

MATLAB Tool

proNEu Documentation

Derivation of Analytical Kinematics & Dynamics

keywords: *dynamics, global kinematics, analytical
equations of motion, projected Newton-Euler, simple
MATLAB tool*

v1.1, Mar. 2012

Preamble

This is the manual for the (very simple, but still quite powerfull) Matlab tool *proNEu*. The tool uses the MATLAB Symbolic Math Toolbox¹ to derive the analytical global kinematics and equations of motion based on projected Newton-Euler methods. In Section 1, a short summary about the theory (kinematics and dynamics) is given in combination with an outline of the implementation in MATLAB.

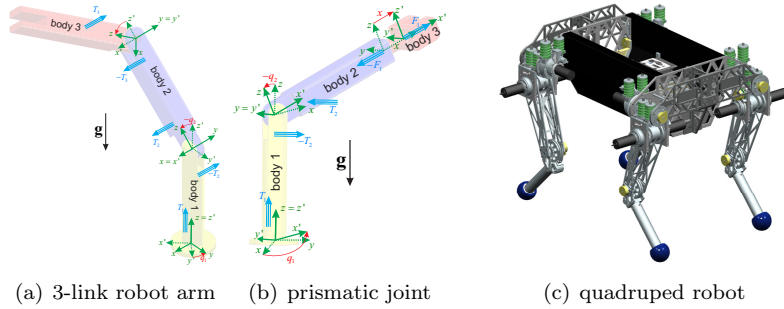


Figure 1: Three examples illustrating the use of this toolbox (Section 3).

Several examples (documented in Section 3 as well as in the source files) highlight how this tool has to be used. The user chooses the generalized coordinates, actuator and link parameters before setting up a very simple kinematic tree of the entire system. The global kinematics and the equations of motion are symbolically derived and the user can visually check the robot configuration. In the examples it is outlined, how the user can finally get function files, compiled mex-function, or C-code that can be used or embedded in any simulation environment.

We intentionally kept this tool very simple for several reasons: First, there exist (often commercially available) complex tools that are often a large overkill for most applications. We do not want to have a sophisticated user front-end that allows adjusting everything without seeing behind the scenes, instead we appreciate tools where we can adapt everything for our needs. In our specific case, we use this tool mostly for model based controllers, where we require to get the dynamics fast and efficiently in simulations or on the actual hardware.

¹<http://www.mathworks.ch/products/symbolic/>

date:	Mar. 2012
authors:	Christian Gehring
version:	1.1
info:	minor fixes and improvements of the documentation, changed color of coordinate systems and added a world frame to plotBodies()
date:	Dec. 2011
authors:	Marco Hutter Christian Gehring
version:	1.0
info:	First appearance of <i>proNEu</i>

Table 1: Revisions

utils/	utility folder
.../computePNE.m	core file for global dynamics and projected Newton-Euler equations
.../dMATdt.m	full differentiation
.../eulerToRotMat_A_IB.m	rotation matrix from B to I frame, x-y-z definition
.../eulerToRotMat_A_BI.m	rotation matrix from I to B frame, x-y-z definition
.../plotBodies.m	visualization of (global) kinematic tree
.../skew.m	get skewing a matrix from vector
.../unskew.m	skew matrix to vector
examples/	example folder
.../QuadrupedFreeFloat/genEoM.m	free floating quadruped Star $LETH$
.../RA3Link/genEoM.m	robot arm with 3 links and 3 revolute joints
.../RA3LinkPrismatic/genEoM.m	robot arm with 3 links and 1 prismatic joint
.../GenerateCode/createFunctionFiles.m	generating matlab and functions
.../GenerateCode/genCCodeMatrix.m	generate c-code of marix function
.../GenerateCode/genCCodeVariables.m	define parameters for c-code
.../GenerateCode/genCCodeExampleFile.m	example for c-code

Table 2: Software file content

Contents

Preamble	i
1 Theoretical Background and Notation	1
1.1 Kinematics	1
1.1.1 Generalized Coordinates	1
1.1.2 Position Vector	1
1.1.3 Velocity (in Moved Systems)	2
1.1.4 Rotation	2
1.1.5 Angular Velocity	3
1.1.6 Jacobian	3
1.2 Dynamics	4
1.2.1 Projected Newton-Euler Equations	4
2 Matlab Tool	5
2.1 Kinematic Tree	5
2.2 Force Elements	6
2.2.1 Torque Actuators	6
2.2.2 Force Actuators	6
2.3 Projected Newton-Euler Equations	7
3 Examples	9
3.1 3-link Robot Arm	9
3.2 3-link Robot Arm with Prismatic Joint	13
3.3 Quadruped Robot Starl <i>ETH</i>	17
3.4 Generating Code	24
3.4.1 Generating Matlab Functions and Mex Functions	24
3.4.2 Generating C-Code	24

Chapter 1

Theoretical Background and Notation

1.1 Kinematics

This section gives a compact overview about the notation for the kinematic and dynamic representation. Each part is accompanied by some code snippets that should highlight how this works in Matlab. Note: these code parts DO NOT belong to a specific example. For complete examples check Section 3.

1.1.1 Generalized Coordinates

We use generalized coordinates \mathbf{q} that can contain in the case of free floating bodies in addition to the joint coordinates \mathbf{q}_r , also un-actuated base coordinates \mathbf{q}_b :

$$\mathbf{q} = \begin{pmatrix} \mathbf{q}_b \\ \mathbf{q}_r \end{pmatrix}. \quad (1.1)$$

```
1 % define generalized coordinates
2 syms x q1 q2 real
3 q = [x,q1,q2]';
4 % define generalized velocities
5 syms Dx Dq1 Dq2 real
6 dq = [Dx,Dq1,Dq2]';
```

1.1.2 Position Vector

A (position) vector from point O to P as a function of generalized coordinates \mathbf{q} expressed in frame B :

$${}^B\mathbf{r}_{OP} = {}^B\mathbf{r}_{OP}(\mathbf{q}). \quad (1.2)$$

```
1 % define certain parameters
2 syms l real
3 % position vector
4 r = [l*sin(q1),l*cos(q1),0];
```

1.1.3 Velocity (in Moved Systems)

The velocity is given through differentiation, whereby special attention is required when differentiating in a moving (with respect to inertial frame I) coordinates system B :

$${}_I\mathbf{r} \rightarrow {}_I\dot{\mathbf{r}} = \frac{d{}_I\mathbf{r}}{dt} \quad (1.3)$$

$${}_B\mathbf{r} \rightarrow {}_B\dot{\mathbf{r}} = \frac{d{}_B\mathbf{r}}{dt} + {}_B\boldsymbol{\omega}_{IB} \times {}_B\mathbf{r} \quad (1.4)$$

In this document, O refers to the origin of a frame, I indicates the inertial frame and B a body fixed frame (which can be moved).

```
1 % derivation of position vectors expressed in inertia frame
2 dr = dMATdt(r,q,dq);
```

1.1.4 Rotation

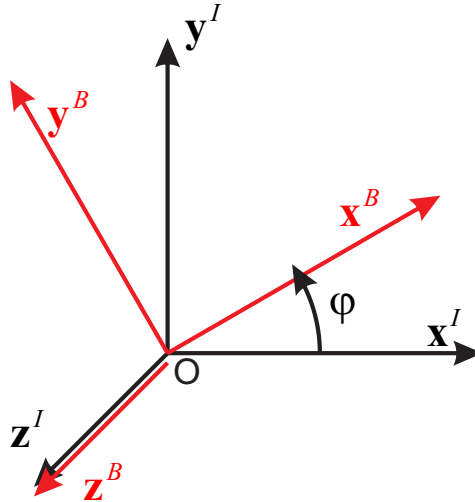


Figure 1.1: Rotated coordinate system B around z axis

The rotation matrix \mathbf{A}_{IB} rotates a vector ${}_B\mathbf{r}$ expressed in frame B to frame I :

$${}_I\mathbf{r} = \mathbf{A}_{IB}{}_B\mathbf{r} \quad (1.5)$$

$${}_B\mathbf{r} = \mathbf{A}_{BI}{}_I\mathbf{r} \quad \mathbf{A}_{BI} = \mathbf{A}_{IB}^T \quad (1.6)$$

$${}_C\mathbf{r} = \mathbf{A}_{CI}{}_I\mathbf{r} \quad \mathbf{A}_{CI} = \mathbf{A}_{CB}\mathbf{A}_{BI} \quad (1.7)$$

In this tool we use the x-y-z convention (see MATLAB code below).

```
1 % rotation matrix around z with angle phi [rad]
2 syms phi real
3 AIB = eulerToRotMat_A_IB(0,0,phi);
4 % Note: we use here the x-y-z definition, so the rotation matrix ...
5     with angles alpha, beta and gamma
6 syms alpha beta gamma real
7 AIB = eulerToRotMat_A_IB(alpha,beta,gamma);
```



```

7 % is equivalent to
8 AIB = eulerToRotMat_A_IB(alpha,0,0)*... % rot around x
9       eulerToRotMat_A_IB(0,beta,0)*... % rot around y
10      eulerToRotMat_A_IB(0,0,gamma); % rot around z

```

More details can be found in the matlab function files:

- /utils/eulerToRotMat_A_IB.m
- /utils/eulerToRotMat_A_BI.m.

