, \phi_L \rangle(z) = \{\, \delta z \mid \\ &(\delta\phi_0 < 0) \text{ or} \\ &(\delta\phi_0 = 0 \text{ and } \delta\phi_1 < 0) \text{ or } \ldots \text{ or} \\ &(\delta\phi_l = 0\ \forall l \in [0 : L-1] \text{ and } \delta\phi_L < 0)\}. \end{aligned} \tag{4}$$

Here, $\delta\phi_l = (\partial\phi_l/\partial z)(z)\, \delta z$. An admissible update direction of LGP is a direction that reduces the objective value of a certain priority level l without increasing the objective values of priority levels higher than l.

B. Priority Handling Based on Pareto Efficiency

Based on the discussion in the previous section, the idea of prioritization is incorporated into the constraint satisfaction problem. For preparation, the constraint conditions are split into $L+1$ groups with the priority levels 0 to L. Let $\mathcal{I}_l$ be the index set of constraint components with the priority level l. For later use, we also define $\mathcal{I}_{[0:l]} = \mathcal{I}_0 \cup \mathcal{I}_1 \cup \ldots \cup \mathcal{I}_l$. The subvector of $c(z)$ with all components with the priority l is written as $c_{\mathcal{I}_l}(z)$, and the subvector with all components whose priority is higher than or equal to l is written as $c_{\mathcal{I}_{[0:l]}}(z)$. Moreover, let us define a *constraint error function* $e(z)$ whose components are given by

$$e_i(z) = \begin{cases} c_i(z) - c_i^{\mathrm{u}}, & c_i(z) > c_i^{\mathrm{u}}, \\ c_i^{\mathrm{l}} - c_i(z), & c_i(z) < c_i^{\mathrm{l}}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Furthermore, let $J(z)$ be the Jacobian matrix of $c(z)$ with respect to z; $J(z) = \frac{\partial}{\partial z} c(z)$.

A natural way to formulate a problem in the form of LGP would be to define a scalar function for each l, say $\phi_l(z)$, such that $c_{\mathcal{I}_l}^{\mathrm{l}} \le c_{\mathcal{I}_l}(z) \le c_{\mathcal{I}_l}^{\mathrm{u}} \Leftrightarrow \phi_l(z) = 0$. A typical example of such a function is $\phi_l(z) = \|e_{\mathcal{I}_l}(z)\|^2$, in which case the sum of squares of constraint errors is minimized for each priority level. In this manner, however, one may observe some irregularity in the decrease of the components of $c(z)$, unless each component is properly normalized to have approximately the same range of values. Such preconditioning becomes difficult when there are hundreds of constraints. To avoid this situation, this paper introduces a modified formulation of LGP which is based on Pareto efficiency. Now, the Pareto efficiency in this context is defined as follows:

Definition 1: For a vector-valued function $e : \mathbb{R}^n \mapsto \mathbb{R}^m$ and a point $z \in \mathbb{R}^n$, a direction δz satisfying the following condition is called a *Pareto direction*:

$$(\exists i \in [1 : m] \text{ s.t. } \delta e_i < 0) \text{ and } (\delta e_i \le 0\ \forall i \in [1 : m]) \tag{6}$$

where $\delta e_i = (\partial e_i/\partial z)\delta z$. The set of Pareto directions at z is denoted by $\mathcal{D}^{\mathrm{par}}\langle e \rangle(z)$. Moreover, when $\mathcal{D}^{\mathrm{par}}\langle e \rangle(z)$ is empty, z is said to be Pareto efficient with respect to $e(z)$.

Intuitively, a Pareto direction is a direction in which some components of $e(z)$ can be reduced without increasing the other components. If such a direction does not exist, then z is Pareto efficient.

Now, let us focus again on (4). The following lemma states that $\mathcal{D}^{\mathrm{lex}}(z)$ is given by the union of multiple Pareto direction sets.

Lemma 1: The following relation holds:

$$\begin{aligned}\mathcal{D}^{\mathrm{lex}}\langle\phi_0,\phi_1,\ldots,\phi_L\rangle(z) &\equiv \\ \mathcal{D}^{\mathrm{par}}\langle\phi_0\rangle(z) &\cup \mathcal{D}^{\mathrm{par}}\langle[\phi_0;\phi_1]\rangle(z)\cup\ldots \\ &\cup \mathcal{D}^{\mathrm{par}}\langle[\phi_0;\phi_1;\ldots;\phi_L]\rangle(z).\end{aligned} \quad (7)$$

Here, $[\phi_0;\phi_1;\ldots;\phi_l](z)$ is a function that returns $[\phi_0(z); \phi_1(z);\ldots;\phi_l(z)]$.

Proof: See Appendix A.

Using the above relation, one can write the local optimality condition of LGP in terms of Pareto efficiency; that is, a point satisfying $\mathcal{D}^{\mathrm{lex}}(z)=\emptyset$ is Pareto efficient with respect to ϕ_0, $[\phi_0;\phi_1],\ldots,$ and $[\phi_0;\phi_1;\ldots;\phi_L]$. Based on this observation, we now introduce a new set of directions by replacing the scalar function ϕ_l in the right-hand side of (7) with the vector-valued function $e_{\mathcal{I}_l}$.

Definition 2: A prioritized Pareto direction is a set given as follows:

$$\begin{aligned}&\mathcal{D}^{\mathrm{lex-par}}\langle e\rangle(z) \\ &= \mathcal{D}^{\mathrm{par}}\langle e_{\mathcal{I}_0}\rangle(z)\cup\mathcal{D}^{\mathrm{par}}\langle e_{\mathcal{I}_{[0:1]}}\rangle(z)\cup\ldots\cup\mathcal{D}^{\mathrm{par}}\langle e\rangle(z).\end{aligned} \quad (8)$$

Moreover, a point z satisfying $\mathcal{D}^{\mathrm{lex-par}}\langle e\rangle(z)=\emptyset$ is called a prioritized Pareto-efficient point.

