Control Design for Two Lab-Processes: The Flexible Servo, the Fan and the Plate

Gustafsson, Kjell; Bernhardsson, Bo

1990

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7456)/1–37/(1990)

Control Design for Two Lab-processes: The Flexible Servo The Fan and the Plate

Kjell Gustafsson
Bo Bernhardsson

Department of Automatic Control
Lund Institute of Technology
July 1990

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Internal Report

Date of issue
July 1990

Document Number
CODEN: LUTFD2/(TFRT-7456)/1–37/(1990)

Author(s)
Kjell Gustafsson, Bo Bernhardsson

Supervisor

Sponsoring organisation

Title and subtitle
Control Design for Two Lab-processes: The Flexible Servo, The Fan and the Plate

Abstract

This report describes the control design for two lab processes. Modern facilities make the calculations less time consuming, and it is possible to try a large number of different controllers. We demonstrate how Matlab and realtime Simnon can be used to achieve this end.

Key words
Automatic Control, Education, Control Design, Pole Placement, LQG

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
37

Recipient's notes

Security classification

1. Introduction

New facilities available at the Department of Automatic Control have made controller design less time consuming. These new facilities are both in form of new hardware, Sun workstation with window system, and new software in the form of Matlab routines. When doing a realistic design the calculations are quite extensive, and for the lab processes available at the Department one has normally settled with trying a few different designs. It is now possible to try a large number of different designs quickly, and the practical design procedure hence changes making the iterativeness more pronounced.

Automatic control is great fun and trying it out in practice is even better. This fact has been our main motivation for doing the projects, but since this reason may seem unserious we instead claim: The purpose for this report is manifold. Firstly, it is to serve as documentation for the designs themselves, and, secondly, when doing the designs there are things we have noted and would like to communicate. It is our hope to exemplify how to use Matlab [Gustafsson, Lilja and Lundh] to do the design and then use realtime Simnon to implement it.

A major message is the iterativeness of the design procedure. This fact has often been discussed with the Computer Aided Control Design (CACE) group at the department, and we would like to thank Mats Andersson and Bernt Nilsson for interesting discussions. We have also had a lot of interesting discussions with Per Hagander.

The two projects originate from our teaching assignments. The design for the flexible servo was done as a large demonstration example in the Digital Control Course and the plate and fan was used as one of the projects in Adaptive Control. The two processes are very similar in terms of open loop pole locations; one slow real pole and two poorly damped complex poles. The control designs are, however, rather different. This is a consequence of different types of disturbances, saturation and other nonlinearities and hence different objectives for the design.

2. The Flexible Servo

The control design for the flexible servo was done as a demo in the Digital Control Course. The aim was to exemplify some of the theory and methods presented in [Åström and Wittenmark]. The standard problems solved during the course are simplified to facilitate hand calculation, but with tools like Matlab and Simnon it is possible to attack realistic problems without getting overwhelmed with the calculations. As a consequence one can concentrate on the control design and try many different controllers. Recently, a set of Matlab routines [Gustafsson, Lilja, and Lundh] have been implemented that further simplifies the task. As will be seen in the sequel we will rely heavily on these routines.

Any realistic design procedure is iterative. The designer tries many different possible designs and in the process he/she builds up a knowledge of the essential properties of the system. This knowledge is then used to deduce reasonable specifications and necessary properties of the controller. It is important to have tools that make the iteration fast. A fast iteration makes it possible to evaluate many different designs, and one may hope for a better final result.

The design itself is described in the slides at the end of this section. As can be seen the design process is iterative and we will here describe how Matlab and Simnon was used to organize the design calculations.

The Process Model

The design was done in Matlab using a linear process model. The real process includes some nonlinear effects (static friction, control signal saturation) and each design was therefore evaluated in Simnon using a full nonlinear model. Both the linear and the nonlinear model were derived from first principles with unknown parameters determined from component data and through practical experiments [Andersson].

All data concerning the model was gathered in one file: `servomod.m`. When called, this routine calculates the state space and polynomial form of the model, as well as other model dependent parameters needed in the design.

```
function [A,B,C,D,b,a,J2] = servomod
% SERVOMOD defines matrices and polynomials for the flexible servo
%
% [A,B,C,D,b,a,J2] = servomod

% Flexible servo constants, compare with files servo.t and nlservo.t

J1 = 22E-6;
J2 = 65E-6;
kf = 11.7E-3;
df = 2E-5;
d1 = 1E-5;
d2 = 1E-5;

km = 0.1;
ki = 0.25;
kw1 = 0.1;
kw2 = 0.1;

kvin = 1/(1+10);      % Resistive voltage divider on input
kvout = 20/(20+100);  % Resistive voltage divider on output

% State space representation

A = [-(d1+df)/J1   df/J1        -kf/J1
      df/J2       -(d2+df)/J2   kf/J2
      1           -1            0     ];

B = kvin*[km*ki/J1 0 0]';

C = kvout*[kw1 0 0; 0 kw2 0];

D = [0; 0];

% Transfer function

[b,a] = ss2tf(A,B,C,D,1);
```

The process model must be extended with the antialiasing filter and a disturbance model before it can be used for the design. Since the controller will be discrete-time a sampled process description is also needed. These type of operations are gathered in a script-file called `defmod.m`.

```
% File: defmod.m

% Get servo model

[A,B,C,D,b,a,J2] = servomod;

% Extend system with model for torque disturbance at second mass

A1 = [A [0; -1/J2; 0]; 0 0 0 0];
B1 = [B; 0];
C1 = [C(1,:) 0];
D1 = 0;

% Extend system with antialiasing filter

A1 = [A1 [0; 0; 0; 0]; C1 -50];
B1 = [B1; 0];
C1 = [0 0 0 0 50];
D1 = D1;

% Now define a second system A2, B2, C2, D2 with x4 excluded .
% We need this when doing the LQG design.

A2 = A1([1,2,3,5],[1,2,3,5]);
B2 = B1([1,2,3,5],:);
C2 = C1(:,[1,2,3,5]);
D2 = D1;

% Sample system

h = 0.02;

[Phi1,Gam1] = c2d(A1,B1,h);
[Phi2,Gam2] = c2d(A2,B2,h);

% Calculate polynomial description of system

[b1d,a1d] = sample(b(1,:)*50,conv(a,[1 50]),h);
[b2d,a2d] = sample(b(2,:),a,h);
```

The Specifications

The designer, normally, has an idea of how the closed loop system should behave. Even if these specifications would not change during the design, their translation to quantitative figures, such as a loss function or certain pole locations, most certainly will. It is these quantitative figures the designer changes when iterating through different possible controllers. It is therefore convenient to keep them in one place for ease of reference and modification.

Two different design methods where used: LQG and pole placement. These methods are in a way equivalent, but they still have quite different properties as design methods. In the first case the designer chooses a loss function and a noise description while pole locations are specified in the second. Some system properties are more easily described using pole locations than

loss functions (and vice versa), and hence certain properties are more easy to get using one of the methods than the other.

For each one of the two methods a script file containing all specification parameters was created. In the case of the LQG design the file was called `deflqg.m`

```
% File: deflqg.m

% Specifications for the LQG design
% Before running this script you must define rho, lambda, v1, v2.
% Suggested values are:
% rho = 5, lambda = 200, v1 = 10000, v2 = 1E7

% Loss function

Q1 = diag([0 1 lambda 0]);
Q2 = rho;
Q12 = [0 0 0 0]';

% Noise description

R1 = diag([v2 0 0 1 0]);
R2 = v1;
R12 = [0 0 0 0 0]';
```

and, similarly, `defrst.m` for the pole placement design.

```
% File: defrst.m

% Specifications for the RST design
% Before running this script you must define zetam, zetao, alpha, beta,
% and gamma. Suggested values are:
% zetam = 0.6, zetao = 0.3, alpha = 0.8, beta = 0.8, gamma = 0.2

% Construct am

% Start with a polynomial corresponding to the two oscillative modes
% of the servo. Change the damping to 'zetam' but keep the frequency
% unchanged.

am1 = poly(exp(h*27*(-zetam/sqrt(1-zetam^2)*[1 1] + [i -i])));

% Contstruct a polynomial corresponding to the antialiasing filter.

am2 = poly(exp(-h*50));

% Construct am from am1, am2 and add one pole in alpha.

am = conv(am1,conv(am2,[1 -alpha]));

% Construct ao

% Start with a polynomial corresponding to the two oscillative modes
% of the servo. Change the damping to 'zetao' but keep the frequency
% unchanged.

ao1 = poly(exp(h*27*(-zetao/sqrt(1-zetao^2)*[1 1] + [i -i])));

% Add two poles at beta +- i*gamma

ao = conv(ao1,poly(beta*[1 1]+gamma*i*[1 -1]));
```

"Free" parameters and reasonable values are listed in the beginning of the files. As the design iteration progressed the parameter values were adjusted, and served as an online documentation of the current status in the design.

The Design and the Evaluation

After having defined both the model and the parameters depending on the specifications it is rather straightforward to do the design. The design commands where kept in two files: `dolqg.m`

```
% File: dolqg.m

% Design specifications

deflqg

% Do the design

[L2,Lv2,lr] = lqrd(Phi2,Gam2,C2,Q1,Q2,Q12)
[K,Kf,Kv] = lqed(Phi1,C1,R1,R2,R12)

l4 = C2*inv(eye(4)-Phi2+Gam2*L2)*Phi1([1,2,3,5],4)*lr

L1 = [L2(1:3) l4 L2(4)]
Lv1 = [Lv2(1:3) 0 Lv2(4)];   % 0 is probably not the correct choice,
                             % but Kv = 0 so it does not matter in LQGD

% Calculate a controller with direct term

[Lx,Ly,Phic,Gamy,Gamyr,Cc,Dy,Dyr] = lqgd(Phi1,Gam1,C1,L1,Lv1,lr,K,Kf,Kv,1);

% Calculate frequency responses needed in the evaluation

gfb = frdss(Phic,Gamy,Cc,Dy,h,-1,[],200);
gff = frdss(Phic,Gamyr,Cc,Dyr,h,-1,[],200);
gp  = frdss(Phi1,Gam1,C1,D1,h,-1,[],200);
gz  = frdss(Phi1,Gam1,[0 0.0167 0 0 0],0,h,-1,[],200);
gl  = fdiv(frdss(Phi2,Phi1([1,2,3,5],4),C2,0,h,-1,[],200),gp);
gn  = frc(50,[1 50],0,gp(:,1));

% Save controller data on a Simnon parameter file

delete sfpar.t
p2sim('sfpar','reg','l',L1,1,'lr',lr,1)
delete dirpar.t
p2sim('dirpar','reg','l',Lx,1,'lr',lr,1,'ly',Ly,1,'k',K,1)

% Calculate a controller without direct term

[Lx,Ly,Phic,Gamy,Gamyr,Cc,Dy,Dyr] = lqgd(Phi1,Gam1,C1,L1,Lv1,lr,K,Kf,Kv);

% Save controller data on a Simnon parameter file

delete nodirpar.t
p2sim('nodirpar','reg','l',Lx,1,'lr',lr,1,'ly',Ly,1,'k',K,1)

% Useful evaluation commands

% Matlab
% evpl(gp,gfb,gff,gl,gn,gz)
%
% Simnon
```

```
% syst servo aafilt dsf sfcon       " state feedback
% syst servo aafilt dsfka sfkacon   " state feedback with Kalman filter
% evaxes
% doeval
```

and `dorst.m`.

```
% File: dorst.m

% Design specifications

defrst

% Do the design

[r,s,t] = rstd(1,b1d,a1d,1,am,ao,[1 -1],1)

% Calculate frequency responses needed in the evaluation

gfb = frd(s,r,h,-1,[],200);
gff = frd(t,r,h,-1,[],200);
gp  = frdss(Phi1,Gam1,C1,D1,h,-1,[],200);
gz  = frdss(Phi1,Gam1,[0 0.0167 0 0 0],0,h,-1,[],200);
gl  = fdiv(frdss(Phi2,Phi1([1,2,3,5],4),C2,0,h,-1,[],200),gp);
gn  = frc(50,[1 50],0,gp(:,1));

% Save controller data on a Simnon parameter file

delete rstpar.t
rst2sim('rstpar','reg',r,s,t,poly(0.5*ones(1,length(r)-1)),1)

% Useful evaluation commands

% Matlab
% evpl(gp,gfb,gff,gl,gn,gz)
% step = dstep(b2d*sum(am)/sum(b2d),am,50);
%
% Simnon
% syst servo aafilt drst rstcon " RST controller
% evaxes                       " new axes
% doeval                       " step, load and noise response
```

The design was evaluated both in Matlab (different frequency domain plots) and in Simnon (response to step/load change and measurement noise), and to facilitate this both `dolqg.m` and `dorst.m` include commands to calculate interesting frequency responses and commands to save the controller parameters on a form readable by Simnon.

