Multibody Dynamics B - Assignment 9
ME41055
Prof. Arend L. Schwab
Head TA: Simon vd. Helm

Rick Staa
#4511328
Lab Date: 31/05/2018
Due Date: 07/05/2018

TA: Shambhuraj Sawant

Intro: 6/6
a: 0.5/0.5
b: 1/1
c: 0.25/0.25
d: 0.5/0.5
e: 0.25/0.25
f: 1/1
g: 0.5/0.5

1 Statement of integrity

my homework is completely in acc
the Academic Integrity

Figure 1: My handwritten statement of integ

2 Acknowledgements

I used [1] in making this assignment when finished I compared i
Kumar (4743873) Niels Hakvoort.

3 Setup overview

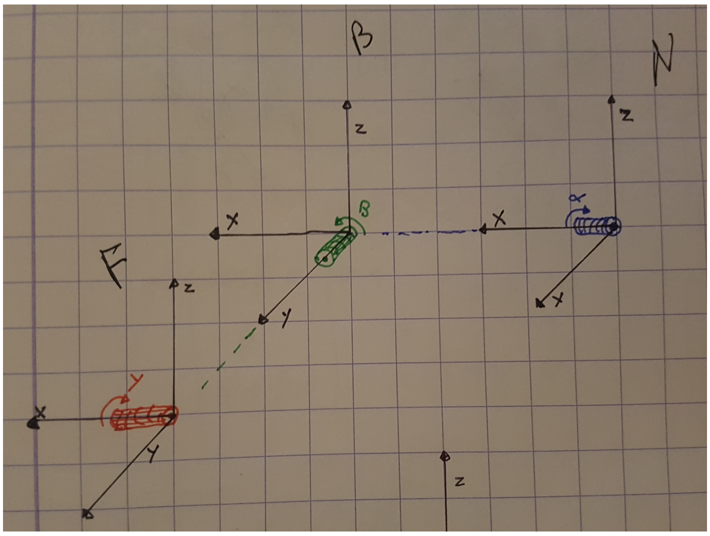

Figure 2: Cans and series representation of the arm mechanism

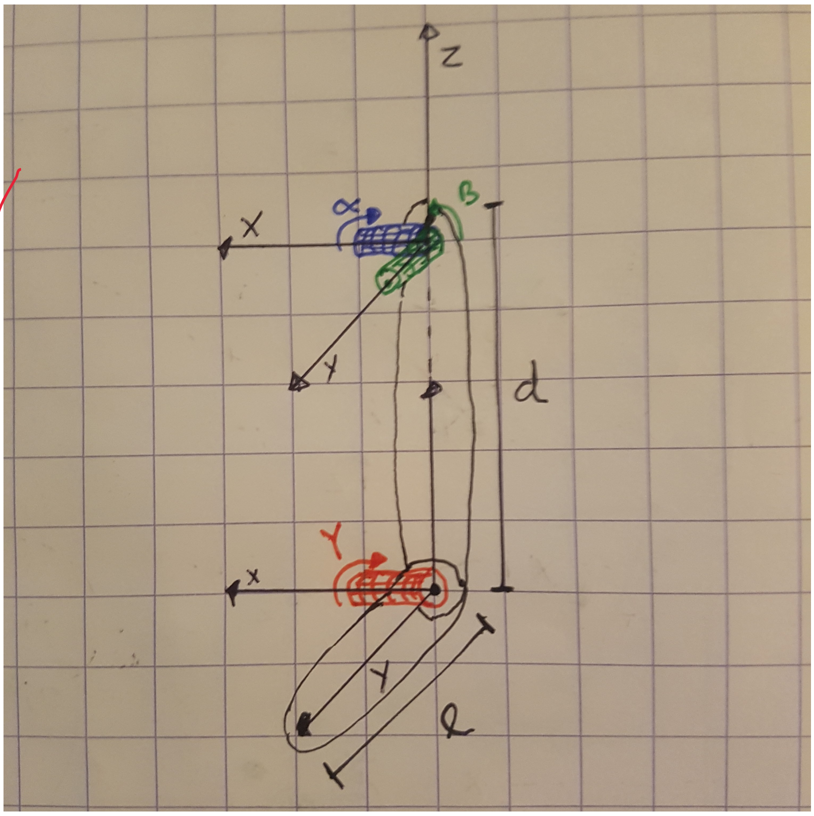

Figure 3: Cans and series representation also including the arm segments

4 Problem Statement

In this assignment we left our 2D world behind and stept into the magical world of 3d multi-body dynamics. We did this by examining a 3d arm model. This model had the following parameters:

$$L_1 = 0.3m \tag{1}$$
$$L_2 = 0.4m \tag{2}$$
$$m_1 = 3m \tag{3}$$
$$m_2 = 3m \tag{4}$$
$$g = 9.81m/s^2 \tag{5}$$
$$\tag{6}$$

In the first questions of the assignments (question a-e) we were asked to neglect eh mass moments of inertia of the rigid bodies. In the last questions of the assignment (questions f-g) we were asked to also take this moments of inertia in account. To solve this problem the following generalized state was defined:

$$q = \begin{bmatrix} \alpha & \beta & \gamma & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix} \tag{7}$$

The full state was defined as:

$$x = \begin{bmatrix} x_1 & y_1 & z_1 & x_2 & y_2 & z_2 & \dot{x_1} & \dot{y_1} & \dot{z_1} & \dot{x_2} & \dot{y_2} & \dot{z_1} \end{bmatrix} \tag{8}$$

To solve for the motion of the arm mechanism we need the following ingredients:

1. Three rotation matrices (R_α, R_β and R_γ) to express the COM coordinates into generalized coordinates.
2. The TMT method to derive the EOM
3. A numerical intergration method (ODE113)

Rotation matrices

The rotation matrices for a rotation around the body frames can be or derived by looking at figure 2 or by using dot products. For our problem we get the following rotation matrices:

$$R_\alpha = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{9}$$

$$R_\beta = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \tag{10}$$

$$R_\gamma = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix} \tag{11}$$

TMT Method

The Virtual Power TMT method is like the Lagrange method but differs in the fact that it doesn't go into the energy domain. Instead it stays in the forces domain and uses an incremental approach to obtain the equations of motion. We can derive the method by first looking at the virtual power equation with included D'Alembert forces:

$$\delta P = (F - M\ddot{x})\delta\dot{x} \tag{12}$$

Following the TMT method makes use of a transformation matrix T to transform the COM positions to the generalized coordinates. As a result we obtain the following equation:

$$\delta P = \delta \dot{x}_i (F_i - M_{ik} \ddot{x}_k) + \delta \dot{q}_k Q_j \tag{13}$$