For convenience, let us call the problem of finding a prioritized Pareto efficient point P-LGP (P is for Pareto). It should be noted that P-LGP is not equivalent to the conventional LGP with $\phi_l = \|e_{\mathcal{I}_l}\|^2$, since (8) is derived by replacing ϕ_l in (7) with $e_{\mathcal{I}_l}$ symbolically. Nevertheless, P-LGP has a useful property that its solution set is invariant under constraint scaling. That is, if $\mathcal{D}^{\mathrm{lex-par}}\langle e\rangle(z)=\emptyset$ for some z, then $\mathcal{D}^{\mathrm{lex-par}}\langle\Gamma e\rangle(z)=\emptyset$, where Γ is an arbitrary positive diagonal matrix. This property is indeed useful since one does not need to be concerned about the scaling of constraint function $c(z)$ affecting the solution set.

Remark 1: Scaling of variables and constraints is still important, since it affects the convergence speed of the iterative algorithm presented in the next section. Moreover, the solution of P-LGP is generally not unique. Therefore, even though the whole solution set is scaling-invariant, a solution that is produced by the algorithm may be scaling-dependent.

C. Algorithm for Finding Prioritized Pareto Efficient Solution

This section presents an iterative method that finds a prioritized Pareto-efficient point. The algorithm repeatedly updates the variable z in a direction given by δz, a prioritized Pareto direction. Algorithm 1 shows the procedures of the algorithm. At each iteration, l is varied from L down to 0 and a Pareto direction with respect to $e_{\mathcal{I}_{[0:l]}}$ is computed for each l. The procedure QP(l,z) computes the following quadratic program to obtain a Pareto direction:

$$\begin{aligned}\text{min.}\quad & \|\Gamma^z\,\delta z\|^2+\|\Gamma^e(\delta e+\gamma e(z))\|^2 \\ \text{subject to}\quad & J_i(z)\,\delta z\le\delta e_i,\ \delta e_i\le 0\ \ \forall i\in\mathcal{I}^+_{[0:l]} \\ & -J_i(z)\,\delta z\le\delta e_i,\ \delta e_i\le 0\ \ \forall i\in\mathcal{I}^-_{[0:l]}.\end{aligned} \quad (9)$$

The decision variables of QP(l,z) are δz and δe, where δz denotes the direction of change of z and δe denotes that of constraint errors. Here, δe must be a variable because the definition of the Pareto direction requires that at least one component of δe must be strictly negative, but which component will be strictly negative is not known in advance. The matrices Γ^z and Γ^e are scaling matrices for δz and δe, respectively, given by positive diagonal matrices. The constant $\gamma>0$ is the desired error correction rate. The symbols $\mathcal{I}^+_{[0:l]}$ ($\mathcal{I}^-_{[0:l]}$) denote the index set of constraints that satisfies $c_i(z)\ge c_i^{\mathrm{u}}$ ($c_i(z)\le c_i^{\mathrm{l}}$) and has priority levels between 0 and l. Note that the components of δe that are not included in either $\mathcal{I}^+_{[0:l]}$ or $\mathcal{I}^-_{[0:l]}$ are actually not involved in the quadratic program; these are included in the decision variables solely for brevity of notation. The following lemma guarantees that QP(l,z) outputs a Pareto direction if one exists.

Algorithm 1 Prioritized Pareto Algorithm

```
 1: $z \leftarrow z_0$
 2: loop
 3:     $\delta z \leftarrow 0$
 4:     for $l = L$ to $0$ do
 5:         $(\delta z, \delta e) \leftarrow$ QP$(l, z)$
 6:         if $\delta e \neq 0$ then
 7:             $\alpha \leftarrow$ LineSearch$(l, z, \delta z)$
 8:             break
 9:         end if
10:     end for
11:     if $\alpha\|\delta z\| < \epsilon$ then
12:         break
13:     end if
14:     $z \leftarrow z + \alpha\,\delta z$
15: end loop
```

Lemma 2: If $\mathcal{D}^{\mathrm{par}}\langle e_{\mathcal{I}_{[0:l]}}\rangle(z)$ is not empty, then the optimal solution $(\delta z^*,\delta e^*)$ of QP(l,z) satisfies $\delta z^*\in\mathcal{D}^{\mathrm{par}}\langle e_{\mathcal{I}_{[0:l]}}\rangle(z)$. Otherwise, the optimal solution is $(\delta z^*,\delta e^*)=(0,0)$.

Proof: See Appendix B.

The quadratic program (9) can be computed by a generic QP solver. In this paper, it is first transformed into an equivalent linear complementarity problem (see [21]) and then computed by the projected Gauss–Seidel algorithm. In this manner, the sparsity of the problem can be fully exploited.

After a Pareto direction is computed, a line search is performed by the procedure LineSearch to determine the best step size α. Like conventional gradient-based optimization methods, the proposed method can benefit from line search for speeding up the convergence of iteration and for reducing oscillatory behavior at near-singular points. Suppose $\delta z\in\mathcal{D}^{\mathrm{par}}\langle e_{\mathcal{I}_{[0:l]}}\rangle(z)$ is obtained at some point of iteration. Then, the step size α is obtained by solving the following constrained scalar minimization problem:

$$\begin{aligned}\text{min.}\quad & \|e_{\mathcal{I}_{[0:l]}}(z+\alpha\,\delta z)\|^2 \\ \text{subject to}\quad & e_{\mathcal{I}_{[0:l]}}(z+\alpha\delta z)\le e_{\mathcal{I}_{[0:l]}}(z).\end{aligned} \quad (10)$$

This problem can be computed by a method similar to the well-known golden section search. The obtained α is then clipped to the range $\alpha^{\min}\le\alpha\le\alpha^{\max}$. The minimum step size $\alpha^{\min}$ must take some positive value. Otherwise, α will be 0 when some nonlinear equality constraints are strictly satisfied at z, and as a result, iteration will terminate even if z is not a stationary point.

Remark 2: The scaling matrices Γ^z and Γ^e strongly influence the convergence speed of the projected Gauss–Seidel algorithm. One simple yet effective way is to set them in such a way that each variable (constraint) component after scaling becomes physically undimensioned.

