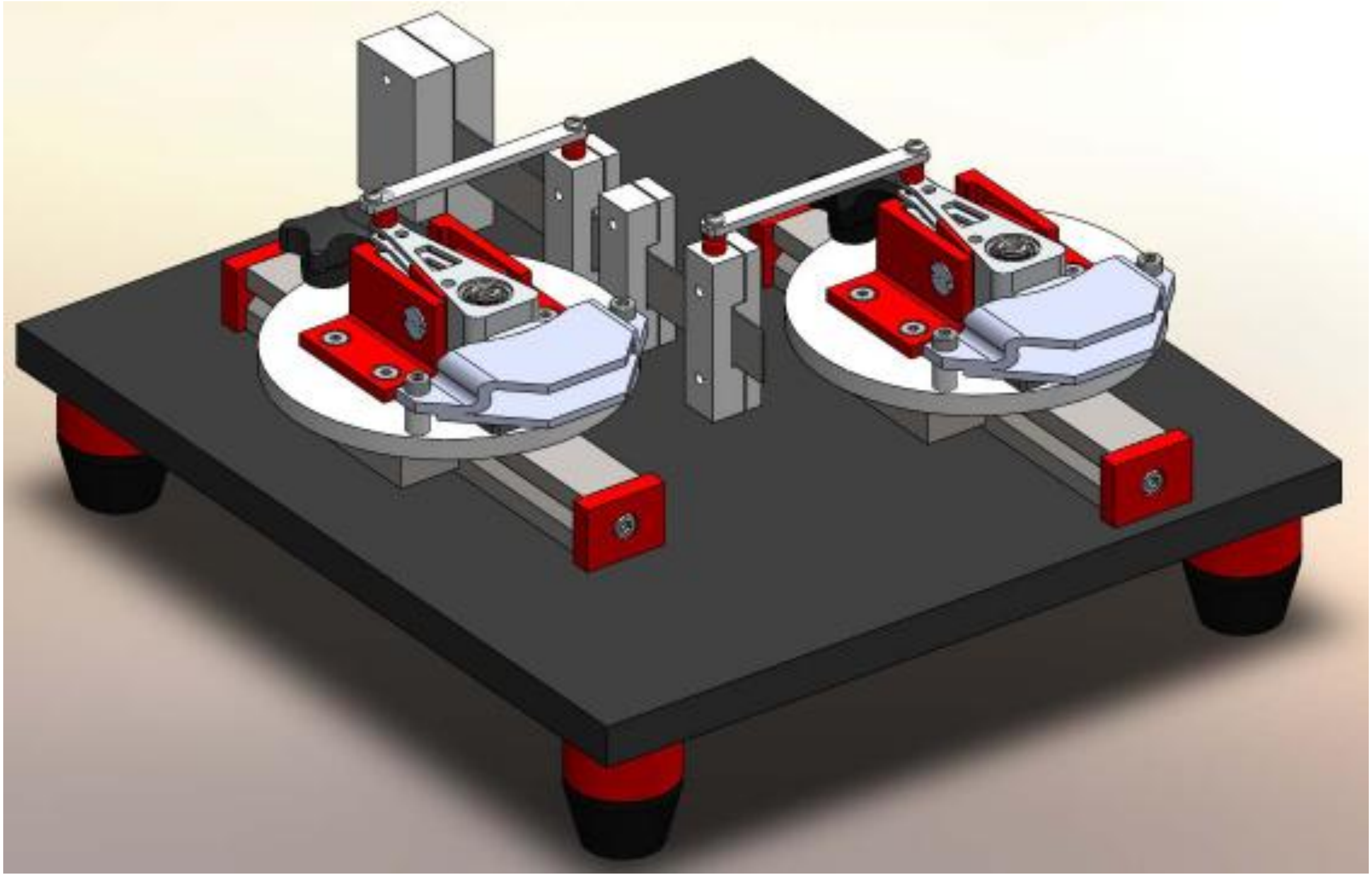
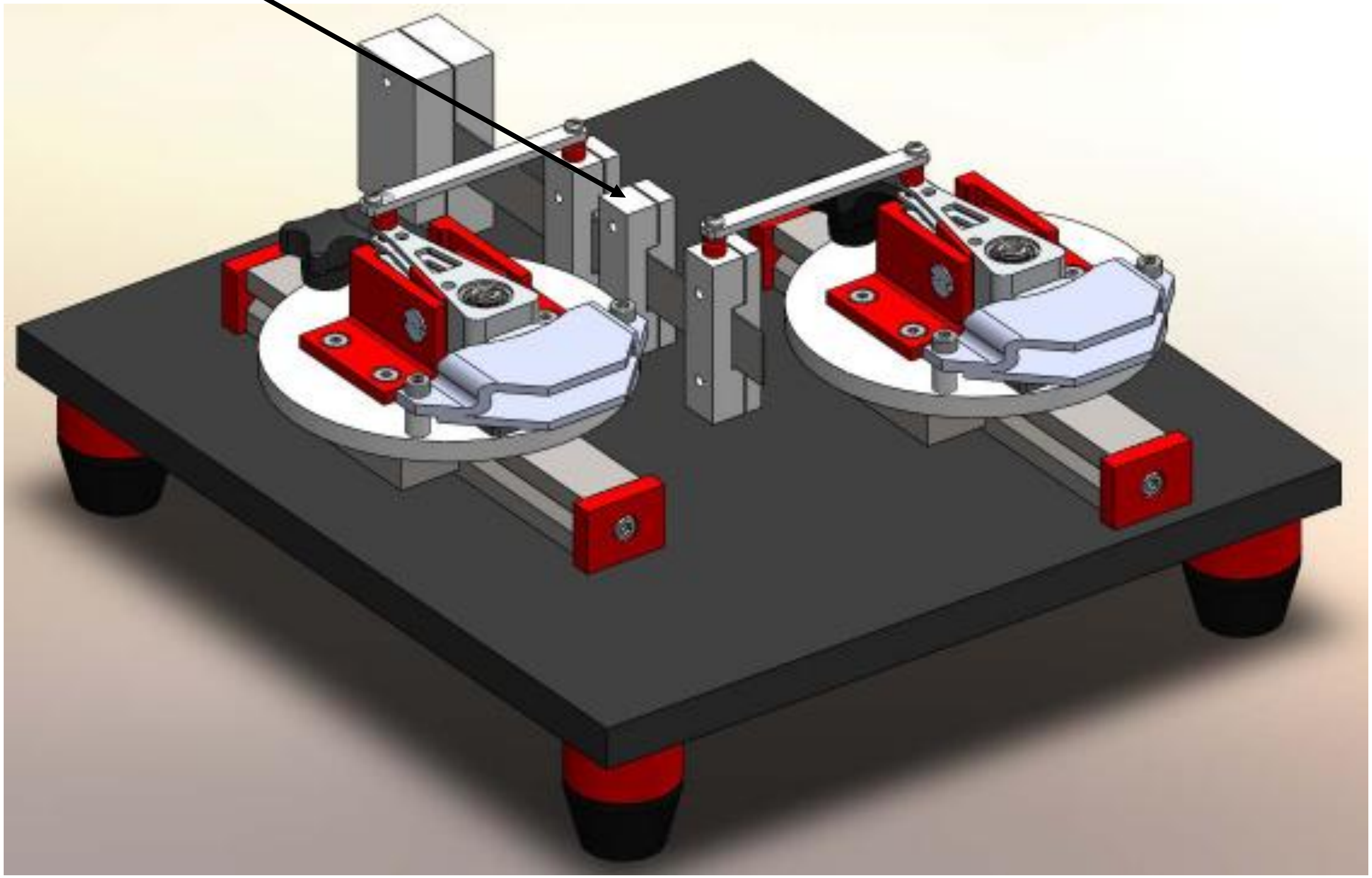
FlexPlant experimentation model



A **mass** attached to a very flexible beam (blade) is to be controlled by mean of two EM actuators acting on different portions of the blade.