Following the virtual accelerations can be derived out of the state x (eq. 29) and filled in in equation 13. After noting that this equation must hold for all virtual velocities and rearranging the equation a bit we obtain:

$$\bar{M} \ddot{q} = \bar{f} \tag{14}$$

Where:

$$\bar{M} = T^T M T \quad \text{and} \quad \bar{f} = T^T (F - MG) - Q \tag{15}$$

In this equation M represents the system mass matrix, F the external force vector,G convective terms, Q the generalized forces working on the body and T the transformation matrix. This transformation matrix can be calculated by taking the Jacobian of the full states vector x w.r.t. the general coordinates. To be able to do this we first need to express the COM coordinates in terms of generalized coordinates. This is done by using the earlier mentioned R_α, R_β and R_γ. For this problem our full state matrix x is defined as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R_\alpha R_\beta \, {}^B\mathrm{r}_{com1} \tag{16}$$

$$\begin{bmatrix} x_2 \\ y_3 \\ z_4 \end{bmatrix} = R_\alpha R_\beta R_\gamma \, {}^B\mathrm{r}_{com2} \tag{17}$$

This results in the following full state:

$$x = \begin{pmatrix} -\frac{\sin(\beta)}{10} \\ \frac{\cos(\beta)\,\sin(\alpha)}{10} \\ -\frac{\cos(\alpha)\,\cos(\beta)}{10} \\ \frac{\sin(\beta)\,\sin(\gamma)}{5} - \frac{3\,\sin(\beta)}{10} \\ \frac{\cos(\alpha)\,\cos(\gamma)}{5} + \frac{3\,\cos(\beta)\,\sin(\alpha)}{10} - \frac{\cos(\beta)\,\sin(\alpha)\,\sin(\gamma)}{5} \\ \frac{\cos(\gamma)\,\sin(\alpha)}{5} - \frac{3\,\cos(\alpha)\,\cos(\beta)}{10} + \frac{\cos(\alpha)\,\cos(\beta)\,\sin(\gamma)}{5} \end{pmatrix} \tag{18}$$

The mass matrix of the first part of this assignment is defined as:

$$M = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & m_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & m_2 \end{bmatrix} \tag{19}$$

The external force vector is defined as:

$$x = \begin{bmatrix} 0 & 0 & -m_1 g & 0 & 0 & -m_2 * g \end{bmatrix} \tag{20}$$

In our example the generalised forces Q are:

$$x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{21}$$

The T and G terms in equation 31 were derived using the symbolic toolbox of MAT LAB and will therefore not be displayed here. After obtaining all these terms the $\ddot{q}$ can be calculated and then passed to the ode solver. The $\ddot{q}$ is calculated by solving the system of equations depicted in 14:

$$\ddot{q} = \bar{M}^{-1} \bar{f} \tag{22}$$

ODE 113

The ode113 solver of MATLAB was used to perform the integration. This solver uses a non-stiff differential equations variable order method. For more information on this ode solver you are referred to the MATLAB DOCUMENTATION.

5 Questions

Question a

See sketch above in figure 2 and 3.

Question b

In this question we were asked to derive the EOM when the moments of inertia's of the body were not taken into account. An explanation of how this was done can be found in 4. The implementation in MATLAB can be found in 5. We were further asked to examine if the derivation of the EOM was done correctly by looking at a number of simple configurations.

Configuration 1

We can first look at the case where both arms are hanging down without any initial velocity. The initial state of this configuration is:

$$q = \begin{bmatrix} 0 & 0 & -pi/2 & 0 & 0 & 0 \end{bmatrix} \tag{23}$$

The result found with this initial state is:

$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.1139^{-28} \\ 0 \\ -0.0414^{-28} \end{bmatrix} \tag{24}$$

This result is as expected since both arms are parallel to the gravity component and thus no moment is created.

Configuration 2

We can also look at the case which both arms are pointing upwards. In this case we would also expect the acceleration in all directions to be equal to zeros. The initial state for this configuration is:

$$q = \begin{bmatrix} pi & 0 & -\pi/2 & 0 & 0 & 0 \end{bmatrix} \tag{25}$$

For this case we get the following result:

$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.2071^{-28} \end{bmatrix} \tag{26}$$

Configuration 3

Now let's look at a more interesting situation in which the upper arm hangs down in the negative z-direction and the lower arm points nord in the positive y-direction. For this configuration we get the following initial state:

$$q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{27}$$

For this case we get the following result:

$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -49.0500 \end{bmatrix} \tag{28}$$

Further $\ddot{(x_2)} = -9.81[m/s^2]$

This result is expected since gravity is pulling the arm to the negative z direction. This causes both a negative linear acceleration of the COM of segment 2 as a negative rotational acceleration of segment 2.

Configuration 3

Lastly let look what happens when we abduct the arm. We then get the following initial state:

$$q = \begin{bmatrix} 0 & 0.5\pi & 0 & 0 & 0 & 0 \end{bmatrix} \tag{29}$$

For this case we get the following result:

$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 98.100000000000009 \\ -98.100000000000009 \\ 0 \end{bmatrix} \tag{30}$$

Further $\ddot{(z_1)} = -9.81[m/s^2]$ and $\ddot{(z_2)} = -9.81[m/s^2]$

This result is expected since in this orientation gravity is pulling in the negative z direction on both arm segments. As a result of this force both segments will experience a moment arround the β and γ joints. In the upper arm shoulder joint this results in a positive angular acceleration $\ddot{\beta}$ and in the lower arm joint this results in a negative angular acceleration γ (see 3).

Question c

Since in an equilibrium the angular accelerations are equal to 0 the 3 torques can be derived by rewriting equation 14 in the following way:

$$Q = T^T(F - MG) \tag{31}$$

This calculation is performed in the matlab script in 5. The resulting torques for the case where $\alpha = 110\,\mathrm{deg}, \beta = -20\,\mathrm{deg} \quad \text{and} \quad \gamma = -20\,\mathrm{deg}$ we get:

$$M_0 = \begin{bmatrix} -10.280852679463344 \\ -1.612553948844058 \\ -0.114084912734962 \end{bmatrix} \tag{32}$$

Question d

If we copy this with MATLAB accuracy set on "format short" we get the following result:

If we however automatically pass it through the integration function we get a very accurate result:

From the fact that the results get better when we increase the calculation precision we can see that the drift is cased by round off errors which accumulate when time goes on. Even in the case when we use the maximal precision (figure 5 there is a small drift. For the 5 second use in this example the drift in the maximum precision case is very small, however decrease the step size

