Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

MATLAB Tool

proNEu Documentation

Derivation of Analytical
Kinematics & **Dynamics**

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

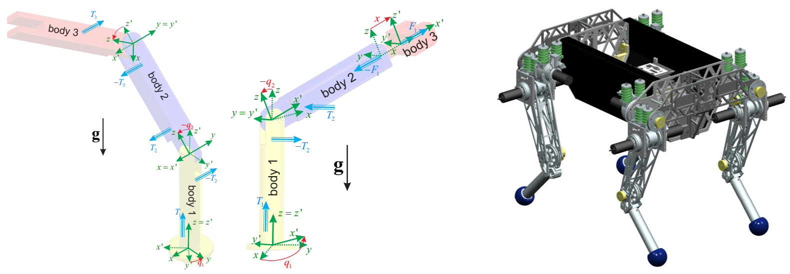
v1.1, Mar. 2012

www.asl.ethz.ch
www.leggedrobotics.ethz.ch

Authors:
Marco Hutter
Christian Gehring

Preamble

This is the manual for the (very simple, but still quite powerfull) Matlab tool *pro*NEu. The tool uses the MATLAB Symbolic Math Toolbox[1] 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.



(a) 3-link robot arm (b) prismatic joint (c) quadruped robot

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 *pro*NEu

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 Starl*ETH*
.../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

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 $\mathbf{q}_b$:

$$\mathbf{q} = \begin{pmatrix} \mathbf{q}_b \\ \mathbf{q}_r \end{pmatrix}. \tag{1.1}$$

```
% define generalized coordinates
syms x q1 q2 real
q = [x,q1,q2]';
% define generalized velocities
syms Dx Dq1 Dq2 real
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:

$${}_B\mathbf{r}_{OP} = {}_B\mathbf{r}_{OP}(\mathbf{q}). \tag{1.2}$$

```
% define certain parameters
syms l real
% position vector
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:

$$
\begin{aligned}
{}_I\mathbf{r} \rightarrow {}_I\dot{\mathbf{r}} &= \frac{d_I\mathbf{r}}{dt} && (1.3)\\
{}_B\mathbf{r} \rightarrow {}_B\dot{\mathbf{r}} &= \frac{d_B\mathbf{r}}{dt} + {}_B\,\omega_{IB} \times {}_B\,\mathbf{r} && (1.4)
\end{aligned}
$$

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).

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

1.1.4 Rotation

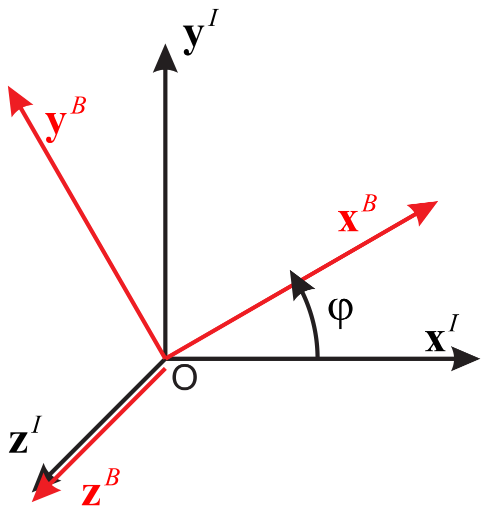


Figure 1.1: Rotated coordinate system B around z axis

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

$$
\begin{aligned}
{}_I\mathbf{r} &= \mathbf{A}_{IB}\,{}_B\mathbf{r} && && (1.5)\\
{}_B\mathbf{r} &= \mathbf{A}_{BI}\,{}_I\mathbf{r} && \mathbf{A}_{BI} = \mathbf{A}_{IB}^T && (1.6)\\
{}_C\mathbf{r} &= \mathbf{A}_{CI}\,{}_I\mathbf{r} && \mathbf{A}_{CI} = \mathbf{A}_{CB}\mathbf{A}_{BI} && (1.7)
\end{aligned}
$$

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

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

```
% is equivalent to
AIB = eulerToRotMat_A_IB(alpha,0,0)*...          % rot around x
            eulerToRotMat_A_IB(0,beta,0)*...        % rot around y
            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 ${}_I\mathbf{\Omega}$ of a rigid body with body fixed frame B is:

$$
\begin{aligned}
{}_I\tilde{\mathbf{\Omega}} &= {}_I\tilde{\boldsymbol{\omega}}_{IB} = \dot{\mathbf{A}}_{IB}\mathbf{A}_{IB}^T && (1.8)\\
\tilde{\mathbf{\Omega}} &= \begin{bmatrix} 0 & -\Omega^z & \Omega^y \\ \Omega^z & 0 & -\Omega^x \\ -\Omega^y & \Omega^x & 0 \end{bmatrix}, \quad \underset{skew}{\overset{unskew}{\rightleftharpoons}} \quad \mathbf{\Omega} = \begin{pmatrix} \Omega^x \\ \Omega^y \\ \Omega^z \end{pmatrix} && (1.9)
\end{aligned}
$$

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

1.1.6 Jacobian

The Jacobian $\mathbf{J}$ is given through:

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

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

$$
\dot{\mathbf{r}} = \mathbf{J}\dot{\mathbf{q}} \tag{1.11}
$$

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

$$
\boldsymbol{\tau} = \mathbf{J}^T\mathbf{F} \tag{1.12}
$$

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 $\mathbf{J}_R = \frac{\partial \mathbf{\Omega}(\mathbf{q})}{\partial \dot{\mathbf{q}}}$ that are identical for all points of one single rigid body.

