

## 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

### 1 Statement of integrity

my homework is completely in accordance  
with the Academic Integrity

Figure 1: My handwritten statement of integrity

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

### 2 Acknowledgements

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

### 3 Setup overview

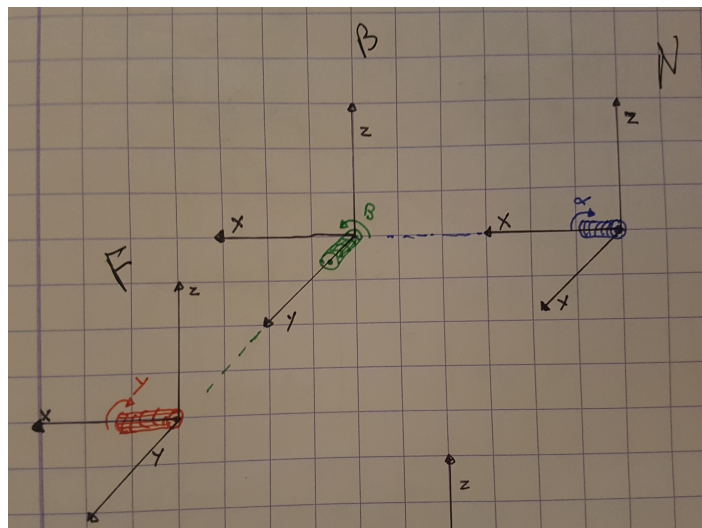
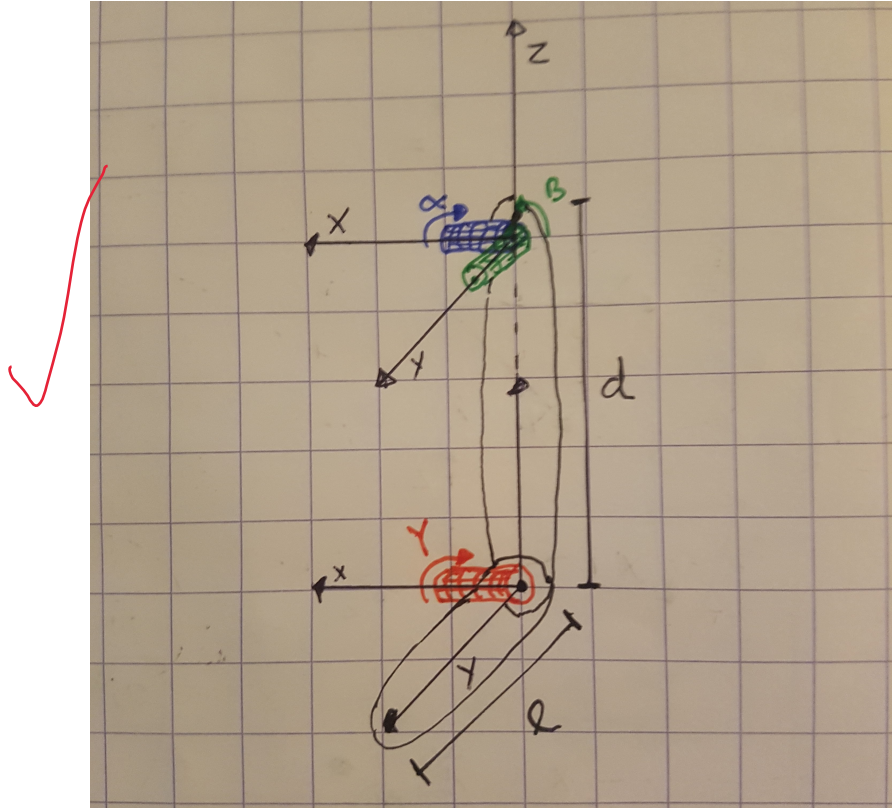


Figure 2: Cans and series representation of the arm mechanism



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

## 4 Problem Statement

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

$$L_1 = 0.3m \quad (1)$$

$$L_2 = 0.4m \quad (2)$$

$$m_1 = 3m \quad (3)$$

$$m_2 = 3m \quad (4)$$

$$g = 9.81m/s^2 \quad (5)$$

$$(6)$$

In the first questions of the assignments (question a-e) we were asked to neglect the 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:

$$q = [ \alpha \quad \beta \quad \gamma \quad \dot{\alpha} \quad \dot{\beta} \quad \dot{\gamma} ] \quad (7)$$

The full state was defined as:

$$x = [ x_1 \quad y_1 \quad z_1 \quad x_2 \quad y_2 \quad z_2 \quad \dot{x}_1 \quad \dot{y}_1 \quad \dot{z}_1 \quad \dot{x}_2 \quad \dot{y}_2 \quad \dot{z}_1 ] \quad (8)$$

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

1. Three rotation matrices ( $R_\alpha$ ,  $R_\beta$  and  $R_\gamma$ ) 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:

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

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

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

## 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:

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

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:

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

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:

$$\bar{M} \ddot{q} = \bar{f} \quad (14)$$

Where:

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

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  $R_\alpha, R_\beta$  and  $R_\gamma$ . For this problem our full state matrix  $x$  is defined as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = R_\alpha R_\beta {}^B r_{com1} \quad (16)$$

$$\begin{bmatrix} x_2 \\ y_3 \\ z_4 \end{bmatrix} = R_\alpha R_\beta R_\gamma {}^B r_{com2} \quad (17)$$

This results in the following full state:

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

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

$$M = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & m_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & m_2 \end{bmatrix} \quad (19)$$

The external force vector is defined as:

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

In our example the generalised forces  $Q$  are:

$$x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (21)$$

The T and G terms in equation 31 were derived using the symbolic toolbox of MATLAB 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:

$$\ddot{q} = \bar{M}^{-1} \bar{f} \quad (22)$$

## 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:

$$q = [ 0 \quad 0 \quad -\pi/2 \quad 0 \quad 0 \quad 0 ] \quad (23)$$

The result found with this initial state is:


$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -0.1139 \cdot 10^{-28} \\ 0 \\ -0.0414 \cdot 10^{-28} \end{bmatrix} \quad (24)$$


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:

$$q = [ \pi \quad 0 \quad -\pi/2 \quad 0 \quad 0 \quad 0 ] \quad (25)$$

For this case we get the following result:




$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.2071 \cdot 10^{-28} \end{bmatrix} \quad (26)$$

### 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 north in the positive y-direction. For this configuration we get the following initial state:

$$q = [ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 ] \quad (27)$$

For this case we get the following result:



$$\ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -49.0500 \end{bmatrix} \quad (28)$$

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:

$$q = [ 0 \quad 0.5\pi \quad 0 \quad 0 \quad 0 \quad 0 ] \quad (29)$$

For this case we get the following result:

$$\checkmark \ddot{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 98.100000000000009 \\ -98.100000000000009 \\ 0 \end{bmatrix} \quad (30)$$

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 around the  $\beta$  and  $\gamma$  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  $\ddot{\gamma}$  (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:

$$Q = T^T(F - MG) \quad (31)$$

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

$$\checkmark M_0 = \begin{bmatrix} -10.280852679463344 \\ -1.612553948844058 \\ -0.114084912734962 \end{bmatrix} \quad (32)$$

### Question d

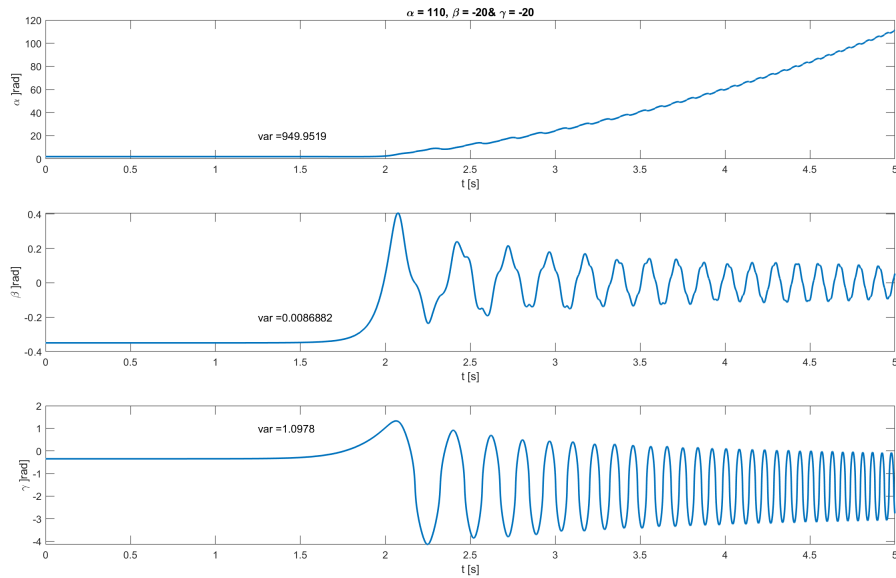
If we copy this with MATLAB accuracy set on "format short" we get the following result:

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

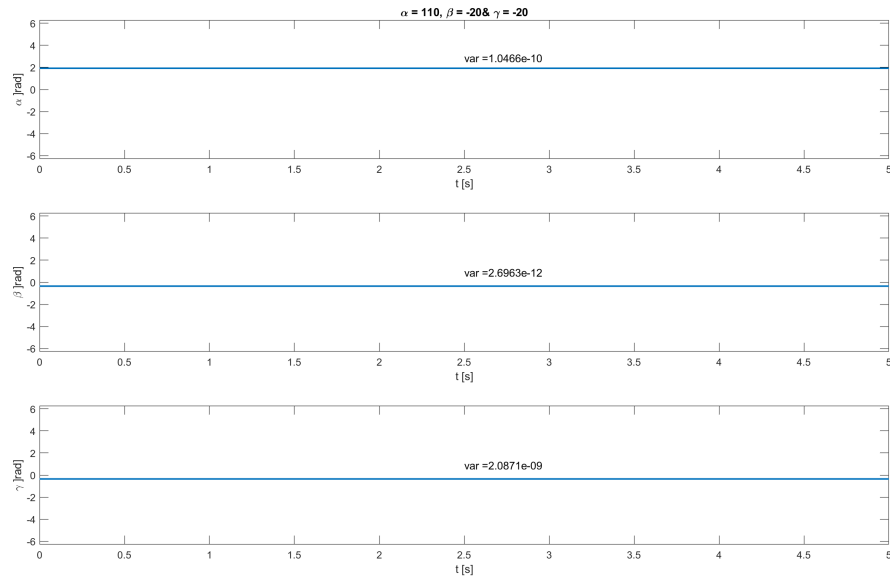
From the fact that the results get better when we increase the calculation precision we can see that the drift is caused by round off errors which accumulate when time goes on. Even in the case when we use the maximal precision (figure 5 there is a small drift. For the 5 second use in this example the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

