Comprehensive theory of differential kinematics and dynamics towards extensive motion optimization framework

Ko Ayusawa and Eiichi Yoshida

Abstract
This paper presents a novel unified theoretical framework for differential kinematics and dynamics for the optimization of complex robot motion. By introducing an 18×18 comprehensive motion transformation matrix, the forward differential kinematics and dynamics, including velocity and acceleration, can be written in a simple chain product similar to an ordinary rotational matrix. This formulation enables the analytical computation of derivatives of various physical quantities (e.g. link velocities, link accelerations, or joint torques) with respect to joint coordinates, velocities and accelerations for a robot trajectory in an efficient manner (O(NJ)), where NJ is the number of the robot’s degree of freedom), which is useful for motion optimization. Practical implementation of gradient computation is demonstrated together with simulation results of robot motion optimization to validate the effectiveness of the proposed framework.

Keywords
motion optimization, forward kinematics, inverse dynamics, Jacobian matrix, gradient computation, comprehensive motion transformation matrix

1. Introduction
In recent robotics research, the importance of optimization has increased in a variety of aspects. Classic inverse kinematics and inverse dynamics problems (Nakamura, 1991) can be handled as optimization problems of joint coordinates or torques subject to some constraints that are derived from physical consistency or desired tasks. Some practical optimization frameworks have been proposed and applied to not only motion planning and control of a humanoid robot (Escande et al., 2014; Suleiman et al., 2008) but also to motion reconstruction (Yamane and Nakamura, 2003), contact estimation, and musculoskeletal analysis (Delp and Loan, 2000; Nakamura et al., 2005) of a digital human model. The model-identification problem (Khalil and Dombre, 2002) is also an optimization problem with respect to the model parameters with inverse dynamics computations. In each of these basic problems, only one type of physical quantity is optimized. For example, inverse kinematics computes the joint coordinates, inverse dynamics the joint torques, identification of the inertial parameters, etc.

However, practical problems are usually combinations of different problems, and different physical quantities must be simultaneously optimized for several different times. Trajectory optimization subject to physical consistent conditions is a typical example. In locomotion planning, because the violation of conditions at a certain time instance leads to future risk of falls, several sets of joint coordinates during a certain period often must be optimized simultaneously to predict future risks. Because this optimization usually requires a huge computational cost, the balancing problem is often simplified, perhaps by utilizing a low-dimensional model (Kajita et al., 2003a). Another example is the kinematic calibration of human body segments (Kirk et al., 2005) from motion capture measurements because the joint angles cannot be measured directly by encoders unlike in standard robot calibration (Khalil and Dombre, 2002). This application requires simultaneous optimization of the geometric parameters and the generalized coordinates (Ayusawa and Yoshida, 2017b).

More recently, the application of optimization has been intensively investigated in the field of anthropomorphic systems: humanoid robotics and human motion analysis. Humanoid robots are expected to execute more complicated and practical tasks such as disaster response (Lim et al., 2016), evaluation of human oriented products as a physical human simulator (Miura et al., 2013), etc. In such...
applications, anthropomorphic motion optimization faces far more complex problems combining modeling, kinematics, dynamics, planning, and control. For instance, the identification of the inertial parameters of a humanoid robot is important to realize precise and dynamic control (Ayusawa et al., 2014). However, the optimal motion generation to maximize the total identification performance is a complex problem of trajectory optimization and a balancing problem (Bonnet et al., 2016). In humanoid applications such as an active dummy for assistive device evaluation, imitation of human-like motion is necessary. This technique, called motion retargeting (Gleicher, 1998; Pollard et al., 2002), usually involves the inverse kinematics problem for both the human and humanoid, identification of the morphing function, and motion control of a robot while considering physical consistency. Yet another example is human simulators: recent detailed simulators are often connected to other simulation systems such as deformation computation (e.g., dynamics simulation using finite element method (FEM) (Allard et al., 2007).) Those simulators are often used to optimize the motion of a digital human or parameters of a product to be designed with the simulator. In such problems, we usually solve a simultaneous problem with modeling, kinematics, and dynamics problems (Ayusawa and Yoshida, 2017b). However, the above issues are currently difficult to solve, and some of them are still open problems due to the complexity of the derivative computation as described below.

To establish a comprehensive optimization framework that can handle critical issues required from the computational and practical points of view, the partial derivative of any physical quantities with respect to the joint coordinates is important when evaluating various types of conditions represented by the coordinates, derivatives, and forces in both Cartesian and joint space, even though some optimization techniques do not require the derivatives. Suleiman et al. (2008) developed the fundamental framework of humanoid motion optimization. Their work utilized the works of Park et al. (1995) and Sohl and Bobrow (2000) and formulated the analytical partial derivatives of the Cartesian coordinates, their derivatives, and joint torques with respect to the joint coordinates and their derivatives. Despite its possibility of extension to handle the many types of problems mentioned above, two issues remain. First, because the formulation mainly focuses on the manifold of Cartesian spaces, it is difficult to handle a free-floating base and spherical joints that are often used in human and humanoid kinematics modeling (Yamane, 2004). To represent the orientation of those joints, the generalized coordinates need to contain the rotation manifolds. The previous work has therefore difficulty in formulating the partial derivatives to handle the differential relationship of manifolds between the Cartesian and generalized coordinates. The second issue is the computational complexity: if we compute the partial derivative of the joint torque for the base-link, the computational complexity is proportional to the square of the degree of freedom (DOF), i.e. $O(N_J^2)$, where $N_J$ is the number of DOF. This leads to huge computational costs for the optimization when dealing with a large-DOF system. In the field of computer animation, Fang and Pollard (2003) presented an efficient method to compute the Jacobian of the total force acting on the whole body with computational complexity $O(N_J)$. Unfortunately, the formulation does not provide the Jacobian for the joint constraint force or joint torque, except for the root link. An efficient formulation about the Jacobian for any joints needs to be investigated.

In this paper, we reformulate the differential kinematics and dynamics for the fast computation of the analytical partial derivatives of Cartesian variables and generalized forces with respect to the joint coordinates and their derivatives. We introduce an 18-dimensional Comprehensive Motion Transformation Matrix (CMTM) in order to formulate the standard forward differential kinematics problem. This formulation makes it possible to reduce the computation of differential forward kinematics of kinematic chain to a simple chain product of the matrices in a similar manner to the standard rotational matrix, or the 6-dimensional matrix used in adjoint map on SE(3) (Park et al., 1995). The CMTM also allows the formulation of an analytical form of several partial derivatives with respect to the joint coordinates and their derivatives including different types of joints. The partial derivatives of link variables are extended form of the basic Jacobian matrix (Khatib, 1987), and can be derived from the same formulation used in the basic Jacobian. The Jacobian of the joint torque is also extended from the linear/angular momentum Jacobian (Kajita et al., 2003b; Sugihara and Nakamura, 2002), which is also formulated in the same manner via the CMTM. The analytical derivative of physical quantities such as the zero moment point (ZMP) (Vukobratovic et al., 1970) can be easily computed with the proposed method. In addition, each computational cost of the new Jacobians is $O(N_J)$. A recent computational technique called automatic differentiation is also expected to compute the Jacobian matrix with $O(N_J)$. Though the automatic differentiation cannot provide the symbolic formula unlike algebraic differentiation, it can quickly compute the derivatives with high accuracy in contrast to numerical differentiation. The computational speed of the proposed method is also compared to the automatic differentiation.

Though it is important to derive the Jacobian matrices theoretically, from a practical point of view in the motion optimization, the direct computation of the Jacobian matrices is not always computationally efficient. When computing the gradient of the cost function or the function for each constraint, its computational cost can be further reduced by decomposing the gradient computation into the combination of kinematic and dynamics computation (Ayusawa and Nakamura, 2012). The decomposed gradient computation (DGC) does not require the direct computation of the Jacobian matrix. Based on our previous work (Ayusawa and Yoshida, 2017a), this paper newly presents the efficient decomposed gradient computation for the proposed
comprehensive theory for motion optimization. Numerical simulations of computational cost and simulation results of motion optimization for a redundant manipulator and a humanoid robot are provided to demonstrate the effectiveness of the proposed method.

2. Motion optimization framework

This section presents the overview of the motion optimization problem and the flow of the computation. Let the generalized coordinates of a robot be \( \mathbf{q} \), with their trajectories parameterized by \( \mathbf{a} \) and time instance \( t : \mathbf{q}(\mathbf{a}, t) \). In this paper, the trajectories are represented by, for example, polynomial interpolation, Fourier series, B-splines, etc. Their derivatives \( \dot{\mathbf{q}} \) and \( \ddot{\mathbf{q}} \) are computed with \( \mathbf{a} \) and \( t \) according to the implemented trajectory parameterization.

