I used [1] in making this assignment when finished I compared initial values with Prajish Kumar (4743873).	**Rick Staa, #4511328** **HW set 4 due 20/03/2018** **ME41060**

1 Statement of entregity

my homework is completely in accordance with the Academic Integrity

2 Problem Statement

In this assignment, we were asked to redo the derivation of the equations of motion of assignment 2 and 3. This time we must use independent generalised coordinates and the Lagrange equation to do this. As a result, the and its parameters (mass, length and inertia) are still those depicted in assignment 1. In this assignment due to the generalised coordinates the initial state changes. For our double pendulum model, since it has 2 degrees of freedom (DOF) a good choice for the generalised coordinates are φ_1 and φ_2 (See Figure 1: Figure of our setup). Using these generalised coordinates, the initial state now becomes $q_0 = [\varphi_1\ \varphi_2]$.

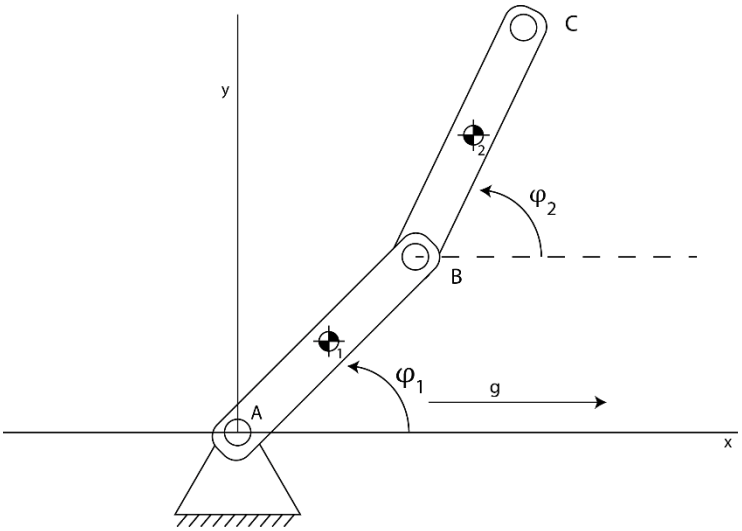

Figure 1: Figure of our setup

3 Derivation of the EOM using the Lagrange

In his method Lagrange makes use of the principle of energy to get the equations of motion (EOM). As explained in [1] these EOM can be derived out of the total derivative of the energy equation of the system. To do this we first need to define a potential energy function V:

$$\frac{\partial V}{\partial x} = -F \tag{1}$$

The energy equation of our system can be calculated by integrating the power over the time. The power of the system is equation to:

$$P = m\ddot{x}\dot{x} \tag{2}$$

So the energy of the system becomes:

$$\int F\dot{x}dt = \int m\ddot{x}\dot{x}dt \tag{3}$$
$$\int Fdx = \int m\ddot{x}d\dot{x} \tag{4}$$

Following we obtain the energy equation by evaluate the integrals. For the case that the forces are constant and conservative we get the following energy equation:

$$T + V = constant \tag{5}$$

From this we can see that the EOM can be derived by taking the total derivative of the energy equation. When extending this result for non-conservative forces we get the following total derivative:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{x}_i}\right) - \frac{\partial V}{\partial x_i} = F_i \tag{6}$$

In this equation T contains the kinetic energy of the system while V contains the potential energy of the system and F_i depicts the energy of the non-conservative forces on the system. To make the resulting equations of motion more compact we can express this equation of motion in terms of our generalised coordinates q (φ_1, φ_2). The new equation now becomes:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = Q_j \tag{7}$$

In this Q_j depicts the generalized forces. These generalised forces are the forces working on the body but now not acting on the COM but on the generalised coordinates. We now need to write this in a matrix vector product again to be able to solve for the unknown accelerations. We can do this by rewriting the first term with the multivariate chain rule:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right) = \frac{\partial}{\partial \dot{q}}\left(\frac{\partial T}{\partial \dot{q}}\right)\ddot{q} + \frac{\partial}{\partial q}\left(\frac{\partial T}{\partial \dot{q}_i}\right)\dot{q} \tag{8}$$

When we full this in in equation (7) and rewrite the formula in a matrix vector product with the known terms at the right side we get:

$$\frac{\partial}{\partial \dot{q}}\left(\frac{\partial T}{\partial \dot{q}_i}\right)\ddot{q} = Qi - \frac{\partial}{\partial q}\left(\frac{\partial T}{\partial \dot{q}_i}\right)\dot{q} - \frac{\partial V}{\partial q_i} + \frac{\partial T}{\partial q_i} \tag{9}$$

This results in the following matrix vector product:

$$M_{ij}\ddot{q}_j = F_i \tag{10}$$

0.3

Solving this matrix vector product gives us the accelerations in terms of the generalised coordinates. We still need to express the accelerations in terms of the COM coordinates.