At the end of each of the files `dolqg.m` and `dorst.m` there is a list of useful evaluation commands. By including such a list (as well as including the list of parameter values in `deflqg.m` and `defrst.m`) the evaluation is facilitated. In a mouse-driven environment one only lists the m-file and then fetches the different commands using the mouse.

Implementation

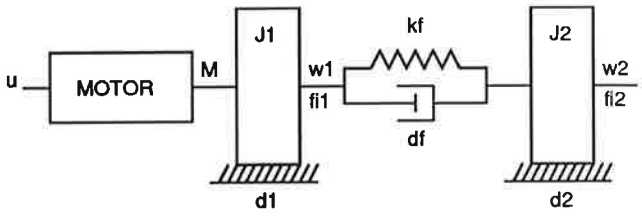
Realtime Simnon provided a very convenient tool for implementing the controllers and testing them on the real process. During the design process all controllers were tried on the nonlinear model using Simnon. It was then a trivial task to convert these files to the realtime environment. In addition, since the Matlab routines where written to automatically save the controller parameters on data files that could be read by Simnon it was easy to save the parameters of interesting controllers for later evaluation on the real process.

Control Design for A Flexible Servo

A practical illustration of some of the concepts in Computer Controlled Systems

Kjell Gustafsson
Department of Automatic Control

A model from first principles



Simplifying assumptions

- neglect sensor dynamics
- neglect motor dynamics
- no static friction
- no disturbances

Torque balance

$$J_1\dot{\omega}_1 = -k_f(\varphi_1 - \varphi_2) - d_1\omega_1 - d_f(\omega_1 - \omega_2) + k_m k_i u$$
$$J_2\dot{\omega}_2 = \quad k_f(\varphi_1 - \varphi_2) - d_2\omega_2 + d_f(\omega_1 - \omega_2)$$

A state space model

Introduce the states

$$x = \begin{pmatrix} \omega_1 & \omega_2 & \varphi_1 - \varphi_2 \end{pmatrix}$$

and regard the angular velocities ω_1 and ω_2 as outputs, then

$$\dot{x}(t) = \begin{pmatrix} -\frac{d_1+d_f}{J_1} & \frac{d_f}{J_1} & -\frac{k_f}{J_1} \\ \frac{d_f}{J_2} & -\frac{d_f+d_2}{J_2} & \frac{k_f}{J_2} \\ 1 & -1 & 0 \end{pmatrix} x(t) + \begin{pmatrix} \frac{k_m k_i}{J_1} \\ 0 \\ 0 \end{pmatrix} u(t)$$

$$y(t) = \begin{pmatrix} k_{\omega_1} & 0 & 0 \\ 0 & k_{\omega_2} & 0 \end{pmatrix} x(t)$$

Some of the parameters are known others need to be measured.

Identification

Frequency analysis: Feed a sinusoid to the process input and measure output.

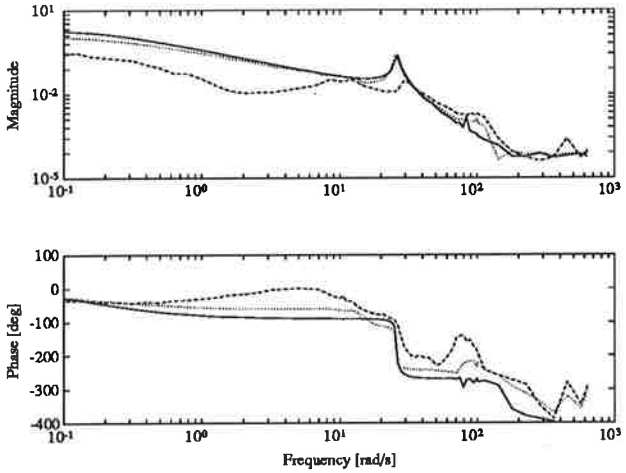
Three different experiment conditions

1. 1.0 Vpp sinusoid and 1.5 V bias
2. 1.5 Vpp sinusoid
3. 3.0 Vpp sinusoid

Observations

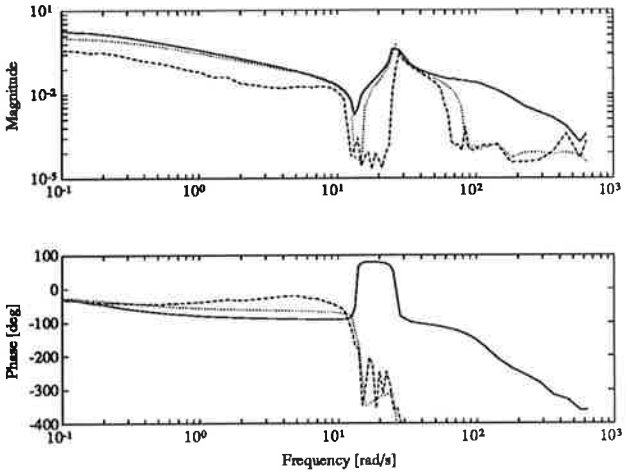
- rather noisy measurements
- sinusoidal measurement disturbance (tacho) corresponding to the angular velocity (0 – 120 rad/s)

Identification (noncolocated sensor)



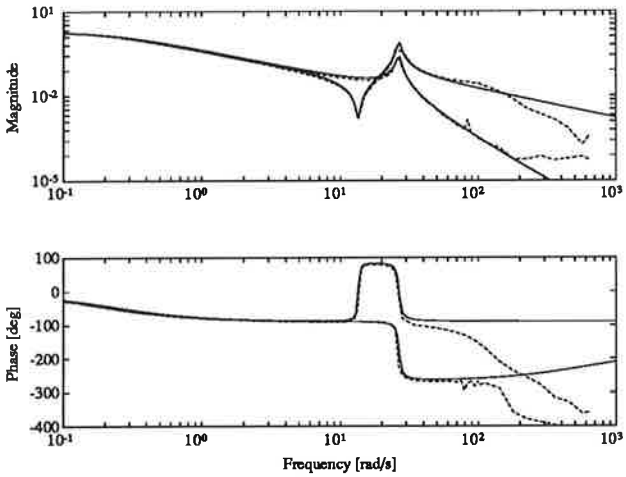
1.0 Vpp sinusoid, 1.5 V bias (full line)
1.5 Vpp sinusoid (dashed line)
3.0 Vpp sinusoid (dotted line)

Identification (colocated sensor)



1.0 Vpp sinusoid, 1.5 V bias (full line)
1.5 Vpp sinusoid (dashed line)
3.0 Vpp sinusoid (dotted line)

Comparison (model – identification)



model (full line)
measured data (dashed line)

Final model

The identification suggest the numerical values

$$\dot{x}(t) = \begin{pmatrix} -1.36 & 0.909 & -532 \\ 0.308 & -0.462 & 180 \\ 1.00 & -1.00 & 0 \end{pmatrix} x(t) + \begin{pmatrix} 103 \\ 0 \\ 0 \end{pmatrix} u(t)$$

$$y(t) = \begin{pmatrix} 0.0167 & 0 & 0 \\ 0 & 0.0167 & 0 \end{pmatrix} x(t)$$

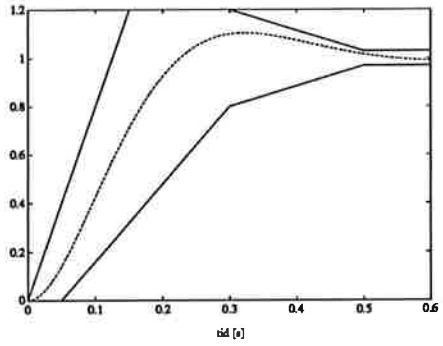
but

- strong static friction
- unmodeled dynamics start at $\approx$ 50 rad/s
- sinusoidal measurement noise
- torque load at second mass

Specifications

Can only measure ω_1 but want to control ω_2.

Want "nice" behavior in spite of friction, load disturbance, and measurement noise. As for step response we interpret nice as



A 0.3 second rise time corresponds to an approximate loop bandwidth of 20 rad/s.

General observations

From the model and the measurements we may deduce

- the friction and the load disturbance requires an integrator in the controller
- the resonance at 25 rad/s is pronounced, must be damped but not moved
- the measurement noise and unmodeled dynamics demands the controller gain to drop as soon as possible
- need controller gain at least up to 25 rad/s to handle resonance and desired step response

Conflicting objectives!

Disturbance model

Assume constant torque affecting the second mass (a reasonable load!). Model this through a state x_4 affecting $\dot{x}_2$ as $1/J_2$

$$A = \begin{pmatrix} -1.36 & 0.909 & -532 & 0 \\ 0.308 & -0.462 & 180 & -15400 \\ 1.00 & -1.00 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 0.0167 & 0 & 0 & 0 \\ 0 & 0.0167 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 103 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

This will force an integrator into the controller.

Sampling interval

CCS recommends

$$\omega h \approx 0.15 - 0.5$$

A reasonable value for ω is 20 rad/s (step response) which leads to

$$h \approx 0.01 - 0.025$$

Unfortunately, the shortest achievable h is 0.02 or 0.03 with our hardware. Tough!

Let's aim for $h = 0.02$.

Anti-aliasing filter

$$\omega_s/2 = \pi/0.02 \approx 160 \text{ rad/s}$$

Use first order filter with $\omega_b = 50$. Would need more filtering but so small margins.

$$H_a(s) = \frac{50}{s+50} \quad \Rightarrow \begin{cases} -10 \text{ dB}, -73^\circ, & \text{at } \omega_s/2 \\ -1 \text{ dB}, -30^\circ, & \text{at resonance} \end{cases}$$

Introduce a new state x_5

$$A = \begin{pmatrix} -1.36 & 0.909 & -532 & 0 & 0 \\ 0.308 & -0.462 & 180 & -15400 & 0 \\ 1.00 & -1.00 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.0167 & 0 & 0 & 0 & -50 \end{pmatrix}$$

$$C = \begin{pmatrix} 0.0167 & 0 & 0 & 0 & 0 \\ 0 & 0.0167 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50 \end{pmatrix} \qquad B = \begin{pmatrix} 103 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Will only measure y_3

LQG-design

Controller

$$u(k) = l_r y_r(k) - L\hat{x}(k)$$

with $\hat{x}(k)$ given by Kalman filter.

1. design L, exclude x_4 since not controllable
2. determine l_4 to eliminate steady state influence from load
3. determine l_r to get correct steady state gain
4. design Kalman filter, direct term
5. connect together, evaluate design

MATLAB, Simnon

No continuous-time loss function

Guestimates of Q_1, Q_{12}, Q_2, R_1, R_{12}, R_2

Sampled model

Need two models one with x_4 and one without

$$\Phi_1 = \begin{pmatrix} 0.8716 & 0.1196 & -9.9709 & -13.356 & 0 \\ 0.0405 & 0.9563 & 3.3852 & -302.69 & 0 \\ 0.0187 & -0.0188 & 0.8624 & 2.9776 & 0 \\ 0 & 0 & 0 & 1.0000 & 0 \\ 0.0002 & 0.0000 & -0.0013 & -0.0010 & 0.3679 \end{pmatrix}$$

$$C_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 50 \end{pmatrix} \qquad \Gamma_1 = \begin{pmatrix} 103 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\Phi_2 = \begin{pmatrix} 0.8716 & 0.1196 & -9.9709 & 0 \\ 0.0405 & 0.9563 & 3.3852 & 0 \\ 0.0187 & -0.0188 & 0.8624 & 0 \\ 0.0002 & 0.0000 & -0.0013 & 0.3679 \end{pmatrix}$$

$$C_2 = \begin{pmatrix} 0 & 0 & 0 & 50 \end{pmatrix} \qquad \Gamma_2 = \begin{pmatrix} 103 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

State feedback

Use Φ_2 and Γ_2 to get $L_2 = \begin{pmatrix} l_1 & l_2 & l_3 & l_5 \end{pmatrix}$

First try:

$$\min \sum x_2^2 + \rho u^2$$

which corresponds to

$$Q_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad Q_{12} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T \qquad Q_2 = \rho$$

Calculate

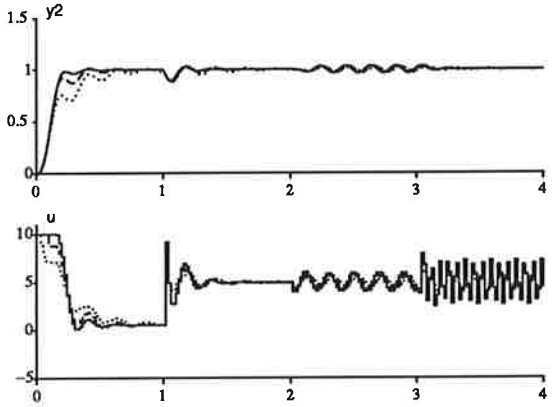
$$l_r = \frac{1}{C_2 (I - \Phi_2 + \Gamma_2 L_2)^{-1} \Gamma_2}$$

$$l_4 = \frac{C_2 (I - \Phi_2 + \Gamma_2 L_2)^{-1} \Phi_1([1,2,3,5],4)}{C_2 (I - \Phi_2 + \Gamma_2 L_2)^{-1} \Gamma_2}$$