Let us concatenate \( \mathbf{q} \), \( \dot{\mathbf{q}} \), and \( \ddot{\mathbf{q}} \) into \( \mathbf{x} \), and consider the physical quantities \( \mathbf{y} \) that are represented by \( \mathbf{x} \). The candidates of \( \mathbf{y} \) could be, for example, the position, orientation, linear and angular velocity, linear and angular acceleration of each link coordinate, the joint torques and the constraint forces acting on the joint coordinates, etc. Let \( y_{ij} \) denotes \( i \)th quantity at the \( j \)th time instance \( t_j \), \( \mathbf{x} \) the coordinates and their derivatives at \( t_j \), and \( \mathbf{Y} \) the entire set of the quantities to be evaluated. In this paper, the set of time instances \( t_j \) is given and constant, and the following optimization problem is to be solved

\[
\min_{\mathbf{a}} \ c(\mathbf{Y}) \tag{1}
\]

subject to \( \forall \ h \ g_k(\mathbf{Y}) \leq 0 \)

where, \( c \) is the cost function to be evaluated, and \( g_k \) is \( k \)th inequality constraint. Because equality constraints can be represented by two inequality constraints, they are summarized and represented by the inequality form.

The above optimization problem is usually computationally expensive. To increase the computational speed, efficient optimization techniques often require analytical gradient computation of the cost function and each constraint. The gradient can be decomposed as follows

\[
\frac{\partial h}{\partial \mathbf{a}} = \sum_{i} \sum_{j} \frac{\partial h}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{a}} \quad (h = c \ or \ g_k) \tag{2}
\]

The gradient \( \partial h / \partial y_{ij} \) is determined by the form of the cost function or the constraints. The partial derivative \( \partial \mathbf{x} / \partial \mathbf{a} \) can be computed from the implemented trajectory parameterization. The term \( \partial y_{ij} / \partial \mathbf{x} \) represents the partial derivative of several types of quantities of multi-body systems with respect to the joint coordinates and their derivatives. A typical example is the partial derivative of the position and orientation of each link with respect to the joint coordinates, which is known as the basic Jacobian (Khatib, 1987). In this case, the derivatives of the velocities, the accelerations, and the joint torques with respect to \( \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} \) are required. By utilizing the analytical formulations for manipulators (Park et al., 1995; Sohl and Bobrow, 2000), the motion optimization framework of Equation (1) was applied to a humanoid robot by Suleiman et al. (2008). Though the utilization can handle many types of motion optimization, there remain theoretical and practical issues in order to solve the motion optimization for humanoid systems.

First, the formulation should be extended for spherical joints, a free-floating base, or perhaps other types of joints in order to be applied to anthropomorphic systems because the formulations are originally for manipulators composed of rotational or translational actuators only. Though spherical joints or a free-floating base can be modeled by multiple rotational or translational joints, it often leads to the singularity problem of representing joint orientations. The link mass properties also need to be assigned to the corresponding multiple joints, which requires additional constraints on inertial properties and makes it difficult to perform dynamics analysis such as identification. The second issue is the computational complexity of the computation. The formulations in Suleiman et al. (2008) basically utilize the classical recursive formula of forward kinematics and inverse dynamics (Luh et al., 1980). The derivatives with respect to the coordinates of one joint are computed according to the recursive formula, and the same procedure is applied for every set of joint coordinates. Therefore, when computing the partial derivative of the variables of one link, the computational complexity is almost \( O(N_f^2) \), where \( N_f \) is the number of DOF.

In the field of computer animation, a similar framework of motion optimization has been applied to generate the motion of a human figure. Fang and Pollard (2003) presented an efficient method to compute the Jacobian of the total force and moment acting on the whole body with computational complexity \( O(N_f) \). Their method utilizes the recursive equations about the total external force and moment acting on the subsystem of the kinematic tree, where the subsystem is constructed recursively by assembling the links from the leaf-side. Unfortunately, the formulation does not provide the Jacobian for the joint constraint forces or joint torques, because the total external force acting on the subsystem is not equal to the joint constraint force, except for the root link. For the optimization including any joint torques, an efficient formulation about the Jacobian for any joints with \( O(N_f) \) is needed.

As mentioned above, one typical example of \( \partial y_{ij} / \partial \mathbf{x} \) is the basic Jacobian, and the computational cost of a standard basic Jacobian is \( O(N_f) \). This paper introduces an \( 18 \times 18 \) matrix that can represent the forward kinematics computation including velocities and accelerations via simple chain products. The matrix has the same features as a \( 6 \times 6 \) transformation matrix that represents position and orientation, as discussed later. By utilizing this matrix together with the formulation of the basic Jacobians, the computation method for an arbitrary \( \partial y_{ij} / \partial \mathbf{x} \) is introduced in this paper. Specifically, the derivation procedure of COM Jacobian (Sugihara and Nakamura, 2002) is utilized for computation of the
derivatives of the joint torques. In the formulation, several types of joints can also be handled, and the computational complexity is \( O(N_J) \). The computation of the Jacobian matrices is detailed in section 4.

The direct computation of Jacobian matrices does not always lead to the fast optimization. The gradient computation of \( h \) can be accelerated by avoiding the direct computation of Jacobian matrices, as seen in an efficient inverse kinematics computation for large-DOF system (Ayusawa and Nakamura, 2012). The computational speed is improved by decomposing the gradient computation into the combination of the forward kinematics and inverse dynamics computation. An efficient computation of the gradient inspired by such a method is introduced in section 5.

3. Mathematical notations and comprehensive motion transformation matrix

This section presents the preliminary notation of variables in the paper, and introduces a useful matrix to represent the forward kinematics computation including velocities and accelerations of the multi-body system.

3.1. Definitions of basic geometric and mechanical variables

1. \( O_n \) and \( E_n \) are \( n \times n \) zero and identity matrices respectively. \( O_{n \times m} \) indicates a \( n \times m \) zero matrix.

2. The skew operator is represented as follows

\[
[x \times] \triangleq \begin{bmatrix}
0 & -x_3 & x_2 \\
x_3 & 0 & -x_1 \\
-x_2 & x_1 & 0
\end{bmatrix}
\]

3. The position and orientation matrix of the coordinate system of a rigid body are \( p \) and \( R \), respectively.

4. Let \( \omega \) and \( \nu \) be the angular and linear velocities represented by the local coordinates, respectively. The following relationship holds

\[
\dot{R} = R[\omega \times] \\
\nu \triangleq R^T \dot{p}
\]

5. The linear and angular velocities are concatenated and defined as in the following vector of spatial velocity

\[
\nu \triangleq \begin{bmatrix}
\nu \\
\omega
\end{bmatrix}
\]

6. The \( 6 \times 6 \) spatial transformation matrix for spatial velocities is defined as follows

\[
A(p, R) \triangleq \begin{bmatrix}
R & [p \times] R \\
O_3 & R
\end{bmatrix}
\]

7. Let us define the operator for the linear and angular velocities as follows

\[
[v \cdot] \triangleq \begin{bmatrix}
[\omega \times] & [\nu \times] \\
O_3 & [\omega \times]
\end{bmatrix}
\]

\[
v_1 \cdot v_2 \triangleq [v_1 \cdot] [v_2 \cdot]
\]

8. The above operator satisfies the binary operation axioms in Appendix A.1. The following important relationships also hold

\[
\dot{A} = A[v \cdot] \quad (4)
\]

\[
A[v \cdot] A^{-1} = [(A \nu) \cdot] \quad (5)
\]

9. The inertial properties of a rigid body consist of mass \( m \), center of mass \( \hat{c} \), and inertia tensor \( I_c \). They can be summarized by the following \( 6 \times 6 \) matrix

\[
M \triangleq \begin{bmatrix}
m E_3 & m [\hat{c} \times] \\
m [\hat{c} \times] & I_c + m [\hat{c} \times][\hat{c} \times]^T
\end{bmatrix}
\]

10. Let the inertial forces of a rigid body be \( \hat{f} \) and the moment around its coordinate be \( n \). They are represented in the global frame. Then, let us define a \( 6 \)-axis force \( f \) represented by the local coordinate as follows

\[
f \triangleq \begin{bmatrix}
R^T \hat{f} \\
R^T n
\end{bmatrix}
\]

11. The equations of motion of a rigid body are

\[
f = M \dot{\nu} - [v \cdot]^T M \nu
\]

A variation of the above equation is written as follows by using matrix \( D \)

\[
\delta f = M \delta \dot{\nu} - D \delta \nu \\
D \triangleq -([M \nu] \hat{c}) - [v \cdot]^T M
\]

12. Operation \([\cdot] \) is defined as follows

\[
[f \cdot] \triangleq \begin{bmatrix}
O_3 \\
[\hat{f} \times] \\
[\hat{f} \times] \\
\hat{f} \times n
\end{bmatrix}
\]

Note that the following relationship between \([\cdot] \) and \([\cdot] \) holds

\[
[\hat{f}_1 \cdot] [\hat{f}_2 \times] = [\hat{f}_2 \cdot] [\hat{f}_1 \times]
\]

3.2. Comprehensive motion transformation matrix (CMTM)

Let us define the following new \( 18 \times 18 \) matrix \( X \) and call it the CMTM

\[
X(A, v, \dot{v}) \triangleq \begin{bmatrix}
X_1 & O_6 & O_6 \\
X_2 & X_1 & O_6 \\
X_3 & X_2 & X_1
\end{bmatrix}
\]

\[
\triangleq \begin{bmatrix}
A & O_6 & O_6 \\
A & O_6 & O_6 \\
\frac{1}{2} A (\dot{v} \cdot) + [v \cdot]^T & A & O_6
\end{bmatrix}
\]

(9)
where variation $\delta x$ has the following relationship

$$[(\delta x)\circ] \triangleq A^{-1}(\delta A)$$

Vector $\delta x$ is the concatenated vector of the variation of the standard 6-dimensional coordinates, velocities, and accelerations.