3.1 Double pendulum without constraints (Question 2 a-d)

As stated earlier the normal double pendulum without additional constraint has 2 generalised coordinates (α, β). We can express the centre of mass of both bodies in these generalised coordinates as follows:

$$x_1 = \frac{L}{2}\cos(\varphi_1) \tag{11}$$

$$y_1 = \frac{L}{2}\sin(\varphi_2) \tag{12}$$

$$x_2 = x_1 + \frac{L}{2}\cos(\varphi_1) + \frac{L}{2}\cos(\varphi_2) \tag{13}$$

$$y_2 = y_1 + \frac{L}{2}\sin(\varphi_1) + \frac{L}{2}\sin(\varphi_2) \tag{14}$$

We then use MATLAB symbolic toolbox to derive the velocities expressed in the generalised coordinates and use these velocities together with the x positions to get the Kinetic and Potential energy.

$$T = \frac{1}{2}\dot{x}^T M \dot{x} \tag{15}$$

$$V = -[mg \quad 0 \quad 0 \quad mg \quad 0 \quad 0]x \tag{16}$$

In these $x = [x_1\ y_1\ \varphi_1\ x_2\ y_2\ \varphi_2]$, $\dot{x} = [\dot{x}_1\ \dot{y}_1\ \dot{\varphi}_1\ \dot{x}_2\ \dot{y}_2\ \dot{\varphi}_2]$ and M is the mass matrix. The full implementation of these steps can be found in Appendix A –

3.1.1 Accelerations (Question 2 a-d)

We were then asked to use the new EOM to recalculate results of assignment 2 a-d. In assignment 2 a-d the following initial conditions were used:

- $x_0 = [0.5\ \pi\ 0.5\pi\ 0\ 0]$
- $x_0 = [0\ 0\ 0\ 0]$
- $x_0 = [0\ 0\ 2\pi\ 2\pi]$

The accelerations calculated with the EOM as they were derived by the Lagrange constraint equations were found to be exactly equal to those derived earlier in assignment 1. To check this run the MATLAB script in Appendix A – The results of this run are:

Table 1: Results of recalculation Assignment 2 a-d

	A2 - b	A2 – c	A2 - d
$\ddot{x}_1$	6.31 m/s^2	0.00 m/s^2	-10.8566 m/s^2
$\ddot{y}_1$	0.00 m/s^2	0.00 m/s^2	0.00 m/s^2
$\ddot{\phi}_1$	-22.93 rad/s^2	0.00 rad/s^2	0.00 rad/s^2
$\ddot{x}_2$	10.51 m/s^2	0.00 m/s^2	$-32.5697\ m/s^2$
$\ddot{y}_2$	0.00 m/s^2	0.00 m/s^2	0.00 m/s^2
$\ddot{\phi}_2$	7.64 rad/s^2	0.00 rad/s^2	0.00 rad/s^2

3.2 Vertical slider constraint (Question 2 e-f)

In assignment 2 e-f we were asked to add a horizontal slider constraint. In this assignment the Lagrange method was used to derive the EOM incorporating this constraint. The difference with question 2 is

that for the Lafrance method to work the constraint needs to be expressed in the generalised coordinates instead of the full state. Since the second bar is now constraint on the vertical axis the pendulum only has 1 DOF. As a result, the second angle is dependent on the first angle as $\varphi_1 = \pi - \varphi_2$. The COM of both the first and the second body can now be expressed with one generalised coordinate φ_1. The MATLAB implementation can be found in Appendix B – Question 2 e-f

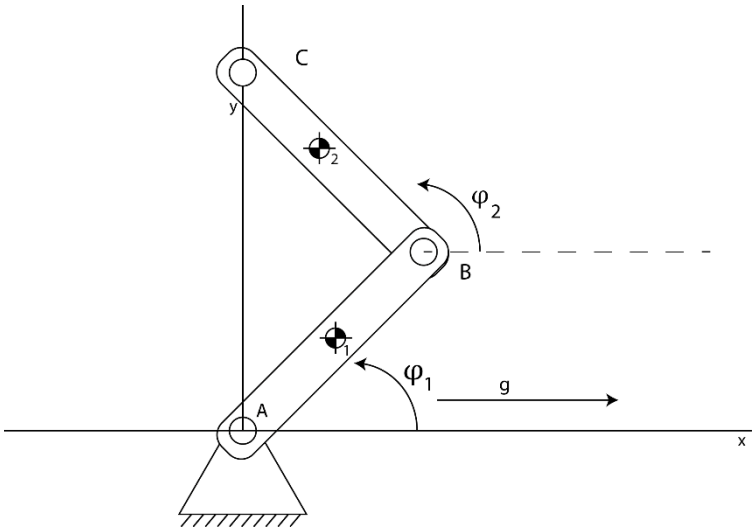

Figure 2: New pendulum setup with the extra slider constraint

3.3 Vertical slider constraint + extra vertical force in B (Question 2 g)

For G we were asked to look at the same situation as 2f but now also add a extra constraint force to bar 1 in point B. When using the Lagrange method, we can add this force to the system as a generalised force. To do this the force needs to be expressed as a generalised force (A force expressed in the generalised coordinates). This is done as follows [1]:

$$Q_j = \frac{\partial x_i}{\partial q_j} F_i = \frac{\partial x_i}{\partial q_j} \begin{bmatrix} 0 & 10 & 10\left(\frac{L}{2}\right) * \cos(\varphi_1) & 0 & 0 & 0 \end{bmatrix} \tag{17}$$

3.3.1 Accelerations (Question 2 e-g)

Table 2: Results of recalculation assignment 2 e-g

	A2 - e	A2 – f	A2 - g
$\ddot{x}_1$	7.36 m/s^2	7.36 m/s^2	0.00 m/s^2
$\ddot{y}_1$	0.00 m/s^2	-10.86 m/s^2	14.44.00 m/s^2
$\ddot{\phi}_1$	-26.76 rad/s^2	$-26.76\ rad/s^2$	52.50 rad/s^2
$\ddot{x}_2$	7.36 m/s^2	7.36 m/s^2	0.00 m/s^2
$\ddot{y}_2$	0.00 m/s^2	$-32.57\ m/s^2$	43.3135 m/s^2
$\ddot{\phi}_2$	26.76 rad/s^2	26.76 rad/s^2	-52.50 rad/s^2