```
% get jacobian from position vector
J = jacobian(r,q);
% get jacobian from rotation speed
Jr = jacobian(Omega,dq);
```

1.2 Dynamics

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

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

with

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

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

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

$$\mathbf{N}_{S_i} = \boldsymbol{\theta}_{S_i} \boldsymbol{\Omega}_i \qquad \text{angular momentum} \tag{1.16}$$

$$\mathbf{F}_{S_i}^a \qquad \text{external forces} \tag{1.17}$$

$$\mathbf{T}_i^a \qquad \text{external torques} \tag{1.18}$$

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

$$\dot{\mathbf{p}}_i = m_i \ddot{\mathbf{r}}_{OS_i} \tag{1.19}$$

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

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

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

$$\boldsymbol{\Omega}_i = \mathbf{J}_{R_i}\dot{\mathbf{q}} \tag{1.22}$$

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

the elements of (1.13) are obtained by

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

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

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

Chapter 2

Matlab Tool

The presented tool uses the Symbolic Math Toolbox[1].

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:

```
% => start setting up the kinematic structure.  Each link needs to ...
    have all the folowing struct elements
% B indicates bodyframe
% P indicates coordinate system of parent element

% body(i).param.m:          body mass
% body(i).param.B_Th:    inertia tensor in body frame w.r.t. CoG
% body(i).param.B_r_COG: CoG in body frame
% body(i).cs.P_r_PO:        position of origin in parent CS
% body(i).cs.A_PB:          rotation from parent CS
% body(i).tree.parent:   tree parent (0=inertial frame)

%% Body 1
i = 1;
% body mass
body(i).param.m = m1;
body(i).param.B_Th = diag([Th1_xx Th1_yy Th1_zz]);
body(i).param.B_r_COG = [0;0;s1];
body(i).cs.P_r_PO = sym([0;0;0]); % ensure a symbolic expression
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,q1);
body(i).tree.parent = 0;
```

Some notes on the notation:

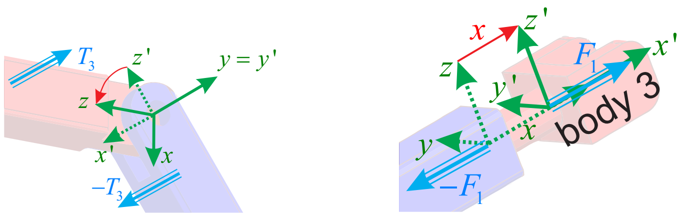
${}_B\boldsymbol{\theta}$:	Inertia of the body with respect to CoG expressed in body fixed frame B
${}_B\mathbf{r}_{CoG}$:	Vector from origin of body fixed frame to CoG expressed in body frame B
${}_P\mathbf{r}_{PO}$:	Translational vector from origin of parent frame P to origin of body frame B expressed 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.



(a) Torque actuator acting in a revolute joint (type = ’rot’).

(b) Force actuator acting in a prismatic joint (type = ’lin’).

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:

```
% torque element acting between environment and body 1
i = 1;
ftel(i).type = 'rot';        % define it to be a rotational = torque
ftel(i).body_P = 0;          % body on which the reaction happens
ftel(i).body_B = 1;          % body on which the action happens
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:

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

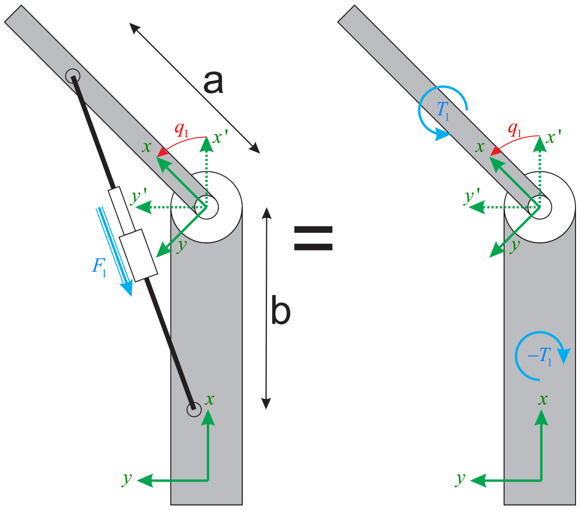


Figure 2.2: A cylinder in combination with a revolute joint has to be modeled as a torque actuator.

