|  |  |
| --- | --- |
| I used [1] in making this assignment when finished I compared initial values with Prajish Kumar (4743873). | **Rick Staa, #4511328**  **HW set 3 due 13/03/2018**  **ME41060** |

# Statement of entregity



# Problem Statement

In this assignment, we were asked to add passive elements, active elements and an impulsive contact to the model we created in assignment 1 and 2. The mass, length and inertia parameters are still those depicted in assignment 1. Further the initial state is still Only the state changes depending on the type of constraints.

## Addition of a passive element

We were asked to add a spring (passive element) to the model that was 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 (Figure 1: Double pendulum with spring). The free length of this spring was said to be and its linear stiffness k = (15/2)(mg/l). We were following asked to calculate the accelerations of the center of mass of the two bodies together with the LaGrangian multipliers for the case when the two bars are at .

A picture containing map

Description generated with high confidence

Figure 1: Double pendulum with spring

## Adding the passive element

The addition of a passive element is very straight forward since most of the ingredients have already been calculated in the previous assignment. The passive (and active) elements can be added to the virtual work equations in a similar way as we would add the regular constrains. The virtual power for a passive element is:

|  |  |  |
| --- | --- | --- |
|  |  | **(1)** |

Together with the regular constraints derived in assignment 2 the full virtual power equation now becomes:

|  |  |  |
| --- | --- | --- |
|  |  | **(2)** |

If we rewrite this virtual power equation in matrix vector form we get the following system:

|  |  |  |
| --- | --- | --- |
|  |  | **(3)** |

Except and all the terms have already been calculated in assignment 2 and 1. From these two depicts the constitutive behaviour or the linear elastic spring force and the transformations from the element forces to the forces at the cm coordinates. As a result, can be obtained by taking the elongation of the spring as an element constraint. This constraint needs to be expressed in the coordinates of the cm’s of the body and differentiated to get it in the right form. The spring constraint expressed can be derived from the elongation of the spring:

|  |  |  |
| --- | --- | --- |
|  |  | **(4)** |

In this is the spring length and is the rest length which was said to be . The spring constrained expressed in the Cm’s of the body then becomes:

By differentiating w.r.t. the state variables we get the following constraint equations.

|  |  |  |
| --- | --- | --- |
|  |  | **(5)** |

The constitutive relationship is as follows:

|  |  |  |
| --- | --- | --- |
|  |  | **(6)** |

We now have all the ingredients to calculate the accelerations for the case when we have a passive element. The MATLAB code implementing this calculation can be found in Appendix A – Passive element

## Calculate accelerations

With we get the following result:

|  |  |
| --- | --- |
|  | 2.1021 |
|  | 0.00 |
|  | -7.6442 |
|  |  |
|  | 0.00 |
|  | -15.2883 |
|  | 0.2274 N |
|  | -1.2733 N |
|  | 0.1819 N |
|  | 0.00 N |

Since this situation is equal to the situation in Assignment 2 and we know the relationship between LaGrange multipliers we can compare the results. The biggest difference apart from scaling effects is element 1 now also has a vertical force component added to it. This is expected as we look at the attachment place of the spring.

# Adding active element

We are now asked to add a motor to the pendulum this setup is shown in Figure 2: Double pendulum with motor.