To handle the differential operation of matrix $X$, the following variation of the 18-dimensional vector is newly defined as follows

$$\delta \xi \triangleq \begin{bmatrix} \delta \alpha \\ \delta \upsilon \\ \delta \eta \end{bmatrix}$$

(11)

where

$$\delta \xi = \delta \upsilon + [(\delta \alpha) \circ, \upsilon]$$

$$\delta \eta = \frac{1}{2} (\delta \upsilon + [(\delta \alpha) \circ, \upsilon] + [(\delta \zeta) \circ, \upsilon])$$

For clarity, we summarize the above equations as follows

$$\delta \xi = S \delta x$$

(12)

Matrix $S$ transforms variation $\delta x$ into a new vector $\delta \xi$, which is written as follows

$$S(\upsilon, \dot{\upsilon}) \triangleq \begin{bmatrix} E_6 & O_6 & O_6 \\ -\frac{1}{2} \left( [\upsilon \times] - [\upsilon \times]^2 \right) & E_6 & O_6 \\ -\frac{1}{2} [\upsilon \times] & \frac{1}{2} [\upsilon \times] & 2E_6 \end{bmatrix}$$

(13)

The inverse matrix of $S$ always exists, and is computed as follows

$$S^{-1} = \begin{bmatrix} E_6 & O_6 & O_6 \\ [\upsilon \times] & E_6 & O_6 \\ [\upsilon \times] & 2[\upsilon \times] & 2E_6 \end{bmatrix}$$

Although the variation $\delta x$ is what we are familiar with in robotic analysis, its usage makes the forthcoming analysis of Jacobian matrices tractable. By using the newly defined variation $\delta \xi$, the analysis becomes easier and clearer, as shown below. Once the Jacobian is derived, it can always be transformed back to $\delta x$ by via matrix $S$.

Let us now define the following matrix and operator

$$[\delta \xi ^*] \triangleq \begin{bmatrix} [\delta \alpha \circ] & O_6 & O_6 \\ [\delta \zeta \circ] & [\delta \alpha \circ] & O_6 \\ [\delta \eta \circ] & [\delta \zeta \circ] & [\delta \alpha \circ] \end{bmatrix}$$

$$\delta \xi_1 \circ \delta \xi_2 \triangleq [\delta \xi_1 ^*] \delta \xi_2$$

By utilizing the above operator, the following relationship holds

$$\delta X = X[(\delta \xi) \circ]$$

(14)

It can be verified by computing each block matrix of $\delta X$,

$$\begin{align*}
\delta X_1 &= \delta A = A[\delta \alpha \circ] = X_1[\delta \alpha \circ] \\
\delta X_2 &= \delta A[\upsilon \circ] + A[\delta \upsilon \circ] = X_1[\delta \zeta \circ] + X_2[\delta \alpha \circ] \\
\delta X_3 &= \frac{1}{2} \delta A (\dot{\upsilon} \circ) + [\upsilon \circ] (\delta \upsilon \circ) \\
&= X_1[\delta \eta \circ] + X_2[\delta \zeta \circ] + X_3[\delta \alpha \circ]
\end{align*}$$

Equation (14) has the same form as Equation (3) of Equation (4). Actually, operator $\delta \xi_1 \circ \delta \xi_2$ satisfies the binary operation axioms in Appendix A.1, which can be easily verified. In addition, the following equation also holds

$$X[\delta \xi ^*] X^{-1} = [(X \delta \xi) ^*]$$

(15)

Note that Equation (15) corresponds with Equation (5).

The set of matrix $X$ and operator $(\circ)$ has similar mathematical features as matrix $A$ and $(,)$ (i.e., the set of the adjoint map and Lie bracket operator). This means that many formulas of the kinematics operations on the position and orientation can be replaced with those operating on velocities and accelerations. Therefore, matrix $X$ can comprehensively handle the kinematics transformation for motion.

### 3.3. Definition and formulas of kinematics chain

This subsection presents the notation for open kinematic chain, and important formulas for the kinematics and dynamics.

1. The kinematic chain is tree-structured, and the indices are chosen from the base link toward the end of branches.
2. $p(i)$ is the index of a root-side link connected to link $i$.
3. $C(i)$ is the set of indices of leaves-side links connected to link $i$.
4. $P(i)$ is the set of all leaves-side links recursively connected to link $i$.
5. $\hat{P}(i)$ is the set of all root-side links recursively connected to link $i$.
6. Let us define the following sets $\hat{P}(i) \triangleq \{i, P(i)\}, \hat{C}(i) \triangleq \{i, C(i)\}$.
7. $s_{ij}$ is the following selection function

$$s_{ij} \triangleq \begin{cases} 1 & (i \in \hat{P}(j)) \\ 0 & (\text{others}) \end{cases}$$

8. Let us represent the quantity of link $i$ such that $y_i$.
9. Let us denote $y_j$ as the relative variable of $y$ from link $i$ to $j$.
10. $q_i$ is the $n_i$ sets of joint variables (angles), where $n_i$ is the number of DOFs of joint $i$, and the following relationship holds between the joint variables and the relative velocities between link $i$ and $p(i)$

$$A_i^{p(i)} = e^{(K, q_i) \circ}$$

(16)
11. Matrix $K_i$ is a $6 \times n_j$ constant matrix defined according to the type of joint $i$. For instance, if joint $i_1$ is a rotational joint, joint $i_2$ is a translational one, joint $i_3$ is a spherical one and joint $i_4$ is a free floating one (6-DOF), the corresponding matrices are as follows

\[
K_{i_1} \triangleq \begin{bmatrix} 0_3^T & e_{i_1}^T \end{bmatrix}^T \quad (rotational)
\]
\[
K_{i_2} \triangleq \begin{bmatrix} e_{i_2}^T & 0_3^T \end{bmatrix}^T \quad (translational)
\]
\[
K_{i_3} \triangleq \begin{bmatrix} O_3 \end{bmatrix} \quad (spherical)
\]
\[
K_{i_4} \triangleq E_6 \quad (freefloating)
\]

where $e_{i_1}$ and $e_{i_2}$ mean the corresponding joint axis direction and $0_3$ is a 3-dimensional zero vector.

12. Vector $\delta \theta_i$ is a variance defined in the tangent vector space of $A_{i}^{0(i)}$, which has the following differential relationship

\[
\delta A_{i}^{0(i)} = A_{i}^{0(i)}[\delta \alpha_{i}^{0(i)} \cdot j] = A_{i}^{0(i)}(K_i \delta \theta_i) \cdot j \quad (17)
\]

Note that, in the case of spherical joints or free floating joints, the tangent vector $\delta \theta_i$ is not equal to the variation of joint variable $\delta q_i$ due to Equation (17); for example, the angular velocity of a spherical joint is not equal to the derivative of the angle-axis vector representing the joint orientation.

13. Vector $\dot{\psi}_i$ represents the joint velocity variables, and the following equations hold between $\dot{\psi}_i$ and the relative coordinates

\[
\dot{v}_i^{0(i)} = K_i \dot{\psi}_i \quad (18)
\]

14. Vector $f_{i}^{0(i)}$ denotes the constraint force of joint $j$, which has the following relationship with inertial force $f_j$ of link $j$ and its links connected to the leaves-side

\[
f_j^{0(i)} = f_j + \sum_{k \in C(j)} A_{j}^{i-1} f_k^{0(i)} \quad (19)
\]

where $j = p(k)$ holds due to $k \in C(j)$. The above recursive formula can be transformed into the following summation formula

\[
f_j^{0(i)} = \sum_{k \in C(j)} A_{j}^{i-1} f_k \quad (20)
\]

15. Vector $\tau_j$ represents the $n_j$ dimensional vector of joint torque and can be extracted from the 6-dimensional vector of joint constraint forces $f_{j}^{0(i)}$ as follows

\[
\tau_j = K_j^{T} f_{j}^{0(i)} \quad (21)
\]

### 3.4. CMTMs in the kinematic chain

Let us consider the following chain product of CMTMs

\[
X_j = X_i X_j^i \quad (22)
\]
4. Computation of arbitrary Jacobians

This section shows the different types of arbitrary Jacobian matrices used in Equation (2) by utilizing CMTM. The formulation in this paper are strictly speaking not Jacobian matrices as in the case of the basic Jacobian. The basic Jacobian is the coefficient matrix in the linear differential relationship between the joint angle velocities and the linear/angular velocities of the corresponding link. Since the integration of angular velocity has no physical meaning, the corresponding part of the basic Jacobian is not equivalent to the partial derivatives of the orientation variable. Several Jacobians introduced in this section also mean the coefficient matrices in the linear differential relationship between the variation of joint variables $\delta \chi$ and the variation of arbitrary physical quantities $\delta \psi$. However, in this paper, we will refer to them as Jacobians for descriptive purposes.

