State Feedback Controller Design

Ing. Dipl. ETH Roberto Bucher

April 18, 2010

1 The plant

1.1 Identification

Figures 1 shows the real plant to be controlled and its driver.

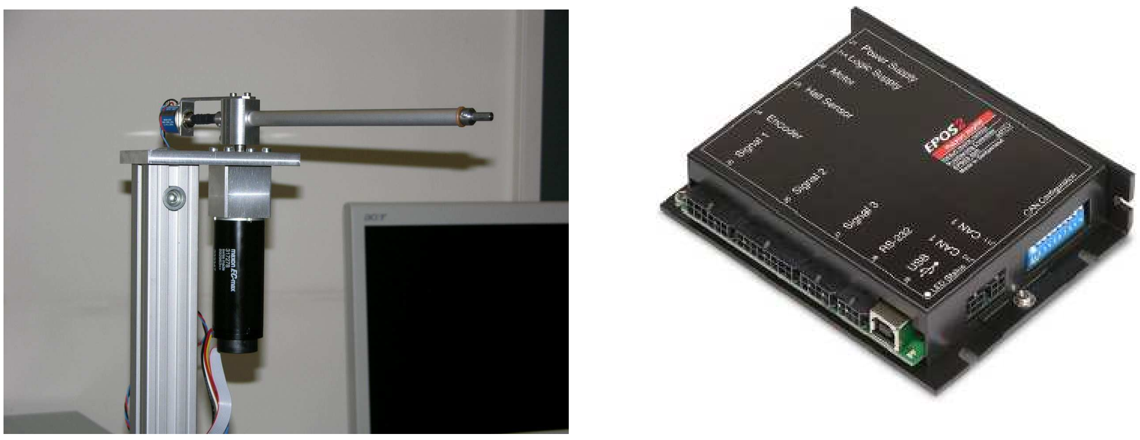


Figure 1: Motor and driver

For state-feedback we need to have our plant in discrete state space form.
We first model our plant as continuous time model, using a Laplace transform representation. Then we can transform the system in the discrete form.
First we create the realt plant using the CAN blocks in ScicosLab (figure 2)

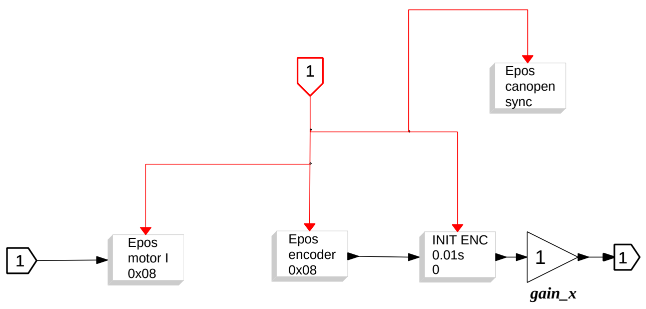


Figure 2: Real plant realized using ScicosLab

In order to find the continous-time model we can for example identify our motor using a simple step response. We collect the response by a step of $500mA$.
Using the Newton laws we can find the mathematical model of our motor and driver:

$$J \cdot \ddot{\varphi} = -D \cdot \dot{\varphi} + K_t \cdot I_a$$

where J is the motor+last inertia, D id the motor friction, K_t is the motor torque constant and I_a is the armature current.
The Laplace transform of the motor becomes

$$G(s) = \frac{\frac{K_t}{J}}{s^2 + \frac{D}{J} \cdot s} = \frac{K}{s \cdot (s + \alpha)}$$

with $K = K_t/J$ and $\alpha = D/J$.
Figure 3 shows the response of the system by a step of $500mA$.

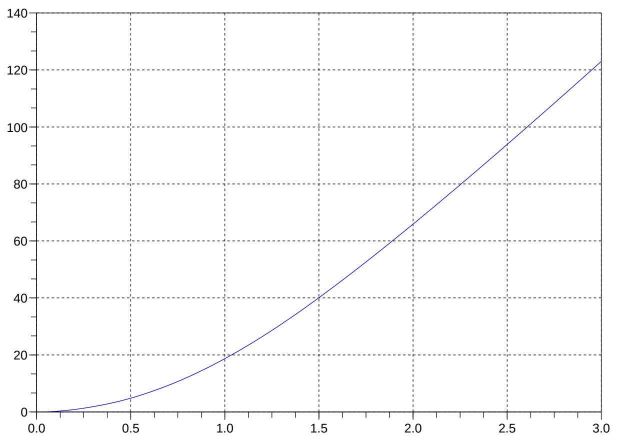


Figure 3: Step response of the plant

The time response of the motor can be found using the inverse Laplace transform of the motor which is

$$y(t) = \mathcal{L}^{-1}[Y(s)] = \mathcal{L}^{-1}[\frac{500}{s} \cdot \frac{K}{s \cdot (s + \alpha)}]$$

$$y(t) = -\frac{500 \cdot K}{\alpha^2} + \frac{500 \cdot K}{\alpha} \cdot t + \frac{500 \cdot K}{\alpha^2} e^{-\alpha \cdot t}$$