1.1.5 Angular Velocity

Given a rotation matrix \mathbf{A}_{IB} , the corresponding rotation speed ${}_I\boldsymbol{\Omega}$ of a rigid body with body fixed frame B is:

$${}_I\tilde{\boldsymbol{\Omega}} = {}_I\tilde{\boldsymbol{\omega}}_{IB} = \dot{\mathbf{A}}_{IB}\mathbf{A}_{IB}^T \quad (1.8)$$

$$\tilde{\boldsymbol{\Omega}} = \begin{bmatrix} 0 & -\Omega^z & \Omega^y \\ \Omega^z & 0 & -\Omega^x \\ -\Omega^y & \Omega^x & 0 \end{bmatrix}, \quad \begin{matrix} \text{unskew} \\ \rightleftharpoons \\ \text{skew} \end{matrix} \quad \boldsymbol{\Omega} = \begin{pmatrix} \Omega^x \\ \Omega^y \\ \Omega^z \end{pmatrix} \quad (1.9)$$

```

1 % rotation matrix around z with vector phi
2 AIB = eulerToRotMat_A_IB(0,0,phi);
3 % differentiate rotation matrix
4 dAIB = dMATdt(AIB,q,dq);
5 % generate rotation speed and unskew it
6 I_Omega = unskew(simplify(dAIB*AIB'));

```

1.1.6 Jacobian

The Jacobian \mathbf{J} is given through:

$$\mathbf{J}(\mathbf{q}) = \frac{\partial \mathbf{r}(\mathbf{q})}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial \mathbf{r}_1}{\partial \mathbf{q}_1} & \frac{\partial \mathbf{r}_1}{\partial \mathbf{q}_2} & \cdots & \frac{\partial \mathbf{r}_1}{\partial \mathbf{q}_n} \\ \frac{\partial \mathbf{r}_2}{\partial \mathbf{q}_1} & \frac{\partial \mathbf{r}_2}{\partial \mathbf{q}_2} & \cdots & \frac{\partial \mathbf{r}_2}{\partial \mathbf{q}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{r}_m}{\partial \mathbf{q}_1} & \frac{\partial \mathbf{r}_m}{\partial \mathbf{q}_2} & \cdots & \frac{\partial \mathbf{r}_m}{\partial \mathbf{q}_n} \end{bmatrix}, \quad \mathbf{q} \in \mathbb{R}^{n \times 1}, \mathbf{r} \in \mathbb{R}^{m \times 1} \quad (1.10)$$

Jacobians are used to map generalized velocities $\dot{\mathbf{q}}$ to Cartesian velocities $\dot{\mathbf{r}}$:

$$\dot{\mathbf{r}} = \mathbf{J}\dot{\mathbf{q}} \quad (1.11)$$

and in its dual problem to map Cartesian forces \mathbf{F} to generalized forces $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \mathbf{J}^T \mathbf{F} \quad (1.12)$$

We differ between translational Jacobians $\mathbf{J}_P = \frac{\partial \mathbf{r}_P(\mathbf{q})}{\partial \mathbf{q}}$, which correspond to a specific point P and rotational Jacobians $\mathbf{J}_R = \frac{\partial \boldsymbol{\Omega}(\mathbf{q})}{\partial \mathbf{q}}$ that are identical for all points of one single rigid body.

```

1 % get jacobian from position vector
2 J = jacobian(r,q);
3 % get jacobian from rotation speed
4 Jr = jacobian(Omega,dq);

```

1.2 Dynamics

The goal of this tool is to get the equations of motion in the following form

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{S}^T \boldsymbol{\tau} \quad (1.13)$$

with

- \mathbf{M} : mass matrix $\in \mathbb{R}^{n \times n}$
- \mathbf{b} : coriolis and centrifugal components $\in \mathbb{R}^{n \times 1}$
- \mathbf{g} : gravitational components $\in \mathbb{R}^{n \times 1}$
- \mathbf{S}^T : selection matrix of the actuated joints $\in \mathbb{R}^{k \times n}$
- $\boldsymbol{\tau}$: generalized forces $\in \mathbb{R}^{k \times 1}$
- n : number of degrees of freedom
- $k \leq n$: number of actuated joints

Note: This tool is written for the most common type of systems. All bindings are skleronomic (time independent) and holonomic. Algebraic differential equations (ADE) are not supported.

1.2.1 Projected Newton-Euler Equations

This framework is based on projected Newton-Euler equations, which can be understood as projection of the conservation of impulse \mathbf{p} and angular momentum \mathbf{N}_S onto generalized coordinates:

$$\sum_{i=1}^N \mathbf{J}_{S_i}^T \dot{\mathbf{p}}_i + \mathbf{J}_{R_i}^T \dot{\mathbf{N}}_{S_i} - \mathbf{J}_{S_i}^T \mathbf{F}_{S_i}^a - \mathbf{J}_{R_i}^T \mathbf{T}_i^a = \mathbf{0} \quad (1.14)$$

$$\mathbf{p}_i(\mathbf{q}) = m_i \dot{\mathbf{r}}_{OS_i} \quad \text{impulse} \quad (1.15)$$

$$\mathbf{N}_{S_i} = \boldsymbol{\theta}_{S_i} \boldsymbol{\Omega}_i \quad \text{angular momentum} \quad (1.16)$$

$$\mathbf{F}_{S_i}^a \quad \text{external forces} \quad (1.17)$$

$$\mathbf{T}_i^a \quad \text{external torques} \quad (1.18)$$

with S_i corresponding to the Center of Gravity of link i . Knowing that the change of impulse and angular momentum can be written as

$$\dot{\mathbf{p}}_i = m_i \ddot{\mathbf{r}}_{OS_i} \quad (1.19)$$

$$\dot{\mathbf{N}}_{S_i} = \boldsymbol{\theta}_{S_i} \dot{\boldsymbol{\Omega}}_i + \boldsymbol{\Omega}_i \times \boldsymbol{\theta}_{S_i} \boldsymbol{\Omega}_i \quad (1.20)$$

where $\boldsymbol{\theta}_{S_i} \in \mathbb{R}^{3 \times 3}$ is the inertia of body i w.r.t. the Center of Gravity. Using the kinematic relations

$$\ddot{\mathbf{r}}_{OS_i} = \mathbf{J}_{S_i} \ddot{\mathbf{q}} + \dot{\mathbf{J}}_{S_i} \dot{\mathbf{q}} \quad (1.21)$$

$$\boldsymbol{\Omega}_i = \mathbf{J}_{R_i} \dot{\mathbf{q}} \quad (1.22)$$

$$\dot{\boldsymbol{\Omega}}_i = \mathbf{J}_{R_i} \ddot{\mathbf{q}} + \dot{\mathbf{J}}_{R_i} \dot{\mathbf{q}} \quad (1.23)$$

the elements of (1.13) are obtained by

$$\mathbf{M}(\mathbf{q}) = \sum_{i=1}^N \mathbf{J}_{S_i}^T m_i \mathbf{J}_{S_i} + \mathbf{J}_{R_i}^T \boldsymbol{\theta}_{S_i} \mathbf{J}_{R_i} \quad (1.24)$$

$$\mathbf{b}(\mathbf{q}) = \sum_{i=1}^N \mathbf{J}_{S_i}^T m_i \dot{\mathbf{J}}_{S_i} \dot{\mathbf{q}} + \mathbf{J}_{R_i}^T \boldsymbol{\theta}_{S_i} \dot{\mathbf{J}}_{R_i} \dot{\mathbf{q}} + \boldsymbol{\Omega}_i \times \boldsymbol{\theta}_{S_i} \boldsymbol{\Omega}_i \quad (1.25)$$

$$\mathbf{g}(\mathbf{q}) = \sum_{i=1}^N -\mathbf{J}_{S_i}^T \mathbf{F}_{S_i}^g \quad (1.26)$$

Chapter 2

Matlab Tool

The presented tool uses the Symbolic Math Toolbox¹.

