Equations Of Motion of Krang on Fixed Wheels

Munzir Zafar

March 30, 2015

In this report we attempt to find the dynamic model of Golem Krang with its wheels fixed. So it is reduced to a serial robot with a tree-structure (due to two arms branching out). Figure 1 shows the frames of references we will be using to determine the transforms and the coordinates on the robot. We denote these frames using symbol R_i where $i \in \mathbb{F} = \{0, 1, 2, 3, 4l, 5l, 6l, 7l, 8l, 9l, 10l, 4r, 5r, 6r, 7r, 8r, 9r, 10r\}$. R_0 is the world frame fixed in the middle of the two wheels. R_1, R_2, R_3 are fixed on the base, spine and torso with their rotations represented by q_{imu}, q_w and q_{torso} respectively. Frames $R_{4l}, ...R_{10l}$ are frames fixed on the links left 7-DOF arm with their motion represented by $q_{1l}, ...q_{7l}$. Similarly, frames $R_{4r}, ...R_{10r}$ are frames fixed on the links right 7-DOF arm with their motion represented by $q_{1r}, ...q_{7r}$. All equations in the following text that do not show r or l in the subscript where they are supposed to, will mean that the respective equations are valid for both subscripts.

We will be using the Kane's formulation. This is done so that our current analysis can easily be merged with the dynamic modelling of wheeled inverted pendulum which is found in terms of quasi-velocities, that prohibit the use of Lagrange for analytical modelling of the robot. Kane's method however is applicable for this problem.

1 Introduction to Kane's Formulation

$$\sum_k \left[m_k \bar{a}_{Gk} \cdot (\bar{v}_{Gk})_j + \left(\frac{d\bar{H}_{Gk}}{dt} \right) \cdot (\bar{\omega}_k)_j \right] = \sum_n \bar{F}_n \cdot (\bar{v}_n)_j + \sum_n \bar{M}_m \cdot (\bar{\omega}_m)_j \quad j = 1...K \tag{1}$$

where

- j is the unique number identifying each generalized co-ordinate in the system
- k is the unique number identifying each rigid body in the system
- n is the unique number identifying each external force acting on the system
- m is the unique number identifying each external torque acting on the system
- m_k is the mass of the kth body
- $\bar{a}_{Gk}$ is the acceleration of the center of mass of kth body
- $\bar{v}_{Gk}$ is the velocity of the center of mass of the kth body

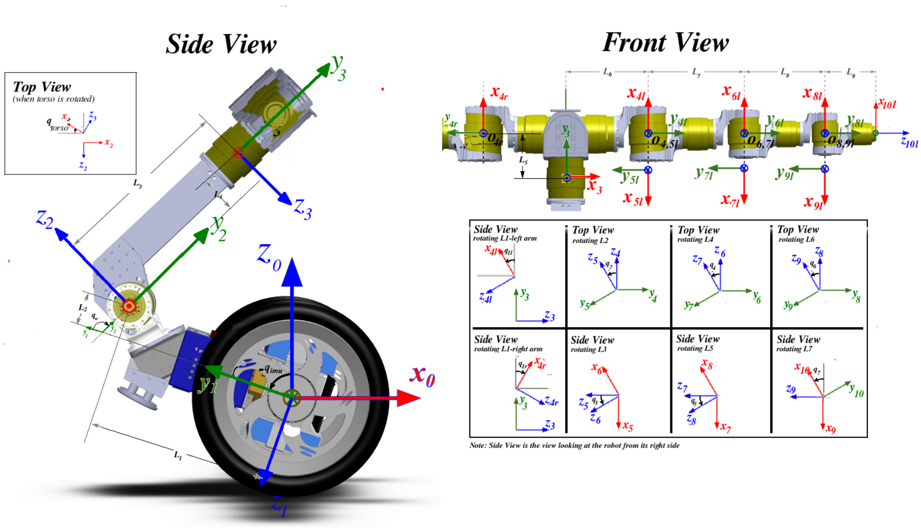

Figure 1: Frames of references on the robot

$\bar{H}_{Gk}$ is the angular momentum of body k about its center of mass

$\bar{\omega}_k$ is the angular velocity of the body k

F_n is the nth external force

M_m is the mth external moment

$\bar{v}_n$ is the velocity of the point at which external Force F_n is acting

$\bar{\omega}_m$ is the angular velocity of the body on which torque is acting relative to the actuator applying the torque

$()_j$ $= \frac{\partial()}{\partial \dot{q}_j}$ the partial derivative of the quantity in brackets () with respect to the generalized velocity $\dot{q}_j$

2 Transformations

The transformation of frame R_i into frame R_j is represented by the homogeneous transformation matrix ${}^iT_j$ such that.

$$ {}^iT_j = \begin{bmatrix} {}^is_j & {}^in_j & {}^ia_j & {}^iP_j \end{bmatrix} = \begin{bmatrix} & {}^iA_j & & {}^iP_j \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & n_x & a_x & P_x \\ s_y & n_y & a_y & P_y \\ s_z & n_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

where ${}^is_j$, ${}^in_j$ and ${}^ia_j$ contain the components of the unit vectors along the x_j, y_j and z_j axes respectively expressed in frame R_i, and where ${}^iP_j$ is the vector representing the coordinates of the origin of frame R_j expressed in frame R_i.

The transformation matrix ${}^iT_j$ can be interpreted as: (a) the transformation from frame R_i to frame R_j and (b) the representation of frame R_j with respect to frame R_i. Using figure 1, we can write down these transformation matrices for our system as follows:

$$ {}^0T_1 = \begin{bmatrix} 0 & sq_{imu} & -cq_{imu} & 0 \\ -1 & 0 & 0 & 0 \\ 0 & cq_{imu} & sq_{imu} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^1T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cq_w & sq_w & L_1 \\ 0 & -sq_w & cq_w & -L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^2T_3 = \begin{bmatrix} -cq_{torso} & 0 & sq_{torso} & 0 \\ 0 & 1 & 0 & L_3 \\ -sq_{torso} & 0 & -cq_{torso} & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

$$ {}^3T_{4l} = \begin{bmatrix} 0 & 1 & 0 & L_6 \\ cq_{1l} & 0 & -sq_{1l} & L_5 \\ -sq_{1l} & 0 & -cq_{1l} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^3T_{4r} = \begin{bmatrix} 0 & -1 & 0 & -L_6 \\ cq_{1r} & 0 & -sq_{1r} & L_5 \\ sq_{1r} & 0 & cq_{1r} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^4T_5 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -cq_2 & -sq_2 & 0 \\ 0 & -sq_2 & cq_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

$$ {}^5T_6 = \begin{bmatrix} -cq_3 & 0 & sq_3 & 0 \\ 0 & -1 & 0 & -L_7 \\ sq_3 & 0 & cq_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^6T_7 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -cq_4 & -sq_4 & 0 \\ 0 & -sq_4 & cq_4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^7T_8 = \begin{bmatrix} -cq_5 & 0 & sq_5 & 0 \\ 0 & -1 & 0 & -L_8 \\ sq_5 & 0 & cq_5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

$$ {}^8T_9 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -cq_6 & -sq_6 & 0 \\ 0 & -sq_6 & cq_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^9T_{10} = \begin{bmatrix} -cq_7 & -sq_7 & 0 & 0 \\ 0 & 0 & -1 & -L_9 \\ sq_7 & -cq_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

3 Velocities and Accelerations of Frames

The angular and linear velocities of the frames can be calculated using the recursive formulation:

$$ {}^j\omega_j = {}^jA_i\,{}^i\omega_i + \dot{q}_j\,{}^je_j \tag{3} $$
$$ {}^j\alpha_j = {}^jA_i\,{}^i\alpha_i + \ddot{q}_j\,{}^je_j + \dot{q}_j\,({}^j\omega_j \times {}^je_j) \tag{4} $$
$$ {}^jV_j = {}^jA_i\left({}^iV_i + {}^i\omega_i \times {}^iP_j\right) \tag{5} $$
$$ {}^ja_j = {}^jA_i\left({}^ia_i + {}^i\alpha_i \times {}^iP_j + {}^i\omega_i \times ({}^i\omega_i \times {}^iP_j)\right) \tag{6} $$

where ${}^i\omega_j$, ${}^i\alpha_j$, ${}^ia_j$ and ${}^iV_j$ denote the angular velocity, linear velocity, angular acceleration and linear acceleration repectively of frame j measured with respect to the world frame and represented in frame i. ${}^je_j$ denotes the direction of local angular velocity of frame j represented in frame j. $i, j \in \mathbb{F}$ identify the frames and i identifies the antecedent frame of j. So, the rotation ${}^jA_i$ and the translation ${}^jP_i$ that appear in these equations can not be directly deduced from the transformations listed in the previous section, as the they all represent ${}^iT_j$ (note the position of i and j). Rather, we need to use following expressions to deduce our matrices:

$$ {}^jA_i = {}^iA_j^T $$
$$ {}^jP_i = -{}^iA_j^T\,{}^iP_j $$

Since frame R_0 is fixed ${}^0\omega_0$ and ${}^0V_0$ are both $\begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$. We can deduce directions of local angular velocities of the frames using figure 1 as follows.

$$ {}^1e_1 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^2e_2 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^3e_3 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, {}^4e_4 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, $$

$$ {}^{5}e_5 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^{6}e_6 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, {}^{7}e_7 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^{8}e_8 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, $$
$$ {}^{9}e_9 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^{10}e_{10} = \begin{bmatrix}0 & 0 & -1\end{bmatrix}^T $$

This information can now be used to derive expressions for the velocities and accelerations of the frames.