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

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

$$T_1 = F_1 b \sin \phi \tag{2.2}$$

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:

```
function [sys, body, ftel] = computePNE(body,ftel,q,dq,tau,I_a_grav)

% INPUT:
% * body:       kinematic tree
% * ftel:       force/torque elements
% * q:          generalized coordinates (symb) in desired order
% * Dq:         coresponding velocities
% * tau:        actuator force/torque array (symb)
% * I_a_grav:   gravity vector (R3)
%
% OUTPUT:
% * sys:        system struct containing dynamcis (all symbolic)
%   .MpNE:      mass matrix
%   .bpNE:      coriolis/centrifugal
%   .gpNE:      gravity terms
%   .SpNE:      actuator selection matrix
%   .fpNE:      =SpNE*T;
%   .param:     input parameters
%   .q:         generalized coordinates
%   .Dq:        generalized velocities
%   .tau        generalized actuator forces
%
% * body.kin:   body contains (among others) now also global ...
    kinematics as:
%   .A_IB:      rotation matrix from B to I (inertial/world frame)
```

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

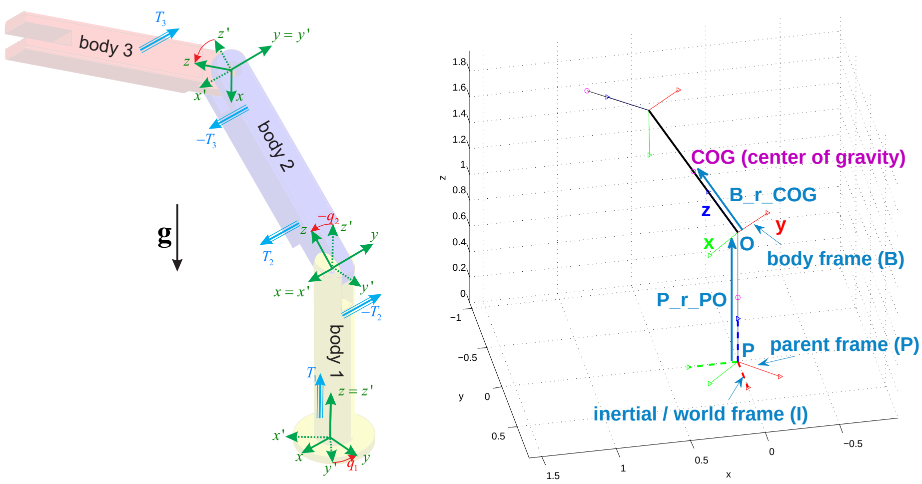
Chapter 3

Examples

This is a collection of validated (with the software Neweul-M^2[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



(a) 3-link arm

(b) Description of the second body

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

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

```
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <http://www.gnu.org/licenses/>.

clc
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimal coordinates and derivatives
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the minimal coordinates that you
% want to use. They can stand for rotational or prismatic joints.

syms q1 q2 q3 real            % define the three angles
q = [q1 q2 q3]';              % vector of generalized coordinates
qDef = [pi/4,pi/4, pi/4]';    % define some values for visualization

syms Dq1 Dq2 Dq3 real         % define the derivatives of the angles
dq = [Dq1 Dq2 Dq3]';          % derivatives of the gen. cord.
dqDef = [0,0,0]';             % define some values

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generalized actuator torques
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the generalized forces (for prismatic ...
    joints)
% and the generalized torques (for rotational joints)
syms T1 T2 T3 real           % define the three torques
T = [T1 T2 T3]';             % torque vector
TDef = [1,2,3]';             % define some values



%%%%%%%%%%%%%%%%%%%%%%
% Parameters
%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the parameters to describe
% the link properties
% i.e.      l1: length to next joint
%           s1: offset CoG
%           m1: link mass
%           Th1.. : inertia properties of link$

syms l1 s1 m1 Th1_xx Th1_yy Th1_zz real                 % link 1
syms l2 s2 m2 Th2_xx Th2_yy Th2_zz real                 % link 2
syms l3 s3 m3 Th3_xx Th3_yy Th3_zz real                 % link 3
% parameter vector
param = [l1 s1 m1 Th1_xx Th1_yy Th1_zz]';               % link 1
param = [param', l2 s2 m2 Th2_xx Th2_yy Th2_zz]';       % link 2
param = [param', l3 s3 m3 Th3_xx Th3_yy Th3_zz]';       % link 3
 % define some values for visualization
paramDef = [1 0.5 1 1 1 1]';                            % link 1
paramDef = [paramDef', 1 0.5 1 1 1 1]';                 % link 2
paramDef = [paramDef', 1 0.5 1 1 1 1]';                 % link 3

%%%%%%%%%%%%%%%%%%%%%%
% Gravity
%%%%%%%%%%%%%%%%%%%%%%
syms g real
I_a_grav = [0; 0; -g];          % gravity vector expressed in ...
    inertial frame
param = [param;g];              % add gravity to the parameter vector
paramDef = [paramDef;9.81];     % define the value
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Kinematic structure - kinematic tree
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% => start setting up the kinematic structure.
% Each link needs to have all the folowing struct elements
% B indicates body frame
% P indicates coordinate system of parent element

% Body 1
i = 1;
body(i).param.m = m1; % mass
body(i).param.B_Th = sym(diag([Th1_xx Th1_yy Th1_zz])); % inertia ...
    in B frame
body(i).param.B_r_COG = sym([0;0;s1]); % CoG position in B frame
body(i).cs.P_r_PO = sym([0;0;0]); % body frame origin in ...
    predecessor frame
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,q1); % rotation of body frame
body(i).tree.parent = 0; % parent element

% Body 2
i = 2;
body(i).param.m = m2;
body(i).param.B_Th = sym(diag([Th2_xx Th2_yy Th2_zz]));
body(i).param.B_r_COG = sym([0;0;s2]);
body(i).cs.P_r_PO = sym([0;0;l1]);
body(i).cs.A_PB = eulerToRotMat_A_IB(q2,0,0);
body(i).tree.parent = 1;

% Body 3
i = 3;
body(i).param.m = m3;
body(i).param.B_Th = sym(diag([Th3_xx Th3_yy Th3_zz]));
body(i).param.B_r_COG = sym([0;0;s3]);
body(i).cs.P_r_PO = sym([0;0;l2]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,q3,0);
body(i).tree.parent = 2;

%%%%%%%%%%%%%%%%%%%%%%%
% Force/torque elements
%%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the force / torque elements that act
% between the bodies

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

% torque element acting between body 1 and body 2
ftel(2).type = 'rot';       % define it to be a rotational = torque
ftel(2).body_P = 1;         % body on which the reaction happens
ftel(2).body_B = 2;         % body on which the action happens
ftel(2).B_T = [T2;0;0];     % torque vector, expressed in B frame

% torque element acting between body 2 and body 3
ftel(3).type = 'rot';       % define it to be a rotational = torque
ftel(3).body_P = 2;         % body on which the reaction happens
ftel(3).body_B = 3;         % body on which the action happens
ftel(3).B_T = [0;T3;0];     % torque vector, expressed in B frame

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Projected Newton Euler equations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% => compute the dynamics by applying the proj. Newton-Euler method
[sys, body, ftel] = computePNE(body,ftel,q,dq,T,I_a_grav,param);
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Save the data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% save the kinematics and dynamics
save('body','body')
save('sys','sys')
% save the numerical values in a struct
values.paramDef = paramDef;
values.qDef = qDef;
values.dqDef = dqDef;
values.TDef = TDef;
save('values','values')
% save the force/torque elements
save('ftel','ftel')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Visualization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot the robot arm
plotBodies(body,param,paramDef,q,qDef);
```