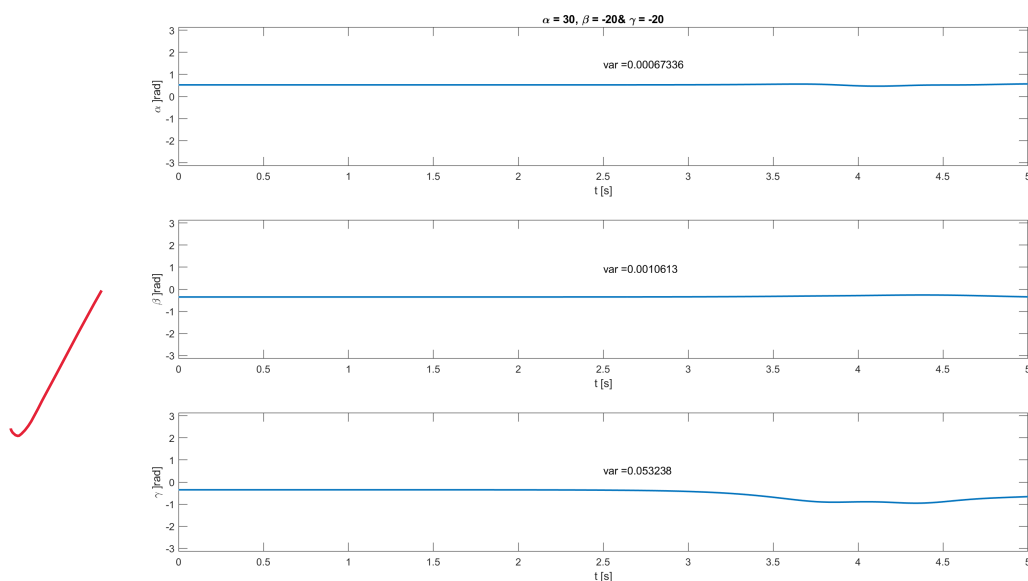
also with this level of precision there still is drift. This drift becomes noticeable for longer simulation times.

### 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:

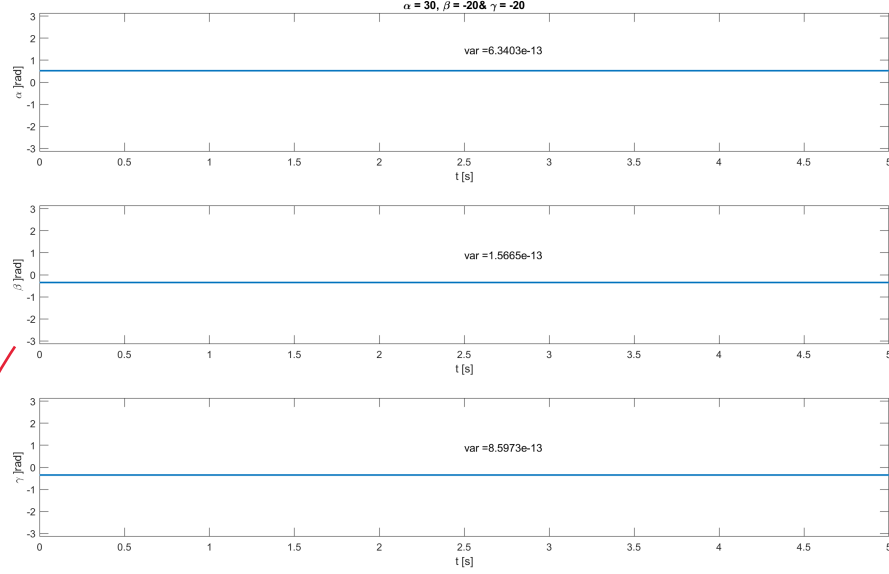
$$M_0 = \begin{bmatrix} -11.266905886022014 \\ 4.083129932327170 \\ -5.507705294606232 \end{bmatrix} \quad (33)$$

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 use Euler's equation of motion for 3D bodies. The equation of motion for the upper arm becomes:

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

and for the lower arm:

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

Since  $\omega_1$  and  $\omega_2$  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 as follows:

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

and

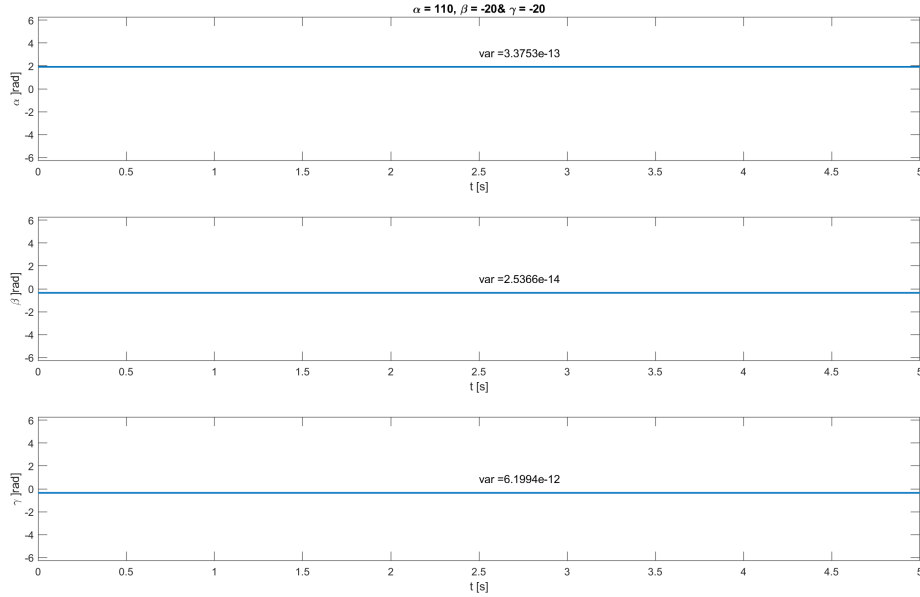
$$\mathcal{F}\omega_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{\gamma} \end{bmatrix} + R_{\beta}^{-1} \mathcal{B}\omega_1 \quad (37)$$

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

$$I_1 = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.004 \end{bmatrix} \quad (38)$$

$$I_2 = \begin{bmatrix} 0.04 & 0 & 0 \\ 0 & 0.002 & 0 \\ 0 & 0 & 0.04 \end{bmatrix} \quad (39)$$

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 probably 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:



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

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



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

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

```
1 %% MBD_B: Assignment 9 - Euler angles and the human arm
2 % Rick Staa (4511328)
3 clear all; close all; clc;
4 fprintf('--- A9 ---\n');
5
6 %% Set up needed symbolic parameters
7 % Create needed symbolic variables
8 syms alpha beta gamma alpha_d beta_d gamma_d alpha_dd beta_dd
   gamma_dd
9
10 % Put in parms struct for easy function handling
11 parms.syms.alpha      = alpha;
12 parms.syms.beta       = beta;
13 parms.syms.gamma      = gamma;
14 parms.syms.alpha_d    = alpha_d;
15 parms.syms.beta_d     = beta_d;
16 parms.syms.gamma_d    = gamma_d;
17 parms.syms.alpha_dd   = alpha_dd;
18 parms.syms.beta_dd    = beta_dd;
19 parms.syms.gamma_dd   = gamma_dd;
20
21 %% Intergration parameters
22 time                  = 5;
23                        % Intergration time
24
25 parms.h               = 1e-3;
26                        % Intergration step
27
28 size
29
30 %% Model Parameters
31 % Lengths and distances
32 parms.L1              = 0.3;
33                        % Length upper arm [m]
34
35 parms.L2              = 0.4;
36                        % Length lower arm [m]
37
38 parms.m1              = 3;
39                        % Mass upper arm [kg]
40
41 parms.m2              = 3;
42                        % Mass lower arm [kg]
```

```

32 %% World parameters
33 % Gravity
34 parms.g = 9.81;
                                     % [parms.m/s^2]
35
36 %% Set Initial states
37 alpha_0 = deg2rad(30);
38 beta_0 = deg2rad(-20);
39 gamma_0 = deg2rad(-20);
40 alpha_d_0 = 0;
41 beta_d_0 = 0;
42 gamma_d_0 = 0;
43 q0 = [alpha_0;beta_0;gamma_0;
        alpha_d_0;beta_d_0;gamma_d_0]; % Put in initial state
        vector
44
45 %% Calculate equilibrium torques
46 torque_calc(parms);
47 q0_tmp = num2cell(q0',1);
48 parms.Q = subs_torque(q0_tmp{:});
49 % parms.Q = [-11.2669 4.0831 -5.5077].';
50
51 %% Derive equation of motion
52 EOM_calc(parms);
53
54 % Calculate symbolic equations of motion and put in parms
55 struct
56
57 %% Calculate movement with ode
58 opt = odeset('AbsTol',1e-6,'RelTol',1
59 e-6,'Stats','on');
60 [t,q] = ode113(@(t,q) ODE_func(t,q), [0
61 time], q0',opt);
62
63 %% Animate movement
64 % animator(t,q,parms);
65
66 %% Plots
67
68 % Create plot label
69 title_str = strcat("\alpha = ", num2str(rad2deg(q0(1))),", \
70 beta = ", num2str(rad2deg(q0(2))),"& \gamma = ", num2str(
71 rad2deg(q0(3))));

```

```

65
66 % Plot angles vs time
67 figure;
68 subplot(3,1,1);
69 f1 = plot(t,q(:,1),'Linewidth',1.5);
70 ylim([-pi pi])
71 x_lim = xlim;
72 x_point = (x_lim(2)-x_lim(1))/2+x_lim(1);
73 text(x_point,mean(q(:,1))+0.3*pi,(strcat('var =',num2str(var(
    q(:,1))))));
74 xlabel('t [s]');
75 ylabel('\alpha [rad]');
76 title(title_str);
77 subplot(3,1,2)
78 f2 = plot(t,q(:,2),'Linewidth',1.5);
79 text(x_point,mean(q(:,2))+0.4*pi,(strcat('var =',num2str(var(
    q(:,2))))));
80 ylim([-pi pi])
81 xlabel('t [s]');
82 ylabel('\beta [rad]');
83 subplot(3,1,3)
84 f3 = plot(t,q(:,3),'Linewidth',1.5);
85 text(x_point,mean(q(:,3))+0.4*pi,(strcat('var =',num2str(var(
    q(:,3))))));
86 ylim([-pi pi])
87 xlabel('t [s]');
88 ylabel('\gamma [rad]');
89
90 %% FUNCTIONS
91
92 %% ANIMATION FUNCTION
93 function animator(t,q,parms)
94 %% Animation
95 % Adapted from A. Schwab's animation code
96
97 % Calculate elbow and wrist
98 x_full = zeros(size(q,1),12);
99 for ii = 1:size(q,1)
100     q_now = num2cell(q(ii,1:3),1);
101     x_full(ii,:) = subs_x_full(q_now{:})';
102 end
103
104 % Create figure

```



```

105 figure
106 set(gca,'fontsize',16)
107 title('Animation arm')
108 shoulder      = plot3(0,0,0,'*g');
109 hold on
110 upper_arm      = plot3([0 x_full(1,7)],[0 x_full(1,8)],[0
    x_full(1,9)],'-b');
111 elbow          = plot3(x_full(1,7),x_full(1,8),x_full(1,9)
    ,'*c');
112 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');
113 wrist          = plot3(x_full(1,10),x_full(1,11),x_full
    (1,12),'*m');
114 set(upper_arm,'LineWidth',5);
115 set(upper_arm,'Color','b')
116 set(lower_arm,'LineWidth',5);
117 set(lower_arm,'Color','r')
118 set(shoulder,'Linewidth',10);
119 set(elbow,'Linewidth',10);
120 set(wrist,'Linewidth',10);
121 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)]);
122 legend('shoulder','upper arm','elbow','lower arm','wrist');
123 nstep = length(t);
124 nskip = 10;
125 for istep = 2:nskip:nstep
126     set(upper_arm,'XData',[0 x_full(istep,7)])
127     set(upper_arm,'YData',[0 x_full(istep,8)])
128     set(upper_arm,'ZData',[0 x_full(istep,9)])
129     set(elbow,'XData',x_full(istep,7))
130     set(elbow,'YData',x_full(istep,8))
131     set(elbow,'ZData',x_full(istep,9))
132     set(lower_arm,'XData',[x_full(istep,7) x_full(istep,10)])
133     set(lower_arm,'YData',[x_full(istep,8) x_full(istep,11)])
134     set(lower_arm,'ZData',[x_full(istep,9) x_full(istep,12)])
135     set(wrist,'XData',x_full(istep,10))
136     set(wrist,'YData',x_full(istep,11))
137     set(wrist,'ZData',x_full(istep,12))
138     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)]);
139     drawnow