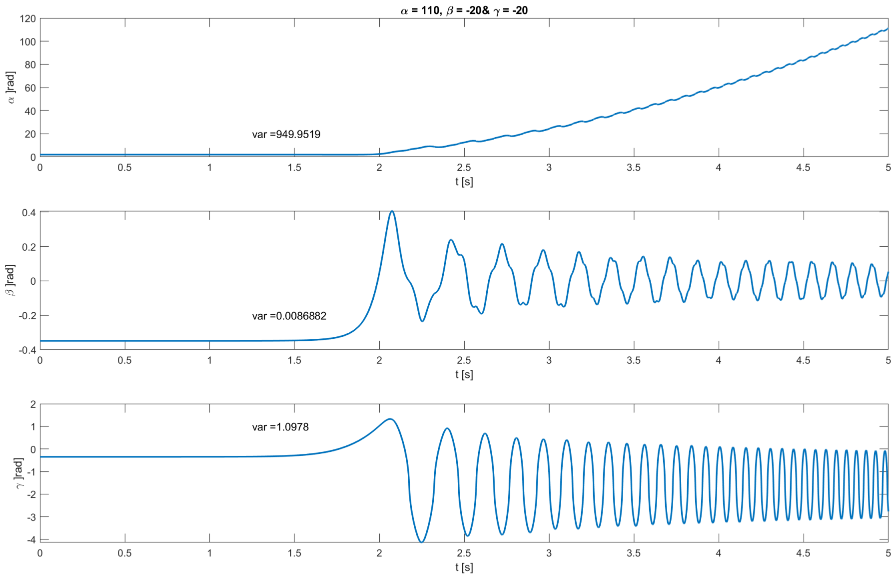

Figure 4: Simulation result when we use 4 decimals while copying

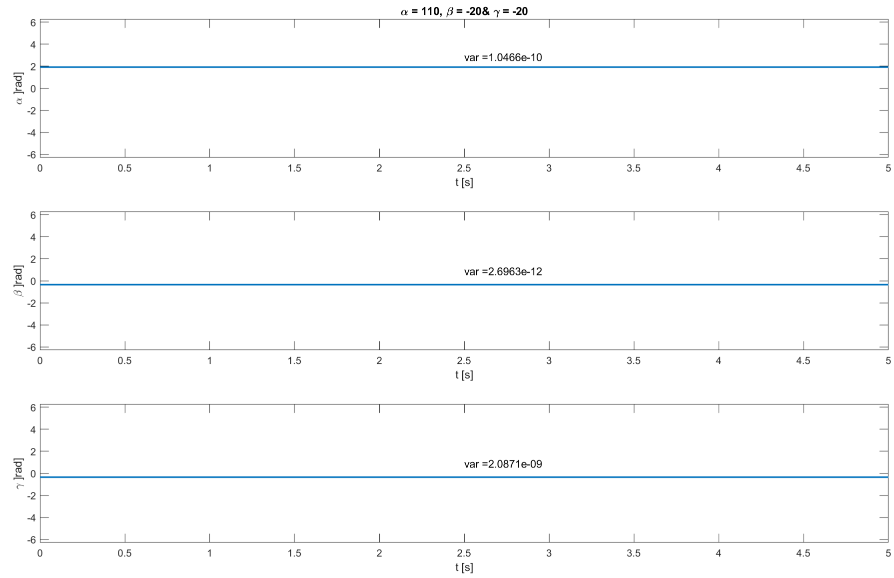

Figure 5: Simulation result when we use 4 decimals while copying

also with this level of precision there still is drift. This drift becomes noticeable for longer simulation times.

Question e

For the case that we start at $\alpha = 30\,\text{deg}, \beta = -20\,\text{deg}$ and $\gamma = -20\,\text{deg}$ we get the following torques:

$$M_0 = \begin{bmatrix} -11.266905886022014 \\ 4.083129932327170 \\ -5.507705294606232 \end{bmatrix} \tag{33}$$

When we then run the simulation we get similar behavior to question d:

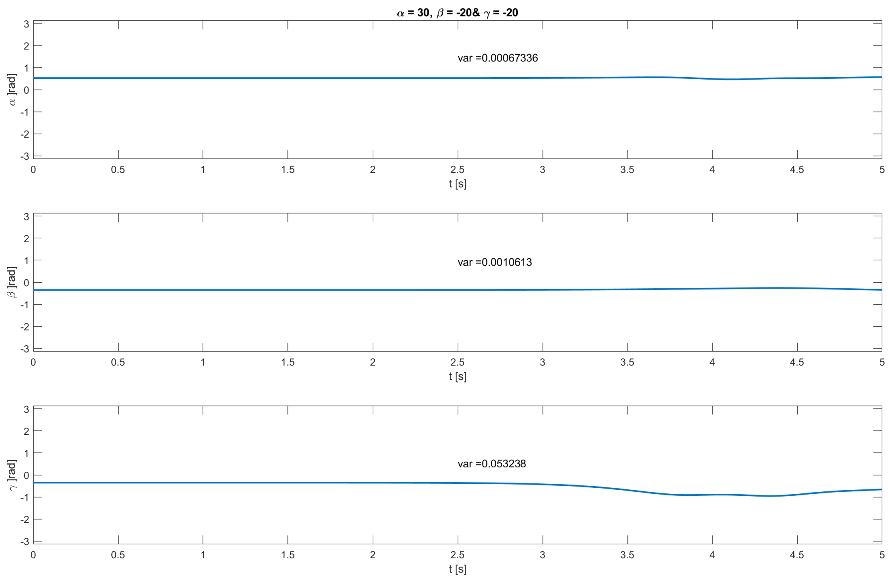

Figure 6: Simulation result when we use 4 decimals while copying

The drift is in this case significantly less than the drift we found in question d. This is because the configuration is closer to the stable configuration (both arms segments down).

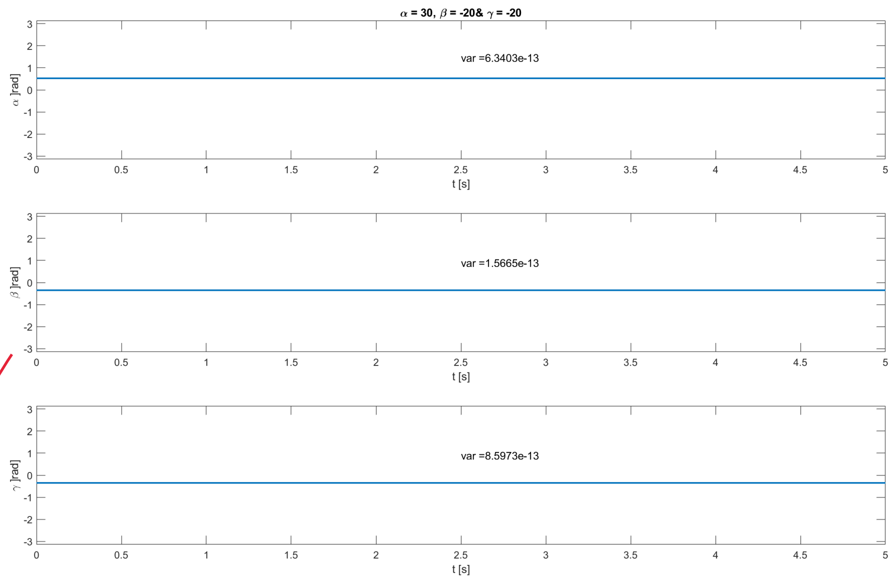

Figure 7: Simulation result when we use 4 decimals while copying

Question f

To incorporate the moments of inertia at the centres of mass we to use Eulers equation of motion for 3D bodies. The equation of motion for the upper arm becomes:

$$ {}^{\mathcal{B}}M_1 = {}^{B} I_1 \, {}^{\mathcal{B}}\dot{\omega}_1 + {}^{\mathcal{B}} \omega_1 \times ({}^{B}I_1 \, {}^{\mathcal{B}}\omega_1) \tag{34} $$

and for the lower arm:

$$ {}^{\mathcal{B}}M_2 = {}^{\mathcal{B}} I_2 \, {}^{B}\dot{\omega}_2 + {}^{\mathcal{B}} \omega_2 \times ({}^{\mathcal{B}}I_2 \, {}^{B}\omega_2) \tag{35} $$

