Multibody Dynamics B - Assignment 10

July 4, 2019

Short problem statement:
We are concerned with a simple mechanical model of the human (left) arm, which consists of two rigid bodies connected by three hinges. The inertial frame is defined as the human is looking North (our x-axis) and thus our y-axis is west and our z-axis is up. The arm is an open loop structure which has its origin in the shoulder, which is defined as O. The hinges in the shoulder are a hinge with an angle α around the y-axis and a hinge with an angle β around the x-axis and in the elbow there is a hinge with an angle γ around the y-axis. The upper arm, body 1, is of length $d = 0.3$ m and the forearm including the hand,body 2, is of length $e = 0.375$ m. The starting position of the hand is such that the upper arm is pointing in the negative z-direction and the forearm is pointing in the positive x direction, so the elbow is located at $(0, 0, -d)$ and the hand is located at $(e, 0, -d)$ and the angles of the hinges in these positions are all zero. The mass of the upper arm is $m_1 = 1.9$ kg and the mass of the forarm is $m_2 = 1.5$ kg. The CoM of the upper hand is located at distance $d/2$ from the shoulder and the CoM of the forarm is located at a distance $4/10 * e$ from the elbow. Furthermore, the mass moments of inertia at the CoM for the two bodies are as follow; for the upper arm: ${}^{U}(I_{xx}, I_{yy}, I_{zz}) = (0.015, 0.014, 0.002)$ and for the forarm: ${}^{F}(I_{xx}, I_{yy}, I_{zz}) = (0.001, 0.019, 0.019)$ kgm^2. These are the principal values about the principal axes, so there are no off-diagonal terms. Lastly we have a gravity field of strength $g = 9.81$ that works in the negative z-direction. We will derive the equation of motion for this system expressed in terms of th independant genearilzed coordinat $\mathbf{q} = (\alpha, \beta, \gamma)$ and their time derivatives, using the TMT method.

a)

We begin with making a sketch of the system, where the hinges are depicted as cans-in-series, this sketch can be seen in figure 1 below.

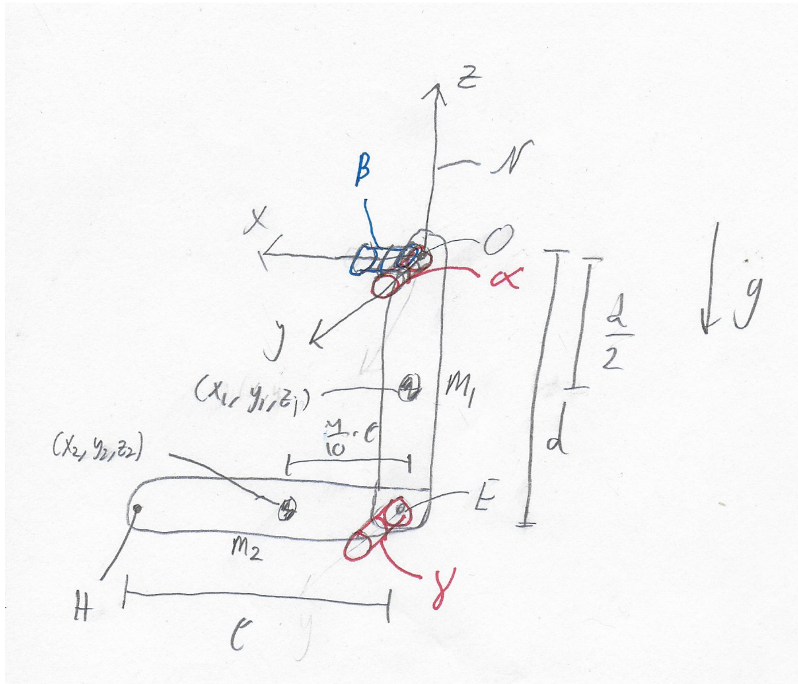

Figure 1: A sketch of the system.

b)