The values of K and α can be foundusing the “leastsq” function in Scicoslab.

```
function z=fun(p,t,y)
z=y+p(1)/p(2)^2-p(1)/p(2)*t-p(1)/p(2)^2*exp(-p(2)*t);
endfunction

p0=[1,4];
Uo=500;

[ff,p]=leastsq(list(fun,t,y/Uo),p0);
```

The transfer function is

$$G(s) = \frac{\Phi(s)}{I_a(s)} = \frac{0.0965319}{0.6558342s + s^2}$$

We can compare the identified model and the collected data (see figure 4).

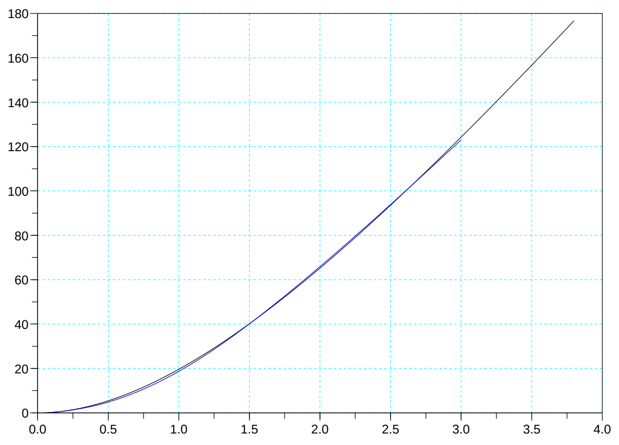


Figure 4: Step response of the plant and of the identified model

The value of K_t can be found in the motor data sheet: $K_t = 126e - 6$; then $J = 0.001365039976$ and $D = 0.000351149780$. From the data sheet we get the maximal motor current ($1670mA$) too.

1.2 The ScicosLab model

```
// Motor parameters
Kt = 0.000126;              // Motor torque constant
Jm = 0.001365039976;        // Motor-Last Inertia
Dm = 0.000351149780;        // Motor-Last damp
Isat = 1670;                // saturation current [mA]

// Continuous-time model of system
a=[0,1;0,-Dm/Jm];
b=[0;1]
c=[Kt/Jm,0];
d=0;
g_ss=syslin('c',a,b,c,d);
Ts=1e-3;                    // Sampling time [s]
g_dss=dscr(g_ss,Ts);        // discrete state space form
```

For the Simulation we can take advantage of the capability of ScicosLab in handling modelica systems. The motor can be represented with the block of figure 5.

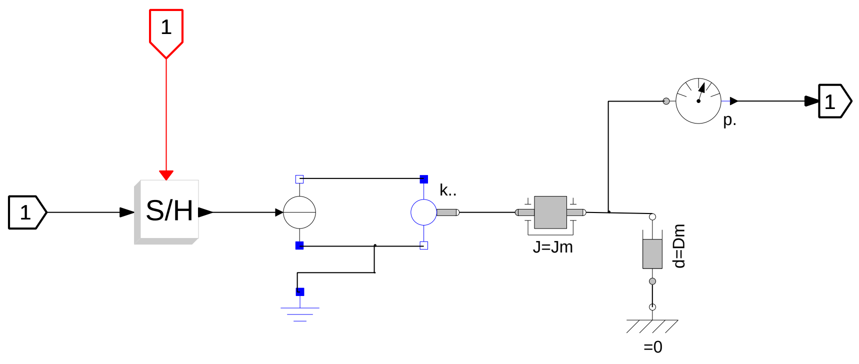


Figure 5: Modelica model of the motor

2 State feedback controller design

2.1 Controller with precompensation

We have a 2. order plant. A state-feedback with precompensation need the placement of two poles in the discrete domain.
The result of the pole placement are the gains k for each state of the plant.

```
// Control design
// Definition of closed-loop poles
wn = 10;
xi = 0.9;

s=%s;

cl_poly = s^2+2*xi*wn*s+wn^2;
cl_poles = roots(cl_poly);
cl_poles_d = exp(cl_poles*Ts);

[A,B,C,D]=abcd(g_dss);
ny=size(C,1);
nx=size(A,1);
nu=size(B,2);
no=nx;
// State-feedback gains without integral part
K=ppol(A,B,cl_poles_d);
```