4 Kane's Left-Hand Side

The inertial forces i.e. the term inside the brackets on the LHS in Kane's formulation can be simplified by expansion and manipulation that results in cancelation of terms leading to a simplified expression. We show the details of this manipulation here. The final expression is the outcome in the end which we will use in our code to find the dynamic model.

$$ \bar{v}_{Gk} = \bar{v}_k + \bar{\omega}_k \times \bar{S}_k \tag{7} $$
$$ \bar{a}_{Gk} = \bar{a}_k + \bar{\alpha}_k \times \bar{S}_k + \bar{\omega}_k \times (\bar{\omega}_k \times \bar{S}_k) \tag{8} $$
$$ \bar{H}_{Gk} = \mathbf{J}_{Gk}\bar{\omega}_k \tag{9} $$
$$ \begin{aligned} \frac{d\bar{H}_{Gk}}{dt} &= \mathbf{J}_{Gk}\bar{\alpha}_k + \bar{\omega}_k \times \mathbf{J}_{Gk}\bar{\omega}_k \\ &= (\mathbf{J}_k + m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times})\bar{\alpha}_k + \bar{\omega}_k \times (\mathbf{J}_k + m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times})\bar{\omega}_k \\ &= \mathbf{J}_k\bar{\alpha}_k + \bar{\omega}_k \times \mathbf{J}_k\bar{\omega}_k + m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\alpha}_k + \bar{\omega}_k \times (m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k) \end{aligned} \tag{10} $$
$$ \begin{aligned} m_k\bar{a}_{Gk} \cdot (\bar{v}_{Gk})_j &= m_k\bar{a}_{Gk} \cdot \left(\bar{v}_k + \bar{\omega}_k \times \bar{S}_k\right)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + m_k\bar{a}_{Gk} \cdot \left(\bar{\omega}_k \times \bar{S}_k\right)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j - m_k\left(\bar{a}_k + \bar{\alpha}_k \times \bar{S}_k + \bar{\omega}_k \times (\bar{\omega}_k \times \bar{S}_k)\right) \cdot \left(\bar{S}_k \times \bar{\omega}_k\right)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j - m_k\left(\bar{a}_k - \bar{S}_k \times \bar{\alpha}_k - \bar{\omega}_k \times (\bar{S}_k \times \bar{\omega}_k)\right) \cdot \left(\bar{S}_k \times (\bar{\omega}_k)_j\right) \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j - m_k\left(\bar{a}_k - \mathbf{S}_k^{\times}\bar{\alpha}_k - \omega_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k\right)^T \mathbf{S}_k^{\times} (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j - m_k\left(\mathbf{S}_k^{\times T}\bar{a}_k - \mathbf{S}_k^{\times T}\mathbf{S}_k^{\times}\bar{\alpha}_k - \mathbf{S}_k^{\times T}\omega_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k\right)^T (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + m_k\left(\mathbf{S}_k^{\times}\bar{a}_k - \mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\alpha}_k - \mathbf{S}_k^{\times}\omega_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k\right)^T (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + m_k\left(\bar{S}_k \times \bar{a}_k - \bar{S}_k \times \bar{S}_k \times \bar{\alpha}_k - \bar{S}_k \times (\bar{\omega}_k \times (\bar{S}_k \times \bar{\omega}_k))\right) \cdot (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + m_k\left(\bar{S}_k \times \bar{a}_k - \bar{S}_k \times \bar{S}_k \times \bar{\alpha}_k + \bar{\omega}_k \times ((\bar{S}_k \times \bar{\omega}_k) \times \bar{S}_k) + (\bar{S}_k \times \bar{\omega}_k) \times (\bar{S}_k \times \bar{\omega}_k)\right) \cdot (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + m_k\left(\bar{S}_k \times \bar{a}_k - \bar{S}_k \times \bar{S}_k \times \bar{\alpha}_k - \bar{\omega}_k \times (\bar{S}_k \times (\bar{S}_k \times \bar{\omega}_k)) + 0\right) \cdot (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + \left(m_k\bar{S}_k \times \bar{a}_k - m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\alpha}_k - \bar{\omega}_k \times (m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k)\right) \cdot (\bar{\omega}_k)_j \end{aligned} \tag{11} $$
$$ \begin{aligned} m_k\bar{a}_{Gk} \cdot (\bar{v}_{Gk})_j + \left(\frac{d\bar{H}_{Gk}}{dt}\right) \cdot (\bar{\omega}_k)_j & \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + \left(m_k\bar{S}_k \times \bar{a}_k - m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\alpha}_k - \bar{\omega}_k \times (m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k)\right) \cdot (\bar{\omega}_k)_j \\ &\quad + \left(\mathbf{J}_k\bar{\alpha}_k + \bar{\omega}_k \times \mathbf{J}_k\bar{\omega}_k + m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\alpha}_k + \bar{\omega}_k \times (m_k\mathbf{S}_k^{\times}\mathbf{S}_k^{\times}\bar{\omega}_k)\right) \cdot (\bar{\omega}_k)_j \\ &= m_k\bar{a}_{Gk} \cdot (\bar{v}_k)_j + \left(m_k\bar{S}_k \times \bar{a}_k + \mathbf{J}_k\bar{\alpha}_k + \bar{\omega}_k \times \mathbf{J}_k\bar{\omega}_k\right) \cdot (\bar{\omega}_k)_j \end{aligned} \tag{12} $$

The terms $\bar{\omega}_k$, $\bar{\alpha}_k$, $\bar{v}_k$, $\bar{a}_k$ will be found using the recursive formulations in eqs. 3-6. And the term $\bar{a}_{Gk}$ will be found using eq.8. $\bar{S}_k = \begin{bmatrix}\mathbf{X}_k & \mathbf{Y}_k & \mathbf{Z}_k\end{bmatrix}^T$ is the center of mass of the joint represented in the local frame. We will use the symbol $\mathbf{MS}_k$ to represent mass times center of mass $(m_k\mathbf{S}_k)$ which is the vector the components of which are $\mathbf{MS} = [\mathbf{MX} \quad \mathbf{MY} \quad \mathbf{MZ}]^T$. Finally, $\mathbf{J}_k = \begin{bmatrix} \mathbf{XX}_k & \mathbf{XY}_k & \mathbf{XZ}_k \\ \mathbf{XY}_k & \mathbf{YY}_k & \mathbf{YZ}_k \\ \mathbf{XZ}_k & \mathbf{YZ}_k & \mathbf{ZZ}_k \end{bmatrix}$ is the inertia matrix of the joint represented in the local frame.

5 Kane's Right-Hand Side

The right hand side of the Kane's forumulation 1 is the sum of some dot product terms. Each term is either the dot product of:

- force applied on the system $\bar{F}_n$
- the linear velocity $\bar{v}_n$ of the point differentiated partially wrt the the unique gerneralized speed $\dot{q}_j$ corresponding to each equation i.e. $\frac{\partial \bar{v}_n}{\partial \dot{q}_j}$

or the dot product of:

- torque applied on the system $\bar{\tau}_n$
- the angular velocity $\bar{\omega}_n$ of the body differentiated partially wrt the the unique gerneralized speed $\dot{q}_j$ corresponding to each equation i.e. $\frac{\partial \bar{\omega}_n}{\partial \dot{q}_j}$

So, in order to analyse the right-hand side of the equation, we need to list down all the forces and torques applied to the system and the points at which they are being applied. They are as follows.

$\bar{\tau}_j$ $= \tau_j \bar{e}_j$ The torque applied by each joint motor fixed on the antecedent joint moving the current joint. There are 17 such torques as

$$\tau_j \in \{\tau_{imu}, \tau_w, \tau_{torso}, \tau_{1l}, ..., \tau_{7l}, \tau_{1r}, ..., \tau_{7r}\}$$

Note that $\tau_{imu} = -\tau_R - \tau_L$ where τ_R and τ_L are torques applied to the right and left wheel respectively, and τ_{imu} is the sum of reactions torques experienced by the base in response.

