An Efficient Framework for Multiple Tasks in Human-like Robots

Jae Won Jeong#1, Pyung Hun Chang#2

Abstract—Human-like robots are required to simultaneously execute multiple tasks. A task-priority strategy plays an important role in implementing multiple tasks. Conventional task-priority strategies suffer from algorithmic singularity and large computational effort. Extended Operational Space (EXOS) formulation provides an algorithmic singularity free and computationally efficient framework for a single robot manipulator with two tasks. An extension to EXOS for multiple tasks, named Task-priority based EXOS (TPEXOS), has been proposed for whole-body control of human-like robots. TPEXOS augments constraint task spaces according to the order of priority without algorithmic singularities. The computational efficiency of TPEXOS excels other conventional task-priority strategies, whose efficacy and efficiency were demonstrated through simulation studies.

I. INTRODUCTION

A number of human-like robots have been developed by private corporations and academic institutions. These human-like robots have branching structures—tree-like topology involving much larger numbers of degrees of freedom (DOFs) than those usually found in conventional industrial robots [1]. Tasks are not limited to the specification of the position and orientation of a single end effector for robots with human-like structures. Multiple tasks are required to be simultaneously performed. Task descriptions may involve the combination of coordinates associated with arms, legs, and head even for the simplest goal of reaching an object in space.

A method for managing multiple tasks of human-like robots uses the task-priority strategy [1]-[3]. The concept of the task-priority strategy was introduced in order to avoid conflicts between tasks [4], [5]. There are large errors on both sides when two tasks conflict with each other. These conflicts can be avoided by performing tasks according to the order of priority. The task with the higher priority is first performed and the task with the lower priority is performed next, utilizing kinematic redundancy.

Conventional task-priority strategies suffer from algorithmic singularities [6]. An algorithmic singularity is an artificial characteristic, while a kinematic singularity is the intrinsic characteristic for robot manipulators [7]. An algorithmic singularity problem occurs when the task with lower priority conflicts with the task with higher priority [6]. The task-priority strategy gives an ill-conditioned solution and large joint velocities close to an algorithmic singularity.

Researchers have considered the algorithmic singularity problem of task-priority strategy. The singularity-robust inverse [8], which is also known as damped least-squares inverse [9], has been developed to overcome the difficulties encountered near singularities. This is obtained at the expense of increased task error [10], although continuity and good conditioning of the solution are ensured. The task-priority redundancy resolution technique, which has no algorithmic singularity in two task levels, was proposed by Chiaverini [6]. However, this algorithm causes an algorithmic singularity problem for more than three tasks and a large error for the secondary task [11].

A recursive formula is needed to prioritize multiple tasks in conventional task-priority strategies. This formula is generally concomitant with large amount of computational effort as the number of tasks increases. The heavy charge of computation requires a large expense for powerful hardware of real-time control. A few algorithms have been developed to improve computational efficiency. The incremental method, which was proposed by [3], [12], solves null space projection matrix more efficiently than the task-priority algorithm of Siciliano and Slotine [13]. An efficient recursive algorithm has been proposed to obtain the operational space inertia matrix of branching mechanism [14].

Extended Operational Space (EXOS) concept was proposed for explicit representation of the operational space of a single end effector and its null space for robot manipulators [15], [16]. This EXOS formulation provides a framework which has no algorithmic singularity and low computational effort. An extension to EXOS formulation, named Task-priority based EXOS (TPEXOS), which can handle multiple tasks according to the order of priority with no algorithmic singularity and low computational effort is described.

The paper is structured as follows: In section II, background for this paper will be described. In section III, TPEXOS formulation will be proposed. The analysis of TPEXOS and comparison with other formulations will be described in section IV. In section V, simulation results will be presented to demonstrate the effectiveness of TPEXOS. Finally, the conclusion will be described in section VI.