Since ω_1 and ω_2 are expressed in the inertial frame and the newton euler equation needs all components to be expressed in the same frame we first need to translate the angular velocities in the inertial frame ${}^{I}\omega$ to those in the two body fixed frames ${}^{B}\omega$. This is done ass follows:

$$ {}^{\mathcal{B}}\omega_1 = \begin{bmatrix} 0 \\ \dot{\beta} \\ 0 \end{bmatrix} + R_{\beta}^{-1} \begin{bmatrix} \dot{\alpha} \\ 0 \\ 0 \end{bmatrix} \tag{36} $$

and

$$
{}^{\mathcal{F}}\omega_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{\gamma} \end{bmatrix} + R_\beta^{-1}\, {}^{\mathcal{B}}\omega_1 \tag{37}
$$

in these equations $\mathcal{B}$ stands fort the upper arm fixed frame and $\mathcal{F}$ for the lower arm fixed frame. The moment of inertia's given in the assignment are:

$$
I_1 = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.004 \end{bmatrix} \tag{38}
$$

$$
I_2 = \begin{bmatrix} 0.04 & 0 & 0 \\ 0 & 0.002 & 0 \\ 0 & 0 & 0.04 \end{bmatrix} \tag{39}
$$

Now the omegas are expressed in the body fixed frame we can append them to the system of equations of equation 22. The MATLAB script implementing this can be found in 5. If we take the initial states and torques of question c we get the following results.

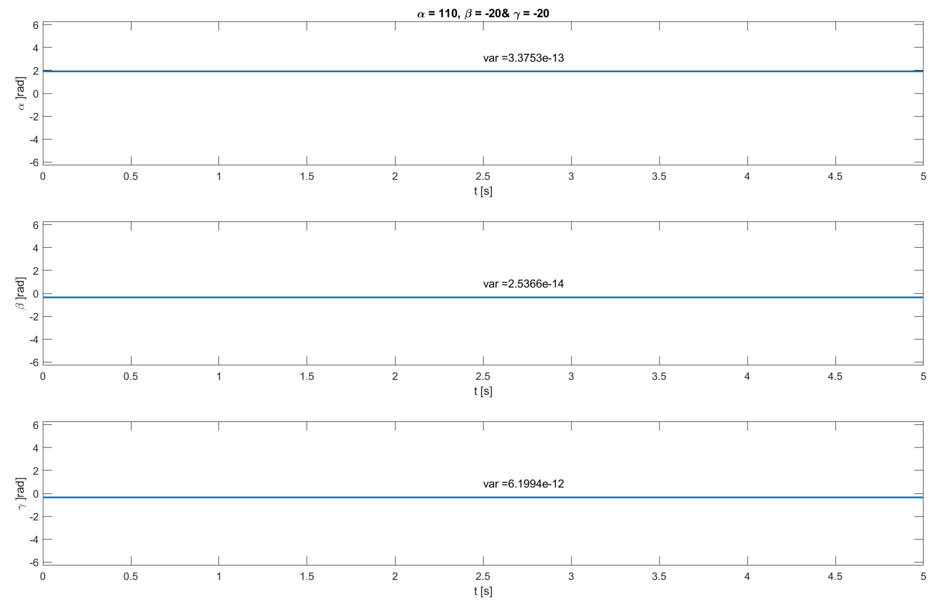

Figure 8: The result of the simulation when the moments of inertia are Incorporated

We see that this is the same result as in C and thus our method of calculating the EOM with the moments of inertia Incorporated is probbably right. I further verified my results by examining both the plots and animation (see the animate function) for different initial conditions and torques.

Question g

The $\bar{M}$ matrix of question c was as follows:

$$\bar{M} = \begin{bmatrix} 0.4920 & -0.0710 & 0.1706 \\ -0.0710 & 0.4372 & 0 \\ 0.1706 & 0 & 0.1200 \end{bmatrix} \tag{40}$$

and the new $\bar{M}$ matrix from question f is:

$$\bar{M} = \begin{bmatrix} 0.5540 & -0.0752 & 0.2082 \\ -0.0752 & 0.4686 & 0 \\ 0.2082 & 0 & 0.1600 \end{bmatrix} \tag{41}$$

From these results we can see that the $\bar{M}$ of with the moments of inertia in there is slightly bigger. Since moment of inertia expresses a body's tendency to resist angular acceleration this would suggest that when we incorporate the moments of inertia we need slightly higher moment and forces to get to the same acceleration. This can be checked by calculating the forces and moments during the movement.

Appendix A

The main MATLAB script part1

```
%% MBD_B: Assignment 9 - Euler angles and the human arm
%  Rick Staa (4511328)
clear all; close all; clc;
fprintf('--- A9 ---\n');

%% Set up needed symbolic parameters
% Create needed symbolic variables
syms alpha beta gamma alpha_d beta_d gamma_d alpha_dd beta_dd gamma_dd

% Put in parms struct for easy function handling
parms.syms.alpha             = alpha;
parms.syms.beta              = beta;
parms.syms.gamma             = gamma;
parms.syms.alpha_d           = alpha_d;
parms.syms.beta_d            = beta_d;
parms.syms.gamma_d           = gamma_d;
parms.syms.alpha_dd          = alpha_dd;
parms.syms.beta_dd           = beta_dd;
parms.syms.gamma_dd          = gamma_dd;

%% Intergration parameters
time                        = 5;                                % Intergration time
parms.h                     = 1e-3;                             % Intergration step size

%% Model Parameters
% Lengths and distances
parms.L1                     = 0.3;                             % Length upper arm [m]
parms.L2                     = 0.4;                             % Length lower arm [m]
parms.m1                     = 3;                               % Mass upper arm [kg]
parms.m2                     = 3;                               % Mass lower arm [kg]

