ISSN 0280–5316
ISRN LUTFD2/TFRT--5542--SE

Dynamic Pictures in Sampled Data Systems

Helena Haglund

Department of Automatic Control
Lund Institute of Technology
November 1995

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

Document name
MASTER THESIS

Date of issue
November 1995

Document Number
ISRN LUTFD2/TFRT--5542--SE

Author(s)
Helena Haglund

Supervisor
Björn Wittenmark

Sponsoring organisation

Title and subtitle
Dynamic Pictures in Sampled Data Systems

Abstract

This Master Thesis Project presents an interactive program intended to be used in a course in Digital control, given by the Department of Automatic Control at Lund Institute of Technology. This application is supposed to give the students a good comprehension of sampled data systems. The areas that are covered are sampling, aliasing, observability, parameter sensitivity, PID-control, pole placement design, robustness, stochastic processes and LQ-control.

The implementation is done in Matlab and in some cases, the simulations are performed in Simulink. Matlabs Graphical User Interface tools are used for building the user interface.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
37

Recipient's notes

Security classification

Contents

1. Introduction

The purpose of this master thesis is to create an interactive program intended to be used in a course in Digital control, given by the department of Automatic Control at Lund Institute of Technology. This application is supposed to help the students to get a good understanding of what sampled data systems really are and also how these systems react to different changes.

All the implementation has been done in Matlab/Simulink. A great deal of the application is based on Matlabs Graphical User Interface tools to make it user friendly and easy to understand.

There are several reasons for using Matlab as a tool for this work. One is that the graphical user interface is very flexible and fairly easy to use. A second reason is the combination of graphics and computations. Finally, by using Matlab, we get a good portability, since Matlab is available on many different platforms.

2. Multimedia in education

Multimedia is the combination of video, sound, text, animation, and graphics with a computer to tie these components together. Multimedia created especially for education constitutes an effective learning environment where the student is an active participant. The main purpose with education is to transfer information, knowledge, and understanding, which nowadays mainly is done via books. The possibility to directly interact with the information, significantly improves and fastens the learning process, because the student can use all the sensory functions. The student might also find the learning more enjoyable and interesting when using multimedia tools.

There are many different ways that multimedia can be used in education. One is as an interactive lesson where a student interacts with the program and can make choices and the other one is as a demonstration tool accompanying a lecture.

There are several things to think about when building a multimedia system, for example what inputs and outputs to be made and what presentation form to be used. More about multimedia can be found in [4].

The result of this master thesis is not truly a multi media application since it only includes graphics and text, but it has the same purpose as multi media tools made for education. This purpose is to create an interactive environment that enhances the learning process and makes the students participate actively.

In a future version of this demo, it would be of interest to include for example short video sequences, illustrating specific features, and also noise to emphasize important happenings. Being able to follow the transients in real time would also be desirable. The interactivity could be increased so that the student could investigate the systems in a wider perspective. Another useful feature could be to include these demos in an electronic book. This electronic book could be used over the Internet, using tools such as Mosaic and Netscape.

3. Graphical User Interface

Matlabs Graphical User Interface tools have been used for this purpose. The User Interface is built on controls. A control is a graphic object that, when

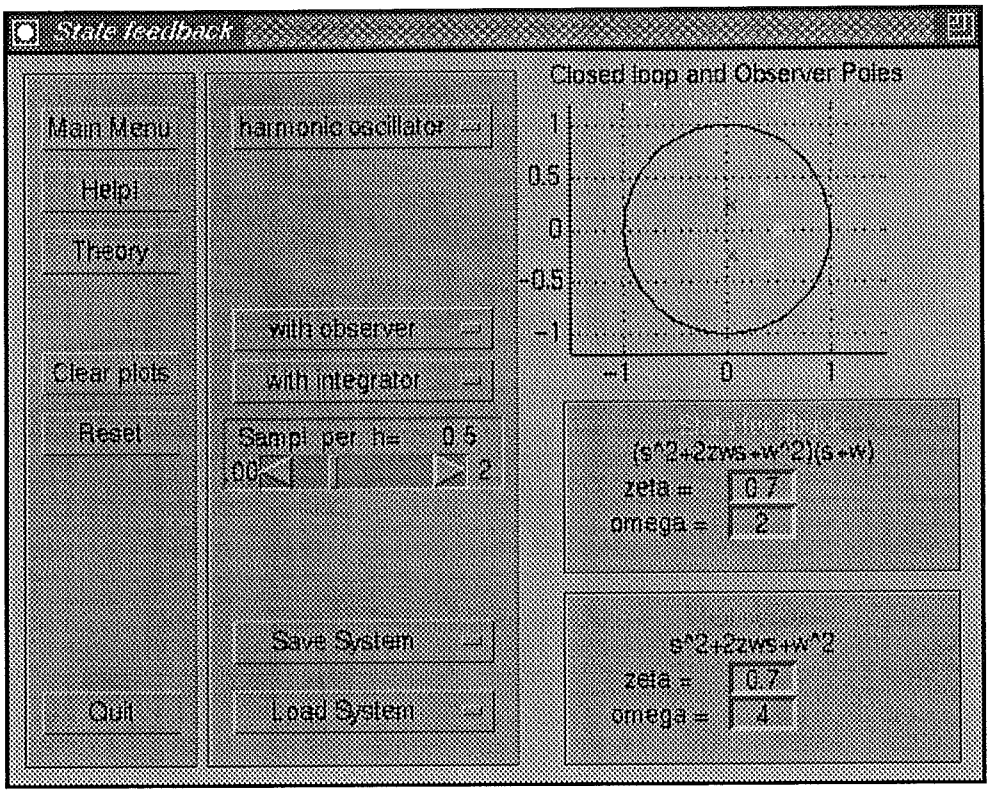

Figure 1. Example of Graphical User Interface.

manipulated with the mouse, causes an action to be performed. There are both visible and unvisible actions that can occur. A visible action is a change that is immediately seen while an unvisible one, enhances a change that modifies future actions. The different types of controls used in this application are:

- Push buttons
- Radio buttons
- Sliders
- Pop-up menus
- Static text
- Editable text
- Frames

A push button performs an action when pressed with the mouse. Radio buttons and pop-up menus give the user a choice between a number of alternatives. If the action, in this case, is visible or unvisible depends on the situation. Static text only displays textual information. Editable text and sliders give the user the possibility to choose a value of a parameter. In the case of editable text, the value can be chosen freely, while with sliders, the value is limited to a predefined range. Finally, frames just form logical groups of other control objects.

An example of a Graphical User Interface from this application can be seen in Fig. 1.

A control is created by writing

```
h = uicontrol(hfig,'PropertyName', PropertyValue,...);
```

where h is the handle of the control. In Matlab each graphical object has a handle, which is used to identify the object. A handle contains the properties of a specific control, for example callback, color, style, visibility, position etc. The style property determines what type of control it is, for example push, radio, frame. The Callback property defines the action Matlab performs when the user activates the control. This property is specifies as:

```
set(h,'Callback','string');
```

where `string` can be, for example, a Matlab command, an M-file or a variable name. Equally the color of the control can be set:

```
set(h,'Color','r');
```

which makes the control, for example a push button, red. The properties can either be set when the control is created or with `set` whenever it is desirable.

Controls belong to a group called Handle Graphics objects, which except for controls also include, for example figures, axes, lines etc. An important feature with, for example, patches and lines, is that they can enhance actions while being moved in a window. Callbacks can be executed when the user takes one of these actions:

- Presses a mouse button while the pointer is within a figure window
- Releases the mouse button
- Moves the pointer within the figure window

For a static object, like a button, an action is performed when the user clicks on it. The desired action can be performed if the objects `callback` is defined. For an object, like a pole, that is supposed to be moved in a plot, actions can be executed both when the mouse button is pressed on it, when the pointer is moving while pressed down and when the mouse button is released. Actions like these are enhanced by specifying the `ButtonDownFcn`, the `WindowButtonMotionFcn` and the `WindowButtonUpFcn` for an object.

If a specific action is performed and there is no callback associated with it, nothing happens.

When a pole is to be moved in a plot the necessary callbacks are specified when building the interface. This is done by writing

```
set(h,'ButtonDownFcn','string1',...
      'WindowButtonMotionFcn','string2',...
      'WindowButtonUpFcn','string3');
```

where `stringx` are the desired Matlab command to be performed at the respective mouse action. The `ButtonDownFcn` executes a callback when the pointer is placed, and a mouse button pressed, over or in the immediate vicinity of the pole. The `WindowButtonMotionFcn` executes a callback when the pole is moved and the `WindowButtonUpFcn` when it is released. When the mouse button is released, the present location must be determined. This is done with

```
point = get(gca,'CurrentPoint');
```

where `gca` is the current axes and `point=[x y]`.

It can also be useful to be able to place a Handle Graphic object, for example, a pole couple, in a desired position in a plot. This is done by

```
set(pole,'XData',x,'YData',y);
```

where `pole` is the handle of the pole couple. This can be created by

```
pole = plot(real(roots(P)), imag(roots(P)));
```

where P is the characteristic polynomial. More about this kind of tools can be found in [5].

4. Modules

When making an effective User Interface in Matlab it is recommended to implement it using the following structure

```
function demo(operation);
if nargin == 0,
        operation = 'init';
end;
if strcmp(operation,'system'),
        %defines a system
elseif strcmp(operation,'calc'),
        %calculations are made
elseif strcmp(operation,'clear'),
        %plots are cleared
                :
                :
elseif strcmp(operation,'init'),
        %create user interface
elseif strcmp(operation,'close'),
        %closing system
end;
```

where `operation` is a string with the value of the currently executing callback function. `Nargin` is the number of input arguments to a function. This number is zero when the function is first called. The condition `nargin == 0` makes `demo(''init'')` be executed. In this callback the user interface is built. More about this kind of function is found in [5]. The reason for using these functions instead of scripts is that the number of M-files is reduced and that the execution of callbacks is faster then calling subroutines, as is done when using scripts. All the modules in this application are implemented as functions. One function for each module. Those that in addition contain Simulink blocks must have a scriptfile, that is executed before the function and makes the necessary variables global in the workspace. This is due to that functions don't operate globally in the workspace, which scriptfiles do. It is a necessary feature for the communication between Matlab and Simulink, that the variables set in Matlab are passed out in the global workspace and vice versa. More about Simulink can be found in [6].

The areas that are covered in this application are:

- Sampling
- Aliasing
- Observability
- Numerics

- PID-control
- Robot mechanism process
- Pole placement design by state feedback
- Pole Placement design based on input-output model
- Robustness
- Noise
- Linear Quadratic control

The different modules are described in detail in the sections that follow.