II. BACKGROUND

It is assumed that there are $k$ tasks to be performed and each task requires $n_i$ DOF. The kinematic equation for $i$-th task is given as

$$\mathbf{x}_i = \mathbf{f}_i(\mathbf{\theta}) \quad \text{for} \ 1 \leq i \leq k,$$

where $\mathbf{x}_i$ is the configuration of the end effector, $\mathbf{f}_i(\mathbf{\theta})$ is the forward kinematic function, and $\mathbf{\theta}$ is the joint configuration. The task-priority strategy requires a precedence order of tasks $\pi = \{1, 2, \ldots, k\}$, where $\pi_i$ indicates the priority of $i$-th task. The task-priority strategy prioritizes tasks according to $\pi_i$, and the task with the highest priority is performed first. The condition of the task-priority strategy is

$$\mathbf{f}_i(\mathbf{\theta}) = \mathbf{x}_i \quad \text{for} \ 1 \leq i \leq k,$$

and the condition of the null space is

$$\mathbf{N}(\mathbf{\theta}) = \mathbf{x}_i - \mathbf{x}_j \quad \text{for} \ 1 \leq i, j \leq k, i \neq j.$$
where \( x_i \in \mathbb{R}^m \) denotes the \( i \)-th task vector with respect to the base frame, \( \theta \in \mathbb{R}^n \) joint vector, and \( f \) a vector consisting of \( m \) scalar functions. The differential kinematics equation of the \( i \)-th task is determined as
\[
x_i = J_i(\theta) \dot{\theta},
\]
where \( (\cdot) \) denotes the time derivative, and \( J_i = \frac{\partial f_i}{\partial \theta} \in \mathbb{R}^{m \times n} \) denotes Jacobians. It is assumed that the \( i \)-th task has lower priority with respect to the previous (\( i-1 \))th task.

A. Review of Task-priority Strategies

The task-priority strategy has been studied by a number of authors. It can be classified into two recursive formulations, F1 and F2, as below.

In F1 and F2, \( P_i \) denotes the null space projection matrix of \( i \)-th task, and \( I \) denotes the identity matrix. Using the incremental algorithm, \( P_i \) is evaluated computationally efficiently as follows:
\[
P_i = P_{i-1} - (J_i P_{i-1})^T (J_i P_{i-1})^{-1},
\]
where \( P_i = I \), and \( (J_i P_{i-1})^T \) denotes the Moore-Penrose pseudoinverse matrix of \( J_i P_{i-1} \). In addition, \( \dot{\theta} \) denotes the required joint velocity in order to execute \( i \) desired tasks (from \( 1^{st} \) task to \( i \)-th task).

1) Formulation 1 (F1)

This formulation was proposed by [4], [5]. The formulation is described as follows:
\[
\dot{\theta}_i = \dot{\theta}_{i-1} + (J_i P_{i-1})^T (\dot{x}_i - J_i \dot{\theta}_i),
\]
where \( \dot{\theta}_0 = 0 \). In (4), \( (J_i P_{i-1})^T \) denotes the singularity-robust (SR) inverse [8] of \( J_i P_{i-1} \), and \( \lambda_i \) denotes the SR inverse scalar gain. The SR inverse is defined as follows:
\[
A^\lambda = A^T (AA^T + \lambda I)^{-1}.
\]

2) Formulation 2 (F2)

Chiaverini proposed the following formulation with two tasks [6]:
\[
\dot{\theta}_i = \dot{\theta}_{i-1} + (I - J_i^T J_i) (J_i^T \dot{x}_i).
\]

An extension to three or more tasks [12] is described as follows:
\[
\dot{\theta}_i = \dot{\theta}_{i-1} + P_{i-1} (J_i^T \dot{x}_i),
\]
where \( \dot{\theta}_0 = 0 \).

B. Algorithmic Singularities

Algorithmic singularities are defined as configurations at which the matrix \( J_i P_{i-1} \) loses rank with full rank Jacobians in task-priority strategies. An algorithmic singularity occurs whenever
\[
R(J_i^T) \cap R(J_{i-1}^T) \neq \{0\}.
\]

\( R(A) \) represents the range space of matrix \( A \), \( J_i \) denotes the Jacobian of the task with higher priority and \( J_{i-1} \) denotes the Jacobian of the task with lower priority, in (8). Therefore, algorithmic singularities occur when the task with lower priority conflicts with the task with higher priority.