We can calculate the precompensation gain

```
//Precompensation
Adnew=A-B*K;
Bdnew=B;
Cdnew=C;
Ddnew=D;

Gzctr=syslin('d',Adnew,Bdnew,Cdnew,Ddnew);
u=ones(1:1000);
y=dsimul(Gzctr,u);

Kpregain=1/y($);
```

2.2 Controller with integrator

If we desire to eliminate the steady-state error we can add an integrator at the input of the controller. In this case we get an additional state an for the controller and we need an additional pole for the pole placement.

```
// Poles for state feedback with integrator
policar = (s+wn)*(s^2+2*xi*wn*s+wn^2);
p_c=roots(policar);

p_d=exp(p_c*Ts);
[Phi,G,C,D]=abcd(g_dss);

[m1,n1]=size(Phi);
[m2,n2]=size(G);
Phi_f=[Phi,zeros(m1,1);-C(1,:)*Ts,1];
G_f=[G;zeros(1,n2)];

// Pole placement
k=ppol(Phi_f,G_f,p_d);
```

3 Observer

3.1 Basics

In our plant it is not possible to measure both the states of the plant. We have to estimate them using an observer. There are two possibilities to implement an observer:

- A full order observer
- A reduced order observer

The choice of the observer is completely independent on the choice of the controller.
In the first case we estimate both the states of the plants and we use them to perform out state feedback. In the second case, we use the output of the plant to extract one of the states. Only one state is estimated in this case.
The advantage of this solution is that we have less calculation in our observer, but we have the disadvantage that the measured state can be affected by the measure noise.
In both case the main idea is to have the estimator with a dynamic quicker as the controlled plant. This can be achieved by placing the continuous poles of the observer more links that the poles of the controlled plants (the poles chosen for the pole placement).

3.2 Full order observer

In this case we must provide two poles for the pole placement of the observer.

```
// Definition of poles for full-order observer
p_oc=10*(p_c(1:2));
p_od=exp(p_oc*Ts);

// Design of the full-order observer
[Ao,Bo,Co,Do]=fullobs(Phi,G,C,D,p_od);
```

The script needs the function “fullobs.sci” to calculate the observer.

```
function [A_f, B_f, C_f, D_f]=fullobs(A,B,C,D,poles)
// Find the full order observer for the system A,B,C,D, with observer poles
"poles"

L = ppol(A',C',poles)'

A_f = A-L*C;
B_f = [B-L*D,L];

[m1,n1]=size(A_f);
[m2,n2]=size(B_f);

C_f = eye(m1,n1);
D_f = zeros(m1,n2);

endfunction
```

3.3 Reduced order observer

In this case we have to provide only a pole for the unknown state which must be estimated.

```
// Definition of poles for reduced-order observer
p_oc=[10*p_c(3)];
p_od=exp(p_oc*Ts);
T=[0,1];
[Ao,Bo,Co,Do]=redobs(Phi,G,C,D,T,p_od);
```

The script calls the function "redobs.sci".

```
function [A_redobs, B_redobs, C_redobs, D_redobs]=redobs(A,B,C,D,T,poles)
// Find the reduced order observer for the system A,B,C,D,
// with observer poles "poles"
// T is the matrix needed to get the pair [C;T] invertible

P=[C;T]
invP=inv([C;T])

AA=P*A*invP

ny=size(C,1)
nx=size(A,1)
nu=size(B,2)
nn=nx-ny;

A11=AA(1:ny,1:ny)
A12=AA(1:ny,ny+1:nx)
A21=AA(ny+1:nx,1:ny)
A22=AA(ny+1:nx,ny+1:nx)

L1=ppol(A22',A12',poles)';

A_redobs=[-L1 eye(nn,nn)]*P*A*invP*[zeros(ny,nn); eye(nn,nn)];
B_redobs=[-L1 eye(nn,nn)]*[P*B P*A*invP*[eye(ny,ny);
            L1]]*[eye(nu,nu) zeros(nu,ny); -D, eye(ny,ny)];
C_redobs=invP*[zeros(ny,nx-ny);eye(nn,nn)];
D_redobs=invP*[zeros(ny,nu) eye(ny,ny);zeros(nx-ny,nu) L1]*[eye(nu,nu)
zeros(nu,ny);
           -D, eye(ny,ny)];
```

```
endfunction
```

4 Implementation - The compact form

4.1 Basics on compact form realization

Figure 6 shows the plant (yellow) with the observer (cyan), the integrator (green) and the state feedback(red).