5. Sampling

5.1 Theory

Almost all physical processes are continuous-time processes, but to be able to control them with a computer they must be transformed to discrete time. This is accomplished by sampling the system. When a system is sampled, only the properties at the sampling instants effect the behavior of the discretised system. All information between the sampling points is lost. The continuous time model of a system looks like

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

and the discrete time model looks like

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned}$$

where

$$\begin{cases} \Phi = e^{Ah} \\ \Gamma = \int_0^h e^{As} ds B \end{cases}$$

In a continuous time system the stability area in the complex s-plane is the left half plane. This area is mapped on to the unit circle, in the complex z-plane, when the system is sampled.

Due to the sampling procedure the properties of the system might change. An example of this is that a system with a stable inverse can get an unstable inverse for certain sampling intervals h and that extra zeros often appear in the sampled form. Another problem is that reachability and/or observability can be lost. These differences increase the importance of having a good understanding about how sampled systems work.

5.2 Goal

The purpose of the `Sampling module` is to illustrate where the poles and zeros of a system end up when the system is sampled and how the sampling interval influences the result.

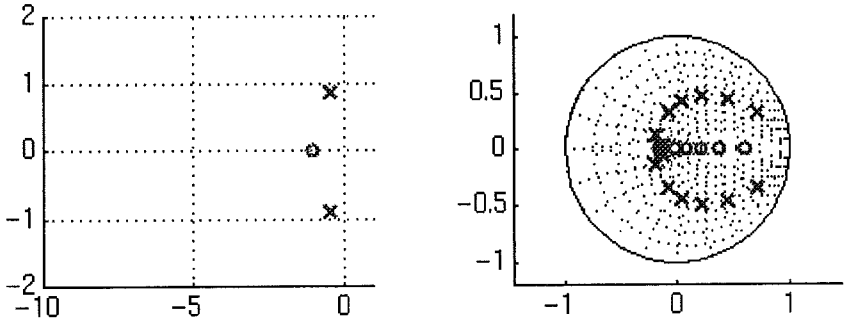

Figure 2. Sampling of system 2.

5.3 User aspects

The user has the possibility to choose among three different examples. The first one is a system with transfer function

$$G(s) = \frac{1}{s^2 + a_1 s + a_2}$$

The second one is a system whose transfer function is

$$G(s) = \frac{s + b}{s^2 + a_1 s + a_2}$$

and the third one is the Robot Mechanism process, described in [1] p.275 and in this paper, page 21 part 10.1. After having chosen a system, the continuous time poles and zeros can be moved by grabbing one of them with the arrow pointer and move them to the desired location. The sampling period can be changed by using the slider. Every change made, results in an immediate updating of the plots.

5.4 Implementation

The sampling of a system is simply done by using `c2d`.

To make the continuous time poles and zeros movable the property `Button DownFcn` of them is set to a callback `sampling(''move'')` as in

```
set(handle,'ButtonDownFcn','sampling(''move'')');:
```

This property implies that, when a pole/zero is clicked on, the corresponding if-statement is executed. The callback `sampling(''move'')` calls this if-statement, in the function `sampling`, as described in page 6 part 4. This happens when a mouse button is pressed, when the pointer is located on or in the immediate vicinity of a pole/zero. This if-statement looks like

```
elseif strcmp(operation,'move');
```

The lines belonging to this statement are then executed. In this callback the figures `WindowButtonMotionFcn` is set to `sampling(''moving'')` and the `WindowButtonUpFcn` is set to `sampling(''moved'')` like

```
set(fig_samp, 'WindowButtonMotionFcn', ...
    'sampling(''moving'');', ...
              'WindowButtonUpFcn', ...
    'sampling(''moved'');');
```

to invoke the right callbacks when the pole/zero is moved and released. This implies that, while moving the pole/zero in the plot, the code in

```
elseif strcmp(operation,'moving');
```

is executed over and over again. After having completed the move, the system is updated in the callback `sampling(''moved'')` and the code in

```
elseif strcmp(operation,'moved');
```

is executed. In the last one, the `WindowButtonMotionFcn` and the `Window ButtonUpFcn` are reset to having no callbacks as in

```
set(fig_samp, 'WindowButtonMotionFcn', ...
    '', ...
              'WindowButtonUpFcn', ...
    '');
```

to prevent the same actions to take place when the user is just moving the pointer over the screen, without having grabbed a pole or a zero. In this callback, other calculations can also take place, like determining current position of the pole/zero and then calculate the present system.

With this procedure, the callback for the caption of the object is always set to a certain string, while the callbacks concerning moving and releasing the object are only set when the object is captured, and taken away when the procedure is finished. This is due to that the `ButtonDownFcn` is strictly connected to a certain object. That is, the callback will only execute when this object is selected. When the object is moved or released, the callbacks `WindowButtonMotionFcn` and `WindowButtonUpFcn` must be connected to the whole figure. This means that these callbacks, if they were always set, would execute as soon as the pointer is moved in the window.

5.5 Example

Consider system 2. The default value of the sampling period h is 0.5. In the z-plane you can see how the poles and the zero are moving when h is increased from 0.5 to 4 (see Fig. 2). The same continuous system can, when sampled, result in discrete time systems with various properties.

6. Aliasing

6.1 Theory

When sampling a signal, a lot of information can be lost if the sampling interval is too long. The Nyquist frequency $\omega_N = \omega_s/2$, where ω_s is the sampling frequency, plays an important role. If the continuous time signal has any frequency higher than ω_N, alias frequencies will appear. According to the Sampling Theorem the sampling frequency must be at least twice as high as the highest frequency component in the signal to avoid frequency aliasing. If this is not the case, alias frequencies will appear, and the original signal can not be distinguished from these.

To avoid aliasing it is necessary to prefilter the signal before sampling. A prefilter with a proper bandwidth removes the unwanted high frequencies. A

standard second-order analog filter has the transfer function

$$G_f(s) = \frac{\omega^2}{(s/w_B)^2 + 2\zeta\omega(s/\omega_B) + \omega^2}$$

where ω_B is the bandwidth of the filter. Higher order filter are obtained by cascading filters like this one.

6.2 Goal

The purpose of the `Aliasing module` is to illustrate what happens to a sinusoidal signal when it is sampled with different frequencies and also the effect of using prefilters with different bandwidths.

6.3 User Aspects

The frequency of the sinusoidal, the Nyquist frequency and possible alias frequencies are shown in a frequency plot. The sampling interval and the signal frequency can be altered. There is an optional filter, which is a Bessel filter of 6:th order with $\omega = 1.90, 1.69, 1.61$ and $\zeta = 0.49, 0.82, 0.98$. The bandwidth of the filter can be changed and is also seen in the frequency plot. The original and the sampled signals are also plotted.

6.4 Implementation

The folded frequency is determined according to the following while-loop,

```
while w_test > 3.14/h,
      w1 = abs(2*3.14/h-w_test);
      w_test = w1;
      if w_test < 3.14/h,
            w_wn_vec = w1*ones(1,length(A));
            pw_wn = plot(w_wn_vec,A,'g--');
            set(pw_wn,'Linewidth',2);
      end;
end;
```

The 6:th order Bessel filter is created using `conv`, which convolves two polynomials of the type described in page 10 part 6.1.

The filtered signal is obtained with `lsim` which gives the time response of a given polynomial transfer function, in this case, the filter.

The continuous time signal is actually a sampled signal, but sampled very fast in comparison with the actual sampled signals.

6.5 Example

Consider a signal with frequency 3.2 rad/s sampled with $h = 1.75$. The sampling clearly leads to aliasing problems. See a) in Fig. 3.

If though the signal is prefiltered with a 6:th order Bessel filter, with bandwidth 0.98 rad/s, the high frequency components of the signal are taken away and aliasing can not occur. See b) in Fig. 3.

In this case the signal consists of only one frequency component due to the fact that it is a pure sinusoidal and therefor the whole signal is extinguished. If this sinusoidal was a disturbance acting on another signal, this signal could be reconstructed properly with the prefilter applied.

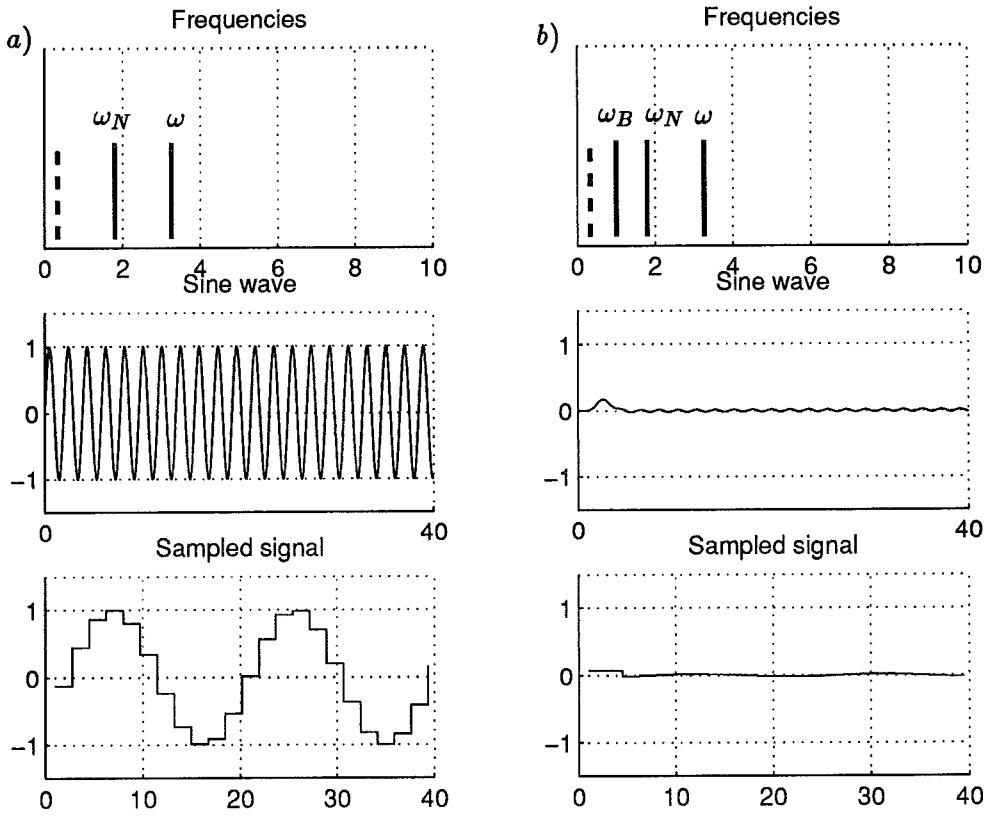

Figure 3. a) Sampling of a signal with frequency 3.2 rad/s with $h = 1.75$.
b) The same signal and sampling period but prefiltered with a 6:th order Bessel filter whose bandwidth is 0.98 rad/s.

7. Observability

7.1 Theory

Observability is a question of whether it is possible to reconstruct the states of a system given only the input and output signals. If this is possible, the system is observable. To determine whether a system is observable or not you can look at the observability matrix W_o. For a $n-$th order sampled system the observability matrix is

$$W_o = \begin{pmatrix} C \\ C\Phi \\ \vdots \\ C\Phi^{n-1} \end{pmatrix}$$

If the rank of the observability matrix is less than n, there exist unobservable states. These states belong to the null space of W_o and are spanned by the vector perpendicular to the vector spanned by the observability matrix. These unobservable states can not be identified because all initial states that lie in this subspace give zero output.