2.1 Kinematic Tree

The robot is described as a kinematic tree. A root element needs to be selected first and from there the individual branches are successively described. Each rigid body has to be described with the following struct:

```
1 % => start setting up the kinematic structure. Each link needs to ...
   have all the following struct elements
2 % B indicates bodyframe
3 % P indicates coordinate system of parent element
4
5 % body(i).param.m:          body mass
6 % body(i).param.B_Th:      inertia tensor in body frame w.r.t. CoG
7 % body(i).param.B_r_COG:   CoG in body frame
8 % body(i).cs.P_r_PO:       position of origin in parent CS
9 % body(i).cs.A_PB:         rotation from parent CS
10 % body(i).tree.parent:     tree parent (0=inertial frame)
11
12 %% Body 1
13 i = 1;
14 % body mass
15 body(i).param.m = m1;
16 body(i).param.B_Th = diag([Th1.xx Th1.yy Th1.zz]);
17 body(i).param.B_r_COG = [0;0;s1];
18 body(i).cs.P_r_PO = sym([0;0;0]); % ensure a symbolic expression
19 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,q1);
20 body(i).tree.parent = 0;
```

Some notes on the notation:

${}_B\theta$: Inertia of the body with respect to CoG expressed in body fixed frame B

${}_B\mathbf{r}_{CoG}$: Vector from origin of body fixed frame to CoG expressed in body frame B

${}_P\mathbf{r}_{PO}$: Translational vector from origin of parent frame P
to origin of body frame B expressed in parent frame P

\mathbf{A}_{PB} : Rotation matrix that rotates a vector expressed in body frame B to parent frame P

See Figure 3.1, which represents a 3-link robot arm, to better understand the definition of the vectors.

¹<http://www.mathworks.ch/products/symbolic/>

2.2 Force Elements

This framework allows to describe both force (prismatic, type = 'lin') and torque (rotational, type='rot') actuators.

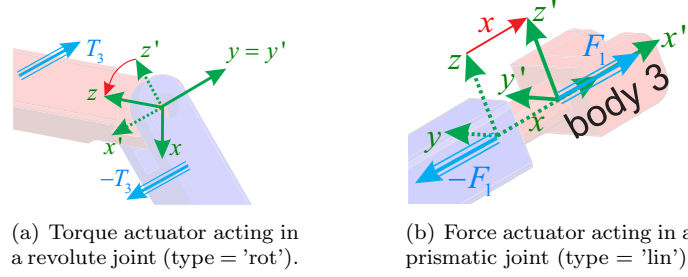


Figure 2.1: Two types of force elements can be defined.

2.2.1 Torque Actuators

This is the most common actuator for all types of robotic arms. They are defined as follows:

```

1 % torque element acting between environment and body 1
2 i = 1;
3 ftel(i).type = 'rot';           % define it to be a rotational = torque
4 ftel(i).body_P = 0;            % body on which the reaction happens
5 ftel(i).body_B = 1;            % body on which the action happens
6 ftel(i).B_T = [0;0;T1];        % torque vector, expressed in B frame

```

For a detailed example, please check Section 3.1.

2.2.2 Force Actuators

Prismatic joints (like hydraulic/pneumatic cylinders, spindle drives, etc.) are defined as follows:

```

1 % force element acting between body 2 and body 3
2 i = 3;
3 ftel(i).type = 'lin';          % define it to be a rotational = torque
4 ftel(i).body_P = 2;            % body on which the reaction happens
5 ftel(i).body_B = 3;            % body on which the action happens
6 ftel(i).P_r_R = sym([0;0;0]); % point of reaction in P frame
7 ftel(i).B_r_A = sym([0;0;0]); % point of action in B frame
8 ftel(i).B_F = [F1;0;0];        % force vector of action

```

For a detailed example, please check Section 3.2.

Note: The case of a cylinder attached to two bodies that are connected over a revolute joint falls (although being a linear actuator) into the category of torque actuators (Section 2.2.1).

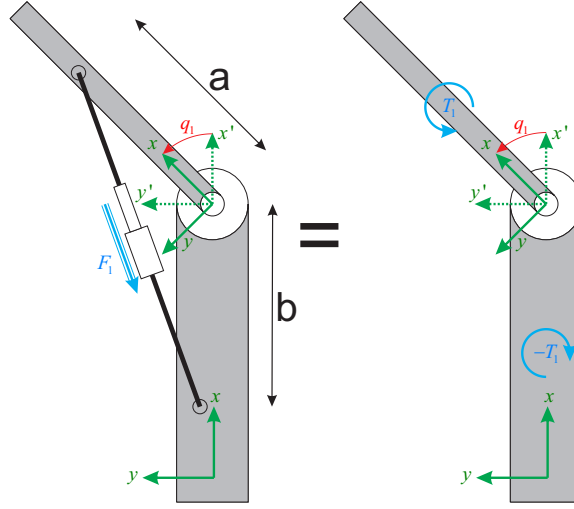


Figure 2.2: A cylinder in combination with a revolute joint has to be modeled as a torque actuator.

The linear force F_1 with the two joint offsets a, b can be transformed to a joint torque $T_1 = f(q_1, a, b, F_1)$:

$$\phi = \tan^{-1} \left(\frac{a \sin q_1}{b + a \cos q_1} \right) \quad (2.1)$$

$$T_1 = F_1 b \sin \phi \quad (2.2)$$

2.3 Projected Newton-Euler Equations

After the setup of the relative body kinematics (Section 2.1) and the force elements (Section 2.2) we can calculate the absolute kinematics and dynamics of the system by applying the projected Newton-Euler method:

```

1 function [sys, body, ftel] = computePNE(body,ftel,q,dq,tau,Ia_grav)
2
3 % INPUT:
4 % * body:      kinematic tree
5 % * ftel:      force/torque elements
6 % * q:         generalized coordinates (symb) in desired order
7 % * Dq:        corresponding velocities
8 % * tau:       actuator force/torque array (symb)
9 % * Ia_grav:   gravity vector (R3)
10 %
11 % OUTPUT:
12 % * sys:       system struct containing dynamics (all symbolic)
13 %   .MpNE:     mass matrix
14 %   .bpNE:     coriolis/centrifugal
15 %   .gpNE:     gravity terms
16 %   .SpNE:     actuator selection matrix
17 %   .fpNE:     =SpNE*I;
18 %   .param:    input parameters
19 %   .q:         generalized coordinates
20 %   .Dq:        generalized velocities
21 %   .tau        generalized actuator forces
22 %
23 % * body.kin:  body contains (among others) now also global ...
24 %               kinematics as:
25 %   .A_IB:     rotation matrix from B to I (inertial/world frame)

```

```
25 % .I_r_O:      vector inertial frame to body frame
26 % .I_dr_O:     velocity of body frame in inertia frame
27 % .I_r_CoG:    center of gravity position represented in inertia frame
28 % .I_dr_CoG:   velcity ...
29 % .I_J_CoG:    CoG jacobian
30 % .I_Jr:       rotation jacobian
31 % .I_Omega:    body rotational speed
32 % ... and a lot more
```

Chapter 3

Examples

This is a collection of validated (with the software Neweul-M²[1]) examples:

Examples/RA3Link/	3-link robot arm (Section 3.1)
Examples/RA3LinkPrismatic/	3-link robot arm with prismatic joint (Section 3.2)
Examples/QuadrupedFreeFloat/	free floating quadruped (Section 3.3)

3.1 3-link Robot Arm

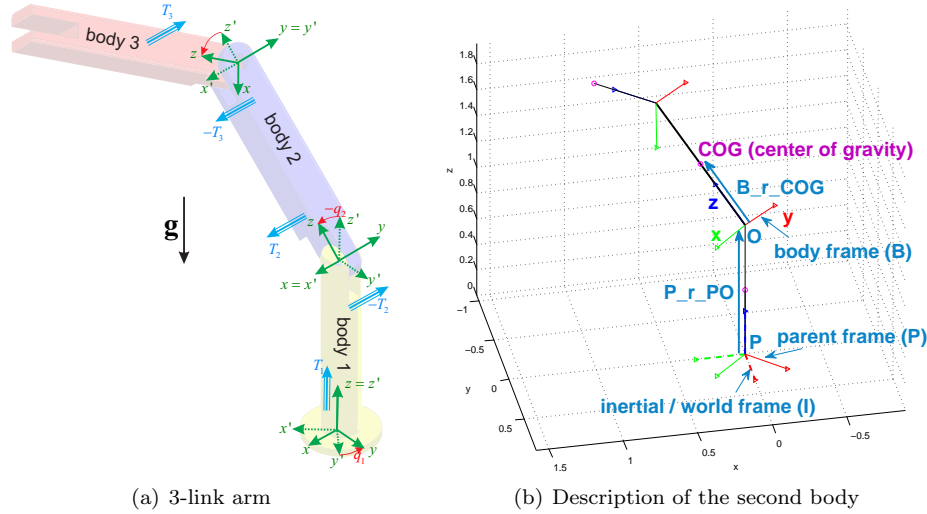


Figure 3.1: 3-link robot arm with three revolute joints ($\mathbf{z}, \mathbf{x}, \mathbf{y}$).

```

1  % genEoM.m
2  %
3  % -> generation of the EoM for a robot arm with three links.
4  %
5  % proNEu: tool for symbolic EoM derivation
6  % Copyright (C) 2011 Marco Hutter, Christian Gehring
7  %
8  % This program is free software: you can redistribute it and/or modify
9  % it under the terms of the GNU General Public License as published by
10 % the Free Software Foundation, either version 3 of the License, or
11 % any later version.
12 %