Now we write down the expressions for the CoM coordinates of the two bodies, $x_i = (x_1, y_1, z_1, x_2, y_2, z_2)$ expressed in the terms of generalized coordinates $\mathbf{q}$ and the systems parameters.
Now we use the rotation matrices to our benefit where we have

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

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

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

This allows us to find the CoM expression as a function of the generalized coordinates where we have

$$x_1 = R_\alpha R_\beta \begin{bmatrix} 0 \\ 0 \\ -d/2 \end{bmatrix}$$

and

$$x_2 = R_\alpha R_\beta \begin{bmatrix} 0 \\ 0 \\ -d \end{bmatrix} + R_\alpha R_\beta R_\gamma \begin{bmatrix} \frac{4}{10}e \\ 0 \\ 0 \end{bmatrix}$$

This gives us

$$x_i = \begin{bmatrix} -\frac{d}{2}cos(\beta)sin(\alpha) \\ \frac{d}{2}sin(\beta) \\ -\frac{d}{2}cos(\alpha)cos(\beta) \\ \frac{2}{5}e(cos(\alpha)cos(\gamma) - cos(\beta)sin(\alpha)sin(\gamma)) - dcos(\beta)sin(\alpha) \\ dsin(\beta) + \frac{2}{5}e(sin(\beta)sin(\gamma)) \\ -\frac{2}{5}e(cos(\gamma)sin(\alpha) + cos(\alpha)cos(\beta)sin(\gamma)) - dcos(\alpha)cos(\beta) \end{bmatrix} \tag{1}$$

and thus we can write our transformation matrix as

$$T_i = \begin{bmatrix} -\frac{d}{2}cos(\beta)sin(\alpha) \\ \frac{d}{2}sin(\beta) \\ -\frac{d}{2}cos(\alpha)cos(\beta) \\ \alpha \\ \beta \\ \frac{2}{5}e(cos(\alpha)cos(\gamma) - cos(\beta)sin(\alpha)sin(\gamma)) - dcos(\beta)sin(\alpha) \\ dsin(\beta) + \frac{2}{5}e(sin(\beta)sin(\gamma)) \\ -\frac{2}{5}e(cos(\gamma)sin(\alpha) + cos(\alpha)cos(\beta)sin(\gamma)) - dcos(\alpha)cos(\beta) \\ \gamma \end{bmatrix}$$

c)

Now we determine, for the initial configuration, the three hinge torques needed to keep the system in static equilibrium.
When we have a static equilibrium that means zero linear velocity and zero angular velocity. Furthermore, we can write the hinge torques as generalized forces in the equation of motion, which can be written, with zero linear- and angular velocity, as just the forces needed to counteract the applied forces. With zero angular velocity the equation of motion can be written as

$$T_{i,j}^T M_i T_{i,j} \ddot{q} = T_{i,j}^T \left(\sum F_i - M_i T_{k,lm} \dot{q}_l \dot{q}_m \right) - T_{hinge}$$

Now there is no acceleration, as there is no linear velocity and thus we have

$$T_{i,j}^T \left(\sum F_i - M_i T_{k,lm} \dot{q}_l \dot{q}_m \right) - T_{hinge} = 0$$

which gives

$$T_{hinge} = T_{i,j}^T \left(\sum F_i - M_i T_{k,lm} \dot{q}_l \dot{q}_m \right)$$

This gives us the following torques for the initial configuration

$$T_\alpha = -2.2072$$
$$T_\beta = 0$$
$$T_\gamma = -2.2072$$

d)

Now we write down the expression for the angular velocities of the two bodies expressed in the individual body-fixed frames as a function of the generalized coordinates, their time derivatives, and the systems parameters.
Now we can express the angular velocities in the body fixed frame using the rotation matrices, using equation 19.62 in the book. This gives us

$$^{B}\omega_1 = [\dot\beta 00]^T + R_\beta[0\dot\alpha 0]^T$$
$$^{B}\omega_2 = [0\dot\gamma 0]^T + R_\gamma[\dot\beta 00]^T + R_\gamma R_\beta[0\dot\alpha 0]^T$$