7.2 Goal

The purpose of the `Observability module` is to illustrate the behavior of a system that has unobservable states and to compare it with observable systems.

7.3 User aspects

In this demo there are three different systems to choose between. The first one is an unobservable system. Its state-space description is

$$x(kh+h) = \begin{pmatrix} 1.1 & -0.3 \\ 1 & 0 \end{pmatrix} x(kh)$$

$$y(kh) = \begin{pmatrix} 1 & -0.5 \end{pmatrix} x(kh)$$

the second system is an observable system, with real eigenvalues, that looks like

$$x(kh+h) = \begin{pmatrix} 1 & 1 \\ -0.16 & 0 \end{pmatrix} x(kh)$$

$$y(kh) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(kh)$$

and the third system is also observable, but with complex eigenvalues

$$x(kh+h) = \begin{pmatrix} -0.5 & 1 \\ -0.6 & 0 \end{pmatrix} x(kh)$$

$$y(kh) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(kh)$$

The initial state is movable and plotted in a phase plane diagram. When a move is completed, the phase plane and the output are plotted.

When system 1 or 2 is investigated there is a button called "Interesting". When this button is pressed, a number of green circles appear in the phase plane diagram. The result, when letting the initial state be located in these ones, gives interesting features of the system.

7.4 Implementation

To compute the observability matrix `obsv` is used.

In the case of the unobservable system, the unobservable subspace and the line perpendicular to this one are created in the following code,

```
wo = obsv(phi,C);
l1 = (-2:0.1:2);
l2 = (wo(1,2)/wo(1,1))*l1;
l3 = (-wo(1,1)/wo(1,2))*l1;
l4 = l3-1.5;
plot(l1,l2,'c-');
plot(l1,l3,'b-');
plot(l1,l4,'b--');
```

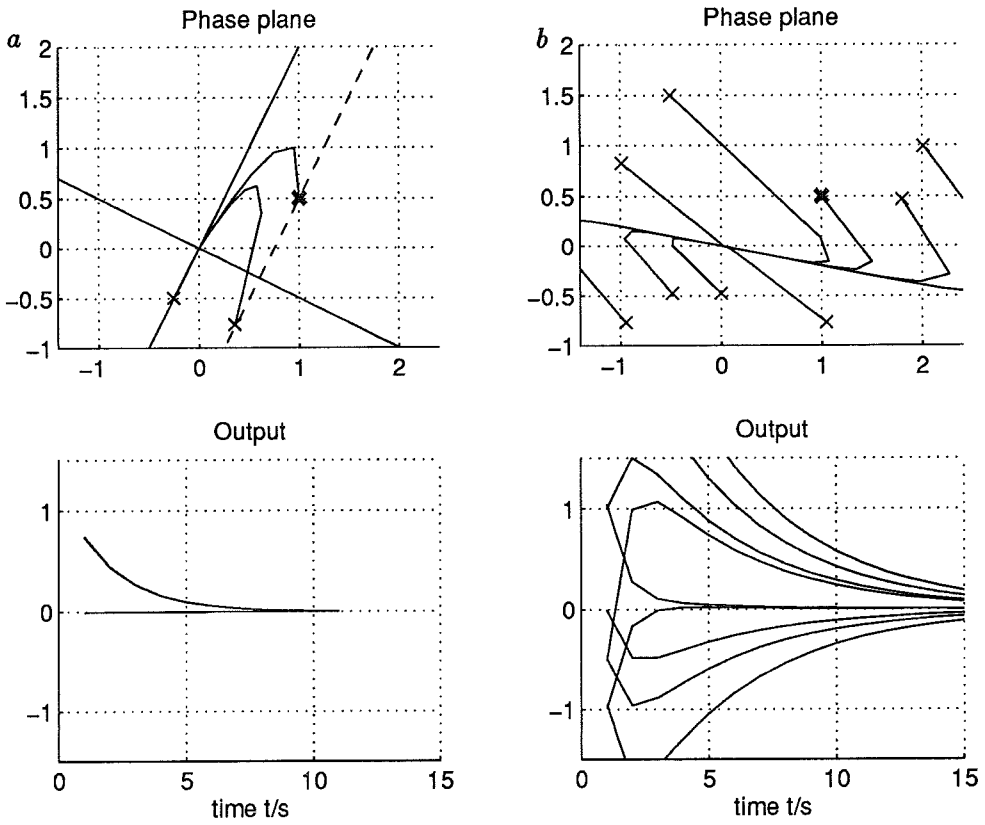

Figure 4. a) The phase and output of system 1, when starting in the unobservable subspace and on a line parallel to it.
b) The phase and output of system 2, when starting in some different locations.

where the line made up by `l1` and `l2` is the vector spanned by the observability matrix, the one made up of `l1` and `l3` is the unobservable subspace and the third one is just a line parallel to this one. The last one is only used for demonstrational purposes.

`Dinitial` is used for calculating the output- and state-vectors that are plotted.

Moving the initial state is accomplished in the same way as the poles and zeros where moved in the `Sampling module`, page 8 part 5.4.

7.5 Example

Consider system 1, which has unobservable states. The outputs, when the system starts anywhere on the dotted line, coincide, while the output is zero when the system starts on the full line parallel to the dotted one. See a) in Fig. 4. The full line represents the unobservable subspace.

System 2 is observable and has real eigenvalues. In b) Fig. 4 is shown that the states approach the origin along two vectors. These are the eigenvectors of Φ. The one corresponding to the slowest eigenvalue of Φ, is the one along which most of the trajectories approach the origin. Only when starting on the "fast" eigenvector, the trajectories follow that direction.

8. Numerics

8.1 Theory

A system can be represented in many different ways. Two of these are the diagonal form and the controllable form. The diagonal form is a diagonalized version of Φ. If Φ has distinct eigenvalues there exists a T such that

$$T\Phi T^{-1} = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}$$

where $\lambda_1 \cdots \lambda_n$ are the eigenvalues of Φ, that is, the poles of the system. In the controllable form the Φ matrix looks like

$$\Phi = \begin{bmatrix} -a_1 & -a_2 & \dots & -a_{n-1} & -a_n \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

where $a_1 \cdots a_n$ are the parameters in the characteristic polynomial

$$z^n + a_1 z^{n-1} + \cdots + a_n$$

of the process.

8.2 Goal

The purpose with the `Numerics module` is to illustrate that a change in the parameters $a_1 \cdots a_n$ in the controllable form has a much larger effect on the system then the same percentual change in the parameters $\lambda_1 \cdots \lambda_n$ in the diagonal form.

8.3 User aspects

There is one system to investigate, which is represented in both diagonal and controllable form. With eigenvalues $\lambda_i = 0.2, 0.3, 0.5, 0.7$ the characteristic polynomial looks like

$$(z - 0.2)(z - 0.3)(z - 0.5)(z - 0.7)$$

The same polynomial can be written as

$$z^4 - 1.7z^3 + 1.01z^2 - 0.247z + 0.021$$

where the numbers are a_i, $i = 1 - 4$. After having chosen one of these forms, the effect of changing the parameters can be investigated according to location of poles and step response. The given system can also be changed to another 4:th order system.

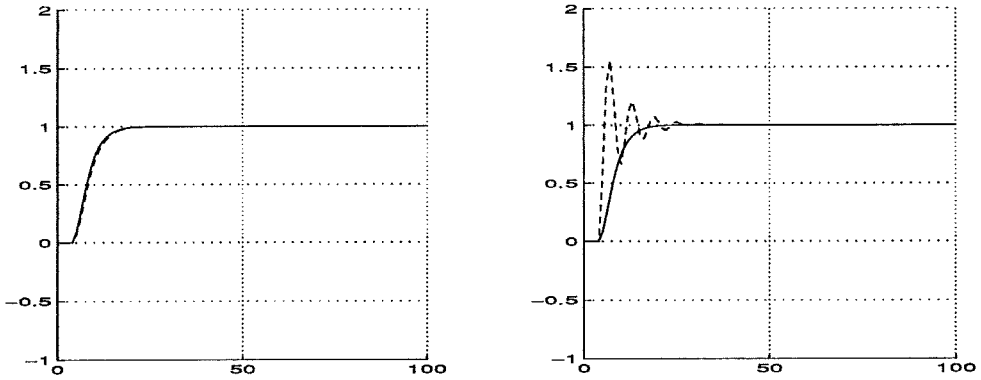

Figure 5. Step response for diagonal (left) and controllable (right) form when the parameter λ_1 is changed from 0.2 to 0.38 and a_1 is changed from -1.7 to -1.2. The dashed curves represent the system after the change.

8.4 Implementation

The discrete time step response is computed using `dstep`.

8.5 Example

Considering the system in controllable form, a change in the parameter a_1 from -1.7 to -1.2 makes the transient become oscillatory, while a change in λ_1 in the diagonal form from 0.2 to 0.38 does hardly not change the step response at all. See Fig. 5.

9. PID-control

9.1 Control algorithm

The PID-controller is a standard tool for solving process control problems. The control action consists of three parts, the P-part that is proportional to the error, the I-part that is proportional to the integral of the error and the D-part that is proportional to the derivate of the error. The continuous algorithm looks like

$$U(s) = K\left[bU_c(s) - Y(s) + \frac{1}{sT_I}(U_c(s) - Y(s)) - \frac{T_D s}{1 + \frac{T_D s}{N}}Y(s)\right]$$

An Euler approximation of the integral part and a backward-difference approximation of the derivate part give the sampled form

$$u(kh) = K\left[bu_c(kh) - y(kh) + \frac{h}{T_I(q-1)}e(kh)\right.$$

$$\left. - \frac{T_D}{h + \frac{T_D}{N}} \cdot \frac{q-1}{q - \frac{T_D}{Nh+T_D}}y(kh)\right]$$

See [1] . This algorithm includes a weighting b on the reference signal. With $b = 1$, as it is when there is no weighting, an overshoot in the step response

can appear. To avoid this, b can be set to a value lower than 1. Another modification of the textbook algorithm is that the derivate part is filtered. This is due to the fact that a pure derivate neither can nor should be implemented. The filter is influenced by the parameter N, which is the derivate gain at high frequences. N is called maximum derivate gain. Typical values of N are in the range $10 - 20$. The algorithm used also has an optional anti-reset-windup feature, which prevents integrator windup to occur, when the control signal is limited.

9.2 Tuning methods

Two different tuning methods are used, Ziegler Nichols and KT (Kappa Tau). Considering the KT-method see [2]. Both Ziegler Nichols method and Kappa Tau consist of two different ways of determining the PID-parameters, the Step Response method and the Ultimate-Sensitivity method.