3.2 3-link Robot Arm with Prismatic Joint

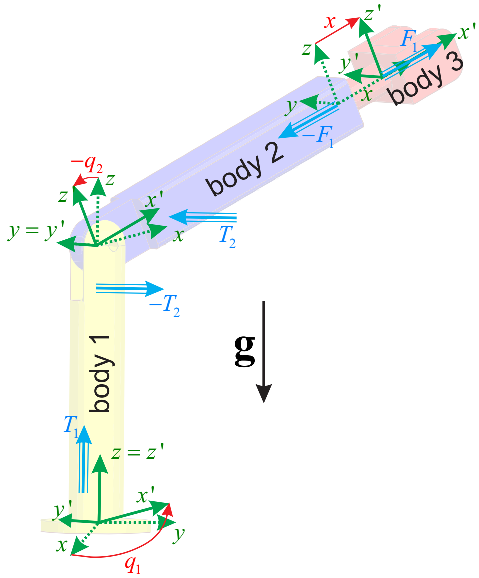


Figure 3.2: 3-link robot arm with two revolute joints ($\mathbf{z}, \mathbf{y}$) and one prismatic joint ($\mathbf{x}$).

```
% 3LinkRobotArmPrismaticLink_genEoM.m
%
% -> exmaple file to generate the EoM of a 3-link robot arm that ...
    has a
% prismatic joint as the last actuator.  For detailed information ...
    and a
% figure with the links and coordinate systems, please have a look ...
    at the
% documentation.
%
% proNEu: tool for symbolic EoM derivation
% Copyright (C) 2011  Marco Hutter, Christian Gehring
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <http://www.gnu.org/licenses/>.

clc
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimal coordinates and velocity
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the minimal coordinates that you
% want to use. They can stand for rotational or prismatic joints.
```