From these results we see that the results of the recalculation of question A2-e and A2-f are exactly equal to those calculated in assignment 2. The big difference (benefit) between this method and the method used in Assignment 2 can be seen by looking at the result of A2-g. Where in A2-g we had no solution due to the linear dependence of the constraints here we have a solution since the linear

dependence is considered by setting $\phi_2 = \pi - \varphi_1$. Since we now only have one generalised coordinate left we get a unique solution.

3.4 Double pendulum with Passive spring constraint (Question 3 a)

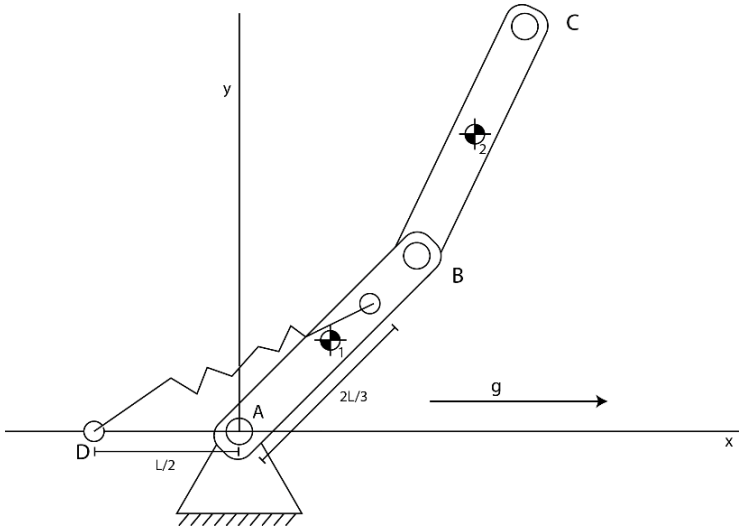

Figure 3: Addition of a passive element

Here were asked to recalculate the results of assignment 3 with the method of Lagrange. In assignment 3 a passive element, an active element and an impact force were added to the model. Let's first start with adding the passive element. This spring element is depicted in Figure 3: Addition of a passive element and is attached to the ground in D, with coordinates (−L/2, 0) and connected to bar 1 in point E, being at 2/3 of the length measured from point A. The free length of this spring was said to be $l_o = \frac{2L}{3}$ and its linear stiffness k = (15/2)(mg/l).With the method of LaGrange adding a spring is easy since a linear spring is a conservative force. As a result, the spring force can just be added to the potential energy equation V. Since the both the potential function as the spring force are defined as being negative (1) we get the following result:

$$V_s = \frac{1}{2}k(l_x - l_0)^2 \tag{18}$$

We when we express the rest length l_o and current length of the spring in the generalised coordinates we get the following result:

$$V_s = \frac{1}{2}k\left(\sqrt{\left(x_1 + \frac{L}{6}\cos(\varphi_1) + \frac{L}{2}\right)^2 + \left(y_1 + \frac{L}{6}\sin(\varphi_1) - 0\right)^2} - l_0\right)^2 \tag{19}$$

The script implementing this new potential force component can be found in Appendix D – Question 3 a

3.5 Double pendulum with active motor element (Question 3 b)

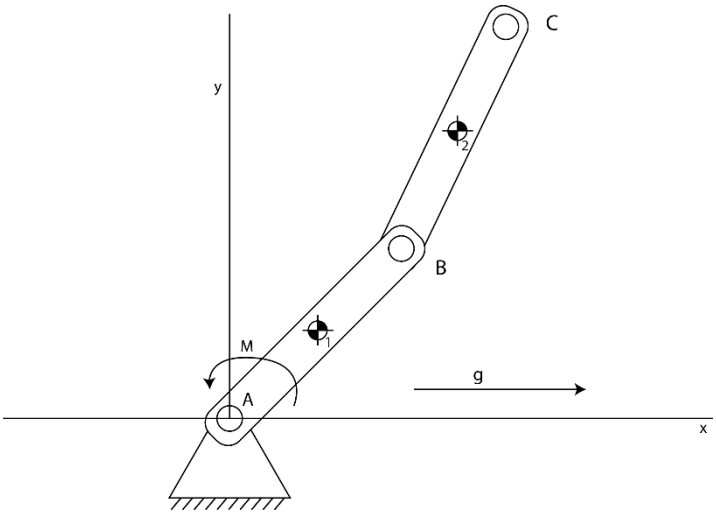

Figure 4: Setup with active motor element

We were then asked to add an active motor element to the pendulum. This motor is assumed to have a constant angular velocity $\omega = \varphi_1 - \varphi_2$ of 120 rpm. As a result, the motor constraint can be written as:

$$C_m(q,t) = \varphi_1 - \omega t \tag{20}$$

This force can be added to LaGrange equation system by making use of the concept of virtual power. To be able to add the constraint we need to differentiate the constraint two times. This is done in the Simulink script of Appendix E – Question 3 b but this can be easily done by hand by making use of the chain rule. This new constraint equation can be added to the virtual power equation by making use of LaGrange multipliers. When stating that the virtual power equation must hold for all virtual velocities that meet the constraints you get the following system of equations:

$$\begin{bmatrix} \bar{M}_{ij} & C_{,q}^T \\ C_{,q} & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q_j \\ -C_{mi,j}\dot{q}_i\dot{q}_j - C_{m,it}\dot{q}_i - C_{m,tt} \end{bmatrix}$$