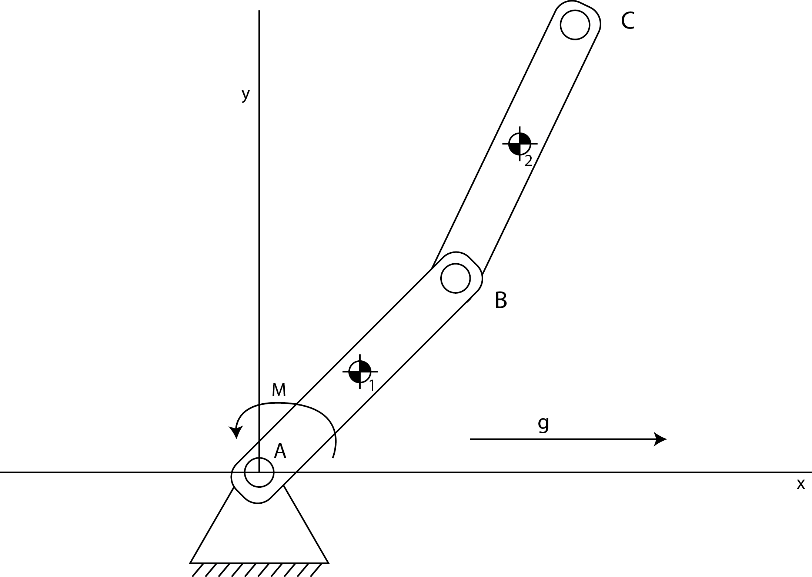


Figure 2: Double pendulum with motor

The active element (motor force) can be added to the already in assignment 1 and 2 derived model equations in a similar way as this was done for the passive element. To do this we now add the following additional constraint to the constraint vector .

|  |  |  |
| --- | --- | --- |
|  |  | **(7)** |

In this depicts the given constant speed of the motor and t the time. Contrary to the passive element in which we knew the spring force at every position for the motor we do not know the motor torque at each position. Since this motor torque can be dependent on a lot of factors (motor characteristics, position, velocity, acceleration, temperature and time) part of the motor virtual power ends up at the left side of the equation. We can therefore add the motor constraint to the kinematic constraint vector.

|  |  |  |
| --- | --- | --- |
|  |  | **(8)** |

Since we now also need to take the full derivative (both with respect to the state and the time) of this constraint vector we now also and up with two new convective terms and . As a result we get the following system of equations:

|  |  |  |
| --- | --- | --- |
|  |  | **(9)** |

The and were derived by MATLAB symbolic toolbox and will since they are not done by hand not be depicted here. Because of the motor force we now get a fifth Laplace multiplier. The MATLAB code for doing this can be found in %% MBD\_B: Assignment 3

% Question 1 - Passive Element

% Rick Staa (4511328)

% Last edit: 05/03/2018

clear all; close all; % clc;

%% Parameters

% Segment 1

parms.L = 0.55; % [m]

parms.w = 0.05; % [m]

parms.t = 0.004; % [m]

parms.p = 1180; % [kg/m^3]

parms.m = parms.p \* parms.w \* parms.t \* parms.L; % [kg]

parms.I = (1/12) \* parms.m \* parms.L^2; % [kg\*m^2]

% World parameters

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

% Spring parameters

parms.k = (15/2)\*parms.m\*parms.g/parms.L; % stiffness of spring

%% Compute constraint matrices

% Use symbolic toolbox to calculate derivatives

syms x1 y1 phi1 x2 y2 phi2

syms dx1 dy1 dphi1 dx2 dy2 dphi2

x = [x1 y1 phi1 x2 y2 phi2];

dx = [dx1 dy1 dphi1 dx2 dy2 dphi2];

parms.x = x;

parms.dx = dx;

% The normal constrain equations

C = [x1-(parms.L/2)\*cos(phi1); y1-(parms.L/2)\*sin(phi1) ; ...

(x2-(parms.L/2)\*cos(phi2))-(x1+(parms.L/2)\*cos(phi1)); ...

(y2-(parms.L/2)\*sin(phi2))-(y1+(parms.L/2)\*sin(phi1))];

% Calculate the jacobian to create the constraint equations

Cx = jacobian(C,x);

Cx = simplify(Cx);

% Take the second derivative to create the gluing constraints

Cd = Cx\*dx';

Cdd = jacobian(Cd,x)\*dx';

Cdd = simplify(Cdd);

% Now also calculate the spring constraint terms

Cs = sqrt((x1 + (parms.L/6)\*cos(phi1) + parms.L/2)^2 + (y1 + (parms.L/6)\*sin(phi1))^2) - 2\*(parms.L/3);

Csx = jacobian(Cs,x); % Jacobian of Cs

Csx = simplify(Csx)';

% Put Csx Cdp Cd cx in parms struct and feed tem into the solver

syst.Cx = Cx;

syst.Cd = Cd;

syst.Cdd = Cdd;

syst.Cs = Cs;

syst.Csx = Csx;

%% A:

x0 = [0.5\*pi 0.5\*pi 0 0];

[xdd] = state\_calc(x0,parms,syst);

xdd = double(vpa(xdd));