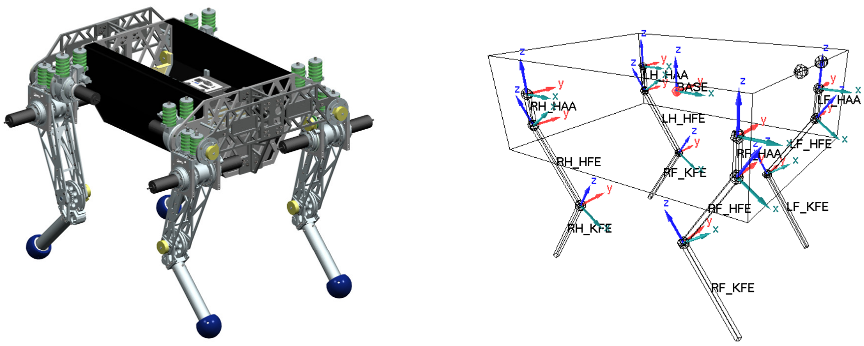
```

syms q1 q2 x real                % generalized coordinates
q = [q1 q2 x]';                  % q1, q2: 'rot'    x:'lin'
qDef = [pi/4,pi/4,0.2]';         % define here IC for visualization

syms Dq1 Dq2 Dx real             % generalized velocities
dq = [Dq1 Dq2 Dx]';
dqDef = [0,0,0]';

%%%%%%%%%%%%%%%%%
% Torques/Forces
%%%%%%%%%%%%%%%%%
% => define in this section the generalized forces (for prismatic ...
    joints)
% and the generalized torques (for rotational joints)

syms T1 T2 F1 real               % generalized Force/Torque vector
T = [T1 T2 F1]';
TDef = [1,2,3]';


%%%%%%%%%%%%%%%%%%%%%%
% Parameters
%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the parameters to describe the link ...
    properties
% i.e.      l1: length to next joint
%           s1: offset CoG
%           m1: link mass
%           Th1.. : inertia properties of link$

syms l1 s1 m1 Th1_xx Th1_yy Th1_zz real
syms l2 s2 m2 Th2_xx Th2_yy Th2_zz real
syms l3 s3 m3 Th3_xx Th3_yy Th3_zz real
param = [l1 s1 m1 Th1_xx Th1_yy Th1_zz]';       % this defines the ...
    order
param = [param', l2 s2 m2 Th2_xx Th2_yy Th2_zz]';
param = [param', l3 s3 m3 Th3_xx Th3_yy Th3_zz]';
paramDef = [1 0.5 1 1 1 1]';           % define here the default values
paramDef = [paramDef', 1 0.5 1 1 1 1]';
paramDef = [paramDef', 1 0.5 1 1 1 1]';


%%%%%%%%%%%%%%%%%%%%%%
% Gravity
%%%%%%%%%%%%%%%%%%%%%%
syms g real
I_a_grav = [0; 0; -g];
param = [param;g];
paramDef = [paramDef;9.81];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Kinematic structure - kinematic tree
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% => start setting up the kinematic structure.
% Each link needs to have all the folowing struct elements
% B indicates body frame
% P indicates coordinate system of parent element

% Body 1
i = 1;
body(i).param.m = m1;            % mass
body(i).param.B_Th = diag([Th1_xx Th1_yy Th1_zz]);  % inertia tensor
body(i).param.B_r_COG = [0;0;s1]; % center of gravity in body frame
body(i).cs.P_r_PO = sym([0;0;0]);  % position of origin in parent CS
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,q1); % rotation from ...
    parent CS
```

```
body(i).tree.parent = 0;     % tree parent (0=inertial frame)

% Body 2
i = 2;
body(i).param.m = m2;
body(i).param.B_Th = diag([Th2_xx Th2_yy Th2_zz]);
body(i).param.B_r_COG = [s2;0;0];
body(i).cs.P_r_PO = [0;0;l1];
body(i).cs.A_PB = eulerToRotMat_A_IB(0,q2,0);
body(i).tree.parent = 1;

% Body 3
i = 3;
body(i).param.m = m3;
body(i).param.B_Th = diag([Th3_xx Th3_yy Th3_zz]);
body(i).param.B_r_COG = [s3;0;0];
body(i).cs.P_r_PO = [l2+x;0;0];
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
body(i).tree.parent = 2;


%%%%%%%%%%%%%%%%%%%%%%%%
% Force/torque elements
%%%%%%%%%%%%%%%%%%%%%%%%
% => define in this section the force / torque elements that act
% between the bodies

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

% torque element acting between body 1 and body 2
i = 2;
ftel(i).type = 'rot';        % define it to be a rotational = torque
ftel(i).body_P = 1;          % body on which the reaction happens
ftel(i).body_B = 2;          % body on which the action happens
ftel(i).B_T = [0;T2;0];      % torque vector, expressed in B frame

% force element acting between body 2 and body 3
i = 3;
ftel(i).type = 'lin';            % define it to be a linear = force
ftel(i).body_P = 2;              % body on which the reaction happens
ftel(i).body_B = 3;              % body on which the action happens
ftel(i).P_r_R = sym([0;0;0]);    % point of reaction in P frame
ftel(i).B_r_A = sym([0;0;0]);    % point of action in B frame
ftel(i).B_F = [F1;0;0];          % force vector of action / this is ...
    optional


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Projected Newton Euler equations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[sys, body, ftel] = computePNE(body,ftel,q,dq,T,I_a_grav,param);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Save the data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% save the kinematics and dynamics
save('body','body')
save('sys','sys')
% save the numerical values in a struct
values.paramDef = paramDef;
values.qDef = qDef;
values.dqDef = dqDef;
```

```
values.TDef = TDef;
save('values','values')
% save the force/torque elements
save('ftel','ftel')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Visualization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
plotBodies(body,param,paramDef,q,qDef);
```

3.3 Quadruped Robot Starl*ETH*

For detailed information about our quadruped Starl*ETH* please consult our homepage[1]. 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).



(a) (b)

Figure 3.3: Free floating quadruped Starl*ETH* with a total of 18 DoF.