Simulation (first design)

Measure all states, use simulation to evaluate

- step response
- load disturbance (torque at second mass)
- noise sensitivity (two sinusoids, 30 rad/s, 100 rad/s)



$\rho = 1$ (full), 5 (dashed), 20 (dotted)

Improved damping

Sufficient speed but bad damping for

$$\rho = 5$$

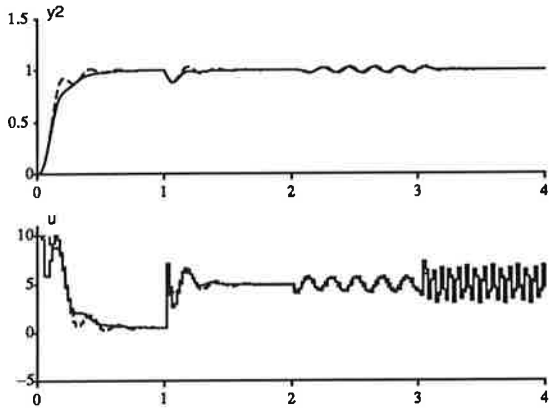
Introduce damping by penalizing x_3, i.e. don't allow too much angular difference

$$Q_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad Q_{12} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T \qquad Q_2 = \rho$$

Note: $x_2 \approx 10 - 50$ rad/s, $x_3 \approx 1 - 3$ rad

$$\Longrightarrow \quad \lambda \approx \left(\frac{10}{1}\right)^2 = 100, \quad \left(\frac{50}{3}\right)^2 = 280$$

Simulation (improved design)



$\rho = 1$, $\lambda = 200$, (full)

$\rho = 1$, $\lambda = 0$, (dashed)

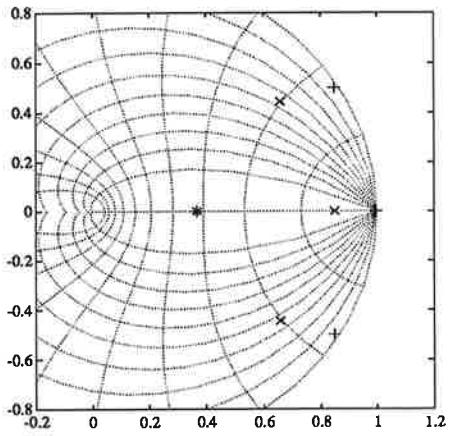
Final control law

Final control law

$$u(k) = 20.00\, l_r - L\,\hat{x}(k)$$

$$L = \begin{pmatrix} 0.2424 & 0.0803 & 2.3049 & -637.0 & 0 \end{pmatrix}$$

Closed (x) and open (+) loop poles



Kalman filter

Process

$$x(k+1) = \Phi x(k) + \Gamma u(k) + v(k)$$

$$y(k) = Cx(k) + e(k)$$

Kalman filter

$$\hat{x}(k+1|k) = (\Phi - KC)\,\hat{x}(k|k-1) + \Gamma u(k) + Ky(k)$$

$$u_{nodir}(k) = -L\hat{x}(k|k-1)$$

$$u_{dir}(k) = -\{L(I - K_f C) + L_v K_v C\}\,\hat{x}(k|k-1) - \{LK_f + L_v K_v\}\,y(k)$$

Noise description

$$E\,vv^T = R_1, \quad E\,ve^T = R_{12}, \quad E\,ee^T = R_{12},$$

gives K, K_f, and K_v through Riccati equation.

Noise description

First try:

Load disturbance

$$R_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

High pass noise (in reality sinusoids) at output y_1 is modeled as white noise after anti-aliasing filter

$$R_2 = \nu_1$$

Also

$$R_{12} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T$$

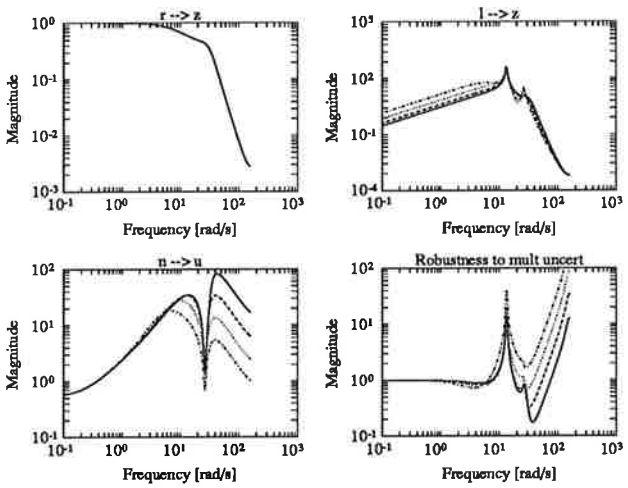
Note: $x_4 \approx 1$ Nm $\iff$ $y_1 \approx 30$ V

$$\Longrightarrow \quad \nu_1 \approx \left(\frac{30}{1}\right)^2 = 900$$

Kalman filter evaluation

Frequency domain evaluation of Kalman filter

- z – angular velocity of second mass
- l – torque load at second mass
- n – measurement noise at first mass

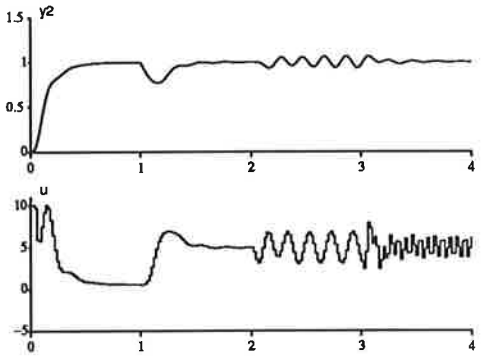


$\nu_1 = 10^2$, (full) $\nu_1 = 10^4$, (dotted)

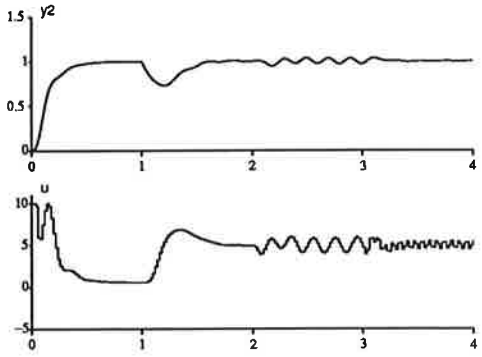
$\nu_1 = 10^3$, (dashed) $\nu_1 = 10^5$, (dash-dotted)

Simulation (Kalman + state feedback)

$\nu_1 = 10^3$

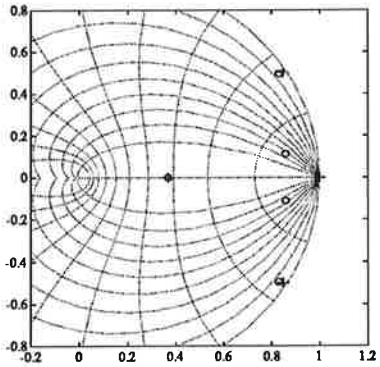


$\nu_1 = 10^4$



Kalman filter poles

Kalman filter poles (o) for $\nu_1 = 10^4$, process (+)



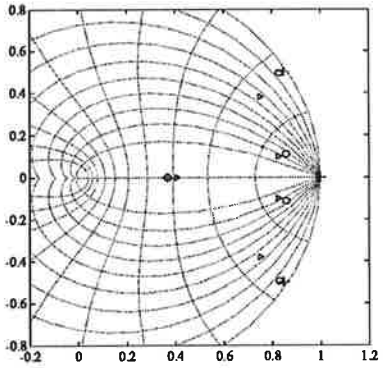
Two very resonant poles as a consequence of no process noise directly on x_1 or x_2. Modify R_1

$$R_1 = \begin{pmatrix} \nu_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Note: $x_1 \approx 1\,\text{rad/s} \Leftrightarrow y_1 \approx 0.1\,\text{V} \Rightarrow \nu_2 \approx \left(\frac{1}{0.1}\right)^2 \nu_1$

Kalman filter poles (improved)

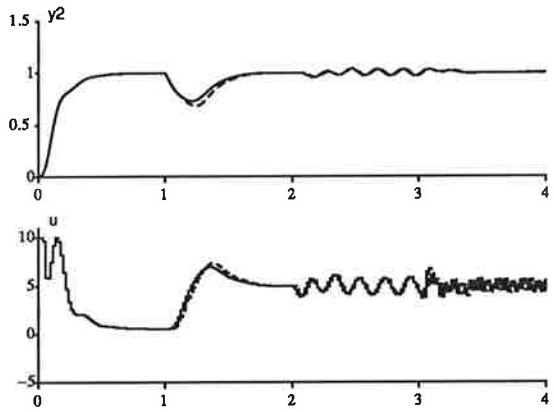
Kalman filter poles (Δ) when $\nu_2 = 10^7$



Hardly any difference in the simulation or in the frequency domain plots. Natural, since neither of these specifically test ν_2-noise properties.

Direct or no direct term

Direct term in Kalman filter gives better control performance



direct term (full), no direct term (dashed)

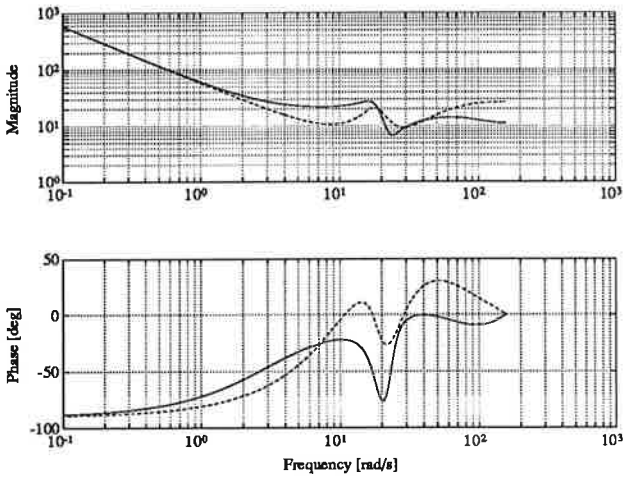
Final controller (LQG)

The controller can be written

$$x_c(k) = \Phi_c x_c(k) + \Gamma_{cu} u_{sat}(k) + \Gamma_{cy} y(k)$$

$$u(k) = l_r y_r(k) + C_c x_c(k) + D_c y(k)$$

Final controller transfer function (almost PI!)



$-H_{uy}$ (full), H_{uy_r} (dashed)

Polynomial design

Controller

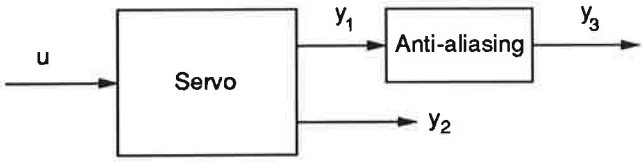
$$u(k) = \frac{T}{R} y_r(k) - \frac{S}{R} y(k)$$

1. model on polynomial form
2. cancel zeros?
3. choice of closed loop poles (A_m, A_o)
4. Diophantine equation, unique solution?
5. evaluate design

Valuable information from the LQG design

MATLAB, Simnon

Model on polynomial form



$$H_{y_3u} = \frac{10^{-3}\left(12.30q^3 - 14.81q^2 - 4.69q + 8.71\right)}{q^4 - 3.06q^3 + 3.66q^2 - 1.94q + 0.35}$$

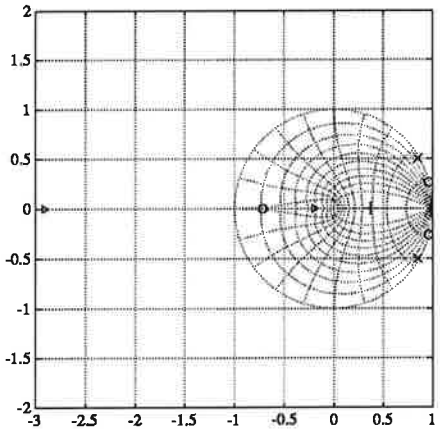
$$H_{y_2u} = \frac{10^{-4}\left(5.06q^2 + 15.76q + 2.95\right)}{q^3 - 2.69q^2 + 2.66q - 0.96}$$

- common poles (except anti-aliasing filter)
- different zeros

Cancel any zeros?

x servo poles, + anti-aliasing pole

o zeros of H_{y_3u}, Δ zeros of H_{y_2u}



When solving

$$AR + BS = B^+ A_o A_m$$

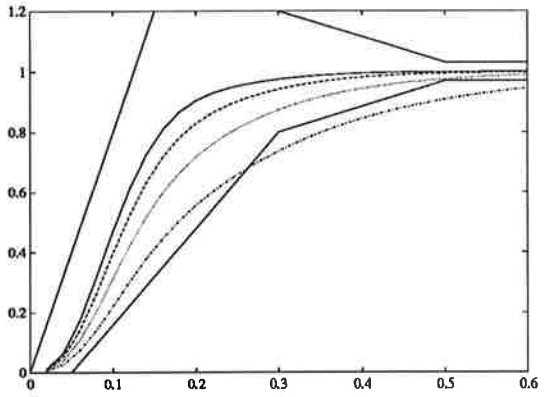
note that B relates to (o) but B^+ to (Δ)

Don't cancel any zeros!

