Equations Of Motion of Krang on Fixed Wheels

Munzir Zafar

March 26, 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 Lagrange formulation with a systematic approach presented in [1] to derive the equations of motion.

1 Introduction to Lagrange Formulation

The Lagrange formulation describes the behavior of a dynamic system in terms of work and energy stored in the system. The Lagrange equations are commonly written in the form:

$$\Gamma_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} \quad for i \in \mathbb{F} \tag{1}$$

where L is the Lagrangian of the robot defined as the difference berween the kinetic energy E and potential energy U of the system:

$$L = E - U$$

1.1 General Form of the Dynamic Equations

The kinetic energy of the system is a quadratic function in the joint velocities such that:

$$E = \frac{1}{2}\dot{\mathbf{q}}^{\mathbf{T}}\mathbf{A}\dot{\mathbf{q}} \tag{2}$$

where $\mathbf{A}$ is the $n \times n$ symmetric and positive definite *inertia matrix* of the robot. Its elements are functions of the joint positions. The (i, j) element of $\mathbf{A}$

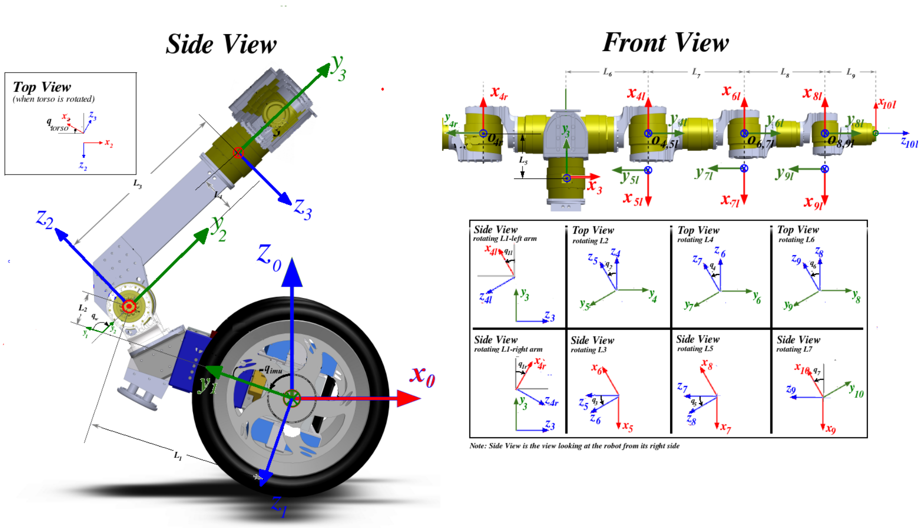

Figure 1: Frames of references on the robot

is denoted A_{ij}. Since the potential energy is a function of the joint positions, equation 1 leads to:

$$\mathbf{\Gamma} = \mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{Q}(\mathbf{q}) \tag{3}$$

where:

- $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ is the $n \times 1$ vector of Coriolis and centrifugal torques, such that: $\mathbf{C}\dot{\mathbf{q}} = \dot{\mathbf{A}}\dot{\mathbf{q}} - \frac{\partial E}{\partial \mathbf{q}}$
- $\mathbf{Q} = \begin{bmatrix} Q_1 & Q_2 & Q_3 & Q_{4l} & \dots & Q_{10l} & Q_{4r} & \dots & Q_{10r} \end{bmatrix}^T$ is the vector of gravity torques

So the dynamic model of a tree-structured robot is described by n coupled and nonlinear second order differential equations. The elements of $\mathbf{A}$, $\mathbf{C}$ and $\mathbf{Q}$ are functions of geometric and inertial parameters of the robot.

1.2 Computation of the elements of A, C and Q

To compute the elements of $\mathbf{A}$, $\mathbf{C}$ and $\mathbf{Q}$, we begin by symbollically computing the expressions of the kinetic and potential energies of all the links of the robot. Then we proceed as follows:

- the elements A_{ii} is equal to the coefficient of $\left(\frac{\dot{q}_i^2}{2}\right)$ in the expression of the kinetic energy, while A_{ij}, for $i \neq j$, is equal to the coefficient of $\dot{q}_i\dot{q}_j$
- for calculating the elements of $\mathbf{C}$, there exist several forms of the vector $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$. Using the *Christoffell symbols* $c_{i,jk}$, the (i,j) elements of the matrix $\mathbf{C}$ can be written as:

$$\begin{cases} C_{ij} = \sum\limits_{k=1}^{n} c_{i,jk}\dot{q}_k \\ c_{i,jk} = \frac{1}{2}\left[\frac{\partial A_{ij}}{\partial q_k} + \frac{\partial A_{ik}}{\partial q_j} - \frac{\partial A_{jk}}{\partial q_i}\right] \end{cases} \tag{4}$$