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

[1] leggedrobotics.ethz.ch

```
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <http://www.gnu.org/licenses/>.
%
% Description of link object
% body(i).param.m         = mass [kg]
% body(i).param.B_Th      = moment of inertia w.r.t. center of gravity
%                         described in local coordinate system [kg*m^2]
% body(i).param.B_r_COG   = position of center of gravity w.r.t. ...
    origin of
%                         local frame described in local coordinate ...
    system
% body(i).cs.P_r_PO       = position of origin of local frame w.r.t.
%                         origin of parent frame described in parent
%                         coordinate system
% body(i).cs.A_PB         = rotation matrix that rotates the parent ...
    frame to
%                         the local frame (3x3-matrix)
% body(i).kin.A_IB        = rotation matrix that rotates the local ...
    frame to
%                         the world frame (3x3-matrix)
% body(i).kin.dA_IB       = time derivative of A_IB (3x3-matrix)
% body(i).kin.A_BI
% body(i).kin.I_r_O      = position of origin of local frame described
%                         in world coordinate system
% body(i).kin.I_dr_O     = time derivative of I_r_O
% body(i).kin.I_r_COG    = position of center of gravity w.r.t. ...
    origin of
%                         world frame described in world coordinate ...
    system
% body(i).kin.I_dr_COG   = time derivative of I_r_COG
% body(i).kin.B_Omega    = angular velocity of joint  described
%                         in world coordinate system
% body(i).kin.B_Omega_tilde  = angular velocity of joint in ...
    skew-symmetric
%                         matrix described in world coordinate system
% body(i).kin.I_J_O      = Jacobian w.r.t. origin of local frame ...
    described
%                         in world coordinate system
% body(i).kin.I_dJ_O     = time derivative of I_J_O
% body(i).kin.I_J_COG    = Jacobian w.r.t. center of gravity ...
    described in
%                         world coordinate system
% body(i).kin.I_dJ_COG   = time derivative of I_J_COG
% body(i).kin.I_Jr       = rotational Jacobian described
%                         in world coordinate system
% body(i).kin.I_dJr      = time derivative of I_Jr
% body(i).dyn.B_F        = force vector described in local coord. ...
    system
% body(i).dyn.B_T        = torque vector described in local coord. ...
    system
% body(i).dyn.I_F_grav   = gravity force vector described in world c.s.
% body(i).tree.parent    = index of parent
%
% LF = left front
% RF = right front
% LH = left hind
% RH = right hind
% HAA = hip abduction/ adduction
% HFE = hip flexion / extension
% KFE = knee flexion / extension

clc
clear

%%%%%%%%%%%%%%%%%%%%%%
% Minimal coordinates
%%%%%%%%%%%%%%%%%%%%%%
```

```
syms qX qY qZ qAL qBE qGA real      % main body free floating
q = [qX qY qZ qAL qBE qGA]';
qDef = zeros(6,1);
syms qLF_HAA qLF_HFE qLF_KFE real   % LF (left front) leg
q = [q',qLF_HAA qLF_HFE qLF_KFE]';
qDef = [qDef', 0 pi/4 -pi/2]';
syms qRF_HAA qRF_HFE qRF_KFE real   % RF (right front) leg
q = [q',qRF_HAA qRF_HFE qRF_KFE]';
qDef = [qDef', 0 pi/4 -pi/2]';
syms qLH_HAA qLH_HFE qLH_KFE real   % LH (left hind) leg
q = [q',qLH_HAA qLH_HFE qLH_KFE]';
qDef = [qDef', 0 -pi/4 pi/2]';
syms qRH_HAA qRH_HFE qRH_KFE real   % RH (right hind) leg
q = [q',qRH_HAA qRH_HFE qRH_KFE]';
qDef = [qDef', 0 -pi/4 pi/2]';

% velocity
syms DqX DqY DqZ DqAL DqBE DqGA real % main body velocity
Dq = [DqX DqY DqZ DqAL DqBE DqGA]';
syms DqLF_HAA DqLF_HFE DqLF_KFE real
Dq = [Dq',DqLF_HAA DqLF_HFE DqLF_KFE]';
syms DqRF_HAA DqRF_HFE DqRF_KFE real
Dq = [Dq',DqRF_HAA DqRF_HFE DqRF_KFE]';
syms DqLH_HAA DqLH_HFE DqLH_KFE real
Dq = [Dq',DqLH_HAA DqLH_HFE DqLH_KFE]';
syms DqRH_HAA DqRH_HFE DqRH_KFE real
Dq = [Dq',DqRH_HAA DqRH_HFE DqRH_KFE]';
DqDef = zeros(size(Dq));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
syms mB sB ThB_xx ThB_yy ThB_zz real
param = [mB sB ThB_xx ThB_yy ThB_zz]';
paramDef = [1 0 1 1 1]';
syms mH sH lH ThH_xx ThH_yy ThH_zz real
param = [param',mH sH lH ThH_xx ThH_yy ThH_zz]';