TABLE I
TRAJECTORIES OF MULTIBODY SYSTEM

i) Variables of ith rigid body		ii) Variables of jth joint	
$\boldsymbol{p}_{i,k} \in \mathbb{R}^3$	position of center of mass	$\boldsymbol{\theta}_{j,k} \in \mathbb{R}$	joint angle
$\boldsymbol{q}_{i,k} \in \mathcal{Q}$	orientation	$\boldsymbol{\nu}_{j,k} \in \mathbb{R}$	joint velocity
$\boldsymbol{v}_{i,k} \in \mathbb{R}^3$	velocity of center of mass	$\boldsymbol{\tau}_{j,k} \in \mathbb{R}$	joint torque
$\boldsymbol{\omega}_{i,k} \in \mathbb{R}^3$	angular velocity	$\boldsymbol{f}_{j,k} \in \mathbb{R}^3$	joint force
		$\boldsymbol{n}_{j,k} \in \mathbb{R}^3$	joint moment

First, the characteristic constants of the fundamental physical dimensions, position, mass, and time, are determined based on the approximate size and the mass of the robot and the workspace, and the time resolution h. Then, for a velocity variable, for example, the scaling coefficient is given by $(L/T)^{-1}$, where L and T are the characteristic constants of position and time, respectively.

III. Multibody Scene and Task Representation

A. Overview

The prioritized constraint satisfaction algorithm that is presented in the previous section is applied to the trajectory planning of multibody systems. Continuous-time trajectory planning of multibody system falls into the category of semi-infinite programming [19], [20]. Therefore, the first step is to determine an appropriate representation of continuous-time trajectories using a finite number of variables. To this sake, a discrete sequence of time instants over the planning horizon $[0, T]$ is introduced

$$t_0, t_1, \dots, t_N, \;\; t_k = kh, \, h = T/N. \tag{11}$$

A trajectory of a multibody system is then expressed in terms of the values of physical quantities at these time instants. Table I tabulates the list of variables that are related to rigid bodies and joints at a discrete time instant t_k. Here, $\mathcal{Q}$ denotes the set of unit quaternions, which are used to express rotations in the 3-D space. For $q \in \mathcal{Q}$ and $p \in \mathbb{R}^3$, the rotation of p by q is written as $q\,p$. For $q_1 \in \mathcal{Q}$ and $q_2 \in \mathcal{Q}$, $q_2\, q_1$ gives the composition of rotation. Moreover, for a vector $\Omega \in \mathbb{R}^3$, a quaternion that expresses a rotation with the rotation axis $\Omega/\|\Omega\|$ and the rotation angle $\|\Omega\|$ is denoted by $q(\Omega)$.

The second step is to choose an appropriate parameterization of the physical quantities of a multibody system. One can consider two different parameterizations: sparse parameterization and dense parameterization. In the sparse (redundant) parameterization, all physical quantities that are listed in Table I are treated as independent variables. In the dense (reduced) parameterization, on the other hand, the rigid body states (position, orientation, velocity, and angular velocity) are functions of the joint angles and velocities, and the kinematics and the dynamics of the multibody system are expressed in the classical joint coordinate space. The sparse parameterization has a modular property; when some rigid bodies and joints are added to (or removed from) the system, changes to its parameterization can be done simply by adding or removing the corresponding variables and constraints. Moreover, kinematic loops can be easily incorporated with no special treatment. The sparse parameterization obviously requires a greater number of variables and constraints than the dense parameterization. This can be a potential drawback, since in general, greater variable dimension indicates greater computation cost and slower convergence. On the other hand, the sparse parameterization lets the landscape of the constraint functions be simpler than that of the dense parameterization, and therefore, it is more likely to accept a greater step size when combined with a novel line search. Some comparison results between the two parameterizations are shown in Section IV.

In the following, a series of constraint conditions is introduced. Some are common to both sparse and dense parameterizations, while some others are unique to one parameterization. Throughout this section, the decision variables of trajectory planning are written in bold italic so that they can be easily distinguished from other symbols.

B. Constraints for the Smoothness of Trajectories

The trajectories of joint angles must be C^1 continuous in order to be executable on real robots. Otherwise, their second derivative (acceleration) will be impulsive and therefore such trajectories will be physically unrealizable. To fulfill this requirement, the joint angle trajectory of the joint j is expressed by a piecewise quadratic function of time given as follows:

$$\theta_j(t) = (1 - s^2)\theta_{j,k} + s^2\theta_{j,k+1} + s(1-s)h\nu_{j,k}$$
$$\text{if } \; t \in [t_k, t_{k+1}). \;\; (s = (t - t_k)/h).$$

Here, the following constraint is imposed to ensure that $\theta_j(t)$ is C^1 continuous at every discrete time instant

$$\frac{\boldsymbol{\theta}_{j,k+1} - \boldsymbol{\theta}_{j,k}}{h} - \frac{\boldsymbol{\nu}_{j,k+1} + \boldsymbol{\nu}_{j,k}}{2} = 0. \tag{12}$$

C. Constraints for Sparse Parameterization

The restriction of relative motion between a pair of rigid bodies imposed by a joint is expressed by the following set of constraints. Consider rigid bodies i_1 and i_2 connected by joint j. At the position level, we have

$$\boldsymbol{p}_{i_1,k} + \boldsymbol{q}_{i_1,k}\,\hat{p}_{i_1,j} = \boldsymbol{p}_{i_2,k} + \boldsymbol{q}_{i_2,k}\,\hat{p}_{i_2,j} \tag{13a}$$
$$\boldsymbol{q}_{i_1,k}\,\hat{q}_{i_1,j}\,q(\eta_j\,\boldsymbol{\theta}_{j,k}) = \boldsymbol{q}_{i_2,k}\,\hat{q}_{i_2,j} \tag{13b}$$

and at the velocity level, we have

$$\boldsymbol{v}_{i_1,k} + \boldsymbol{\omega}_{i_1,k} \times (\boldsymbol{q}_{i_1,k}\hat{p}_{i_1,j}) = \boldsymbol{v}_{i_2,k} + \boldsymbol{\omega}_{i_2,k} \times (\boldsymbol{q}_{i_2,k}\hat{p}_{i_2,j}) \tag{14a}$$
$$\boldsymbol{\omega}_{i_1,k} + (\boldsymbol{q}_{i_1,k}\hat{q}_{i_1,j}\eta_j)\boldsymbol{\nu}_{j,k} = \boldsymbol{\omega}_{i_2,k}. \tag{14b}$$

Here, $\hat{p}_{i,j}$ and $\hat{q}_{i,j}$ are parameters that determine the position and orientation of joint j, respectively, from the local coordinate frame of rigid body i. Moreover, η_j denotes the rotation axis of joint j.