4.1. Jacobians of link posture, velocity and acceleration

Let us compute matrix $J_j$ that converts the variation $\delta \chi_{all}$ of all joints to variation $\delta \chi$ for link $j$ as follows

$$\delta \chi_j = J_j \delta \chi_{all} = \sum_k J_{(j,k)} \delta \chi_k$$

(31)

where, $J_{(j,k)}$ is the block matrix of $J_j$ related to joint $k$.

The matrix $J_{(j,k)}$ can be directly computed as follows

$$J_{(j,k)} = S_{(j,k)} \tilde{X}_k G_k$$

(32)

where

$$\tilde{X}_k \triangleq S_j^{-1} X_j S_j^{(k)}$$

(33)

This subsection contains the proof that Equation (32) holds. The Jacobian $J_j$ is analogous to the basic Jacobian (Khatib, 1987) as shown in Figure 2.

As mentioned in the previous section, matrix $X$ has the same features as $A$. Matrix $J_j$ can be computed in a similar manner when computing basic Jacobians as the following proof.

**Proof:** Let us consider $X_j$ of link $j$. The following chain products hold among CMTMs

$$X_j = X_{\hat{P}(j)} X^{(k)}_k X^{(k)}_j \quad (k \in \hat{P}(j))$$

Then, the variation of $X_j$ can be computed according to the above chain products, and we have the following

$$\delta X_j = \sum_{k \in \hat{P}(j)} X_{\hat{P}(j)} \delta X^{(k)}_k X^{(k)}_j$$

where $\delta X^{(k)}_k$ is the variation of the $k$th link.
By utilizing Equation (14) and Equation (15), the above equation can be transformed into
\[ X_j[(\delta \xi_j)^\ast] = \sum_{k \in P(j)} X_{p(k)} X_k \delta \xi_k^{p(k)} \] 
\[ = X_j \sum_{k \in P(j)} [(X_j^T \delta \xi_k^{p(k)})^\ast] \]

According to the above equation, the following equation also holds
\[ \delta \xi_j = \sum_{k \in P(j)} X_k^T \delta \xi_k^{p(k)} \tag{34} \]

The coefficient matrix of Equation (34) represents the Jacobian matrix with respect to \( \delta \xi \), and each block matrix is equal to relative CMTM \( \xi_k^{p(k)} \).

Because the desired Jacobian matrix is with respect to \( \delta x \), let us compute it by transforming variations of Equation (34) from \( \delta \xi \) to \( \delta x \) and from \( \delta x \) to \( \delta y \).

First, by utilizing Equation (12), the following equation is obtained
\[ \delta x_j = \sum_{k \in P(j)} S_j^{-1} X_k^T \delta \xi_k^{p(k)} \] 
\[ = \sum_{k} S_{j(k)} X_k \delta \xi_k^{p(k)} \tag{35} \]

The next transformation can be performed with Equation (30) as follows
\[ \delta x_j = \sum_{k} S_{j(k)} X_k \delta x_k \]

From Equation (35), the Jacobian matrix shown in Equation (31) was finally derived as given in Equation (32).

The computation of \( J_j \) in Equation (31) requires \( J_{(j,k)} \) for all \( k \). Each \( J_{(j,k)} \) can be obtained by the matrix products according to Equation (32) and Equation (33). Therefore, the computational complexity of \( J_j \) is \( O(N_j) \). Because the direct computation of 18 x 18 matrix products in Equation (33) is computationally inefficient, the solution of 6 x 6 block matrices in \( X_j \) is given in Appendix A.2.

4.2. Jacobians of link inertial forces

This subsection derives matrix \( L_j \) that converts variation \( \delta \chi \) of all joints to force variation \( \delta f_j \) of link \( j \) as follows
\[ \delta f_j = L_j \delta \chi = \sum_{k} L_{(j,k)} \delta \chi_k \tag{36} \]

where, \( L_{(j,k)} \) is the block matrix of \( L_j \) related to joint \( k \).

The matrix \( L_{(j,k)} \) can be computed as follows
\[ L_{(j,k)} = H_j J_{(j,k)} \tag{37} \]

where
\[ H_j \triangleq \begin{bmatrix} O_6 & D_j & M_j \end{bmatrix} \]

This subsection verifies Equation (37).

**Proof:**

Let us first consider the variation of equations of motion of link \( j \). According to Equation (7), the following equations are obtained
\[ \delta f_j = H_j \delta x_j \]

From Equation (31), the above equation can be also transformed into the following
\[ \delta f_j = \sum_{k} H_k J_{(j,k)} \delta x_k \tag{38} \]

Therefore, the Jacobian matrix shown in Equation (36) could be derived as Equation (37).

For the sake of the subsequent discussions, let us transform Equation (38) to a linear form with respect to \( \delta \xi_j \) by using Equation (12)
\[ \delta f_j = H_j S_j^{-1} \delta \xi_j \tag{39} \]

4.3. Jacobians of joint constraint forces

In this subsection, we compute matrix \( N_j \) that converts the variations \( \delta \chi_{\text{all}} \) of all joints to force variation \( \delta \chi_j^{p(0)} \) of link \( j \) as follows
\[ \delta \chi_j^{p(0)} = N_j \delta \chi_{\text{all}} = \sum_{k} L_{(j,k)} \delta \chi_k \tag{40} \]

where, \( N_{(j,k)} \) is the block matrix of \( N_j \) related to joint \( k \).

The matrix \( N_{(j,k)} \) can be computed as follows
\[ N_{(j,k)} = \begin{bmatrix} \hat{H}_j J_{(j,k)} & \begin{bmatrix} A_k^{-T} \hat{H}_k J_{(j,k)} - [J_k^{(p(k)), T} T G_k] \end{bmatrix} \end{bmatrix} \tag{41} \]

where, matrix \( \hat{H}_j \) can be recursively computed from leafside links as follows
\[ \hat{H}_j = H_j + \sum_{k \in C(j)} A_k^{-T} \hat{H}_k S_k^{-1} X_k S_j \tag{42} \]

Additionally, matrix \( T \) is defined as follows
\[ T \triangleq [E_6 \ O_6 \ O_6] \]

Let us verify Equation (41) in this subsection.
Proof: From Equation (19), the following equation about the joint constraint force \( f_j^{(i)} \) can be obtained
\[
\delta f_j^{(i)} = \delta f_j + \sum_{k \in \mathcal{C}(j)} (A_k^{(i)} - T \delta f_k^{(i)} + \delta A_k^{(i)} - T f_k^{(i)})
\]
According to Equation (39), the above equation can be transformed into
\[
\delta f_j^{(i)} = (H_j S_j^{-1}) \delta \xi_j
+ \sum_{k \in \mathcal{C}(j)} (A_k^{(i)} - T \delta f_k^{(i)} + \delta A_k^{(i)} - T f_k^{(i)}) \tag{43}
\]
On the other hand, the following equation holds from Equation (34)
\[
\delta \xi_j = X_p^{(j)} \left( \sum_{k \in \mathcal{P}(j)} X_k^{(i)} \delta \xi_k^{(i)} \right) + \delta \xi_j^{(i)} \tag{44}
\]
Because \( A^{-T} \) and \( X \) are transformation matrices, Equation (44) and Equation (43) have the same form as Equation (71) and Equation (72) given in Appendix A.3, respectively. According to the similar derivations of Equation (73), Equation (74), Equation (75) in Appendix A.3, the following recursive formula can be obtained
\[
\delta f_j^{(i)} = \tilde{H}_j \delta \chi_j + \tilde{h}_j \tag{45}
\]
where
\[
\tilde{H}_j = \sum_{k \in \mathcal{C}(j)} (A_k^{(i)} - T \tilde{H}_k S_k^{-1} \delta \xi_k^{(i)})
+ \delta A_k^{(i)} - T f_k^{(i)} + A_k^{(i)} - T \tilde{h}_k \tag{46}
\]
It should be noted that, if the set of \( A^{-T} \) and \( X \) is replaced with \( A^{-T} \) and \( A \) and if there exists no bias terms such as \( \delta \xi_j^{(i)} \), the transformation based on Appendix A.3 has the same formula when introducing the linear and angular momentum Jacobian.