```

```

13 % This program is distributed in the hope that it will be useful,
14 % but WITHOUT ANY WARRANTY; without even the implied warranty of
15 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 % GNU General Public License for more details.
17 %
18 % You should have received a copy of the GNU General Public License
19 % along with this program. If not, see <http://www.gnu.org/licenses/>.
20
21 clc
22 clear all
23
24 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
25 % Minimal coordinates and derivatives
26 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
27 % => define in this section the minimal coordinates that you
28 % want to use. They can stand for rotational or prismatic joints.
29
30 syms q1 q2 q3 real % define the three angles
31 q = [q1 q2 q3]'; % vector of generalized coordinates
32 qDef = [pi/4, pi/4, pi/4]'; % define some values for visualization
33
34 syms Dq1 Dq2 Dq3 real % define the derivatives of the angles
35 dq = [Dq1 Dq2 Dq3]'; % derivatives of the gen. cord.
36 dqDef = [0, 0, 0]'; % define some values
37
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 % Generalized actuator torques
40 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41 % => define in this section the generalized forces (for prismatic ...
    joints)
42 % and the generalized torques (for rotational joints)
43 syms T1 T2 T3 real % define the three torques
44 T = [T1 T2 T3]'; % torque vector
45 TDef = [1, 2, 3]'; % define some values
46
47
48
49 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50 % Parameters
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52 % => define in this section the parameters to describe
53 % the link properties
54 % i.e.      l1: length to next joint
55 %          s1: offset CoG
56 %          m1: link mass
57 %          Th1.. : inertia properties of link$
58
59 syms l1 s1 m1 Th1.xx Th1.yy Th1.zz real % link 1
60 syms l2 s2 m2 Th2.xx Th2.yy Th2.zz real % link 2
61 syms l3 s3 m3 Th3.xx Th3.yy Th3.zz real % link 3
62 % parameter vector
63 param = [l1 s1 m1 Th1.xx Th1.yy Th1.zz]'; % link 1
64 param = [param', l2 s2 m2 Th2.xx Th2.yy Th2.zz]'; % link 2
65 param = [param', l3 s3 m3 Th3.xx Th3.yy Th3.zz]'; % link 3
66 % define some values for visualization
67 paramDef = [1 0.5 1 1 1 1]'; % link 1
68 paramDef = [paramDef', 1 0.5 1 1 1 1]'; % link 2
69 paramDef = [paramDef', 1 0.5 1 1 1 1]'; % link 3
70
71 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
72 % Gravity
73 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
74 syms g real
75 I_a-grav = [0; 0; -g]; % gravity vector expressed in ...
    inertial frame
76 param = [param; g]; % add gravity to the parameter vector
77 paramDef = [paramDef; 9.81]; % define the value

```



```

78
79 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
80 % Kinematic structure — kinematic tree
81 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
82 % => start setting up the kinematic structure.
83 % Each link needs to have all the following struct elements
84 % B indicates body frame
85 % P indicates coordinate system of parent element
86
87 % Body 1
88 i = 1;
89 body(i).param.m = m1; % mass
90 body(i).param.B_Th = sym(diag([Th1.xx Th1.yy Th1.zz])); % inertia ...
    in B frame
91 body(i).param.B_r_COG = sym([0;0;s1]); % CoG position in B frame
92 body(i).cs.P_r_PO = sym([0;0;0]); % body frame origin in ...
    predecessor frame
93 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,q1); % rotation of body frame
94 body(i).tree.parent = 0; % parent element
95
96 % Body 2
97 i = 2;
98 body(i).param.m = m2;
99 body(i).param.B_Th = sym(diag([Th2.xx Th2.yy Th2.zz]));
100 body(i).param.B_r_COG = sym([0;0;s2]);
101 body(i).cs.P_r_PO = sym([0;0;l1]);
102 body(i).cs.A_PB = eulerToRotMat_A_IB(q2,0,0);
103 body(i).tree.parent = 1;
104
105 % Body 3
106 i = 3;
107 body(i).param.m = m3;
108 body(i).param.B_Th = sym(diag([Th3.xx Th3.yy Th3.zz]));
109 body(i).param.B_r_COG = sym([0;0;s3]);
110 body(i).cs.P_r_PO = sym([0;0;l2]);
111 body(i).cs.A_PB = eulerToRotMat_A_IB(0,q3,0);
112 body(i).tree.parent = 2;
113
114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
115 % Force/torque elements
116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
117 % => define in this section the force / torque elements that act
118 % between the bodies
119
120 % torque element acting on body 1
121 ftel(1).type = 'rot'; % define it to be a rotational = torque
122 ftel(1).body_P = 0; % body on which the reaction happens
123 ftel(1).body_B = 1; % body on which the action happens
124 ftel(1).B_T = [0;0;T1]; % torque vector, expressed in B frame
125
126 % torque element acting between body 1 and body 2
127 ftel(2).type = 'rot'; % define it to be a rotational = torque
128 ftel(2).body_P = 1; % body on which the reaction happens
129 ftel(2).body_B = 2; % body on which the action happens
130 ftel(2).B_T = [T2;0;0]; % torque vector, expressed in B frame
131
132 % torque element acting between body 2 and body 3
133 ftel(3).type = 'rot'; % define it to be a rotational = torque
134 ftel(3).body_P = 2; % body on which the reaction happens
135 ftel(3).body_B = 3; % body on which the action happens
136 ftel(3).B_T = [0;T3;0]; % torque vector, expressed in B frame
137
138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
139 % Projected Newton Euler equations
140 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
141 % => compute the dynamics by applying the proj. Newton-Euler method
142 [sys, body, ftel] = computePNE(body,ftel,q,dq,T,I_a_grav,param);

```

```
143
144 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
145 % Save the data
146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
147 % save the kinematics and dynamics
148 save('body','body')
149 save('sys','sys')
150 % save the numerical values in a struct
151 values.paramDef = paramDef;
152 values.qDef = qDef;
153 values.dqDef = dqDef;
154 values.TDef = TDef;
155 save('values','values')
156 % save the force/torque elements
157 save('ftel','ftel')
158
159 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
160 % Visualization
161 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
162 % plot the robot arm
163 plotBodies(body,param,paramDef,q,qDef);
```

3.2 3-link Robot Arm with Prismatic Joint

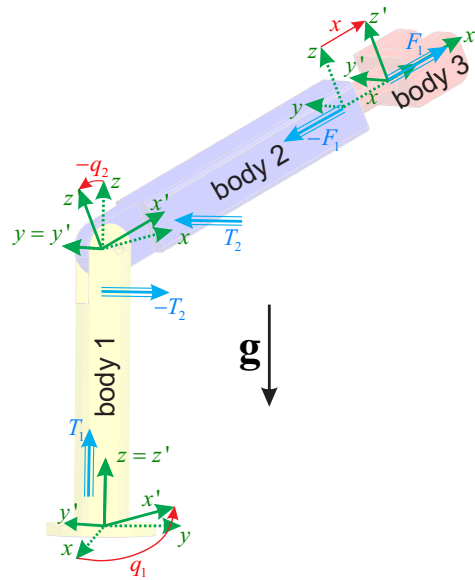


Figure 3.2: 3-link robot arm with two revolute joints (z, y) and one prismatic joint (x).

```

1 % 3LinkRobotArmPrismaticLink_genEoM.m
2 %
3 % -> example file to generate the EoM of a 3-link robot arm that ...
4 %   has a
5 %   prismatic joint as the last actuator. For detailed information ...
6 %   and a
7 %   figure with the links and coordinate systems, please have a look ...
8 %   at the
9 %   documentation.
10 %
11 % proNEu: tool for symbolic EoM derivation
12 % Copyright (C) 2011 Marco Hutter, Christian Gehring
13 %
14 % This program is free software: you can redistribute it and/or modify
15 % it under the terms of the GNU General Public License as published by
16 % the Free Software Foundation, either version 3 of the License, or
17 % any later version.
18 %
19 % This program is distributed in the hope that it will be useful,
20 % but WITHOUT ANY WARRANTY; without even the implied warranty of
21 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
22 % GNU General Public License for more details.
23 %
24 % You should have received a copy of the GNU General Public License
25 % along with this program. If not, see <http://www.gnu.org/licenses/>.
26 %
27 clc
28 clear all
29
30 % =====
31 % Minimal coordinates and velocity
32 % =====
33 % => define in this section the minimal coordinates that you
34 % want to use. They can stand for rotational or prismatic joints.