In the Step Response method, the unit step response of the process is determined experimentally. Draw the tangent to the step response with the steepest slope. The intersection of this tangent with the vertical axes gives a and the intersection with the horizontal axes gives L. The slope of the tangent is R.

In the Ultimate-Sensitivity method, the key idea is to determine the point where the Nyquist curve of the open-loop system intersects the negative real axis. This is accomplished by making the controller pure proportional and then increase the gain until the closed-loop system reaches the stability limit. The gain K_u and the oscillation period T_u, at this point, are then determined.

Ziegler Nichols tuning method With Ziegler Nichols, the PID-parameters are calculated as

$$\begin{aligned} K &= 0.6K_u \\ T_I &= T_u/2 \\ T_D &= T_u/8 \end{aligned}$$

when using the Ultimate-Sensitivity method. When using the Step Response method, they are computed according to

$$\begin{aligned} K &= 1.2/RL \\ T_I &= 2L \\ T_D &= 0.5L \end{aligned}$$

Kappa Tau tuning method The KT method is based on three parameters, instead of, as in the Ziegler Nichols method, only two. The additional parameter is the gain ratio κ, defined as

$$\kappa = \frac{1}{K_p K_u}$$

where K_p is the static gain of the process. This method has been evaluated for two different values of maximum sensitivity M_s, 1.4 and 2.0. In this implementation $M_s = 1.4$ is used, which gives less overshoot in the step response than $M_s = 2$. In the Ultimate Sensitivity case, when the process is stable and

has no integrator, the PID-parameters can be calculated from the following formulas

$$
\begin{aligned}
K &= K_u \cdot 0.33e^{-0.31\kappa - 1.0\kappa^2} \\
T_I &= T_u \cdot 0.76e^{-1.6\kappa - 0.36\kappa^2} \\
T_D &= T_u \cdot 0.17e^{-0.46\kappa - 2.1\kappa^2} \\
b &= 0.58e^{-1.3\kappa + 3.5\kappa^2}
\end{aligned}
$$

When using the Step Response method on a stable process with an integrator, the PID-parameters are computed according to

$$
\begin{aligned}
K &= 1/a \cdot 5.6e^{-8.8\kappa + 6.8\kappa^2} \\
T_I &= L \cdot 1.1e^{6.7\kappa - 4.4\kappa^2} \\
T_D &= L \cdot 1.7e^{-6.4\kappa 2.0\kappa^2} \\
b &= 0.12e^{6.9\kappa - 6.6\kappa^2}
\end{aligned}
$$

With the KT-tuning method, not only K, T_I and T_D are determined, but also b, the weighting on the reference signal.

9.3 Goal

The purpose of the `PID module` is to illustrate automatic tuning and the influence on the step response and on the control signal when the sampling period or the PID-parameters are changed. The aim is also to show the effect of using anti-reset-windup when the control signal is limited.

9.4 User aspects

There are four optional systems to choose among and also the opportunity to create one. The first system is a minimum phase system with transfer function

$$G(s) = \frac{1}{(s+1)^4}$$

The second system has a zero in the right half plane

$$G(s) = \frac{1-2s}{(s+1)^3}$$

The third one is a system with time delay that looks like

$$G(s) = \frac{e^{-5s}}{(s+1)^3}$$

and the fourth system has an integrator. The transfer function is

$$G(s) = \frac{1}{s(s+1)^3}$$

The first two systems have been taken from [3], where they have been further investigated. System 3 and 4 are used as examples in [2].

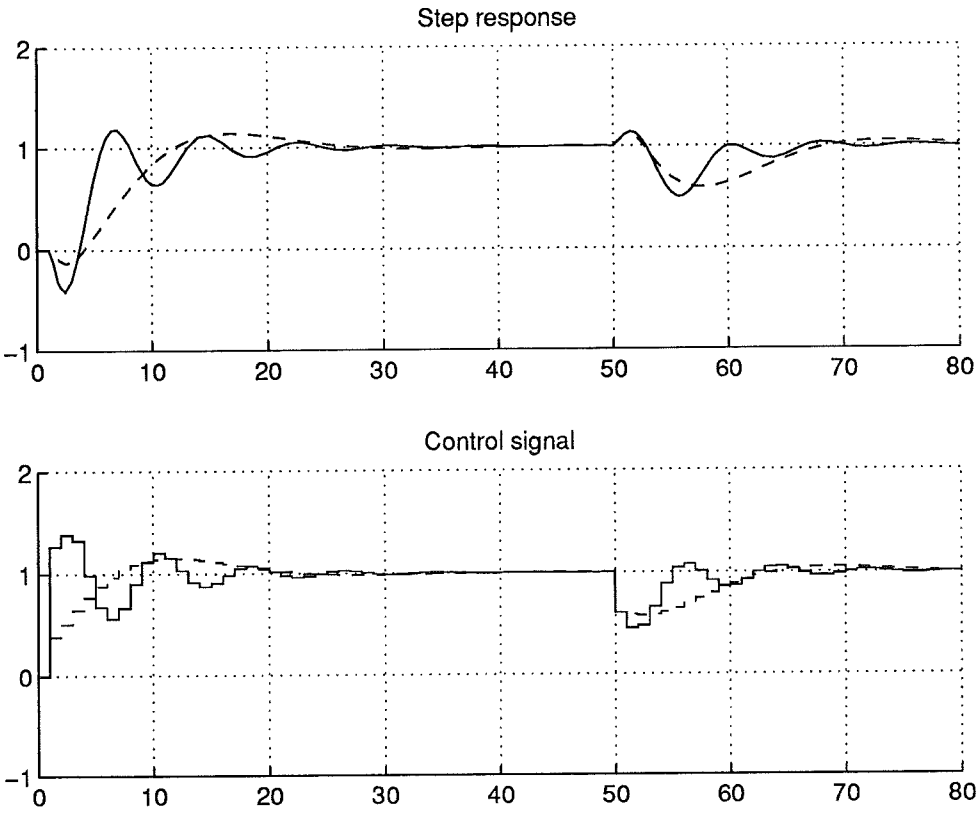

Figure 6. Autotuning for system 2 with ZN-method (full line) and KT-method (dashed line).

When using a tuning method, the sampling period h is calculated according to

$$h = \frac{1.8 \cdot T_D}{N}$$

This sampling period is longer than usually recommended in textbooks. The reason for this is to keep the simulation time reasonably short.

After having selected a process, there is a choice between doing the tuning yourself or using one of the two tuning methods mentioned earlier. After having chosen tuning method, there is the possibility to change the PID-parameters manually, to investigate the effects in the step response and the control signal. Limitations can also be applied to the process. The simulations can be run with or without anti-reset-windup.

9.5 Implementation

The simulations are done in Simulink. The implementation can be seen in Fig. 8 with anti-reset-windup and in Fig. 9 without anti-reset-windup. The regulator is in both cases implemented in discrete form according to the formula in page 15 part 9.1. The weighting of the reference signal, b, is only determined when using Kappa Tau tuning. In all other cases it is set to one. The value of the limitation of the control signal and the saturation in the anti-reset-windup have the same values.

If a system the user creates includes a time-delay, it has to be treated in a special way before the controller parameters can be calculated with one

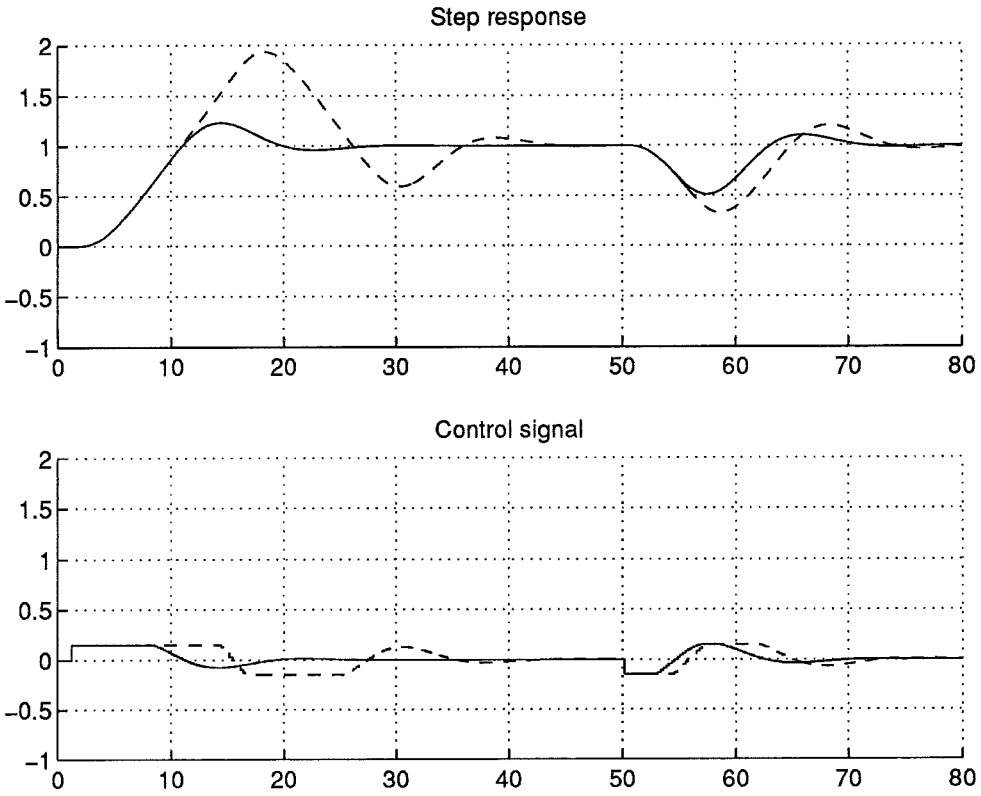

Figure 7. PID-control of system 4 with limitations on the control signal. The result without anti windup is shown in dashed lines and the result with anti windup in full lines.

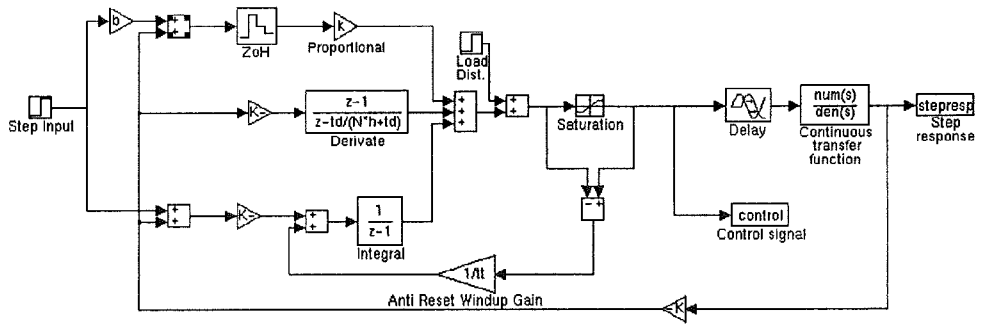

Figure 8. Simulink implementation of PID-controller with anti-reset-windup.