%% A: Create state space matrices for the case when we add a spring

function [xdd] = state\_calc(x0,parms,syst)

% Get system matrices out

structname\_fields = fields(parms);

for i = 1:size(fields(parms))

eval\_str = [structname\_fields{i,:},'=','parms.',structname\_fields{i,:},';'];

eval(eval\_str);

end

structname\_fields = fields(syst);

for i = 1:size(fields(syst))

eval\_str = [structname\_fields{i,:},'=','syst.',structname\_fields{i,:},';'];

eval(eval\_str);

end

% Calculate missing initial states

x1 = 0.5\*parms.L\*cos(x0(1));

y1 = 0.5\*parms.L\*sin(x0(2));

x1 = 0;

y1 = parms.L/2;

Cx = subs(Cx, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cdd = subs(Cdd, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cs = subs(Cs, {'phi1','phi2','x1','y1'},...

[x0(1),x0(2),x1,y1]);

Csx = subs(Csx, {'phi1','phi2','x1','y1'},...

[x0(1),x0(2),x1,y1]);

sigma = parms.k\*Cs; % Spring Force

% Create matrices

M = diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I]);

A = [M Cx';Cx zeros(4,4)];

F = [parms.m\*parms.g 0 0 parms.m\*parms.g 0 0]' - Csx\*sigma; % Updated F vector

b = [F;-Cdd];

xdd = A\b;

end

Appendix B.

## Calculate accelerations

With we get the following result:

|  |  |
| --- | --- |
|  | 0.00 |
|  | -43.4263 |
|  | 0.00 |
|  | 7.36 |
|  | -130.28 |
|  | -26.75 |
|  | 1.5917 N |
|  | 22.5469 N |
|  | 0.3183 N |
|  | 16.9102 N |
|  | -0.5253 N |

Since = -0.52 Nm in clockwise direction the motor power supplied to the mechanical system is equal to:

|  |  |  |
| --- | --- | --- |
|  |  | **(10)** |

# Adding an impulsive contact

In this part of the question we were asked to calculate the impulsive forces for the case when the bars hit the wall when they are vertically up. This was said to occur with an angular speed of in the counter clockwise direction. The constraint equations for the impact position are as follows:

|  |  |  |
| --- | --- | --- |
|  |  | **(11)** |

|  |  |  |
| --- | --- | --- |
|  |  | **(12)** |

By differentiating these constraints and filling them in in virtual power equation we get the following result:

|  |  |  |
| --- | --- | --- |
|  |  | **(13)** |

Next to get the impulsive forces we need to look at the contacts during a very short impact time. We do this by integrating over the time and then taking the limit from .

|  |  |  |
| --- | --- | --- |
|  |  | **(14)** |

Since the term is equal to and the to this results in the following:

|  |  |  |
| --- | --- | --- |
|  |  | **(15)** |

After some reordering we get the following matrix vector product:

|  |  |  |
| --- | --- | --- |
|  |  | **(16)** |

In which the e term represents the coefficient of restitution and is given as:

|  |  |  |
| --- | --- | --- |
|  |  | **(17)** |

Using this and our constraint velocity equation we get:

|  |  |  |
| --- | --- | --- |
|  |  | **(18)** |

## Calculate and post impact velocities, contact impulses, kinetic energy and work.

These calculations need to be done at the earlier names initial state . The velocities, contact impulses, kinetic energy and work are calculate in the MATLAB code in %% MBD\_B: Assignment 3

% Question 2 - Active Element

% Rick Staa (4511328)

% Last edit: 05/03/2018

clear all; close all; % clc;

%% Parameters

% Segment 1

parms.L = 0.55; % [m]

parms.w = 0.05; % [m]

parms.t = 0.004; % [m]

parms.p = 1180; % [kg/m^3]

parms.m = parms.p \* parms.w \* parms.t \* parms.L; % [kg]

parms.I = (1/12) \* parms.m \* parms.L^2; % [kg\*m^2]

% World parameters

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

% Spring parameters

parms.k = (15/2)\*parms.m\*parms.g/parms.L; % stiffness of spring

% Motor constraint

omega = -120\*(2\*pi/60); % motor speed

%% Compute constraint matrices

% Use symbolic toolbox to calculate derivatives

syms x1 y1 phi1 x2 y2 phi2 t

syms dx1 dy1 dphi1 dx2 dy2 dphi2

x = [x1 y1 phi1 x2 y2 phi2];

xd = [dx1 dy1 dphi1 dx2 dy2 dphi2];

parms.x = x;

parms.xd = xd;

% The normal constrain equations

C = [x1-(parms.L/2)\*cos(phi1); y1-(parms.L/2)\*sin(phi1) ; ...