paramDef = [paramDef',1 -0.01 -0.02 1 1 1]';
syms mT sT lT ThT_xx ThT_yy ThT_zz real
param = [param',mT sT lT ThT_xx ThT_yy ThT_zz]';
paramDef = [paramDef',1 -0.1 -0.2 1 1 1]';
syms mS sS lS ThS_xx ThS_yy ThS_zz real
param = [param',mS sS lS ThS_xx ThS_yy ThS_zz]';
paramDef = [paramDef',1 -0.1 -0.2 1 1 1]';
syms b2hx b2hy real
param = [param',b2hx b2hy]';
paramDef = [paramDef',0.2 0.15]';
syms g real;
param = [param',g]';
paramDef = [paramDef',9.81]';


%%%%%%%%%%%%%%%%%%%%%%
% Torques
%%%%%%%%%%%%%%%%%%%%%%
syms TLF_HAA TLF_HFE TLF_KFE real   % LF torques
T = [TLF_HAA TLF_HFE TLF_KFE]';
syms TRF_HAA TRF_HFE TRF_KFE real   % RF torques
T = [T',TRF_HAA TRF_HFE TRF_KFE]';
syms TLH_HAA TLH_HFE TLH_KFE real   % LH torques
T = [T',TLH_HAA TLH_HFE TLH_KFE]';
syms TRH_HAA TRH_HFE TRH_KFE real   % RH torques
T = [T', TRH_HAA TRH_HFE TRH_KFE]';

%%%%%%%%%%%%%%%%%%%%%%
% Gravity
%%%%%%%%%%%%%%%%%%%%%%
```

```
% gravitational constant
I_a_grav = [0; 0; -g];


%% DYNAMICS

% Main Body
i = 1;
body(i).param.m = mB;
body(i).param.B_Th = sym(diag([ThB_xx ThB_yy ThB_zz]));
body(i).param.B_r_COG = sym([0;0;sB]);
body(i).cs.P_r_PO = sym([qX;qY;qZ]);
body(i).cs.A_PB = eulerToRotMat_A_IB(qAL,qBE,qGA);
body(i).tree.parent = 0;

% LF_HAA
i = i+1;
body(i).param.m = mH;
body(i).param.B_Th = sym(diag([ThH_xx ThH_yy ThH_zz]));
body(i).param.B_r_COG = [0;0;sH];
body(i).cs.P_r_PO = sym([b2hx;b2hy;0]);
body(i).cs.A_PB = eulerToRotMat_A_IB(qLF_HAA,0,0);
body(i).tree.parent = 1;

% LF_HFE
i = i+1;
body(i).param.m = mT;
body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
body(i).param.B_r_COG = [0;0;sT];
body(i).cs.P_r_PO = sym([0;0;lH]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLF_HFE,0);
body(i).tree.parent = i-1;

% LF_KFE
i = i+1;
body(i).param.m = mS;
body(i).param.B_Th = sym(diag([ThS_xx ThS_yy ThS_zz]));
body(i).param.B_r_COG = [0;0;sS];
body(i).cs.P_r_PO = sym([0;0;lT]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLF_KFE,0);
body(i).tree.parent = i-1;

% LF_Foot
i = i+1;
body(i).param.m = 0;
body(i).param.B_Th = diag([0, 0, 0]);
body(i).param.B_r_COG = [0;0;0];
body(i).cs.P_r_PO = sym([0;0;lS]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
body(i).tree.parent = i-1;

% RF_HAA
i = i+1;
body(i).param.m = mH;
body(i).param.B_Th = sym(diag([ThH_xx ThH_yy ThH_zz]));
body(i).param.B_r_COG = [0;0;sH];
body(i).cs.P_r_PO = sym([b2hx;-b2hy;0]);
body(i).cs.A_PB = eulerToRotMat_A_IB(qRF_HAA,0,0);
body(i).tree.parent = 1;

% RF_HFE
i = i+1;
body(i).param.m = mT;
body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
body(i).param.B_r_COG = [0;0;sT];
body(i).cs.P_r_PO = sym([0;0;lH]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRF_HFE,0);
```

```
body(i).tree.parent = i-1;

% RF_KFE
i = i+1;
body(i).param.m = mS;
body(i).param.B_Th = sym(diag([ThS_xx ThS_yy ThS_zz]));
body(i).param.B_r_COG = [0;0;sS];
body(i).cs.P_r_PO = sym([0;0;lT]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRF_KFE,0);
body(i).tree.parent = i-1;

% RF_Foot
i = i+1;
body(i).param.m = 0;
body(i).param.B_Th = diag([0, 0, 0]);
body(i).param.B_r_COG = [0;0;0];
body(i).cs.P_r_PO = sym([0;0;lS]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
body(i).tree.parent = i-1;

% LH_HAA
i = i+1;
body(i).param.m = mH;
body(i).param.B_Th = sym(diag([ThH_xx ThH_yy ThH_zz]));
body(i).param.B_r_COG = [0;0;sH];
body(i).cs.P_r_PO = sym([-b2hx;b2hy;0]);
body(i).cs.A_PB = eulerToRotMat_A_IB(qLH_HAA,0,0);
body(i).tree.parent = 1;

% LH_HFE
i = i+1;
body(i).param.m = mT;
body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
body(i).param.B_r_COG = [0;0;sT];
body(i).cs.P_r_PO = sym([0;0;lH]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLH_HFE,0);
body(i).tree.parent = i-1;

% LH_KFE
i = i+1;
body(i).param.m = mS;
body(i).param.B_Th = sym(diag([ThS_xx ThS_yy ThS_zz]));
body(i).param.B_r_COG = [0;0;sS];
body(i).cs.P_r_PO = sym([0;0;lT]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qLH_KFE,0);
body(i).tree.parent = i-1;

% LH_Foot
i = i+1;
body(i).param.m = 0;
body(i).param.B_Th = diag([0, 0, 0]);
body(i).param.B_r_COG = [0;0;0];
body(i).cs.P_r_PO = sym([0;0;lS]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