```

```

32
33 syms q1 q2 x real % generalized coordinates
34 q = [q1 q2 x]'; % q1, q2: 'rot' x:'lin'
35 qDef = [pi/4,pi/4,0.2]'; % define here IC for visualization
36
37 syms Dq1 Dq2 Dx real % generalized velocities
38 dq = [Dq1 Dq2 Dx]';
39 dqDef = [0,0,0]';
40
41 %%%%%%%%%%%%%%%
42 % Torques/Forces
43 %%%%%%%%%%%%%%%
44 % => define in this section the generalized forces (for prismatic ...
    joints)
45 % and the generalized torques (for rotational joints)
46
47 syms T1 T2 F1 real % generalized Force/Torque vector
48 T = [T1 T2 F1]';
49 TDef = [1,2,3]';
50
51
52 %%%%%%%%%%%%%%%
53 % Parameters
54 %%%%%%%%%%%%%%%
55 % => define in this section the parameters to describe the link ...
    properties
56 % i.e. l1: length to next joint
57 % s1: offset CoG
58 % m1: link mass
59 % Th1.. : inertia properties of link$
60
61 syms l1 s1 m1 Th1_xx Th1_yy Th1_zz real
62 syms l2 s2 m2 Th2_xx Th2_yy Th2_zz real
63 syms l3 s3 m3 Th3_xx Th3_yy Th3_zz real
64 param = [l1 s1 m1 Th1_xx Th1_yy Th1_zz]'; % this defines the ...
    order
65 param = [param', l2 s2 m2 Th2_xx Th2_yy Th2_zz]';
66 param = [param', l3 s3 m3 Th3_xx Th3_yy Th3_zz]';
67 paramDef = [1 0.5 1 1 1 1]'; % define here the default values
68 paramDef = [paramDef', 1 0.5 1 1 1 1]';
69 paramDef = [paramDef', 1 0.5 1 1 1 1]';
70
71
72 %%%%%%%%%%%%%%%
73 % Gravity
74 %%%%%%%%%%%%%%%
75 syms g real
76 I_a_grav = [0; 0; -g];
77 param = [param;g];
78 paramDef = [paramDef;9.81];
79
80 %%%%%%%%%%%%%%%
81 % Kinematic structure — kinematic tree
82 %%%%%%%%%%%%%%%
83 % => start setting up the kinematic structure.
84 % Each link needs to have all the following struct elements
85 % B indicates body frame
86 % P indicates coordinate system of parent element
87
88 % Body 1
89 i = 1;
90 body(i).param.m = m1; % mass
91 body(i).param.B_Th = diag([Th1_xx Th1_yy Th1_zz]); % inertia tensor
92 body(i).param.B_r.COG = [0;0;s1]; % center of gravity in body frame
93 body(i).cs.P_r.PO = sym([0;0;0]); % position of origin in parent CS
94 body(i).cs.A_PB = eulerToRotMatA_IB(0,0,q1); % rotation from ...
    parent CS

```

```

95 body(i).tree.parent = 0;    % tree parent (0=inertial frame)
96
97 % Body 2
98 i = 2;
99 body(i).param.m = m2;
100 body(i).param.B_Th = diag([Th2_xx Th2_yy Th2_zz]);
101 body(i).param.B_r_COG = [s2;0;0];
102 body(i).cs.P_r_PO = [0;0;l1];
103 body(i).cs.A_PB = eulerToRotMat_A_IB(0,q2,0);
104 body(i).tree.parent = 1;
105
106 % Body 3
107 i = 3;
108 body(i).param.m = m3;
109 body(i).param.B_Th = diag([Th3_xx Th3_yy Th3_zz]);
110 body(i).param.B_r_COG = [s3;0;0];
111 body(i).cs.P_r_PO = [l2+x;0;0];
112 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
113 body(i).tree.parent = 2;
114
115
116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
117 % Force/torque elements
118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119 % => define in this section the force / torque elements that act
120 % between the bodies
121
122 % torque element acting on body 1
123 i = 1;
124 ftel(i).type = 'rot';          % define it to be a rotational = torque
125 ftel(i).body_P = 0;           % body on which the reaction happens
126 ftel(i).body_B = 1;           % body on which the action happens
127 ftel(i).B_T = [0;0;T1];       % torque vector, expressed in B frame
128
129 % torque element acting between body 1 and body 2
130 i = 2;
131 ftel(i).type = 'rot';          % define it to be a rotational = torque
132 ftel(i).body_P = 1;           % body on which the reaction happens
133 ftel(i).body_B = 2;           % body on which the action happens
134 ftel(i).B_T = [0;T2;0];       % torque vector, expressed in B frame
135
136 % force element acting between body 2 and body 3
137 i = 3;
138 ftel(i).type = 'lin';          % define it to be a linear = force
139 ftel(i).body_P = 2;           % body on which the reaction happens
140 ftel(i).body_B = 3;           % body on which the action happens
141 ftel(i).P_r_R = sym([0;0;0]); % point of reaction in P frame
142 ftel(i).B_r_A = sym([0;0;0]); % point of action in B frame
143 ftel(i).B_F = [F1;0;0];       % force vector of action / this is ...
    optional
144
145
146 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
147 % Projected Newton Euler equations
148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
149 [sys, body, ftel] = computePNE(body,ftel,q,dq,T,I_a_grav,param);
150
151 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
152 % Save the data
153 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
154 % save the kinematics and dynamics
155 save('body','body')
156 save('sys','sys')
157 % save the numerical values in a struct
158 values.paramDef = paramDef;
159 values.qDef = qDef;
160 values.dqDef = dqDef;

```

```
161 values.TDef = TDef;
162 save('values','values')
163 % save the force/torque elements
164 save('ftel','ftel')
165
166 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
167 % Visualization
168 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
169 plotBodies(body,param,paramDef,q,qDef);
```

3.3 Quadruped Robot StarlETH

For detailed information about our quadruped StarlETH please consult our homepage¹. Here is a brief outline: each leg has 3 degrees of freedom (x-rotation for hip abduction/adduction, followed by y-rotation for hip flexion/extension as well as y-rotation for knee flexion/extension) and is connected to a free floating main body.

This model shows also how to model a free floating body as well as ground contact. Adding dummy-bodies as feet allow to directly get the support Jacobians needed for modeling the hard ground contact (impact as well as contact constraints).

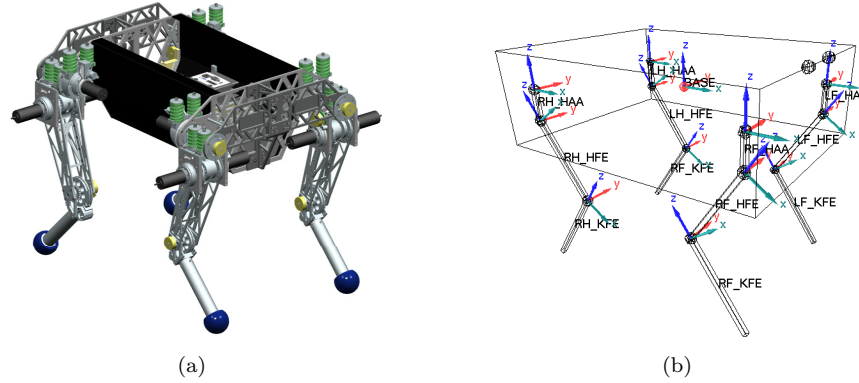


Figure 3.3: Free floating quadruped StarlETH with a total of 18 DoF.

```

1 % genEoM.m
2 % -> derivation of equations of motion for a free floating Quadruped
3 %
4 %      M(q)ddq + b(q,dq) + g(q) + Js(q)'*Fs = S'*T
5 %
6 % * M(q)      = Mass matrix
7 % * b(q,dq)   = coriolis and centrifugal terms
8 % * g(q)      = gravitational terms
9 % * Js(q)     = ground contact (support) jacobian
10 % * Fs       = grouond contact (support) force
11 % * S        = actuator selection matrix
12 % * T        = acutator torque vector
13 %
14 % In this example file, these equations are derived
15 % using projected Newton- Euler equations.
16 %
17 % The example is a quadruped robot with 3 DoF per leg
18 %
19 % proNEu: tool for symbolic EoM derivation
20 % Copyright (C) 2011 Marco Hutter, Christian Gehring
21 %
22 % This program is free software: you can redistribute it and/or modify
23 % it under the terms of the GNU General Public License as published by
24 % the Free Software Foundation, either version 3 of the License, or
25 % any later version.
26 %
27 % This program is distributed in the hope that it will be useful,
28 % but WITHOUT ANY WARRANTY; without even the implied warranty of
29 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
30 % GNU General Public License for more details.
31 %