Then writing them in a linear-in-speed form, i.e. $\omega_i = B_{ij}(q_k)\dot q_j$ we have, where we get the matrix B by taking the jacobian of the angular velocities with regard to $\dot q = [\dot\alpha\ \dot\beta\ \dot\gamma]^T$;

$$^{B}\omega_1 = \begin{bmatrix} 0 & 1 & 0 \\ cos(\beta) & 0 & 0 \\ -sin(\beta) & 0 & 0 \end{bmatrix}\begin{bmatrix} \dot\alpha \\ \dot\beta \\ \dot\gamma \end{bmatrix}$$
$$^{B}\omega_2 = \begin{bmatrix} sin(\beta)sin(\gamma) & cos(\gamma) & 0 \\ cos(\beta) & 0 & 1 \\ -cos(\gamma)sin(\beta) & sin(\gamma) & 0 \end{bmatrix}$$

e)

We now derive the equation of motion for the arm in terms of the generalized coordinate **q**, their time derivatives and the system parameters using the TMT-method.
Using the TMT-method, where we use the virtual power expression, d'Alambert forces and the euler equations, we write the virtual power expression as

$$\delta P = \delta\dot x_i(F_i - Mij\ddot x_j) + (M_b - I_b\dot\omega_b + \omega_b \times (Ib\omega_b))\delta\omega_b$$

Now we will use the fact that we can write $x_i = T_i(q_i)$, where $\dot x_i = T_{i,j}(\dot q_j)$ and $\ddot x_k = T_{k,l}(\ddot q_l + T_{k,lm}\dot q_l\dot q_m$, where $T_{k,lm}\dot q_l\dot q_m$ are the convective terms. Furthermore we can write $\omega_b = \omega_b(q,\dot q) = B_b(q)\dot q$ and $\dot\omega_b = B_b\ddot q + B_{b,q}\dot q\dot q$ where $B_{b,q}\dot q\dot q$ are the convective terms. Since we introduced the generalized coordinates we have introduced generalized forces Q_j, which in this case are the torques needed to keep the system in static equilibrium, which we add to our power expression which gives

$$\delta P = \delta\dot x_i(F_i - Mij\ddot x_j) + (M_b' - [I_b\dot\omega_b + \omega_b \times (Ib\omega_b)])\delta\omega_b + \delta\dot q_j Q_j$$

We define again the CoM coordinates with regard to the generalized coordinates as is shown above and make use of the transformation matrix defined in **b)**. Now the TMT-method tells us that we have a solution of the kind

$$\bar{M}\ddot{q} = \bar{Q}$$

Substituting in for δP and then for all $\delta\dot{x}_b$ and $\delta\omega_b$ we get the EOM which looks like,

$$\bar{M} = \sum_b [T_{b,q}^T M_b T_{b,q} + B_b^T I_b B_b]\ddot{q}$$

$$\bar{Q} = Q + \sum_b [T_{b,q}(f - M_b h_b) + B_b^T (M_b' - Ibg_b - \omega_b \times (Ib\omega_b))]$$

where h_b are the convective terms for the Newton part, g_b are the convective terms for the Euler part, M_b' are the applied torques in the body-fixed frame, which are zero, B_b are the matrices for the linear-in-speed format for the angular velocities, I_b are the mass moment matrices for the individual bodies and $T_{b,q}$ is the jacobian of the transformation matrix with regard to **q**. Now since we have two bodies $b = 2$, and thus we take the transformation matrix relevant to each body, i.e. $T_{1,q} = [x_1, \alpha, \beta]^T$ and $T_{2,q} = [x_2, \gamma]^T$ and $M_1 = diag([m1, m1, m1, 0, 0])$ and $M_2 = diag([m2, m2, m2, 0])$. Furthermore, $I_{b,1} = diag({}^U[I_{xx}, I_{yy}, I_{zz}])$ and $I_{b,2} = diag({}^F[I_{xx}, I_{yy}, I_{zz}])$ and B_1 and B_2 are the matrices found in **d)** for the linear-in-speed form for the angular velocities.