(x2-(parms.L/2)\*cos(phi2))-(x1+(parms.L/2)\*cos(phi1)); ...

(y2-(parms.L/2)\*sin(phi2))-(y1+(parms.L/2)\*sin(phi1)); ...

(phi1 - omega\*t)]; % Add extra motor constraint

% Calculate the jacobian to create the constraint equations

Cx = jacobian(C,x);

Cx = simplify(Cx);

% Constraint derivative with respect to time

Ct = simplify(jacobian(C,t)); % First derivative to time

Ctt = simplify(jacobian(Ct,t)); % double derivative to time

% Take the second derivative to create the gluing constraints

Cd = Cx\*xd';

Cdd = jacobian(Cd,x)\*xd';

Cdd = simplify(Cdd); % Convective term constraints

Cdt = simplify(jacobian(Cd,t)); % Convective to time

% Put Csx Cdp Cd cx in parms struct and feed tem into the solver

syst.Cx = Cx;

syst.Cd = Cd;

syst.Cdd = Cdd;

syst.Ct = Ct;

syst.Ctt = Ctt;

syst.Cdt = Cdt;

%% A:

x0 = [0.5\*pi 0.5\*pi omega omega];

[xdd] = state\_calc(x0,parms,syst);

xdd = double(vpa(xdd));

%% A: Create state space matrices for the case when we add a spring

function [xdd] = state\_calc(x0,parms,syst)

% Get system parameters out

structname\_fields = fields(parms);

for i = 1:size(fields(parms))

eval\_str = [structname\_fields{i,:},'=','parms.',structname\_fields{i,:},';'];

eval(eval\_str);

end

% Get symbolic expressions out

structname\_fields = fields(syst);

for i = 1:size(fields(syst))

eval\_str = [structname\_fields{i,:},'=','syst.',structname\_fields{i,:},';'];

eval(eval\_str);

end

Cx = subs(Cx, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cdd = subs(Cdd, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Ct = subs(Ct, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cdt = subs(Cdt, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

% Create matrices

M = diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I]);

A = [M Cx';Cx zeros(5,5)];

F = [parms.m\*parms.g 0 0 parms.m\*parms.g 0 0]';

b = [F;-(Cdd+Cdt+Ctt)];

xdd = A\b;

end

Appendix C – Impulsive impact force. The kinetic energy before impact, after impact and the work done by the contact impulses were calculated as successively:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **(19)** | |
|  |  | **(20)** |
|  |  | **(21)** |

### e = 1

|  |  |
| --- | --- |
|  | 3.46 |
|  | 0.00 |
|  | -12.57 |
|  | 10.37 |
|  | 0.00 |
|  | -12.57 |
|  | -0.30 N |
|  | 0.00N |
|  | -1.20 N |
|  | 0.00 N |
|  | -1.80 N |
|  | -1.50 N |
|  | 8.27 J |
|  | 8.27 J |
|  | 0.00 J |

### e = 0.5

|  |  |
| --- | --- |
|  | 1.73 |
|  | 0.00 |
|  |  |
|  | 5.19 |
|  | 0.00 |
|  | -6.28 |
|  | -0.22 N |
|  | 0.00N |
|  | -0.90 N |
|  | 0.00 N |
|  | -1.35 N |
|  | -1.12 N |
|  | 8.27 J |
|  | 2.07 J |
|  | -6.15 J |

### e = 0

|  |  |
| --- | --- |
|  | 0.00 |
|  | 0.00 |
|  | 0.00 |
|  | 0.00 |
|  | 0.00 |
|  | 0.00 |
|  | -0.15 N |
|  | 0.00N |
|  | -0.60 N |
|  | 0.00 N |
|  | -0.90 N |
|  | -0.75 N |
|  | 8.27 J |
|  | 0.00 J |
|  | -8.20 J |

## Discussion

From this we see that for a fully elastic collision no work is done by the contact impulses, so no energy is lost and that in the case of a fully plastic collision all energy is converted into work and no energy is left.