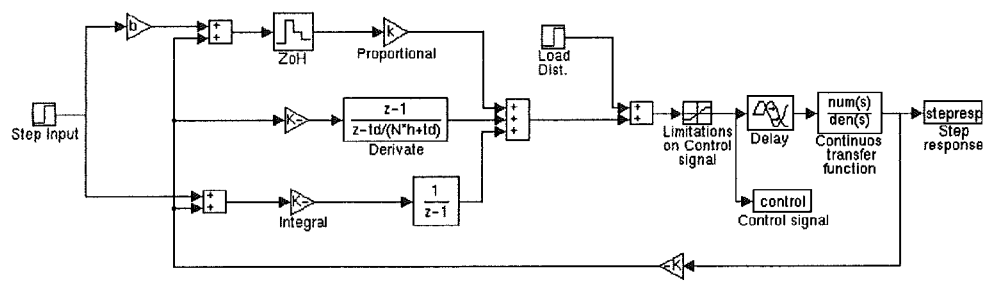

Figure 9. Simulink implementation of PID-controller without anti-reset-windup.

of the tuning methods. The time-delay e^{-s*del} ,where *del* is the delay, is approximated with a transfer function of second order. This is done as follows,

```
if abs(del)<eps,
       numd=1;
       dend=1;
       return;
end;
numd=[del^2 -6*del 12];
dend=[del^2 6*del 12];
```

The resulting transfer function for the process is calculated like

```
num = conv(num,numd);
den = conv(den,dend);
```

where conv is a function that convolves the vectors given inside the parenthesis.

The controller parameters are calculated using one of the two specified tuning method. In all cases, but for when using KT tuning on system 4, the Ultimate-Sensitivity method is applied. In the case of KT tuning on system 4, the Step Response method is instead used. The PID-parameters in this case, and also for system 3 with KT tuning and system 4 with Ziegler Nichols tuning, are not calculated in the program. They are instead taken from [2], where they are already determined. The tuning of the other systems are done in the program, according to the equations mentioned earlier. In the case of the Ultimate-Sensitivity method, `margin` is used, in which K_u and $2\pi/T_u$ are calculated.

When choosing an optional system with "Make own" the Ultimate Sensitivity method is always used. Unfortunately it is not always possible to directly compute the parameters with this method. When a problem arises, an error message appears and the tuning must be made manually.

9.6 Example 1

Consider system 2

$$G(s) = \frac{1 - 0.2s}{(1+s)^3}$$

In Fig. 6 autotuning with Ziegler Nichols method (full line) and KT method (dotted line) can be seen.

9.7 Example 2

Consider system 4

$$G(s) = \frac{1}{s(s+1)^3}$$

when using Ziegler Nichols tuning method. Put a limit on the control signal that is ± 0.15. The step response and control signal are shown in dashed lines in Fig. 7. The result when adding an anti reset windup feature to the regulator is shown in full lines. We can see that anti reset windup reduces the overshoot.

10. Robot mechanism process

10.1 Theory

This system consists of a motor that drives a load consisting of two masses coupled with a spring with spring constant, k. The moments of inertia are J_1 and J_2 and the damping in the spring is d. See Fig. 10. The input signal is the motor current I and the output is the angular velocity

$$\omega_2 = \sqrt{k\frac{J_1 + J_2}{J_1 J_2}}$$

The process is described by

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 & -1 \\ \alpha - 1 & -\beta_1 & \beta_1 \\ \alpha & \beta_2 & -\beta_2 \end{bmatrix} x + \begin{bmatrix} 0 \\ \gamma \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ \delta \\ 0 \end{bmatrix} v$$

$$y = \begin{bmatrix} 0 & 0 & \omega_0 \end{bmatrix} x$$

where

$$\alpha = J_1/(J_1 + J_2)$$

$$\beta_1 = d/(J_1\omega_0)$$

$$\beta_2 = d/(J_2\omega_0)$$

$$\gamma = k_I/(J_1\omega_0)$$

$$\delta = 1/(J_1\omega_0)$$

The states in this representation are

$$x_1 = \varphi_1 - \varphi_2$$

$$x_2 = \omega_1/\omega_0$$

$$x_3 = \omega_2/\omega_0$$

This mechanism is further described in [1] Chapter 9.

10.2 Goal

The purpose of the `Robot mechanism module` is to illustrate what different properties a system can get when some physical parameters are changed.

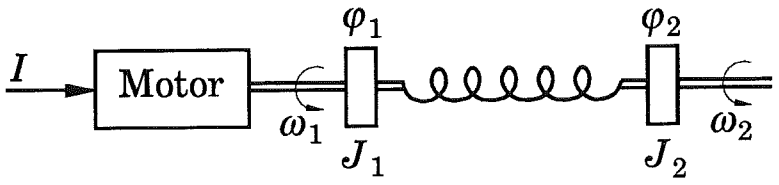

Figure 10. Robot mechanism process.

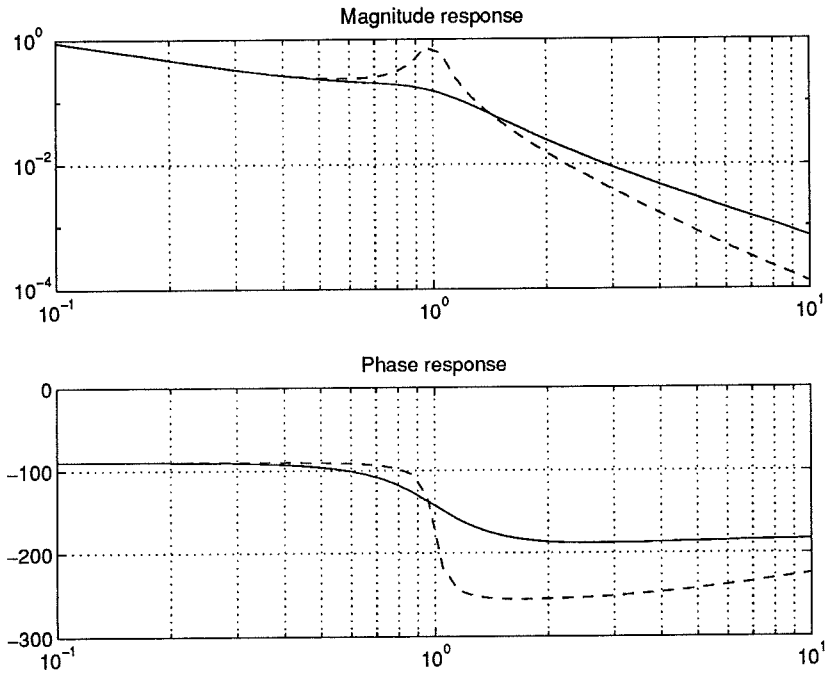

Figure 11. Bode plots for the robot mechanism process with damping $d = 0.05$ (dashed lines) and $d = 0.76$ (full lines).

10.3 User aspects

It is possible to change k, d, J_1 and J_2. The result of these changes can be seen in pole/zero plots, impulse response and Bode plots. The nominal values of the parameters are: $k = 1$, $d = 1$, $J_1 = 10/9$ and $J_2 = 10$.

10.4 Implementation

The Bode-plots are calculated with `bode` and then the magnitude is plotted with the function `loglog` and the phase with `semilogx`. The impulse response is made with `impulse`.

10.5 Example

Changing the damping d from 0.05 to 0.76 makes the impulse response much less oscillatory as can be seen in Fig. 12. The Bode plots are seen in Fig. 11. The dashed lines shows the result with $d = 0.05$ and the full lines $d = 0.76$.

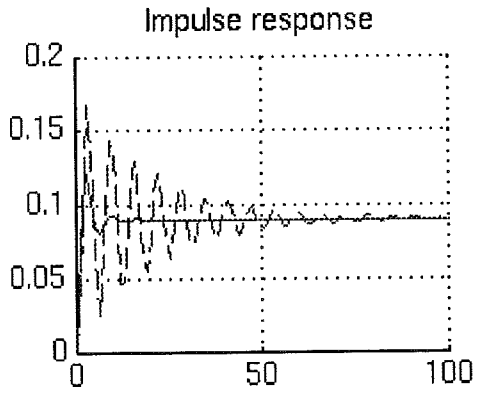

Figure 12. Impulse response for the robot mechanism process with damping $d = 0.05$ (dashed lines) and $d = 0.76$ (full lines).

11. Pole placement design by state feedback

11.1 Theory

State space design is based on the internal model of the system. The purpose is to arrange a feedback so that the poles of the closed-loop system get prescribed values. The state feedback is linear and can be expressed as

$$u(kh) = -Lx(kh) + l_c u_c$$

(see Fig. 13). In the formal specification of the problem, the design parameters are the sampling period and the closed-loop poles in the characteristic equation

$$z^2 + p_1 z + p_2$$

The problem is thus often stated as specifying the sampling period h, the natural frequency ω and the relative damping ζ in the characteristic equation

$$s^2 + 2\zeta\omega s + \omega^2$$

which admit a more direct physical interpretation. If all the state variables can be measured directly the aim is to find a matrix L such that the matrix $\Phi - \Gamma L$ has prescribed eigenvalues, given by the specifications.

If the states can't be measured directly an observer has to be used, which determines the states from the inputs and outputs of the system (see Fig. 14). There are several kinds of observers. The one used in this application is of Kalman type and is described by

$$\hat{x}(k+1|k) = \Phi\hat{x}(k|k-1) + \Gamma u(k) + K\left[y(k) - C\hat{x}(k|k-1)\right]$$

K is determined by specifying the observer polynomial so that the matrix $\Phi - KC$ gets desired eigenvalues. The design parameters are the same as for the state feedback.

A controller of this type may not be able to handle load disturbances. In this case an appropriate method to overcome this problem is to introduce an integrator which introduces a new state that integrates the error $e = u_c - y$ (see Fig. 13 and 14).

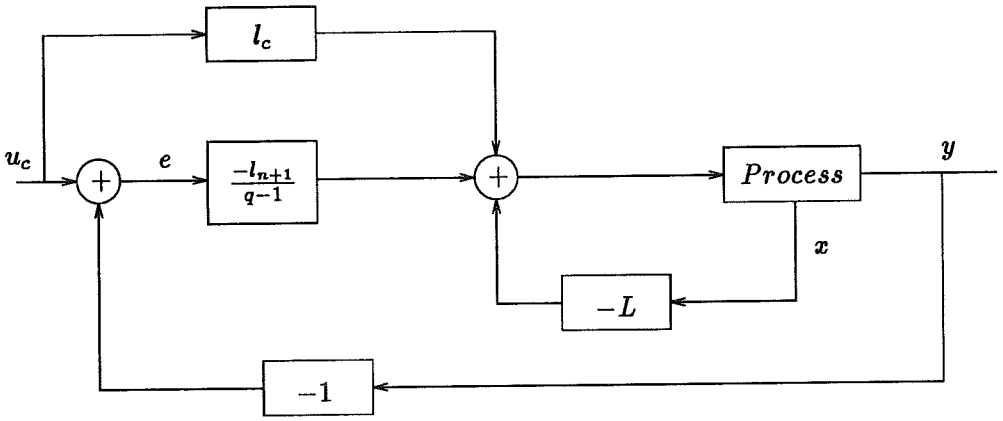

Figure 13. State feedback with integrator.