Choosing desired poles (1)

servo + anti-aliasing ⇒ 4 poles

- keep anti-aliasing pole
- damp (ζ_m) oscillatory mode, don't change frequency
- remaining pole at α

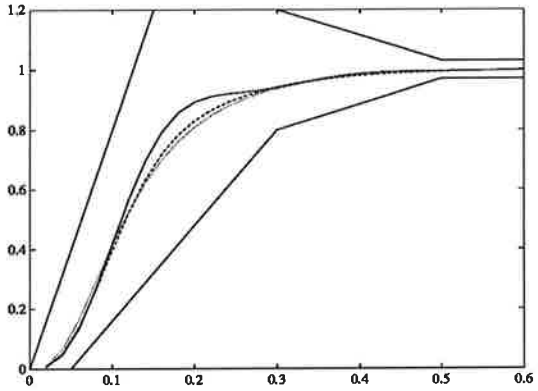


Damping $\zeta_m = 0.6$ and

$\alpha = 0.75$, (full) $\alpha = 0.85$, (dotted)

$\alpha = 0.80$, (dashed) $\alpha = 0.90$, (dash-dotted)

Choosing closed loop poles (2)



Real pole at $\alpha = 0.8$ and

$\zeta_m = 0.4$, (full) $\zeta_m = 0.6$, (dotted)
$\zeta_m = 0.8$, (dashed)

Final choice

$\alpha = 0.8$ $\zeta_m = 0.6$

Choosing observer poles

$$\deg A_o \geq 2\deg A - \deg A_m - \deg B^+ + l - 1$$
$$= 8 - 4 - 0 + 1 - 1 = 4$$

$\deg A_o = 4$ $\Rightarrow$ reduced order observer!

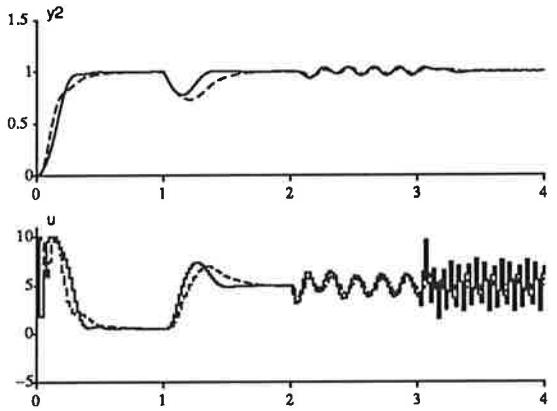
- two poles corresponding to the oscillatory mode, change damping (ζ_o) but not frequency
- LQG suggests two poles at $\beta \pm \gamma i \approx 0.8 \pm 0.2i$

Try

$\zeta_o = 0.3$, $\beta = 0.8$, $\gamma = 0.2$

Unique solution for $\deg R = \deg S = \deg T = 4$

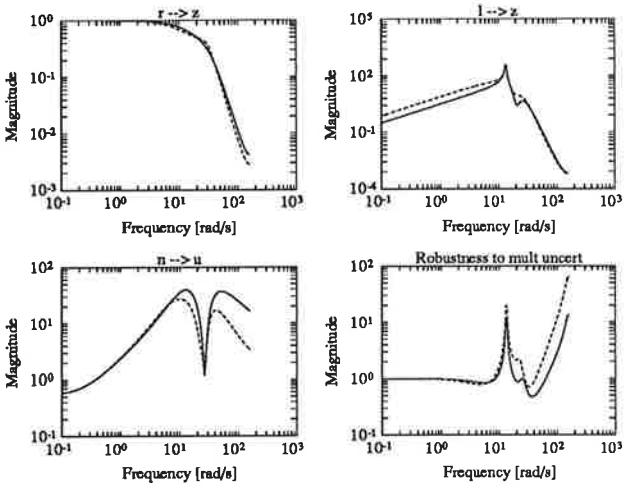
Simulation (RST – LQG)



RST (full), LQG (dashed)

Note how the control signal saturation degrades the step response.

Frequency domain (RST – LQG)



RST (full), LQG (dashed)

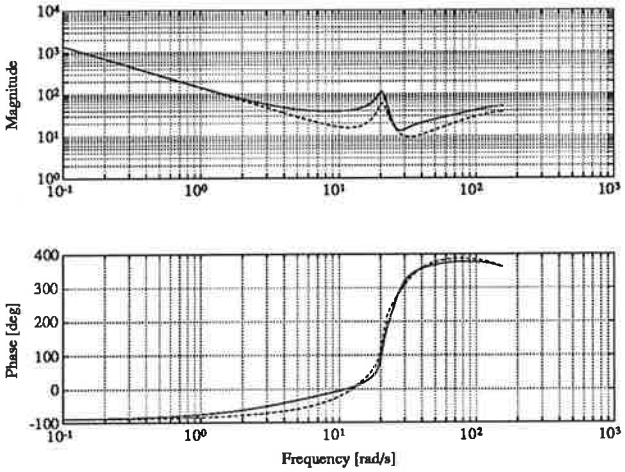
Final controller (RST)

The controller can be written

$$A_w u(k) = T y_r(k) - S y(k) + (A_w - R) u_{sat}(k)$$

with A_w for anti-windup. (in the simulations: $A_w = (q - 0.5)^4$).

Higher gain than LQG design.

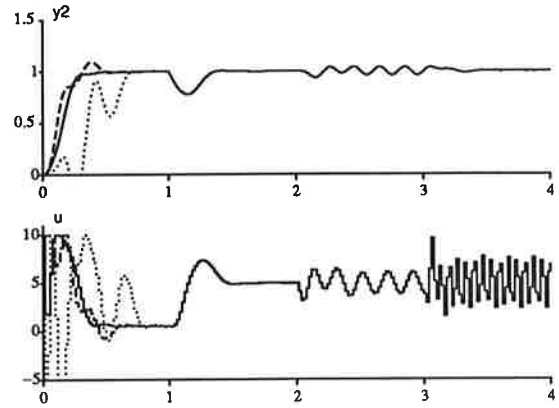


$-H_{uy}$ (full), H_{uy_r} (dashed)

Anti-windup important

Higher bandwidth in the second design (RST) makes control signal saturate.

Anti-windup automagically handled in LQG implementation. Must choose A_w in RST.



$A_w = (q - 0.5)^4$, (full) $A_w = q^4$, (dotted)
no anti-windup, (dashed)

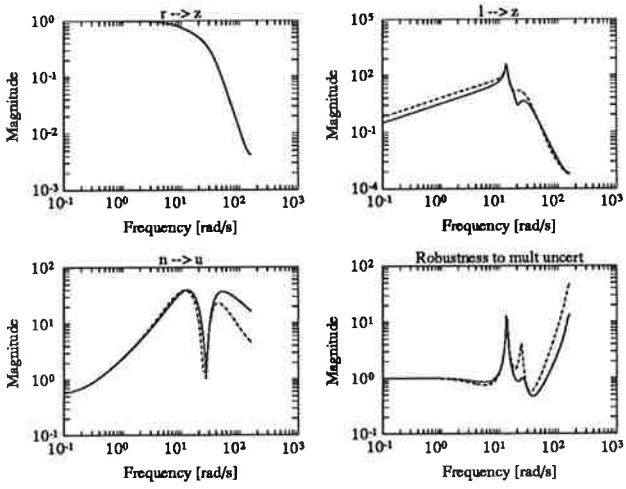
Higher order observer

Need more filtering!

Could increase degree of A_o, but no longer unique solution. How exploit non-uniqueness?

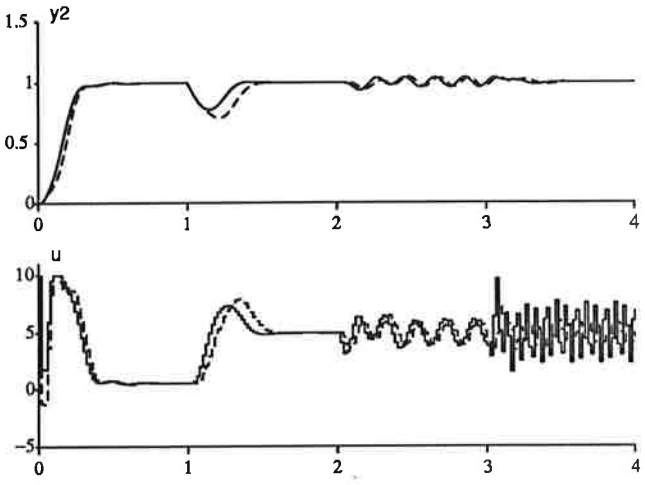
Example:

- include a pole at 0.7 in A_0
- $\deg S < \deg R$ gives uniqueness



$\deg A_o = 4$ (full), $\deg A_o = 5$ (dashed)

Simulation (high order observer)



$deg A_o = 4$ (full), $deg A_o = 5$ (dashed)

PID control

PID controller

$$U(s) = K\Big(bY_r(s) - Y(s) + \frac{1}{sT_I}\left(Y_r(s) - Y(s)\right) - \frac{sT_D}{s\frac{T_D}{N}+1}Y(s)\Big)$$

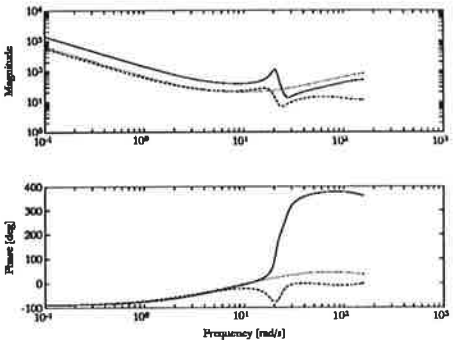
The parameters

$$K = 20, \quad T_I = 0.3, \quad T_a = 1$$
$$T_D = 0.03, \quad N = 5, \quad b = 0.6$$

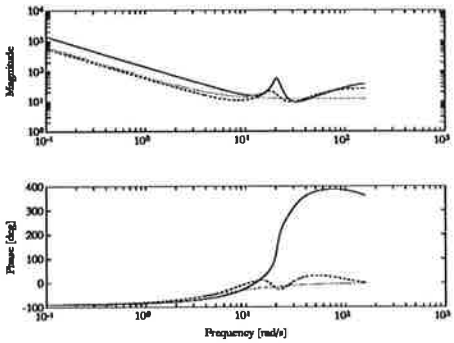
gives a rather well-tuned closed loop.

T_a is the anti-windup time constant.

RST – LQG – PID

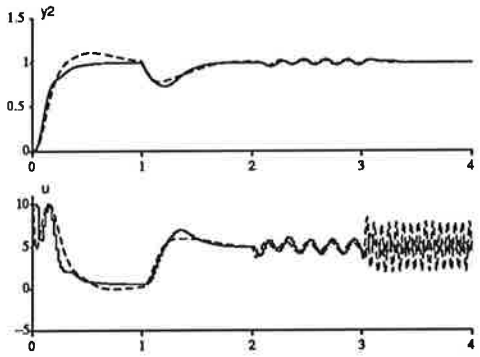


$-H_{uy}$ RST (full), LQG (dashed), PID (dotted)

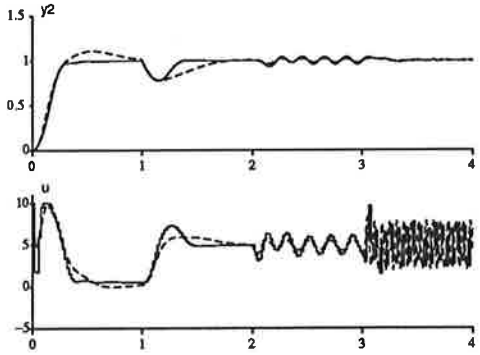


H_{uy_r} RST (full), LQG (dashed), PID (dotted)

Simulation (RST – LQG – PID)

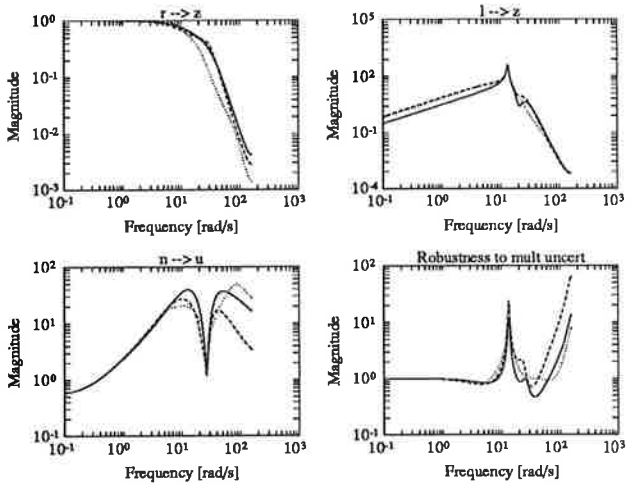


LQG (full), PID (dashed)



RST (full), PID (dashed)

Frequency domain (RST – LQG – PID)



RST (full), LQG (dashed), PID (dotted)

Observations

- iterative design
- good tools essential
- need good model
- time response easier to relate to poles than loss function
- much freedom (too much?) in polynomial design
- how use non-uniqueness?