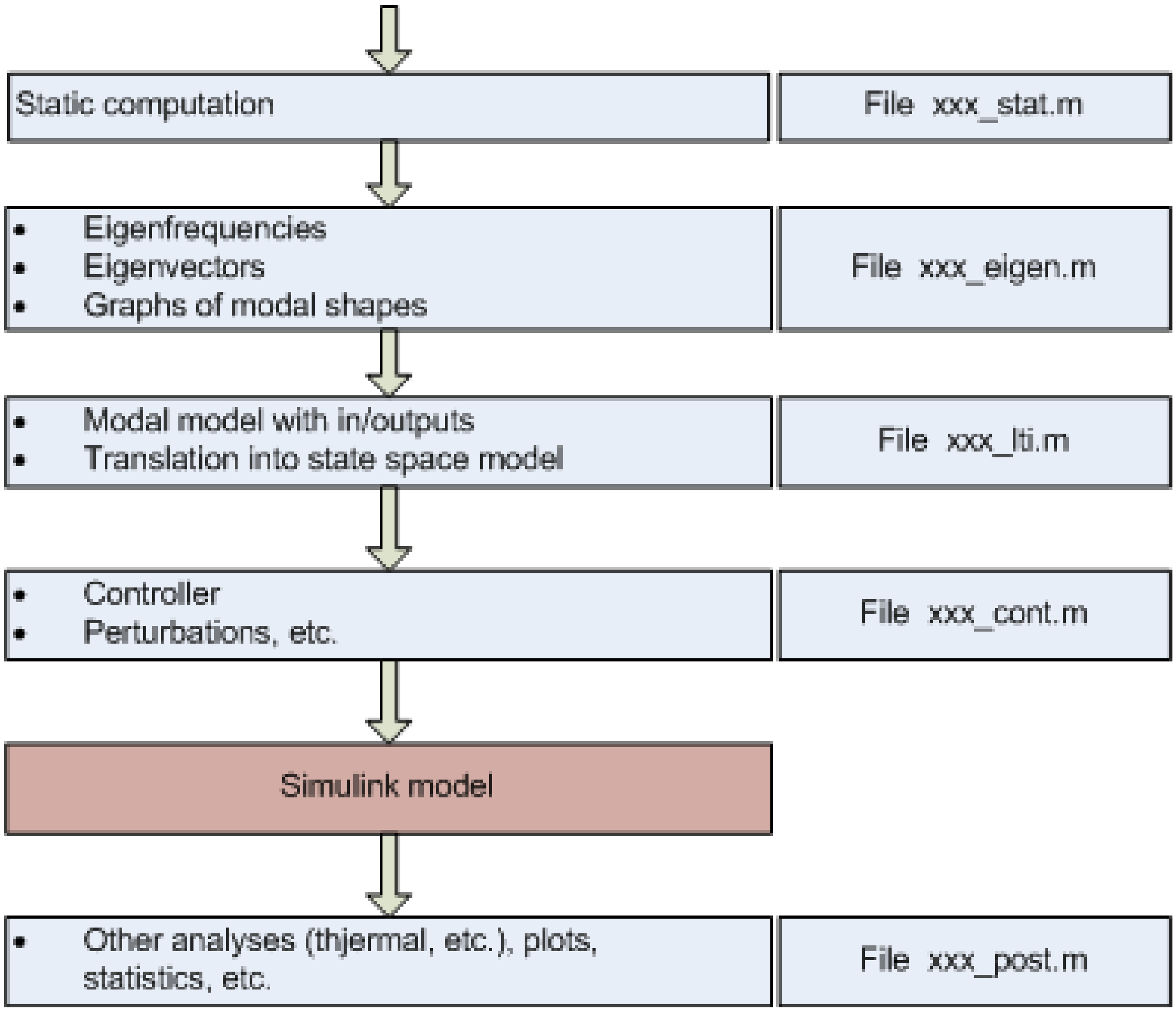


Integrated modeling based on Matlab

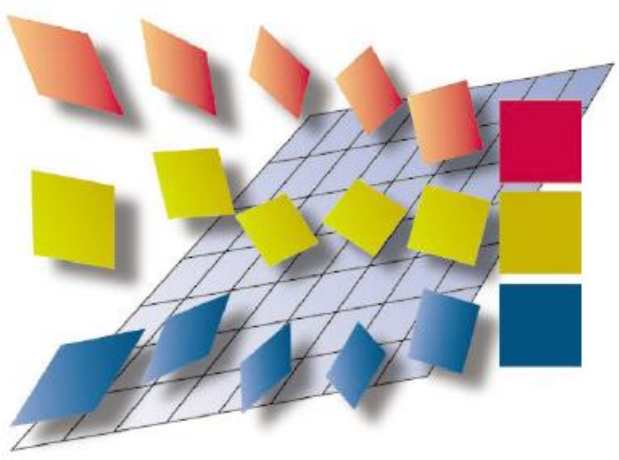
- A MAIN script *xxx_main.m* calls the modules dealing with the various aspects of the system:
  - Inputs and parameters
  - Structures
  - Static and dynamic response
  - Mcatronic aspects (actuators, etc.)
  - Feedback loops
  - Other relevant: thermal, optical, …

Proposed structure of the model

Problem input data: • Geometry • Caracteristics of materials • Load andf limit conditions • Parameters of finite lement types • Parameters of motorisation et control	File xxx_dat.m
Geometry of the finite lement model: • Nodes • Elements • Types • Nodes where forces and measurements are applied	File xxx_geo.m
Topology of the finite lement model: • Degrees of freedom • Verification of number and type of elements • Drawing of the model	File xxx_fem.m
Assembling of matrices K et M (and eventually C) the finite lement model	File xxx_KCM.m


Static computation
File xxx_stat.m
Eigenfrequencies
Eigenvectors
Graphs of modal shapes
File xxx_eigen.m
Modal model with in/outputs
Translation into state space model
File xxx_lti.m
Controller
Perturbations, etc.
File xxx_cont.m
Simulink model
Other analyses (thjermal, etc.), plots, statistics, etc.
File xxx_post.m

Some FEM toolboxes for Matlab



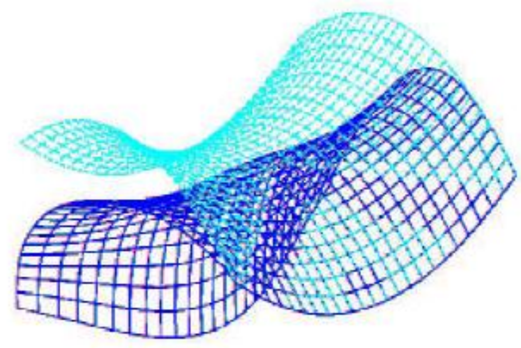
CALFEM

A FINITE ELEMENT TOOLBOX

Version 3.4

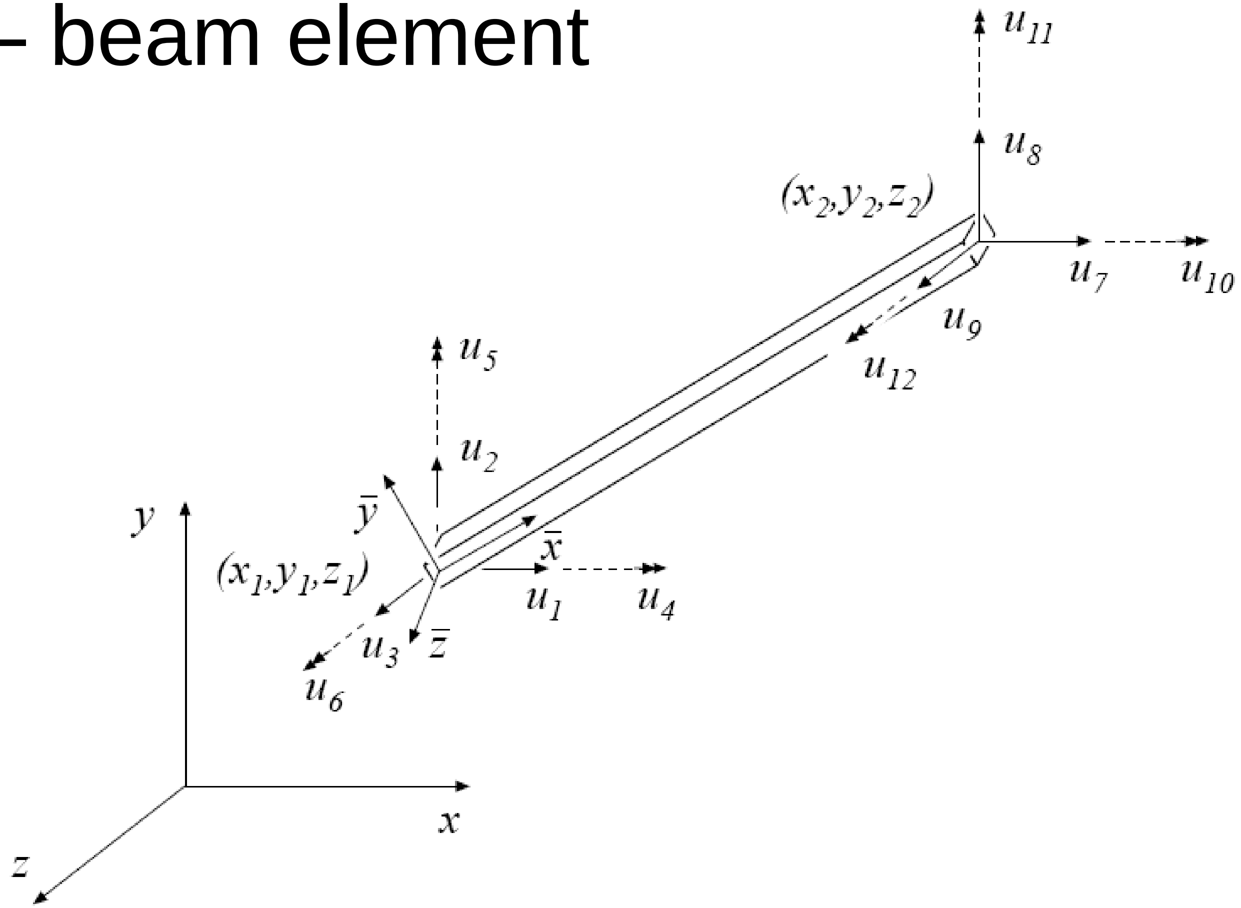
OpenFEM

A finite element toolbox for Matlab and Scilab



Extras: «home-made» Matlab toolbox «maison» with various useful functions and extensions for CalFem and dynamic simulation.

beam3d – beam element



- Syntax:

```
[ke, me] = beam3d (ex,ey,ez,eo,ep);
```

- Element based on **beam3e** of CalFem: we added the computation of the mass matrix **[me]**.

The input variables

$$\mathsf{ex} = [\ x_1 \ \ x_2\ ]$$
$$\mathsf{ey} = [\ y_1 \ \ y_2\ ] \qquad \mathsf{eo} = [\ x_{\bar{z}} \ \ y_{\bar{z}} \ \ z_{\bar{z}}\ ]$$
$$\mathsf{ez} = [\ z_1 \ \ z_2\ ]$$

supply the element nodal coordinates x_1, y_1, etc. as well as the direction of the local beam coordinate system $(\bar{x}, \bar{y}, \bar{z})$. By giving a global vector $(x_{\bar{z}}, y_{\bar{z}}, z_{\bar{z}})$ parallel with the positive local $\bar{z}$ axis of the beam, the local beam coordinate system is defined. The variable

$$\mathsf{ep} = [\ E \ \ G \ \ A \ \ I_{\bar{y}} \ \ I_{\bar{z}} \ \ K_v\ ]$$

supplies the modulus of elasticity E, the shear modulus G, the cross section area A, the moment of inertia with respect to the $\bar{y}$ axis I_y, the moment of inertia with respect to the $\bar{z}$ axis I_z, and St Venant torsional stiffness K_v.