In this $C_{mi,j}$, $C_{m,it}$ and $C_{m,tt}$ are the partial derivatives of the motor constraint equation with respect to the generalised coordinates and time respectively. Q_j are the non-conservative forces, $\overline{M_{ij}}$ the generalised mass matrix. This mass matrix can be written as $\bar{M}_{ij} = X_{ij}^T M_{ij} X_{ij}$ in which X_{ij} is the Jacobean of x. The lambda now contains the force of the motor.

3.6 Double pendulum with impact force (Question 3 c)

For this we can use the formulas we derived in assignment A to calculate the impact forces. Since we are now working with generalised coordinates we only need to express them into generalised coordinates. When we do this we get the following result:

$$\begin{bmatrix} \bar{M}_{ij} & C_{k,j}^T \\ C_{k,j} & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_j^+ \\ \rho_k \end{bmatrix} = \begin{bmatrix} s_i + \bar{M}_{ij}\dot{q}^- \\ -eC_{k,j}\dot{q}^- \end{bmatrix} \tag{21}$$

Now similar to assignment 3 C the $C_{k,j}$ matrix contains the constraints on B and C. These constraints are:

$$Lcos(\varphi_1) = 0 \quad (22)$$

$$Lcos(\varphi_1) + Lcos(\varphi_2) = 0 \quad (23)$$

The code solving the LaGrangian equations of motion can be found in Appendix E – Question 3 C

3.6.1 Accelerations (Question 3 a-b)

Table 3: Results of recalculation Assignment 3 a-b

	A3 - a	**A3 – b**
$\ddot{x}_1$	2.10 m/s^2	0.00 m/s^2
$\ddot{y}_1$	0.00 m/s^2	-43.43 m/s^2
$\ddot{\phi}_1$	-7.64 rad/s^2	0.00 rad/s^2
$\ddot{x}_2$	8.40 m/s^2	$-93.47\ m/s^2$
$\ddot{y}_2$	0.00 m/s^2	$-130.28\ m/s^2$
$\ddot{\phi}_2$	-15.29 rad/s^2	339.90 rad/s^2

0.3

The motor torque is calculated to be -3.1263 Nm meaning we have a power input of 39.2862 W. Since some of my values are somewhat different to the values of Question A3 there is a small error in there. I however had not time to find the error.

3.6.2 Accelerations (Question 3 C)

The results of the impact equations are the same as in assignment 3C.

Table 4: Results of recalculation Assignment 3 a-b

	e = 0	**e =0.5**	**e = 1**
$\ddot{x}_1$	0.00 m/s^2	1.73 m/s^2	3.46 m/s^2
$\ddot{y}_1$	0.00 m/s^2	0.00 m/s^2	0.00 m/s^2
$\ddot{\phi}_1$	0.00 rad/s^2	$-6.28\ rad/s^2$	$-12.37\ rad/s^2$
$\ddot{x}_2$	0.00 m/s^2	5.19 m/s^2	10.37 m/s^2
$\ddot{y}_2$	0.00 m/s^2	0.00 m/s^2	0.00 m/s^2
$\ddot{\phi}_2$	0.00 rad/s^2	-6.28 rad/s^2	-12.57 rad/s^2
λ_1	1.		
λ_2			

0.3

3.7 How to get the forces

You can get the forces by post calculating them with the constraint matrix you obtained in Assignment 2. This constraint matrix is:

0

Too short

$$\boldsymbol{C} = \begin{bmatrix} x_1 - \frac{L}{2}cos(\phi_1) \\ y_1 - \frac{L}{2}sin(\phi_1) \\ -x_1 - \frac{L}{2}cos(\varphi_1) - \frac{L}{2}cos(\varphi_2) + x_2 \\ -y_1 - \frac{L}{2}sin(\varphi_1) - \frac{L}{2}sin(\varphi_2) + y_2 \end{bmatrix}$$

4 References

[1] A. L. Schwab, " Reader: MultiBody Dynamics B," in *Multibody Dynamics*, Delft, The Netherlands: TU Delft, 2018.

Appendix A – Question 2 a-d

In this appendix you will find the code in which the new Lagrange constraint equations were used to redo assignment 2 a-d. In assignment 2 a-d there were no constraints.