- The Q_i element of the vector $\mathbf{Q}$ is calculated according to:

$$Q_i = \frac{\partial U}{\partial q_i} \tag{5}$$

2 Finding A, C and Q for our robot

In this section we determine the symbolic expression for the total kinetic energy E of the robot.

2.1 Transformations

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

$${}^iT_j = \begin{bmatrix} {}^is_j & {}^in_j & {}^ia_j & {}^iP_j \end{bmatrix} = \begin{bmatrix} & {}^iA_j & & {}^iP_j \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & n_x & a_x & P_x \\ s_y & n_y & a_y & P_y \\ s_z & n_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

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:

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

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

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

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

2.2 Angular and Linear Velocities of Frames

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

$$ {}^j\omega_j = {}^jA_i {}^i\omega_i + {}^je_j\dot{q}_j \tag{7} $$
$$ {}^jV_j = {}^jA_i \left({}^iV_i + {}^i\omega_i \times {}^iP_j\right) \tag{8} $$

where ${}^i\omega_j$ and ${}^iV_j$ denote the angular and linear velocities 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.

$$ {}^1e_1 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^2e_2 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^3e_3 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, {}^4e_4 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, $$
$$ {}^5e_5 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^6e_6 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, {}^7e_7 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^8e_8 = \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, $$
$$ {}^9e_9 = \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, {}^{10}e_{10} = \begin{bmatrix}0 & 0 & -1\end{bmatrix}^T $$

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

2.3 Kinetic Energy

The kinetic energy of the robot is given as:

$$ E = \sum_{j \in \mathbb{F}} E_j \tag{9} $$

where E_j denotes the kinetic energy of link j, which can be computed by

$$ E_j = \frac{1}{2}(\omega_j^T I_{Gj}\omega_j + M_j V_{Gj}^T V_{Gj}) \tag{10} $$

where the velocity of the center of mass can be expressed as:

$$ V_{Gj} = V_j + \omega_j \times S_j $$

and since:

$$ J_j = I_{Gj} - M_j\hat{S}_j\hat{S}_j $$

equation 10 becomes:

$$ E_j = \frac{1}{2}(\omega_j^T J_{Gj}\omega_j + M_j V_j^T V_j + 2\mathbf{MS}_j^T(V_j \times \omega_j)) \tag{11} $$

See section A in the appendix to know the details of the derivation.

2.4 Potential Energy

The total potential energy U of the robot is given by:

$$U = \sum_{j \in \mathbb{F}} U_j = \sum_{j \in \mathbb{F}} -M_j \mathbf{g}^T (L_{0,j} + S_j) \tag{12}$$

where $L_{0,j}$ is the position vector from the origin O_0 to O_j and $\mathbf{g}$ is the gravitatonal acceleration. Projecting the vectors appearing in 12 into frame R_0, we obtain:

$$\begin{aligned} U_j &= -M_j \, {}^0\mathbf{g}^T ({}^0P_j + {}^0A_j \, {}^jS_j) && (13) \\ &= -{}^0\mathbf{g}^T (M_j \, {}^0P_j + {}^0A_j \, {}^j\mathbf{MS}_j) && (14) \\ &= -\begin{bmatrix} {}^0\mathbf{g}^T & 0 \end{bmatrix} {}^0T_j \begin{bmatrix} {}^j\mathbf{MS}_j \\ M_j \end{bmatrix} && (15) \end{aligned}$$

Given the frames defined in figure 1, ${}^0\mathbf{g} = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T$.