Main script   lame_main.m

```
clear
close all
format compact

lame_dat
lame_geo2a
lame_fem

lame_KCM
lame_static
lame_eigen

% ------ Dynamic response -----------------------------------------
freqvec = logspace(0,3,100)';   w=2*pi*freqvec;
[sys] = lame_lti(mdof,N_motor1,N_motor2,n_1,n_M,freq,Egv,w);

Ts = 100e-6
sysd = c2d(sys,Ts);
Tend = 0.3
```

lame_dat.m
all input parameters

```
% Materials
  pois = 0.3;
% Acier
  rho_st = 7800.;
  E_st = 210000e6;  G_st = E_st / (2*(1+pois));
% Alu
  rho_Al = 2800.;
  E_Al = 69000e6;   G_Al = E_Al / (2*(1+pois));

% Geometry
  L = 0.110;     % longueur totale de la poutre
  H = 0.01275;   % hauteur
  W = 0.0005;    % épaisseur
  np = 22;       % nombre d'éléments

  L1 = 0.040;
  L2 = 0.070;
  L3 = 0.100;

  HM = 0.040;
  WM = 0.010;
  LM = 0.010;

  L_act   = 0.036;
  L_motor = 0.030;
  t_bar = 0.006;];
```

lame_dat.m (suite)

```
% Constante des moteurs
  Kt = 0.06;     % Nm/A
  Lt = 0.001;    % mH
  Rt = 11;       % Ohm


% Proprietés des éléments
% Lame
h = W;              b = H;
[A, Iy, Iz, J] = rect_pro (b,h);
Ep_lame = [E_st  G_st  A Iy Iz J A*rho_st];

% Masselette
h = WM;             b = WM;
[A, Iy, Iz, J] = rect_pro (b,h);
Ep_block = [E_st  G_st  A Iy Iz J A*rho_st];

% Elément rigide sans masse
r = 0.010;
[A, Iy, Iz, J] = circ_pro (r);
Ep_stiff = [E_Al  G_Al  A Iy Iz J A*rho_Al/1000];

% Barre de transmission
h = t_bar;                    b = t_bar;
[A, Iy, Iz, J] = rect_pro (b,h);
Ep_bar = [E_Al  G_Al  A Iy Iz J A*rho_Al];

% Charnière flexible
h = 1e-4;           b = 1e-4;
[A, Iy, Iz, J] = rect_pro (b,h);
Ep_H = [E_st  G_st  A Iy Iz J*1000 A*rho_st];
```

lame_geo2a.m

Model geometry:

- Nodes
- Elements and type
- Nodes for boundary conditions and loads

```
% Model geometry
Coord (1,:) = [0  0  0];
n_fix = 1;

% Flex blade elements
n = size(Coord,1);
ne = 0;
dL = L/np;
for i = 1:np
    Coord (n+i,:) = [L/np*i  0  0];
    Elem(ne+i,:) = [n+i-1 n+i];
end
n_extreme = np;
```

lame_geo2a.m (2)

```
% Masselettes ...
for i = 1:np
    if (abs(L1 - Coord(n+i,1)) <= dL/2)
        Coord(n+i,1) = L1;
        Coord(n+i-1,1) = L1-LM/2;
        Coord(n+i+1,1) = L1+LM/2;
        n_1 = n+i;
        eMass = [n+i n+i+1];
        break
    end
end

for i = 1:np
    if (abs(L2 - Coord(n+i,1)) <= dL/2)
        Coord(n+i,1) = L2;
        Coord(n+i-1,1) = L2-LM/2;
        Coord(n+i+1,1) = L2+LM/2;
        n_M = n+i;
        eMass = [eMass  n+i n+i+1];
        break
    end
end

for i = 1:np
    if (abs(L3 - Coord(n+i,1)) <= dL/2)
        Coord(n+i,1) = L3;
        Coord(n+i-1,1) = L3-LM/2;
        Coord(n+i+1,1) = L3+LM/2;
        n_2 = n+i;
        eMass = [eMass  n+i n+i+1];
        break
    end
end
```

lame_geo2a.m (3)

```
% Masselettes ...
for i = 1:np
    if (abs(L1 - Coord(n+i,1)) <= dL/2)
        Coord(n+i,1) = L1;
        Coord(n+i-1,1) = L1-LM/2;
        Coord(n+i+1,1) = L1+LM/2;
        n_1 = n+i;
        eMass = [n+i n+i+1];
        break
    end
end

for i = 1:np
    if (abs(L2 - Coord(n+i,1)) <= dL/2)
        Coord(n+i,1) = L2;
        Coord(n+i-1,1) = L2-LM/2;
        Coord(n+i+1,1) = L2+LM/2;
        n_M = n+i;
        eMass = [eMass  n+i n+i+1];
        break
    end
end

for i = 1:np
    if (abs(L3 - Coord(n+i,1)) <= dL/2)
        Coord(n+i,1) = L3;
        Coord(n+i-1,1) = L3-LM/2;
        Coord(n+i+1,1) = L3+LM/2;
        n_2 = n+i;
        eMass = [eMass  n+i n+i+1];
        break
    end
end
eMass
eFlex = setdiff([1:np], eMass)
eMass
eFlex = setdiff([1:np], eMass)
```

lame_geo2a.m (4)

```
% Eléments rigides pour visualisation

n = size(Coord,1);  ne = size(Elem,1);
Coord(n+1,:) = Coord(n_1,:) + [0 0 HM/2];
Coord(n+2,:) = Coord(n_1,:) - [0 0 HM/2];
Elem(ne+1,:) = [ n_1 n+1 ];
Elem(ne+2,:) = [ n_1 n+2 ];
eStiff = [ne+1  ne+2];

n = size(Coord,1);  ne = size(Elem,1);
Coord(n+1,:) = Coord(n_M,:) + [0 0 HM/2];
Coord(n+2,:) = Coord(n_M,:) - [0 0 HM/2];
Elem(ne+1,:) = [ n_M n+1 ];
Elem(ne+2,:) = [ n_M n+2 ];
eStiff = [eStiff  ne+1  ne+2];

n = size(Coord,1);  ne = size(Elem,1);
Coord(n+1,:) = Coord(n_2,:) + [0 0 HM/2];
Coord(n+2,:) = Coord(n_2,:) - [0 0 HM/2];
Elem(ne+1,:) = [ n_2 n+1 ];
Elem(ne+2,:) = [ n_2 n+2 ];
eStiff = [eStiff  ne+1  ne+2];
```

lame_geo2a.m
(end)

```
% Actionneur-1
n = size(Coord,1);  ne = size(Elem,1);
Coord(n+1,:) = Coord(n_1,:)+[0        -0.001
0];
Coord(n+2,:) = Coord(n_1,:)+[0        -
L_act+0.001  0];
Coord(n+3,:) = Coord(n_1,:)+[0        -L_act
0];
Coord(n+4,:) = Coord(n_1,:)+[L_motor  -L_act
0];
N_motor1 = n+4;
Elem (ne+1,:) = [n_1 n+1];
Elem (ne+2,:) = [n+1 n+2];
Elem (ne+3,:) = [n+2 n+3];
Elem (ne+4,:) = [n+3 n+4];
eAct  = [ne+2  ne+4];
eH    = [ne+1  ne+3];

% Actionneur-2
n = size(Coord,1);  ne = size(Elem,1);
Coord(n+1,:) = Coord(n_2,:)+[0        0.001
0];
Coord(n+2,:) = Coord(n_2,:)+[0        L_act-
0.001  0];
Coord(n+3,:) = Coord(n_2,:)+[0        L_act
0];
Coord(n+4,:) = Coord(n_2,:)+[L_motor  L_act
0];
N_motor2 = n+4;
Elem (ne+1,:) = [n_2 n+1];
Elem (ne+2,:) = [n+1 n+2];
Elem (ne+3,:) = [n+2 n+3];
Elem (ne+4,:) = [n+3 n+4];
eAct  = [eAct ne+2  ne+4];
eH    = [eH  ne+1  ne+3];
```

lame_fem.m

Topology of the finite elements model:

- Dof(), Edof(), Ex(), Ey(), Ez()
- Plots

```
% ------ topology -----------------------------------------
[n_nodes,n_dof,n_elem,n_nel,Dof,Edof] = topol (Coord,Elem);
[Ex,Ey,Ez] = coordxtr(Edof,Coord,Dof,n_nel);

% Check elements
Number_of_elements = size(Elem,1)
Check = length([ eFlex ePoutre])

Coord_xy(:,1) = Coord(:,1);       Coord_xy(:,2) = Coord(:,2);
figure; femdraw2 (Coord_xy,Ex,Ey,[1 4 2 5 0.]);
ylabel('y'); grid;

figure; femdraw3 (Coord,Ex,Ey,Ez,[1 4 2 5 0.]);
grid; axis('equal');
```

lame_KCM.m

```
% ------ generate element matrices, assemble in global matrices
nd = n_nodes*n_dof;
clear K M C
K = zeros(nd); M = zeros(nd); C = zeros(nd);
nr_elem = 0;

loc = 'Stiff elements';
for ie = eStiff
  eo(ie,:) = eo_beam (ie,Ex,Ey,Ez);
  [ke,me] = beam3d (Ex(ie,:),Ey(ie,:),Ez(ie,:),eo(ie,:),Ep_stiff);
  K = assem(Edof(ie,:),K,ke);
  M = assem(Edof(ie,:),M,me);
  nr_elem = nr_elem+1;
  m_elem(ie) = L_elem(ie) * Ep_stiff(7);
end

loc = 'Flex beam elements';
for ie = eFlex
  eo(ie,:) = [0 0 1];
  [ke,me] = beam3d (Ex(ie,:),Ey(ie,:),Ez(ie,:),[0 0 1],Ep_lame);
  K = assem(Edof(ie,:),K,ke);
  M = assem(Edof(ie,:),M,me);
  nr_elem = nr_elem+1;
  m_elem(ie) = L_elem(ie) * Ep_lame(7);
end


% ... etc ...


Check = nr_elem
```

lame_static.m

```
% Calcul statique
% ------ Boundary conditions ---------------------------------------
bc = [];
[b,bc,nb] = fix_point (bc,n_fix,Dof);
[b,bc,nb] = fix_xyz   (bc,N_motor1,Dof);
[b,bc,nbc] = fix_1d (bc,N_motor1,Dof,[4 5]);
[b,bc,nb] = fix_xyz   (bc,N_motor2,Dof);
[b,bc,nbc] = fix_1d (bc,N_motor2,Dof,[4 5]);

%  ------ Load - Z moment ----------------------------------
p = zeros(size(K,1),1);
i = N_motor1
   dof_exc = (i-1)*n_dof+6;
   p(dof_exc) = 0.1;   % Nm

[X,R,xyzF] = fe_stat (K,p,b,n_dof,n_nodes);

disp ('Moment 0.1 Nm about Z');
disp ('xyz n_1 (mm)');
disp (xyzF(n_1,1:3)*1000)

disp ('xyz beam extreme (mm)');
disp (xyzF(n_extreme,1:3)*1000)

Edb = extract (Edof,X);

figure; femdraw2 ([Coord(:,1) Coord(:,2)],Ex,Ey,[2 4 2 4 0.]);
Edbxy = [Edb(:,1) Edb(:,2) Edb(:,6) Edb(:,7) Edb(:,8) Edb(:,12)];
femdisp2 (Ex,Ey,Edbxy,[1 5 0 5],1); ylabel('y');
title('Static analysis 0.1 Nm torque');
```

lame_eigen.m

```
% Eigenmodes and eigenvectors
% we obtain:   n_modes = nomber of computed modes
%              freq = eigenfrequencies (Hz)
%              Egv = eigenvectors
```