# Unknown Velocity jumps instead of unknown velocities after impact

When we use the velocity jumps as the unknowns the following things change in the impulsive equations:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **(22)** | |
|  |  | **(23)** |
|  |  | **(24)** |
|  |  | **(25)** |
|  |  | **(26)** |
|  |  | **(27)** |

These equations will result in the following model equations:

|  |  |  |
| --- | --- | --- |
|  |  | **(28)** |

Advantage)

Mass matrix now only on one side of the Vector Matrix Product.

Disadvantage:

You still need to compute the velocities after impact since you now are only looking at the difference.

# References

[1] A. L. Schwab, “Virtual power and Lagrance multipliers,” in *Multibody Dynamics*, Delft, The Netherlands: TU Delft, 2018.

# Appendix A – Passive element

%% MBD\_B: Assignment 3

% Question 1 - Passive Element

% Rick Staa (4511328)

% Last edit: 05/03/2018

clear all; close all; % clc;

%% Parameters

% Segment 1

parms.L = 0.55; % [m]

parms.w = 0.05; % [m]

parms.t = 0.004; % [m]

parms.p = 1180; % [kg/m^3]

parms.m = parms.p \* parms.w \* parms.t \* parms.L; % [kg]

parms.I = (1/12) \* parms.m \* parms.L^2; % [kg\*m^2]

% World parameters

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

% Spring parameters

parms.k = (15/2)\*parms.m\*parms.g/parms.L; % stiffness of spring

%% Compute constraint matrices

% Use symbolic toolbox to calculate derivatives

syms x1 y1 phi1 x2 y2 phi2

syms dx1 dy1 dphi1 dx2 dy2 dphi2

x = [x1 y1 phi1 x2 y2 phi2];

dx = [dx1 dy1 dphi1 dx2 dy2 dphi2];

parms.x = x;

parms.dx = dx;

% The normal constrain equations

C = [x1-(parms.L/2)\*cos(phi1); y1-(parms.L/2)\*sin(phi1) ; ...

(x2-(parms.L/2)\*cos(phi2))-(x1+(parms.L/2)\*cos(phi1)); ...

(y2-(parms.L/2)\*sin(phi2))-(y1+(parms.L/2)\*sin(phi1))];

% Calculate the jacobian to create the constraint equations

Cx = jacobian(C,x);

Cx = simplify(Cx);

% Take the second derivative to create the gluing constraints

Cd = Cx\*dx';

Cdd = jacobian(Cd,x)\*dx';

Cdd = simplify(Cdd);

% Now also calculate the spring constraint terms

Cs = sqrt((x1 + (parms.L/6)\*cos(phi1) + parms.L/2)^2 + (y1 + (parms.L/6)\*sin(phi1))^2) - 2\*(parms.L/3);

Csx = jacobian(Cs,x); % Jacobian of Cs

Csx = simplify(Csx)';

% Put Csx Cdp Cd cx in parms struct and feed tem into the solver

syst.Cx = Cx;

syst.Cd = Cd;

syst.Cdd = Cdd;

syst.Cs = Cs;

syst.Csx = Csx;

%% A:

x0 = [0.5\*pi 0.5\*pi 0 0];

[xdd] = state\_calc(x0,parms,syst);

xdd = double(vpa(xdd));

%% A: Create state space matrices for the case when we add a spring

function [xdd] = state\_calc(x0,parms,syst)

% Get system matrices out

structname\_fields = fields(parms);

for i = 1:size(fields(parms))

eval\_str = [structname\_fields{i,:},'=','parms.',structname\_fields{i,:},';'];

eval(eval\_str);

end

structname\_fields = fields(syst);

for i = 1:size(fields(syst))

eval\_str = [structname\_fields{i,:},'=','syst.',structname\_fields{i,:},';'];

eval(eval\_str);

end

% Calculate missing initial states

x1 = 0.5\*parms.L\*cos(x0(1));

y1 = 0.5\*parms.L\*sin(x0(2));

x1 = 0;

y1 = parms.L/2;

Cx = subs(Cx, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cdd = subs(Cdd, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cs = subs(Cs, {'phi1','phi2','x1','y1'},...