f)

Now we coded the equation of motion into Matlab and looked at a few different configurations where we could predict the resulting accelerations of generalized coordinates.
We first looked at how the two bodies moved without implementing the torques for static equilibrium, as that felt more intuitive, to just let it be free and only applied forces were working on the bodies. **The code for the equation of motion can be seen in the appendix.** The first configuration we looked at was the initial configuration.
configuration 1:
First we look at the initial position, where all angles are zero and all velocities are zero, i.e. $q = [0\ 0\ 0\ 0\ 0\ 0]$, the EoM gave us

$$\ddot{q}_1 = \begin{bmatrix} 0 \\ 0 \\ 41.8436 \end{bmatrix}$$

This make sense as the gravitational field will pull the forarm down, resulting in a positive acceleration, with regard to y-axis pointing west, of the forarm.
configuration 2:
Now we will look at a position where the arm is straight down and the velocities are zero, i.e. $q = [0\ 0\ \pi/2\ 0\ 0\ 0]$, this should result in no acceleration, the EoM gave us

$$\ddot{q}_2 = \begin{bmatrix} 0.1641 * 10^{-14} \\ 0 \\ 0.3021 * 10^{-14} \end{bmatrix}$$

This is what we predicted, the small acceleration is just do to round of error in Matlab.
configuration 3:

Next we look at a configuration where both arms are up and the velocities are zero, i.e. $q = [\pi\ 0\ \pi/2\ 0\ 0\ 0]$, this should also result in no acceleration, the EoM gave us

$$\ddot{q}_3 = \begin{bmatrix} -0.3456 * 10^{-14} \\ 0 \\ 0.0192 * 10^{-14} \end{bmatrix}$$

This is again what we predicted and the small acceleration is just due to round off errors in Matalab.

configuration 4:

Lastly to see if the applied torques worked as expected in the EoM as generalized forces, which were found in **c**), we implemented them and looked at the initial configuration, if the torques are correctly implemented into the EoM we should have no acceleration of the arms around the hinges, the EoM gave us

$$\ddot{q}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Which is what we expected, and thus the torques are correctly implemented and we can conclude that the EoM is correct, as the results are plausible.

g)

Now we calculates the reduced generalized mass-matrix for the initial configuration, i.e. all angles zero.

We used Matlab to calculate the reduced generalized mass-matrix for the initial configuration which gave us

$$\bar{M} = \begin{bmatrix} 0.2445 & 0 & 0.0527 \\ 0 & 0.1938 & 0 \\ 0.0527 & 0 & 0.0527 \end{bmatrix}$$

g)

Now we put the system in a ball catch posture, where $(\alpha,\ \beta\ \gamma) = (-70°, 70°, -30°)$ and determine the hinge torques needed to maintina this posture in static equilibrium.

In the same manner as in **c** we find the hinge torques, which we find as

$$T_\alpha = -2.6164$$
$$T_\beta = 1.9627$$
$$T_\gamma = 0.8135$$

i)

Now we check our results by a means of a forward dynamic analysis for 5 seconds.

We use the Runge-Kutta 4th order numerical integration method to analyze the motion of the arm through the period, which follows the scheme

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + h/2, y_n + h/2 * k_1)$$
$$k_3 = ff(t_n + h/2, y_n + h/2 * k_2)$$
$$k_4 = ff(t_n + h, y_n + h * k_3)$$
$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Now the plots of the Euler angles of the arm can be seen in figure 2 below.