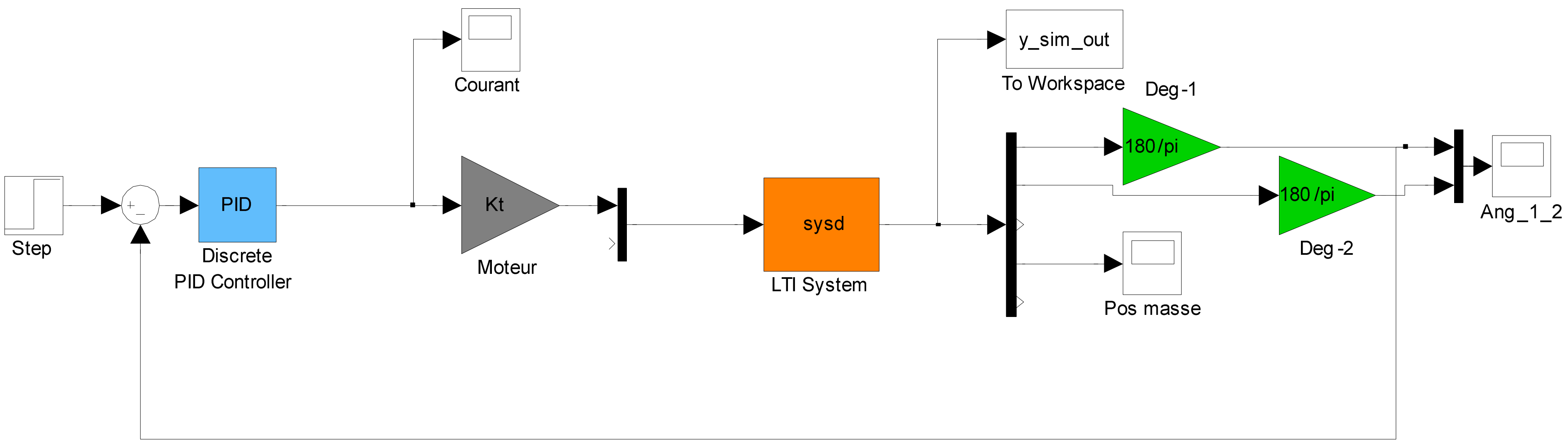
```
% ------ Boundary conditions ---------------------------------------
bc = [];
[b,bc,nb] = fix_point (bc,n_fix,Dof);
[b,bc,nb] = fix_xyz   (bc,N_motor1,Dof);
[b,bc,nbc] = fix_1d (bc,N_motor1,Dof,[4 5]);
[b,bc,nb] = fix_xyz   (bc,N_motor2,Dof);
[b,bc,nbc] = fix_1d (bc,N_motor2,Dof,[4 5]);

disp ('... Calculating eigenmodes')
[L,Egv] = eigen (K,M,bc);
freq = sqrt(L)/(2*pi);
disp ('fréq (Hz) =')
disp (freq(1:4))
n_modes = length (freq)

for i = [1:5]
  plt_mode (Coord,Ex,Ey,Ez,Edof,real(Egv),real(freq),i,0.001);
end

mdof = sdt_mdof(n_nodes,n_dof);
```

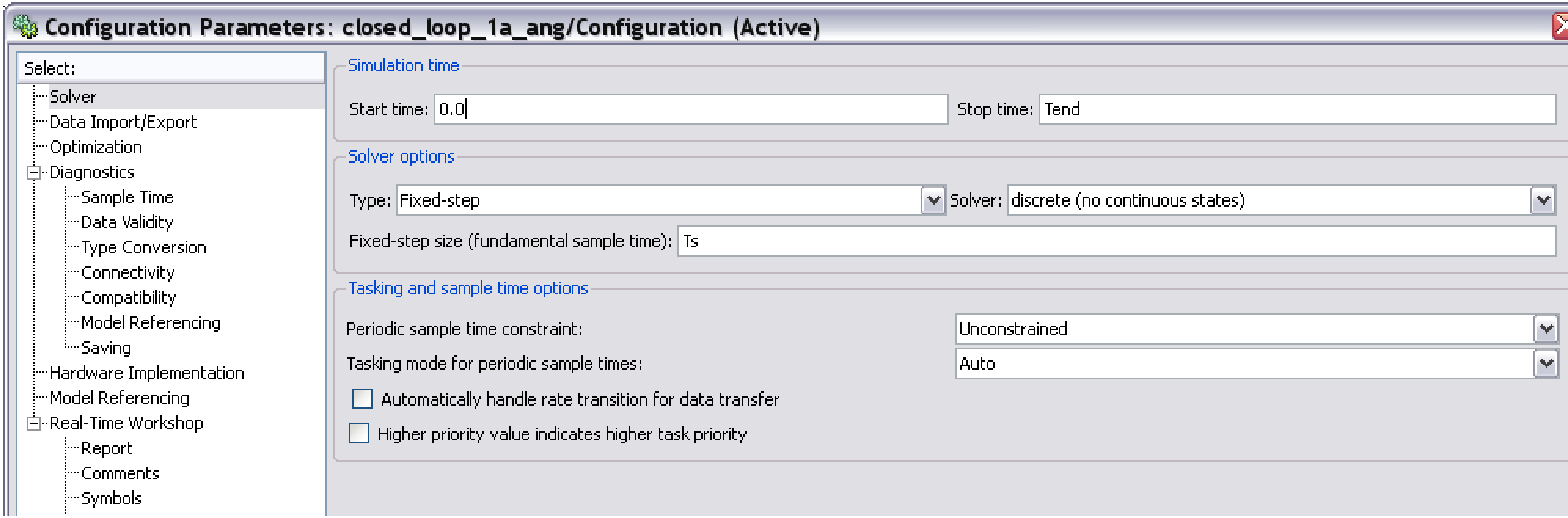
ex1_lti.m *lti = linear time invariant*

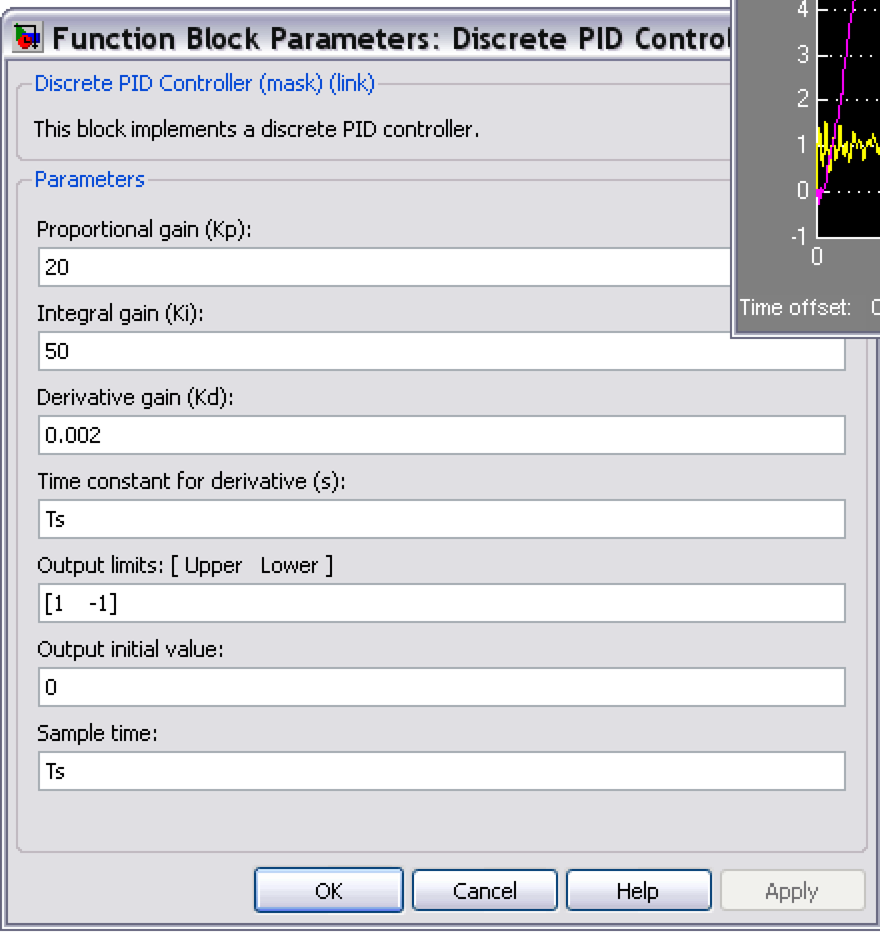
```
% Transfer function
function [sys] = lame_lti(mdof,N_motor1,N_motor2,n_1,n_M,freq,Egv,w)

nm = size(Egv,2);
zm = zeros(size(Egv));

% Inputs (drivers) -------------------------------------------------
b_Mot1 = fe_c(mdof,[N_motor1+0.06]',[1])';
pb1 = Egv'*b_Mot1;
b_Mot2 = fe_c(mdof,[N_motor2+0.06]',[1])';
pb2 = Egv'*b_Mot2;

pb = [ pb1 pb2 ];
size_pb = size(pb)

% Outputs ----------------------------------------------------------
c_Mot1 = fe_c(mdof,[N_motor1+0.06]',[1]);
cp1  = c_Mot1*Egv;
c_Mot2 = fe_c(mdof,[N_motor2+0.06]',[1]);
cp2  = c_Mot2*Egv;
c_S1 = fe_c(mdof,[n_1+0.02]',[1]);
cps1  = c_S1*Egv;
c_SM = fe_c(mdof,[n_M+0.02]',[1]);
cpsM  = c_SM*Egv;
c_S2 = fe_c(mdof,[n_M+0.02]',[1]);
cps2  = c_S2*Egv;

cp = [ cp1 zeros(1,nm); ; cp2 zeros(1,nm);
      cps1 zeros(1,nm); cpsM zeros(1,nm); cps2 zeros(1,nm) ];

sda = 0.01                 % structural damping = 1/2*Q
damp_col = ones(nm,1)*sda;

n_modes = length(freq)
[a,b,c,d] = nor2ss (freq*2*pi,damp_col,pb,cp);
sys = ss(a,b,c,d);

size_of_sys = size(sys)

my_plot_bode (w,sys(1,1),'b','Sensor - motor-1');
my_plot_bode (w,sys(2,1),'b','N_1 - motor-1');
my_plot_bode (w,sys(3,1),'b','N_m - motor-1');
```