The SR inverse is used to overcome the algorithmic singularity problem in (4). The SR inverse offers robustness to algorithmic singularities at the expense of reduced tracking accuracy [9]. If small SR inverse scalar gain is used, then tracking will be improved. However, the solution might get large near singularities. Therefore, \( k \) scalar gains must be tuned for \( k \) tasks.

The \( (J_i P_{i-1})^T \) is explicitly not used for algorithmic singularity robustness in (6). However, \( (J_i P_{i-1})^T \) is required for the evaluation of \( P_i \) for more than three tasks in (7).

C. EXOS concept

EXOS has been proposed for effective analysis and real-time control of the robot manipulators with kinematic redundancy [15], [16]. EXOS consists of operational space and its null space. The operational space is used to describe the manipulator end effector motion, and the null space is used to express self-motion.

The differential kinematics equation of the robot manipulator end effector is given as
\[
\ddot{x} = J_0 \dot{\theta},
\]
where \( x \in \mathbb{R}^n \) denotes the location of end effector with respect to the base frame, \( \theta \in \mathbb{R}^n \) joint velocity, and \( J \in \mathbb{R}^{r \times n} \) corresponding Jacobian. The degree of redundancy \( (n-m) \) is denoted by \( r \).

Now, define \( Z \in \mathbb{R}^{r \times n} \) as a matrix consisting of the orthonormal basis vectors spanning the null space. In addition, define \( \dot{x}_N \) as null space velocity. Then, we have a complementary mapping relationship at velocity level between joint space and constraint task space, which is
\[
\dot{x}_N \equiv Z \dot{\theta}.
\]

By using (9) and (10), EXOS Jacobian is defined as
\[
J_E = \begin{bmatrix} J_0^T Z^T \\ \dot{x}_N^T \end{bmatrix},
\]
and EXOS velocity is defined as
\[
\dot{x}_E \equiv \begin{bmatrix} \dot{x}^T \\ \dot{x}_N^T \end{bmatrix}.
\]

Therefore, EXOS differential kinematics equation is determined as
\[
\dot{x}_E = J_E \dot{\theta}.
\]

Thus, the inverse kinematics is obtained as
\[
\dot{\theta} = J_E^{-1} \dot{x}_E.
\]

EXOS formulation has advantages as follows:

- Algorithmic singularity does not exist in (14). The determinant of EXOS Jacobian is given as
\[
\det(J_E J_E^T) = \sqrt{\det(JJ^T)}.
\]

If there is no kinematic
singularity (i.e., $\mathbf{J}$ has full rank), then EXOS Jacobian has full rank.

- The inverse kinematic solution is obtained with better computational efficiency using EXOS formulation. EXOS Jacobian augments a minimum number of null space basis vectors. The dynamic equations of robot manipulators can be obtained in a very compact form.
- EXOS formulation enables to treat the differential kinematics of redundant robot manipulators as if they were nonredundant manipulators. The kinematics and dynamics of robot manipulators are obtained in the same manner as in the nonredundant case.

### III. TPEXOS

TPEXOS, an extension to EXOS, is proposed for managing multiple tasks without losing advantages of the original EXOS. It is assumed that there are $k$ tasks to be performed and each task requires $m_i$ DOFs and the human-like robot has $n$ DOFs.

#### A. Null Space Matrix Decomposition

Define the null space matrix for $i$-th task, $\mathbf{Z}_i$, as a matrix consisting of the orthonormal basis vectors spanning the null space of the $(i-1)$th task. Then, $\mathbf{Z}_i$ satisfies following relationships:

$$
\mathbf{Z}_i \mathbf{J}_i^T = \mathbf{0} \quad \text{(for } 2 \leq i \leq k),
$$

$$
\mathbf{Z}_i \mathbf{Z}_j^T = \begin{cases} 
1 & \text{if } (i = j) \quad \text{(for } 2 \leq i, j \leq k), \\
0 & \text{if } (i \neq j)
\end{cases},
$$

where $\mathbf{J}_i$ denotes the Jacobian of $1$st task, and $\mathbf{I}$ denotes the identity matrix. The dimension of $\mathbf{Z}_i$ is defined as follows:

$$
\mathbf{Z}_i \in \mathbb{R}^{n \times \mathcal{N}_i} \text{ if } (i \neq k),
$$

$$
\mathbb{R}^{n \times n} \text{ if } (i = k),
$$

where $r = n - \sum_{i=1}^{k-1} m_i$.