```

```matlab
%% World parameters
% Gravity
parms.g                     = 9.81;                      % [parms.m/s^2]

%% Set Initial states
alpha_0                     = deg2rad(30);
beta_0                      = deg2rad(-20);
gamma_0                     = deg2rad(-20);
alpha_d_0                   = 0;
beta_d_0                    = 0;
gamma_d_0                   = 0;
q0                          = [alpha_0;beta_0;gamma_0;alpha_d_0;beta_d_0;gamma_d_0];     % Put in initial state vector

%% Calculate equilibrium torques
torque_calc(parms);
q0_tmp                      = num2cell(q0',1);
parms.Q                     = subs_torque(q0_tmp{:});
% parms.Q                     = [-11.2669 4.0831 -5.5077].';

%% Derive equation of motion
EOM_calc(parms);                                         % Calculate symbolic equations of motion and put in parms struct

%% Calculate movement with ode
opt                         = odeset('AbsTol',1e-6,'RelTol',1e-6,'Stats','on');
[t,q]                       = ode113(@(t,q) ODE_func(t,q), [0 time], q0',opt);

%% Animate movement
% animator(t,q,parms);

%% Plots

% Create plot label
title_str = strcat("\alpha = ", num2str(rad2deg(q0(1))),", \beta = ", num2str(rad2deg(q0(2)),"& \gamma = ", num2str(rad2deg(q0(3))));
```

```

% Plot angles vs time
figure;
subplot(3,1,1);
f1 = plot(t,q(:,1),'Linewidth',1.5);
ylim([-pi pi])
x_lim = xlim;
x_point = (x_lim(2)-x_lim(1))/2+x_lim(1);
text(x_point,mean(q(:,1))+0.3*pi,(strcat('var =',num2str(var(
    q(:,1))))));
xlabel('t [s]');
ylabel('\alpha ]rad]');
title(title_str);
subplot(3,1,2)
f2 = plot(t,q(:,2),'Linewidth',1.5);
text(x_point,mean(q(:,2))+0.4*pi,(strcat('var =',num2str(var(
    q(:,2))))));
ylim([-pi pi])
xlabel('t [s]');
ylabel('\beta ]rad]');
subplot(3,1,3)
f3 = plot(t,q(:,3),'Linewidth',1.5);
text(x_point,mean(q(:,3))+0.4*pi,(strcat('var =',num2str(var(
    q(:,3))))));
ylim([-pi pi])
xlabel('t [s]');
ylabel('\gamma ]rad]');

%% FUNCTIONS

%% ANIMATION FUNCTION
function animator(t,q,parms)
%% Animation
% Adapted from A. Schwab's animation code

% Calculate elbow and wrist
x_full                = zeros(size(q,1),12);
for ii = 1:size(q,1)
    q_now               = num2cell(q(ii,1:3),1);
    x_full(ii,:)        = subs_x_full(q_now{:})';
end

% Create figure
```

```
figure
set(gca,'fontsize',16)
title('Animation arm')
shoulder            = plot3(0,0,0,'*g');
hold on
upper_arm           = plot3([0 x_full(1,7)],[0 x_full(1,8)],[0
   x_full(1,9)],'-b');
elbow               = plot3(x_full(1,7),x_full(1,8),x_full(1,9)
   ,'*c');
lower_arm           = plot3([x_full(1,7) x_full(1,10)],[x_full
   (1,8) x_full(1,11)],[x_full(1,9) x_full(1,12)],'-r');
wrist               = plot3(x_full(1,10),x_full(1,11),x_full
   (1,12),'*m');
set(upper_arm,'LineWidth',5);
set(upper_arm,'Color','b')
set(lower_arm,'LineWidth',5);
set(lower_arm,'Color','r')
set(shoulder,'Linewidth',10);
set(elbow,'Linewidth',10);
set(wrist,'Linewidth',10);
axis([-(parms.L1+parms.L2) (parms.L1+parms.L2) -(parms.L1+
   parms.L2) (parms.L1+parms.L2) -(parms.L1+parms.L2) (parms.
   L1+parms.L2)]);
legend('shoulder','upper arm','elbow','lower arm','wrist');
nstep = length(t);
nskip = 10;
for istep = 2:nskip:nstep
    set(upper_arm,'XData',[0 x_full(istep,7)])
    set(upper_arm,'YData',[0 x_full(istep,8)])
    set(upper_arm,'ZData',[0 x_full(istep,9)])
    set(elbow,'XData',x_full(istep,7))
    set(elbow,'YData',x_full(istep,8))
    set(elbow,'ZData',x_full(istep,9))
    set(lower_arm,'XData',[x_full(istep,7) x_full(istep,10)])
    set(lower_arm,'YData',[x_full(istep,8) x_full(istep,11)])
    set(lower_arm,'ZData',[x_full(istep,9) x_full(istep,12)])
    set(wrist,'XData',x_full(istep,10))
    set(wrist,'YData',x_full(istep,11))
    set(wrist,'ZData',x_full(istep,12))
    axis([-(parms.L1+parms.L2) (parms.L1+parms.L2) -(parms.L1
       +parms.L2) (parms.L1+parms.L2) -(parms.L1+parms.L2) (
       parms.L1+parms.L2)]);
    drawnow
```

```
    pause(1e-10)
end
end

%% ODE function handle
function [qdd] = ODE_func(t,q)
q_now = num2cell(q',1);
qdd   = subs_qdd(q_now{1:end});
qdd   = [q(4);q(5);q(6);qdd];
end

%% Calculate COM velocities
function [qdd,xdd] = state_calc(t,q)

% Create matrices
qdd                 = zeros(size(q,1),6);
xdd                 = zeros(size(q,1),6);

% Calculate qdd and xdd
for ii = 1:size(q,1)
    q_now_1         = num2cell(q(ii,:),1);
    qdd_tmp         = subs_qdd(q_now_1{1:end}).';
    qdd(ii,:)       =[q(ii,4:6),qdd_tmp];
    q_now_2         = num2cell([q(ii,:).';qdd_tmp.'].',1);
    xdd(ii,:)       = subs_xdd(q_now_2{1:end}).';
end
end

%% Calculate COM torques
function torque_calc(parms)

% Unpack symbolic variables from varargin
alpha           = parms.syms.alpha;
beta            = parms.syms.beta;
gamma           = parms.syms.gamma;
alpha_d         = parms.syms.alpha_d;
beta_d          = parms.syms.beta_d;
gamma_d         = parms.syms.gamma_d;

% Create generalized coordinate vectors
q               = [alpha;beta;gamma];
qd              = [alpha_d;beta_d;gamma_d];