$-\bar{\tau}_j$ $= -\tau_j \bar{e}_j$ The reaction torque experienced by each antecedent joint.

$\bar{F}_{gk}$ $= m_k \bar{g}$ is the weight of each joint k

$\bar{F}_{el}, \bar{\tau}_{el}$ Force and torque applied by the environment on the left hand end-effector at point E_l

$\bar{F}_{er}, \bar{\tau}_{er}$ Force and torque applied by the environment on the right hand end-effector at point E_r

5.1 Joint Torques

The contribution of motor torque of joint k and its reaction, on the right-hand side of Kane's equation corresponding to generalized speed $\dot{q}_j$ is:

$$\tau_k \bar{e}_k \cdot \frac{\partial \bar{\omega}_k}{\partial \dot{q}_j} - \tau_k \bar{e}_k \cdot \frac{\partial \bar{\omega}_{a(k)}}{\partial \dot{q}_j} \tag{13}$$

where $a(k)$ = antecedent frame of k. The first term is the contribution of action torques and the second term is the contribution of the reaction torques. Since

$$\bar{\omega}_k = \dot{q}_1\bar{e}_1 + \dot{q}_2\bar{e}_1 + \dot{q}_3\bar{e}_3 + ... + \dot{q}_k\bar{e}_k$$

$$\frac{\partial\bar{\omega}_k}{\partial\dot{q}_j} = \begin{cases} 0 & \text{if } k < j \\ \bar{e}_j & \text{if } k \geq j \end{cases} \tag{14}$$

If we take the summation of all joint torque contibutions each described by expression 13 i.e. for $k = 1...K$, to get get the right-hand side contribution by all joint torques in the kane's equaiton corresponding to a generalized speed j, we will get a very simplified solution

$$\begin{aligned}
&\sum_{k=1}^{K}\left(\tau_k\bar{e}_k \cdot \frac{\partial\bar{\omega}_k}{\partial\dot{q}_j} - \tau_k\bar{e}_k \cdot \frac{\partial\bar{\omega}_{k-1}}{\partial\dot{q}_j}\right) \\
&\quad = \sum_{k=1}^{j-1}\left(\tau_k\bar{e}_k \cdot \frac{\partial\bar{\omega}_k}{\partial\dot{q}_j} - \tau_k\bar{e}_k \cdot \frac{\partial\bar{\omega}_{k-1}}{\partial\dot{q}_j}\right) \\
&\quad\quad + \tau_j\bar{e}_j \cdot \frac{\partial\bar{\omega}_j}{\partial\dot{q}_j} - \tau_j\bar{e}_j \cdot \frac{\partial\bar{\omega}_{j-1}}{\partial\dot{q}_j} \\
&\quad\quad + \sum_{k=j+1}^{K}\left(\tau_k\bar{e}_k \cdot \frac{\partial\bar{\omega}_k}{\partial\dot{q}_j} - \tau_k\bar{e}_k \cdot \frac{\partial\bar{\omega}_{k-1}}{\partial\dot{q}_j}\right) \\
&\quad = \sum_{k=1}^{j-1}(0-0) + \tau_j\bar{e}_j \cdot \bar{e}_j - 0 + \sum_{k=j+1}^{K}(\tau_k\bar{e}_k \cdot \bar{e}_j - \tau_k\bar{e}_k \cdot \bar{e}_j) \\
&\quad = \tau_j
\end{aligned} \tag{15}$$

So the total joint torques contributions on the RHS of each Kane's equation is just the joint torque actuating the generalized speed wrt which the equaiton is being evaluated.

5.2 End-effector Forces and Torques

Let $\bar{F}_e$ and $\bar{\tau}_e$ be the force and torque being applied by the environment on the end-effector. The contribution on the RHS of Kane's equaiton corresponding to generalized speed $\dot{q}_j$ will be:

$$\bar{F}_e \cdot \frac{\partial\bar{v}_e}{\partial\dot{q}_j} + \bar{\tau}_e \cdot \frac{\partial\bar{\omega}_K}{\partial\dot{q}_j} \tag{16}$$

where $\bar{v}_e$ is the linear velocity of the point E on the end-effector on which the force is being applied. And $\bar{\omega}_K$ is the angular velocity of the last joint on which the end-effector is mounted.

$$\begin{aligned}
\bar{v}_e &= \bar{v}_K + \bar{\omega}_K \times \bar{r}_{e/O_K} \\
&= \bar{v}_{K-1} + \bar{\omega}_{K-1} \times \bar{r}_{O_K/O_{K-1}} + \bar{\omega}_K \times \bar{r}_{E/O_K} \\
&\ldots \\
&= \bar{v}_0 + \bar{\omega}_0 \times \bar{r}_{O_1/O_0} + \bar{\omega}_1 \times \bar{r}_{O_2/O_1} + \bar{\omega}_2 \times \bar{r}_{O_3/O_2} + ... + \bar{\omega}_{K-1} \times \bar{r}_{O_K/O_{K-1}} + \bar{\omega}_K \times \bar{r}_{O_{K+1}/O_K} \\
&= \bar{v}_0 + \sum_{k=0}^{K}\left(\bar{\omega}_k \times \bar{r}_{O_{k+1}/O_k}\right)
\end{aligned} \tag{17}$$

where we have replace E with O_{K+1} for the convenience of writing the closed-form expression. When partially differentiated wrt $\dot{q}_j$ we can use eq. 14 to get:

$$\begin{aligned}
\frac{\partial \bar{v}_e}{\partial \dot{q}_j} &= \frac{\partial \bar{v}_0}{\partial \dot{q}_j} + \sum_{k=0}^{K} \left(\frac{\partial \bar{\omega}_k}{\partial \dot{q}_j} \times \bar{r}_{O_{k+1}/O_k} \right) \\
&= 0 + \sum_{k=0}^{j-1} \left(\frac{\partial \bar{\omega}_k}{\partial \dot{q}_j} \times \bar{r}_{O_{k+1}/O_k} \right) + \sum_{k=j}^{K} \left(\frac{\partial \bar{\omega}_k}{\partial \dot{q}_j} \times \bar{r}_{O_{k+1}/O_k} \right) \\
&= \sum_{k=0}^{j-1} \left(0 \times \bar{r}_{O_{k+1}/O_k} \right) + \sum_{k=j}^{K} \left(\bar{e}_j \times \bar{r}_{O_{k+1}/O_k} \right) \\
&= \bar{e}_j \times \sum_{k=j}^{K} \bar{r}_{O_{k+1}/O_k} \\
&= \bar{e}_j \times \left(\bar{r}_{O_{j+1}/O_j} + \bar{r}_{O_{j+2}/O_{j+1}} + \bar{r}_{O_{j+3}/O_{j+2}} + ... + \bar{r}_{O_{K+1}/O_K} \right) \\
&= \bar{e}_j \times \bar{r}_{O_{K+1}/O_j} \\
&= \bar{e}_j \times \bar{r}_{E/O_j}
\end{aligned} \tag{18}$$

Substituting eqs. 18 and 14 in expression 16, we get

$$\begin{aligned}
&\bar{F}_e \cdot \bar{e}_j \times \bar{r}_{E/O_j} + \bar{\tau}_e \cdot \bar{e}_j \\
&= \begin{bmatrix} \left[\bar{e}_j \times \bar{r}_{E/O_j} \right]^T & \bar{e}_j^T \end{bmatrix} \begin{bmatrix} \bar{F}_e \\ \bar{\tau}_e \end{bmatrix}
\end{aligned}$$

If we write all kane's equations in the form of a vector then the right hand side contribution of end-effector force and torque will become,

$$\begin{aligned}
&\begin{bmatrix} \left[\bar{e}_1 \times \bar{r}_{E/O_1} \right]^T & \bar{e}_1^T \\ \left[\bar{e}_2 \times \bar{r}_{E/O_2} \right]^T & \bar{e}_2^T \\ \left[\bar{e}_3 \times \bar{r}_{E/O_3} \right]^T & \bar{e}_3^T \\ ... & \\ \left[\bar{e}_K \times \bar{r}_{E/O_K} \right]^T & \bar{e}_K^T \end{bmatrix} \begin{bmatrix} \bar{F}_e \\ \bar{\tau}_e \end{bmatrix} \\
&= \mathbb{J}^T \mathbb{f}_e
\end{aligned} \tag{19}$$

where

$$\begin{aligned}
\mathbb{J} &= \begin{bmatrix} \bar{e}_1 \times \bar{r}_{E/O_1} & ... & \bar{e}_K \times \bar{r}_{E/O_K} \\ \bar{e}_1 & ... & \bar{e}_K \end{bmatrix} \\
\mathbb{f} &= \begin{bmatrix} \bar{F}_e \\ \bar{\tau}_e \end{bmatrix}
\end{aligned}$$