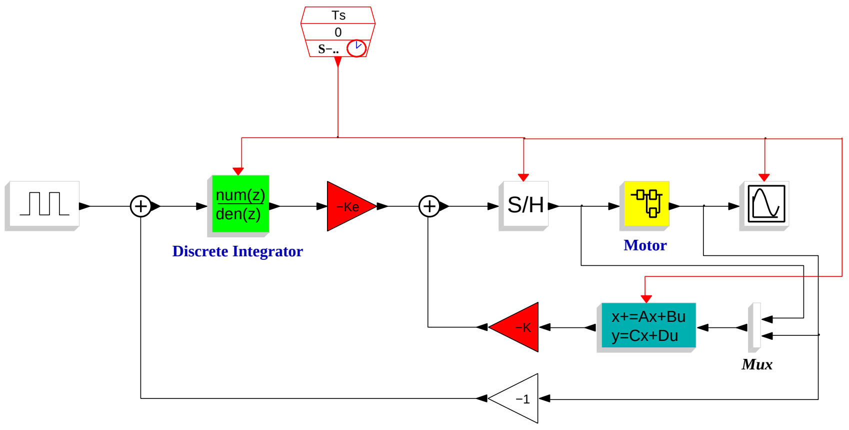


Figure 6: State feedback with integrator and observer

One of the problems in the representation of figure 6 is that the control signal $u(t)$ depends on the output of the plant and of the control signal $u(t)$ self. This implementation leads to an "algebraic loop" in the building of the final controller.A solution is to create a dynamic system from the "Controller-Observer-Integrator" part where the control signal $u(t)$ appears only as output. The resulting block has two inputs (reference signal $r(t)$ and plant output $y(t)$) and one output (control signal $u(t)$).
Figure 7 shows the system with the controller in compact form.

4.2 Controller with precompensation

For this case we need the following values:

- Observer matrices Ao, Bo, Co, Do.
- State feedback gains in the vector K

```
obs = syslin(Ts,Ao,Bo,Co,Do);
Contr=comp_form(g_dss,obs,k);
Contr(7)=Ts;
```

The "comp_form.sci" function contains the following code

```
function [Contr]=comp_form(sys,obs,K)
// Create the compact form of the Observer ABCD and the
// gain K,
//
```

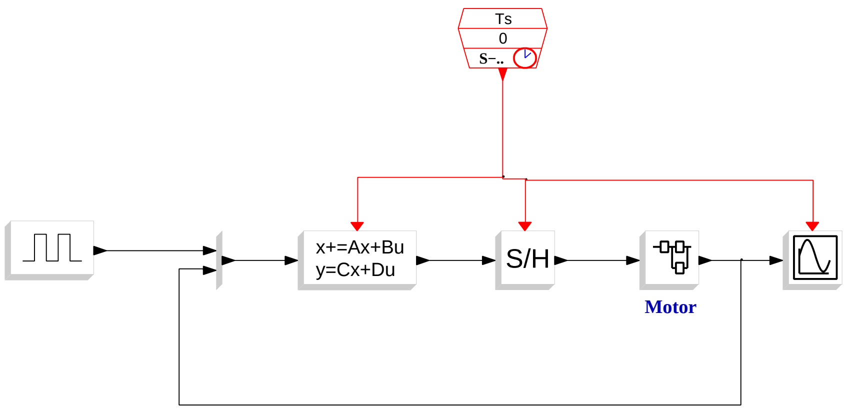


Figure 7: State feedback in compact form

```
// sys: system
// obs: Observer
// K: state feedback gains

Ts = sys(7);

nx = size(sys.A,1);    // states
ny = size(sys.C,1);    // outputs
nu = size(sys.B,2);    // inputs
no = size(obs.A,1); // observed states

Bu = obs.B(:,1:nu);
By = obs.B(:,nu+1:$);
Du = obs.D(:,1:nu);
Dy = obs.D(:,nu+1:$);

X = inv(eye(nu,nu)+K*Du);

Ac = obs.A-Bu*X*K*obs.C;
Bc = [Bu*X,By-Bu*X*K*Dy]
Cc = -X*K*obs.C;
Dc = [X,-X*K*Dy]
Contr = syslin(Ts,Ac,Bc,Cc,Dc)

endfunction
```

4.3 Controller with integrator

For this case we need the following values:

- Observer matrices Ao, Bo, Co, Do.
- State feedback gains K with the gains including Ke.
- The vector q to choose which state is used for the integral error of the controller (not needed if the plant has a single output).
- The sampling time Ts.

```
obs = syslin(Ts,Ao,Bo,Co,Do);
Contr=comp_form_i(g_dss,obs,k);
Contr(7)=Ts;
```

The scripts call the function “comp_form_i.sci”

```
function [Contr]=comp_form_i(sys,obs,K,Cy)
// Create the compact form of the Observer ABCD and the
// gain K, using an integrator at the input to eliminate the
// steady state error
//
// sys: System
// obs: Observer
// K: state feedback gains
// Cy: matrix to extract the output for the steady state feedback

[larg,rarg]=argn(0);

if rarg ~= 7 then
        Cy = [1];
end

Ts = sys(7);

ny = size(sys.C,1);    // outputs
nu = size(sys.B,2);    // inputs
nx = size(sys.A,1);    // states
no = size(obs.A,1);    // observed states
ni = size(Cy,1);       // intgrated signals

B_obsu = obs.B(:,1:nu);
B_obsy = obs.B(:,nu+1:nu+ny);
D_obsu = obs.D(:,1:nu);
D_obsy = obs.D(:,nu+1:nu+ny);

Ke = K(:,$-ni+1:$);
K = K(:,1:$-ni);
X = inv(eye(nu,nu)+K*D_obsu);

A_ctr = [obs.A-B_obsu*X*K*obs.C, -B_obsu*X*Ke;   zeros(ni,no), eye(ni,ni)];
B_ctr = [zeros(no,ni) -B_obsu*X*K*D_obsy+B_obsy; eye(ni,ni)*Ts -Cy*Ts];
C_ctr = [-X*K*obs.C  -X*Ke];
D_ctr = [zeros(nu,ni) -X*K*D_obsy];

Contr = syslin(Ts,A_ctr,B_ctr,C_ctr,D_ctr);
endfunction
```

5 Anti-windup

5.1 Implementation

The control signal $u(t)$ is normally saturated. In order to avoid problems with the integration part of the controller, due to the saturation of the control signal, an anti-windup mechanism can be implemented.
The compact form of the controller+observer+integrator in the two case of full observer and reduced order observer have different order.

- The controller in compact form in the case of the full order observer (ctr_fobs) is a 3. order system.
- The controller in compact form in the case of the reduced order observer (ctr_robs) is a 2. order system.

In the following example we set a filter for the anti-windup with 2 (3) discrete poles at 0.1.

```
// Anti windup for full order observer with poles at 0.1
[in,fbk] = set_aw(Contr,[0.1,0.1,0.1]);
gss_in=tf2ss(in);
gss_fbk=tf2ss(fbk);
```

or

```
// Anti windup for reduced order observerwith poles at 0.1
[in,fbk] = set_aw(Contr,[0.1,0.1]);
gss_in=tf2ss(in);
gss_fbk=tf2ss(fbk);
```

The function “set_aw.sci” looks as follow

```
function [in,fbk]=set_aw(Contr,p)
// Calculate system for anri-windup
// from controlle Contr

[larg,rarg]=argn(0);

[n,m]=size(Contr.A);
den=poly(spec(Contr.A),'z')
if rarg == 1 then
  tmp = syslin('d',den/z^n);
else
  d = poly(p,'z')
  tmp = syslin('d',den/d);
end

id=eye(n,m);

g_contr=Contr.C*inv(z*id-Contr.A)*Contr.B+Contr.D;

//g_contr=ss2tf(Contr)
fbk=syslin(Contr(7),1-tmp);
in =syslin(Contr(7),g_contr*tmp);

endfunction
```

Figure 8 shows the resulting ScicosLab block diagrams for both cases.
Figure 9 shows the result of the simulation.

6 Feed-forwarding

6.1 Controller with integrator

The basic representation of a system with integrator and feed forward compensation is shown in figure 10.
The compact form of the compensator with integrator and feed forward is represented in figure 11.
Now it is possible to recalculate from the block diagram in figure 10 the anti-windup controller.