```

```

140     pause(1e-10)
141 end
142 end
143
144 %% ODE function handle
145 function [qdd] = ODE_func(t,q)
146 q_now = num2cell(q',1);
147 qdd    = subs_qdd(q_now{1:end});
148 qdd    = [q(4);q(5);q(6);qdd];
149 end
150
151 %% Calculate COM velocities
152 function [qdd,xdd] = state_calc(t,q)
153
154 % Create matrices
155 qdd      = zeros(size(q,1),6);
156 xdd      = zeros(size(q,1),6);
157
158 % Calculate qdd and xdd
159 for ii = 1:size(q,1)
160     q_now_1      = num2cell(q(ii,:),1);
161     qdd_tmp      = subs_qdd(q_now_1{1:end}).';
162     qdd(ii,:)    = [q(ii,4:6),qdd_tmp];
163     q_now_2      = num2cell([q(ii,:).';qdd_tmp.'].',1);
164     xdd(ii,:)    = subs_xdd(q_now_2{1:end}).';
165 end
166 end
167
168 %% Calculate COM torques
169 function torque_calc(parms)
170
171 % Unpack symbolic variables from varargin
172 alpha      = parms.syms.alpha;
173 beta       = parms.syms.beta;
174 gamma      = parms.syms.gamma;
175 alpha_d    = parms.syms.alpha_d;
176 beta_d     = parms.syms.beta_d;
177 gamma_d    = parms.syms.gamma_d;
178
179 % Create generalized coordinate vectors
180 q          = [alpha;beta;gamma];
181 qd         = [alpha_d;beta_d;gamma_d];
182

```

```

183 % Create rotation matrices
184 R_alpha      = rot_x(alpha);
185 R_beta       = rot_y(beta);
186 R_gamma      = rot_x(gamma);
187
188 % express COM positions in inertial frame and put them in a
    vector
189 r1_I          = R_alpha*R_beta*[0;0;-parms.L1/3];
190 r2_I          = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*
    R_beta*R_gamma*[0;0.5*parms.L2;0];
191
192 % Create mass matrix
193 parms.M        = diag([parms.m1,parms.m1,parms.m1,parms.m2,
    parms.m2,parms.m2]);
194
195 % Put in one state vector
196 x              = [r1_I;r2_I];
197
198 % Compute the jacobian of state and constraints
199 Jx_q           = simplify(jacobian(x,q.'));
200
201 %% Calculate convective component
202 Jx_dq          = jacobian(Jx_q*qd,q);
203
204 % Add forces F=[M2,F3_x,F3_y,M3,F4_x,F4_y,M4,F5_x,F5_y,M5,
    F6_x];
205 F_B            = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.
    g];
206
207 % Calculate result expressed in generalized coordinates
208 Q_0 = (Jx_q.'*(F_B-parms.M*Jx_dq*qd));
209
210 % Get x_full to animate the body
211 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
212
213 end
214
215 %% Calculate (symbolic) Equations of Motion four our setup
216 function EOM_calc(parms)
217

```

```

218 % Unpack symbolic variables from varargin
219 alpha          = parms.syms.alpha;
220 beta           = parms.syms.beta;
221 gamma          = parms.syms.gamma;
222 alpha_d        = parms.syms.alpha_d;
223 beta_d         = parms.syms.beta_d;
224 gamma_d        = parms.syms.gamma_d;
225 alpha_dd       = parms.syms.alpha_dd;
226 beta_dd        = parms.syms.beta_dd;
227 gamma_dd       = parms.syms.gamma_dd;
228
229 % Create rotation matrices
230 R_alpha        = rot_x(alpha);
231 R_beta         = rot_y(beta);
232 R_gamma        = rot_x(gamma);
233
234 % Create generalized coordinate vectors
235 q             = [alpha;beta;gamma];
236 qd            = [alpha_d;beta_d;gamma_d];
237 qdd           = [alpha_dd;beta_dd;gamma_dd];
238
239 % express COM positions in inertial frame and put them in a
    vector
240 r1_I          = R_alpha*R_beta*[0;0;-parms.L1/3];
241 r2_I          = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*
    R_beta*R_gamma*[0;0.5*parms.L2;0];
242
243 % Create mass matrix
244 parms.M       = diag([parms.m1,parms.m1,parms.m1,parms.m2,
    parms.m2,parms.m2]);
245
246 % Put in one state vector
247 x            = [r1_I;r2_I];
248
249 % Create full state for animation
250 x_elbow       = R_alpha*R_beta*[0;0;parms.L1];
251 x_wrist       = R_alpha*R_beta*[0;0;parms.L1]+R_alpha*
    R_beta*R_gamma*[0;parms.L2;0];
252 x_full        = [x;x_elbow;x_wrist];
253
254 % Compute the jacobian of state and constraints
255 Jx_q          = simplify(jacobian(x,q.'));
256

```

```

257 %% Calculate convective component
258 Jx_dq          = jacobian(Jx_q*qd,q);
259
260 % Solve with virtual power
261 M_bar          = Jx_q.'*parms.M*Jx_q;
262
263 % Add forces F=[M2,F3_x,F3_y,M3,F4_x,F4_y,M4,F5_x,F5_y,M5,
264   F6_x];
265 F_B            = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.
   g];
266 F              = Jx_q.'*(F_B-parms.M*Jx_dq*qd)-parms.Q;
   % Forces expressed in the inertial frame in
   generalised coordinates
267
268 % Calculate result expressed in generalized coordinates
269 qdp            = M_bar\F;
270
271 % Get result back in COM coordinates
272 xd             = Jx_q*qd;
273 xdd            = simplify(jacobian(xd,qd.'))*qdd + simplify(
   jacobian(xd,q.'))*qdd;
274
275 %% Convert to function handles
276 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
277
278 % Get back to COM coordinates
279 matlabFunction(simplify(x),'File','subs_x');
   % Create function
   handle of EOM in terms of generalised coordinates
280
281 % Get xdp COM coordinates
282 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
283
284 % Get x_full to animate the body
285 matlabFunction(simplify(x_full),'file','subs_x_full');
   % Create function handle of EOM
   in terms of generalised coordinates