[x0(1),x0(2),x1,y1]);

Csx = subs(Csx, {'phi1','phi2','x1','y1'},...

[x0(1),x0(2),x1,y1]);

sigma = parms.k\*Cs; % Spring Force

% Create matrices

M = diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I]);

A = [M Cx';Cx zeros(4,4)];

F = [parms.m\*parms.g 0 0 parms.m\*parms.g 0 0]' - Csx\*sigma; % Updated F vector

b = [F;-Cdd];

xdd = A\b;

end

# Appendix B – Active element

%% MBD\_B: Assignment 3

% Question 2 - Active Element

% Rick Staa (4511328)

% Last edit: 05/03/2018

clear all; close all; % clc;

%% Parameters

% Segment 1

parms.L = 0.55; % [m]

parms.w = 0.05; % [m]

parms.t = 0.004; % [m]

parms.p = 1180; % [kg/m^3]

parms.m = parms.p \* parms.w \* parms.t \* parms.L; % [kg]

parms.I = (1/12) \* parms.m \* parms.L^2; % [kg\*m^2]

% World parameters

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

% Spring parameters

parms.k = (15/2)\*parms.m\*parms.g/parms.L; % stiffness of spring

% Motor constraint

omega = -120\*(2\*pi/60); % motor speed

%% Compute constraint matrices

% Use symbolic toolbox to calculate derivatives

syms x1 y1 phi1 x2 y2 phi2 t

syms dx1 dy1 dphi1 dx2 dy2 dphi2

x = [x1 y1 phi1 x2 y2 phi2];

xd = [dx1 dy1 dphi1 dx2 dy2 dphi2];

parms.x = x;

parms.xd = xd;

% The normal constrain equations

C = [x1-(parms.L/2)\*cos(phi1); y1-(parms.L/2)\*sin(phi1) ; ...

(x2-(parms.L/2)\*cos(phi2))-(x1+(parms.L/2)\*cos(phi1)); ...

(y2-(parms.L/2)\*sin(phi2))-(y1+(parms.L/2)\*sin(phi1)); ...

(phi1 - omega\*t)]; % Add extra motor constraint

% Calculate the jacobian to create the constraint equations

Cx = jacobian(C,x);

Cx = simplify(Cx);

% Constraint derivative with respect to time

Ct = simplify(jacobian(C,t)); % First derivative to time

Ctt = simplify(jacobian(Ct,t)); % double derivative to time

% Take the second derivative to create the gluing constraints

Cd = Cx\*xd';

Cdd = jacobian(Cd,x)\*xd';

Cdd = simplify(Cdd); % Convective term constraints

Cdt = simplify(jacobian(Cd,t)); % Convective to time

% Put Csx Cdp Cd cx in parms struct and feed tem into the solver

syst.Cx = Cx;

syst.Cd = Cd;

syst.Cdd = Cdd;

syst.Ct = Ct;

syst.Ctt = Ctt;

syst.Cdt = Cdt;

%% A:

x0 = [0.5\*pi 0.5\*pi omega omega];

[xdd] = state\_calc(x0,parms,syst);

xdd = double(vpa(xdd));

%% A: Create state space matrices for the case when we add a spring

function [xdd] = state\_calc(x0,parms,syst)

% Get system parameters out

structname\_fields = fields(parms);

for i = 1:size(fields(parms))

eval\_str = [structname\_fields{i,:},'=','parms.',structname\_fields{i,:},';'];

eval(eval\_str);

end

% Get symbolic expressions out

structname\_fields = fields(syst);

for i = 1:size(fields(syst))

eval\_str = [structname\_fields{i,:},'=','syst.',structname\_fields{i,:},';'];

eval(eval\_str);

end

Cx = subs(Cx, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cdd = subs(Cdd, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Ct = subs(Ct, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

Cdt = subs(Cdt, {'phi1','phi2','dphi1','dphi2'},...