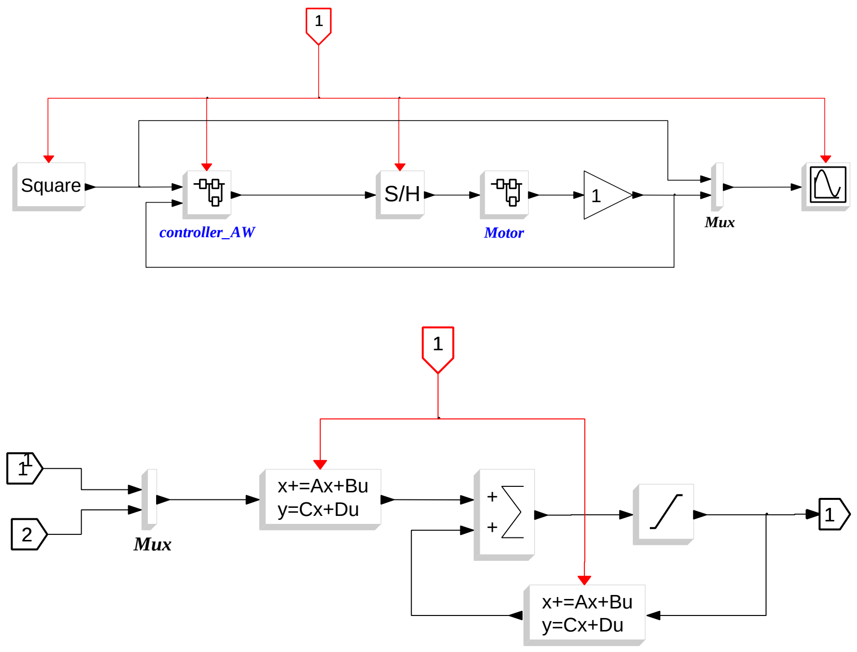


Figure 8: System with AW controller (top) and particular of the anti windup controller (bottom)

The signal $u(t)$is now

$$u(t) = F2 \cdot r - F1 \cdot G2 \cdot r + G2 \cdot y$$

where

- $G1 = U(z)/R(z)$
- $G2 = U(z)/Y(z)$

```
// Feed forward compensation
wn_ff = 20;
F1 = syslin('c',wn_ff^2/(s^2+2*wn_ff*s+wn_ff^2));
F2 = F1/gc;
F1z = ss2tf(dscr(F1,Ts));
F2z = ss2tf(dscr(F2,Ts));

// Feedforward compensation + state feedback + reduced or full order observer
[a,b,c,d]=abcd(Contr);
[m,n]=size(a);
id=eye(m,n);
Gc_trf = ss2tf(Contr);
G2_ff = Gc_trf(:,2);
G_ff =G2_ff;
ctr_ff = tf2ss(G_ff);
ctr_ff = minss(ctr_ff);
[m,n]=size(ctr_ff.A);
p=0.1*ones(1,n);
[in,fbk] = set_aw(ctr_ff,p);
gss_ff_in = tf2ss(in);
gss_ff_fbk = tf2ss(fbk);
```

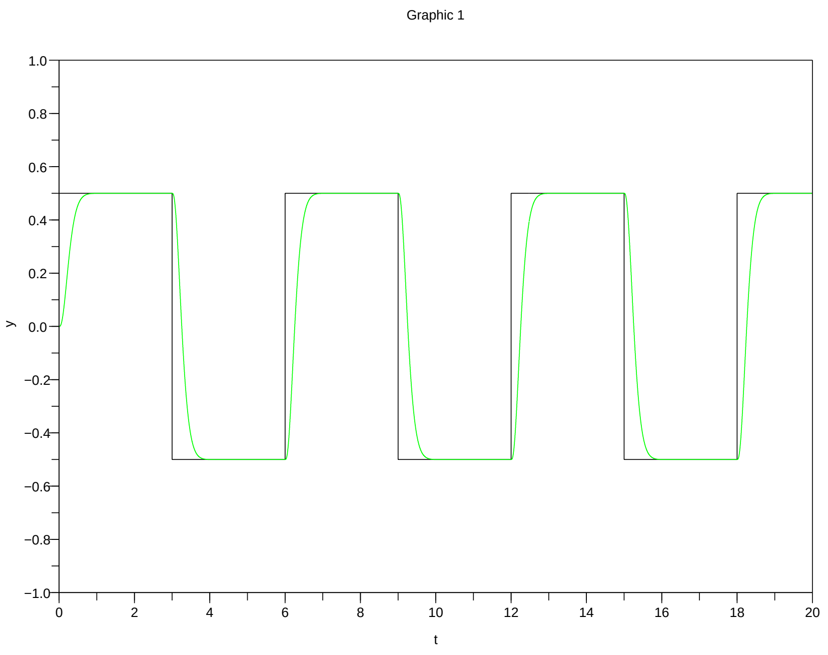


Figure 9: Simulation of the controlled motor with state feedback, integrator, observer and anti windup

The resulting Scicos block diagram is represented in figure 12.
Figure 13 shows the result of the simulation.

7 Results

7.1 Plots

After designing the controller with state feedback, integrator, reduced order observer and anti windup we van compare the simulation and the real plant response (see Figure 14).