Null space matrices, $\mathbf{Z}_i$ (for $2 \leq i \leq k$), are obtained by the recursive formula from $\mathbf{Z}_1$ to $\mathbf{Z}_2$ as follows:

$$
\mathbf{Z}_i = \begin{cases} 
\text{Null}(\mathbf{J}_i^T \mathbf{J}_i) & \text{if } (i = k) \\
\text{Null}(\mathbf{J}_i^T \mathbf{J}_i) & \text{if } (i \neq k)
\end{cases},
$$

where the $\text{Null}(\cdot)$ represents the matrix consisting of orthonormal basis vectors spanning null space of $(\cdot)$, and the augmented Jacobian, $\mathbf{J}_i^T$, and the augmented null space matrix, $\mathbf{Z}_i^T$ are defined as follows:

$$
\mathbf{J}_i^T \triangleq \begin{bmatrix} 
\mathbf{J}_{1}^T & \mathbf{J}_{2}^T & \cdots & \mathbf{J}_{i-1}^T
\end{bmatrix}^T,
$$

$$
\mathbf{Z}_i^T \triangleq \begin{bmatrix} 
\mathbf{Z}_{1}^T & \mathbf{Z}_{2}^T & \cdots & \mathbf{Z}_{i-1}^T
\end{bmatrix}^T.
$$

#### B. Constraint Task Space Augmentation

Define $\mathbf{x}_n$ (for $2 \leq i \leq k$) as the constraint task velocity for $i$-th task. The dimension of $\mathbf{x}_n$ is defined as follows:

$$
\mathbf{x}_n \in \mathbb{R}^{m_i} \text{ if } (i \neq k),
$$

$$
\mathbb{R}^n \text{ if } (i = k).
$$

We have a complementary mapping relationship at velocity level between joint space and constraint task space, which is

$$
\mathbf{x}_n = \mathbf{Z}_i \hat{\mathbf{\theta}}.
$$

By using (2) and (21), let us define the TPEXOS as a space that consists of the primary task space, $\mathbf{x}_i$, and constraint task space, $\mathbf{x}_n$ (for $2 \leq i \leq k$).

Then, the TPEXOS Jacobian, $\mathbf{J}_E \in \mathbb{R}^{n \times \mathcal{N}_i}$, is defined as

$$
\mathbf{J}_E \equiv \begin{bmatrix} 
\mathbf{J}_{1}^T & \mathbf{Z}_{2}^T & \cdots & \mathbf{Z}_{i-1}^T
\end{bmatrix}^T,
$$

and the TPEXOS task velocity, $\hat{\mathbf{x}}_E \in \mathbb{R}^{n}$, as

$$
\hat{\mathbf{x}}_E \equiv \begin{bmatrix} 
\hat{\mathbf{x}}_1^T & \mathbf{x}_2^T & \cdots & \mathbf{x}_n^T
\end{bmatrix}^T.
$$

Now, the TPEXOS differential kinematics equation is determined as

$$
\hat{\mathbf{x}}_E = \mathbf{J}_E \hat{\mathbf{\theta}}.
$$

#### C. TPEXOS Inverse Kinematics

Forward kinematics is available from the defining relationship for TPEXOS as:

$$
\hat{\mathbf{x}}_{E_d} = \mathbf{J}_E \hat{\mathbf{\theta}}_d,
$$

where $\hat{\mathbf{x}}_{E_d}$ and $\hat{\mathbf{\theta}}_d$ are the desired TPEXOS task velocity and the corresponding joint velocity, respectively. The desired TPEXOS task velocity is defined as