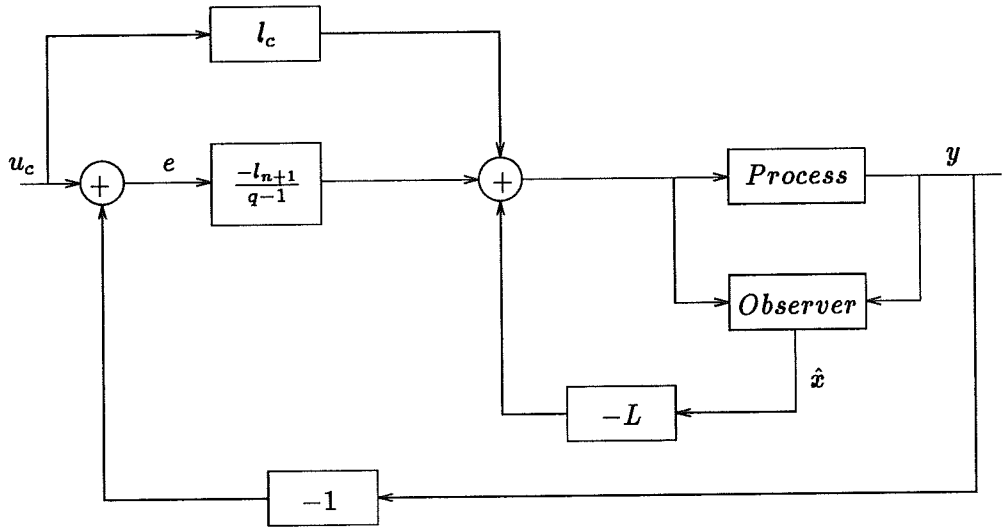

Figure 14. State feedback with observer and integrator.

11.2 Goal

The purpose with the `State Feedback module` is to illustrate one way of accomplishing pole placement design. It is also supposed to show the effect of changing the design parameters, the good influence on the system when using an integrator and also the importance of a good observer design.

11.3 User aspects

There are two predefined systems and also the possibility to create one. The first system is the double integrator with transfer function

$$G(s) = \frac{1}{s^2}$$

and the second one is the harmonic oscillator which transfer function is

$$G(s) = \frac{1}{s^2 + 1}$$

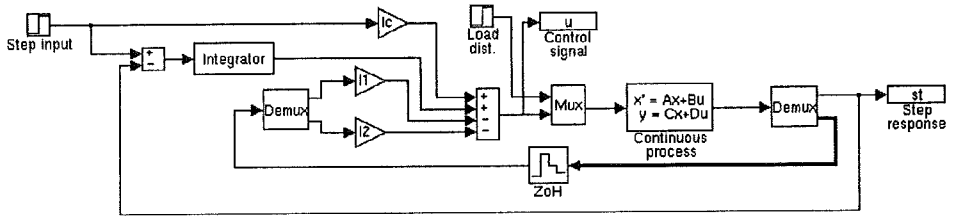

Figure 15. Simulink implementation of state feedback without observer.

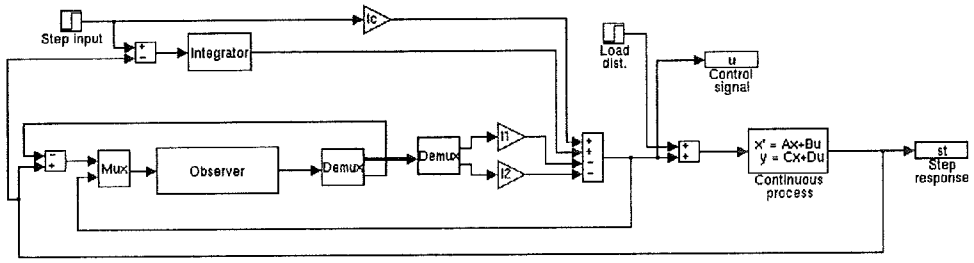

Figure 16. Simulink implementation of state feedback with observer.

When a system has been selected, it is possible to change h, ω, and ζ for the state feedback, to investigate the response of the system. The use of integral action and observer are optional when running the simulations. When using observer, ω and ζ in the observer polynomial, can also be chosen freely. After each change step response and control signal are plotted.

11.4 Implementation

The simulations are done in Simulink. The implementation can be seen in Fig. 15 without observer and in Fig. 16 with observer. The integrator block includes the third element of the L-vector, l_3, which has to be determined when integral action is applied. The block l_c is calculated as to give unit static gain form u_c to y. The process is given in continuous time. The observer is given as a state-space representation in discrete time.

To be able to apply state feedback to a system, the process, of order n, has to be controllable. This can be checked by using `ctrb` which calculates the controllability matrix W_c and then by using `rank` to determine whether W_c has rank n or not.

The desired closed loop polynomial is calculated according to,

```
ac=[1 2*om*zeta om*om];
rch = roots(ac)*h;
ad  = real(poly(exp(rch)));
```

where `ac` is the continuous time and `ad` is the discrete time characteristic polynomial.

11.5 Example

Consider the double integrator

$$G(s) = \frac{1}{s^2}$$

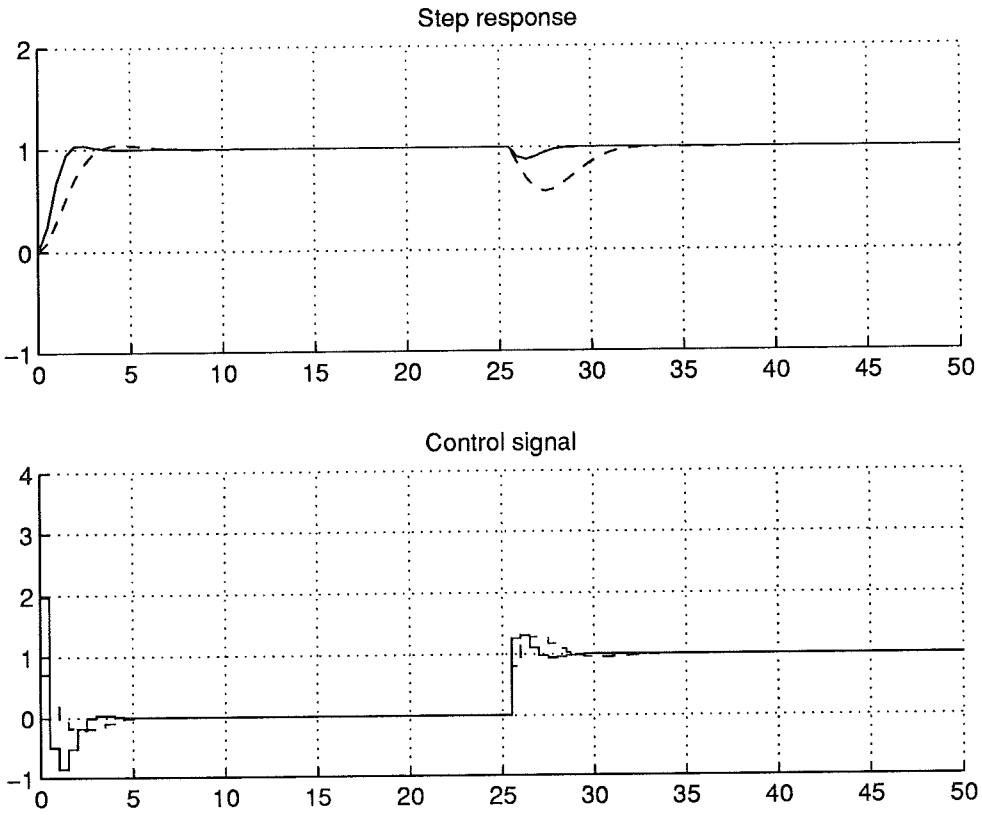

Figure 17. Step response and control signal for $\omega = 1$ (dotted line) and $\omega = 2$ (full line), when using state feedback design on the double integrator.

with sampled state-space representation

$$x(kh+h) = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix} x(kh) + \begin{pmatrix} h^2/2 \\ h \end{pmatrix} u(kh)$$

$$y(kh) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(kh)$$

The value of h is set to 0.5, the states are measured directly and the controller has an integrator. The full-line step response corresponds to $\omega = 2$ and the dotted one to $\omega = 1$ (see Fig. 17). It is clearly seen that the higher value of ω gives a faster response to changes in the reference signal and faster recovery after load disturbances.

12. Pole placement design based on input-output model

12.1 Theory

The pole placement design is in this case based on the input-output model of the system. The purpose is to find a control law of the form

$$R(q)u(kh) = -S(q)y(kh) + T(q)u_c(kh)$$

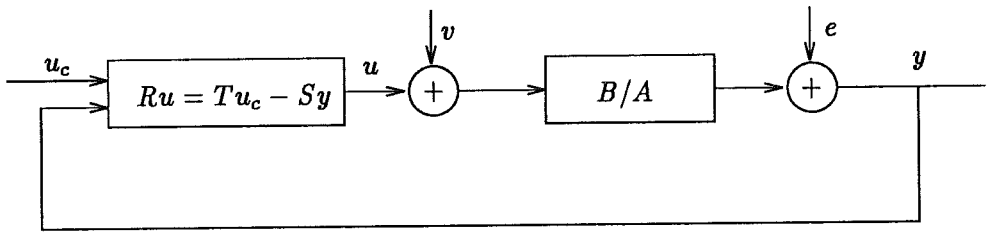

Figure 18. Closed loop system with RST-regulator.

such that the closed-loop system has the desired input-output relation given by

$$H_m(z) = \frac{B_m(z)}{A_m(z)}$$

and an observer with the characteristic polynomial $A_o(z)$. When controlling a system

$$H(z) = \frac{B(z)}{A(z)}$$

the specifications that have to be made, are the desired closed-loop transfer function, the observer polynomial, and whether there should be integral action or not. The design parameters are the sampling period h, the natural frequency ω, the relative damping ζ in

$$A_m(s) = s^2 + 2\zeta\omega s + \omega^2$$

and the location of the observerpole(s) α_o in

$$A_o(s) = (s + \alpha_o)^{degA_o}$$

The closed-loop discrete time characteristic polynomial looks like

$$A_m(z) = z^2 - 2e^{-\zeta\omega h}cos(wh\sqrt{1-\zeta^2})z + e^{-2\zeta\omega h}$$

When controlling a process with a regulator of this type, the closed loop system (see Fig. 18) obtained is

$$\frac{BT}{AR + BS} = \frac{B_m}{A_m}$$

The first step in the design method is to decide if there are zeros in $B(q)$ that may not be cancelled. The B-polynomial is then split in two parts, B^- and B^+, where B^- contains the zeros that are not to be cancelled and B^+ is monic. B^- must also be a part of $B_m = B^- B'_m$, which is the numerator of the desired closed loop transfer function. B'_m can be chosen in a way so that the stationary gain is 1. The degrees of A_o, R'_1 and S also have to be determined according to predefined formulas. The next step is to solve the equation

$$(z-1)^l AR'_1 + B^- S = A_o A_m$$

with respect to R'_1 and S. Finally

$$R = B^+(z-1)^l R'_1$$

and

$$T = B'_m A_o$$

12.2 Goal

The purpose of the `Pole placement design module` is to illustrate the input/output version of accomplishing pole placement design. It is also supposed to show the effects of changing specifications and design parameters and the good influence on the system when using an integrator.