Demonstration

3. The Fan and Plate Process

The second process consists of a fan and a plate, mounted as in Figure 1.

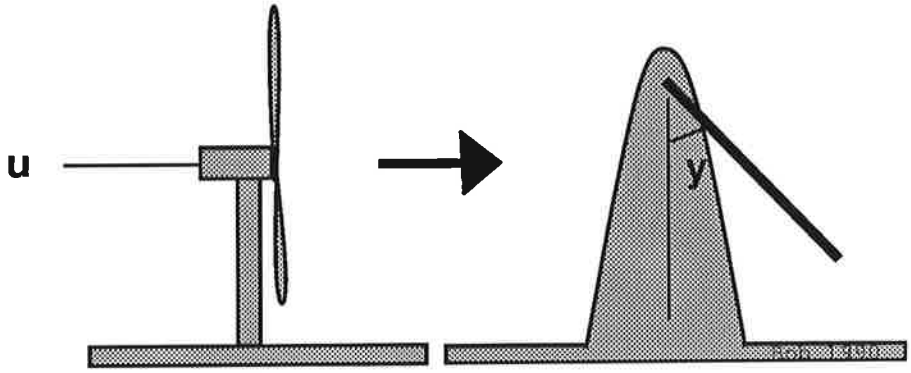


Figure 1. The Fan and Plate Process

The input to the process is the voltage to the fan while the output is the plate angle. The control objective is to keep the plate at a given angle using the fan. It is possible to vary the distance between the fan and the plate, thereby changing the process gain and time delay. The process has been used for projects in the courses identification and adaptive control.

Identification

By studying the process in open loop one observes the following details: The fan motor has a rise time of about half a second. The plate has a resonance at about 1 Hz (6 rad/s). This resonance is excited by rather much noise. The noise is probably due to turbulence. The turbulence is larger for larger plate angles. The process is also nonlinear around large angles: there is e.g. a maximal achievable angle on the plate.

The program LOGGER, see [Gustafsson] was used to save input and output using a Pseudo Random Binary Sequence as input signal. The experiments were performed with the fan as close to the plate as possible, giving minimal time delay.

The choice of experimental condition was rather iterative and went along the following lines: The period of the uncontrolled plate is about 6 rad/s and the rule of thumb from e.g. [Åström-Wittenmark]

$$\omega h = 0.2 - 0.6$$

gives a sampling time of 30-100 ms. The mean of the PRBS was chosen to keep the plate hanging at a small positive angle. The amplitude was chosen as large as possible without getting into the possibly nonlinear region. The period was chosen to give input energy that excites the eigenfrequency of the plate.

Five different experiments were performed. They were all used for identification and some of them also used for design. After some iterations, experiment number 4 , see below, was considered the best and it will be used in the following discussion. The setups in the experiments were:

```
%                    1      2      3      4      5
%   Tsamp(ms)      100    100    100     30     30
%   mean (V)       0.6      1    0.6    0.6    1.2
%   amp  (V)       0.3    0.5    0.3    0.3    0.3
%   per             10     10      5     10     15
%   n              500    500    500   1000   1000
%
```

The data was analyzed in Matlab and different Armax and Box-Jenkins models were tested using the system identification toolbox. It turned out that there was not much difference between the two methods and in the following only Armax will be described.

Figure 2 shows how well the model predicts the real data for four different model structures (about 20 were tested). They are named according to the system identification toolbox.

$$2103: \quad q(q^2 + a_1 q + a_2)y = b_3 u + q^3 e$$

$$3133: \quad (q^3 + a_1 q^2 + a_2 q + a_3)y = b_3 u + (q^3 + c_1 q^2 + c_2 q + c_3)e$$

$$3331: \quad (q^3 + a_1 q^2 + a_2 q + a_3)y = (b_1 q^2 + b_2 q + b_3)u + (q^3 + c_1 q^2 + c_2 q + c_3)e$$

$$33310: \quad q^7(q^3 + a_1 q^2 + a_2 q + a_3)y = b_3 u + q^7(q^3 + c_1 q^2 + c_2 q + c_3)e$$

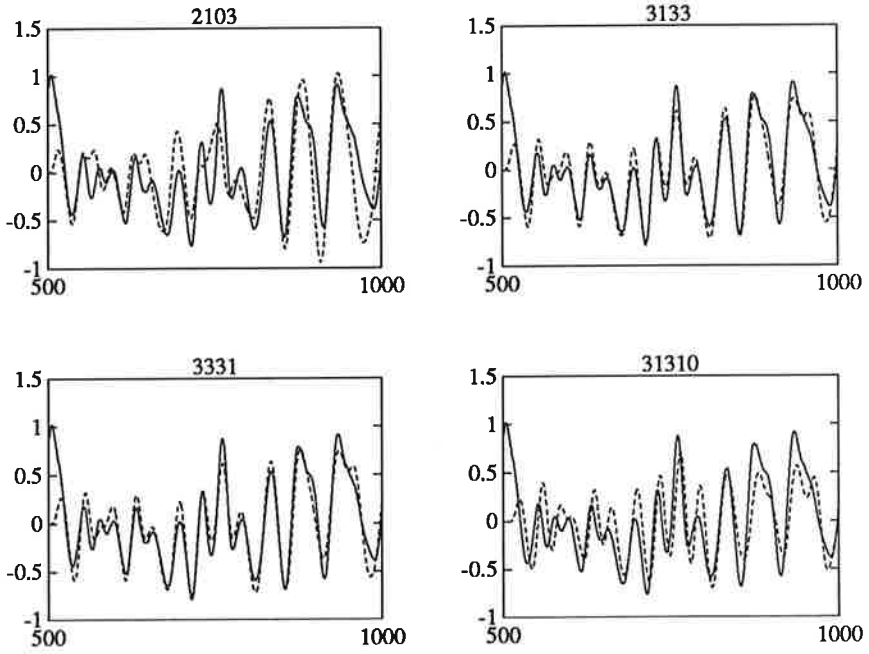


Figure 2. Experiments 2103, 3133, 3331 and 31310, real process=full, simulated model=dashed

From this, one clearly sees that a time delay of 10 samples (0.3 sec) is too large. Both 3133 and 3331 are good models and we choose 3133 because of simplicity.

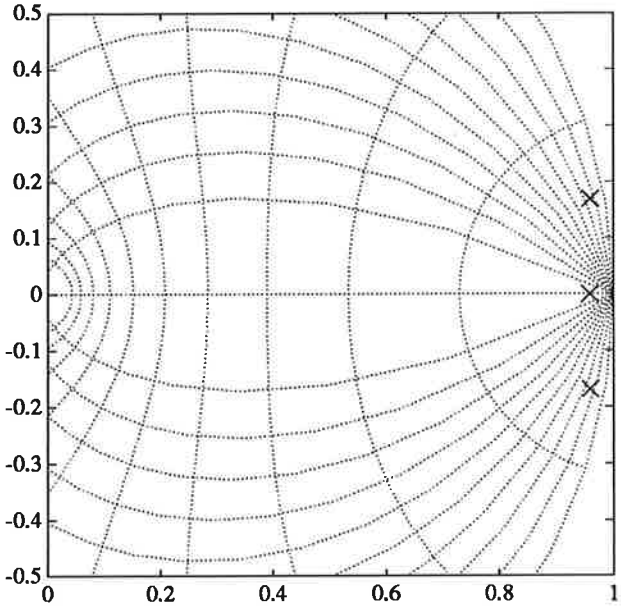


Figure 3. Open loop poles of model 3133, $h = 0.03$s

Figure 3 shows the poles of the model 3133. The slow real pole is due to the motor, it has a time constant of about half a second. The two badly damped poles are due to the plate. They correspond to $\omega = 6$ and $\zeta = 0.15$.

Figure 4 shows the Bode diagrams for $B(q)/A(q)$ and $C(q)/A(q)$ for experiment 4 using model 3133.

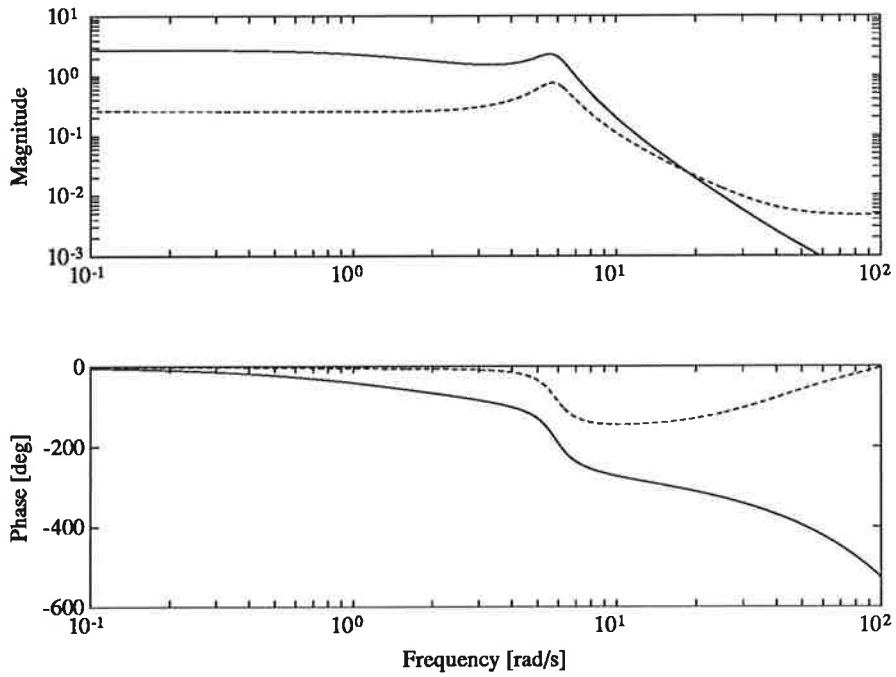


Figure 4. Bode diagram for model 3133, B/A (full), C/A (dashed)

A good model is hence of third order, without any zeros and there is a lot of disturbances near the interesting frequencies. The identified parameters,

using $h = 0.030s$ are (parameters on first row, standard deviations on second row):

```
B =
         0         0         0    0.0034
         0         0         0    0.0002
A =
    1.0000   -2.8786    2.7907   -0.9108
         0    0.00120    0.0155    0.0076
C =
    1.0000   -1.5112    0.7326   -0.1906
         0    0.0448    0.0739    0.0448
```

Design

PID-controller

As a start a PID-controller was calculated using the Ziegler-Nichols self oscillation method. This gave

$$K = 0.25$$
$$T_i = 0.55$$
$$T_d = 0.14$$
$$N = 5$$

The PID-controller was discretized and implemented in realtime Simnon, as a discrete-time system with $h = 0.03s$. The resulting performance is shown in Figure 5.

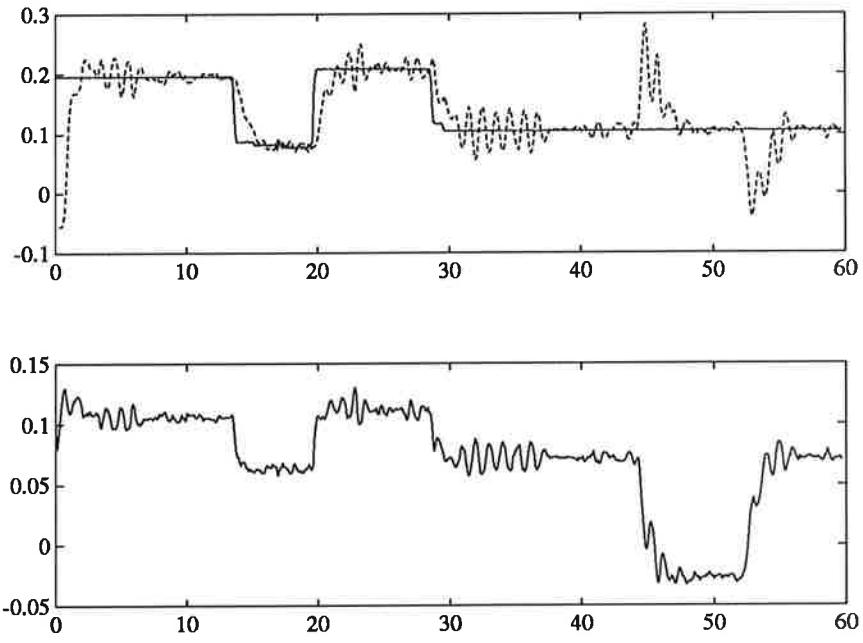


Figure 5. Time responses for PID-controller on real process, y (full), y_r (dashed) and u (full).