$$
\hat{\mathbf{x}}_{E_d} \equiv \begin{bmatrix} 
\hat{\mathbf{x}}_{1d}^T & \hat{\mathbf{x}}_{2d}^T & \cdots & \hat{\mathbf{x}}_{nd}^T
\end{bmatrix}^T,
$$

where $\hat{\mathbf{x}}_{id}$ denotes the desired primary task velocity and $\hat{\mathbf{x}}_{nd}$ denotes the desired constraint task velocity for the $i$-th task. The optimal joint velocity for the $i$-th task, $\hat{\mathbf{\theta}}_{i,\text{opt}} \in \mathbb{R}^{n}$, is obtained by using the Moore-Penrose pseudoinverse of $\mathbf{J}_i$ as
\[ \dot{\theta}_{\text{opt}} = J_i^* \hat{x}_d. \]  

(27)

Therefore, the constraint task velocities are defined as

\[ \dot{x}_{\text{id}} = Z_i^* \theta_{\text{opt}}. \]  

(28)

The inverse kinematics is expressed as:

\[ \dot{\theta}_d = J^{-1}_e \dot{x}_d. \]  

(29)

D. Selection of Null Space Matrices

In (18), the null space matrices can be obtained by using either the singular value decomposition (SVD) method [17] or the method of Chang [18].

The integration method, an updating law, is over 5 times more efficient than the SVD method and has no discontinuity problem which can occur in the method of Chang [16].

By the integration method, the augmented null space matrix, \( Z^*_2 = [Z^*_2 \quad Z^*_3 \quad \cdots \quad Z^*_n]^T \), can be obtained as follows:

\[ Z^*_2(t) = Z^*_2(t - \Delta t) + Z^*_2(t)\Delta t, \]  

(30)

\[ \dot{Z}^*_2(t)^T = J^*_e(t - \Delta t)^{-1} \left[ -J_1(t)Z^*_2(t - \Delta t)^T - \frac{1}{2}k_z(I - Z^*_2(t - \Delta t)Z^*_2(t - \Delta t)^T) \right]. \]  

(31)

where \( \Delta t \) denotes sampling time.

- The initial \( Z^*_2(0) \) is computed by the SVD method or the method of Chang using (18).
- \( Z^*_2(t) \) is updated in real time using (30).

The integration error can be significantly reduced when a scalar gain, \( k_z \), is introduced [16]. The modified form is as follows:

\[ \dot{Z}^*_2(t)^T = J^*_e(t - \Delta t)^{-1} \left[ -J_1(t)Z^*_2(t - \Delta t)^T - k_zJ_1(t)Z^*_2(t - \Delta t)^T \right]. \]  

(32)

IV. DISCUSSION

A. Algorithmic Singularity Robustness

By using the definition of TPEXOS Jacobian, \( J^*_e \), in (22) and \( Z_i \) in (15) and (16), the determinant of \( J^*_e \) was derived. Since

\[ J^*_e = \begin{bmatrix} J_1^* & 0 & \cdots & 0 \\ 0 & Z_2^*Z_2 & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & Z_n^*Z_n \end{bmatrix}, \]  

(33)

it immediately follows that

\[ \det(J^*_e) = \det(J_1^*) \det(Z_2^*Z_2) \cdots \det(Z_n^*Z_n^T). \]  

(34)

The determinant of TPEXOS Jacobian is determined as

\[ \det(J^*_e) = \sqrt{\det(J_1^*)}. \]  

(35)

Equation (35) shows that TPEXOS Jacobian has the following properties:

- \( J^*_e \) is singular only when \( \det(J_1^*) = 0 \), which occurs if and only if \( J_1 \) loses its rank.
- At any configuration without kinematic singular point, there is no other singularity. Hence, no algorithmic singularity exists in the TPEXOS Jacobian.