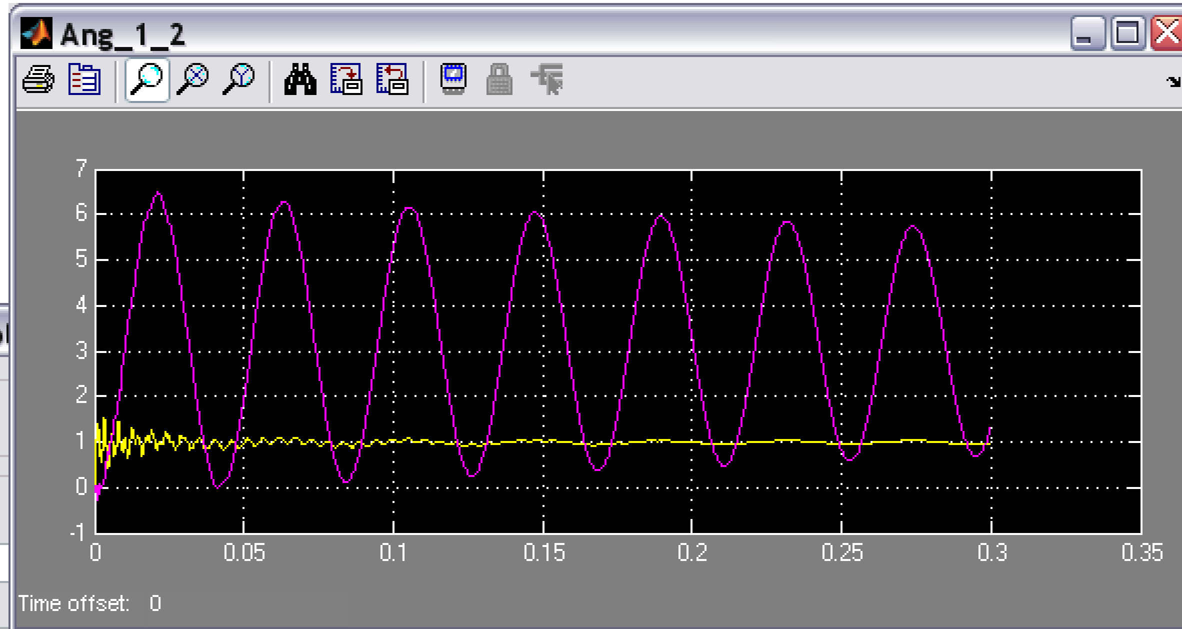
Simulink

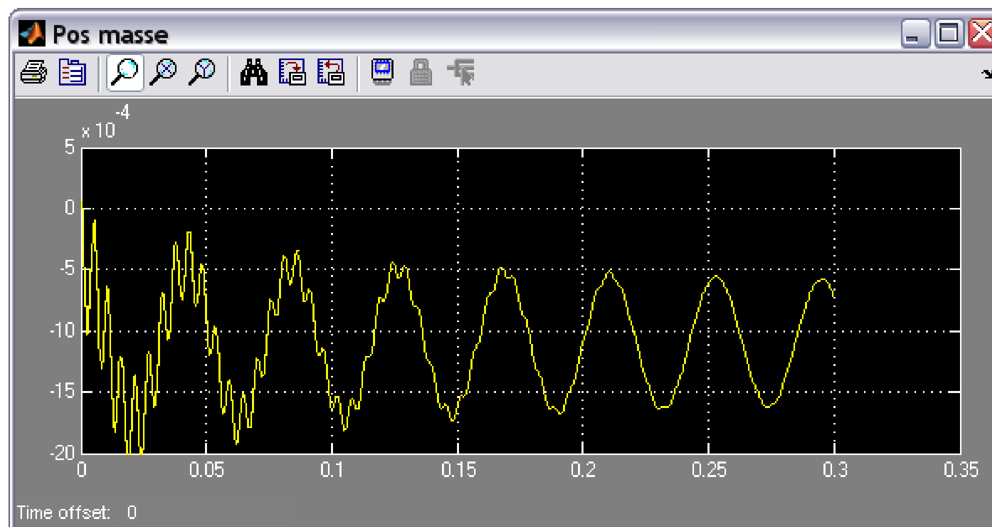
1: Control of first motor only, sensor *co-located*




Configuration Parameters: closed_loop_1a_ang/Configuration (Active)
Select:
Solver
Data Import/Export
Optimization
Diagnostics
Sample Time
Data Validity
Type Conversion
Connectivity
Compatibility
Model Referencing
Saving
Hardware Implementation
Model Referencing
Real-Time Workshop
Report
Comments
Symbols
Simulation time
Start time: 0.0
Stop time: Tend
Solver options
Type: Fixed-step
Solver: discrete (no continuous states)
Fixed-step size (fundamental sample time): Ts
Tasking and sample time options
Periodic sample time constraint:
Unconstrained
Tasking mode for periodic sample times:
Auto
Automatically handle rate transition for data transfer
Higher priority value indicates higher task priority

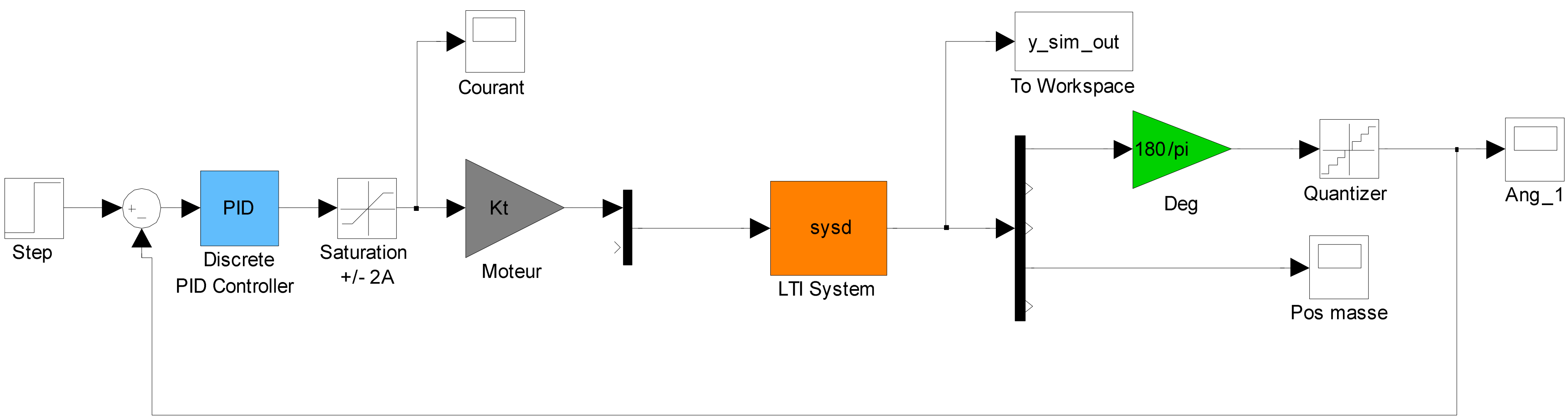

Function Block Parameters: Discrete PID Control
Discrete PID Controller (mask) (link)
This block implements a discrete PID controller.
Parameters
Proportional gain (Kp):
20
Integral gain (Ki):
50
Derivative gain (Kd):
0.002
Time constant for derivative (s):
Ts
Output limits: [Upper Lower]
[1 -1]
Output initial value:
0
Sample time:
Ts
OK
Cancel
Help
Apply