Let us expand the term of \( A_k^{(i)} - T \tilde{h}_k \) in Equation (46) by its recursive computation from the base-link toward the end-links. In addition, by using the transformation from \( \delta \xi \) to \( \delta x \) with Equation (12), Equation (46) can be transformed into
\[
\tilde{h}_j \sum_{k \in \mathcal{C}(j)} (A_k^{(i)} - T \tilde{H}_k S_k^{-1} \delta \xi_k^{(i)})
+ \delta A_k^{(i)} - T f_k^{(i)} + A_k^{(i)} - T \tilde{h}_k \tag{46}
\]
There also exists the following conversion of variation \( \delta A_k^{(i)} - T f_k^{(i)} \)
\[
\delta A_k^{(i)} - T f_k^{(i)} = -A_k^{(i)} - T [\delta \alpha_k^{(i)} \cdot J_k] f_k^{(i)}
= -A_k^{(i)} - T [f_k^{(i)} \cdot \delta \alpha_k^{(i)}]
\]
According to the above equation, \( \tilde{h}_j \) can be written as:
\[
\tilde{h}_j = \sum_{k \in \mathcal{C}(j)} \left( A_k^{(i)} - T \tilde{H}_k \tilde{X}_{(k)} - A_k^{(i)} - T \left[ f_k^{(i)} \cdot \tilde{z} \right] T \right) \delta \chi_k^{(i)} \tag{47}
\]
Note that, \( \tilde{X}_{(k)} = S_k^{-1} \chi_k^{(i)} \) is from Equation (33).

By substituting Equation (42) and Equation (47) for Equation (45), and by converting the variations from \( \delta \chi_k^{(i)} \) to \( \delta \chi_k^{(i)} \), the following equation holds
\[
\delta f_j^{(i)} = \sum_{k \in \mathcal{P}(j)} \tilde{H}_j \tilde{X}_{(j,k)} \delta \chi_k^{(i)}
+ \sum_{k \in \mathcal{C}(j)} A_k^{(i)} - T \left( \tilde{H}_j \tilde{X}_{(j,k)} - [f_k^{(i)} \cdot \tilde{z}] T G_k \right) \delta \chi_k^{(i)} \tag{48}
\]
From Equation (48), the Jacobian matrix shown in Equation (40) can be finally derived as given in Equation (41).

\[\Box\]

The important advancement from the conventional formulation (Suleiman et al., 2008) is the recursive formula of inertial matrices in Equation (42), which achieves significant improvement in computational efficiency. The computation of Equation (41) requires matrix \( \tilde{H}_j \). It means that \( \tilde{H}_j \) for all link \( j \) needs to be computed in advance according to recursive formula Equation (42). This computational complexity to update \( \tilde{H}_j \) for all links is \( O(N_j) \). After updating \( \tilde{H}_j \), matrix \( N_{(j,k)} \) for all \( j \) and \( k \) can be directly computed by Equation (41). The computation of \( N_j \) requires \( N_{(j,k)} \) for all \( k \). Therefore, the computational complexity of \( N_j \) including recursive computation of Equation (42) is \( O(N_j) \).

The direct computation of Equation (42) with 18 × 18 matrices is computationally inefficient. The final form of the 6 × 18 matrix \( \tilde{H}_j \) derived from Equation (42) is written down in Appendix A.4. The matrix \( \tilde{H}_j \) contains several physical quantities such as the total mass, the center of total mass, the total inertia tensor, the total linear/angular momentum, etc. This feature of the matrix is also detailed in Appendix A.4.

4.4. Jacobians of joint torques

Let us compute matrix \( \tilde{N}_j \) which converts variation \( \delta \chi \) of all joints to joint torque variation \( \delta \tau_j \) of joint \( j \) as follows
\[
\delta \tau_j = \tilde{N}_j \delta \chi_k = \sum_k \tilde{N}_{(j,k)} \delta \chi_k \tag{49}
\]
where, \( \tilde{N}_{(j,k)} \) is the block matrix of \( \tilde{N}_j \) related to joint \( k \).

The variance of the joint torque \( \delta \tau_j \) can be obtained from Equation (21) as follows
\[
\delta \tau_j = K_j^T \delta f_j^{(i)}
\]
Therefore, \( \tilde{N}_{(j,k)} \) can be easily obtained by using Equation (40)
\[
\tilde{N}_{(j,k)} = K_j^T N_{(j,k)} \tag{50}
\]
4.5. An application: Jacobian of ZMP

Since ZMP is often used for the analysis of the balancing problem of humanoid systems, its Jacobian matrix will be useful in the motion optimization framework. This subsection presents the method of computing the Jacobian matrix of ZMP.

The total external forces acting on the kinematic chain are equivalent to \( F^{W}_{\text{ex}} \), where index \( W \) means the world coordinate. Note that the floating base-link \( \theta \) is connected to the world coordinate via a 6-DOF free joint. By redefining \( F_{\text{ex}}^{j} \triangleq F^{W}_{\text{ex}} \), we can obtain ZMP projected, for example, on the \( x-y \) plane

\[
P_{\text{ZMP}} \triangleq \begin{bmatrix} -F_{\text{ex}(5)}/F_{\text{ex}(3)} \\ F_{\text{ex}(4)}/F_{\text{ex}(3)} \\ 0 \end{bmatrix}
\]  

(51)

where

\[
F_{\text{ex}} \triangleq \begin{bmatrix} F_{\text{ex}(1)} \\ \vdots \\ F_{\text{ex}(6)} \end{bmatrix}
\]

Let us compute matrix \( Z \) which converts variation \( \delta x_{\text{all}} \) of all joints to ZMP variation \( \delta p_{\text{ZMP}} \) as follows

\[
\delta p_{\text{ZMP}} = Z \delta x_{\text{all}} = \sum_{k} Z_{(k)} \delta x_{k}
\]

(52)

where, \( Z_{(k)} \) is the block matrix of \( Z \) related to joint \( k \).

The variance of Equation (51) can be formulated as follows

\[
\delta p_{\text{ZMP}} = \hat{Z} \delta F_{\text{ex}} = (Z \delta f^{W}_{\text{ex}})
\]

where

\[
\hat{Z} \triangleq \begin{bmatrix} 0 & 0 & F_{\text{ex}(5)}/F_{\text{ex}(3)} & 0 & -1/F_{\text{ex}(3)} & 0 \\ 0 & 0 & -F_{\text{ex}(4)}/F_{\text{ex}(3)} & 1/F_{\text{ex}(3)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\]

Therefore, the Jacobian \( Z_{(k)} \) can be computed as

\[
Z_{(k)} = \hat{Z} N_{(W,k)}
\]

(53)

5. Advanced implementation with decomposed gradient computation

Let us consider the following function

\[
h(x_{1}, x_{2}, ..., f_{1}, f_{2}, ..., f^{(1)}_{1}, f^{(2)}_{2}, ...)
\]

This function corresponds to the cost function or each function of the corresponding constraint in Equation (1). The function contains several physical quantities such as link coordinates, link inertial forces, joint forces, etc. The gradient of the function with respect to each physical quantities is as follows

\[
\frac{\partial h}{\partial x_{1}}, ..., \frac{\partial h}{\partial f_{1}}, ..., \frac{\partial h}{\partial f^{(1)}_{1}}, ...
\]

When solving the optimization problem such as Equation (1), the gradient of the function with respect to the joint variables \( x_{\text{all}} \) is needed to compute Equation (2). By using the corresponding Jacobian matrices, the gradient can be computed as

\[
\frac{\partial h}{\partial x_{\text{all}}} = \frac{\partial h}{\partial J_{1}} + ... + \frac{\partial h}{\partial f_{1}} L_{1} + ... + \frac{\partial h}{\partial f^{(1)}_{1}} N_{1} + ... \quad (54)
\]

In many cases, the cost function or each constraint function has a simple structure with respect to each physical quantity. A typical example for cost functions is the summation of the quadratic form of each physical quantity such as

\[
h = \frac{\omega_{1}}{2} ||x_{1} - x_{1}^{\text{ref}}||^{2} + ...
\]

\[
+ \frac{\omega_{2}}{2} ||f_{1} - f_{1}^{\text{ref}}||^{2} + ...
\]

\[
+ \frac{\omega_{N}}{2} ||f^{(1)}_{1} - f^{(1)}_{1}||^{2} + ...
\]

where, \( \omega_{x_{j}} \) represents a weighing factor. In such a case, the gradients \( \partial h/\partial x_{j} \), \( \partial h/\partial f_{j} \), and \( \partial h/\partial f^{(1)}_{j} \) for all \( j \) can be easily computed with computational complexity \( O(N_{j}) \). However, if \( \omega_{x_{j}} \neq 0 \) for all \( * \) and \( j \), the computation of Equation (54) requires \( N_{j} \) times of computation of the Jacobian matrices for each type of physical quantities. This leads the computational complexity of Equation (54) is \( O(N_{j}^{2}) \).