B. Computational Effort

The computational effort is measured in terms of floating point operations (flops) [17]. We assume that Jacobians, \( J_i \), and desired task velocities, \( \dot{x}_{id} \), are given. The computational effort at each iteration step, \( N_{TPEXOS} \), is introduced as follows:

\[ N_{TPEXOS} = N_{\text{null}} + N_{x} + N_{x_{\text{inv}}}, \]  

(36)

where \( N_{\text{null}} \) denotes the computational effort required for null space matrices, \( Z_i \), and \( N_x \) denotes the effort to obtain the desired TPEXOS constraint task velocities, \( \dot{x}_{id} \), and \( N_{x_{\text{inv}}} \) denotes the effort to compute (29).

\[ N_{Znull} \] with the integration method, is given as

\[ N_{\text{null}} = (n + 1)n^2 + (2n^2 + n) \frac{n(n + 1)}{3} m_n - m_n^2. \]  

(37)

\( N_x \) is given as follows:

\[ N_x = \sum_{i=1}^k N_{\text{pseudo-inv}}(m, n) + \sum_{i=1}^k N_{\text{null}}(m, n) + N_{\text{null}}(r, n) \]

\[ \left( N_{\text{pseudo-inv}}(m, n) = \frac{2m^4}{3} + (n + 1)m^2 + \frac{9n - 2}{3} m_n - m_n^2 \right. \]

\[ \left. N_{\text{null}}(m, n) = m_n(2n - 1) \right) \]  

(38)

where \( N_{\text{pseudo-inv}}(n, m) \) denotes the effort for the optimized joint velocity for the i-th task, \( \dot{\theta}_{\text{opt}} = J_i^* \dot{x}_{id} \), by Gaussian elimination [10], and \( N_{\text{null}}(m, n) \) denotes the effort for the multiplication of \( Z_i \) and \( \dot{\theta}_{\text{opt}} \). The inverse of TPEXOS Jacobian matrix in (29) can be defined as

\[ J^{-1}_e = \begin{bmatrix} J^{-1}_1 & (Z_2^*)^{-1} \end{bmatrix}. \]  

(39)

Thus, \( N_{x_{\text{inv}}} \) is given as

\[ N_{x_{\text{inv}}} = 2n^2 + (3m^2 - 1) + m^3 - \frac{(m_n^2 + m_n)}{2} \]  

(40)

The comparison of computational effort among the TPEXOS, F1 and F2 is shown in Fig. 2 (the computational effort of F1 and F2 is described in Appendix). For the sake of simplicity, it is assumed that all Jacobians, \( J_i \), have the same size (\( m = 3 \)). This simulation results show that the TPEXOS has better computational efficiency than F1 and F2.

V. CASE STUDY

A. Simulation Environment

The TPEXOS was implemented and verified in MSC.visualNastran® environment which is integrated with MATLAB Simulink®. A human-like robot (Fig. 3) which has 17 DOF was used for a simulated experiment. The human-like
Fig. 2  Comparison the computational effort to evaluate $\theta$: (a) as the DOF of human-like robot increases for 5 tasks, (b) as the number of tasks increases in 30 DOF human-like robot.

Fig. 3  17 DOF Human-like robot solid model was developed using SoldWorks, a CAD software.

B. Task Description

In this simulation, the robot was commanded to reach a target position with its right hand while keeping self-balance and desired upper body orientation. The tasks and task-priority allotment are described as follows:

1) Self-balance

The self-balance task is defined by the global center of gravity coordinates, $x_1$, and the associated Jacobian, $J_1$, which can be expressed as

$$x_1 = M^{-1} \sum_{i=1}^{n} m(i)x_{\text{com}}(i),$$

$$J_1 = M^{-1} \sum_{i=1}^{n} m(i)J_{\text{com}}(i),$$

where $m(i)$ denotes the mass of link $i$, and $x_{\text{com}}(i)$ denotes the center of mass of link $i$, and $J_{\text{com}}(i)$ denotes the associated Jacobian, and $M$ denotes the total mass of the human-like robot. The global center of gravity is controlled at the center of the foot.

2) Hand position

The right hand position task is defined as following a trajectory to reach a target position. The $x_2$ denotes hand position vector with respect to the base frame, and $J_2$ denotes the associated Jacobian.

3) Upper body orientation

The upper body orientation task is defined as keeping the orientation 45 degrees about y axis. The $x_3$ denotes upper body orientation vector and $J_3$ denotes the associated Jacobian.

The described tasks were not simultaneously achievable. The goal was located too distant for the robot to reach out its hand while keeping its balance.