```

```
% Create rotation matrices
R_alpha          = rot_x(alpha);
R_beta           = rot_y(beta);
R_gamma          = rot_x(gamma);

% express COM positions in inertial frame and put them in a vector
r1_I             = R_alpha*R_beta*[0;0;-parms.L1/3];
r2_I             = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*R_beta*R_gamma*[0;0.5*parms.L2;0];

% Create mass matrix
parms.M          = diag([parms.m1,parms.m1,parms.m1,parms.m2,parms.m2,parms.m2]);

% Put in one state vector
x                = [r1_I;r2_I];

% Compute the jacobian of state and constraints
Jx_q             = simplify(jacobian(x,q.'));

%% Calculate convective component
Jx_dq            = jacobian(Jx_q*qd,q);

% Add forces F=[M2,F3_x,F3_y,M3,F4_x,F4_y,M4,F5_x,F5_y,M5,F6_x];
F_B              = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.g];

% Calculate result expressed in generalized coordinates
Q_0 = (Jx_q.'*(F_B-parms.M*Jx_dq*qd));

% Get x_full to animate the body
matlabFunction(simplify(Q_0),'vars',[alpha,beta,gamma,alpha_d,beta_d,gamma_d],'file','subs_torque');                   % Create function handle of EOM in terms of generalised coordinates

end

%% Calculate (symbolic) Equations of Motion four our setup
function EOM_calc(parms)

```

```
% Unpack symbolic variables from varargin
alpha               = parms.syms.alpha;
beta                = parms.syms.beta;
gamma               = parms.syms.gamma;
alpha_d             = parms.syms.alpha_d;
beta_d              = parms.syms.beta_d;
gamma_d             = parms.syms.gamma_d;
alpha_dd            = parms.syms.alpha_dd;
beta_dd             = parms.syms.beta_dd;
gamma_dd            = parms.syms.gamma_dd;

% Create rotation matrices
R_alpha             = rot_x(alpha);
R_beta              = rot_y(beta);
R_gamma             = rot_x(gamma);

% Create generalized coordinate vectors
q                   = [alpha;beta;gamma];
qd                  = [alpha_d;beta_d;gamma_d];
qdd                 = [alpha_dd;beta_dd;gamma_dd];

% express COM positions in inertial frame and put them in a vector
r1_I                = R_alpha*R_beta*[0;0;-parms.L1/3];
r2_I                = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*R_beta*R_gamma*[0;0.5*parms.L2;0];

% Create mass matrix
parms.M             = diag([parms.m1,parms.m1,parms.m1,parms.m2,parms.m2,parms.m2]);

% Put in one state vector
x                   = [r1_I;r2_I];

% Create full state for animation
x_elbow             = R_alpha*R_beta*[0;0;parms.L1];
x_wrist             = R_alpha*R_beta*[0;0;parms.L1]+R_alpha*R_beta*R_gamma*[0;parms.L2;0];
x_full              = [x;x_elbow;x_wrist];

% Compute the jacobian of state and constraints
Jx_q                = simplify(jacobian(x,q.'));

```

```matlab
%% Calculate convective component
Jx_dq           = jacobian(Jx_q*qd,q);

% Solve with virtual power
M_bar           = Jx_q.'*parms.M*Jx_q;

% Add forces F=[M2,F3_x,F3_y,M3,F4_x,F4_y,M4,F5_x,F5_y,M5,F6_x];
F_B             = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.g];
F               = Jx_q.'*(F_B-parms.M*Jx_dq*qd)-parms.Q;                   % Forces expressed in the inertial frame in generalised coordinates

% Calculate result expressed in generalized coordinates
qdp             = M_bar\F;

% Get result back in COM coordinates
xd              = Jx_q*qd;
xdd             = simplify(jacobian(xd,qd.'))*qdd + simplify(jacobian(xd,q.'))*qdd;

%% Convert to function handles
matlabFunction(simplify(qdp),'vars',[alpha,beta,gamma,alpha_d,beta_d,gamma_d],'File','subs_qdd');                                % Create function handle of EOM in terms of generalised coordinates

% Get back to COM coordinates
matlabFunction(simplify(x),'File','subs_x');                                    % Create function handle of EOM in terms of generalised coordinates

% Get xdp COM coordinates
matlabFunction(simplify(xdd),'vars',[alpha,beta,gamma,alpha_d,beta_d,gamma_d,alpha_dd,beta_dd,gamma_dd],'File','subs_xdd');                                    % Create function handle of EOM in terms of generalised coordinates

% Get x_full to animate the body
matlabFunction(simplify(x_full),'file','subs_x_full');                          % Create function handle of EOM in terms of generalised coordinates
```

```
% Get M_bar as function
matlabFunction(simplify(M_bar),'file','subs_M_bar');
                                % Create function handle of EOM
    in terms of generalised coordinates

end

%% Rotation matrices
% x rotation matrix
function R_alpha = rot_x(angle)
R_alpha = [   1           0           0          ;...
    0           cos(angle)    -sin(angle);...
    0           sin(angle)    cos(angle)];
end

% y rotation matrix
function R_beta = rot_y(angle)
R_beta = [ cos(angle)            0           sin(angle);...
    0                    1           0;...
    -sin(angle)           0           cos(angle)];
end

% z rotation matrix
function R_gamma = rot_z(angle)
R_gamma = [cos(angle)            -sin(angle)           0;...
    sin(angle)            cos(angle)            0;...
    0                    0                   1];
end
```

Appendix B

The main MATLAB script part2

```
%% MBD_B: Assignment 9 - Euler angles and the human arm
%  Rick Staa (4511328)
clear all; close all; clc;
fprintf('--- A9 ---\n');

%% Set up needed symbolic parameters
% Create needed symbolic variables
```

```
syms alpha beta gamma alpha_d beta_d gamma_d alpha_dd beta_dd gamma_dd

% Put in parms struct for easy function handling
parms.syms.alpha            = alpha;
parms.syms.beta             = beta;
parms.syms.gamma            = gamma;
parms.syms.alpha_d          = alpha_d;
parms.syms.beta_d           = beta_d;
parms.syms.gamma_d          = gamma_d;
parms.syms.alpha_dd         = alpha_dd;
parms.syms.beta_dd          = beta_dd;
parms.syms.gamma_dd         = gamma_dd;

%% Intergration parameters
time                        = 5;                % Intergration time
parms.h                     = 1e-3;             % Intergration step size

%% Model Parameters
% Lengths and distances
parms.L1                    = 0.3;              % Length upper arm [m]
parms.L2                    = 0.4;              % Length lower arm [m]
parms.m1                    = 3;                % Mass upper arm [kg]
parms.m2                    = 3;                % Mass lower arm [kg]
parms.I1                    = diag([0.025 0.025 0.004]);    % Inertial matrix body 1 (Expressed in body frame)
parms.I2                    = diag([0.040 0.002 0.040]);    % Inertial matrix body 2 (Expressed in body frame)

%% World parameters
% Gravity
parms.g                     = 9.81;             % [parms.m/s^2]