```

```

285
286 % Get M_bar as function
287 matlabFunction(simplify(M_bar),'file','subs_M_bar');
      % Create function handle of EOM
      in terms of generalised coordinates
288
289 end
290
291 %% Rotation matrices
292 % x rotation matrix
293 function R_alpha = rot_x(angle)
294 R_alpha = [ 1 0 0 ;...
295 0 cos(angle) -sin(angle);...
296 0 sin(angle) cos(angle)];
297 end
298
299 % y rotation matrix
300 function R_beta = rot_y(angle)
301 R_beta = [ cos(angle) 0 sin(angle);...
302 0 1 0;...
303 -sin(angle) 0 cos(angle)];
304 end
305
306 % z rotation matrix
307 function R_gamma = rot_z(angle)
308 R_gamma = [cos(angle) -sin(angle) 0;...
309 sin(angle) cos(angle) 0;...
310 0 0 1];
311 end

```

## Appendix B

### The main MATLAB script part2

```

1 %% MBD_B: Assignment 9 - Euler angles and the human arm
2 % Rick Staa (4511328)
3 clear all; close all; clc;
4 fprintf('--- A9 ---\n');
5
6 %% Set up needed symbolic parameters
7 % Create needed symbolic variables

```

```

8 | syms alpha beta gamma alpha_d beta_d gamma_d alpha_dd beta_dd
   | gamma_dd
9 |
10 | % Put in parms struct for easy function handling
11 | parms.syms.alpha      = alpha;
12 | parms.syms.beta       = beta;
13 | parms.syms.gamma      = gamma;
14 | parms.syms.alpha_d    = alpha_d;
15 | parms.syms.beta_d     = beta_d;
16 | parms.syms.gamma_d    = gamma_d;
17 | parms.syms.alpha_dd   = alpha_dd;
18 | parms.syms.beta_dd    = beta_dd;
19 | parms.syms.gamma_dd   = gamma_dd;
20 |
21 | %% Intergration parameters
22 | time                  = 5;
   |                               % Intergration time
23 | parms.h               = 1e-3;
   |                               % Intergration step
   | size
24 |
25 | %% Model Parameters
26 | % Lengths and distances
27 | parms.L1              = 0.3;
   |                               % Length upper arm [m]
28 | parms.L2              = 0.4;
   |                               % Length lower arm [m]
29 | parms.m1              = 3;
   |                               % Mass upper arm [kg]
30 | parms.m2              = 3;
   |                               % Mass lower arm [kg]
31 | parms.I1              = diag([0.025 0.025 0.004]);
   |                               % Inertial matrix body 1 (Expressed in body
   | frame)
32 | parms.I2              = diag([0.040 0.002 0.040]);
   |                               % Inertial matrix body 2 (Expressed in body
   | frame)
33 |
34 | %% World parameters
35 | % Gravity
36 | parms.g               = 9.81;
   |                               % [parms.m/s^2]
37 |

```

```

38 %% Set Initial states
39 alpha_0 = deg2rad(110);
40 beta_0 = deg2rad(-20);
41 gamma_0 = deg2rad(-20);
42 alpha_d_0 = 0;
43 beta_d_0 = 0;
44 gamma_d_0 = 0;
45 q0 = [alpha_0;beta_0;gamma_0;
      alpha_d_0;beta_d_0;gamma_d_0]; % Put in initial state
      vector
46
47 %% Calculate equilibrium torques
48 torque_calc(parms);
49 q0_tmp = num2cell(q0',1);
50 parms.Q = subs_torque(q0_tmp{:});
51
52 %% Derive equation of motion
53 EOM_calc(parms);
54
55 % Calculate symbolic equations of motion and put in parms
56 struct
57
58 %% Calculate movement with ode
59 opt = odeset('AbsTol',1e-6,'RelTol',1
60 e-6,'Stats','on');
61 [t,q] = ode113(@(t,q) ODE_func(t,q), [0
62 time], q0',opt);
63
64 %% Animate movement
65 % animator(t,q,parms);
66
67 %% Plots
68
69 % Create plot label
70 title_str = strcat("\alpha = ", num2str(rad2deg(q0(1))),", \
71 beta = ", num2str(rad2deg(q0(2))),"& \gamma = ", num2str(
rad2deg(q0(3))));
72
73 % Plot angles vs time
74 figure;
75 subplot(3,1,1);
76 f1 = plot(t,q(:,1),'Linewidth',1.5);
77 ylim([-2*pi 2*pi])

```



```

72 x_lim = xlim;
73 x_point = (x_lim(2)-x_lim(1))/2+x_lim(1);
74 text(x_point,mean(q(:,1))+0.4*pi,(strcat('var =',num2str(var(
    q(:,1))))));
75 xlabel('t [s]');
76 ylabel('\alpha [rad]');
77 title(title_str);
78 subplot(3,1,2)
79 f2 = plot(t,q(:,2),'Linewidth',1.5);
80 text(x_point,mean(q(:,2))+0.4*pi,(strcat('var =',num2str(var(
    q(:,2))))));
81 ylim([-2*pi 2*pi])
82 xlabel('t [s]');
83 ylabel('\beta [rad]');
84 subplot(3,1,3)
85 f3 = plot(t,q(:,3),'Linewidth',1.5);
86 text(x_point,mean(q(:,3))+0.4*pi,(strcat('var =',num2str(var(
    q(:,3))))));
87 ylim([-2*pi 2*pi])
88 xlabel('t [s]');
89 ylabel('\gamma [rad]');
90
91 %% FUNCTIONS
92
93 %% ANIMATION FUNCTION
94 function animator(t,q,parms)
95 %% Animation
96 % Adapted from A. Schwab's animation code
97
98 % Calculate elbow and wrist
99 x_full = zeros(size(q,1),12);
100 for ii = 1:size(q,1)
101     q_now = num2cell(q(ii,1:3),1);
102     x_full(ii,:) = subs_x_full(q_now{:})';
103 end
104
105 % Create figure
106 figure
107 set(gca,'fontsize',16)
108 title('Animation arm')
109 shoulder = plot3(0,0,0,'*g');
110 hold on