From Newton–Euler equations of motion, we have

$$\frac{m_i}{h}(\boldsymbol{v}_{i,k+1} - \boldsymbol{v}_{i,k}) = m_i g + \sum_{j \in \mathcal{J}(i)} \rho(i,j)\boldsymbol{f}_{j,k} \tag{15a}$$

$$\frac{I_i(\boldsymbol{q}_{i,k})}{h}(\boldsymbol{\omega}_{i,k+1} - \boldsymbol{\omega}_{i,k}) = \boldsymbol{\omega}_{i,k} \times I_i(\boldsymbol{q}_{i,k})\,\boldsymbol{\omega}_{i,k}$$
$$+ \sum_{j \in \mathcal{J}(i)} \rho(i,j)[\boldsymbol{n}_{j,k} + (\boldsymbol{q}_{i,k}\,\hat{p}_{i,j}) \times \boldsymbol{f}_{j,k}]. \tag{15b}$$

Here, m_i denotes the mass of the rigid body and $I_i(\boldsymbol{q}_{i,k})$ denotes the inertia matrix with respect to the global coordinate frame (thus, dependent on the orientation $\boldsymbol{q}_{i,k}$). Moreover, $\mathcal{J}(i)$ denotes the set of joint indices connected to a rigid body i, and $\rho(i,j)$ is a function which takes 1 or -1 depending on whether the force of joint j applies positively or negatively to a rigid body i.

D. Constraints for Dense Parameterization

Due to limitation of space, we consider a simple case in which all rigid bodies belong to the same kinematic tree that has no loop. Moreover, the rigid body at the root of the tree is fixed in space. Extension to more general settings (multiple trees, floating base, loops) is possible but not explained here. From the equations of motion in the joint coordinate space, we have the following constraint:

$$\frac{M(\boldsymbol{\theta}_k)}{h}(\boldsymbol{\nu}_{k+1} - \boldsymbol{\nu}_k) + c(\boldsymbol{\theta}_k, \boldsymbol{\nu}_k) = \boldsymbol{\tau}_k. \quad (16)$$

Here, $\boldsymbol{\theta}_k$, $\boldsymbol{\nu}_k$, and $\boldsymbol{\tau}_k$ are vectors composed of the angles, velocities, and torques of all joints that belong to the tree, respectively. Moreover, $M(\boldsymbol{\theta}_k)$ and $c(\boldsymbol{\theta}_k, \boldsymbol{\nu}_k)$ are the inertia matrix and the Coriolis/centrifugal/gravitational force vector in the joint coordinate, respectively.

E. Tasks and Miscellaneous Constraints

A reaching task constrains the position and the orientation of a pair of rigid bodies to match during a certain time interval. Constraints for a reaching task n imposed between two rigid bodies with indices i_1 and i_2 for a time interval $[\underline{t}_n, \overline{t}_n]$ are given as follows:

$$\begin{aligned} \boldsymbol{p}_{i_1,k} &= \boldsymbol{p}_{i_2,k} \\ \boldsymbol{q}_{i_1,k} &= \boldsymbol{q}_{i_2,k} \qquad \forall\, k \text{ s.t. } t_k \in [\underline{t}_n, \overline{t}_n]. \end{aligned} \quad (17)$$

Here, orientation-matching constraint may be removed to realize position-only reaching, depending on the needs.

A collision avoidance task is expressed as a constraint that the bounding spheres of a pair of rigid bodies do not intersect during the task interval

$$\|\boldsymbol{p}_{i_1,k} - \boldsymbol{p}_{i_2,k}\|^2 \geq (r_{i_1} + r_{i_2})^2 \quad \forall\, k \text{ s.t. } t_k \in [\underline{t}_n, \overline{t}_n]. \quad (18)$$

Here, r_i denotes the radius of the bounding sphere of a rigid body i. One may use convex hulls instead of bounding spheres for more precise collision avoidance. Here, the technique that is proposed in [15] would be used to ensure that the distance function is C^1. Note that $\boldsymbol{p}_{i,k}$ that appears in the above two constraints is a function of $\boldsymbol{\theta}_k$ when the dense parameterization is used.

Range constraints can be imposed on any scalar-valued or vector-valued variable. Typical usage of range constraints include joint movable ranges and torque limits.

Some rigid bodies must have their trajectories fixed. These include static objects, the base links of robotic arms, and objects that follow predefined reference trajectories. In the sparse parameterization, the trajectory of a rigid body i can be made fixed by the following slight modifications:

1) Treat the variables $(\boldsymbol{p}_{i,k}, \boldsymbol{v}_{i,k}, \boldsymbol{q}_{i,k}, \boldsymbol{\omega}_{i,k})$ as constants;
2) Disable the constraints (15a)(15b) for rigid body i.

In a similar manner, joint trajectory can be fixed as well.

Remark 3: A special care is required for handling quaternion variables, which appear in the sparse parameterization. It is inefficient to treat a quaternion q as a vector in $\mathbb{R}^4$ with an additional constraint $\|q\| = 1$, because it introduces extra nonlinear equality constraints into the problem. Instead, the direction of change of q is expressed by a vector $\delta q \in \mathbb{R}^3$. Given δq and a step size α, q is updated by $q := q(\alpha\, \delta q)\, q$ (recall the definition of $q(\Omega)$ mentioned earlier). Differentiation of constraint functions with respect to quaternions is made based on this update law. The details are omitted due to limitation of space.

IV. Examples

The performance of the proposed planning algorithm is evaluated in some examples. In all examples, the algorithm is implemented in a C++ program and run on a computer with 2.8-GHz CPU.