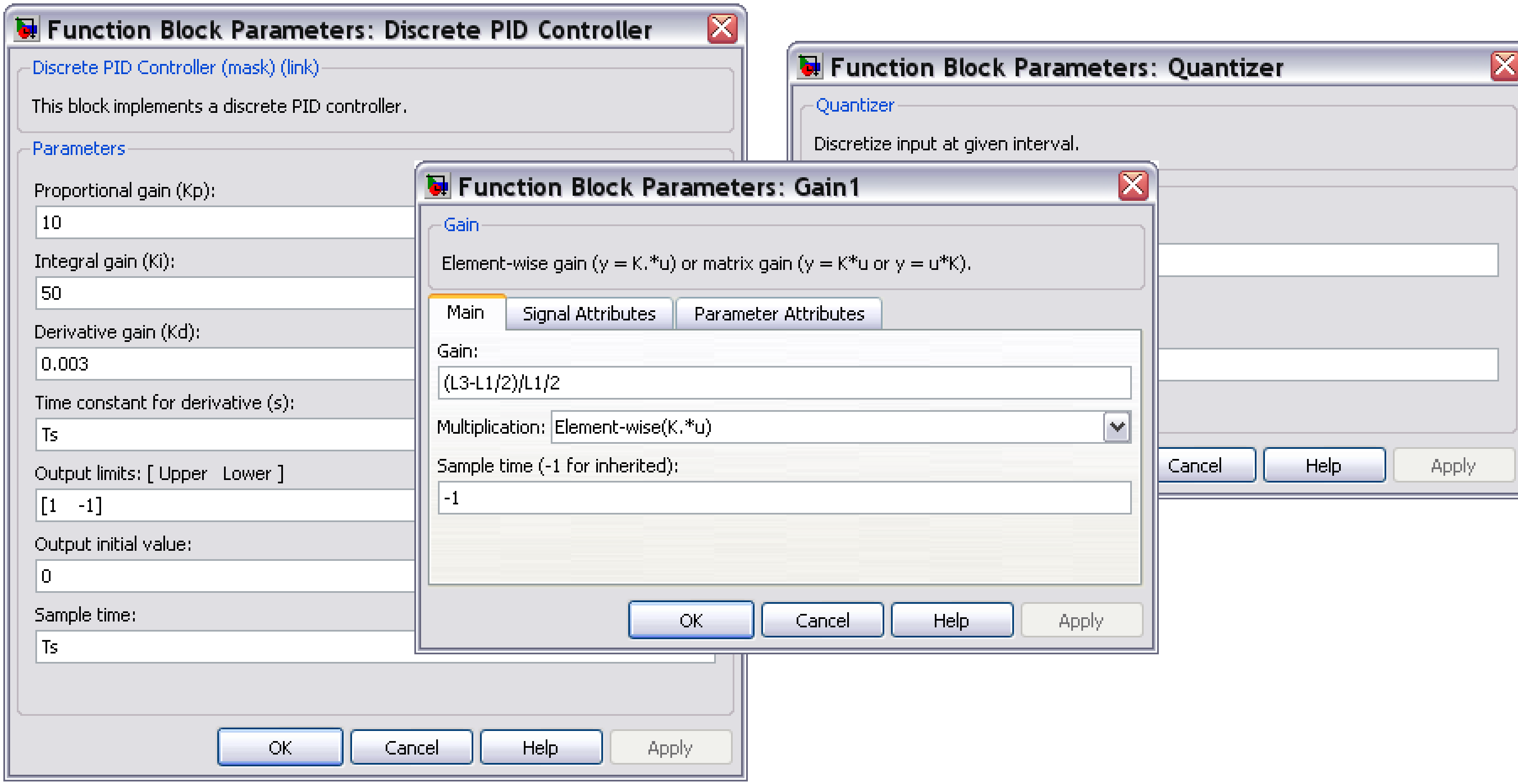

Ang_1_2
7
6
5
4
3
2
1
0
-1
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time offset: 0


Pos masse
x 10
-4
5
0
-5
-10
-15
-20
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time offset: 0

Simulink

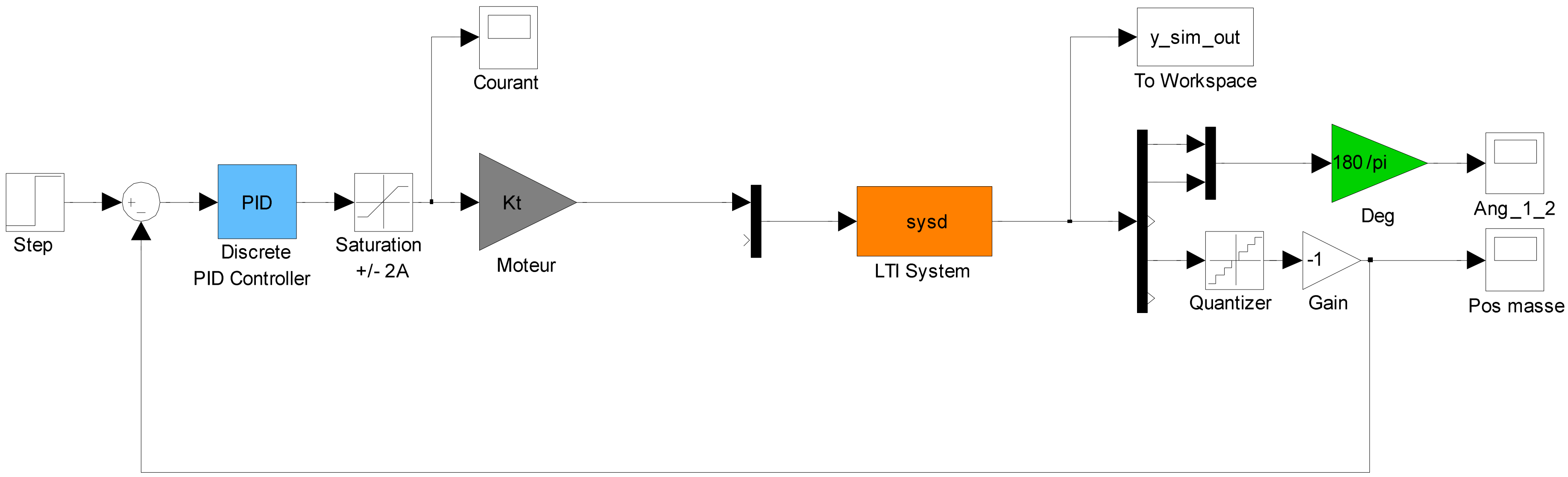
1a: Control of first motor only,
sensor *co-located digital*, current limited

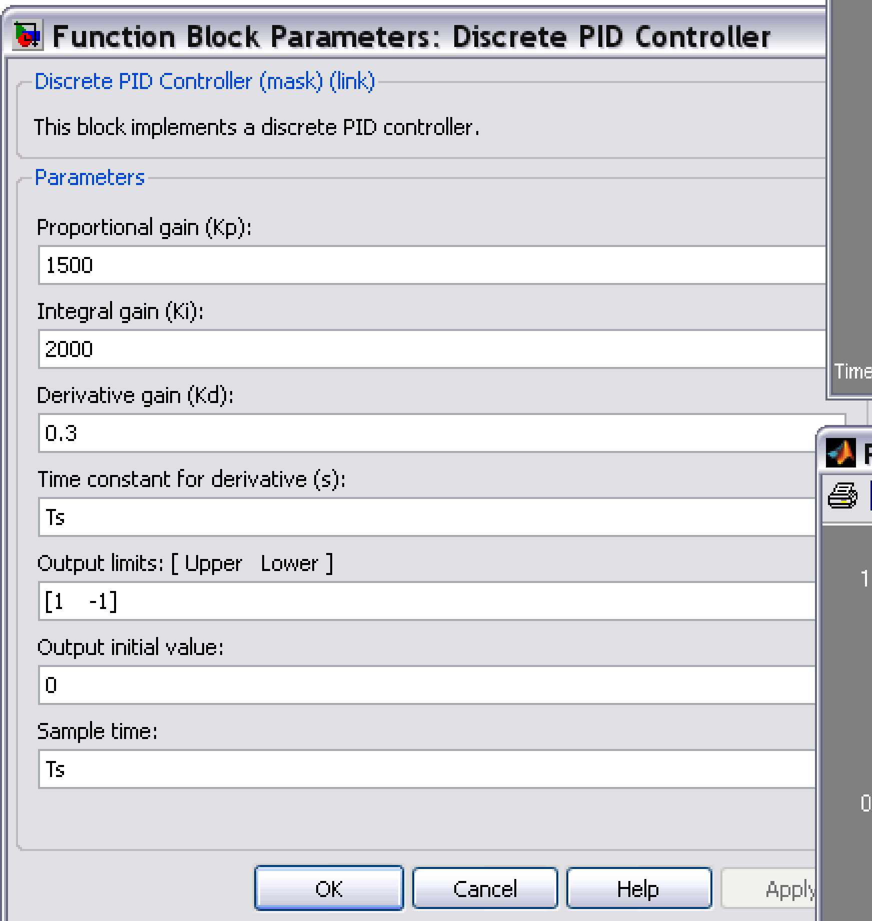


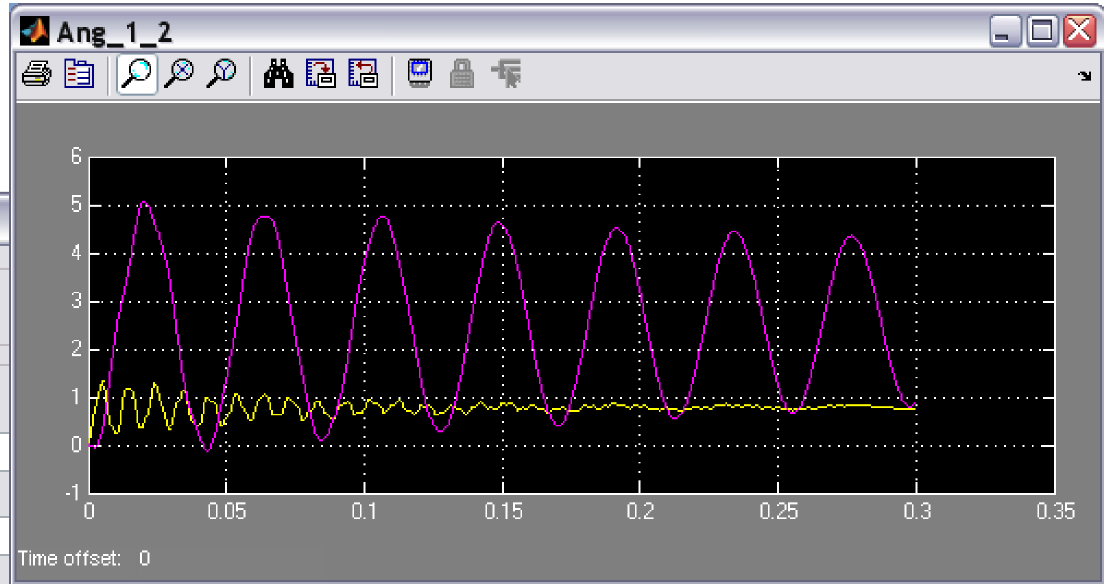

Function Block Parameters: Discrete PID Controller
Discrete PID Controller (mask) (link)
This block implements a discrete PID controller.
Parameters
Proportional gain (Kp):
10
Integral gain (Ki):
50
Derivative gain (Kd):
0.003
Time constant for derivative (s):
Ts
Output limits: [Upper Lower]
[1 -1]
Output initial value:
0
Sample time:
Ts
OK
Cancel
Help
Apply
Function Block Parameters: Quantizer
Quantizer
Discretize input at given interval.
Cancel
Help
Apply
Function Block Parameters: Gain1
Gain
Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).
Main
Signal Attributes
Parameter Attributes
Gain:
(L3-L1/2)/L1/2
Multiplication:
Element-wise(K.*u)
Sample time (-1 for inherited):
-1
OK
Cancel
Help
Apply