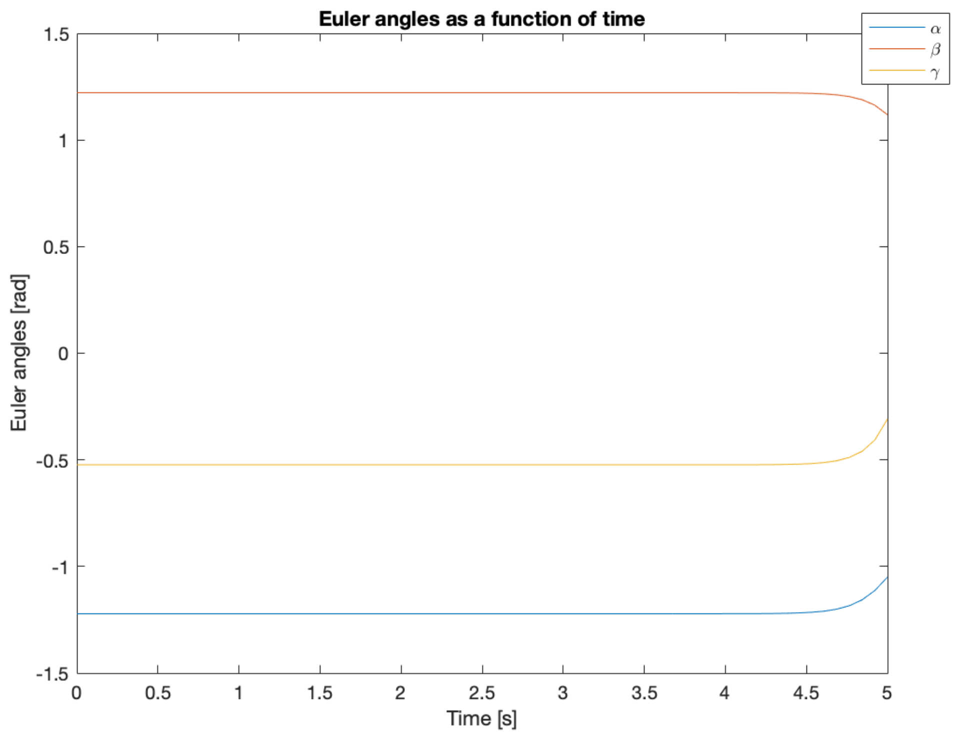

Figure 2: Euler angles as a function of time for a ball catch posture of $(\alpha,\ \beta\ \gamma) = (-70°, 70°, -30°)$, with a time step of 0.0781 seconds

Looking at the figure we notice that the arm is able to keep its posture steady for 4.5 seconds but then starts moving. This is due to the round off errors in Matlab, since the system is inherently unstable in this configuration and thus any 'perturbation' to the system in static equilibrium will result in it loosing balance. Furthermore, the errors actually cause oscillatory behaviour, so depending on the oscillation, and thus step size, we get different motion, for example if the step size is decreased the motion is reversed to what we see on figure 2, and if the step size is really really low the arm don't move, thus confirming the fact that the motion of the arms are due to errors in Matlab, which results in the system becoming unstable.

i)

Now we look at a different posture, $(\alpha,\ \beta\ \gamma) = (30°, 20°, 60°)$ and analyze the forward dynamics of the system.
The torques needed to keep the system in static equilibrium for the new position are found and put into the EoM. Now we plot the Euler angles as a function of time which can be seen in figure 3 below.