The PID controller works rather badly. It is badly damped and rather sensitive to load disturbances. It could be tuned for better damping and

less load disturbance sensitivity but then the performance to step changes in reference value is degraded.

The step-response methods were also tested, both Ziegler-Nichols and Cohen-Coons [Hägglund], see Figure 6. They performed very poorly.

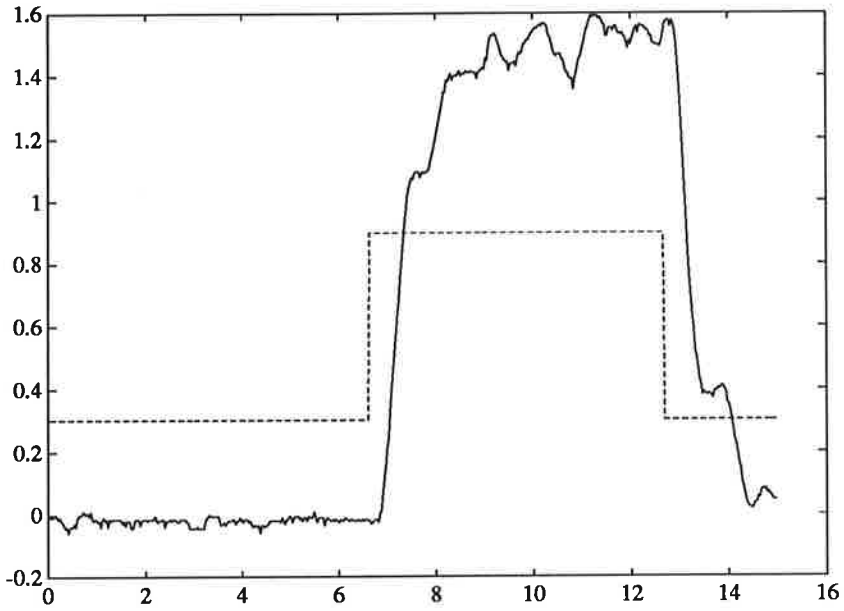


Figure 6. Step-response for real process, output (full), input (dashed)

The resulting parameters (Coohen-Coons) are

$$K = 3.1$$
$$T_i = 0.44$$
$$T_d = 0.11$$
$$N = 5$$

These parameters give an unstable closed loop system. This is an example of the fact that the step response method often works very poorly for oscillative systems.

Pole placement

Different pole placement designs were then calculated using Matlab and the routines in PPBOX. The designs were performed in discrete-time. The pole placement was iterative and many more designs have been tested than the ones documented below. In the presentation here we will follow the following line:

- Choice of A_m polynomial.
- Choice of A_o polynomial.
- Introduction of band stop filter.

No analog presampling filter was considered necessary since there was not much high frequency noise present in the measurement signal. The controller should have an integrator since we do not want a steady state control error.

Closed loop bandwidth A suitable closed loop bandwidth was found by placing the poles in a butterworth pattern:

$$A_m = A_o = (s + w_m)(s^2 + 2s\zeta_m w_m + w_m^2)$$

and varying $\omega_m = 3, 6, 9$, ($\zeta_m = 0.7$). Figures 7 and 8 were created by the commands `evpl` and `yush`, `yupl` in PPBOX. One clearly sees that $w_m = 6$ is good. This corresponds to not moving the eigenfrequency from the plates natural frequency. The main tradeoff is between high noise sensitivity (high gain) and bad disturbance rejection (low gain). Note that both chosing a too slow and too fast closed loop system gives a non robust system with respect to multiplicative uncertainty. low is non robust.

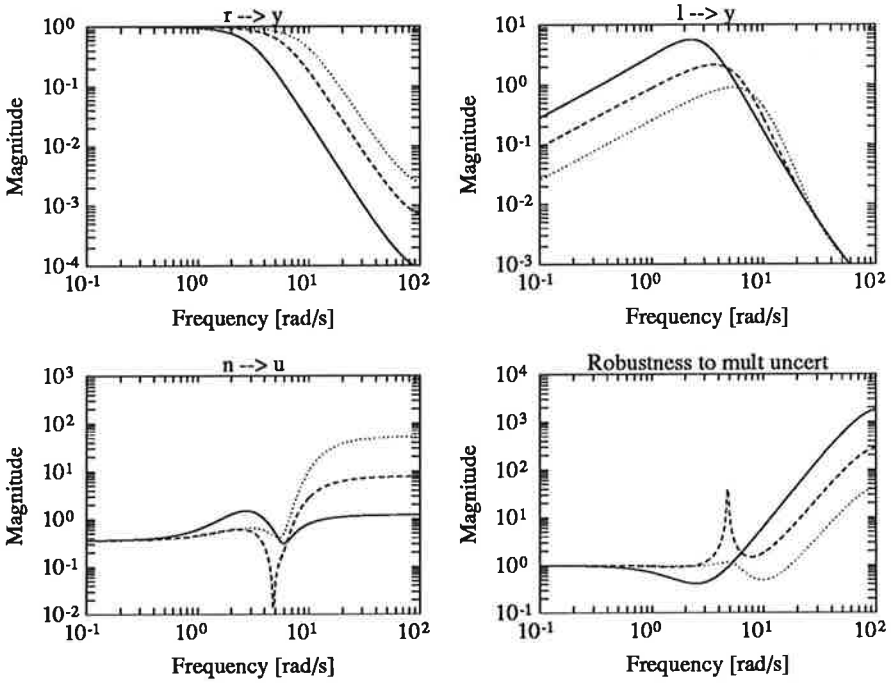


Figure 7. Varying closed loop bandwidth $\omega_m = 3$ (full), 6 (dashed), 9 (dotted)

It was also tried to vary the damping of the closed loop polynomial. The same trade off as before shows up and the choice of $\zeta_m = 0.7$ turns out to be good.

Observer From the above $w_m = 6$ was chosen and different observers were then tried. Figure 9 and 10 show the response when the observer poles are varied in a butterworth pattern ($\zeta = 0.7$) with $w_o = 4, 6, 8$. One sees that the tradeoff is between low noise sensitivity and good disturbance rejection. $w_o = 6$ is chosen, a higher observer bandwidth will give more than 10 times amplification of measurement noise on input signal for high frequencies.

After this different ζ_o were tried. Figure 11 shows the result. From the Figures the value $\zeta_o = 0.1$ was chosen.

New choice of A_m The best design found so far does not change any zeros, and places closed loop poles in

$$A_m = A_o = (s + w_m)(s^2 + 2s\zeta_m w_m + w_m^2)$$

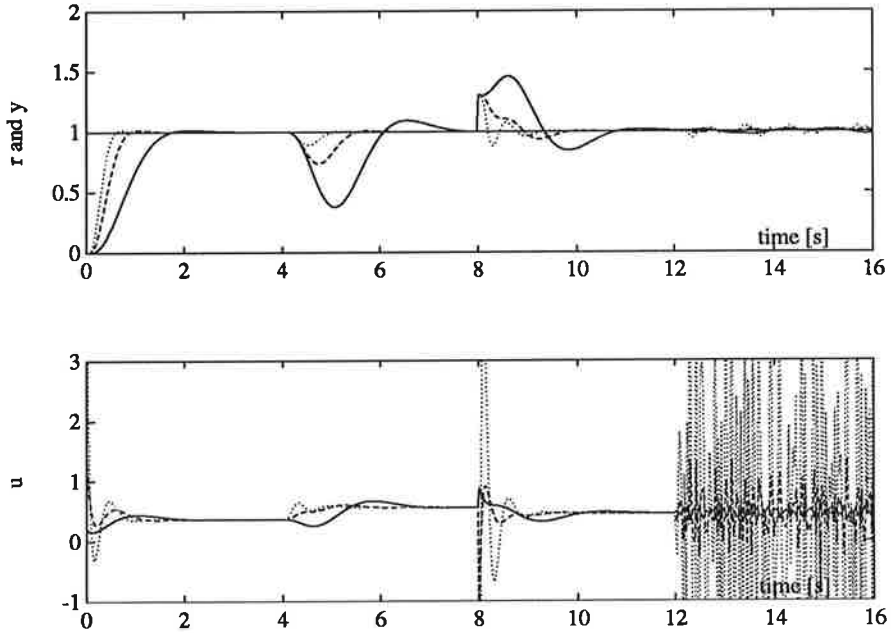


Figure 8. Simulated time response, $\omega_m = 3$ (full), 6 (dashed), 9 (dotted)

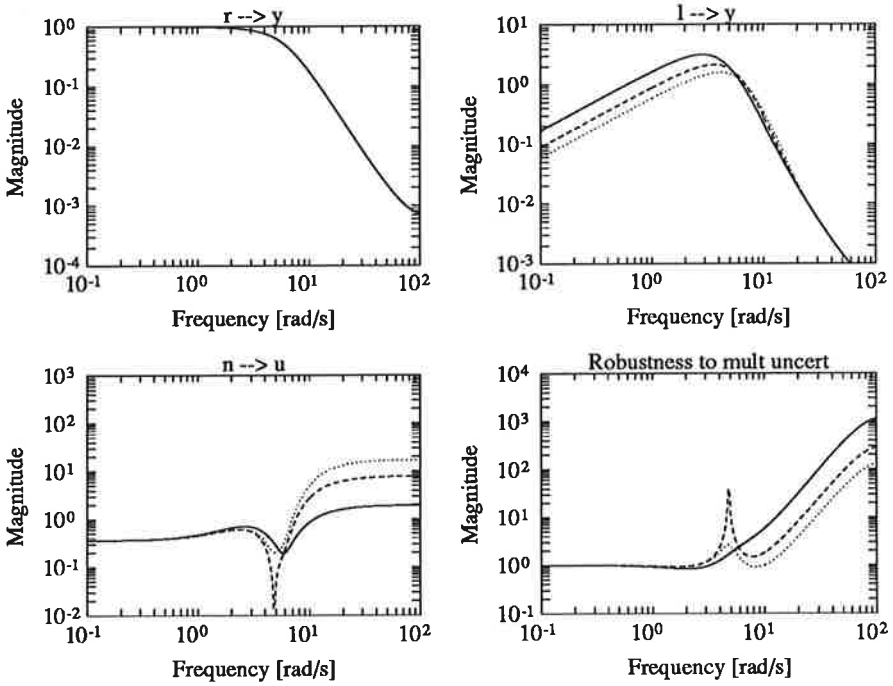


Figure 9. Varying observer bandwidth, $w_o = 4$ (solid), 6 (dashed) and 8 (dotted)

with $\omega_m = 6$ and $\zeta_m = 0.7$. It was also tried to make the real pole slower than the other poles. Different designs having

$$A_m = (s + \alpha w_m)(s^2 + 2s\zeta_m w_m + w_m^2)$$

with varying α were tried. However a small value of α gives bad disturbance rejection and a too large α a noise sensitive controller, i.e. the same trade off as before. It turns out that $\alpha = 1$ is a reasonable compromise for this system.

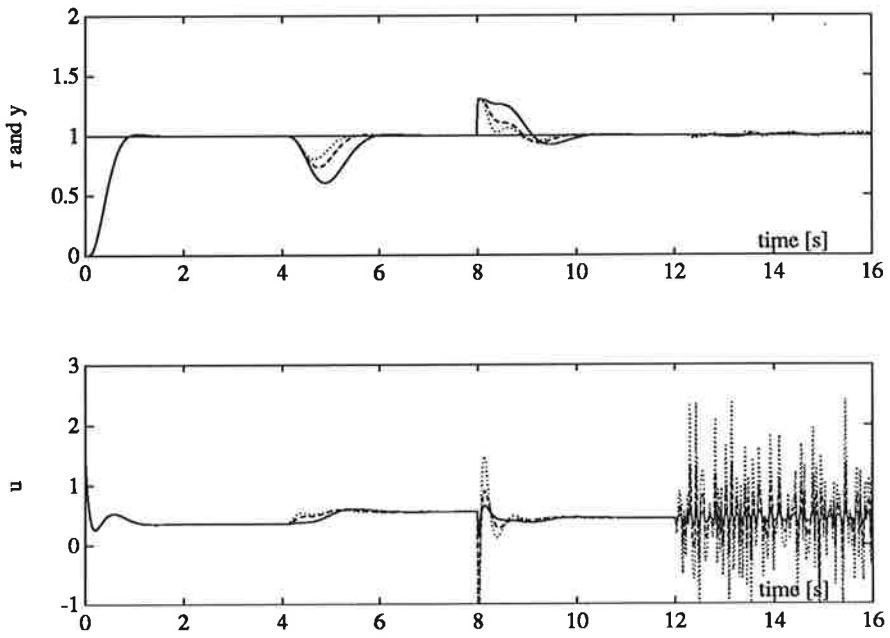


Figure 10. $\omega_o = 4$ (solid), 6 (dashed) and 8 (dotted)