A. Reaching Task of a 3-Degree-of-Freedom Robotic Arm

First, the planning algorithm is applied to a reaching task of a 3-DOF robotic arm. The first objective of this example is to demonstrate the capability of the prioritized trajectory planner to produce a variety of trajectories according to different constraint settings. For the sake of visualization, the motion of the robotic arm is limited on a 2-D plane. The second objective is to compare computational performances between different problem formulations, scene representations, and solution algorithms. The robotic arm is composed of five rigid bodies: the base link, three intermediate links (1, 2, and 3), and the hand. These bodies are connected by joints in this order. The angle of the joint connecting link 3 and the hand is fixed; the hand is therefore fixed to one end of link 3. The task is to move the hand to two targets (1 and 2) sequentially. To achieve this, reaching tasks without orientation matching are created between the hand and the targets, which are named task 1 and 2. The duration of the tasks are: $[\underline{t}_1, \overline{t}_1] = [2.45, 2.55]$, $[\underline{t}_2, \overline{t}_2] = [4.45, 4.55]$. The length of the planning horizon is set as $T = 5$ and the time resolution is set as $h = 0.5$, which implies $N = 10$.

Planned trajectories under some different settings are shown in Fig. 1. The results are obtained by the proposed algorithm using the sparse parameterization. In all cases, the priority of task 1 is set as 1 and that of task 2 is set as 2. The priority of other constraints are set as 0. In case (a), both targets are inside the reachable range of the hand. As a result, the hand is moved to the targets during the respective task intervals. In case (b), target 1 is outside the reachable range. In this case, the distance to target 1 is minimized, while reaching target 2 precisely. In case (c), a velocity limit $[-20, 20]$ [°/s] is imposed on each joint with the priority 0 so that the robot cannot reach both the targets. As a result, target 1 is precisely reached, while the distance to the target 2 is minimized using the remaining degrees of freedom. In case (c), some residual error is observed in the constraint of task 1. This is mainly due to the fact that the Gauss–Seidel iteration in QP is terminated after a finite number of iterations; therefore, the residual error can be reduced by increasing the number of iterations, at the expense of greater computation cost. Case (d) is a case of two incompatible tasks. Here, target 1 is in a different position and it is to be reached by the middle of the second link. Moreover, the task duration is set as $[\underline{t}_1, \overline{t}_1] = [4.45, 4.55]$, which is the same as task 2. The result shows that task 1 is achieved precisely, while the distance between the hand and target 2 is minimized.

Next, we compare the computational performance of four different combinations: Pareto-sparse, Pareto-dense, WGP-dense, and LGP-dense. Case (b) of the above example is used for comparison. Pareto-sparse and Pareto-dense are the combinations of the proposed prioritized Pareto algorithm and the sparse and dense scene parameterizations, respectively. WGP(LGP)-dense compute the conventional WGP(LGP) based on the dense scene parameterization using the IPOPT nonlinear constrained optimization solver [22] (see [23] for application to trajectory planning of humanoids).

WGP-dense minimizes the weighted cost function (2), while LGP-dense computes a series of constrained minimization problems (3). Here, $\phi_l(z)$ is given by $\phi_l(z) = \|e_l(z)\|^2$. For WGP-dense, the weighting parameters are set as $w_0 = 10.0$, $w_1 = 1.0$, and $w_2 = 1.0$. For each setting, we consider cases with and without the dynamics constraints [(15a) and (15b) for the sparse parameterization and (16) for the dense

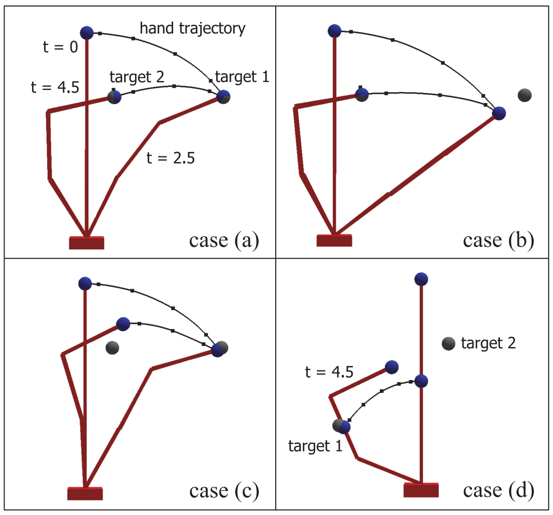

Fig. 1. Planned trajectories in reaching task of a 3-DOF robotic arm.

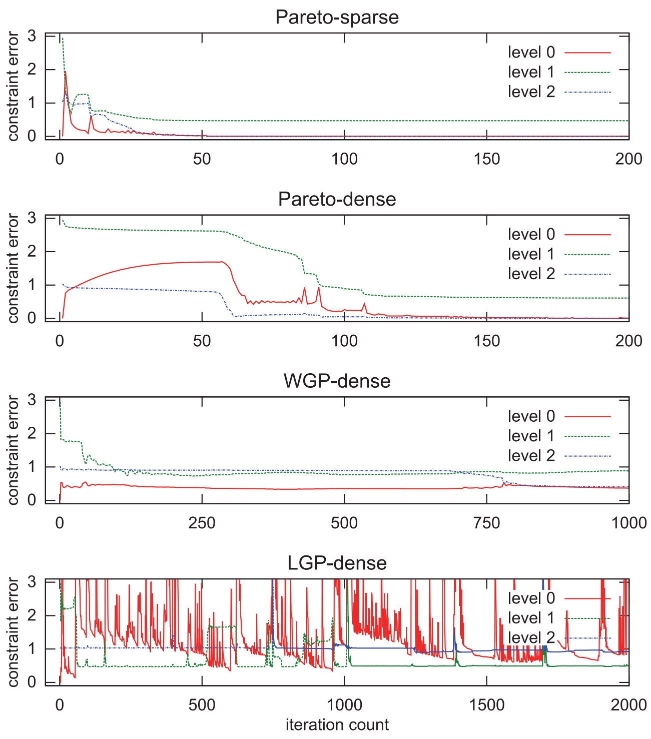

Fig. 2. Comparison of convergence results [for case (b), dynamics constraint enabled].