This section introduces an efficient computation method of Equation (54) with computational complexity \( O(N_{j}) \) by avoiding the direct computation of the Jacobian matrices. The fast gradient computation is useful when solving a large-scale nonlinear optimization for a humanoid robot or a human, as mentioned in Ayusawa and Nakamura (2012). The computational cost of each iterative computation during optimization can be dramatically reduced by the combination of the fast gradient computation and the fast direction search algorithms such as the conjugate gradient method, the limited-memory quasi-Newton method, etc (Fletcher, 1987).

5.1. Gradient computation of arbitrary functions of \( x_{j} \)

Let us consider an arbitrary function \( h \) whose variables are link coordinates \( x_{j} \) for all \( j \). This subsection introduces the computation method of the gradient \( \partial h/\partial x_{\text{all}} \), if the gradient \( \partial h/\partial x_{j} \) for all \( j \) is given.

The simple formulation when computing \( \partial h/\partial x_{\text{all}} \) can be written by using Equation (31) as follows

\[
\frac{\partial}{\partial x_{\text{all}}} h(x_{1}, x_{2}, ...) = \sum_{j} \frac{\partial h}{\partial x_{j}} \frac{\partial x_{j}}{\partial x_{\text{all}}} = \sum_{j} \frac{\partial h}{\partial x_{j}} J_{j}
\]

(55)
This subsection introduces an efficient method of computing Equation (55).

The first step computes the gradient $\partial h / \partial \xi_i^{(0)}$, which can be decomposed as follows

$$\frac{\partial h}{\partial \xi_i^{(0)}} = \sum_k \frac{\partial h}{\partial \xi_k} \frac{\partial \xi_k}{\partial \xi_i^{(0)}}$$

By using Equation (34), we can further have

$$\frac{\partial h}{\partial \xi_i^{(0)}} = \sum_k \frac{\partial h}{\partial \xi_k} X_k^{(i)}$$

The above summation formulas can be transformed into the recursive formula as follows

$$\frac{\partial h}{\partial \xi_i^{(0)}} = \frac{\partial h}{\partial \xi_i} + \sum_{k \in C(i)} \frac{\partial h}{\partial \xi_k} X_k^{(i)}$$

Equation (56) indicates that, if the gradient $\partial h / \partial x_i$ for all $i$ is given, the gradient $\partial h / \partial x_i^{(0)}$ can be computed recursively from the end-links.

Finally, let us convert the variable $\xi$ into $x$. The following relationship can be obtained according to Equation (12).

$$\frac{\partial \xi_i}{\partial x_i} = S_i$$

By using the above relationships, Equation (56) can be transformed into the following

$$\frac{\partial h}{\partial x_i} = \frac{\partial h}{\partial x_i} S_i + \sum_{k \in C(i)} \frac{\partial h}{\partial x_k} X_k^{(i)}$$

Equation (57) indicates that, if the gradient $\partial h / \partial x_i$ for all $i$ is given, the gradient $\partial h / \partial x_i^{(0)}$ can be computed recursively from the end-links.

5.2. Gradient computation of arbitrary functions of $f_j$

Let us consider an arbitrary function $h$ whose variables are link inertial forces $f_j$, for all $j$. This subsection introduces an efficient computation method of the gradient $\partial h / \partial x_{ai}$, if the gradient $\partial h / \partial f_j$, for all $j$ is given.

The simple formulation when computing $\partial h / \partial x_{ai}$ can be written according to Equation (36) as follows

$$\frac{\partial h}{\partial x_{ai}} = \sum_j \frac{\partial h}{\partial f_j} \frac{\partial f_j}{\partial x_{ai}}$$

This subsection presents a method of computing Equation (59).

First, let us consider the gradient $\partial h / \partial x_i$. It can be decomposed as follows

$$\frac{\partial h}{\partial x_i} = \sum_k \frac{\partial h}{\partial f_k} \frac{\partial f_k}{\partial x_i}$$

From Equation (38), we can have

$$\frac{\partial f_j}{\partial x_i} = H_j$$

Finally, the above equations can be transformed into

$$\frac{\partial h}{\partial x_i} = \frac{\partial h}{\partial f_j} H_j$$

Equation (60) provides $\partial h / \partial x_i$, the gradient $\partial h / \partial x_{ai}$ can be computed in the same manner when computing Equation (55) in the previous section.

5.3. Gradient computation of arbitrary functions of $f_j^{(0)}$

Let us consider an arbitrary function $h$ whose variables are joint constraint forces $f_j^{(0)}$, for all $j$. This subsection introduces an efficient method to compute the gradient $\partial h / \partial x_{ai}$, if the gradient $\partial h / \partial f_j^{(0)}$, for all $j$ is given.

The simple formulation when computing $\partial h / \partial x_{ai}$ can be written by using Equation (40) as follows

$$\frac{\partial h}{\partial x_{ai}} = \sum_j \frac{\partial h}{\partial f_j^{(0)}}$$

This subsection introduces its efficient computation.

First, let us formulate the variation of $f_j^{(0)}$ by using Equation (20) as follows

$$\delta f_j^{(0)} = \sum_{k \in C(i)} (A_i^{-T} f_k + \delta (A_i^{-T} f_k))$$

From Equation (38), we can have

$$\frac{\partial h}{\partial f_j^{(0)}} = \frac{\partial h}{\partial f_j}$$

Finally, the above equations can be transformed into

$$\frac{\partial h}{\partial x_{ai}} = \frac{\partial h}{\partial f_j} H_j$$

Since Equation (60) provides $\partial h / \partial x_i$, the gradient $\partial h / \partial x_{ai}$ can be computed in the same manner when computing Equation (55) in the previous section.
The matrix variation $\delta (A_k^{-1})$ can be obtained from the relationship $A_k^{-1} A_k^{-1} = E$, and we can have
\[
\delta (A_k^{-1}) = \left[A_k^{-1} (\delta A'_k) A_k^{-1}\right] = \left[A_k^{-1} (\delta A'_k \ast) A_k^{-1}\right] = -[\delta A'_k \ast] A_k^{-1}
\]
Therefore, the term $\delta (A_k^{-T}) f_k$ can be transformed as follows
\[
\delta (A_k^{-T}) f_k = -A_k^{-T} [\delta A'_k \ast] A_k^{-1} = -A_k^{-T} [f_k \ast] \delta A'_k
\]
By substituting $\delta A'_k = \delta A_k - A_k^{-1} \delta \alpha_j$ into the above equation, we have
\[
\delta (A_k^{-T}) f_k = -A_k^{-T} [f_k \ast] \delta A_k + A_k^{-T} [f_k \ast] A_k^{-1} \delta \alpha_j
\]
By using this relationship, Equation (62) can be transformed into
\[
\delta f_j^{(p)} = \sum_{k \in \mathcal{C}(j)} A_k^{-T} (\delta f_k - [f_k \ast] \delta A_k) + \left( \sum_{k \in \mathcal{C}(j)} A_k^{-1} f_k \right) \delta \alpha_j
\]
By substituting Equation (40) and $\delta \alpha_s = T \delta x_s$ into the above, we have
\[
\delta f_j^{(p)} = \sum_{k \in \mathcal{C}(j)} A_k^{-T} (\delta f_k - [f_k \ast] T \delta x_k) + [f_j \ast] T \delta x_j
\]
When the function contains the joint torque Jacobian $\partial h/\partial x_{\text{all}}$, the gradient can be computed from the following recursive form
\[
d_k = \left( \frac{\partial h}{\partial f^{(p)}_j} \right)^T + \sum_{j \in \mathcal{C}(k)} A_k^{-1} d_j
\]
Finally, the gradient $\partial h/\partial x_k$ can be computed from the following steps.

1. Compute Equation (66) for all $k$ from the end-links.
2. Compute Equation (65) for all $k$.

Since $\partial h/\partial x_k$ for all $k$ is provided, the gradient $\partial h/\partial x_{\text{all}}$ can be computed in the same manner when computing Equation (55).

### 5.4. Gradient computation of other functions

The joint torque $\tau_j$ can be represented by $f_j^{(p)}$ according to Equation (21). Therefore, the gradient of an arbitrary function $h$ of the joint torque $\tau_j$ with respect to $f_j^{(p)}$ can be decomposed into
\[
\frac{\partial h}{\partial f_j^{(p)}} = \frac{\partial h}{\partial \tau_j} \frac{\partial \tau_j}{\partial f_j^{(p)}} = \frac{\partial h}{\partial \tau_j} K_j
\]
when the function contains the joint torque $\tau_j$, the gradient can be computed from Equation (61) and Equation (67).