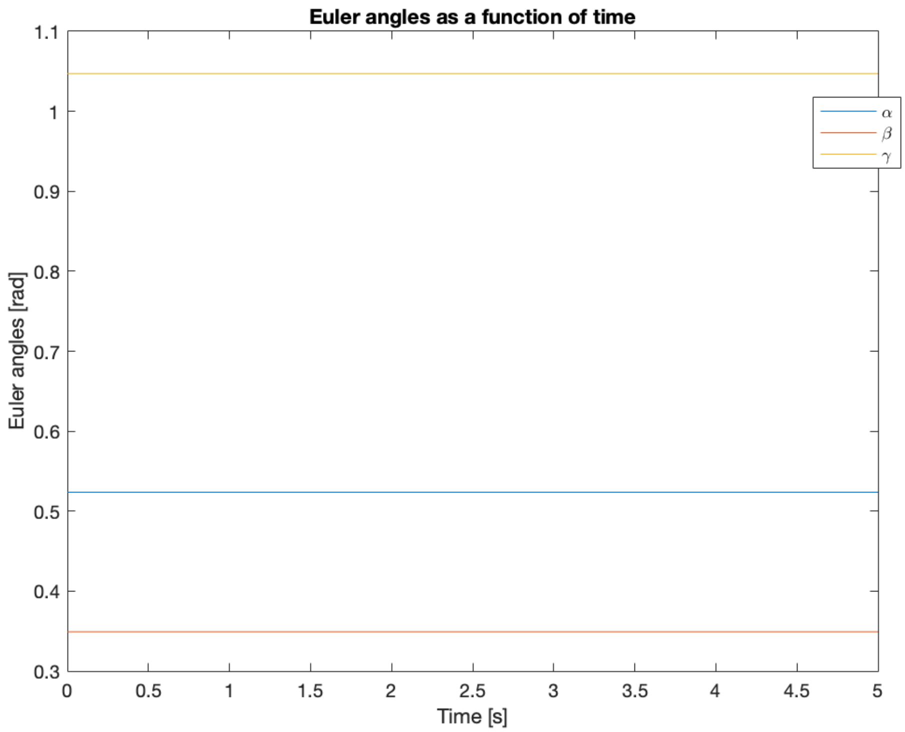

Figure 2: Euler angles as a function of time for a ball catch posture of $(\alpha,\ \beta\ \gamma) = (30°, 20°, 60°)$ with a time step of 0.0781 seconds

Here the arm does not move. This is likely due to the fact that the errors are not big enough since in this position we are closer to an equilibrium position, that is when the hand is pointing down. Thus bigger errors are needed for the system to become unstable in this configuration. If the step-size is increased we notice that the system becomes very unstable, and is never really able to maintain its posture in the static equilibrium, it oscillates around the initial angles and eventually becomes unstable, returning to an equilibrium position, which is when the hand is hanging down.