The matrix $\mathbb{J}$ is refered to as the Jacobian. And $\mathbb{f}$ is the wrench (formal term to refer to a force/torque couple). This whole theory was assuming a single serial chain of the robot with a single end-effector. For the case of krang, we will have two wrenches $\mathbb{f}_{el}$ and $\mathbb{f}_{er}$ applied at two end-effectors on the right and the left arms respectively. The points on which this wrench is being sensed on the two end-effectors is E_L and E_R. So the contribution becomes:

$$\mathbb{J}_L^T \mathbb{f}_{el} + \mathbb{J}_R^T \mathbb{f}_{er} \tag{20}$$

where

- $\mathbb{J}_L = \begin{bmatrix} e_1 \times \bar{r}_{E_L/O_1} & e_2 \times \bar{r}_{E_L/O_2} & e_3 \times \bar{r}_{E_L/O_3} & e_{4l} \times \bar{r}_{E_L/O_{4l}} & \dots & e_{10l} \times \bar{r}_{E_L/O_{10l}} & O_{3\times 7} \\ e_1 & e_2 & e_3 & e_{4l} & \dots & e_{10l} & O_{3\times 7} \end{bmatrix}$
- $\mathbb{J}_{10r} = \begin{bmatrix} e_1 \times \bar{r}_{E_R/O_1} & e_2 \times \bar{r}_{E_R/O_2} & e_3 \times \bar{r}_{E_R/O_3} & O_{3\times 7} & e_{4r} \times \bar{r}_{E_R/O_{4r}} & \dots & e_{10r} \times \bar{r}_{E_R/O_{10r}} \\ e_1 & e_2 & e_3 & O_{3\times 7} & e_{4r} & \dots & e_{10r} \end{bmatrix}$

5.3 Gravitational Forces

The contribution of the weight of a joint k to the right-hand side of the equation corresponding to generalized speed $\dot{q}_j$ is

$$m_k \bar{g} \cdot \frac{\partial \bar{v}_{Gk}}{\partial \dot{q}_j} \tag{21}$$

Now

$$\begin{aligned}
\bar{v}_{Gk} &= \bar{v}_k + \bar{\omega}_k \times \bar{r}_{Gk/O_k} \\
&= \bar{v}_{k-1} + \bar{\omega}_{k-1} \times r_{O_k/O_{k-1}} + \bar{\omega}_k \times \bar{r}_{Gk/O_k} \\
&= \bar{v}_0 + \sum_{i=1}^{k} \left(\bar{\omega}_{i-1} \times r_{O_i/O_{i-1}}\right) + \bar{\omega}_k \times \bar{r}_{Gk/O_k} \\
\frac{\partial \bar{v}_{Gk}}{\partial \dot{q}_j} &= \frac{\partial \bar{v}_0}{\partial \dot{q}_j} + \sum_{i=1}^{k} \left(\frac{\partial \bar{\omega}_{i-1}}{\partial \dot{q}_j} \times r_{O_i/O_{i-1}} \right) + \frac{\partial \bar{\omega}_k}{\partial \dot{q}_j} \times \bar{r}_{Gk/O_k} \\
&= 0 + \sum_{i=1}^{k} \left(\frac{\partial \bar{\omega}_{i-1}}{\partial \dot{q}_j} \times r_{O_i/O_{i-1}} \right) + \frac{\partial \bar{\omega}_k}{\partial \dot{q}_j} \times \bar{r}_{Gk/O_k} \\
&= \begin{cases} 0 & \text{if } k < j \\ \sum\limits_{i=j+1}^{k} \left(\bar{e}_j \times r_{O_i/O_{i-1}}\right) + \bar{e}_j \times \bar{r}_{Gk/O_k} & \text{if } k \geq j \end{cases} \\
&= \begin{cases} 0 & \text{if } k < j \\ \bar{e}_j \times \left(\sum\limits_{i=j+1}^{k} \bar{r}_{O_i/O_{i-1}} + \bar{r}_{Gk/O_k} \right) & \text{if } k \geq j \end{cases} \\
&= \begin{cases} 0 & \text{if } k < j \\ \bar{e}_j \times \bar{r}_{Gk/O_j} & \text{if } k \geq j \end{cases}
\end{aligned} \tag{22}$$

The total contributions of all joints will therefore be:

$$\begin{aligned}
&\sum_{k=1}^{K} \left(m_k \bar{g} \cdot \frac{\partial \bar{v}_{Gk}}{\partial \dot{q}_j} \right) \\
&= \bar{g} \cdot \left(\sum_{k=1}^{j-1} m_k \frac{\partial \bar{v}_{Gk}}{\partial \dot{q}_j} + \sum_{k=j}^{K} m_k \frac{\partial \bar{v}_{Gk}}{\partial \dot{q}_j} \right) \\
&= \bar{g} \cdot \left(\sum_{k=1}^{j-1} 0 + \sum_{k=j}^{K} m_k \left(e_j \times \bar{r}_{Gk/O_j}\right) \right) \\
&= \bar{g} \cdot \left(e_j \times \sum_{k=j}^{K} m_k \bar{r}_{Gk/O_j} \right)
\end{aligned} \tag{23}$$

Note that this derivation is assuming a single serial chain. In the case of Krang however, if j corresponds to a speed of joint before the branching takes place i.e. if $j \in \{imu, w, torso\}$ the range of summation will include all joints in the tree above the current joint. If j corresponds to the speed of joint in one of the branches i.e. if $j \in \{1l, ..., 7l\}$ or $j \in \{1r, ..., 7r\}$ then the range of summation will only be consisting of joints following the current joint in the specific branch.

6 MATLAB code For Finding the Dynamic Model

The dynamic model is generated using a script `dynamicModel.m` found in the folder *stableForceInteraction/Implementation/1-ForceControlWhileBalancing/1-ControlProblem1/1-DynamicModeling/2-DynamicModelOfTreeStructuredRobot/2-matlab/Kanes*. The function populates the frame information in a map container using `getKrangFrames()`, then calculates the left-hand side of the Kane's terms using the function `kanesLHS()` and finally constructs the **A** and **C** matrices in the dynamic model using the function `getAC()`. These functions and the data structures are briefly discussed in the following subsections.

6.1 Map Container for all the Frame Information

The function `getKrangFrames()` populates the information in a map container `f`. A map container is a data structure in MATLAB that stores a list of data that is retrievable using a key. We store a `frame` structure in each cell of the map and use strings $s \in \mathbf{S} = \{$ `'0'`, `'1'`, `'2'`, `'3'`, `'4l'`, `'5l'`, `'6l'`, `'7l'`, `'8l'`, `'9l'`, `'10l'`, `'4r'`, `'5r'`, `'6r'`, `'7r'`, `'8r'`, `'9r'`, `'10r'` $\}$ as a key to retrieve information. The `frame` structure stores the following elements:

- `x` the unit vector along x-axis represented in the antecedent frame
- `y` the unit vector along y-axis represented in the antecedent frame
- `z` the unit vector along z-axis represented in the antecedent frame
- `P` the position of the origin frame represented in the antecedent frame
- `e` the unit vector along direction of positive rotation of the frame represented in the local frame
- `a` the string $\in \mathbf{S}$ (defined above) that is the key that maps to the antecedent frame
- `q` the symbolic variable used for representing the generalized position q associated with this frame
- `dq` the symbolic variable used for representing the generalized speed $\dot{q}$ associated with this frame
- `ddq` the symbolic variable used for representing the generalized acceleration $\ddot{q}$ associated with this frame
- `o` the row number in the inertia matrix **A** (i.e. in the final dynamic model $\mathbf{A}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{Q} = \mathbf{F}$) that corresponds to the current joint
- `param` array of ten symbolic variables used to represent the inertial parameters of the joint $[m \quad \mathbf{MX} \quad \mathbf{MY} \quad \mathbf{MZ} \quad \mathbf{XX} \quad \mathbf{YY} \quad \mathbf{ZZ} \quad \mathbf{XY} \quad \mathbf{XZ} \quad \mathbf{YZ}]^T$

6.2 Functions associated with the Map Container

There are a number of functions that take the map container `f` as the input argument and construct a useful information as an output. Here is a list of those functions:

`isBefore(f, key1, key2)` returns `true` if frame 1 (identified by `key1`) is before frame 2 (identified by `key2`) in the tree structure of the robot

`Rot(f, key1, key2)` returns rotation transform ${}^{j}A_i$ i.e. represention of frame i (identified by `key1`) in frame j (identified by `key2`)

`Tf(f, key1, key2)` returns rotation transform ${}^{j}T_i$ i.e. represention of frame i (identified by `key1`) in frame j (identified by `key2`)

`qVec(f)` generates the vector $\mathbf{q}$ containing generalized positions of all frames