Similar to the case of the joint torque, ZMP $p_{ZMP}$ can be represented by $f_w^{(p)}$ according to Equation (51). The gradient of an arbitrary function $h$ of $p_{ZMP}$ with respect to $f_w^{(p)}$ can be decomposed into
\[
\frac{\partial h}{\partial f_w^{(p)}} = \frac{\partial h}{\partial p_{ZMP}} \frac{\partial p_{ZMP}}{\partial f_w^{(p)}} = \frac{\partial h}{\partial p_{ZMP}} \hat{Z}
\]
When the function contains $p_{ZMP}$, the gradient can be computed from Equation (61) and Equation (68).
6. Numerical evaluation

6.1. Comparison of computation time of Jacobian matrices

We here show the comparison of the computation times of the Jacobian matrices for the three approaches: the proposed method shown in Section 4, the traditional method in Suleiman et al. (2008), and the automatic differentiation method. Since the formulations in Suleiman et al. (2008) only handle 1-DOF joints, it was tested by using a serial manipulator with $N$ rotational joints. The Jacobian matrix of the joint torque of the first rotational joint was computed by changing the number of joints. We implemented the automatic differentiation of the joint torque computation by using Adept (Hogan, 2014): a combined automatic differentiation and array library for C++. The methods were tested on the computer with Intel(R) Xeon(R) CPU E3-1535M v5.

The proposed method generated the same Jacobian matrices as those from the other two methods. Figure 3 shows the results of the computational time of the two methods, demonstrating its correctness.

The computational complexity of the conventional method was $O(N^2)$ and that of the proposed method was $O(N)$. The computational time was significantly improved in the large-DOF cases. Though the computational complexity of the automatic differentiation method is also $O(N)$, the computational time of each DOF case was about 5 times higher than that of the proposed method. Since the proposed method is based on the efficient recursive formula, it shows better computational performance than the automatic differentiation method.

6.2. Comparison of computation time of gradient

This subsection evaluates the performance of the proposed method of computing the gradient of the cost function. We tested the method by using the same serial manipulator model used in the previous subsection. The following two cases were considered.

(A) The cost function contains the joint torque of only one joint: $c = ||\tau_1||^2$

(B) The cost function contains the joint torque for all joints:
$$c = \sum_j ||\tau_j||^2$$

We computed $\partial c / \partial x_{ij}$ for the two cases by the two approaches: the proposed decomposed gradient computation (DGC) shown in Section 5 and the normal approach according to Equation (54) without DGC. In the normal approach, the Jacobian matrices were computed by the method shown in Section 4.

The comparison of computation times in case (A) is shown in Figure 4, and that in case (B) is in Figure 5. The solid lines represent the computation time with DGC, and the dotted lines indicate the time without DGC. The gradient computation without DGC in case (A) requires the Jacobian matrix of the joint torque of only one joint. This leads that the computational complexity without DGC is also $O(N_j)$, which can be seen from the dotted line in Figure 4. The improvement of computational speed by DGC is not significant for small number of evaluated quantities. However, the gradient computation without DGC in case (B) needs the Jacobians for all the joints, and the computational complexity becomes $O(N_j^2)$, as shown in Figure 5. In the both cases, the computational complexity was $O(N_j)$ when computing the gradient with DGC. When the number of evaluated quantities in the cost function increases, the proposed method significantly reduced the computational time of the gradient computation.

6.3. Motion optimization of spherical joint manipulator

The proposed method is applied to a redundant serial robot manipulator composed of five spherical joints, in order to validate the Jacobian for the case of spherical joints. The 15-DOF manipulator moved in a complex environment cluttered with non-convex obstacles and no gravity. Each
Fig. 6. Snapshots of the generated motion of the redundant robot in a cluttered environment. Starting from the initial position \([0\ 0\ 1]^T\), the target end-effector positions at \([0\ 1\ 0]^T\) (\(t = 1.0\ s\)) and \([1\ 0\ 0]^T\) (\(t = 2.0\ s\)) were achieved without collision to obstacles.
Fig. 7. Resultant joint torque trajectories of the base spherical joint. The torque limit of small value of ±0.1 Nm is satisfied throughout the motion.

Fig. 8. Resultant joint torque trajectories of all the joints except for the base joint. All the 12 lines are within the torque limit of ±10 Nm throughout the motion.

The optimal trajectory for dynamic parameter identification is called the persistent exciting (PE) trajectory (Gautier and Khalil, 1992). To generate the PE trajectory, the condition number of the “regressor matrix” obtained from the joint trajectory should be minimized while maintaining dynamic constraints. Although we showed an analytical framework to optimize the condition number in our previous work (Ayusawa et al., 2017), the stability was considered only statically by using the center of mass (CoM), which is conservative and may limit the identification performance. In the resultant motions, both feet were placed to the ground because the dynamic stability condition became too severe to be satisfied only by the CoM condition. This limitation makes it difficult to generate dynamic leg motions by standing on one leg for better identification. In the proposed framework, the analytical computation of the Jacobian of the ZMP can be provided, which can guarantee the dynamic stability constraint in the trajectory optimization.

We have derived a dynamically stable optimal PE trajectory on one leg by constraining the ZMP inside the area of 4 cm and 1 cm around the center of the standing foot in front and lateral direction. The trajectory is parameterized by B-Spline using physical properties from the robot CAD model. We also added constraints on forces and torque applied on the ankle so that the horizontal forces $F_{\text{ex}}(1), F_{\text{ex}}(2)$ and torque $F_{\text{ex}}(6)$ stay within ±20 N, ±20 N, ±4 Nm respectively to avoid slipping. To prevent the jumping of the robot, we also set the limitation on the vertical force $F_{\text{ex}}(2)$ such that $|F_{\text{ex}}(2) - M_r g| \leq 50$ N, where $M_r$ is the total mass of the robot. Similar to the example in the previous section, all equality and inequality constraints were treated as penalty functions by the penalty-function method. Then, the computation was performed by the proposed decomposed gradient computation.

As shown in the snapshots in Figure 9, the humanoid HRP-4 (Kaneko et al., 2011) successfully performed the optimized PE trajectory on dynamic simulator Choreonoid (Nakaoka, 2012), which validates the feasibility. The total computation time without DGC was 12826 s. DGC accelerated the optimization so that its computation time was 2857 s. The result of the optimized condition number was 124.2, small enough to indicate that a dynamically stable optimal trajectory for dynamics identification of the robot were generated successfully by using the proposed framework.

7. Conclusion

This paper presented a comprehensive theory of differential kinematics and dynamics to derive analytical partial derivatives of both link/joint quantities with respect to joint coordinates and their derivatives. First, the $18 \times 18$ comprehensive motion transformation matrix (CMTM) and 18-dimensional product operation was introduced for comprehensive kinematics formulation, which allows a simple chain product of the matrices to represent the differential
Fig. 9. Snapshots of optimized one-leg PE trajectory for dynamic parameter identification. The humanoid robot HRP-4 can successfully perform resultant whole-body motion by also exciting the free leg.

forward kinematics of a kinematics chain. They also have the same features as the rotation matrix and the $6 \times 6$ transformation matrix, and their product operations. By utilizing CMTM, the partial derivative of the link coordinates and their derivatives were derived in the same manner as when introducing the basic Jacobians by just replacing the $6 \times 6$ transformation matrices with CMTM in their formulations.

This novel theoretical framework added the following contributions to current work. The partial derivatives of each generalized force with respect to the joint coordinates and their derivatives were also demonstrated. The derivation procedure also has a similarity to that of the linear and angular momentum Jacobians. The formulation can also handle different types of joints such as spherical joints or free-floating bases. We have also shown that the analytical derivative of physical quantities like ZMP can be easily computed with the proposed method, which is an important advantage for practical human/humanoid motion optimization. By utilizing the CMTM, each new Jacobian could be computed with $\mathcal{O}(N_J)$, which was verified by the comparison of computational times from the proposed and conventional methods. The proposed method was also compared to the automatic differentiation and showed better computational performance thanks to the efficient recursive formula of the method.

The optimization problem often can be solved efficiently by avoiding the direct computation of Jacobian matrices, as indicated in Ayusawa and Nakamura (2012). The fast gradient computation algorithms were also proposed to lead to more practical implementation. Evaluation of cost function composed of different types of physical quantities usually requires a heavy computational cost. Together with the decomposed gradient computation method (Ayusawa and Nakamura, 2012), the gradient computation could be performed with computational complexity $\mathcal{O}(N_J)$.

A couple of application examples were presented to demonstrate the usefulness of the proposed framework. We showed the dynamic trajectory optimization of a redundant serial robot manipulator composed of spherical joints. A collision-free dynamic motion was successfully generated in a cluttered environment with non-convex obstacles, while simultaneously imposing a strong torque limit. This validates the basic trajectory optimization capacity of the proposed framework under severe constraints. Another
application is optimization of the PE trajectory for identification of dynamic parameters of a humanoid robot. In this example, the analytical gradient of ZMP with respect to joint angle and its derivatives was utilized to guarantee the balance. Dynamic one-leg PE motions were generated and their validity was confirmed via a dynamic simulator.