The self-balance task is assigned the highest priority. The hand position task and the upper body orientation task are assigned the secondary priority and the third priority, respectively.

C. Simulation Results

A sequence of snapshots from simulation is shown in Fig. 5. The simulation results are described in Fig. 6. Simulation results showed that the assigned tasks were performed according to priority without algorithmic singularity.

The hand was unable to reach its goal due to self-balance. Hence, the hand position task started to conflict with the self-balance task after 5 seconds. Algorithmic singularity problem can be occurred in conventional task-priority strategies because $J_1P_1$ which must be used in the conventional task-priority strategies became ill-conditioned after 5 seconds.
However, there was no algorithmic singularity problem in TPEXOS framework. The self-balance task, which has the highest priority, did not affect by the other tasks, and it was controlled precisely. The hand reached the closest possible position.

VI. Conclusion

A computationally efficient framework which prioritizes multiple tasks without algorithmic singularity is described in this paper. TPEXOS is proposed for multiple tasks as the extension to EXOS without losing advantages of the original EXOS. Desired tasks are projected into TPEXOS to avoid constraint violations according to the order of priority.

Inverse kinematics solution for the whole body control of human-like robots is obtained without algorithmic singularities based on the TPEXOS. Simulation results show that priorities among multiple tasks have been performed without algorithmic singularities. A set of tasks, such as balance, the position of hand, and upper body orientation, has been examined in this study at the whole-body level.

The computational efficiency of the proposed framework was improved with the comparative study of conventional task-priority strategies. This is useful for real-time applications with many task-priority levels.

APPENDIX

1) Computational effort of formulation 1

The computational effort is evaluated as follows:

\[ N_{p1} = N_{NS_{projection}} + N_{optimization}. \]  

In (42), \( N_{NS_{projection}} \) is obtained as follows:

\[
N_{NS_{projection}} = N_p + N_j
\]

where \( N_p \) denotes the effort for evaluating null space projection matrices, \( P_{i-1} \), using the incremental algorithm in (3), and \( N_j \) denotes the effort for the multiplication of Jacobians, \( J_i \), and null space projection matrices, \( P_{i-1} \). 

\[ N_{optimization} \] is obtained as follows:
where $N_{\text{task vec}}$ denotes the effort for computing $(\hat{x}_i - J \hat{\theta}_{i-1})$, and $N_{\text{SR inv}}$ denotes the effort for computing SR inverse of $(J_p \hat{x}_i)$, and the multiplication of $(J_p \hat{x}_i)^{k+1}$ and $(\hat{x}_i - J \hat{\theta}_{i-1})$, and $N_{\text{add}}$ denotes the effort for summation of joint velocity vectors.

2) Computational effort of formulation 2

The computational effort is evaluated as follows:

$$N_{F2} = N_{\text{NS projection}} + N_{\text{optimization}}$$

In (45), $N_{\text{optimization}}$ is obtained as follows:

$$N_{\text{optimization}} = \sum_{i=1}^{k} N_{\text{pseudo inv GE}}(n, m_i) + N_{\text{add}}$$

where $N_{\text{pseudo inv GE}}$ denotes the effort for local optimization of each task, $(J_p \hat{x}_i)$, and $N_{\text{add}}$ denotes the effort for addition of joint velocity vectors.

In (45), $N_{\text{NS projection}}$ is obtained as follows:

$$N_{\text{NS projection}} = N_p + N_{\text{proj}}$$

$$N_p = \sum_{i=1}^{k} n^2 (m_i^2 + \frac{1}{2}) + n(3m_i^2 - \frac{1}{2}) + m_i^3 - \frac{(m_i^2 + m_i)}{2}$$

$$N_{\text{proj}} = \sum_{i=1}^{k} n^2 (2m_i - n(m_i))$$

where $N_p$ denotes the effort for evaluating null space projection matrices, $P_{i-1}$, using the incremental algorithm in (3), and $N_{\text{proj}}$ denotes the effort for multiplication of $P_{i-1}$ and $(J_p \hat{x}_i)$.

REFERENCES