12.3 User aspects

There are two predefined systems, the double integrator and the harmonic oscillator, which are the same ones as in the State space case. See page 25 part 11.3. There is also a possibility to make an arbitrary second order system. When a system has been selected it is possible to change the A_m-polynomial, the observer polynomial A_o, the sampling period, and whether there should be integrator or not. It is also possible to choose between cancelling or not cancelling the process zeros. After each modification the step response and the control signal are plotted.

12.4 Implementation

The characteristic polynomial of the closed-loop system is calculated in the following lines

```
Acm = [1 2*om*zeta om*om];
rch = roots(Acm)*h;
Adm  = real(poly(exp(rch)));
```

where `om` is the natural frequency ω and `zeta` is the damping ζ. `Adm` is the discrete closed-loop characteristic polynomial.

The RST-design is done by solving the Diophantine equation.

The step response with a load disturbance at $t = 50$ is created with

```
time=0:h:100;
v1 = zeros(50/h,1);
v = [v1;-0.2*ones(length(time)-length(v1),1)];
uc = ones(length(v),1);
n = size(conv(Ad,R),2)-size(conv(Bd,S),2);
[y1,x]=dlsim(conv(Bd,T),...
conv(Ad,R)+[zeros(1,n) conv(Bd,S)],uc);
[y2,x] = dlsim(conv(Bd,R),...
conv(Ad,R)+[zeros(1,n) conv(Bd,S)],v);
y = y1+y2;
[u1,x] = dlsim(T,R,uc);
[u2,x] = dlsim(S,R,y);
u = u1-u2;
```

where `uc` is the reference signal and `v` is the load disturbance coming in on the process. The total response is created by sending u_c through

$$\frac{BT}{AR+BS}$$

and sending v through

$$\frac{BR}{AR+BS}$$

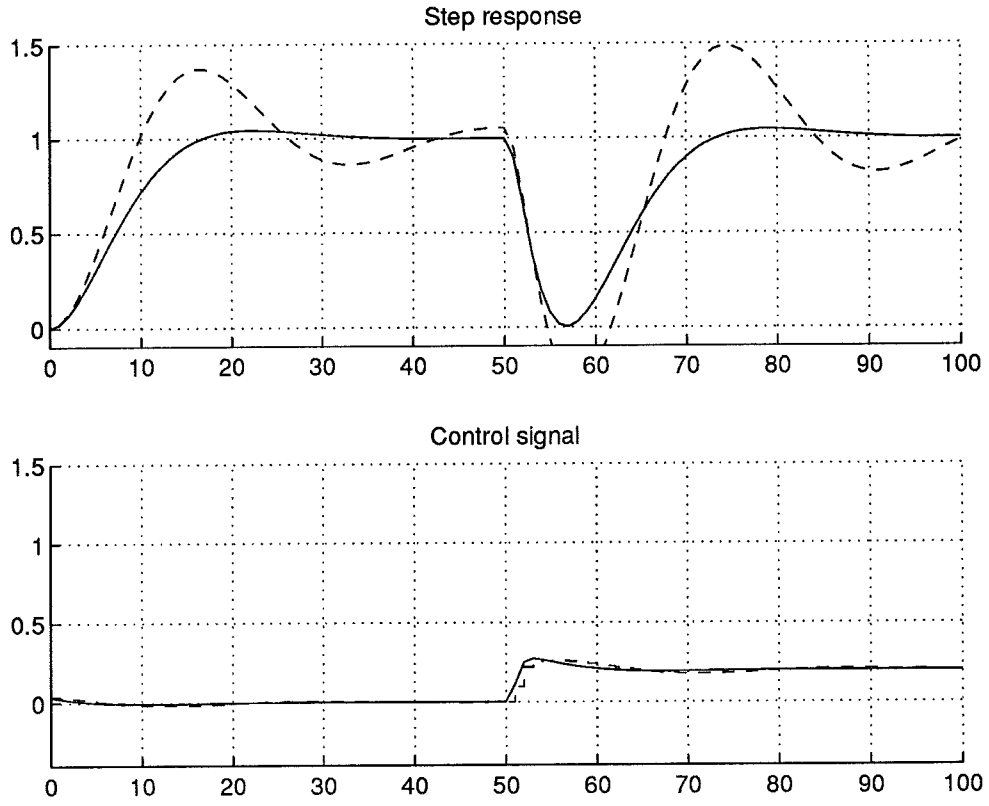

Figure 19. Step response and control signal for $\zeta = 0.3$ (dashed line) and $\zeta = 0.7$ (full line), when using Pole placement design on the double integrator. The controller has an integrator.

according to the formula of the closed-loop system

$$y = \frac{BT}{AR + BS} u_c + \frac{BR}{AR + BS} v + \frac{AR}{AR + BS} e$$

where $A =$ `Ad` and $B =$ `Bd`.

12.5 Example

Let the system be the double integrator. As can be seen in Fig. 19, $\zeta = 0.3$ gives a much more oscillatory behavior than $\zeta = 0.7$. The higher ζ the better damping. In this example the controller has an integrator.

13. Robustness

13.1 Theory

Designs of control systems are usually based on simplified models of the real process. Because of this it is important to understand how modeling errors will influence the closed-loop properties. An incorrect model can result in the closed-loop system becoming unstable. Considering a pole-placement design

on the model

$$H = \frac{B}{A}$$

Let H^0 be the pulse-transfer function of the real system. Assuming that H and H^0 have the same number of poles outside the unit disc and that H_m is stable gives that the closed-loop system is stable if

$$|H(z) - H^0(z)| < \left|\frac{H(z)T(z)}{H_m(z)S(z)}\right| = \left|\frac{H(z)}{H_m(z)}\right|\left|\frac{H_{ff}(z)}{H_{fb}(z)}\right|$$

for $|z| = 1$, where

$$H_{ff}(z) = \frac{T(z)}{R(z)}$$

and

$$H_{fb}(z) = \frac{S(z)}{R(z)}$$

This inequality is, however conservative. It is often sufficient to have good model precision only in certain frequency ranges.

13.2 Goal

The purpose of the `Robustness module` is to illustrate what happens to stability and performance when a regulator is designed for a nominal process and the process in reality is a different one.

13.3 User aspects

The process used in this module is the robot mechanism in [1] p.275, which is a third order system. The properties that can be changed are the damping d, the moments of inertia J_1 and J_2, and the spring constant k. Before changing these variables an RST-design should be done. This is accomplished by choosing h and the desired closed-loop characteristic polynomial A_m. The regulator is always designed with no zeros cancelled, with an integrator, and with the observer twice as fast as the desired closed-loop response. After having designed a regulator, the poles and zeros and the step response for the closed-loop system can be evaluated for different true processes by changing the physical parameters.

13.4 Implementation

The characteristic polynomial of the closed-loop system is calculated in the following lines

```
Acm1 = [1 2*om*zeta om*om];
Acm2 = [1 alf*om];
Acm = conv(Acm1,Acm2);
rch = roots(Acm)*h;
Adm  = real(poly(exp(rch)));
```

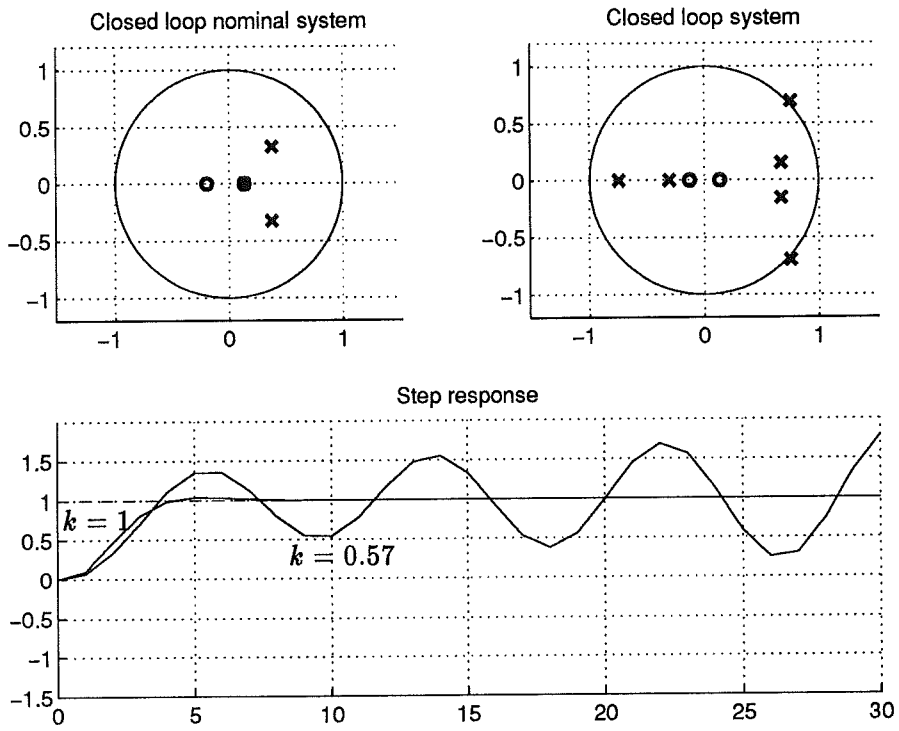

Figure 20. An RST-regulator, designed for the robot mechanism process with $k = 1$, used on this nominal system and on a system with $k = 0.57$.

where `om` is the natural frequency ω, `zeta` is the damping ζ and `alf` is a constant equal to 2. `Adm` is the discrete closed-loop characteristic polynomial.

The RST-design is done in the same way as in the `Pole placement module` page 29 part 12.4.

The step response is also created as in the `Pole placement module` except for the fact that in this case there is no load disturbance.

13.5 Example

An RST-regulator designed for the nominal process gives a fast and non oscillatory step response and the closed-loop poles are situated inside the unit circle. If this regulator is used on a process that is the same as the nominal one, but for that $k = 0.57$ instead of 1, the closed-loop system becomes unstable. This can be seen in Fig. 20. The regulator has $\zeta = 0.7$ and $\omega = 1$.