MATLAB CODE

```
%% MBD_B: Assignment 4 - Double pendulum systematic approach
%  Rick Staa (4511328)
%  Last edit: 19/03/2018
% Question A: Redoo assignment 1

clear all; close all; clc;
fprintf('--- A4_a ---\n');
fprintf('First lets redo A2 - In this case there are no constraints\n')

%% Script settings and parameters
variable              = {'x1dp' 'y1dp' 'phi1dp' 'x2dp' 'y2dp' 'phi2dp'}';

%% Parameters
% Segment 1
parms.L               = 0.55;                                          % [parms.m]
parms.w               = 0.05;                                          % [parms.m]
parms.t               = 0.004;                                         % [parms.m]
parms.p               = 1180;                                          % [kg/parms.m^3]
parms.m               = parms.p * parms.w * parms.t * parms.L;         % [kg]
parms.I               = (1/12) * parms.m * parms.L^2;                  % [kg*parms.m^2]

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

%% Calculate state_dp for initial states
% A2 - a:
% A1 - b
x0              = [0.5*pi 0.5*pi 0 0];
xdp_A1_b        = double(state_calc(x0,parms));
fprintf('\nThe result for A1 - b is:\n');
disp(table(variable,xdp_A1_b));

% A1 - c
x0              = [0 0 0 0];
xdp_A1_c        = double(state_calc(x0,parms));
fprintf('\nThe result for A1 - c is:\n');
disp(table(variable,xdp_A1_c));

% A1 - d
w               = (60/60)*2*pi;             % Convert to rad/s
x0              = [0 0 w w];
xdp_A1_d        = double(state_calc(x0,parms));
fprintf('\nThe result for A1 - d is:\n');
disp(table(variable,xdp_A1_d));

% Calculate velocities
x_p_A1_d        = [-(parms.L/2)*sin(x0(1))*x0(3); ...
                    (parms.L/2)*cos(x0(1))*x0(3); ...
                   -(parms.L/2)*sin(x0(1))*x0(3) - (parms.L/2)*sin(x0(1))*x0(3) -
(parms.L/2)*sin(x0(2))*x0(4); ...
                    (parms.L/2)*cos(x0(1))*x0(3) + (parms.L/2)*cos(x0(1))*x0(3) +
(parms.L/2)*cos(x0(2))*x0(4)];

               fprintf('The accompaning velocities are:\n');
disp(table({'x1p';'y1p';'x2p';'y2p'},x_p_A1_d));

%% Express COM in generalised coordinates
function xdp = state_calc(x0,parms)
syms phi1 phi2 phi1p phi2p

% Create generalized coordinate vectors
q               = [phi1; phi2];
qp              = [phi1p; phi2p];

% COM of the bodies expressed in generalised coordinates
x1              = (parms.L/2)*cos(phi1);
y1              = (parms.L/2)*sin(phi1);
x2              = x1 + (parms.L/2)*cos(phi1) + (parms.L/2) * cos(phi2);
y2              = y1 + (parms.L/2)*sin(phi1) + (parms.L/2) * sin(phi2);

% Calculate derivative of COM expressed in generalised coordinates (We need this for the energy
equation)
```

```
x               = [x1;y1;phi1;x2;y2;phi2];
Jx_q            = simplify(jacobian(x,q));
xp              = Jx_q*qp;

%% Compute energies
T               = 0.5*xp.'*diag([parms.m;parms.m;parms.I;parms.m;parms.m;parms.I])*xp;          %
Kinetic energy
V               = -([parms.m*parms.g 0 0 parms.m*parms.g 0 0]*x);                    % Potential
energy

%% Calculate the terms of the jacobian
Q               = 0;                            % Non-conservative forces

% Partial derivatives of Kinetic energy
T_q             = simplify(jacobian(T,q));
T_qp            = simplify(jacobian(T,qp));
T_qpqp          = simplify(jacobian(T_qp,qp));
T_qpq           = simplify(jacobian(T_qp,q));

% Partial derivatives of Potential energy
V_q             = simplify(jacobian(V,q));
V_qp            = simplify(jacobian(V,qp));
V_qpqp          = simplify(jacobian(V_qp,qp));

% Make matrix vector product
M               = T_qpqp;
F               = Q + T_q' - V_q' - T_qpq*qp;

% Solve Mqdp=F to get the accelerations
qdp             = inv(M)*F;

%% Get back to COM coordinates
xdp             = simplify(jacobian(xp,qp))*qdp+simplify(jacobian(xp,q))*qp;
xdp             = subs(xdp,{phi1 phi2 phi1p phi2p},{x0(1) x0(2) x0(3) x0(4)});

end
```

Appendix B – Question 2 e-f

In this appendix you will find the code in which the new Lagrange constraint equations were used to redo assignment 2 e-f. In assignment 2 e-f there was slider constraint that moved on a vertical line through the organ.