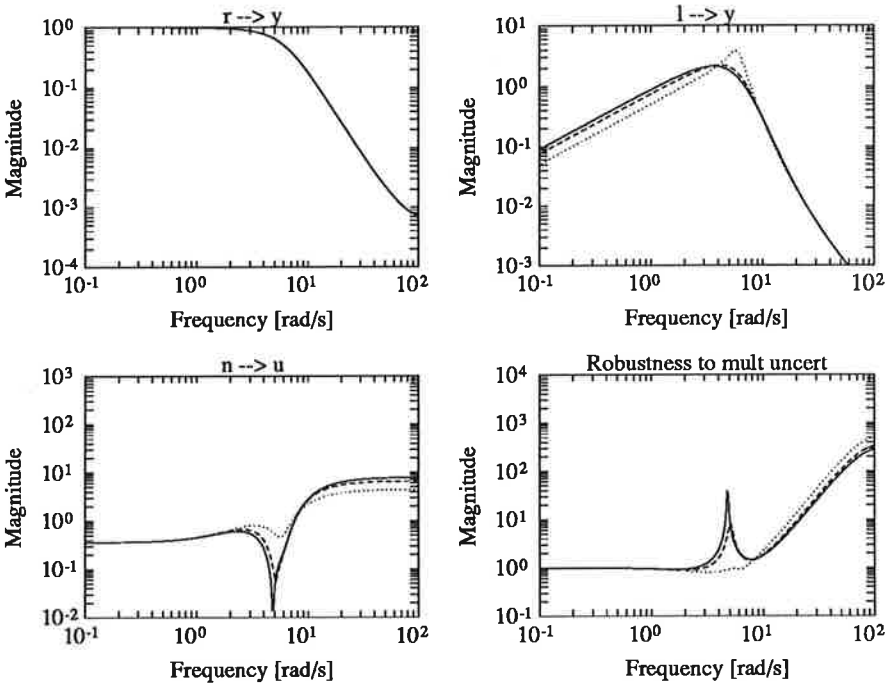


Figure 11. $\zeta_o = 0.7$ (full), 0.5 (dashed), 0.3 (dotted)

Introduction of Notch filter Since there are so large disturbances due to the turbulence it was also tried to put a factor in $R(q)$ introducing a filtering in the controller around 6 rad/s. The controller will then be of order five:

$$R(q) = (q-1)(q^2 - 2qe^{\zeta\omega h}\cos(\omega h\sqrt{1-\zeta^2}) + e^{2\zeta\omega h})R'(q)$$

The results for $\omega = 6$ and different ζ values are shown in Figure 12 and 13. Note that a small ζ value gives a very narrow bandpass filter. The value $\zeta = 0.3$ was chosen.

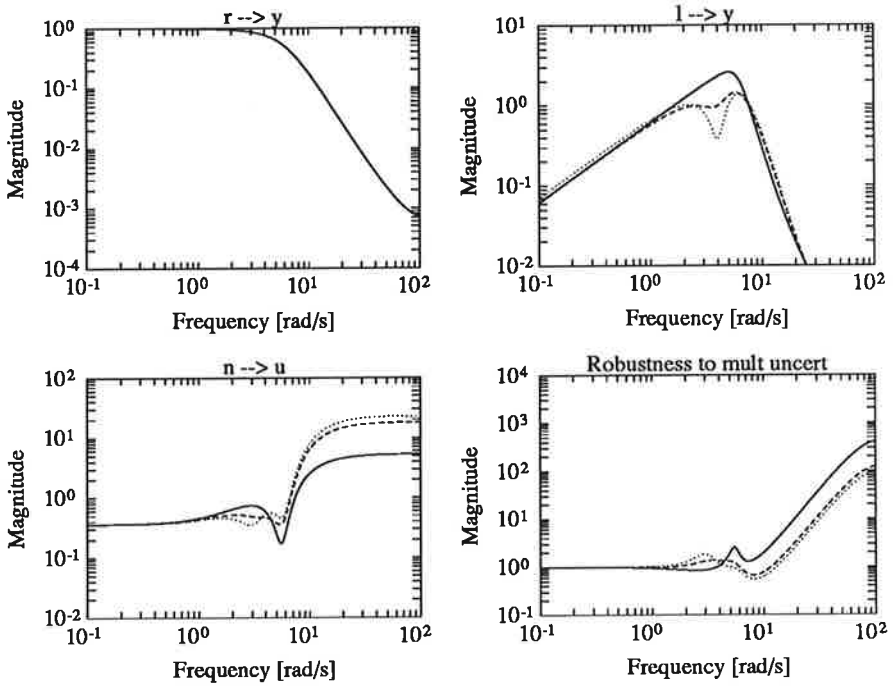


Figure 12. without filter (full), with filter $\zeta = 0.3$ (dashed) and 0.1 (dotted)

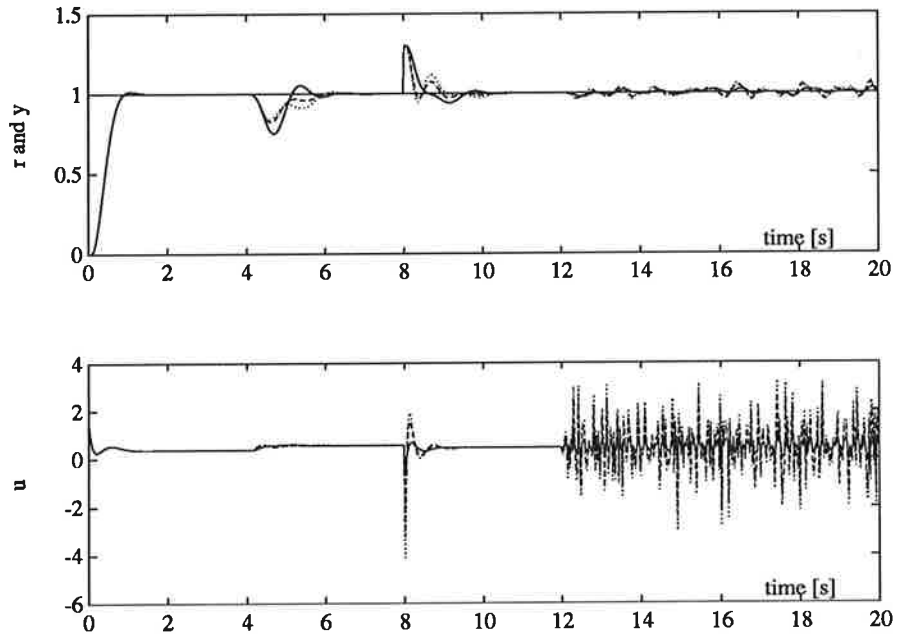


Figure 13. without filter (full), with filter $\zeta = 0.3$ (dashed) and 0.1 (dotted)

The designs described aboved were all implemented and checked with the real process with realtime-Simnon and the controller implementation in Paclib. This was done along with the iterative designs above.

There where large problems to implement the fifth order regulators. This was found to be due to numerical round off in Simnon and this was solved using an implementation with delta operators, see the following section. After these modifications the controller worked well. The problems with the turbulence are smaller than with the PID controller. The system is also better damped.

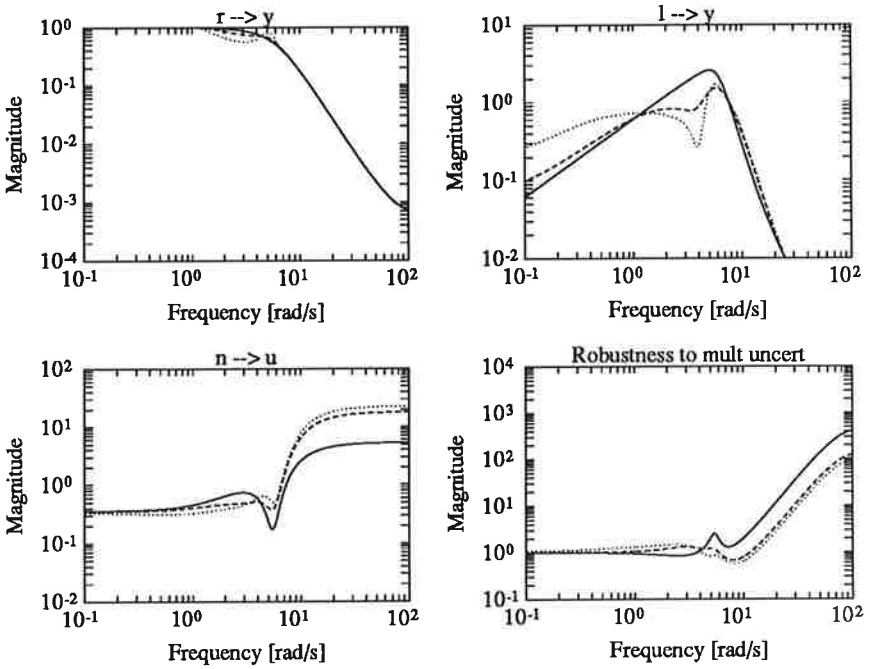


Figure 14. Same as Figure 12 but with numerical round off

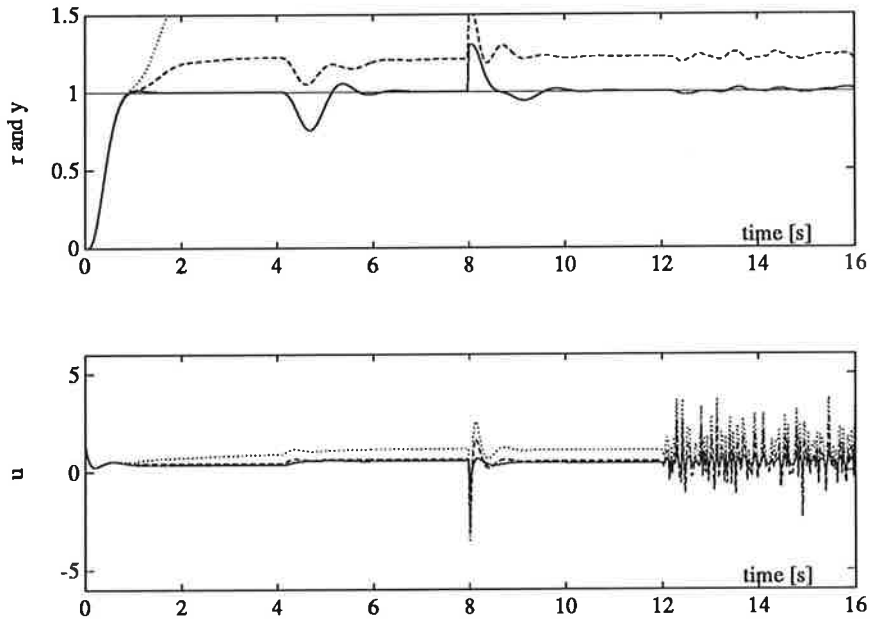


Figure 15. Same as Figure 13 but with numerical round off

A Delta operator implementation of RST-controllers

Figure 14 and 15 shows what happens if the parameters in R, S and T are multipied by $(1 + 10^{-6}!cdotrand())$. This corresponds approximately to a round off to 6 valid figures.

There is a steady state error, very large for the fifth order controllers. This is due to that $S(1) = T(1) = 10^{-4}$ is required to get unit gain for the closed loop system. This is spoiled due to numerical round off.