```

¹leggedrobotics.ethz.ch

```

32 % You should have received a copy of the GNU General Public License
33 % along with this program. If not, see <http://www.gnu.org/licenses/>.
34 %
35 % Description of link object
36 % body(i).param.m      = mass [kg]
37 % body(i).param.B_Th   = moment of inertia w.r.t. center of gravity
38 %                      = described in local coordinate system [kg*m^2]
39 % body(i).param.B_r_COG = position of center of gravity w.r.t. ...
40 %                      = origin of
41 %                      = local frame described in local coordinate ...
42 %                      = system
43 % body(i).cs.P_r_PO     = position of origin of local frame w.r.t.
44 %                      = origin of parent frame described in parent
45 %                      = coordinate system
46 % body(i).cs.A_PB       = rotation matrix that rotates the parent ...
47 %                      = frame to
48 %                      = the local frame (3x3-matrix)
49 % body(i).kin.A_IB      = rotation matrix that rotates the local ...
50 %                      = frame to
51 %                      = the world frame (3x3-matrix)
52 % body(i).kin.dA_IB     = time derivative of A_IB (3x3-matrix)
53 % body(i).kin.A_BI      =
54 %                      = position of origin of local frame described
55 %                      = in world coordinate system
56 % body(i).kin.I_r_O     = time derivative of I_r_O
57 % body(i).kin.I_dr_O    = position of center of gravity w.r.t. ...
58 % body(i).kin.I_r_COG   = origin of
59 %                      = world frame described in world coordinate ...
60 %                      = system
61 % body(i).kin.I_dr_COG  = time derivative of I_r_COG
62 % body(i).kin.B_Omega   = angular velocity of joint described
63 %                      = in world coordinate system
64 % body(i).kin.B_Omega_tilde = angular velocity of joint in ...
65 %                      = skew-symmetric
66 %                      = matrix described in world coordinate system
67 % body(i).kin.I_J_O     = Jacobian w.r.t. origin of local frame ...
68 %                      = described
69 %                      = in world coordinate system
70 % body(i).kin.I_dJ_O    = time derivative of I_J_O
71 % body(i).kin.I_J_COG   = Jacobian w.r.t. center of gravity ...
72 %                      = described in
73 %                      = world coordinate system
74 % body(i).kin.I_dJ_COG  = time derivative of I_J_COG
75 % body(i).kin.I_Jr      = rotational Jacobian described
76 %                      = in world coordinate system
77 % body(i).kin.I_dJr     = time derivative of I_Jr
78 % body(i).dyn.B_F       = force vector described in local coord. ...
79 %                      = system
80 % body(i).dyn.B_T       = torque vector described in local coord. ...
81 %                      = system
82 % body(i).dyn.I_F_grav  = gravity force vector described in world c.s.
83 % body(i).tree.parent   = index of parent
84 %
85 % LF = left front
86 % RF = right front
87 % LH = left hind
88 % RH = right hind
89 % HAA = hip abduction/ adduction
90 % HFE = hip flexion / extension
91 % KFE = knee flexion / extension
92 %
93 %
94 %
95 %
96 %
97 %
98 %
99 %
100 %
101 %
102 %
103 %
104 %
105 %
106 %
107 %
108 %
109 %
110 %
111 %
112 %
113 %
114 %
115 %
116 %
117 %
118 %
119 %
120 %
121 %
122 %
123 %
124 %
125 %
126 %
127 %
128 %
129 %
130 %
131 %
132 %
133 %
134 %
135 %
136 %
137 %
138 %
139 %
140 %
141 %
142 %
143 %
144 %
145 %
146 %
147 %
148 %
149 %
150 %
151 %
152 %
153 %
154 %
155 %
156 %
157 %
158 %
159 %
160 %
161 %
162 %
163 %
164 %
165 %
166 %
167 %
168 %
169 %
170 %
171 %
172 %
173 %
174 %
175 %
176 %
177 %
178 %
179 %
180 %
181 %
182 %
183 %
184 %
185 %
186 %
187 %
188 %
189 %
190 %
191 %
192 %
193 %
194 %
195 %
196 %
197 %
198 %
199 %
200 %
201 %
202 %
203 %
204 %
205 %
206 %
207 %
208 %
209 %
210 %
211 %
212 %
213 %
214 %
215 %
216 %
217 %
218 %
219 %
220 %
221 %
222 %
223 %
224 %
225 %
226 %
227 %
228 %
229 %
230 %
231 %
232 %
233 %
234 %
235 %
236 %
237 %
238 %
239 %
240 %
241 %
242 %
243 %
244 %
245 %
246 %
247 %
248 %
249 %
250 %
251 %
252 %
253 %
254 %
255 %
256 %
257 %
258 %
259 %
260 %
261 %
262 %
263 %
264 %
265 %
266 %
267 %
268 %
269 %
270 %
271 %
272 %
273 %
274 %
275 %
276 %
277 %
278 %
279 %
280 %
281 %
282 %
283 %
284 %
285 %
286 %
287 %
288 %
289 %
290 %
291 %
292 %
293 %
294 %
295 %
296 %
297 %
298 %
299 %
300 %
301 %
302 %
303 %
304 %
305 %
306 %
307 %
308 %
309 %
310 %
311 %
312 %
313 %
314 %
315 %
316 %
317 %
318 %
319 %
320 %
321 %
322 %
323 %
324 %
325 %
326 %
327 %
328 %
329 %
330 %
331 %
332 %
333 %
334 %
335 %
336 %
337 %
338 %
339 %
340 %
341 %
342 %
343 %
344 %
345 %
346 %
347 %
348 %
349 %
350 %
351 %
352 %
353 %
354 %
355 %
356 %
357 %
358 %
359 %
360 %
361 %
362 %
363 %
364 %
365 %
366 %
367 %
368 %
369 %
370 %
371 %
372 %
373 %
374 %
375 %
376 %
377 %
378 %
379 %
380 %
381 %
382 %
383 %
384 %
385 %
386 %
387 %
388 %
389 %
390 %
391 %
392 %
393 %
394 %
395 %
396 %
397 %
398 %
399 %
400 %
401 %
402 %
403 %
404 %
405 %
406 %
407 %
408 %
409 %
410 %
411 %
412 %
413 %
414 %
415 %
416 %
417 %
418 %
419 %
420 %
421 %
422 %
423 %
424 %
425 %
426 %
427 %
428 %
429 %
430 %
431 %
432 %
433 %
434 %
435 %
436 %
437 %
438 %
439 %
440 %
441 %
442 %
443 %
444 %
445 %
446 %
447 %
448 %
449 %
450 %
451 %
452 %
453 %
454 %
455 %
456 %
457 %
458 %
459 %
460 %
461 %
462 %
463 %
464 %
465 %
466 %
467 %
468 %
469 %
470 %
471 %
472 %
473 %
474 %
475 %
476 %
477 %
478 %
479 %
480 %
481 %
482 %
483 %
484 %
485 %
486 %
487 %
488 %
489 %
490 %
491 %
492 %
493 %
494 %
495 %
496 %
497 %
498 %
499 %
500 %
501 %
502 %
503 %
504 %
505 %
506 %
507 %
508 %
509 %
510 %
511 %
512 %
513 %
514 %
515 %
516 %
517 %
518 %
519 %
520 %
521 %
522 %
523 %
524 %
525 %
526 %
527 %
528 %
529 %
530 %
531 %
532 %
533 %
534 %
535 %
536 %
537 %
538 %
539 %
540 %
541 %
542 %
543 %
544 %
545 %
546 %
547 %
548 %
549 %
550 %
551 %
552 %
553 %
554 %
555 %
556 %
557 %
558 %
559 %
560 %
561 %
562 %
563 %
564 %
565 %
566 %
567 %
568 %
569 %
570 %
571 %
572 %
573 %
574 %
575 %
576 %
577 %
578 %
579 %
580 %
581 %
582 %
583 %
584 %
585 %
586 %
587 %
588 %
589 %
590 %
591 %
592 %
593 %
594 %
595 %
596 %
597 %
598 %
599 %
600 %
601 %
602 %
603 %
604 %
605 %
606 %
607 %
608 %
609 %
610 %
611 %
612 %
613 %
614 %
615 %
616 %
617 %
618 %
619 %
620 %
621 %
622 %
623 %
624 %
625 %
626 %
627 %
628 %
629 %
630 %
631 %
632 %
633 %
634 %
635 %
636 %
637 %
638 %
639 %
640 %
641 %
642 %
643 %
644 %
645 %
646 %
647 %
648 %
649 %
650 %
651 %
652 %
653 %
654 %
655 %
656 %
657 %
658 %
659 %
660 %
661 %
662 %
663 %
664 %
665 %
666 %
667 %
668 %
669 %
670 %
671 %
672 %
673 %
674 %
675 %
676 %
677 %
678 %
679 %
680 %
681 %
682 %
683 %
684 %
685 %
686 %
687 %
688 %
689 %
690 %
691 %
692 %
693 %
694 %
695 %
696 %
697 %
698 %
699 %
700 %
701 %
702 %
703 %
704 %
705 %
706 %
707 %
708 %
709 %
710 %
711 %
712 %
713 %
714 %
715 %
716 %
717 %
718 %
719 %
720 %
721 %
722 %
723 %
724 %
725 %
726 %
727 %
728 %
729 %
730 %
731 %
732 %
733 %
734 %
735 %
736 %
737 %
738 %
739 %
740 %
741 %
742 %
743 %
744 %
745 %
746 %
747 %
748 %
749 %
750 %
751 %
752 %
753 %
754 %
755 %
756 %
757 %
758 %
759 %
760 %
761 %
762 %
763 %
764 %
765 %
766 %
767 %
768 %
769 %
770 %
771 %
772 %
773 %
774 %
775 %
776 %
777 %
778 %
779 %
780 %
781 %
782 %
783 %
784 %
785 %
786 %
787 %
788 %
789 %
790 %
791 %
792 %
793 %
794 %
795 %
796 %
797 %
798 %
799 %
800 %
801 %
802 %
803 %
804 %
805 %
806 %
807 %
808 %
809 %
810 %
811 %
812 %
813 %
814 %
815 %
816 %
817 %
818 %
819 %
820 %
821 %
822 %
823 %
824 %
825 %
826 %
827 %
828 %
829 %
830 %
831 %
832 %
833 %
834 %
835 %
836 %
837 %
838 %
839 %
840 %
841 %
842 %
843 %
844 %
845 %
846 %
847 %
848 %
849 %
850 %
851 %
852 %
853 %
854 %
855 %
856 %
857 %
858 %
859 %
860 %
861 %
862 %
863 %
864 %
865 %
866 %
867 %
868 %
869 %
870 %
871 %
872 %
873 %
874 %
875 %
876 %
877 %
878 %
879 %
880 %
881 %
882 %
883 %
884 %
885 %
886 %
887 %
888 %
889 %
890 %
891 %
892 %
893 %
894 %
895 %
896 %
897 %
898 %
899 %
900 %
901 %
902 %
903 %
904 %
905 %
906 %
907 %
908 %
909 %
910 %
911 %
912 %
913 %
914 %
915 %
916 %
917 %
918 %
919 %
920 %
921 %
922 %
923 %
924 %
925 %
926 %
927 %
928 %
929 %
930 %
931 %
932 %
933 %
934 %
935 %
936 %
937 %
938 %
939 %
940 %
941 %
942 %
943 %
944 %
945 %
946 %
947 %
948 %
949 %
950 %
951 %
952 %
953 %
954 %
955 %
956 %
957 %
958 %
959 %
960 %
961 %
962 %
963 %
964 %
965 %
966 %
967 %
968 %
969 %
970 %
971 %
972 %
973 %
974 %
975 %
976 %
977 %
978 %
979 %
980 %
981 %
982 %
983 %
984 %
985 %
986 %
987 %
988 %
989 %
990 %
991 %
992 %
993 %
994 %
995 %
996 %
997 %
998 %
999 %