MATLAB CODE

```
%% MBD_B: Assignment 4 - Double pendulum systemetic approach
%  Rick Staa (4511328)
%  Last edit: 19/03/2018
% Question A: Redoo assignment 1

clear all; close all; clc;
fprintf('--- A4_a ---\n');
fprintf('First lets redo A1 - In this case there are no constraints\n')

%% Script settings and parameters
variable            = {'x1dp' 'y1dp' 'phi1dp' 'x2dp' 'y2dp' 'phi2dp'}';

%% Parameters
% Segment 1
parms.L             = 0.55;                                         % [parms.m]
parms.w             = 0.05;                                         % [parms.m]
parms.t             = 0.004;                                        % [parms.m]
parms.p             = 1180;                                         % [kg/parms.m^3]
parms.m             = parms.p * parms.w * parms.t * parms.L;        % [kg]
parms.I             = (1/12) * parms.m * parms.L^2;                 % [kg*parms.m^2]

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

%% Calculate state_dp for initial states
% A2 - a:
% A1 - e
x0              = [0.5*pi 0.5*pi 0 0];
xdp_A1_e        = double(state_calc(x0,parms));
fprintf('\nThe result for A1 - e is:\n');
disp(table(variable,xdp_A1_e));
```

```
% A1 - f
w               = (60/60)*2*pi;             % Convert to rad/s
x0              = [0.5*pi 0.5*pi w 0];
xdp_A1_f        = double(state_calc(x0,parms));
fprintf('\nThe result for A1 - f is:\n');
disp(table(variable,xdp_A1_f));

%% Express COM in generalised coordinates
function xdp = state_calc(x0,parms)
syms phi1 phi2 phi1p phi2p

% Create generalized coordinate vectors
phi2            = pi - phi1;                                % Add extra constraint
q               = [phi1];
qp              = [phi1p];

% COM of the bodies expressed in generalised coordinates
x1              = (parms.L/2)*cos(phi1);
y1              = (parms.L/2)*sin(phi1);
x2              = x1 + (parms.L/2)*cos(phi1) + (parms.L/2) * cos(phi2);
y2              = y1 + (parms.L/2)*sin(phi1) + (parms.L/2) * sin(phi2);

% Calculate derivative of COM expressed in generalised coordinates (We need this for the energy
equation)
x               = [x1;y1;phi1;x2;y2;phi2];
Jx_q            = simplify(jacobian(x,q));
xp              = Jx_q*qp;

%% Compute energies
T               = 0.5*xp.'*diag([parms.m;parms.m;parms.I;parms.m;parms.m;parms.I])*xp;          %
Kinetic energy
V               = -([parms.m*parms.g 0 0 parms.m*parms.g 0 0]*x);                   % Potential
energy

%% Calculate the terms of the jacobian
Q                = 0;                           % Non-conservative forces

% Partial derivatives of Kinetic energy
T_q             = simplify(jacobian(T,q));
T_qp            = simplify(jacobian(T,qp));
T_qpqp          = simplify(jacobian(T_qp,qp));
T_qpq           = simplify(jacobian(T_qp,q));

% Partial derivatives of Potential energy
V_q             = simplify(jacobian(V,q));
V_qp            = simplify(jacobian(V,qp));
V_qpqp          = simplify(jacobian(V_qp,qp));

% Make matrix vector product
M               = T_qpqp;
F               = Q + T_q' - V_q' - T_qpq*qp;

% Solve Mqdp=F to get the accelerations
qdp             = inv(M)*F;

%% Get back to COM coordinates
xdp             = simplify(jacobian(xp,qp))*qdp+simplify(jacobian(xp,q))*qp;
xdp             = subs(xdp,{phi1 phi2 phi1p phi2p},{x0(1) x0(2) x0(3) x0(4)});

end
```

Appendix C – Question 2 g

In this appendix you will find the code in which the new Lagrange constraint equations were used to redo assignment 2 g. In assignment 2 g there was slider constraint that moved on a vertical line through the organ and additionally also a vertical force was applied to the point B.

MATLAB CODE

```
%% MBD_B: Assignment 4 - Double pendulum systemetic approach
%  Rick Staa (4511328)
%  Last edit: 19/03/2018
% Question A: Redoo assignment 1

% clear all; close all; clc;
fprintf('--- A4_a ---\n');
```

```
fprintf('First lets redo A2 - In this case there are no constraints\n')

%% Script settings and parameters
variable              = {'x1dp' 'y1dp' 'phi1dp' 'x2dp' 'y2dp' 'phi2dp'}';

%% Parameters
% Segment 1
parms.L               = 0.55;                                          % [parms.m]
parms.w               = 0.05;                                          % [parms.m]
parms.t               = 0.004;                                         % [parms.m]
parms.p               = 1180;                                          % [kg/parms.m^3]
parms.m               = parms.p * parms.w * parms.t * parms.L;         % [kg]
parms.I               = (1/12) * parms.m * parms.L^2;                  % [kg*parms.m^2]

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

%% Calculate state_dp for initial states
% A2 - a:
% A1 - g
x0              = [0 pi 0 0];
xdp_A1_e        = double(state_calc(x0,parms));
fprintf('\nThe result for A2 - g is:\n');
disp(table(variable,xdp_A1_e));


%% Express COM in generalised coordinates
function xdp = state_calc(x0,parms)
syms phi1 phi2 phi1p phi2p

% Create generalized coordinate vectors
phi2            = pi - phi1;                              % Add extra constraint
q               = [phi1];
qp              = [phi1p];

% COM of the bodies expressed in generalised coordinates
x1              = (parms.L/2)*cos(phi1);
y1              = (parms.L/2)*sin(phi1);
x2              = x1 + (parms.L/2)*cos(phi1) + (parms.L/2) * cos(phi2);
y2              = y1 + (parms.L/2)*sin(phi1) + (parms.L/2) * sin(phi2);

% Calculate derivative of COM expressed in generalised coordinates (We need this for the energy
equation)
x               = [x1;y1;phi1;x2;y2;phi2];
Jx_q            = simplify(jacobian(x,q));
xp              = Jx_q*qp;

%% Compute energies
T               = 0.5*xp.'*diag([parms.m;parms.m;parms.I;parms.m;parms.m;parms.I])*xp;           %
Kinetic energy
V               = -([parms.m*parms.g 0 0 parms.m*parms.g 0 0]*x);                    % Potential
energy

%% Calculate the terms of the jacobian

% Partial derivatives of Kinetic energy
T_q             = simplify(jacobian(T,q));
T_qp            = simplify(jacobian(T,qp));
T_qpqp          = simplify(jacobian(T_qp,qp));
T_qpq           = simplify(jacobian(T_qp,q));

% Partial derivatives of Potential energy
V_q             = simplify(jacobian(V,q));
V_qp            = simplify(jacobian(V,qp));
V_qpqp          = simplify(jacobian(V_qp,qp));

% Non-conservative forces
Q               = Jx_q.'*[0 10 10*(parms.L/2)*cos(phi1) 0 0 0].';

% Make matrix vector product
M               = T_qpqp;
F               = Q + T_q' - V_q' - T_qpq*qp;

% Solve Mqdp=F to get the accelerations
qdp             = inv(M)*F;

%% Get back to COM coordinates
xdp             = simplify(jacobian(xp,qp))*qdp+simplify(jacobian(xp,q))*qp;
xdp             = subs(xdp,{phi1 phi2 phi1p phi2p},{x0(1) x0(2) x0(3) x0(4)});

end
```