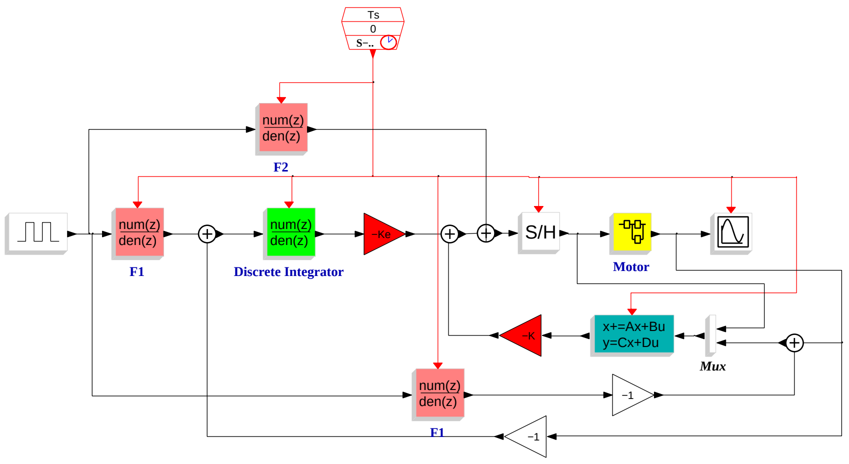


Figure 10: Controller with feed forward compensation

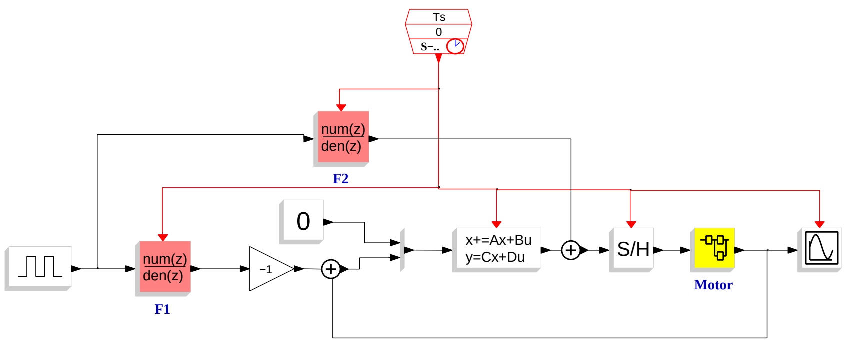
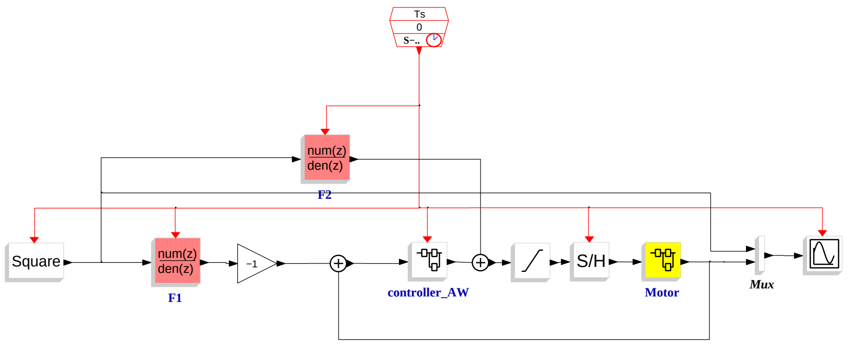


Figure 11: Controller with feed forward compensation in compact form



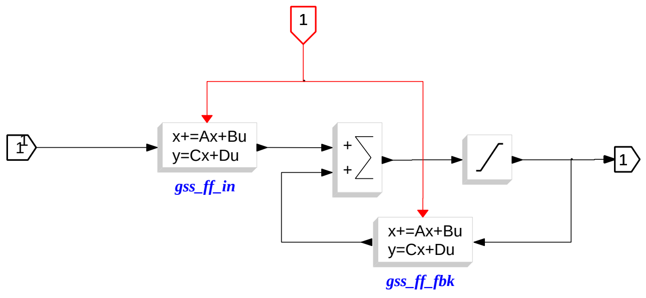


Figure 12: Controller with feed forward compensation in compact form and anti windup - Complete schema (top) and controller (bottom)