Appendix A

Matlab code

```
%% CoM positions

syms alpha beta gamma dalpha dbeta dgamma ddalpha ddbeta ddgamma
% syms e d m1 m2 g

d = 0.3;
e = 0.375;

m1 = 1.9;
m2 = 1.5;

g = 9.81;

% Rotation matrices
R_alpha = [cos(alpha) 0 sin(alpha);...
            0 1 0;...
            -sin(alpha) 0 cos(alpha)];

R_beta = [1 0 0;...
          0 cos(beta) -sin(beta);...
          0 sin(beta) cos(beta)];

R_gamma = [cos(gamma) 0 sin(gamma);...
           0 1 0;...
           -sin(gamma) 0 cos(gamma)];

% Upper arm
x_u = R_alpha*R_beta*[0;0;-d/2];

% Forarm
x_f = R_alpha*R_beta*[0;0;-d] + R_alpha*R_beta*R_gamma*[4/10*e;0;0];

% Transformation matrix
Ti = [x_u;alpha;beta;x_f;gamma];

T1 = [x_u;alpha;beta];

T2 = [x_f;gamma];


%% Angular velocities

Bomega_u = [dbeta;0;0] + R_beta.'*[0;dalpha;0];

% This gives matrix
B_u = [0 1 0;...
       cos(beta) 0 0;...
       -sin(beta) 0 0];


Bomega_f = [0;dgamma;0] + R_gamma.'*[dbeta;0;0] + ...
```

```
    R_gamma.'*R_beta.'*[0;dalpha;0];

% This gives matrix
B_f = [sin(beta)*sin(gamma) cos(gamma) 0;...
        cos(beta) 0 1;...
        -cos(gamma)*sin(beta) sin(gamma) 0];


%% EoM f)
% Mass moment of inertia
% Upper arm
Ixx_u = 0.015;
Iyy_u = 0.014;
Izz_u = 0.002;

% Forarm
Ixx_f = 0.001;
Iyy_f = 0.019;
Izz_f = 0.019;

Ib = [Iyy_u 0 0;...
      0 Ixx_u 0;...
      0 0 Iyy_f];

Ib1 = [Ixx_u 0 0;...
       0 Iyy_u 0;...
       0 0 Izz_u];

Ib2 = [Ixx_f 0 0;...
       0 Iyy_f 0;...
       0 0 Izz_f];

% mass matrix
M = diag([m1 m1 m1 0 0 m2 m2 m2 0]);

M1 = diag([m1 m1 m1 0 0]);

M2 = diag([m2 m2 m2 0]);


% Make T
q = [alpha;beta;gamma];
qd = [dalpha;dbeta;dgamma];
qdd = [ddalpha;ddbeta;ddgamma];

% Find Tij where xd = Tij*qd
Tij = jacobian(Ti,q);
Tij = simplify(Tij);
matlabFunction(Tij,'File','Tijhw10');

Tij1 = jacobian(T1,q);
Tij1 = simplify(Tij1);

Tij2 = jacobian(T2,q);
Tij2 = simplify(Tij2);
```

```
% Find velocity
Tiv = Tij*qd;
matlabFunction(Tiv,'File','Tivhw10');

% Find convective terms
h = jacobian((Tij*qd),q)*qd;
matlabFunction(h,'File','hhw10');

h1 = jacobian((Tij1*qd),q)*qd;

h2 = jacobian((Tij2*qd),q)*qd;


% Find acceleration, where xdd = Tij*qdd + h
Tacc = Tij*qdd + h;
matlabFunction(Tacc,'File','Tacchw10');

% Find reduced mass matrix, Tij'*M*Tij
Mbar = Tij1.'*M1*Tij1 + Tij2.'*M2*Tij2 + B_u.'*Ib1*B_u + B_f.'*Ib2*B_f;

% Mbar = Tij.'*M*Tij + B_f.'*Ib*B_f;
Mbar = simplify(Mbar);
matlabFunction(Mbar,'File','Mbarhw10');

% Find applied forces
Fi = M*[0;0;-g;0;0;0;0;-g;0];
Fi1 = M1*[0;0;-g;0;0];
Fi2 = M2*[0;0;-g;0];

% Torques applied to keep stabalized
% angles = [0;0;0;0;0;0];
angles = [deg2rad(30);deg2rad(20);deg2rad(60);0;0;0];   %Change angles as ...
    needed for torque
Q = torq(angles);
% Q=0;

% applied torques
Mt = 0;

% Convective terms for Euler
g1 = jacobian(B_u*qd,q)*qd;
g2 = jacobian(B_f*qd,q)*qd;


% Find reduced force matrix
Qbar = Tij1.'*(Fi1-M1*h1) + Tij2.'*(Fi2-M2*h2) + B_u.'*(Mt-Ib1*g1 - ...
    cross(Bomega_u,(Ib1*Bomega_u))) + B_f.'*(Mt-Ib2*g2 - ...
    cross(Bomega_f,(Ib2*Bomega_f))) + Q;
Qbar = simplify(Qbar);
matlabFunction(Qbar,'File','Qbarhw10');
%%
% Acceleration
acc = Mbar\Qbar;

matlabFunction(acc,'File','acchw10');