body(i).tree.parent = i-1;

% RH_HAA
i = i+1;
body(i).param.m = mH;
body(i).param.B_Th = sym(diag([ThH_xx ThH_yy ThH_zz]));
body(i).param.B_r_COG = [0;0;sH];
body(i).cs.P_r_PO = sym([-b2hx;-b2hy;0]);
body(i).cs.A_PB = eulerToRotMat_A_IB(qRH_HAA,0,0);
body(i).tree.parent = 1;

% RH_HFE
i = i+1;
```

```
body(i).param.m = mT;
body(i).param.B_Th = sym(diag([ThT_xx ThT_yy ThT_zz]));
body(i).param.B_r_COG = [0;0;sT];
body(i).cs.P_r_PO = sym([0;0;lH]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRH_HFE,0);
body(i).tree.parent = i-1;

% RH_KFE
i = i+1;
body(i).param.m = mS;
body(i).param.B_Th = sym(diag([ThS_xx ThS_yy ThS_zz]));
body(i).param.B_r_COG = [0;0;sS];
body(i).cs.P_r_PO = sym([0;0;lT]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,qRH_KFE,0);
body(i).tree.parent = i-1;

% RH_Foot
i = i+1;
body(i).param.m = 0;
body(i).param.B_Th = diag([0, 0, 0]);
body(i).param.B_r_COG = [0;0;0];
body(i).cs.P_r_PO = sym([0;0;lS]);
body(i).cs.A_PB = eulerToRotMat_A_IB(0,0,0);
body(i).tree.parent = i-1;

%%%%%%%%%%%%%%%%%%%%%%%%
% Force/torque elements
%%%%%%%%%%%%%%%%%%%%%%%%

% Torque at FL
i = 1;
ftel(i).type = 'rot';        % define it to be a rotational = torque
ftel(i).body_P = 1;          % body on which the reaction happens
ftel(i).body_B = 2;          % body on which the action happens
ftel(i).B_T = [TLF_HAA;0;0];     % torque vector, expressed in B frame

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 2;
ftel(i).body_B = 3;
ftel(i).B_T = [0;TLF_HFE;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 3;
ftel(i).body_B = 4;
ftel(i).B_T = [0;TLF_KFE;0];

% Torque at FR
i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 1;
ftel(i).body_B = 6;
ftel(i).B_T = [TRF_HAA;0;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 6;
ftel(i).body_B = 7;
ftel(i).B_T = [0;TRF_HFE;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 7;
ftel(i).body_B = 8;
ftel(i).B_T = [0;TRF_KFE;0];
```

```
% Torque at LH
i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 1;
ftel(i).body_B = 10;
ftel(i).B_T = [TLH_HAA;0;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 10;
ftel(i).body_B = 11;
ftel(i).B_T = [0;TLH_HFE;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 11;
ftel(i).body_B = 12;
ftel(i).B_T = [0;TLH_KFE;0];

% Torque at RH
i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 1;
ftel(i).body_B = 14;
ftel(i).B_T = [TRH_HAA;0;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 14;
ftel(i).body_B = 15;
ftel(i).B_T = [0;TRH_HFE;0];

i = i+1;
ftel(i).type = 'rot';
ftel(i).body_P = 15;
ftel(i).body_B = 16;
ftel(i).B_T = [0;TRH_KFE;0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Projected Newton Euler equations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[sys, body, ftel] = computePNE(body,ftel,q,Dq,T,I_a_grav,param);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Visualization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot StarlETH
plotBodies(body,param,paramDef,q,qDef);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get ground contact jacobians
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% this would be the solution if all legs were at the ground
Js = [...
    body(5).kin.I_J_O;...
    body(9).kin.I_J_O;...
    body(13).kin.I_J_O;...
    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:

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

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

```
% compile this matlab function to a mex function
emlc -o mexFunc/Mfunc_mex matlabFunc/Mfunc.m
emlc -o mexFunc/bfunc_mex matlabFunc/bfunc.m
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:

```
% function schar=genCCodeMatrix(filename, m, arrayname)
%
% -> generates C-Code from the symbolic matrix m.
% The code is temporarily stored in a file.
%
% INPUTS:
%   path          path to C-code temporary file (add trailing /)
%   filename      name of the temporary file
%   m             symbolic matrix
%   arrayname     name of the C-code array
%
% OUTPUTS:
%   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.

```
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:

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