```

```
%% Set Initial states
alpha_0                     = deg2rad(110);
beta_0                      = deg2rad(-20);
gamma_0                     = deg2rad(-20);
alpha_d_0                   = 0;
beta_d_0                    = 0;
gamma_d_0                   = 0;
q0                          = [alpha_0;beta_0;gamma_0;alpha_d_0;beta_d_0;gamma_d_0];    % Put in initial state vector

%% Calculate equilibrium torques
torque_calc(parms);
q0_tmp                      = num2cell(q0',1);
parms.Q                     = subs_torque(q0_tmp{:});

%% Derive equation of motion
EOM_calc(parms);                                        % Calculate symbolic equations of motion and put in parms struct

%% Calculate movement with ode
opt                         = odeset('AbsTol',1e-6,'RelTol',1e-6,'Stats','on');
[t,q]                       = ode113(@(t,q) ODE_func(t,q), [0 time], q0',opt);

%% Animate movement
% animator(t,q,parms);

%% Plots

% Create plot label
title_str = strcat("\alpha = ", num2str(rad2deg(q0(1))),", \beta = ", num2str(rad2deg(q0(2))),"& \gamma = ", num2str(rad2deg(q0(3))));

% Plot angles vs time
figure;
subplot(3,1,1);
f1 = plot(t,q(:,1),'Linewidth',1.5);
ylim([-2*pi 2*pi])
```

```
x_lim = xlim;
x_point = (x_lim(2)-x_lim(1))/2+x_lim(1);
text(x_point,mean(q(:,1))+0.4*pi,(strcat('var =',num2str(var(
    q(:,1))))));
xlabel('t [s]');
ylabel('\alpha ]rad]');
title(title_str);
subplot(3,1,2)
f2 = plot(t,q(:,2),'Linewidth',1.5);
text(x_point,mean(q(:,2))+0.4*pi,(strcat('var =',num2str(var(
    q(:,2))))));
ylim([-2*pi 2*pi])
xlabel('t [s]');
ylabel('\beta ]rad]');
subplot(3,1,3)
f3 = plot(t,q(:,3),'Linewidth',1.5);
text(x_point,mean(q(:,3))+0.4*pi,(strcat('var =',num2str(var(
    q(:,3))))));
ylim([-2*pi 2*pi])
xlabel('t [s]');
ylabel('\gamma ]rad]');

%% FUNCTIONS

%% ANIMATION FUNCTION
function animator(t,q,parms)
%% Animation
% Adapted from A. Schwab's animation code

% Calculate elbow and wrist
x_full                  = zeros(size(q,1),12);
for ii = 1:size(q,1)
    q_now               = num2cell(q(ii,1:3),1);
    x_full(ii,:)        = subs_x_full(q_now{:})';
end

% Create figure
figure
set(gca,'fontsize',16)
title('Animation arm')
shoulder            = plot3(0,0,0,'*g');
hold on
```

```
upper_arm           = plot3([0 x_full(1,7)],[0 x_full(1,8)],[0 x_full(1,9)],'-b');
elbow               = plot3(x_full(1,7),x_full(1,8),x_full(1,9),'*c');
lower_arm           = plot3([x_full(1,7) x_full(1,10)],[x_full(1,8) x_full(1,11)],[x_full(1,9) x_full(1,12)],'-r');
wrist               = plot3(x_full(1,10),x_full(1,11),x_full(1,12),'*m');
set(upper_arm,'LineWidth',5);
set(upper_arm,'Color','b')
set(lower_arm,'LineWidth',5);
set(lower_arm,'Color','r')
set(shoulder,'Linewidth',10);
set(elbow,'Linewidth',10);
set(wrist,'Linewidth',10);
axis([-(parms.L1+parms.L2) (parms.L1+parms.L2) -(parms.L1+parms.L2) (parms.L1+parms.L2) -(parms.L1+parms.L2) (parms.L1+parms.L2)]);
legend('shoulder','upper arm','elbow','lower arm','wrist');
nstep = length(t);
nskip = 10;
for istep = 2:nskip:nstep
    set(upper_arm,'XData',[0 x_full(istep,7)])
    set(upper_arm,'YData',[0 x_full(istep,8)])
    set(upper_arm,'ZData',[0 x_full(istep,9)])
    set(elbow,'XData',x_full(istep,7))
    set(elbow,'YData',x_full(istep,8))
    set(elbow,'ZData',x_full(istep,9))
    set(lower_arm,'XData',[x_full(istep,7) x_full(istep,10)])
    set(lower_arm,'YData',[x_full(istep,8) x_full(istep,11)])
    set(lower_arm,'ZData',[x_full(istep,9) x_full(istep,12)])
    set(wrist,'XData',x_full(istep,10))
    set(wrist,'YData',x_full(istep,11))
    set(wrist,'ZData',x_full(istep,12))
    axis([-(parms.L1+parms.L2) (parms.L1+parms.L2) -(parms.L1+parms.L2) (parms.L1+parms.L2) -(parms.L1+parms.L2) (parms.L1+parms.L2)]);
    drawnow
    pause(1e-10)
end
end

%% ODE function handle
```

```
function [qdp] = ODE_func(t,q)
    q_now = num2cell(q',1);
    qdp   = subs_qdp(q_now{1:end});
    qdp   = [q(4);q(5);q(6);qdp];
end

%% Calculate COM velocities
function [qdd,xdd] = state_calc(t,q)

% Create matrices
qdd                 = zeros(size(q,1),6);
xdd                 = zeros(size(q,1),6);

% Calculate qdd and xdd
for ii = 1:size(q,1)
    q_now_1         = num2cell(q(ii,:),1);
    qdd_tmp         = subs_qdd(q_now_1{1:end}).';
    qdd(ii,:)       =[q(ii,4:6),qdd_tmp];
    q_now_2         = num2cell([q(ii,:).';qdd_tmp.'].',1);
    xdd(ii,:)       = subs_xdd(q_now_2{1:end}).';
end
end

%% Calculate COM torques
function torque_calc(parms)

% Unpack symbolic variables from varargin
alpha           = parms.syms.alpha;
beta            = parms.syms.beta;
gamma           = parms.syms.gamma;
alpha_d         = parms.syms.alpha_d;
beta_d          = parms.syms.beta_d;
gamma_d         = parms.syms.gamma_d;

% Create generalized coordinate vectors
q               = [alpha;beta;gamma];
qd              = [alpha_d;beta_d;gamma_d];

% Create rotation matrices
R_alpha         = rot_x(alpha);
R_beta          = rot_y(beta);
R_gamma         = rot_x(gamma);