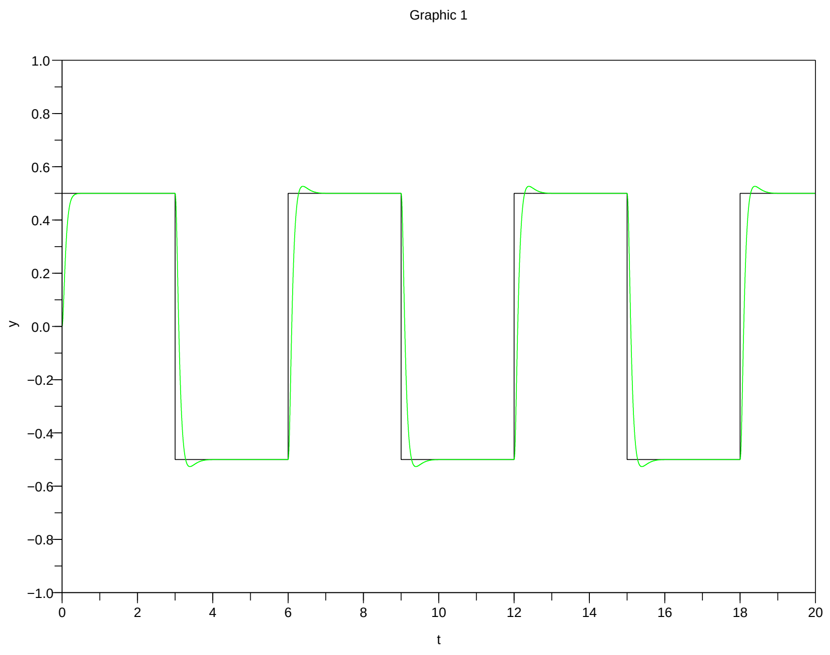


Figure 13: Simulation of the controlled motor with state feedback, integrator, observer and feed forward

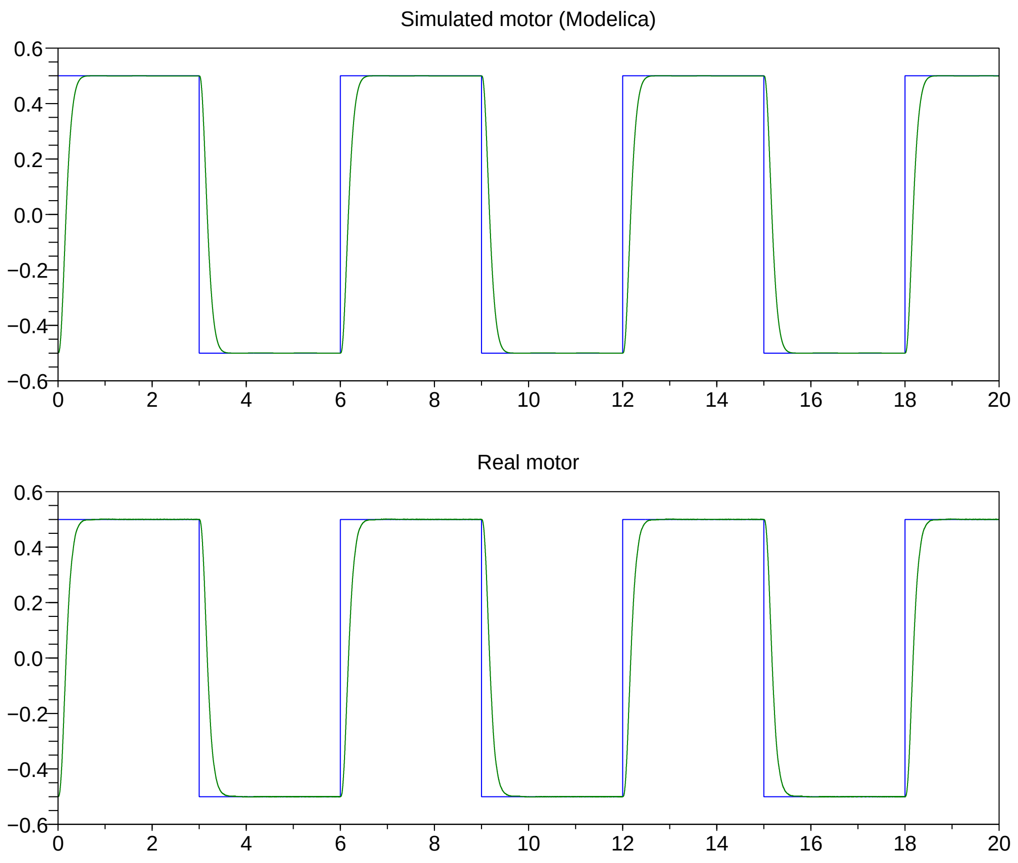


Figure 14: Simulated (top) and real (bottom) data