```


[illegible]

```

155 % gravitational constant
156 I_a_grav = [0; 0; -g];
157
158
159 %% DYNAMICS
160
161 % Main Body
162 i = 1;
163 body(i).param.m = mB;
164 body(i).param.B_Th = sym(diag([ThB_xx ThB_yy ThB_zz]));
165 body(i).param.B_r_COG = sym([0;0;sB]);
166 body(i).cs.P_r_PO = sym([qX;qY;qZ]);
167 body(i).cs.A_PB = eulerToRotMat_A_IB(qAL,qBE,qGA);
168 body(i).tree.parent = 0;
169
170 % LF_HAA
171 i = i+1;
172 body(i).param.m = mH;
173 body(i).param.B_Th = sym(diag([ThH_xx ThH_yy ThH_zz]));
174 body(i).param.B_r_COG = [0;0;sH];
175 body(i).cs.P_r_PO = sym([b2hx;b2hy;0]);
176 body(i).cs.A_PB = eulerToRotMat_A_IB(qLF_HAA,0,0);
177 body(i).tree.parent = 1;
178
179 % LF_HFE
180 i = i+1;
181 body(i).param.m = mT;
182 body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
183 body(i).param.B_r_COG = [0;0;sT];
184 body(i).cs.P_r_PO = sym([0;0;lH]);
185 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLF_HFE,0);
186 body(i).tree.parent = i-1;
187
188 % LF_KFE
189 i = i+1;
190 body(i).param.m = mS;
191 body(i).param.B_Th = sym(diag([ThS_xx ThS_yy ThS_zz]));
192 body(i).param.B_r_COG = [0;0;sS];
193 body(i).cs.P_r_PO = sym([0;0;lT]);
194 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLF_KFE,0);
195 body(i).tree.parent = i-1;
196
197 % LF_Foot
198 i = i+1;
199 body(i).param.m = 0;
200 body(i).param.B_Th = diag([0, 0, 0]);
201 body(i).param.B_r_COG = [0;0;0];
202 body(i).cs.P_r_PO = sym([0;0;lS]);
203 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
204 body(i).tree.parent = i-1;
205
206 % RF_HAA
207 i = i+1;
208 body(i).param.m = mH;
209 body(i).param.B_Th = sym(diag([ThH_xx ThH_yy ThH_zz]));
210 body(i).param.B_r_COG = [0;0;sH];
211 body(i).cs.P_r_PO = sym([b2hx;-b2hy;0]);
212 body(i).cs.A_PB = eulerToRotMat_A_IB(qRF_HAA,0,0);
213 body(i).tree.parent = 1;
214
215 % RF_HFE
216 i = i+1;
217 body(i).param.m = mT;
218 body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
219 body(i).param.B_r_COG = [0;0;sT];
220 body(i).cs.P_r_PO = sym([0;0;lH]);
221 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRF_HFE,0);

```

```

222 body(i).tree.parent = i-1;
223
224 % RF_KFE
225 i = i+1;
226 body(i).param.m = mS;
227 body(i).param.B_Th = sym(diag([ThS.xx ThS.yy ThS.zz]));
228 body(i).param.B_r_COG = [0;0;sS];
229 body(i).cs.P_r_PO = sym([0;0;lT]);
230 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRF_KFE,0);
231 body(i).tree.parent = i-1;
232
233 % RF_Foot
234 i = i+1;
235 body(i).param.m = 0;
236 body(i).param.B_Th = diag([0, 0, 0]);
237 body(i).param.B_r_COG = [0;0;0];
238 body(i).cs.P_r_PO = sym([0;0;lS]);
239 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
240 body(i).tree.parent = i-1;
241
242 % LH_HAA
243 i = i+1;
244 body(i).param.m = mH;
245 body(i).param.B_Th = sym(diag([ThH.xx ThH.yy ThH.zz]));
246 body(i).param.B_r_COG = [0;0;sH];
247 body(i).cs.P_r_PO = sym([-b2hx;b2hy;0]);
248 body(i).cs.A_PB = eulerToRotMat_A_IB(qLH_HAA,0,0);
249 body(i).tree.parent = 1;
250
251 % LH_HFE
252 i = i+1;
253 body(i).param.m = mT;
254 body(i).param.B_Th = sym(diag([ThT.xx ThT.yy ThT.zz]));
255 body(i).param.B_r_COG = [0;0;sT];
256 body(i).cs.P_r_PO = sym([0;0;lH]);
257 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLH_HFE,0);
258 body(i).tree.parent = i-1;
259
260 % LH_KFE
261 i = i+1;
262 body(i).param.m = mS;
263 body(i).param.B_Th = sym(diag([ThS.xx ThS.yy ThS.zz]));
264 body(i).param.B_r_COG = [0;0;sS];
265 body(i).cs.P_r_PO = sym([0;0;lT]);
266 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLH_KFE,0);
267 body(i).tree.parent = i-1;
268
269 % LH_Foot
270 i = i+1;
271 body(i).param.m = 0;
272 body(i).param.B_Th = diag([0, 0, 0]);
273 body(i).param.B_r_COG = [0;0;0];
274 body(i).cs.P_r_PO = sym([0;0;lS]);
275 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
276 body(i).tree.parent = i-1;
277
278 % RH_HAA
279 i = i+1;
280 body(i).param.m = mH;
281 body(i).param.B_Th = sym(diag([ThH.xx ThH.yy ThH.zz]));
282 body(i).param.B_r_COG = [0;0;sH];
283 body(i).cs.P_r_PO = sym([-b2hx;-b2hy;0]);
284 body(i).cs.A_PB = eulerToRotMat_A_IB(qRH_HAA,0,0);
285 body(i).tree.parent = 1;
286
287 % RH_HFE
288 i = i+1;