TABLE II
PROBLEM SIZE AND COMPUTATION PERFORMANCE IN EXAMPLE A, CASE (B)

i) Number of variables and constraints

		n_z	n_c $l = 0$	$l = 1$	$l = 2$
sparse	w/o (15a)(15b)	908	652	6	6
sparse	w/o (15a)(15b)	908	892	6	6
dense	w/o (16)	124	84	6	6
dense	w/o (16)	124	124	6	6

ii) Number of iterations and computation time

		phase	iter.	comp.time[s]
Pareto-sparse	w/o (15a)(15b)	-	19	0.738
Pareto-sparse	w/o (15a)(15b)	-	52	2.958
Pareto-dense	w/o (16)	-	12	0.145
Pareto-dense	w/o (16)	-	121	3.290
WGP-dense	w/o (16)	-	53	1.805
WGP-dense	w/o (16)	-	1000	35.395
LGP-dense	w/o (16)	0	1	0.088
		1	21	1.310
		2	22	2.061
LGP-dense	w/o (16)	0	1	0.092
		1	1000	57.813
		2	1000	84.230

parameterization]. This is because these constraints have great influence on the convergence property of the methods. Finally, the results of WGP(LGP) combined with the sparse parameterization are not presented here. This is because quaternions must be treated as 4-D vectors with norm constraints in IPOPT, and in our experience, this resulted in very poor convergence and frequent optimization failure.

Fig. 2(a)–(d) shows the convergence results of the above four settings with the dynamics constraints enabled. In each figure, the lines depict the change of $\sqrt{\phi_l}$ of priority levels 0, 1, and 2 during iteration. Table II(i) shows the number of variables and that of constraints in each priority level. Table II(ii) shows for each setting the number of iterations for convergence and the total computation time. For LGP-dense, the iteration count and the computation time at different minimization phases (minimization of ϕ_0, ϕ_1, and ϕ_2) are shown. The maximum iteration count of IPOPT is set as 1000. Pareto-sparse shows the fastest convergence, both in terms of the number of iterations and computation time. Pareto-dense shows relatively slow convergence compared with Pareto-sparse. One reason is that the dense nature of constraints prevented the algorithm from taking a large step size at each iteration. Moreover, we observe temporary increase of constraint errors, while theoretically constraint errors should be nonincreasing. This is because the step size α is lower-bounded by $\alpha^{\min}$, and as a result, errors of nonlinear equality constraints may increase when the constraints are almost strictly satisfied. The computation time of WGP-dense is similar to that of the above two, but it exhibits some residual error at each priority level after convergence. While tuning the weights, we observed that increasing the weight w_l would reduce the residual error of ϕ_l, but at the same time it would increase the residual errors of other priority levels. This indicates that WGP is likely to converge to a local minimum. In LGP-dense, the ϕ_0 phase terminates almost immediately, since constraints with priority 0 are all satisfied at the starting point. The ϕ_1 and ϕ_2 phases show nonmonotone convergence. Temporary increase of cost functions is observed during the "restoration phase" of IPOPT. This reflects the difficulty of minimizing a cost function, while maintaining a set of nonlinear equality constraints $\phi_l(z) = \phi_l^* \, \forall l \in [0 : L-1]$, which involve a large number of variables.

B. Reaching Task of a 6-Degree-of-Freedom Robotic Arm

In this section, a reaching task of a 6-DOF robotic arm, which involves 3-D motion, is investigated. The aim of this example is to demonstrate the use of joint velocity limit constraint with low priority, and to investigate the influence of the time resolution h on constraint errors between discrete time instants. Three targets are to be reached by the robot hand, with the task execution periods $[1.45, 1.55]$, $[2.95, 3.05]$, and $[4.7, 4.8]$. At this time, not only position-matching but orientation matching constraints are enabled as well. Moreover, a single obstacle is placed in the workspace and a collision avoidance task is assigned to each link of the robotic arm and the obstacle. Furthermore, range

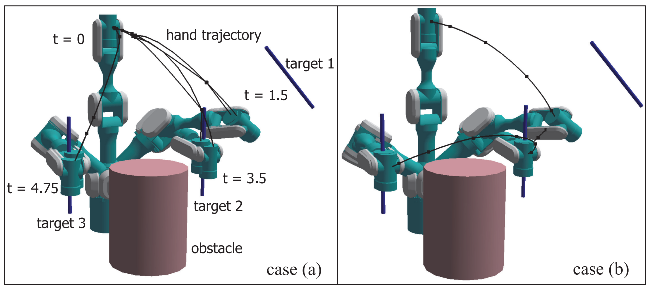

Fig. 3. Planned trajectories in reaching task of a 6-DOF robotic arm.

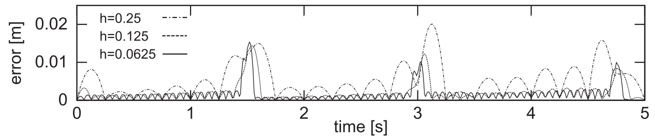

Fig. 4. Constraint errors for different values of h.

constraints are imposed on the joint velocity. In case (c) in the previous example, velocity limits had the priority 0, but this time velocity limits are set as $[0, 0]$ and are assigned a priority lower than other types of constraints. In this manner, a trajectory that achieves reaching and collision avoidance with smaller and thus efficient movement is planned. The types of constraints and their priorities are: velocity limit (3), reaching (2), collision avoidance (1), and other constraints (0). The planning horizon is set as $T = 5$, $h = 0.25$, and $N = 20$. The planned trajectories that are obtained by the Pareto-sparse setting are visualized in Fig. 3. It is observed in the figures that the robot moves its hand to the three targets, while avoiding collision with the obstacle.

In particular, Target 1 is outside the reachable range of the robot; therefore, the distance between the hand and the target is made as small as possible, while their orientation is matched. Although it is difficult to see from a single view point, each link of the robotic arm and the obstacle never intersect throughout the planning horizon. Case (a) is shown for comparison; here, the dynamics constraints (15 a) and (15 b), and the velocity limit are turned off. The planner still produces a kinematically realizable trajectory without these constraints. The figure shows, however, that the planned trajectory is winding and the hand travels a long distance. The reason we obtain such a trajectory is that the planning algorithm has a property that it outputs a trajectory that is close to the initial trajectory, which in this case is a trajectory that the robotic arm stays still in the upright posture. In case (b), on the other hand, the dynamics and the velocity limit constraints are turn on. Thanks to these constraints, the planned trajectory exhibits more efficient movement.