`dqVec(f)` generates the vector $\dot{\mathbf{q}}$ containing generalized speeds of all frames

`ddqVec(f)` generates the vector $\ddot{\mathbf{q}}$ containing generalized accelerations of all frames

`mass(f, key)` returns the mass of the frame identified by the `key`

`mCOM(f, key)` returns the mass times COM ($\mathbf{MS} = [\mathbf{MX} \quad \mathbf{MY} \quad \mathbf{MZ}]^T$) of the joint identified by the `key` represented in the local frame

`inertiaMat(f, key)` returns the inertia matrix ($\mathbf{J} = \begin{bmatrix} \mathbf{XX} & \mathbf{XY} & \mathbf{XZ} \\ \mathbf{XY} & \mathbf{YY} & \mathbf{YZ} \\ \mathbf{XZ} & \mathbf{YZ} & \mathbf{ZZ} \end{bmatrix}$) of the joint indentified by `key` represented in the local frame

`angVel(f, key)` returns the symbolic expression for the angular velocity ($\bar{\omega}$) of the joint represented in local frame calculated recursively using eq. 3

`angAcc(f, key)` returns the symbolic expression for the angular acceleration ($\bar{\alpha}$) of the joint represented in local frame calculated recursively using eq. 4

`linVel(f, key)` returns the symbolic expression for the linear velocity ($\bar{v}$) of the joint represented in local frame calculated recursively using eq. 5

`linAcc(f, key)` returns the symbolic expression for the linear acceleration ($\bar{a}$) of the joint represented in local frame calculated recursively using eq. 6

6.3 Kane's Left-Hand Side

Kane's formulation (eq.1) gives a number of equations, one for each generalized speed $\dot{q}_j$. The left-hand side of each equation is a summation of a number of terms one for each joint k. Each term is calculated using eq.12 which gives a simplified version of what occured in the brackets on which the summation is operating on the left-hand side of the original Kane's equation.

The function `kanesLHS()` is written to calculate the symbolic expression for the left hand side of each equation. The function iterates through each joint k using the keys to the map container. For each joint it:

- evaluates the symbolic expressions for the respective joint velocities and accelerations ($\bar{\omega}_k$, $\bar{\alpha}_k$, $\bar{v}_k$ and $\bar{a}_k$) using the helper functions described in the previous subsection

- evaluates for this joint k the symbolic expression for the term in the brackets, on which the summation is operating on the left-hand side of the original Kane's equation, using eq.12. A term for each equation with respect to each generalized speed $\dot{q}_j$ (i.e. a sub-loop iterating through all generalized speeds). This gives the contribution of the inertial forces of joint k to the left-hand side of each equation in the Kane's formulation.
- adds the final expression to an accumulator (to apply the summation)

A vector of size the same as the number of Kane's eqations is thus generated each element of which corresponds to a unique generalized speed q_j and represents the sum of all joint contributions with respect to the unique joint speed corresponding to this element. So each element represents the left-hand side of the each equation in the Kane's formulation.

Finally, once all the symbolic expressions are generated using `kanesLHS()`, we collect the coefficients of each $\ddot{q}_j$ and $\dot{q}_j$ in each equation to generate our $\mathbf{A}$ and $\mathbf{C}$ matrices in the final dynamic equation $\mathbf{A}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{Q} = \mathbf{F}$. These matrices are saved in the symbolic variable matrices `AA` and `CC`. These variables can be loaded into the workspace by loading the file `kanesLHS.mat`.

6.4 Kane's Right-Hand Side

The evaluation of right-hand is written in a separate script `kanesRHS.m`. It uses the original closed form expressions for the evaluation of the forces i.e. eqs. 13, 16 and 21. The derivation of the simplified expressions presented in eqs. 15, 19 and 23 could also be used but we did not use them. We can write a separate code later to verify the results of the former by comparing it with the result of derived expressions. We have not done that as yet.

The result of the right-hand side evaluation is stored in the symbolic variable vectors `a, b1, b2, b3, b4 and c` containing contributions of $\bar{\tau}_j$, $\bar{F}_{el}$, $\bar{\tau}_{el}$, $\bar{F}_{er}$, $\bar{\tau}_{er}$ and $\bar{F}_{gk}$ respectively. The code can be found in the appendix.

7 Appendix

We present the code listings of the various functions in this section.

7.1 `dynamicModel.m`

```
f = getKrangFrames(17);
[KK, QQ, KHist] = kanesLHS(f);
[AA, CC] = getAC(f, KK);
```