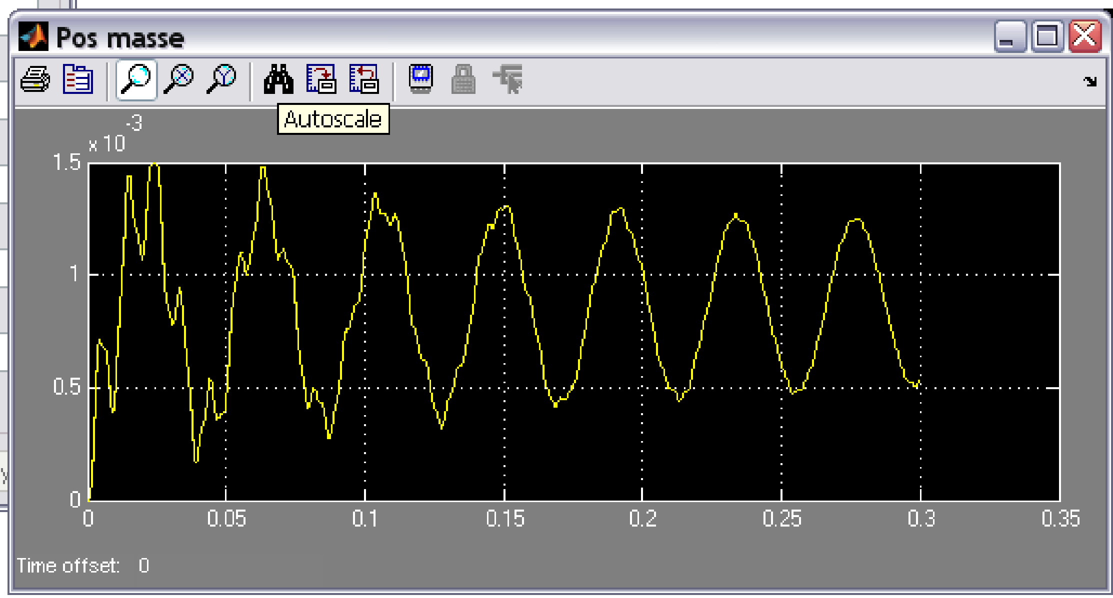
Simulink

2: Control of first motor,
sensor on central mass, *non co-located*



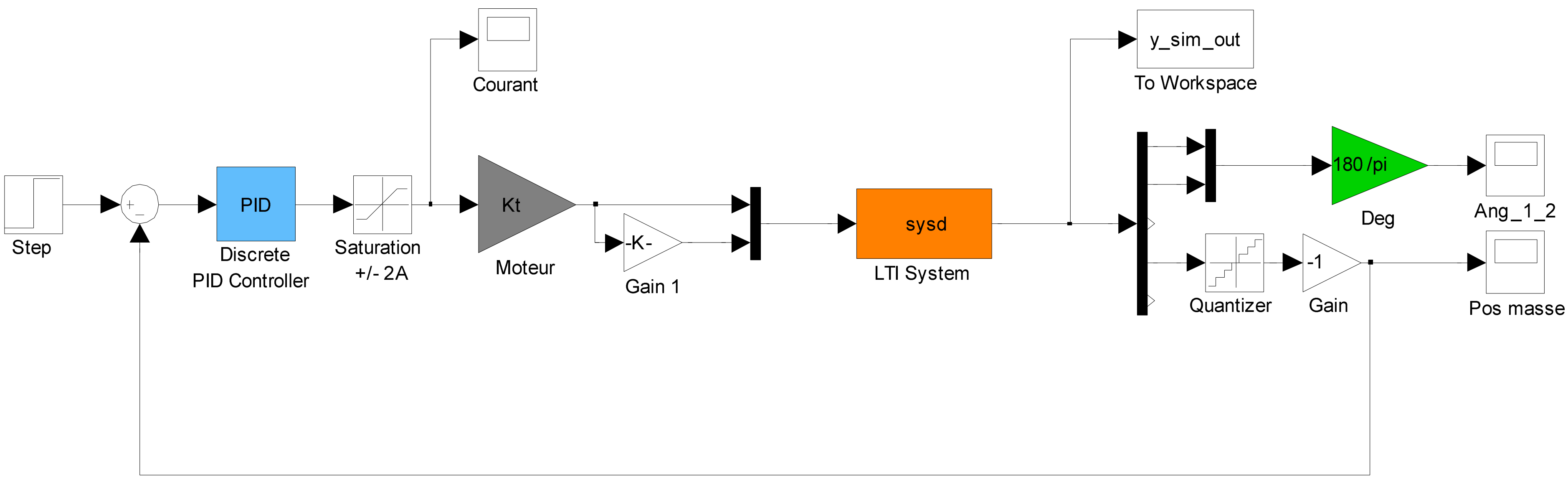

Function Block Parameters: Discrete PID Controller
Discrete PID Controller (mask) (link)
This block implements a discrete PID controller.
Parameters
Proportional gain (Kp):
1500
Integral gain (Ki):
2000
Derivative gain (Kd):
0.3
Time constant for derivative (s):
Ts
Output limits: [Upper Lower]
[1 -1]
Output initial value:
0
Sample time:
Ts
OK
Cancel
Help
Apply


Ang_1_2
6
5
4
3
2
1
0
-1
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time offset: 0

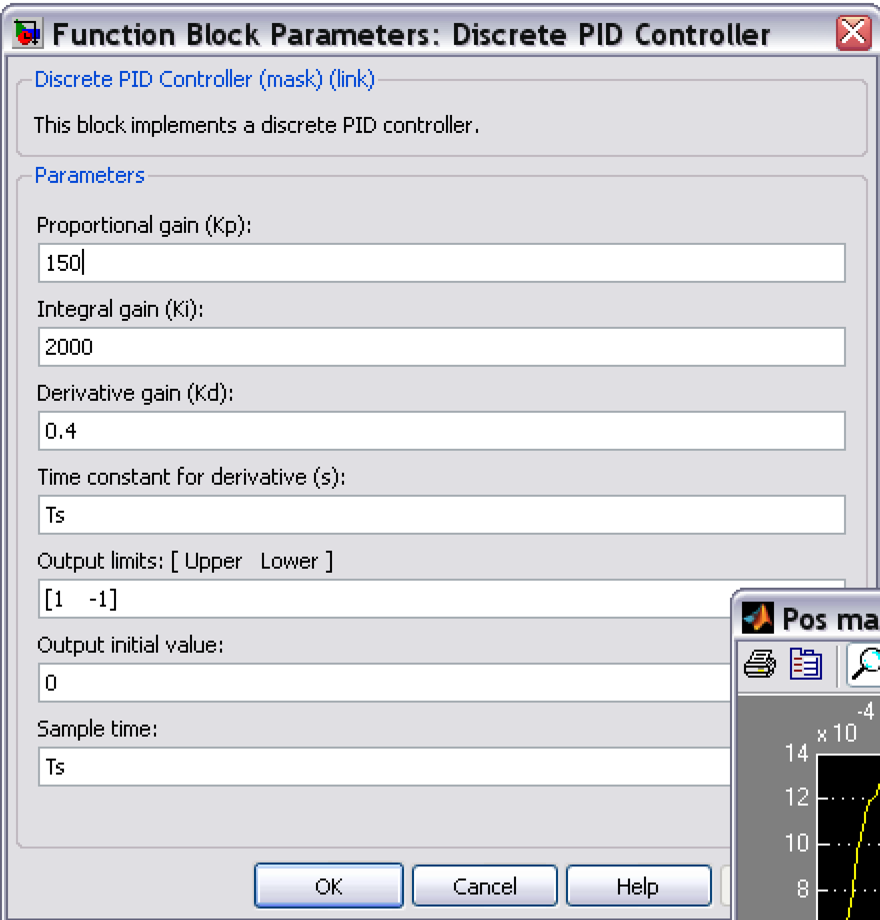

Pos masse
Autoscale
x 10
-3
1.5
1
0.5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time offset: 0