```

```
% express COM positions in inertial frame and put them in a vector
r1_I              = R_alpha*R_beta*[0;0;-parms.L1/3];
r2_I              = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*R_beta*R_gamma*[0;0.5*parms.L2;0];

% Create mass matrix
parms.M           = diag([parms.m1,parms.m1,parms.m1,parms.m2,parms.m2,parms.m2]);

% Put in one state vector
x                 = [r1_I;r2_I];

% Compute the jacobian of state and constraints
Jx_q              = simplify(jacobian(x,q.'));

%% Calculate convective component
Jx_dq             = jacobian(Jx_q*qd,q);

% Calculate angular velocities in the body fixed frames
omega_1           = [0;beta_d;0]+R_beta\[alpha_d;0;0];
T_omega_1_qd      = jacobian(omega_1,qd);
omega_2           = [gamma_d;0;0]+R_gamma\omega_1;
T_omega_2_qd      = jacobian(omega_2,qd);
omega             = [omega_1;omega_2];

% Create new Transformation matrix
T_new             = [Jx_q;T_omega_1_qd;T_omega_2_qd];

% Create new Mass matrix
M_new              = blkdiag(parms.M,parms.I1,parms.I2);

% Calculate convective terms
I_c               = blkdiag(parms.I1,parms.I2);
Ic_omega          = (I_c*omega);
omega_cross       = [cross(omega(1:3,1),Ic_omega(1:3,1)) ...
                    ;cross(omega(4:end,1),Ic_omega(4:end,1))];

% Calculate qdp
F_ext             = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.g;0;0;0;0;0;0];          % External forces on COMs
G                 = [Jx_dq*qd;omega_cross];
Q_0               = T_new.'*(F_ext-M_new*G);
```

```
% Get x_full to animate the body
matlabFunction(simplify(Q_0),'vars',[alpha,beta,gamma,alpha_d,beta_d,gamma_d],'file','subs_torque');                              % Create function handle of EOM in terms of generalised coordinates

end

%% Calculate (symbolic) Equations of Motion four our setup
function EOM_calc(parms)

% Unpack symbolic variables from varargin
alpha               = parms.syms.alpha;
beta                = parms.syms.beta;
gamma               = parms.syms.gamma;
alpha_d             = parms.syms.alpha_d;
beta_d              = parms.syms.beta_d;
gamma_d             = parms.syms.gamma_d;
alpha_dd            = parms.syms.alpha_dd;
beta_dd             = parms.syms.beta_dd;
gamma_dd            = parms.syms.gamma_dd;

% Create rotation matrices
R_alpha             = rot_x(alpha);
R_beta              = rot_y(beta);
R_gamma             = rot_x(gamma);

% Create generalized coordinate vectors
q                   = [alpha;beta;gamma];
qd                  = [alpha_d;beta_d;gamma_d];
qdd                 = [alpha_dd;beta_dd;gamma_dd];

% express COM positions in inertial frame and put them in a vector
r1_I                = R_alpha*R_beta*[0;0;-parms.L1/3];
r2_I                = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*R_beta*R_gamma*[0;0.5*parms.L2;0];

% Create mass matrix
parms.M             = diag([parms.m1,parms.m1,parms.m1,parms.m2,parms.m2,parms.m2]);

```

```
% - Uncomment when you want full symbolic expression -
syms m1 m2 Ix1 Iy1 Iz1 Ix2 Iy2 Iz2
parms.M         = diag([m1,m1,m1,m2,m2,m2]);
parms.I1        = [Ix1 0 0;0 Iy1 0;0 0 Iz1];
parms.I2        = [Ix2 0 0;0 Iy2 0;0 0 Iz2];
% - Uncomment when you want full symbolic expression -

% Put in one state vector
x                = [r1_I;r2_I];

% Create full state for animation
x_elbow          = R_alpha*R_beta*[0;0;parms.L1];
x_wrist          = R_alpha*R_beta*[0;0;parms.L1]+R_alpha*
   R_beta*R_gamma*[0;parms.L2;0];
x_full           = [x;x_elbow;x_wrist];

% Compute the jacobian of state and constraints
Jx_q             = simplify(jacobian(x,q.'));

%% Calculate convective component
Jx_dq            = jacobian(Jx_q*qd,q);

% Calculate angular velocities in the body fixed frames
omega_1          = [0;beta_d;0]+R_beta\[alpha_d;0;0];
T_omega_1_qd     = jacobian(omega_1,qd);
omega_2          = [gamma_d;0;0]+R_gamma\omega_1;
T_omega_2_qd     = jacobian(omega_2,qd);
omega            = [omega_1;omega_2];

% Create new Transformation matrix
T_new            = [Jx_q;T_omega_1_qd;T_omega_2_qd];

% Create new Mass matrix
M_new             = blkdiag(parms.M,parms.I1,parms.I2);

% Create new TMT matrix
M_bar            = simplify(T_new.'*M_new*T_new);

% Calculate convective terms
I_c              = blkdiag(parms.I1,parms.I2);
Ic_omega         = (I_c*omega);
omega_cross      = [cross(omega(1:3,1),Ic_omega(1:3,1)) ...
                    ;cross(omega(4:end,1),Ic_omega(4:end,1))];
```

```
% Calculate qdp
F_ext               = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.g;0;0;0;0;0;0];           % External forces on COMs
G                   = [Jx_dq*qd;omega_cross];
Q_bar               = simplify(T_new.'*(F_ext-M_new*G) - parms.Q);

% Calculate reesult expressed in generalized coordinates
qdp                 = M_bar\Q_bar;

% Get result back in COM coordinates
xd                  = Jx_q*qd;
xdd                 = simplify(jacobian(xd,qd.'))*qdd + simplify(jacobian(xd,q.'))*qdd;

%% Convert to function handles
matlabFunction(simplify(qdp),'vars',[alpha,beta,gamma,alpha_d,beta_d,gamma_d],'File','subs_qdp');                                 % Create function handle of EOM in terms of generalised coordinates

% Get back to COM coordinates
matlabFunction(simplify(x),'File','subs_x');                                         % Create function handle of EOM in terms of generalised coordinates

% Get xdp COM coordinates
matlabFunction(simplify(xdd),'vars',[alpha,beta,gamma,alpha_d,beta_d,gamma_d,alpha_dd,beta_dd,gamma_dd],'File','subs_xdd');                                            % Create function handle of EOM in terms of generalised coordinates

% Get x_full to animate the body
matlabFunction(simplify(x_full),'file','subs_x_full');                               % Create function handle of EOM in terms of generalised coordinates

% Get M_bar as function
matlabFunction(simplify(M_bar),'file','subs_M_bar');                                 % Create function handle of EOM in terms of generalised coordinates

```

```
end

%% Rotation matrices
% x rotation matrix
function R_alpha = rot_x(angle)
    R_alpha = [   1            0            0            ;...
                  0            cos(angle)   -sin(angle);...
                  0            sin(angle)   cos(angle)];
end

% y rotation matrix
function R_beta = rot_y(angle)
    R_beta = [ cos(angle)              0            sin(angle);
        ...
                 0                     1            0;...
               -sin(angle)             0            cos(angle)];
end

% z rotation matrix
function R_gamma = rot_z(angle)
    R_gamma = [cos(angle)             -sin(angle)             0;
        ...
               sin(angle)              cos(angle)             0;
                  ...
                  0                       0
                     1];
end
```

References

[1] Arend L. Schwab. Reader: MultiBody Dynamics B. In *Multibody Dynamics*, chapter 3. TU Delft, Delft, The Netherlands, 2018.