7.2 `getKrangFrames()`

```
function f = getKrangFrames(nFrames)

% This function generates a map. The keys to the maps are string literals
% in {'1','2','3','4l','5l', ...'10l','4r','5r', ...'10r'}
% and the values are structs that have members associated with the
% respective joints of the robot.
% 'x', 'y', 'z', 'P' define the unit vectors and position of origin of the
% frame represented in the antecedent frame
% 'e' is the local angular speed of the frame represented in the same frame
```

```
% 'a' contains the key to the antecedent frame
% 'param' contains the inertial parameters of the respective link
% 'q', 'dq' and 'ddq' contain associated joint pos, vel and acc variables
% 'o' defines the row of inertia matrix A (in the dynamic equations) that
% corresponds to the current joint.

syms q_imu q_w q_torso q_1l q_2l q_3l q_4l q_5l q_6l q_7l real
syms q_1r q_2r q_3r q_4r q_5r q_6r q_7r L1 L2 L3 L4 L5 L6 L7 L8 L9 real
syms dq_imu dq_w dq_torso dq_1l dq_2l dq_3l dq_4l dq_5l dq_6l dq_7l real
syms dq_1r dq_2r dq_3r dq_4r dq_5r dq_6r dq_7r L1 L2 L3 L4 L5 L6 L7 L8 L9 real
syms ddq_imu ddq_w ddq_torso ddq_1l ddq_2l ddq_3l ddq_4l ddq_5l ddq_6l ddq_7l real
syms ddq_1r ddq_2r ddq_3r ddq_4r ddq_5r ddq_6r ddq_7r real
syms m_1 MX_1 MY_1 MZ_1 XX_1 XY_1 XZ_1 YY_1 YZ_1 ZZ_1 real
syms m_2 MX_2 MY_2 MZ_2 XX_2 XY_2 XZ_2 YY_2 YZ_2 ZZ_2 real
syms m_3 MX_3 MY_3 MZ_3 XX_3 XY_3 XZ_3 YY_3 YZ_3 ZZ_3 real
syms m_4l MX_4l MY_4l MZ_4l XX_4l XY_4l XZ_4l YY_4l YZ_4l ZZ_4l real
syms m_5l MX_5l MY_5l MZ_5l XX_5l XY_5l XZ_5l YY_5l YZ_5l ZZ_5l real
syms m_6l MX_6l MY_6l MZ_6l XX_6l XY_6l XZ_6l YY_6l YZ_6l ZZ_6l real
syms m_7l MX_7l MY_7l MZ_7l XX_7l XY_7l XZ_7l YY_7l YZ_7l ZZ_7l real
syms m_8l MX_8l MY_8l MZ_8l XX_8l XY_8l XZ_8l YY_8l YZ_8l ZZ_8l real
syms m_9l MX_9l MY_9l MZ_9l XX_9l XY_9l XZ_9l YY_9l YZ_9l ZZ_9l real
syms m_10l MX_10l MY_10l MZ_10l XX_10l XY_10l XZ_10l YY_10l YZ_10l ZZ_10l real
syms m_4r MX_4r MY_4r MZ_4r XX_4r XY_4r XZ_4r YY_4r YZ_4r ZZ_4r real
syms m_5r MX_5r MY_5r MZ_5r XX_5r XY_5r XZ_5r YY_5r YZ_5r ZZ_5r real
syms m_6r MX_6r MY_6r MZ_6r XX_6r XY_6r XZ_6r YY_6r YZ_6r ZZ_6r real
syms m_7r MX_7r MY_7r MZ_7r XX_7r XY_7r XZ_7r YY_7r YZ_7r ZZ_7r real
syms m_8r MX_8r MY_8r MZ_8r XX_8r XY_8r XZ_8r YY_8r YZ_8r ZZ_8r real
syms m_9r MX_9r MY_9r MZ_9r XX_9r XY_9r XZ_9r YY_9r YZ_9r ZZ_9r real
syms m_10r MX_10r MY_10r MZ_10r XX_10r XY_10r XZ_10r YY_10r YZ_10r ZZ_10r real

nullSym = sym([0; 0; 0]); nullSymParam = sym(zeros(10,1));
frame.x = nullSym; frame.y = nullSym; frame.z = nullSym; frame.P = nullSym;
frame.e = nullSym; frame.a = ''; frame.param = nullSymParam;
frame.q = sym(0); frame.dq = sym(0); frame.o = 0;
frame.angVel = sym(0); frame.linVel = sym(0);
frame.angAcc = sym(0); frame.linAcc = sym(0);
frame.gotAngVel = false; frame.gotLinVel = false;
frame.gotAngAcc = false; frame.gotLinAcc = false;

frame.x = sym([1; 0; 0]); frame.y = sym([0; 1; 0]);
frame.z = sym([0; 0; 1]); frame.P = sym([0; 0; 0]);
frame.e = [0; 0; 0]; frame.a = '0';
frame.q = sym(0); frame.dq = sym(0); frame.o = 0;
frame.param = nullSymParam;
% f('0') = frame;
F{1} = frame;

frame.x = sym([0; -1; 0]); frame.y = [sin(q_imu); 0; cos(q_imu)];
frame.z = [-cos(q_imu); 0; sin(q_imu)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0]; frame.a = '0';
frame.q = q_imu; frame.dq = dq_imu; frame.ddq = ddq_imu; frame.o = 1;
frame.param = [m_1 MX_1 MY_1 MZ_1 XX_1 XY_1 XZ_1 YY_1 YZ_1 ZZ_1];
% f('1') = frame;
F{2} = frame;

frame.x = sym([1; 0; 0]); frame.y = [0; cos(q_w); -sin(q_w)];
frame.z = [0; sin(q_w); cos(q_w)]; frame.P = [0; L1; -L2];
frame.e = [-1; 0; 0];  frame.a = '1';
frame.q = q_w; frame.dq = dq_w; frame.ddq = ddq_w; frame.o = 2;
frame.param = [m_2 MX_2 MY_2 MZ_2 XX_2 XY_2 XZ_2 YY_2 YZ_2 ZZ_2];
% f('2') = frame;
F{3} = frame;

frame.x = [-cos(q_torso); 0; -sin(q_torso)]; frame.y = sym([0; 1; 0]);
frame.z = [sin(q_torso); 0; -cos(q_torso)]; frame.P = [0; L3; L4];
frame.e = [0; -1; 0];  frame.a = '2';
frame.q = q_torso; frame.dq = dq_torso; frame.ddq = ddq_torso; frame.o = 3;
frame.param = [m_3 MX_3 MY_3 MZ_3 XX_3 XY_3 XZ_3 YY_3 YZ_3 ZZ_3];
% f('3') = frame;
F{4} = frame;

frame.x = [0; cos(q_1l); -sin(q_1l)]; frame.y = sym([1; 0; 0]);
frame.z = [0; -sin(q_1l); -cos(q_1l)]; frame.P = sym([L6; L5; 0]);
frame.e = [0; -1; 0];  frame.a = '3';
frame.q = q_1l; frame.dq = dq_1l; frame.ddq = ddq_1l; frame.o = 4;
frame.param = [m_4l MX_4l MY_4l MZ_4l XX_4l XY_4l XZ_4l YY_4l YZ_4l ZZ_4l];
% f('4l') = frame;
F{5} = frame;

frame.x = sym([-1; 0; 0]); frame.y = [0; -cos(q_2l); -sin(q_2l)];
frame.z = [0; -sin(q_2l); cos(q_2l)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0];  frame.a = '4l';
frame.q = q_2l; frame.dq = dq_2l; frame.ddq = ddq_2l; frame.o = 5;
frame.param = [m_5l MX_5l MY_5l MZ_5l XX_5l XY_5l XZ_5l YY_5l YZ_5l ZZ_5l];
% f('5l') = frame;
F{6} = frame;

frame.x = [-cos(q_3l); 0; sin(q_3l)]; frame.y = sym([0; -1; 0]);
frame.z = [sin(q_3l); 0; cos(q_3l)]; frame.P = sym([0; -L7; 0]);
frame.e = [0; -1; 0];  frame.a = '5l'; frame.o = 6;
frame.q = q_3l; frame.dq = dq_3l; frame.ddq = ddq_3l;
frame.param = [m_6l MX_6l MY_6l MZ_6l XX_6l XY_6l XZ_6l YY_6l YZ_6l ZZ_6l];
% f('6l') = frame;
F{7} = frame;

frame.x = sym([-1; 0; 0]); frame.y = [0; -cos(q_4l); -sin(q_4l)];
```

```
frame.z = [0; -sin(q_4l); cos(q_4l)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0];  frame.a = '6l'; frame.o = 7;
frame.q = q_4l; frame.dq = dq_4l; frame.ddq = ddq_4l;
frame.param = [m_7l MX_7l MY_7l MZ_7l XX_7l XY_7l XZ_7l YY_7l YZ_7l ZZ_7l];
% f('7l') = frame;
F{8} = frame;

frame.x = [-cos(q_5l); 0; sin(q_5l)]; frame.y = sym([0; -1; 0]);
frame.z = [sin(q_5l); 0; cos(q_5l)]; frame.P = sym([0; -L8; 0]);
frame.e = [0; -1; 0];  frame.a = '7l'; frame.o = 8;
frame.q = q_5l; frame.dq = dq_5l; frame.ddq = ddq_5l;
frame.param = [m_8l MX_8l MY_8l MZ_8l XX_8l XY_8l XZ_8l YY_8l YZ_8l ZZ_8l];
% f('8l') = frame;
F{9} = frame;

frame.x = sym([-1; 0; 0]); frame.y = [0; -cos(q_6l); -sin(q_6l)];
frame.z = [0; -sin(q_6l); cos(q_6l)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0];  frame.a = '8l'; frame.o = 9;
frame.q = q_6l; frame.dq = dq_6l; frame.ddq = ddq_6l;
frame.param = [m_9l MX_9l MY_9l MZ_9l XX_9l XY_9l XZ_9l YY_9l YZ_9l ZZ_9l];
% f('9l') = frame;
F{10} = frame;

frame.x = [-cos(q_7l); 0; sin(q_7l)]; frame.y = [-sin(q_7l); 0; -cos(q_7l)];
frame.z = sym([0; -1; 0]); frame.P = [0; -L9; 0];
frame.e = [0; 0; -1];  frame.a = '9l';
frame.q = q_7l; frame.dq = dq_7l; frame.ddq = ddq_7l; frame.o = 10;
frame.param = [m_10l MX_10l MY_10l MZ_10l XX_10l XY_10l XZ_10l YY_10l YZ_10l ZZ_10l];
% f('10l') = frame;
F{11} = frame;

frame.x = [0; cos(q_1r); sin(q_1r)]; frame.y = sym([-1; 0; 0]);
frame.z = [0; -sin(q_1r); cos(q_1r)]; frame.P = [-L6; L5; 0];
frame.e = [0; -1; 0];  frame.a = '3';
frame.q = q_1r; frame.dq = dq_1r; frame.ddq = ddq_1r; frame.o = 11;
frame.param = [m_4r MX_4r MY_4r MZ_4r XX_4r XY_4r XZ_4r YY_4r YZ_4r ZZ_4r];
% f('4r') = frame;
F{12} = frame;

frame.x = sym([-1; 0; 0]); frame.y = [0; -cos(q_2r); -sin(q_2r)];
frame.z = [0; -sin(q_2r); cos(q_2r)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0];  frame.a = '4r';
frame.q = q_2r; frame.dq = dq_2r; frame.ddq = ddq_2r; frame.o = 12;
frame.param = [m_5r MX_5r MY_5r MZ_5r XX_5r XY_5r XZ_5r YY_5r YZ_5r ZZ_5r];
% f('5r') = frame;
F{13} = frame;

frame.x = [-cos(q_3r); 0; sin(q_3r)]; frame.y = sym([0; -1; 0]);
frame.z = [sin(q_3r); 0; cos(q_3r)]; frame.P = sym([0; -L7; 0]);
frame.e = [0; -1; 0];  frame.a = '5r';
frame.q = q_3r; frame.dq = dq_3r; frame.ddq = ddq_3r; frame.o = 13;
frame.param = [m_6r MX_6r MY_6r MZ_6r XX_6r XY_6r XZ_6r YY_6r YZ_6r ZZ_6r];
% f('6r') = frame;
F{14} = frame;

frame.x = sym([-1; 0; 0]); frame.y = [0; -cos(q_4r); -sin(q_4r)];
frame.z = [0; -sin(q_4r); cos(q_4r)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0];  frame.a = '6r'; frame.o = 14;
frame.q = q_4r; frame.dq = dq_4r; frame.ddq = ddq_4r; frame.o = 14;
frame.param = [m_7r MX_7r MY_7r MZ_7r XX_7r XY_7r XZ_7r YY_7r YZ_7r ZZ_7r];
% f('7r') = frame;
F{15} = frame;

frame.x = [-cos(q_5r); 0; sin(q_5r)]; frame.y = sym([0; -1; 0]);
frame.z = [sin(q_5r); 0; cos(q_5r)]; frame.P = sym([0; -L8; 0]);
frame.e = [0; -1; 0];  frame.a = '7r'; frame.o = 15;
frame.q = q_5r; frame.dq = dq_5r; frame.ddq = ddq_5r; frame.o = 15;
frame.param = [m_8r MX_8r MY_8r MZ_8r XX_8r XY_8r XZ_8r YY_8r YZ_8r ZZ_8r];
% f('8r') = frame;
F{16} = frame;

frame.x = sym([-1; 0; 0]); frame.y = [0; -cos(q_6r); -sin(q_6r)];
frame.z = [0; -sin(q_6r); cos(q_6r)]; frame.P = sym([0; 0; 0]);
frame.e = [-1; 0; 0];  frame.a = '8r'; frame.o = 16;
frame.q = q_6r; frame.dq = dq_6r; frame.ddq = ddq_6r; frame.o = 16;
frame.param = [m_9r MX_9r MY_9r MZ_9r XX_9r XY_9r XZ_9r YY_9r YZ_9r ZZ_9r];
% f('9r') = frame;
F{17} = frame;

frame.x = [-cos(q_7r); 0; sin(q_7r)]; frame.y = [-sin(q_7r); 0; -cos(q_7r)];
frame.z = sym([0; -1; 0]); frame.P = [0; -L9; 0];
frame.e = [0; 0; -1];  frame.a = '9r';
frame.q = q_7r; frame.dq = dq_7r; frame.ddq = ddq_7r; frame.o = 17;
frame.param = [m_10r MX_10r MY_10r MZ_10r XX_10r XY_10r XZ_10r YY_10r YZ_10r ZZ_10r];
% f('10r') = frame;
F{18} = frame;

keys = {'0', '1', '2', '3', '4l', '5l', '6l', '7l', '8l', ...
    '9l', '10l', '4r', '5r', '6r', '7r', '8r', '9r', '10r'};

for i=1:nFrames+1; keySet{i} = keys{i}; frames{i} = F{i}; end

f = containers.Map( keySet, frames );
```