3 Effects of forces and torques on the end effectors

The well known relationship between joint torques and end effector forces on a simple serial robot is:

$$\mathbf{\Gamma} = \mathbb{J}_n^T \mathbb{f}_{e@n}$$

where

- $\mathbf{\Gamma}$ is the vector of torques of the individual joints in the chain
- $\mathbb{f}_{e@n} = \begin{bmatrix} f_{e@n} \\ \tau_{e@n} \end{bmatrix}$ is the wrench applied by the robot at the origin of the nth frame (i.e. the last link in the chain which has the end-effector mounted on it). This wrench is usually represented in frame R_n or in the world frame R_0 denoted as ${}^n\mathbb{f}_{e@n}$ or ${}^0\mathbb{f}_{e@n}$ respectively.
- $\mathbb{J}_n$ is $6 \times n$ Jacobian matrix of the robot calculated using:

  $$\mathbb{J}_n = \begin{bmatrix} e_1 \times L_{1,n} & ... & e_n \times L_{n,n} \\ e_1 & ... & e_n \end{bmatrix}$$

  where e_j denotes the unit vectors along the local angular velocities of the frame j and $L_{j,n}$ is the position vector from O_j to O_n. These vectors are expressed in the same frame as the wrench $\mathbb{f}_{e@n}$. So for ${}^0\mathbb{f}_{e@n}$ all vectors in the Jacobian matrix will be expressed in frame 0 and the Jacobian will be denoted as ${}^0\mathbb{J}_n$. Similarly for ${}^n\mathbb{f}_{e@n}$ the Jacobian will be denoted ${}^n\mathbb{J}_n$.

3.1 Jacobians for the two-armed robot

For the case of krang, we will have two wrenches $\mathbb{f}_{el@10l}$ and $\mathbb{f}_{er@10r}$ applied at two end-effectors on the right and the left arms respectively. As previously el and er are identifying the wrench and $10l$ and $10r$ are idenfying the frames at whose origin the wrenches are being applied. The joint torques will now be calculated using the equation:

$$\mathbf{\Gamma} = \mathbb{J}_{10l}^T \mathbb{f}_{el@10l} + \mathbb{J}_{10r}^T \mathbb{f}_{er@10r} \tag{16}$$

where

- $\mathbf{\Gamma} = \begin{bmatrix} \tau_1 & \tau_2 & \tau_3 & \tau_{4l} & \dots & \tau_{10l} & \tau_{4r} & \dots & \tau_{10r} \end{bmatrix}^T$
- $\mathbb{J}_{10l} = \begin{bmatrix} e_1 \times L_{1,10l} & e_2 \times L_{2,10l} & e_3 \times L_{3,10l} & e_{4l} \times L_{4l,10l} & \dots & e_{10l} \times L_{10l,10l} & O_{3\times7} \\ e_1 & e_2 & e_3 & e_{4l} & \dots & e_{10l} & O_{3\times7} \end{bmatrix}$
- $\mathbb{J}_{10r} = \begin{bmatrix} e_1 \times L_{1,10r} & e_2 \times L_{2,10r} & e_3 \times L_{3,10r} & O_{3\times7} & e_{4r} \times L_{4r,10r} & \dots & e_{10r} \times L_{10r,10r} \\ e_1 & e_2 & e_3 & O_{3\times7} & e_{4r} & \dots & e_{10r} \end{bmatrix}$

4 Other terms in the Lagrange Equations

4.1 Considering Friction

The most often employed model for friction is composed of Coulomb friction together with viscous friction. Therefor, the friction torque at joint i is written as:

$$\Gamma_{fi} = F_{ci} sign(\dot{q}_i) + F_{vi}\dot{q}_i$$

To take into account the friction in the dynamic model of a robot we add the vector $\mathbf{\Gamma}_f$ to the right side of the Lagrange equation (i.e. the vector of generalized forces), such that:

$$\mathbf{\Gamma}_f = \mathbf{diag}(\dot{\mathbf{q}})\mathbf{F_v} + \mathbf{diag}[\mathbf{sign}(\dot{\mathbf{q}}]\mathbf{F_c} \tag{17}$$

where

- $\mathbf{F}_v = \begin{bmatrix} F_{v1} & F_{v2} & F_{v3} & F_{v4l} & \dots & F_{v10l} & F_{v4r} & \dots & F_{v10r} \end{bmatrix}^T$
- $\mathbf{F}_c = \begin{bmatrix} F_{c1} & F_{c2} & F_{c3} & F_{c4l} & \dots & F_{c10l} & F_{c4r} & \dots & F_{c10r} \end{bmatrix}^T$
- $\mathbf{diag}(\dot{\mathbf{q}}) is the diagonal matrix whose elements are the components of \dot{\mathbf{q}}$

4.2 Considering rotor inertia

The kinetic energy of the rotor (and transmission system) and actuator j, is given by the expression $\frac{1}{2} I_{aj} \dot{q}_j^2$. The inertial parameter I_{aj} denotes the equivalent inertia referred to the joint velocity. It is given by:

$$I_{aj} = N_j^2 J_{mj} \tag{18}$$

where J_{mj} is the moment of inertia of the rotor and transmissions of actuator j, N_j is the transmission ratio of the joint axis, equal to $\frac{\dot{q}_{mj}}{\dot{q}_j}$ where $\dot{q}_{mj}$ denotes the rotor velocity of actuator j. In the case of a prismatic joint, I_{aj} is an equivalent mass.