Next, planning results with different values of h (0.25, 0.125, and 0.0625) are compared to investigate the influence of h on constraint errors between discrete time points. Here, we focus on the translational kinematic constraint (13 a). The result is shown in Fig. 4. The horizontal axis depicts the time along the planning horizon and the vertical axis depicts the constraint error, which is given by the norm of the difference of the left- and right-hand sides of (13 a) summed over all joints. To evaluate (13 a) at arbitrary time instants $t \in [t_k, t_{k+1})$, $p(t)$ (subscript is omitted) is obtained by quadratic interpolation using p_k, v_k, and p_{k+1}; and $q(t)$ is obtained by spherical linear interpolation (SLERP) using q_k and q_{k+1}. In every case, the constraint error tends to show relatively large values in the middle of discrete time points. Nevertheless, the error is not more than 0.02[m], which is considered sufficiently small for most trajectory planning applications. Table III shows the problem size and the computational cost of each case. Iteration was terminated when $\alpha\|\delta z\|$ became smaller than the threshold value 0.05. Although not used in this paper, a warm-start technique that uses a previously obtained trajectory as an initial guess for optimization with a finer time resolution may be useful for reducing the computation time.

TABLE III
PROBLEM SIZE AND COMPUTATION PERFORMANCE IN EXAMPLE B

h	n_z	n_c				iter.	comp.
		0	1	2	3		time[s]
0.25	2682	2598	336	36	126	103	8.38
0.125	5322	5118	656	36	246	220	46.22
0.0625	10602	10158	1296	36	486	429	209.84

V. Conclusion

In future work, the developed trajectory planner will be incorporated into a real-time control loop in which planning is executed at each control cycle to adapt to unexpected events such as change of obstacle positions. The current computational performance of the algorithm allows a control period of a few seconds, which is considered acceptable for a high-level control loop. The accuracy of trajectory realization on a real robot should also be investigated.

Appendix A
Proof of Lemma 1

For ease of notation, let us write the right-hand side of (7) $\mathcal{D}^{\text{par}}_{0:L}$. Let $\delta z \in \mathcal{D}^{\text{lex}}\langle \phi_0, \phi_1, \ldots, \phi_L \rangle(z)$. By definition, there exists $l \in [0 : L]$ such that

$$\delta\phi_{l'} = 0 \; \forall l' \in [0 : l-1] \text{ and } \delta\phi_l < 0.$$

This implies $\delta z \in \mathcal{D}^{\text{par}}\langle[\phi_0; \phi_1; \ldots; \phi_l]\rangle$, and therefore, $\delta z \in \mathcal{D}^{\text{par}}_{0:L}$.

Next, let $\delta z \in \mathcal{D}^{\text{par}}_{0:L}$. By definition, there exists $l \in [0 : L]$, such that $\delta z \in \mathcal{D}^{\text{par}}\langle[\phi_0; \phi_1; \ldots; \phi_l]\rangle(z)$. This implies that there exists $l' \in [0 : l]$ such that

$$\delta\phi_{l''} = 0 \; \forall l'' \in [0 : l'-1] \text{ and } \delta\phi_{l'} < 0,$$

and therefore, $\delta z \in \mathcal{D}^{\text{lex}}\langle \phi_0, \phi_1, \ldots, \phi_L \rangle(z)$. This completes the proof.

Appendix B
Proof of Lemma 2

From the constraint definition, we have that either $\delta z^* \in \mathcal{D}^{\text{par}}\langle e_{\mathcal{I}_{[0:l]}}\rangle(z)$ or $\delta z^* = 0$ must hold. Moreover, if $\delta z^* = 0$, then $\delta e^* = 0$. Therefore, it is trivial that $\mathcal{D}^{\text{par}}\langle e_{\mathcal{I}_{[0:l]}}\rangle(z) = \emptyset \Rightarrow (\delta z^*, \delta e^*) = (0, 0)$.

What remains to be proved is $\mathcal{D}^{\text{par}}\langle e_{\mathcal{I}_{[0:l]}}\rangle(z) \neq \emptyset \Rightarrow (\delta z^*, \delta e^*) \neq (0, 0)$. Given a Pareto direction δz, one can construct a feasible solution $(\delta z, \delta e)$ for which at least one component of δe is strictly negative. Moreover, $(\beta\delta z, \beta\delta e)$ with $\beta > 0$ is a feasible solution and $\beta\delta z$ is a Pareto direction. Now, the difference of the cost of $(\beta\delta z, \beta\delta e)$ and that of $(0, 0)$ is given by

$$\begin{aligned}\|\Gamma^z\beta\delta z\|^2 + \|\Gamma^e(\beta\delta e + \gamma e(z))\|^2 - \|\Gamma^e\gamma e(z)\|^2 \\ = \beta^2(\|\Gamma^z\delta z\|^2 + \|\Gamma^e\delta e\|^2) + 2\beta\gamma\Gamma^e\delta e^{\mathsf{T}} e(z).\end{aligned}$$

Here, the term $\delta e^{\mathsf{T}} e(z)$ is strictly negative. Therefore, the right-hand side can be made negative by choosing sufficiently small β. This implies $(\beta\delta z, \beta\delta e)$ with appropriate choice of β gives a smaller cost than $(0, 0)$, and therefore, $(\delta z^*, \delta e^*) \neq (0, 0)$. This completes the proof.

REFERENCES

[1] A. Witkin and M. Kass, "Spacetime Constraints," in *Proc. 15th Annu. Conf. Compute Graph. Interact. Techn.*, 1988, pp. 159–168.

[2] M. Diehl, H. G. Bock, H. Diedam, and P. B. Wieber,"Fast direct multiple shooting algorithms for optimal robot control," in *Fast Motions in Biomechanics and Robotics*, (Lecture Notes in Control and Information Sciences), New York, NY, USA: Springer, 2006, vol. 340, pp. 65–93.

[3] S. H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, "Newton-type algorithms for dynamics-based robot movement optimization," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 657–667, Aug. 2005.

[4] K. Mombaur, "Using optimization to create self-stable human-like running," *Robotica*, vol. 27, no. 3, pp. 321–330, 2008.

[5] K. Wampler and Z. Popović, "Optimal gait and form for animal locomotion," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 60-1–60-8, 2009.

[6] S. Lengagne, P. Mathieu, A. Kheddar, and E. Yoshida, "Generation of dynamic motions under continuous constraints: Efficient computation using B-splines and taylor polynomials," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 698–703.