```

```

289 body(i).param.m = mT;
290 body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
291 body(i).param.B_r_COG = [0;0;sT];
292 body(i).cs.P_r_PO = sym([0;0;lH]);
293 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRH_HFE,0);
294 body(i).tree.parent = i-1;
295
296 % RH_KFE
297 i = i+1;
298 body(i).param.m = mS;
299 body(i).param.B_Th = sym(diag([ThS_xx ThS_yy ThS_zz]));
300 body(i).param.B_r_COG = [0;0;sS];
301 body(i).cs.P_r_PO = sym([0;0;lT]);
302 body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRH_KFE,0);
303 body(i).tree.parent = i-1;
304
305 % RH_Foot
306 i = i+1;
307 body(i).param.m = 0;
308 body(i).param.B_Th = diag([0, 0, 0]);
309 body(i).param.B_r_COG = [0;0;0];
310 body(i).cs.P_r_PO = sym([0;0;lS]);
311 body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
312 body(i).tree.parent = i-1;
313
314 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
315 % Force/torque elements
316 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
317
318 % Torque at FL
319 i = 1;
320 ftel(i).type = 'rot';           % define it to be a rotational = torque
321 ftel(i).body_P = 1;           % body on which the reaction happens
322 ftel(i).body_B = 2;           % body on which the action happens
323 ftel(i).B_T = [TLF_HAA;0;0];   % torque vector, expressed in B frame
324
325 i = i+1;
326 ftel(i).type = 'rot';
327 ftel(i).body_P = 2;
328 ftel(i).body_B = 3;
329 ftel(i).B_T = [0;TLF_HFE;0];
330
331 i = i+1;
332 ftel(i).type = 'rot';
333 ftel(i).body_P = 3;
334 ftel(i).body_B = 4;
335 ftel(i).B_T = [0;TLF_KFE;0];
336
337 % Torque at FR
338 i = i+1;
339 ftel(i).type = 'rot';
340 ftel(i).body_P = 1;
341 ftel(i).body_B = 6;
342 ftel(i).B_T = [TRF_HAA;0;0];
343
344 i = i+1;
345 ftel(i).type = 'rot';
346 ftel(i).body_P = 6;
347 ftel(i).body_B = 7;
348 ftel(i).B_T = [0;TRF_HFE;0];
349
350 i = i+1;
351 ftel(i).type = 'rot';
352 ftel(i).body_P = 7;
353 ftel(i).body_B = 8;
354 ftel(i).B_T = [0;TRF_KFE;0];
355

```

```

356 % Torque at LH
357 i = i+1;
358 ftel(i).type = 'rot';
359 ftel(i).body_P = 1;
360 ftel(i).body_B = 10;
361 ftel(i).B.T = [TLH.HAA;0;0];
362
363 i = i+1;
364 ftel(i).type = 'rot';
365 ftel(i).body_P = 10;
366 ftel(i).body_B = 11;
367 ftel(i).B.T = [0;TLH.HFE;0];
368
369 i = i+1;
370 ftel(i).type = 'rot';
371 ftel(i).body_P = 11;
372 ftel(i).body_B = 12;
373 ftel(i).B.T = [0;TLH.KFE;0];
374
375 % Torque at RH
376 i = i+1;
377 ftel(i).type = 'rot';
378 ftel(i).body_P = 1;
379 ftel(i).body_B = 14;
380 ftel(i).B.T = [TRH.HAA;0;0];
381
382 i = i+1;
383 ftel(i).type = 'rot';
384 ftel(i).body_P = 14;
385 ftel(i).body_B = 15;
386 ftel(i).B.T = [0;TRH.HFE;0];
387
388 i = i+1;
389 ftel(i).type = 'rot';
390 ftel(i).body_P = 15;
391 ftel(i).body_B = 16;
392 ftel(i).B.T = [0;TRH.KFE;0];
393
394 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
395 % Projected Newton Euler equations
396 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
397 [sys, body, ftel] = computePNE(body,ftel,q,Dq,T,I_a_grav,param);
398
399 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
400 % Visualization
401 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
402 % plot StarlETH
403 plotBodies(body,param,paramDef,q,qDef);
404
405 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
406 % Get ground contact jacobians
407 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
408 % this would be the solution if all legs were at the ground
409 Js = [...
410     body(5).kin.I_J_O;...
411     body(9).kin.I_J_O;...
412     body(13).kin.I_J_O;...
413     body(17).kin.I_J_O];

```

3.4 Generating Code

There are different possibilities to generate code that can be embedded in simulations or controllers:

- .m function
- compiled mex functions
- C-code to embed

The matlab script *createFunctionFiles.m* gives you some example code.

3.4.1 Generating Matlab Functions and Mex Functions

The Symbolic Toolbox of Matlab provides the function *matlabFunction* to generate a matlab function from a symbolic matrix in a m-file:

```
1 % generate a matlab function
2 matlabFunction(sys.MpNE, 'file', 'matlabFunc/Mfunc', 'vars', [sys.q; sys.param]);
3 matlabFunction(sys.bpNE, 'file', 'matlabFunc/bfunc', 'vars', [sys.q; sys.dq; sys.param]);
4 matlabFunction(sys.gpNE, 'file', 'matlabFunc/gfunc', 'vars', [sys.q; sys.param]);
```

The generated files can be used to compile *mex-files* as follows:

```
1 % compile this matlab function to a mex function
2 emlc -o mexFunc/Mfunc_mex matlabFunc/Mfunc.m
3 emlc -o mexFunc/bfunc_mex matlabFunc/bfunc.m
4 emlc -o mexFunc/gfunc_mex matlabFunc/gfunc.m
```

Note: The mex-functions are executed much faster than the matlab-functions for systems with a lot of degrees of freedom.

3.4.2 Generating C-Code

The Symbolic Toolbox of Matlab comes along with the function *ccode* that generates C-code from a symbolic matrix. The function *ccode* is able to write optimized C-Code in a file. Note that the function can also print the code in the command window of Matlab, but that code is not optimized.

The function *ccode* makes extensive use of auxiliary variables that need to be defined, e.g. as 'const double'. The following function adds the definitions and renames the array name:

```
1 % function schar=genCCodeMatrix(filename, m, arrayname)
2 %
3 % -> generates C-Code from the symbolic matrix m.
4 % The code is temporarily stored in a file.
5 %
6 % INPUTS:
7 %   path          path to C-code temporary file (add trailing /)
8 %   filename      name of the temporary file
9 %   m             symbolic matrix
10 %   arrayname     name of the C-code array
11 %
12 % OUTPUTS:
13 %   schar        string containing the C-Code
```

The function creates a temporary file that could be included somewhere in your code, and outputs the C-code in string.

The Matlab function *genCCodeMatrix* does not add a definition of the array, e.g.

```
1 double MpNE[3][3];
```

because the definition might be in another location of your code, e.g. in a class as a member variable. Moreover, the array needs to be initialized with zero values, because the function *ccode* omits array entries that are zero.

The following Matlab function is an example for generating C-code:

```
1 % function genCCodeExampleFile(filename, sys, values)
2 %
3 % -> generates an example C-code source file that computes the ...
4 % parts of the
5 % equations of motions (EoM): MpNE*ddq + bpNE + gpNE = fpNE
6 %
7 % INPUTS:
8 %   path      path to C-code files
9 %   filename  name of the C-code file, e.g. 'eom_main.c'
10 %   sys       struct of symbolic EoM generated by genEoM.m
11 %   values    struct containing the values paramDef, qDef, dqDef, ...
12 %             and TDef
13 %
14 % Compile the source code by 'g++ eom_main.c -o eom_main -lm' and run
15 % './eom_main'.
```

It creates a simple application that computes the components of the EoM and prints them to the command line.

Bibliography

- [1] T. Kurz, P. Eberhard, C. Henninger, and W. Schiehlen. From neweul to Neweul-M2: symbolical equations of motion for multibody system analysis and synthesis. *Multibody System Dynamics*, 24(1):25–41, January 2010.