In order to consider the rotor inertia in the dynamic model of the robot, we add the inertia (or mass) I_{aj} to the A_{jj} element of the matrix $\mathbf{A}$.

5 MATLAB code

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/Lagrange*. The function populates the frame information in a map container using `getKrangFrames()`, then calculates the total kinetic energy, total potential energy, then the matrices **A**, **C** and **Q** using functions `totalKE()`, `totalPE()`, `findA()`, `findC()` and `findQ()`. The resulting matrices are saved in the symbolic variables `AA`, `CC` and `QQ` which can be loaded in the workspace by loading the mat file `Lagrange.mat`.

5.1 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 $[m \quad \mathbf{MX} \quad \mathbf{MY} \quad \mathbf{MZ} \quad \mathbf{XX} \quad \mathbf{YY} \quad \mathbf{ZZ} \quad \mathbf{XY} \quad \mathbf{XZ} \quad \mathbf{YZ}]^T$

5.2 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

`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. 7

`linVel(f, key)` returns the symbolic expression for the linear velocity ($\bar{v}$) of the joint represented in local frame calculated recursively using eq. 8

5.3 Main functions

The main functions that are used in the script to find the dynamic model are briefly explained here:

`totalKE()` Finds the symbolic expression for the total kinetic energy of the system. This is done by calling function `KE()` for every joint which in turn uses the eq.11. The quantities such as inertia, velocities etc. needed for evaluating the kinetic energy expression are supplied by the helper functions dicussed in the previous section

`totalPE()` Finds the symbolic expression for the total potential energy of the system. This is done by calling function `PE()` for every joint which in turn uses the eq.15 to find out the potential energy

`findA()` This function uses the method as described in section 1.2 to find the $\mathbf{A}$. The elements A_{ii} is equal to the coefficient of $\left(\frac{\dot{q}_i^2}{2}\right)$ in the expression of the kinetic energy, while A_{ij}, for $i \neq j$, is equal to the coefficient of $\dot{q}_i\dot{q}_j$

`findC()` This function uses the equation 4 to evaluate the symbolic expressions for the elements of $\mathbf{C}$ matrix `findQ()` This function uses the equation 5 to evaluate the symbolic expressions for the elements of **Q** matrix

References

[1] Wisama Khalil and Etienne Dombre. *Modeling, identification and control of robots*. Butterworth-Heinemann, 2004.

A Expression for Kinetic Energy

We show here how the equation 11 was derived from 10. Equation 10 is:

$$E_j = \frac{1}{2}(\omega_j^T I_{Gj}\omega_j + M_j V_{Gj}^T V_{Gj}) \tag{19}$$

where the velocity of the center of mass can be expressed as:

$$V_{Gj} = V_j + \omega_j \times S_j$$

and since:

$$J_j = I_{Gj} - M_j \hat{S}_j \hat{S}_j^T$$

So equation 19 becomes:

$$\begin{aligned}
E_j &= \frac{1}{2}(\omega_j^T (J_j + M_j\hat{S}_j\hat{S}_j)\omega_j + M_j(V_j + \omega_j \times S_j)^T(V_j + \omega_j \times S_j)) \\
E_j &= \frac{1}{2}(\omega_j^T J_j\omega_j + M_j V_j^T V_j + \omega_j^T M_j\hat{S}_j\hat{S}_j\omega_j + M_j V_j^T(\omega_j \times S_j) \\
&\qquad + M_j(\omega_j \times S_j)^T V_j + M_j(\omega_j \times S_j)^T(\omega_j \times S_j))
\end{aligned}$$