Appendix D – Question 3 a

In this appendix you will find the code in which the new Lagrange constraint equations were used to redo assignment 3 a. In assignment 3 a passive spring was added to the system.

MATLAB CODE

```
%% ME41055 - Multibody Dynamics B - HW Set 4
% Submitted by Prajish Sekoor Lakshmana Sankar #4743873
% Due 20th March, 2018

% Redoing HW 3 - a
% clc; clear all;

%% Initialize variables

l = 0.55;    % length of bar
w = 0.05;    % width of bar
t = 0.004;   % thickness of bar
rho = 1180;  % density of bar material
g = 9.81;    % gravitational acceleration

m = rho*l*w*t; % mass of each bar
I = (1/12)*m*(l^2 + w^2); % mass moment of inertia of each bar about its CM

k = (15/2)*m*g/l; % stiffnedd of spring

%% Compute the derivatives of constraint matrix

% Use symbolic toolbox
syms phi1 phi2 dphi1 dphi2

% Defining the genaralized coordinates
q = [phi1 phi2]';
dq = [dphi1 dphi2]';

% Expressing coordinates of CM using generalized coordinates
% Here, x = [x1 y1 phi1 x2 y2 phi2]';
x = [0.5*l*cos(phi1);...
     0.5*l*sin(phi1);...
     phi1;...
     l*cos(phi1)+0.5*l*cos(phi2);...
     l*sin(phi1)+0.5*l*sin(phi2);...
     phi2];

Q = simplify(jacobian(x,q')); % Computing the jacobian

dx = Q*dq; % This is dx/dt=dx/dq*(dq)
%% Computing energies

T = 0.5*dx'*diag([m,m,I,m,m,I])*dx;

V = -([m*g 0 0 m*g 0 0]*x);

Cs = sqrt((0.5*l*cos(phi1) + (l/6)*cos(phi1) + l/2)^2 + (0.5*l*sin(phi1) ...
    + (l/6)*sin(phi1))^2) - 2*l/3;
Vs = 0.5*k*(Cs)^2; % The potential energy due to the spring

V = V + Vs; % Total potential energy
%% Getting the terms of the equations of motion

T1 = simplify(jacobian(T,dq'))';

T1_1 = simplify(jacobian(T1,dq')); % coefficient of the ddq term.
T1_2 = simplify(jacobian(T1,q'))*dq; % convective terms

T2 = simplify(jacobian(T,q'))';

T3 = simplify(jacobian(V,q'))';

ddq = inv(T1_1)*(-T1_2 + T2 - T3 );

%% Initial Conditions

phi1_init = pi/2;
phi2_init = pi/2;
dphi1_init = 0;
dphi2_init = 0;
```

```
%% Converting back to CM coordinates

ddx = simplify(jacobian(dx,dq'))*ddq + simplify(jacobian(dx,q'))*dq;

ddx = double(subs(ddx, [phi1,phi2,dphi1,dphi2],...
   [phi1_init,phi2_init,dphi1_init,dphi2_init]))

%% END %%
```

Appendix E – Question 3 b

In this appendix you will find the code in which the new Lagrange constraint equations were used to redo assignment 3 b. In assignment 3 b active motor was added to the system.