Two Matlab routines were written, `polyq2d` and `polyd2q`, that transforms

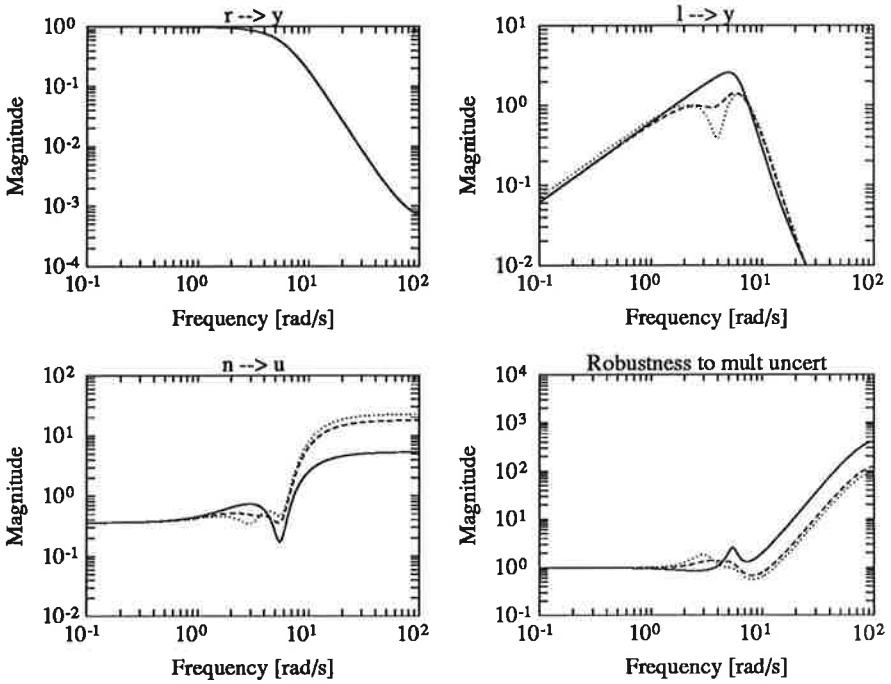


Figure 16. Same as Figure 14 but with delta form implementation

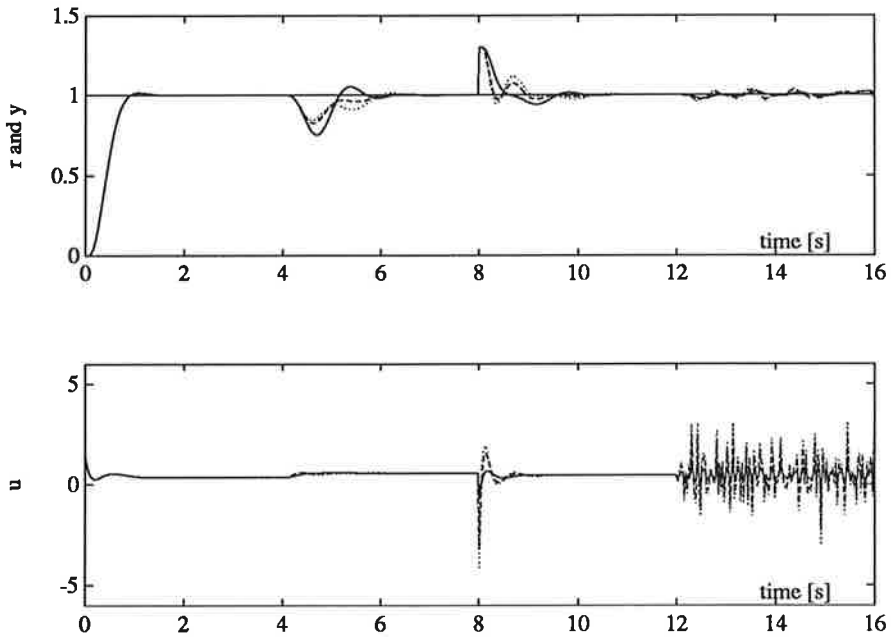


Figure 17. Same as Figure 15 but with delta form implementation

a polynomial in forward shift to polynomial in the delta operator

$$\delta = \frac{q-1}{h}$$

and back.

The controller in Paclib is implemented in observable form using forward shift operators. This controller was rewritten to delta form and implemented in realtime Simnon.

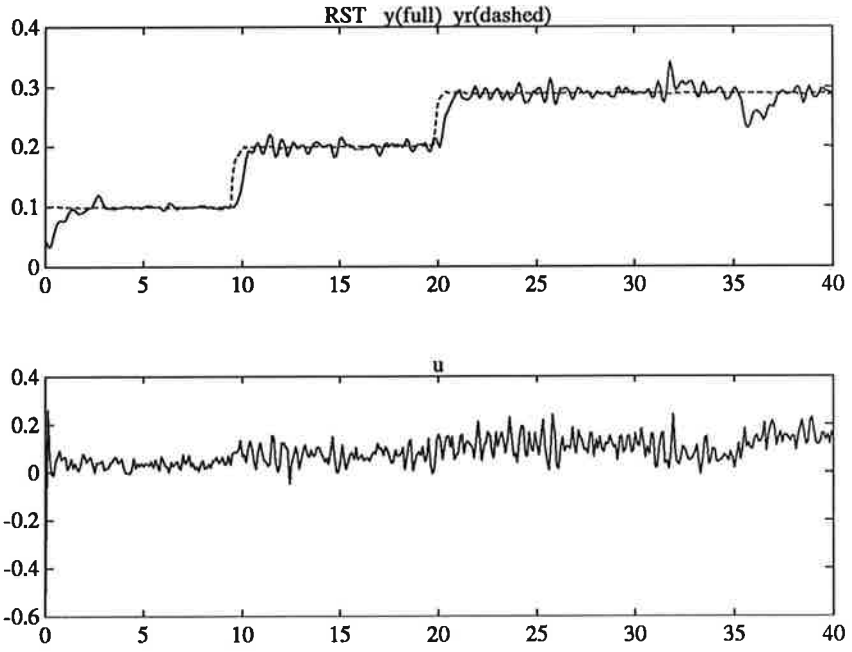


Figure 18. Performance of RST-regulator with notch filter ($\zeta = 0.1$). Load disturbances are at time=32 and 35. Compare this Figure with Figure 5.

The results in Matlab was very good and are shown in Figures 16 and 17. The disturbance in the sixth decimal can not be noticed any more. Even using as few as three valid figures was possible using delta operators, at least in the Matlab simulations. The delta form controller worked well on the real process, see Figure 18. The performance was considered satisfactorily and no more design were done.

The final design (with notch filter) is given by ($h = 0.03$ s)

```
" rst2d.t
" Created in Matlab at 1990-7-6 5:54:33
"
[reg]
r1 :  5.02782
r2 :  11.1412
r3 :  5.93689
r4 :  3.68313
r5 :  0
s0 :  12.542
s1 :  16.0767
s2 :  20.6164
s3 :  11.8944
s4 :  5.26107
s5 :  1.00789
t0 :  1.36876
t1 :  4.64124
t2 :  7.61997
t3 :  7.18227
t4 :  3.88962
t5 :  1.00789
```

4. References

ÅSTRÖM, K. J., and WITTENMARK, B. (1990): *Computer Controlled Systems, Theory and Design, 2nd ed.*, Prentice-Hall, Englewood Cliffs, pp. 250–254, 271.

ANDERSSON, M. (1988): "Reglering av flexibelt servo," Laboration 4, AK FED. Department of Automatic Control, Lund..

GUSTAFSSON, K., LILJA, M., and LUNDH, M. (1990): "A Collection of Matlab Routines for Control System Analysis and Synthesis," Report TFRT-7454, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

GUSTAFSSON, K. (1990): "**logger** – a program for data logging," Report TFRT-7457, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

HÄGGLUND, T. (1990): *Praktisk processreglering*, Studentlitteratur.

SSPA SYSTEMS (1990): *Simnon, User's Guide.*

THE MATHWORKS (1990): *Pro-Matlab, User's Guide.*

5. Appendices

Delta form RST-controller

```
DISCRETE SYSTEM reg
"
" Delta'-form of Wallenborg regulator
" Discrete time polynomial regulator
" with input/output relation
"
" R(d)u(k) = -S(d)y(k) + T(d)yr(k)
"
" R(d) = d^n + r1*d^(n-1) + ... + rn
" S(d) = s0*d^n + s1*d^(n-1) + ... + sn
" T(d) = t0*d^n + t1*d^(n-1) + ... + tn
"
" maximum polynomial degree (n) = 6
" d = (q-1)/(om*h)
"
" The regulator is implemented as a minimal
" realisation on observable canonical form.
"
" Anti Reset Windup mechanism implemented as
" in CCS pp 371-372 with observer char. eqn
"
" det(zI-F+KC) = P(z) = z^n + p1*z^n-1 + ... + pn
"
" The antireset windup can be shut off using aw:0

STATE x1 x2 x3 x4 x5 x6
NEW   nx1 nx2 nx3 nx4 nx5 nx6

TIME t
TSAMP ts
```

```
INITIAL
   omh = om*h
   bs1 = -s1+r1*s0
   bs2 = -s2+r2*s0
   bs3 = -s3+r3*s0
   bs4 = -s4+r4*s0
   bs5 = -s5+r5*s0
   bs6 = -s6+r6*s0

   bt1 = t1-r1*t0
   bt2 = t2-r2*t0
   bt3 = t3-r3*t0
   bt4 = t4-r4*t0
   bt5 = t5-r5*t0
   bt6 = t6-r6*t0

   ds = -s0
   dt = t0

   k1 = p1-r1
   k2 = p2-r2
   k3 = p3-r3
   k4 = p4-r4
   k5 = p5-r5
   k6 = p6-r6
SORT

y = ADIN(0,t) - y0      " measured process output signal
yr = ADIN(1,t) - y0r    " reference value
uclip = DAOUT(0,u + u0) " control signal

dum1 = DAOUT(1,0.4)     " used to create reference value
dum2 = DAOUT(2,-0.1)    " used to create reference value
dum3 = SDAOUT(0,0)      " safety value


u = x1 + ds*y + dt*yr

w = uclip-u-u0
nx1 = x1 + omh*(-r1*x1 + x2 + bs1*y + bt1*yr + aw*k1*w)
nx2 = x2 + omh*(-r2*x1 + x3 + bs2*y + bt2*yr + aw*k2*w)
nx3 = x3 + omh*(-r3*x1 + x4 + bs3*y + bt3*yr + aw*k3*w)
nx4 = x4 + omh*(-r4*x1 + x5 + bs4*y + bt4*yr + aw*k4*w)
nx5 = x5 + omh*(-r5*x1 + x6 + bs5*y + bt5*yr + aw*k5*w)
nx6 = x6 + omh*(-r6*x1      + bs6*y + bt6*yr + aw*k6*w)


ts = t + h

" parameters
om : 6
h  : 0.03           " sampling interval
aw : 0              " antiwindup, 1 - on, 0 - off
y0r: 0.0            " Dc-level on yr
y0 : 0.0            " Dc-level on y
u0 : 0.0            " Dc-level on u
r1 : 0.0
r2 : 0.0
r3 : 0.0
r4 : 0.0
r5 : 0.0
r6 : 0.0
```

```
s0 : 0.0
s1 : 0.0
s2 : 0.0
s3 : 0.0
s4 : 0.0
s5 : 0.0
s6 : 0.0
t0 : 0.0
t1 : 0.0
t2 : 0.0
t3 : 0.0
t4 : 0.0
t5 : 0.0
t6 : 0.0
p1 : 0.0
p2 : 0.0
p3 : 0.0
p4 : 0.0
p5 : 0.0
p6 : 0.0
END
```

Matlab functions for delta operators

```
function [pd] = polyq2d(p,h);
% function polyq2d(p,h);
% Transforms a polynomial p (row vector) in q to its delta form
% using sampling period h.
% q = 1+h*delta
%
%
[nr,nc]=size(p);
dum = ones(1,nc);
a=[];
for i = 1:nc,
  a = [dum(1:nc);a];
  dum = h*cumsum([0 dum]);
end
pd=p*rot90(a);
end

function [p] = polyd2q(pd,h);
% function polyq2d(pd,h);
% Transforms a polynomial pd (row vector) in delta to q
% using sampling period h.
% delta=(q-1)/h
%
%
p=0;
dum = 1;
while length(pd)>0,
  p = addpoly(p,pd(length(pd))*dum);
  dum = conv(dum,[1 -1]/h);
  pd = pd(1:length(pd)-1);
end
end
```

Matlab code to generate Figure 7 and 8

```
% Design1, compare different w in Am
% a,b,h given
while b(1)==0, b=b(2:length(b)), end;
```

```
om1 = 3;
om2 = 6;
om3 = 9;
amc1 = polyc(om1,0.7,-om1);
am1 = polyc2d(amc1,h);
amc2 = polyc(om2,0.7,-om2);
am2 = polyc2d(amc2,h);
amc3 = polyc(om3,0.7,-om3);
am3 = polyc2d(amc3,h);

om1o = 3;
om2o = 6;
om3o = 9;
aoc1 = polyc(om1o,0.7,-om1o);
ao1 = polyc2d(aoc1,h);
aoc2 = polyc(om2o,0.7,-om2o);
ao2 = polyc2d(aoc2,h);
aoc3 = polyc(om3o,0.7,-om3o);
ao3 = polyc2d(aoc3,h);

[r1,s1,t1]=rstd(b,1,a,1,am1,ao1,[1 -1])
[r2,s2,t2]=rstd(b,1,a,1,am2,ao2,[1 -1])
[r3,s3,t3]=rstd(b,1,a,1,am3,ao3,[1 -1])

gp = frd(b,a,h,-1,[],300);
gfb = frd([s1;s2;s3],[r1;r2;r3],h,-1,[],300);
gff = frd([t1;t2;t3],[r1;r2;r3],h,-1,[],300);

evpl(gp,gfb,gff,[-1 2 -4 0 ; -1 2 -3 1; -1 2 -2 3; -1 2 -1 4])

if input('hardcopy (yes - 1)? ')==1,
  meta fig7
  !gpp -dps fig7
  delete fig7.met
  !mv fig7.ps ../documentation/fig7.ps
end

trldn = yusignals(0:h:16,1,0,-0.2,4,0.3,8,0.003,12);
tryu1 = yusimd(b,a,r1,s1,t1,trldn);
tryu2 = yusimd(b,a,r2,s2,t2,trldn);
tryu3 = yusimd(b,a,r3,s3,t3,trldn);
yupl(tryu1,tryu2,tryu3,[0 16 0 2 -1 3]);

if input('hardcopy (yes - 1)? ')==1,
  meta fig8
  !gpp -dps fig8
  delete fig8.met
  !mv fig8.ps ../documentation/fig8.ps
end
```