[x0(1),x0(2),x0(3),x0(4)]);

% Create matrices

M = diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I]);

A = [M Cx';Cx zeros(5,5)];

F = [parms.m\*parms.g 0 0 parms.m\*parms.g 0 0]';

b = [F;-(Cdd+Cdt+Ctt)];

xdd = A\b;

end

# Appendix C – Impulsive impact force

%% MBD\_B: Assignment 3

% Question 3 - impact

% Rick Staa (4511328)

% Last edit: 05/03/2018

clear all; close all; % clc;

%% Parameters

% Segment 1

parms.L = 0.55; % [m]

parms.w = 0.05; % [m]

parms.t = 0.004; % [m]

parms.p = 1180; % [kg/m^3]

parms.m = parms.p \* parms.w \* parms.t \* parms.L; % [kg]

parms.I = (1/12) \* parms.m \* parms.L^2; % [kg\*m^2]

% World parameters

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

% Motor constraint

omega = 120\*(2\*pi/60);

parms.e = 1; % Coefficient of restitution

%% Compute constraint matrices

% Use symbolic toolbox to calculate derivatives

syms x1 y1 phi1 x2 y2 phi2 t

syms dx1 dy1 dphi1 dx2 dy2 dphi2

x = [x1 y1 phi1 x2 y2 phi2];

xd = [dx1 dy1 dphi1 dx2 dy2 dphi2];

parms.x = x;

parms.xd = xd;

% The normal constrain equations

C = [x1-(parms.L/2)\*cos(phi1); y1-(parms.L/2)\*sin(phi1) ; ...

(x2-(parms.L/2)\*cos(phi2))-(x1+(parms.L/2)\*cos(phi1)); ...

(y2-(parms.L/2)\*sin(phi2))-(y1+(parms.L/2)\*sin(phi1)); ...

(x1 + (parms.L/2)\*cos(phi1)); ...

(x2 + (parms.L/2)\*cos(phi2))]; % Extra constraint

% Calculate the jacobian to create the constraint equations

Cx = jacobian(C,x);

Cx = simplify(Cx);

% Put Csx Cdp Cd cx in parms struct and feed tem into the solver

syst.Cx = Cx;

%% C:

parms.phi1\_0 = pi/2; % angle of bar 1 (with x-axis)

parms.phi2\_0 = pi/2; % angle of bar 2

phi1d\_0 = 120\*(2\*pi/60); % angular velocity of bar 1

phi2d\_0 = 120\*(2\*pi/60); % angular velocity of bar 2

x1d\_0 = -(parms.L/2)\*sin(parms.phi1\_0)\*phi1d\_0;

x2d\_0 = -(3\*parms.L/2)\*sin(parms.phi2\_0)\*phi2d\_0;

y1d\_0 = 0;

y2d\_0 = 0;

% Calculate impatct forces

x0 = [x1d\_0;y1d\_0;phi1d\_0;x2d\_0;y2d\_0;phi2d\_0];

[xdd] = state\_calc(x0,parms,syst);

xdd = double(vpa(xdd));

M = diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I]);

K\_be = 0.5\*x0'\*M\*x0;

K\_af = 0.5\*xdd(1:6)'\*M\*xdd(1:6);

W = 0.5\*xdd(7:end)'\*x0\*(1-parms.e);

disp(xdd)

disp(K\_be)

disp(K\_af)

disp(W)

%% A: Create state space matrices for the case when we add a spring

function [xdd] = state\_calc(x0,parms,syst)

% Get system matrices out

structname\_fields = fields(parms);

for i = 1:size(fields(parms))

eval\_str = [structname\_fields{i,:},'=','parms.',structname\_fields{i,:},';'];

eval(eval\_str);

end

structname\_fields = fields(syst);

for i = 1:size(fields(syst))

eval\_str = [structname\_fields{i,:},'=','syst.',structname\_fields{i,:},';'];

eval(eval\_str);

end

Cx = subs(Cx, {'phi1','phi2','dphi1','dphi2'},...

[parms.phi1\_0,parms.phi2\_0,x0(3),x0(6)]);

% Create matrices

M = diag([parms.m,parms.m,parms.I,parms.m,parms.m,parms.I]);

A = [M Cx';Cx zeros(6,6)];

F = [M\*x0];

b = [F;-parms.e\*Cx\*x0];

xdd = A\b;

end