7.3 kanesLHS()

```
function [K, Q, KHist] = kanesLHS(f, varargin)

syms g real

if nargin == 1
    w0 = sym([0 0 0]');
    v0 = sym([0 0 0]');
    alpha0 = sym([0 0 0]');
    a0 = sym([0 0 0]');
else
    w0 = varargin{1}(:,1);
    v0 = varargin{1}(:,2);
    alpha0 = varargin{1}(:,3);
    a0 = varargin{1}(:,4);
end

dq=dqVec(f);
key = keys(f);
K=sym(zeros(length(dq),1));
KHist=sym(zeros(length(dq),length(dq)));
Q=sym(zeros(length(dq),1));
for i=1:length(key)

    % Do nothing for frame 0
    if(isequal(key{i}, '0')); continue; end

    % Kinematics and Inertials Params
    w=angVel(f, key{i}, w0);
    alpha=angAcc(f, key{i}, [w0 alpha0]);
    v=linVel(f, key{i}, [w0 v0]);
    a=linAcc(f, key{i}, [w0 alpha0 v0 a0]);
    J=inertiaMat(f, key{i});
    m=mass(f, key{i});
    mS=mCOM(f, key{i});
    S=mS/m;

    % Inertial Forces and Torques
    vG = v + cross(w, S);
    maG = m*(a + cross(alpha, S) + cross(w, cross(w, S)));
    dHnew = cross(mS, a) + J*alpha + cross(w, J*w);

    % Kanes LHS contributions
    for j=1:length(K)
        disp(['kanesF: i=', num2str(i), ', j=', num2str(j), ', key=', key{i}]);
        KHist(j,f(key{i}).o) = maG'*diff(v, dq(j)) + dHnew'*diff(w, dq(j));
        K(j) = K(j) + KHist(j,f(key{i}).o);
    end

    % Gravity terms
    T = Tf(f, '0', key{i});
    ROT = T(1:3,1:3);
    mg = m*ROT*[0 0 -g]';
    for j=1:length(Q)
        disp(['kanesQ: i=', num2str(i), ', j=', num2str(j)]);
        Q(j)=Q(j)-mg'*diff(vG, dq(j));
    end
end
```

7.4 getAC()

```
function [A, C] = getAC(f, K)

dq=dqVec(f);
ddq=ddqVec(f);
n=length(dq);
A = sym(zeros(n,n)); C = sym(zeros(n,n));
for i=1:n
    for j=1:n
        A(i,j)=getcoeff(K(i),ddq(j),1);
        C(i,j)=getcoeff(K(i), dq(j),2)*dq(j);
        ccc = getcoeff(K(i),dq(j),1);
        C(i,j) = C(i,j)+ccc;
        for k=1:n
            disp(['getAC: i=', num2str(i), ', j=', num2str(j), ', k=', num2str(k)]);
            C(i,j) = C(i,j) - 0.5*(getcoeff(ccc,dq(k),1))*dq(k);
        end
    end
end

% A=simplify(A);
% C=simplify(C);
```

7.5 angVel()

```
function w = angVel(f, key, varargin)

% Calculate angular velocity of the current frame represented in the same
% frame.
% 'f' is the map container of the information for all the frames
% 'key' identifies the frame whose angular veclocity is demanded
% One more argument can define non-zero angular velocity of the base frame

disp(['    Computing angVel of ', key]);


if(f(key).gotAngVel)
    w=f(key).angVel;
else

    if(nargin == 2)
        w0 = [0; 0; 0];
    else w0 = varargin{1};
    end

    if(isequal(f(key).a, '0')) % if it's the first link in the chain
        w = Rot(f, f(key).a, key) * w0 + f(key).e*f(key).dq;
    else % recursive call if the frame is not the first link in the chain
        w = Rot(f, f(key).a, key) * angVel(f, f(key).a) + f(key).e*f(key).dq;
    end

    frame = f(key);
    frame.angVel = w;
    frame.gotAngVel = true;
    f(key) = frame;
end

disp(['    Computed angVel of ', key]);
```

7.6 angAcc()

```
function alpha = angAcc(f, key, varargin)

% Calculate angular acceleration of the current frame represented in the same
% frame.
% 'f' is the map container of the information for all the frames
% 'key' identifies the frame whose angular acceleration is demanded
% One more argument can define non-zero angular velocity of the base frame

disp(['    Computing angAcc of ', key]);

if(f(key).gotAngAcc)
    alpha=f(key).angAcc;
else

    if(nargin == 2)
        w0 = [0; 0; 0];
        alpha0 = [0; 0; 0];
    else
        w0 = varargin{1}(:,1);
        alpha0 = varargin{1}(:,2);
    end

    % Angular acceleration of antecendent frame
    if(isequal(f(key).a, '0')) % if it's the first link in the chain
        alpha_a = Rot(f, f(key).a, key) * alpha0;
    else % recursive call if the frame is not the first link in the chain
        alpha_a = Rot(f, f(key).a, key) * angAcc(f, f(key).a, [w0 alpha0]);
    end

    % Angular acceleration of the current frame
    alpha = alpha_a + f(key).e * f(key).ddq ...
        + f(key).dq * cross(angVel(f, key, w0), f(key).e);

    frame = f(key);
    frame.angAcc = alpha;
    frame.gotAngAcc = true;
    f(key) = frame;
end

disp(['    Computed angAcc of ', key]);
```

7.7 linVel()

```
function V = linVel(f, key, varargin)

% Calculate linear velocity of the current frame measured in the world
```

```
% frame represented in the current frame
% 'f' is the map conatainer containing the information of the frames of
% the robot
% 'key' identifies the current frame
% Optional arguments:
% 'w0' is the angular velocity of the base frame
% 'V0' is the linear velocity of the base frame

disp(['   Computing linVel of ', key]);

if(f(key).gotLinVel)
    V=f(key).linVel;
else

    if(nargin == 2)
        w0 = [0;0;0];
        V0= [0;0;0];
    else
        w0 = varargin{1}(:,1);
        V0 = varargin{1}(:,2);
    end

    if(isequal(f(key).a, '0')) % if it's the first link in the chain
        V = Rot(f, f(key).a, key) * (V0 + cross(w0, f(key).P));
    else % if it's some other link
        V = Rot(f, f(key).a, key) * (linVel(f, f(key).a) + ...
            cross(angVel(f, f(key).a), f(key).P));
    end

    frame = f(key);
    frame.linVel = V;
    frame.gotLinVel = true;
    f(key) = frame;

end

disp(['   Computed linVel of ', key]);
```