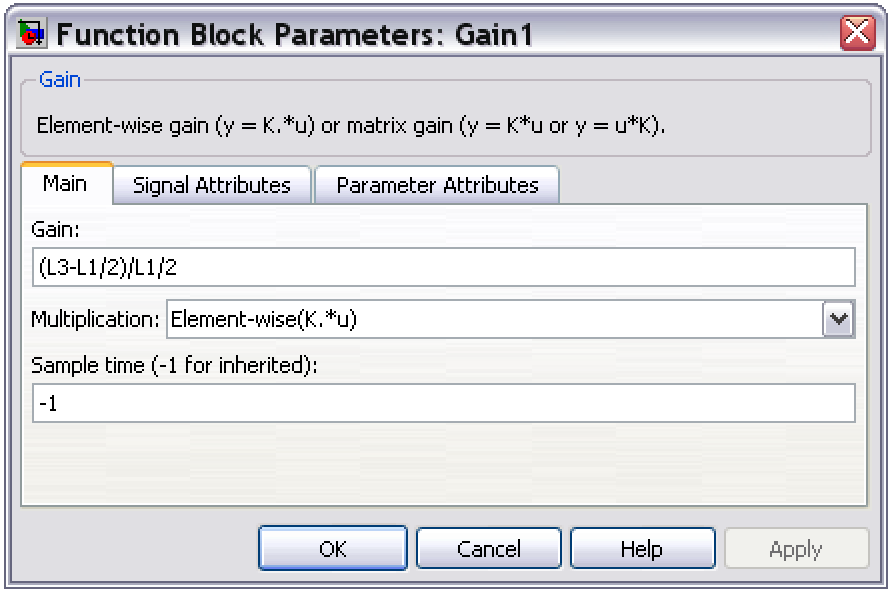
Simulink

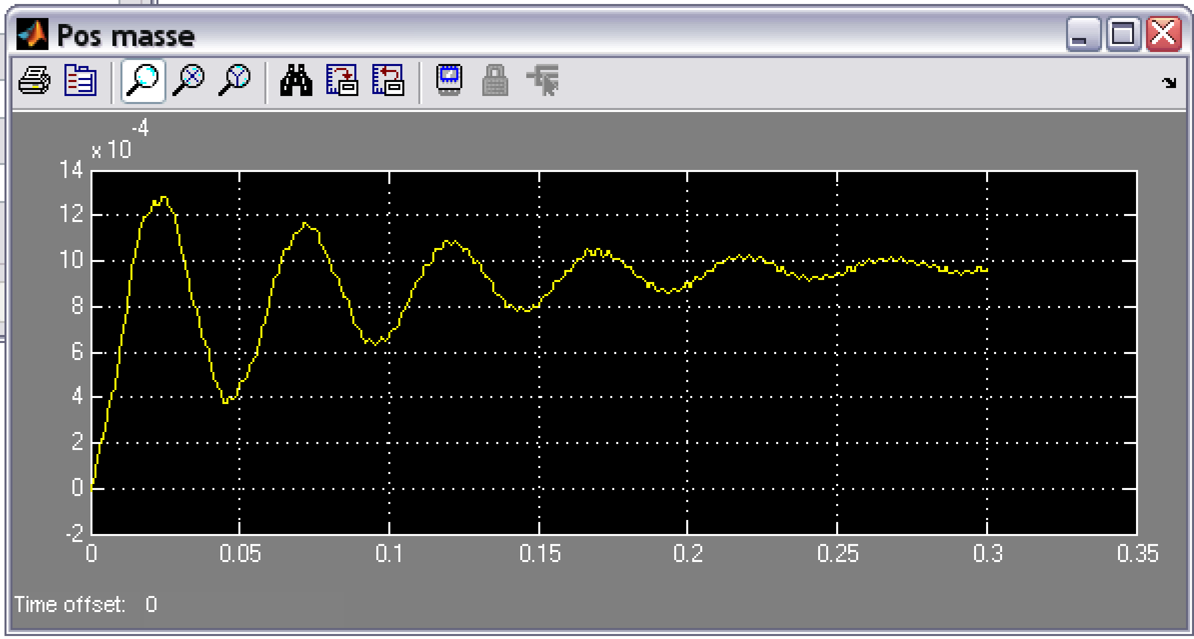
3: Control of **both motors**,
sensor on the central mass *non co-located*



A single controller for both motors !

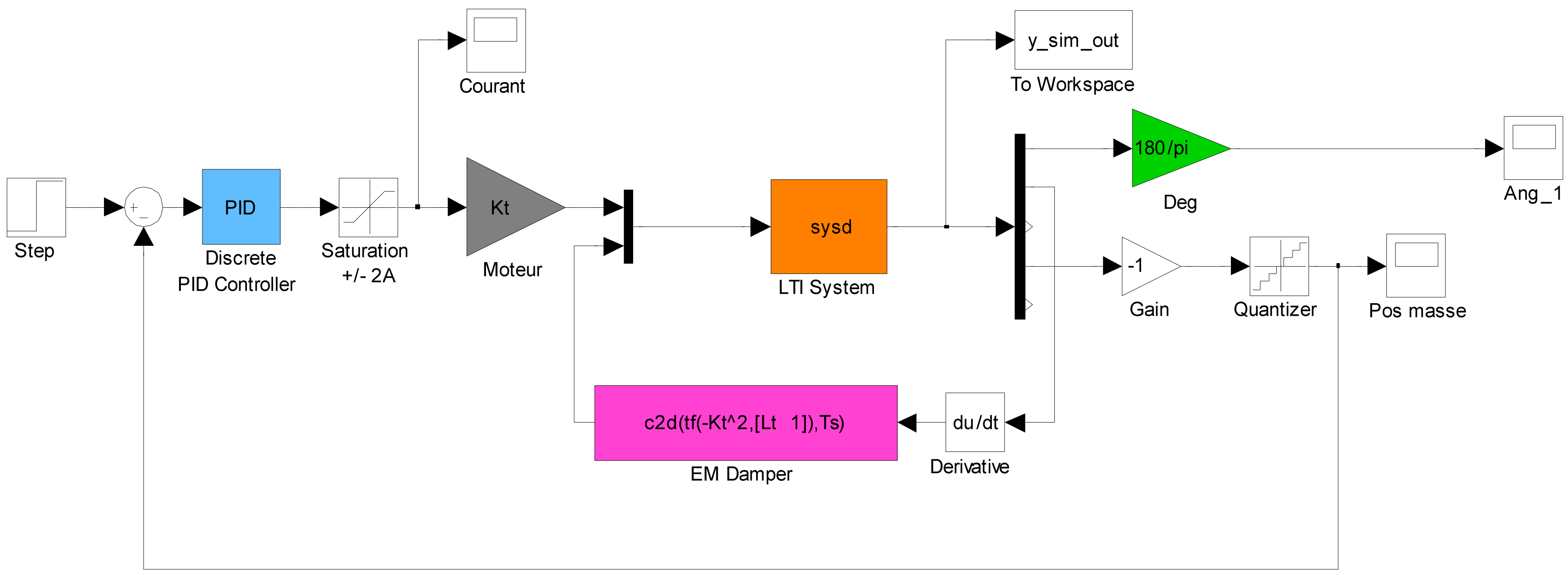

Function Block Parameters: Discrete PID Controller
Discrete PID Controller (mask) (link)
This block implements a discrete PID controller.
Parameters
Proportional gain (Kp):
150
Integral gain (Ki):
2000
Derivative gain (Kd):
0.4
Time constant for derivative (s):
Ts
Output limits: [Upper Lower]
[1 -1]
Output initial value:
0
Sample time:
Ts
OK
Cancel
Help

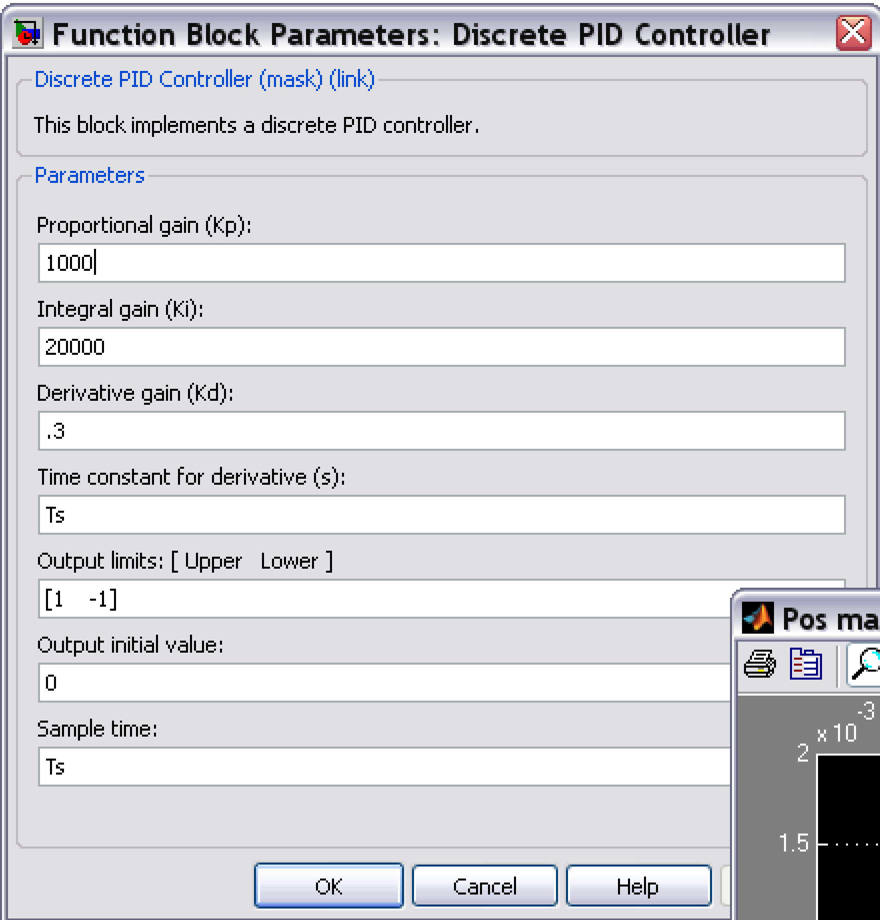

Function Block Parameters: Gain1
Gain
Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).
Main
Signal Attributes
Parameter Attributes
Gain:
(L3-L1/2)/L1/2
Multiplication: Element-wise(K.*u)
Sample time (-1 for inherited):
-1
OK
Cancel
Help
Apply


Pos masse
x 10
-4
14
12
10
8
6
4
2
0
-2
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time offset: 0

Simulink

4: Control of the first motor,
sensor on the mass *non co-located,*
passive electromagnetic damping with the second motor





Transfer function (torque / angular_velocity) for a passive EM damper:

$$TF = \frac{K_T^2}{s\,L_{coil} + R}$$