MATLAB CODE

```
%% MBD_B: Assignment 4 - Double pendulum systemetic approach
%  Rick Staa (4511328)
%  Last edit: 19/03/2018
% Question A: Redoo assignment 1

% clear all; close all; clc;
fprintf('--- A4_a ---\n');
fprintf('First lets redo A3 - In this case there are no constraints\n')

%% Script settings and parameters
variable              = {'x1dp' 'y1dp' 'phi1dp' 'x2dp' 'y2dp' 'phi2dp'}';

%% Parameters
% Segment 1
parms.L               = 0.55;                                          % [parms.m]
parms.w               = 0.05;                                          % [parms.m]
parms.t               = 0.004;                                         % [parms.m]
parms.p               = 1180;                                          % [kg/parms.m^3]
parms.m               = parms.p * parms.w * parms.t * parms.L;         % [kg]
parms.I               = (1/12) * parms.m * parms.L^2;                  % [kg*parms.m^2]
parms.omega           = -120*(2*pi/60);

% World parameters
parms.g               = 9.81;                                          % [parms.m/s^2]
parms.omega           =-120*(2*pi/60);

%% Calculate the accelerations

x0(1) = 0.5*pi;
x0(2) = 0.5*pi;
x0(3) = parms.omega;
x0(4) = parms.omega;

[dxx,Motor_torque] = state_calc(x0,parms);

%% Compute the derivatives of constraint matrix
function [xdp, Motor_torque] = state_calc(x0,parms)
% Create symbolic variables
syms phi1 phi2 phi1p phi2p t

% Define genaralized coordinates
q = [phi1 phi2]';
qp = [phi1p phi2p]';

% Ex[ress coordinatates of CM in generalised coordinates
x1              = (parms.L/2)*cos(phi1);
y1              = (parms.L/2)*sin(phi1);
x2              = x1 + (parms.L/2)*cos(phi1) + (parms.L/2) * cos(phi2);
y2              = y1 + (parms.L/2)*sin(phi1) + (parms.L/2) * sin(phi2);
x               = [x1;y1;phi1;x2;y2;phi2];
Xj_q = simplify(jacobian(x,q'));
xp = Xj_q*qp;

%% Constraints
C = phi1 - parms.omega*t;
Cq = simplify(jacobian(C,q'));
Cp = Cq*qp;
Cdp = simplify(jacobian(Cp,q')*qp);
```

```
%% Calculate potential en kinetic energy
T = 0.5*xp'*diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I])*xp;
V = -([parms.m*parms.g 0 0 parms.m*parms.g 0 0]*xp);

%% Building the Lagrance equations of motion
Tqp = simplify(jacobian(T,qp'))';
Tqpqp = simplify(jacobian(Tqp,qp'));
Tqpq = simplify(jacobian(Tqp,q'))*qp;
Tq = simplify(jacobian(T,q'))';
Vq = simplify(jacobian(V,q'))';
M = [Tqpqp Cq';Cq 0];
F = [-Tqpq + Tq - Vq; -Cdp];

% Calculate acceleration
qdp = inv(M)*(F);
qdp = (subs(qdp, [phi1,phi1p,phi2,phi2p],[x0(1),x0(2),x0(3),x0(4)]));

%% Express back in  CM coordinates
xdp = simplify(jacobian(xp,qp'))*qdp(1:2) + simplify(jacobian(xp,q'))*qp;
Motor_torque = double(qdp(3));
xdp = double(subs(xdp, [phi1,phi2,phi1p,phi2p],[x0(1),x0(2),x0(3),x0(4)]));
end
```

Appendix E – Question 3 C

In this appendix you will find the code in which the new Lagrange constraint equations were used to redo assignment 3 c. In assignment 3 c Impact equations were added to the system.

```
%% MBD_B: Assignment 4 - Double pendulum systemetic approach
%  Rick Staa (4511328)
%  Last edit: 19/03/2018
% Question A: Redoo assignment 1

% clear all; close all; clc;
fprintf('--- A4_a ---\n');
fprintf('First lets redo A3 - In this case there is an impact constrant\n')

%% Script settings and parameters
variable              = {'x1dp' 'y1dp' 'phi1dp' 'x2dp' 'y2dp' 'phi2dp'}';

%% Parameters
% Segment 1
parms.L               = 0.55;                                          % [parms.m]
parms.w               = 0.05;                                          % [parms.m]
parms.t               = 0.004;                                         % [parms.m]
parms.p               = 1180;                                          % [kg/parms.m^3]
parms.m               = parms.p * parms.w * parms.t * parms.L;         % [kg]
parms.I               = (1/12) * parms.m * parms.L^2;                  % [kg*parms.m^2]
parms.omega           = -120*(2*pi/60);

% World parameters
parms.g               = 9.81;                                          % [parms.m/s^2]
parms.omega           = -120*(2*pi/60);
parms.e               = 0;

%% Calculate the accelerations
x0(1) = 0.5*pi;
x0(2) = 0.5*pi;
x0(3) = parms.omega;
x0(4) = parms.omega;

[X_g] = state_calc(x0,parms);

%% Compute the derivatives of constraint matrix
function [X_g] = state_calc(x0,parms)
% Create symbolic variables
syms phi1 phi2 phi1p phi2p t

% Define genaralized coordinates
q = [phi1 phi2]';
qp = [phi1p phi2p]';

% Ex[ress coordinatates of CM in generalised coordinates
x1              = (parms.L/2)*cos(phi1);
y1              = (parms.L/2)*sin(phi1);
x2              = x1 + (parms.L/2)*cos(phi1) + (parms.L/2) * cos(phi2);
y2              = y1 + (parms.L/2)*sin(phi1) + (parms.L/2) * sin(phi2);
```

```
x               = [x1;y1;phi1;x2;y2;phi2];
Xj_q            = simplify(jacobian(x,q'));
xp              = Xj_q*qp;

%% Set constraints
C               = [parms.L*cos(phi1);parms.L*cos(phi1)+parms.L*cos(phi2);];
Cq              = simplify(jacobian(C,q'));

%% Define Matrices
M               = Xj_q.'*diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I])*Xj_q;        %
Reduced generalised M matrix
M_big           = [M Cq.';Cq zeros(2,2)];                   % Full M matrix
F               = [M*x0(3:4).'; -parms.e*Cq*x0(3:4).'];      % Impact Forces
X_g             = inv(M_big)*F;
X_g             = double(subs(X_g, [phi1,phi2,phi1p,phi2p],[x0(1),x0(2),x0(3),x0(4)]))
X_g             = simplify(jacobian(x,q'))*X_g(1:2);
X_g             = double(subs(X_g, [phi1,phi2,phi1p,phi2p],[x0(1),x0(2),x0(3),x0(4)])) % Impact
velocities

end
```