```

```

111 upper_arm          = plot3([0 x_full(1,7)],[0 x_full(1,8)],[0
    x_full(1,9)], '-b');
112 elbow              = plot3(x_full(1,7),x_full(1,8),x_full(1,9)
    , '*c');
113 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');
114 wrist              = plot3(x_full(1,10),x_full(1,11),x_full
    (1,12), '*m');
115 set(upper_arm, 'LineWidth',5);
116 set(upper_arm, 'Color','b')
117 set(lower_arm, 'LineWidth',5);
118 set(lower_arm, 'Color','r')
119 set(shoulder, 'Linewidth',10);
120 set(elbow, 'Linewidth',10);
121 set(wrist, 'Linewidth',10);
122 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)]);
123 legend('shoulder','upper arm','elbow','lower arm','wrist');
124 nstep = length(t);
125 nskip = 10;
126 for istep = 2:nskip:nstep
127     set(upper_arm, 'XData',[0 x_full(istep,7)])
128     set(upper_arm, 'YData',[0 x_full(istep,8)])
129     set(upper_arm, 'ZData',[0 x_full(istep,9)])
130     set(elbow, 'XData',x_full(istep,7))
131     set(elbow, 'YData',x_full(istep,8))
132     set(elbow, 'ZData',x_full(istep,9))
133     set(lower_arm, 'XData',[x_full(istep,7) x_full(istep,10)])
134     set(lower_arm, 'YData',[x_full(istep,8) x_full(istep,11)])
135     set(lower_arm, 'ZData',[x_full(istep,9) x_full(istep,12)])
136     set(wrist, 'XData',x_full(istep,10))
137     set(wrist, 'YData',x_full(istep,11))
138     set(wrist, 'ZData',x_full(istep,12))
139     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)]);
140     drawnow
141     pause(1e-10)
142 end
143 end
144
145 %% ODE function handle

```

```

146 function [qdp] = ODE_func(t,q)
147     q_now = num2cell(q',1);
148     qdp    = subs_qdp(q_now{1:end});
149     qdp    = [q(4);q(5);q(6);qdp];
150 end
151
152 %% Calculate COM velocities
153 function [qdd,xdd] = state_calc(t,q)
154
155 % Create matrices
156 qdd = zeros(size(q,1),6);
157 xdd = zeros(size(q,1),6);
158
159 % Calculate qdd and xdd
160 for ii = 1:size(q,1)
161     q_now_1 = num2cell(q(ii,:),1);
162     qdd_tmp = subs_qdd(q_now_1{1:end}).';
163     qdd(ii,:) = [q(ii,4:6),qdd_tmp];
164     q_now_2 = num2cell([q(ii,:).';qdd_tmp.'].',1);
165     xdd(ii,:) = subs_xdd(q_now_2{1:end}).';
166 end
167 end
168
169 %% Calculate COM torques
170 function torque_calc(parms)
171
172 % Unpack symbolic variables from varargin
173 alpha = parms.syms.alpha;
174 beta  = parms.syms.beta;
175 gamma = parms.syms.gamma;
176 alpha_d = parms.syms.alpha_d;
177 beta_d  = parms.syms.beta_d;
178 gamma_d = parms.syms.gamma_d;
179
180 % Create generalized coordinate vectors
181 q = [alpha;beta;gamma];
182 qd = [alpha_d;beta_d;gamma_d];
183
184 % Create rotation matrices
185 R_alpha = rot_x(alpha);
186 R_beta  = rot_y(beta);
187 R_gamma = rot_x(gamma);
188

```

```

189 % express COM positions in inertial frame and put them in a
    vector
190 r1_I          = R_alpha*R_beta*[0;0;-parms.L1/3];
191 r2_I          = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*
    R_beta*R_gamma*[0;0.5*parms.L2;0];
192
193 % Create mass matrix
194 parms.M        = diag([parms.m1,parms.m1,parms.m1,parms.m2,
    parms.m2,parms.m2]);
195
196 % Put in one state vector
197 x              = [r1_I;r2_I];
198
199 % Compute the jacobian of state and constraints
200 Jx_q           = simplify(jacobian(x,q.'));
201
202 %% Calculate convective component
203 Jx_dq          = jacobian(Jx_q*qd,q);
204
205 % Calculate angular velocities in the body fixed frames
206 omega_1        = [0;beta_d;0]+R_beta\[alpha_d;0;0];
207 T_omega_1_qd    = jacobian(omega_1,qd);
208 omega_2        = [gamma_d;0;0]+R_gamma\omega_1;
209 T_omega_2_qd    = jacobian(omega_2,qd);
210 omega          = [omega_1;omega_2];
211
212 % Create new Transformation matrix
213 T_new          = [Jx_q;T_omega_1_qd;T_omega_2_qd];
214
215 % Create new Mass matrix
216 M_new          = blkdiag(parms.M,parms.I1,parms.I2);
217
218 % Calculate convective terms
219 I_c            = blkdiag(parms.I1,parms.I2);
220 Ic_omega       = (I_c*omega);
221 omega_cross    = [cross(omega(1:3,1),Ic_omega(1:3,1)) ...
    ;cross(omega(4:end,1),Ic_omega(4:end,1))];
222
223
224 % Calculate qdp
225 F_ext          = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.
    g;0;0;0;0;0;0]; % External forces on COMs
226 G              = [Jx_dq*qd;omega_cross];
227 Q_0            = T_new.'*(F_ext-M_new*G);