Noting that the last term:

$$\begin{aligned}
M_j(\omega_j \times S_j)^T(\omega_j \times S_j) &= (-)(-)M_j(S_j \times \omega_j)^T(S_j \times \omega_j) \\
&= M_j(\hat{S}_j\omega_j)^T(\hat{S}_j\omega_j) \\
&= M_j\omega_j^T\hat{S}_j^T\hat{S}_j\omega_j \\
&= -M_j\omega_j^T\hat{S}_j\hat{S}_j\omega_j
\end{aligned}$$

cancels out the third term. And noting that the fourth and fifth terms are equal, we are left with:

$$E_j = \frac{1}{2}(\omega_j^T J_j\omega_j + M_j V_j^T V_j + 2M_j(\omega_j \times S_j)^T V_j)$$

The last term in the above expression can be simplified as follows:

$$\begin{aligned}
M_j(\omega_j \times S_j)^T V_j &= M_j(\hat{\omega}_j S_j)^T V_j \\
&= M_j S_j^T \hat{\omega}_j^T V_j \\
&= -M_j S_j^T \hat{\omega}_j V_j \\
&= -M_j S_j^T(\omega_j \times V_j) \\
&= \mathbf{MS}_j^T(V_j \times \omega_j)
\end{aligned}$$

so we end up with:

$$E_j = \frac{1}{2}(\omega_j^T J_{Gj}\omega_j + M_j V_j^T V_j + 2\mathbf{MS}_j^T(V_j \times \omega_j))$$

B MATLAB Code Listings

In this section we present the code for the MATLAB functions discussed in the report.

B.1 dynamicModel.m

```
% author: Munzir Zafar
% date: Aug 2, 2014
% brief: Finding the dynamic model of krang

f = getKrangFrames(17);
E = totalKE(f);
U = totalPE(f);
A = findA(f, E);
C = findC(f, A);
Q = findQ(f, U);
```

B.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 );
```

B.3 totalKE()

```
function E = totalKE(f, varargin)

% Calculate the expression for the total KE of the robot
% 'f' is the map container containing the frames information of the robot

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

key = keys(f);
E = sym(0);
for i=1:length(key)
    if(~isequal(key{i}, '0'))
     E = E + KE(f, key{i}, w0, V0);
    end
end
```

B.4 totalPE()

```
function U = totalPE(f)

% Calculate expression for the total potential energy of the system

key = keys(f);
U=sym(0);
for i=1:length(key)
    if(~isequal(key{i}, '0'))
        U = U + PE(f, key{i});
    end
end
```

B.5 findA()

```
function A = findA(f, E)

% Compute the matrix A from the kinetic energy expression
% 'f' contains map of the frames
% 'E' contains the symbolic expression for the kinetic energy

% Declaring an empty A matrix
key = keys(f);
A=sym(zeros(length(key)-1, length(key)-1));

dq=dqVec(f);

for i=1:length(dq)
    A(i,i)=2*getcoeff(E, dq(i), 2);
    coeffdq_i = getcoeff(E, dq(i), 1);
    for j=i+1:length(dq)
        A(i,j)=getcoeff(coeffdq_i, dq(j), 1);
        A(j,i)=A(i,j);
    end
end
```

B.6 findC()

```
function C = findC(f, A)

key = keys(f);
n = length(key)-1;

% Get partial derivatives of A_ij wrt q_k and store them in dA
dA = sym(zeros(n, n, n));
q = qVec(f); dq = dqVec(f);
for i=1:n
    for j=1:n
        for k=1:n
            dA(i, j, k) = diff(A(i, j), q(k));
        end
    end
end

% Find Christoffel coefficients c_ijk and use them to find C_ij
c = sym(zeros(n, n, n));
C = sym(zeros(n, n));
for i=1:n
    for j=1:n
        for k=1:n
            c(i, j, k) = 0.5*(dA(i, j, k) + dA(i, k, j) - dA(j, k, i));
            C(i, j) = C(i, j) + c(i, j, k)*dq(k);
        end
    end
end
```

B.7 findQ()

```
function Q = findQ(f, U)

% Compute the symbolic expression for Q

q = qVec(f);
Q = sym(zeros(length(q), 1));
for i=1:length(q)
    Q(i) = diff(U, q(i));
end
```

B.8 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 nonzero angular velocity of the base frame

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
```

B.9 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

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

B.10 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 assures 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
```

B.11 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
```

B.12 mass()

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

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

B.13 mCOM()

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

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

B.14 inertiaMat()

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

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

B.15 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
```

B.16 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
```

B.17 getcoeff()

```
function c = getcoeff(P, 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
```