```

```matlab
%% Find torque
% Torque needed for initial position, all angles and angular velocities
% zero
ang = [0;0;0;0;0;0];

torques = torq(ang);

%% A few configurations
% Configuration 1, initial position
a1 = [0;0;0;0;0;0];
alpha = a1(1);
beta = a1(2);
gamma = a1(3);
dalpha = a1(4);
dbeta = a1(5);
dgamma = a1(6);
acc1 = acchw10(alpha,beta,dalpha,dbeta,dgamma,gamma);
ddalpha = acc1(1);
ddbeta = acc1(2);
ddgamma = acc1(3);

accCoM1 = Tacchw10(alpha,beta,dalpha,dbeta,ddalpha,ddbeta,ddgamma,dgamma,gamma);


% Configuration 2, arm down
a2 = [0;0;pi/2;0;0;0];
alpha = a2(1);
beta = a2(2);
gamma = a2(3);
dalpha = a2(4);
dbeta = a2(5);
dgamma = a2(6);
acc2 = acchw10(alpha,beta,dalpha,dbeta,dgamma,gamma);
ddalpha = acc2(1);
ddbeta = acc2(2);
ddgamma = acc2(3);

accCoM2 = Tacchw10(alpha,beta,dalpha,dbeta,ddalpha,ddbeta,ddgamma,dgamma,gamma);

% Configuration 3, arm up
a3 = [pi;0;pi/2;0;0;0];
alpha = a3(1);
beta = a3(2);
gamma = a3(3);
dalpha = a3(4);
dbeta = a3(5);
dgamma = a3(6);
acc3 = acchw10(alpha,beta,dalpha,dbeta,dgamma,gamma);
ddalpha = acc3(1);
ddbeta = acc3(2);
ddgamma = acc3(3);

accCoM3 = Tacchw10(alpha,beta,dalpha,dbeta,ddalpha,ddbeta,ddgamma,dgamma,gamma);

%% g) Reduced mass matrix for all angles zero
alpha = 0;
```

```matlab
beta = 0;
gamma = 0;

Mbar = Mbarhw10(beta,gamma);

%% h)

alpha = deg2rad(-70);
beta = deg2rad(70);
gamma = deg2rad(-30);

dalpha = 0;
dbeta = 0;
dgamma = 0;

ang = [alpha;beta;gamma;dalpha;dbeta;dgamma];

torques_ball = torq(ang);


%% i) numerical integration, Runge-Kutte 4th order

% setup
time = 5;
nn=6;
N=2^nn;
h = time./N;

t = 0;

% Initialize angles and angular velocities
alpha = deg2rad(30);
beta = deg2rad(20);
gamma = deg2rad(60);

dalpha = 0;
dbeta = 0;
dgamma = 0;

y0 = [alpha;beta;gamma;dalpha;dbeta;dgamma];
ang = [alpha;beta;gamma;dalpha;dbeta;dgamma];


for i=1:N
    k1 = state(y0);
    k2 = state(y0+h/2*k1);
    k3 = state(y0+h/2*k2);
    k4 = state(y0+h*k3);
    qn = y0 + 1/6*h*(k1+2*k2+2*k3+k4);

    y0 = qn;
    q_n(i,:) = qn;
    t = t+h;
    T(i) = t;
end

```

```matlab



%% Plots
q_plot = [ang';q_n];
T_plot = [0;T'];

figure(1)
plot(T_plot,q_plot(:,1),T_plot,q_plot(:,2),T_plot,q_plot(:,3))
xlabel('Time [s]')
ylabel('Euler angles [rad]')
title('Euler angles as a function of time')
legend('\alpha','\beta','\gamma')


figure(2)
plot(T_plot,q_plot(:,4),T_plot,q_plot(:,5),T_plot,q_plot(:,6))
xlabel('Time [s]')
ylabel('ANgular velocities [rad/s]')
title('angular velocities as a function of time')
legend('\omega_\alpha','\omega_\beta','\omega_\gamma')






%% Functions

% Finding torque needed for static equlibrium
function torques = torq(ang)
% Find torques needed
alpha = ang(1);
beta = ang(2);
gamma = ang(3);
dalpha = ang(4);
dbeta = ang(5);
dgamma = ang(6);

% setup, parameters
m1 = 1.9;
m2 = 1.5;

g = 9.81;

% Mass matrix
M = diag([m1 m1 m1 0 0 m2 m2 m2 0]);

% Applied forces
Fi = -M*[0;0;-g;0;0;0;0;-g;0];

% convective terms
h = hhw10(alpha,beta,dalpha,dbeta,dgamma,gamma);
Tij = Tijhw10(alpha,beta,gamma);

% Torques equal the right hand side
```

```
torques = Tij.'*(Fi-M*h);
end

function states = state(y)
alpha = y(1);
beta = y(2);
gamma = y(3);
dalpha = y(4);
dbeta = y(5);
dgamma = y(6);

states = acchw10(alpha,beta,dalpha,dbeta,dgamma,gamma);

states = [dalpha;dbeta;dgamma;states(1);states(2);states(3)];

end
```