[7] M. Posa and R. Tedrake,"Direct trajectory optimization of rigid body dynamical systems through contact," in *Algorithmic Foundations of Robotics X*, New York, NY, USA: Springer, 2013, vol. 86, pp. 527–542.

[8] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *Int. J. Humanoid Robot.*, vol. 2, no. 4, pp. 505–518, 2005.

[9] L. Sentis, J. Park, and O. Khatib, "Compliant control of multicontact and center-of-mass behaviors in humanoid robots," *IEEE Trans. Robot.*, vol. 26, no. 3, pp. 483–501, Jun. 2010.

[10] F. Keith, N. Mansard, S. Miossec, and A. Kheddar, "Optimization of tasks warping and scheduling for smooth sequencing of robotic actions," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 1609–1614.

[11] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *Int. J. Robot. Res.*, vol. 30, no. 12, pp. 1435–1460, 2011.

[12] M. de Lasa, A. Hertzmann, "Prioritized optimization for task-space control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 5755–5762.

[13] M. de Lasa, I. Mordatch, and A. Herzmann, "Feature-based locomotion controllers," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 131-1–131-10, 2010.

[14] O. Kanoun, F. Lamiraux, and P. B. Wieber, "Kinematic control of redundant manipulators: Generalizing the task priority framework to inequality tasks," *IEEE Trans. Robot.*, vol. 27, no. 4, pp. 785–792, Aug. 2011.

[15] A. Escande, S. Miossec, and A. Kheddar, "Continuous gradient proximity distance for humanoids free-collision optimized-postures," in *Proc. IEEE-RAS 7th Int. Conf. Humanoid Robots*, 2007, pp. 188–195.

[16] Y. Tazaki, H. Sugiura, H. Janssen, and C. Goerick, "Decentralized planning for dynamic motion generation of multi-link robotic systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 1582–1587.

[17] S. Suzuki, Y. Tazaki, and T. Suzuki, "Simultaneous optimization of timing and trajectory in sequential and parallel tasks of humanoid robots," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2011, pp. 596–601.

[18] D. Jones and M. Tamiz, "*Practical goal programming" in International Series in Operations Research & Management Science*, New York, NY, USA: Springer-Verlag, 2010.

[19] R. Hettich,"An implementation of a discretization method for semi-infinite programming," in *Mathematical Programming*, New York, NY, USA: Springer, 1986, vol. 34, pp. 354–361.

[20] R. Reemtsen and J. J. Rückmann, "*Semi-infinite programming " in Nonconvex Optimization and Its Applications*. New York, NY, USA: Springer-Verlag, 1998.

[21] R. Cottle, J. Pang, and R. E. Stone, "The linear complementarity problem," *Classics Appl. Math.*, vol. 60, 1992. Available: http://www.amazon.com/Complementarity-Problem-Classics-Applied-Mathematics/dp/0898716861/ref=sr_1_1?s=books&ie=UTF8&qid=1399969978&sr=1-1&keywords=linear+complementarity+problem.

[22] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," in *Mathematical Programming*, New York, NY, USA: Springer, 2006, vol. 106, pp. 25–57.

[23] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, "Generation of whole-body optimal dynamic multi-contact motions," *Int. J. Robot. Res.*, vol. 32, no. 9–10, pp. 1104–1119, 2013.

Balancing in Dynamic, Unstable Environments Without Direct Feedback of Environment Information

Umashankar Nagarajan and Katsu Yamane

***Abstract*—This paper studies the balancing of simple planar bipedal robot models in dynamic, unstable environments such as seesaw, bongoboard, and board on a curved floor. This paper derives output feedback controllers that successfully stabilize seesaw, bongoboard, and curved floor models using only global robot information and with no direct feedback of the dynamic environment and, hence, demonstrates that direct feedback of environment information is not essential for successfully stabilizing the models considered in this paper. This paper presents an optimization to derive stabilizing output feedback controllers that are robust to disturbances on the board. It analyzes the robustness of the derived output feedback controllers to disturbances and parameter uncertainties and compares their performance with similarly derived robust linear quadratic regulator controllers. This paper also presents nonlinear simulation results of the output feedback controllers' successful stabilization of bongoboard, seesaw, and curved floor models.**

***Index Terms*—Balance control, legged robots, robot control, underactuated robots.**

I. INTRODUCTION

Balancing and postural stabilization is one of the most widely researched topics in bipedal robotics [1]–[3]. Unlike balancing in the sagittal (frontal) plane, where bipedal robots can exploit their legs' passive dynamics [4], [5], significant active control is essential to stabilize their motions in the coronal (lateral) plane [6]. Several balancing control strategies for stabilizing the unstable dynamics of 3-D passive dynamic walkers in their coronal planes were presented in [7]. The balance recovery strategies of humans balancing on slacklines were studied in [8]. In [9], humans balancing on tightropes and slacklines were modeled as cart-poles balancing on circular tracks, and several balancing controllers for these simplified models were derived.

Momentum-based control strategies that successfully stabilize humanoid robots on non-level, rocking floors were presented in [10] and [11]. They directly determined center of pressure and ground reaction forces at each support foot to achieve the desired momenta. They, however, did not deal with unstable environments like seesaw or bongoboard. Controllers that enable planar bipedal robots to walk on a rolling cylinder were presented in [12] and [13]. Approximate value function-based control approaches to stabilize humanoid robots on bongoboards were presented in [14], while adaptive policy-mixing control strategies for stabilizing humanoid robots on a seesaw were presented in [15]. However, all these approaches used the environment information directly for feedback control.

This paper studies the balancing of simple planar bipedal robot models, modeled as four-bar linkages, in dynamic, unstable environments,

Manuscript received September 16, 2013; revised January 26, 2014; accepted April 23, 2014. Date of publication May 22, 2014; date of current version September 30, 2014. This paper was recommended for publication by Associate Editor N. Mansard and Editor A. Kheddar upon evaluation of the reviewers' comments.

U. Nagarajan was with Disney Research, Pittsburgh PA 15213 USA. He is now with Honda Research Institute USA Inc., Mountain View, CA 94043 USA (e-mail: unagarajan@hra.com).

K. Yamane is with Disney Research, Pittsburgh PA 15213 USA (e-mail: kyamane@disneyresearch.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2014.2320797