The mathematical features of CMTM and its operator have a similarity to those of the rotational matrix and the cross product or those of the spatial transformation matrix and the screw operation. The rotational or screw motion of a rigid body has been studied from the view point of a Lie group and Lie algebra. Future work will focus on the features of CMTM from that point of view.

This paper represented the trajectory variables in the motion optimization of the joint coordinates and introduced the Jacobian of forces with respect to the joint coordinates according to the inverse dynamic formula. Concerning control issues, we often consider the joint torques as controller input variables and optimize them. Though the Jacobian with respect to the joint torques can be obtained by computing the inverse matrices, there is another possibility of inverse Jacobian formulation according to the efficient formula used in forward dynamics computation (Featherstone, 1983), which will be addressed in our future work.

There is still room for improvement in computation time of the proposed method by using parallel computation for multi-body systems (Featherstone, 2008). An algorithm for parallel computation will be investigated for future applications such as the real-time control of a humanoid robot or the motion analysis for a complicated human skeletal system.

**Funding**

This work was partially supported by a JSPS Grant-in-Aid for Scientific Research (A) (grant number 17H00768).

**ORCID iDs**

Ko Ayusawa, https://orcid.org/0000-0001-8188-4204

Eiichi Yoshida, https://orcid.org/0000-0002-3077-6964

**References**


### A. Appendix

#### A.1. Binary operation axioms

The bilinearity, alternativity, Jacobi identity axioms and anticommutativity with a binary operation $[,]$ are introduced

\[
[(ax_1+bx_2),x_3] = a[x_1,x_3]+b[x_2,x_3]
\]

\[
[x_3,(ax_1+bx_2)] = a[x_3,x_1]+b[x_3,x_2]
\]

\[
[x_1,x_1] = 0
\]

\[
[x_1,[x_2,x_3]] + [x_1,[x_2,x_3]] + [x_1,[x_2,x_3]] = 0
\]

\[
[x_1,x_2] = -[x_2,x_1]
\]

(In this paper, $[,]$ corresponds with $[\cdot,\cdot]$, $[,]$, or $[\cdot]$.)

#### A.2. Structure of $18 \times 18$ matrix $\hat{X}_k$ in Equation (33)

Here are $6 \times 6$ block matrices in $\hat{X}_k$ of Equation (33), when writing down with the notation of $\hat{X}_i^j$ of Equation (33),

\[
\hat{X}_k = \begin{bmatrix} A_k^{-1} & O_6 & O_6 \\ \hat{X}_k^{(2)} & A_k^{-1} & O_6 \\ \hat{X}_k^{(3)} & \hat{X}_k^{(4)} & A_k^{-1} \end{bmatrix}
\]

where

\[
\hat{X}_k^{(2)} \triangleq A_k^{-1} [\tilde{V}_k(x_k)]
\]

\[
\hat{X}_k^{(3)} \triangleq A_k^{-1} [(\tilde{V}_k(x_k)-A_k v_k) v_k]
\]

\[
\hat{X}_k^{(4)} \triangleq A_k^{-1} [\tilde{V}_k(x_k)] - [(v_k(x_k)+A_k(x_k)) v_k]
\]

\[
\tilde{V}_k \triangleq v_k - v_k
\]

\[
\tilde{V}_k \triangleq v_k - v_k - [v_k,v_k]
\]

#### A.3. Recursive formulas of kinematic chain

Let $S_i^{‘}$ denote an arbitrary transformation matrix that is non-singular and satisfies $S_k^{‘} = S_j^{‘} S_i^{‘}$. In regards to $S_i^{‘}$, physical quantity $a_i \in \mathbb{R}^n$ has the following recursive formulas

\[
a_i = S_i^{‘} a_i + c_i
\]

where, $c_i$ is a bias term. Similarly, let us consider another transformation matrix $U_i^{‘}$ and physical quantity $b_i \in \mathbb{R}^n$. There also exists the following recursive formulas

\[
b_i = P_i V_i a_i + d_i + \sum_{j \in C(i)} U_i^{‘} b_i
\]
where, \( d_i \) is a bias term, \( V_i \in \mathbb{R}^{m \times m} \) is an arbitrary non-singular matrix, and \( P_i \in \mathbb{R}^{n \times m} \) maps the space of \( d_i \) into \( c_i \).

Then, let us assume that Equation (72) has the following form

\[
\mathbf{b}_i = \pounds_i V_i \mathbf{a}_i + \hat{\mathbf{d}}_i \tag{73}
\]

By substituting the above equation and Equation (71), Equation (72) is

\[
\mathbf{b}_i = \left( P_i + \sum_{j \in C(i)} U_j^{(P)} V_j S_j V_i^{-1} \right) V_i \mathbf{a}_i + \mathbf{d}_i + \sum_{j \in C(i)} U_j^{(P)} V_j c_j + \hat{\mathbf{d}}_i
\]

From the comparison between the terms of the above equation and those of Equation (73), the following formulas are finally obtained

\[
\hat{\pounds}_i = P_i + \sum_{j \in C(i)} U_j^{(P)} V_j S_j V_i^{-1} \tag{74}
\]

\[
\hat{\mathbf{d}}_i = \mathbf{d}_i + \sum_{j \in C(i)} U_j^{(P)} V_j c_j + \hat{\mathbf{d}}_i \tag{75}
\]

### A.4. Structure of \( 6 \times 18 \) matrix \( \hat{\mathbf{H}}_j \) in Equation (42)

Let us write down \( 6 \times 18 \) matrix \( \hat{\mathbf{H}}_j \) into three \( 6 \times 6 \) block matrices \( \hat{\mathbf{K}}_j, \hat{\mathbf{D}}_j \) and \( \hat{\mathbf{M}}_j \) as follows

\[
\hat{\mathbf{H}}_j = \begin{bmatrix}
\hat{\mathbf{K}}_j & \hat{\mathbf{D}}_j & \hat{\mathbf{M}}_j
\end{bmatrix}
\]

\[
= \sum_{k \in C(j)} A_k^{-T} [O_k \ D_k \ M_k] S_k^{-1} X_k^{(j(k-1))} S_{p(k)}^{-1} \tag{76}
\]

By substituting the each component of \( X \) and \( S \) according to Equation (9) and Equation (13), we have

\[
\hat{\mathbf{M}}_j = M_j + \sum_{k \in C(j)} A_k^{-T} \tilde{M}_k A_k^{-1} \tag{77}
\]

\[
\hat{\mathbf{D}}_j = D_j + \sum_{k \in C(j)} A_k^{-T} \left( \hat{D}_k - \tilde{M}_k [v_j', 1] \right) A_k^{-1} \tag{78}
\]

\[
\hat{\mathbf{K}}_j = O_6 \tag{79}
\]

Let us examine the mechanical meaning of \( \hat{\mathbf{M}}_j \) and \( \hat{\mathbf{D}}_j \). The matrix \( \hat{\mathbf{M}}_j \) consists of the total mass \( \hat{m}_j \), the center of total mass \( \hat{c}_j \), and the total inertia tensor \( \hat{I}_j \) around the origin of link frame \( j \) of the kinematic sub-chain consisting of link \( j \) in \( \hat{\mathcal{C}}(j) \)

\[
\hat{\mathbf{M}}_j = \begin{bmatrix}
\hat{m}_j & \hat{m}_j [\hat{c}_j \times]^T \\
\hat{m}_j [\hat{c}_j \times] & \hat{I}_j
\end{bmatrix} \tag{80}
\]

The matrix \( \hat{\mathbf{D}}_j \) can be transformed as follows

\[
\hat{\mathbf{D}}_j = \sum_{j \in C(i)} A_j^{-T} \left( D_j - M_j [v_j', 1] \right) A_j^{-1}
\]

\[
= - \sum_{j \in C(i)} \left( A_j^{-T} (M_j [v_j', 1] + [v_j', 1]^T M_j) A_j^{-1} \right)
\]

\[
+ \sum_{j \in C(i)} \left( A_j^{-T} [v_j, z] \right)
\]

\[
- \sum_{j \in C(i)} \left( [A_j^{-T} M_j [v_j, z] \right) \tag{81}
\]

Let us here compute the derivative of the total inertia matrix \( \hat{\mathbf{M}}_i \) as follows

\[
\hat{\mathbf{M}}_i = - \sum_{j \in C(i)} A_j^{-T} (M_j [v_j, 1] + [v_j, 1]^T M_j) A_j^{-1}
\]

The total linear and angular momentum of the sub-chain can be represented by

\[
\hat{\mathbf{p}}_i = \sum_{j \in C(i)} A_j^{-T} M_j v_j
\]

Therefore, \( \hat{\mathbf{D}}_i \) can also be represented by the total momentum and the derivative of the total inertia matrix as follows

\[
\hat{\mathbf{D}}_i = \hat{\mathbf{M}}_i + \hat{\mathbf{M}}_i [v_i, 1] - [\hat{\mathbf{p}}_i, z] \tag{82}
\]