```

```

228
229 % Get x_full to animate the body
230 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
231
232 end
233
234 %% Calculate (symbolic) Equations of Motion four our setup
235 function EOM_calc(parms)
236
237 % Unpack symbolic variables from varargin
238 alpha      = parms.syms.alpha;
239 beta       = parms.syms.beta;
240 gamma      = parms.syms.gamma;
241 alpha_d    = parms.syms.alpha_d;
242 beta_d     = parms.syms.beta_d;
243 gamma_d    = parms.syms.gamma_d;
244 alpha_dd   = parms.syms.alpha_dd;
245 beta_dd    = parms.syms.beta_dd;
246 gamma_dd   = parms.syms.gamma_dd;
247
248 % Create rotation matrices
249 R_alpha     = rot_x(alpha);
250 R_beta      = rot_y(beta);
251 R_gamma     = rot_x(gamma);
252
253 % Create generalized coordinate vectors
254 q           = [alpha; beta; gamma];
255 qd          = [alpha_d; beta_d; gamma_d];
256 qdd         = [alpha_dd; beta_dd; gamma_dd];
257
258 % express COM positions in inertial frame and put them in a
    vector
259 r1_I        = R_alpha*R_beta*[0;0;-parms.L1/3];
260 r2_I        = R_alpha*R_beta*[0;0;-parms.L1]+R_alpha*
    R_beta*R_gamma*[0;0.5*parms.L2;0];
261
262 % Create mass matrix
263 parms.M     = diag([parms.m1, parms.m1, parms.m1, parms.m2,
    parms.m2, parms.m2]);
264

```

```

265 % - Uncomment when you want full symbolic expression -
266 syms m1 m2 Ix1 Iy1 Iz1 Ix2 Iy2 Iz2
267 parms.M      = diag([m1,m1,m1,m2,m2,m2]);
268 parms.I1     = [Ix1 0 0;0 Iy1 0;0 0 Iz1];
269 parms.I2     = [Ix2 0 0;0 Iy2 0;0 0 Iz2];
270 % - Uncomment when you want full symbolic expression -
271
272 % Put in one state vector
273 x            = [r1_I;r2_I];
274
275 % Create full state for animation
276 x_elbow      = R_alpha*R_beta*[0;0;parms.L1];
277 x_wrist      = R_alpha*R_beta*[0;0;parms.L1]+R_alpha*
    R_beta*R_gamma*[0;parms.L2;0];
278 x_full       = [x;x_elbow;x_wrist];
279
280 % Compute the jacobian of state and constraints
281 Jx_q         = simplify(jacobian(x,q.'));
282
283 %% Calculate convective component
284 Jx_dq        = jacobian(Jx_q*qd,q);
285
286 % Calculate angular velocities in the body fixed frames
287 omega_1      = [0;beta_d;0]+R_beta\[alpha_d;0;0];
288 T_omega_1_qd = jacobian(omega_1,qd);
289 omega_2      = [gamma_d;0;0]+R_gamma\omega_1;
290 T_omega_2_qd = jacobian(omega_2,qd);
291 omega        = [omega_1;omega_2];
292
293 % Create new Transformation matrix
294 T_new        = [Jx_q;T_omega_1_qd;T_omega_2_qd];
295
296 % Create new Mass matrix
297 M_new        = blkdiag(parms.M,parms.I1,parms.I2);
298
299 % Create new TMT matrix
300 M_bar        = simplify(T_new.'*M_new*T_new);
301
302 % Calculate convective terms
303 I_c          = blkdiag(parms.I1,parms.I2);
304 Ic_omega     = (I_c*omega);
305 omega_cross  = [cross(omega(1:3,1),Ic_omega(1:3,1)) ...
306                ;cross(omega(4:end,1),Ic_omega(4:end,1))];

```

```

307
308 % Calculate qdp
309 F_ext = [0;0;-parms.m1*parms.g;0;0;-parms.m2*parms.
      g;0;0;0;0;0;0]; % External forces on COMs
310 G = [Jx_dq*qd;omega_cross];
311 Q_bar = simplify(T_new.'*(F_ext-M_new*G) - parms.Q)
      ;
312
313 % Calculate reesult expressed in generalized coordinates
314 qdp = M_bar\Q_bar;
315
316 % Get result back in COM coordinates
317 xd = Jx_q*qd;
318 xdd = simplify(jacobian(xd,qd.'))*qdd + simplify(
      jacobian(xd,q.'))*qdd;
319
320 %% Convert to function handles
321 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
322
323 % Get back to COM coordinates
324 matlabFunction(simplify(x),'File','subs_x');
      % Create function
      handle of EOM in terms of generalised coordinates
325
326 % Get xdp COM coordinates
327 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
328
329 % Get x_full to animate the body
330 matlabFunction(simplify(x_full),'file','subs_x_full');
      % Create function handle of EOM
      in terms of generalised coordinates
331
332 % Get M_bar as function
333 matlabFunction(simplify(M_bar),'file','subs_M_bar');
      % Create function handle of EOM
      in terms of generalised coordinates
334

```

```

335 end
336
337 %% Rotation matrices
338 % x rotation matrix
339 function R_alpha = rot_x(angle)
340     R_alpha = [ 1      0      0      ;...
341                0      cos(angle)  -sin(angle);...
342                0      sin(angle)   cos(angle)];
343 end
344
345 % y rotation matrix
346 function R_beta = rot_y(angle)
347     R_beta = [ cos(angle)      0      sin(angle);
348               ...
349               0      1      0;...
350               -sin(angle)  0      cos(angle)];
351 end
352 % z rotation matrix
353 function R_gamma = rot_z(angle)
354     R_gamma = [cos(angle)      -sin(angle)      0;
355               ...
356               sin(angle)      cos(angle)      0;
357               ...
358               0      0      1];
359 end

```



## References

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