7.8 linAcc()

```
function a = linAcc(f, key, varargin)

% Calulate linear velocity of the current frame measured in the world
% frame represented in the current frame
% 'f' is the map conatainer containing the information of the frames of
% the robot
% 'key' identifies the current frame
% Optional arguments:
% 'w0' is the angular velocity of the base frame
% 'V0' is the linear velocity of the base frame

disp(['   Computing linAcc of ', key]);

if(f(key).gotLinAcc)
    a=f(key).linAcc;
else

    if(nargin == 2)
        w0 = [0;0;0];
        alpha0 = [0;0;0];
        V0= [0;0;0];
        a0 = [0;0;0];
    else
        w0 = varargin{1}(:,1);
        alpha0 = varargin{1}(:,2);
        V0 = varargin{1}(:,3);
        a0 = varargin{1}(:,4);
    end

    % Vel/Acc of antecedent frame represented in antedecent frame
    if(isequal(f(key).a, '0')) % if it's the first link in the chain
        w_a = w0;
        alpha_a = alpha0;
        a_a = a0;
    else % if it's some other link
        w_a = angVel(f, f(key).a, w0);
        alpha_a = angAcc(f, f(key).a, [w0 alpha0]);
        a_a = linAcc(f, f(key).a, [w0 alpha0 V0 a0]);
    end

    % Acceleration of the current frame
    a = Rot(f, f(key).a, key) * (a_a + cross(alpha_a, f(key).P) ...
        + cross(w_a, cross(w_a, f(key).P)));

    frame = f(key);
    frame.linAcc = a;
    frame.gotLinAcc = true;
    f(key) = frame;
```

```
end
disp(['   Computed linAcc of ', key]);
```

7.9 Rot()

```
function A = Rot(f, key1, key2)

% Find the rotation transform of a frame
% f denotes the map container that contains information of the frames.
% key1 identifies the frame1 that is being represented
% key2 identifies frame2 in which frame1 is being represented
% This function assumes frame1 immediately follows frame2 or frame2
% immediately follows frame1

if(isequal(f(key1).a, key2)) % if frame2 is the antecedent of frame1
    A = [f(key1).x f(key1).y f(key1).z];
elseif(isequal(f(key2).a, key1)) % if frame1 is the antecedent of frame2
    A = [f(key2).x'; f(key2).y'; f(key2).z'];
end
```

7.10 Tf()

```
function T = Tf(f, key1, key2)

% Find the transform of a frame
% f denotes the map container that contains information of the frames.
% key1 identifies the frame1 that is being represented
% key2 identifies frame2 in which frame1 is being represented

if(isBefore(f, key2, key1)) % if frame2 is before frame1 in the chain
    T = [f(key1).x f(key1).y f(key1).z f(key1).P; 0 0 0 1];
    key = f(key1).a;
    while(~isequal(key, key2))
        T = [f(key).x f(key).y f(key).z f(key).P; 0 0 0 1] * T;
        key = f(key).a;
    end
elseif(isBefore(f, key1, key2)) % if frame1 is before frame2 in the chain
    T = [f(key2).x' -f(key2).x'*f(key2).P; ...
         f(key2).y' -f(key2).y'*f(key2).P; ...
         f(key2).z' -f(key2).z'*f(key2).P; ...
         0 0 0 1];
     key = f(key2).a;
     while(~isequal(key, key1))
         T = T * [f(key).x' -f(key).x'*f(key).P; ...
              f(key).y' -f(key).y'*f(key).P; ...
              f(key).z' -f(key).z'*f(key).P; ...
              0 0 0 1];
         key = f(key).a;
     end
end
```

7.11 mass()

```
function m = mass(f, key)

param = f(key).param;
m = param(1);
```

7.12 mCOM()

```
function MS = mCOM(f, key)

param = f(key).param;
MS = param(2:4)';
```

7.13 inertiaMat()

```
function J = inertiaMat(f, key)

param = f(key).param;
J = param([5 6 7; 6 8 9; 7 9 10]);
```

7.14 qVec()

```
function q = qVec(f)

% Generate q vector in the order defined by member 'o' of the frames
% contained in 'f'

key = keys(f);
n = length(key)-1;
q = sym(zeros(n, 1));
for i=1:length(key)
    if(isequal(key{i}, '0')); continue; end
    q(f(key{i}).o) = f(key{i}).q;
end
```

7.15 dqVec()

```
function dq = dqVec(f)

% Generate dq vector in the order defined by member 'o' of the frames
% contained in 'f'

key = keys(f);
n = length(key)-1;
dq = sym(zeros(n, 1));
for i=1:length(key)
    if(isequal(key{i}, '0')); continue; end
    dq(f(key{i}).o) = f(key{i}).dq;
end
```

7.16 ddqVec()

```
function ddq = ddqVec(f)

% Generate ddq vector in the order defined by member 'o' of the frames
% contained in 'f'

key = keys(f);
n = length(key)-1;
ddq = sym(zeros(n, 1));
for i=1:length(key)
    if(isequal(key{i}, '0')); continue; end
    ddq(f(key{i}).o) = f(key{i}).ddq;
end
```

7.17 getcoeff()

```
function c = getcoeff(P, x, a)

% in the symbolic expression P return the coefficient of the term x^a

[C, T] = coeffs(P, x);
n=length(C);
exists = 0;
for i=1:n
    if(isequal(T(i),x^a));
        exists = 1;
        break;
    end
end

if(exists)
    c = C(i);
else
    c = 0;
end
```

7.18 kanesRHS.m

```
clear all
load kanesAndLagrange_03_27_15.mat

% creating a new f with symbolic variables for torques in it
% tau_imu = -tau_l - tau_r (sum of reaction of torques applied to wheels)
syms tau_imu tau_w tau_torso real
syms tau_1l tau_2l tau_3l tau_4l tau_5l tau_6l tau_7l real
syms tau_1r tau_2r tau_3r tau_4r tau_5r tau_6r tau_7r  real

tauVec = [tau_imu tau_w tau_torso ...
    tau_1l tau_2l tau_3l tau_4l tau_5l tau_6l tau_7l ...
    tau_1r tau_2r tau_3r tau_4r tau_5r tau_6r tau_7r]';
key = keys(f);
for i=1:length(key)
    if(isequal(key{i}, '0')); continue; end
    frame = f(key{i});
    frame.tau = tauVec(frame.o);
    f(key{i}) = frame;
end

% Finding the contribution of body motor torques to kane's RHS
dq = dqVec(f);
a = sym(zeros(length(dq),1));
aa = sym(zeros(length(dq),length(dq)));
for i=1:length(key)

    % if key is '0' leave it
    if(isequal(key{i}, '0')); continue; end

    % Collect relevant info of current frame and previous frame
    frame = f(key{i});
    e = frame.e;
    tau = frame.tau;
    w = angVel(f, key{i}, sym([0 0 0]'));
    R = Rot(f, key{i}, frame.a);
    wlast = angVel(f, frame.a, sym([0 0 0]'));

    % Calculate the contribution
    for j=1:length(dq)
        aa(j,frame.o) = tau*e'*diff(w,dq(j)) + (-R*tau*e)'*diff(wlast,dq(j));
        a(j) = a(j) + aa(j,frame.o);
    end
end

% Contribution of forces/torques on the endeffector

syms Flx Fly Flz Frx Fry Frz Tlx Tly Tlz Trx Try Trz real
Fl = [Flx Fly Flz]'; Tl = [Tlx Tly Tlz]'; % expressed in frame 10l
Fr = [Frx Fry Frz]'; Tr = [Trx Try Trz]'; % expressed in frame 10r

b1 = sym(zeros(length(dq),1));
b2 = sym(zeros(length(dq),1));
v = linVel(f, '10l', sym([0 0 0]'));
w = angVel(f, '10l', sym([0 0 0]'));
for i=1:length(dq)
    b1(i) = b1(i) + Fl'*diff(v, dq(i));
    b2(i) = b2(i) + Tl'*diff(w, dq(i));
end

b3 = sym(zeros(length(dq),1));
b4 = sym(zeros(length(dq),1));
v = linVel(f, '10r', sym([0 0 0]'));
w = angVel(f, '10r', sym([0 0 0]'));
for i=1:length(dq)
    b3(i) = b3(i) + Fr'*diff(v, dq(i));
    b4(i) = b4(i) + Tr'*diff(w, dq(i));
end

% Contributions from the forces of gravity

syms g real
c = sym(zeros(length(dq),1));

for i=1:length(key)

    % if key is '0' leave it
    if(isequal(key{i}, '0')); continue; end

    % Get necessary info
    w = angVel(f, key{i}, sym([0 0 0]'));
    v = linVel(f, key{i}, [sym([0 0 0]') sym([0 0 0]')]);
    m = mass(f, key{i});
    S = mCOM(f, key{i})/m;
    T = Tf(f, '0', key{i});

    % Calculate contribution
    vG = v + cross(w, S);
    ROT = T(1:3,1:3);
    mg = m*ROT*[0 0 -g]';
    for j=1:length(dq)
        c(j)= c(j) + mg'*diff(vG, dq(j));
    end
```

```
end
```