The closed-loop system has six poles and five zeros. Besides the clearly visible poles and zeros, there are some whose locations will be described here. In the nominal case there are four poles and three zeros located in 0.14. There is also one zero in -2.67. In the case where $k = 0.57$, there are three zeros in 0.14 and one zero in -2.34.

14. Noise

14.1 Theory

Stochastic models are often used to describe disturbances. A stochastic process

can be considered as a function $x(t,\omega)$. The realization of the process is an ordinary time function where $\omega = \omega_0$. Stationary Gaussian processes are completely characterized by their mean-value function and their covariance function. The covariance function is defined by

$$r_{xx}(s,t) = cov[x(s),x(t)] = E[x(s) - m(s)][x(t) - m(t)]^T$$

The spectral density has a good physical interpretation. The integral

$$2\int_{\omega_1}^{\omega_2} \Phi(\omega)d\omega$$

represents the power of the signal in a certain frequency band (ω_1,ω_2).

For a process

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

where $v(k)$ is white noise with zero mean and covariance R_1 the stationary covariance function can be calculated according to the formula

$$r(\tau) = \Phi^\tau P, \qquad \tau \geq 0$$

where

$$P = cov[x(k),x(k)]$$

is given by

$$P = \Phi P \Phi^T + R_1$$

14.2 Goal

The purpose of the `Noise module` is to illustrate how the covariance function, spectrum and realization varies when white noise is filtered through systems with different transfer functions.

14.3 User Aspects

In this module there are two different processes to chose among. The first one has a transfer function of the form

$$H(z) = \frac{b}{z+a}$$

and the second one looks like

$$H(z) = \frac{b_0 z + b_1}{z^2 + a_1 z + a_2}$$

The numerator is always adjusted to make the output variance equal to 1. The poles and zeros in the different processes are shown in a pole-zero plot and they can all be moved to arbitrary positions just by grabbing them with the pointer. The plots are immediately updated after each move. The covariance function, the spectrum and the realization are plotted. All realizations are done with the same seed for the white noise generator.

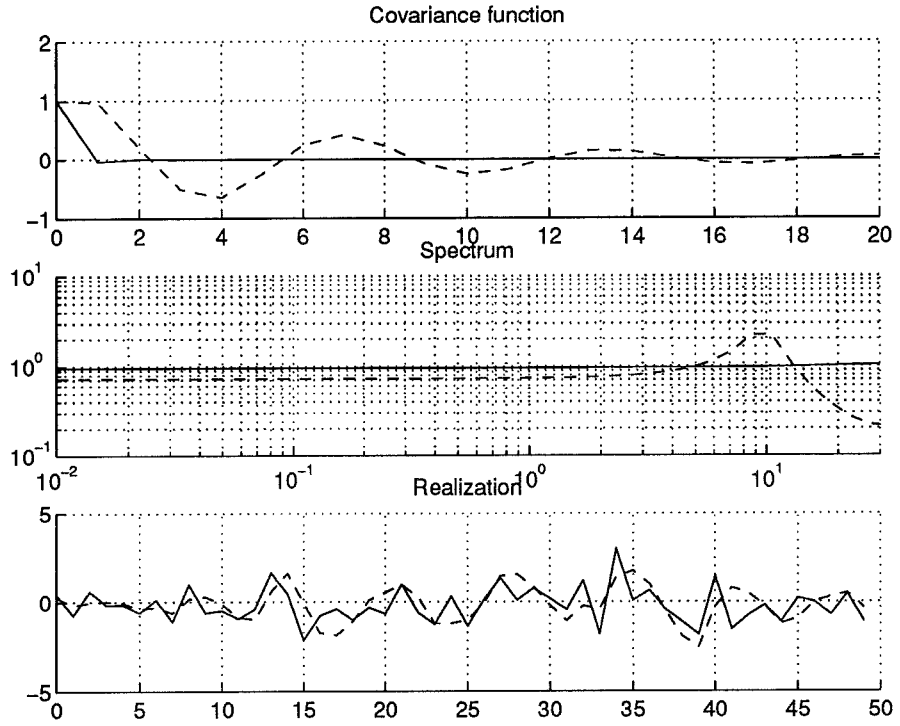

Figure 21. Comparison between a system with poles and zero in origo (full lines) and a system with poles near the unit circle (dashed lines), when sending white noise through it.

14.4 Implementation

The covariance function is calculated as follows,

```
Pk = eye(size(phi,1));
tau = 50;
rx   = Pk;
ry(1) = C*rx*C';
for k=1:tau,
     rx = phi*rx;
     ry(k+1) = C*rx*C';
end;
```

where ry is the covariance of the output y. Pk is the covariance of the state x and rx is the covariance function of the state.

Moving the poles and zeros is accomplished in the same way as in the Sampling module, page 8 part 5.4.

14.5 Example

Consider a system with transfer function

$$H(z) = \frac{z}{z^2}$$

Both poles and zeros are situated in the origin i.e. the process is a pure delay. The output is also white noise. The covariance function is a δ-function and the spectrum is flat.

Compare with a system with transfer function

$$H(z) = \frac{0.56z}{z^2 - 0.97z + 0.74}$$

In this case the poles are situated near the unit circle, in $0.48 \pm 0.71i$. When sending white noise through this system it takes a longer time for the covariance to settle to zero and the spectrum implies much lower damping.

15. Linear Quadratic control

15.1 Theory

Linear quadratic control (LQ-control) is an optimal design method. The problem is formulated as to minimize a criterion, which is a quadratic function of the states and the control signal. Considering the deterministic case, the process to be controlled is

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

with one input signal. The problem is then to determine a control strategy

$$u(k) = -L(k)x(k) + Mu_c(k)$$

that minimizes the loss function

$$J = E\left\{\sum_{k=0}^{N-1}\left[x^T(kh)Q_1x(kh) + \rho u^2(kh)\right] + x^T(Nh)Q_0x(Nh)\right\}$$

where Q_1 is positive semidefinite. The statitionary controller i.e the case when $N \to \infty$ is determined. The larger ρ is, the less is the allowed variation in the control signal.

15.2 Goal

The purpose of the `LQ module` is to illustrate the effect, in the step response and the control signal, when changing the weighting factor ρ.

15.3 User aspects

There are three predefined systems, the double integrator, a system with transfer function

$$H(z) = \frac{z + 0.5}{z^2 - z + 0.3}$$

and a system with transfer function

$$H(z) = \frac{z + 1.3}{z^2 - z + 0.3}$$

When one of these system is selected, the weighting factor ρ can be changed. The open- and closed-loop poles and zeros are shown in pole/zero plots and the step response and control signal are also plotted.

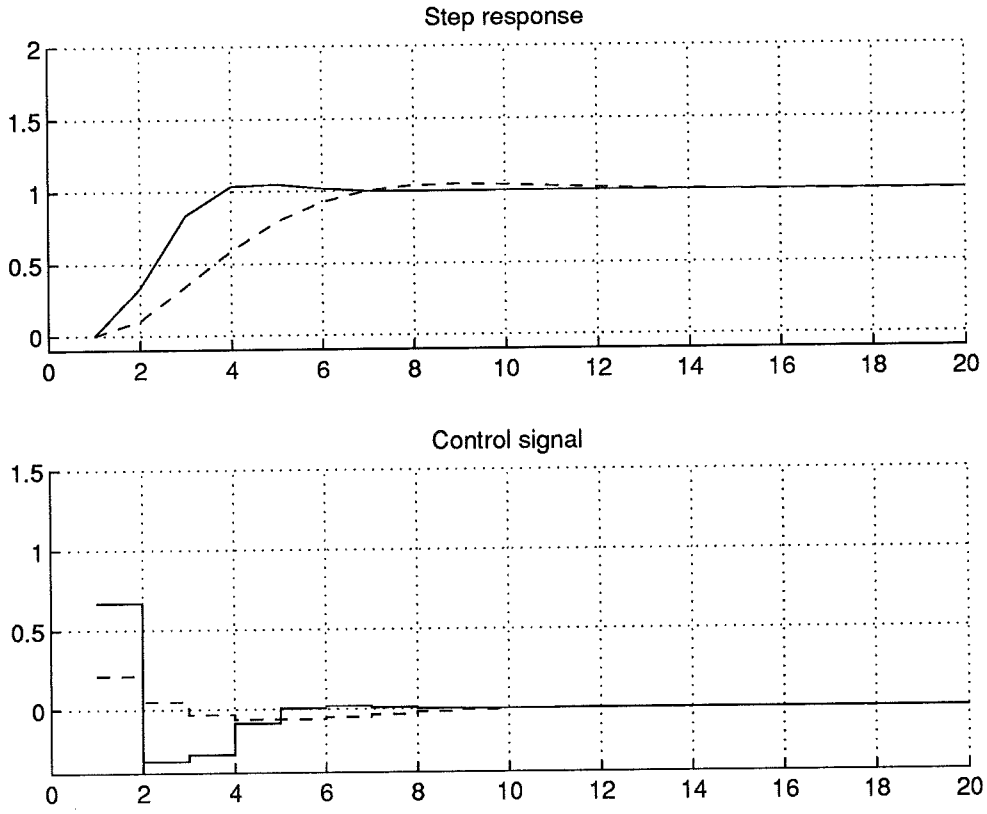

Figure 22. Step response and control signal for $\rho = 10$ (dashed line) and $\rho = 0.4$ (full line) when using LQ-control.

15.4 Implementation

To get a steady state value of one, the following lines are used to calculate M in

$$u(k) = -Lx(k) + Mu_c(k)$$

```
n = length(phi);
newPhi = eye(n)-phi+gam*L;
M = inv(C*inv(newphi)*gam);
```

The step response is computed using `dstep`.

15.5 Example

Consider system 1, the double integrator. The step response and the control signal when $\rho = 10$ can be seen in Fig. 22 in dashed lines. The full lines shows the result with $\rho = 0.4$. It can be seen that the step response becomes much faster with a lower value of ρ. This is due to the fact that when ρ is low a larger control signal is allowed.

16. How to use the demonstration tool

The main menu is reached by opening Matlab and then write `ccsdemo`. From this menu all modules can be started. When the user is finished with one module he/she can return to the main menu and choose another one.

It is possible to exit the program from either the main menu or the modules.

There are eleven modules which are all interactive. Variables and/or poles and zeros are possible to alter. The variables are altered either by sliders or editable windows, depending on the application.

The results are shown as plots that are immediately updated after each performed change.

If the user makes some kind of non allowed action, an error message is shown in the present window. This message disappears when a correction is made.

17. Conclusions

Matlab is a very pleasing environment to work in. The combination of calculations in Matlab, simulations in Simulink and the graphical user interface tools makes it very flexible. It is well adapted to use for purposes like the one dealt with in this Master Thesis Project.

There are though some small extra features that would be desirable. Concerning the graphical environment, it would be of interest to have the opportunity to disable a control. This is today possible to accomplish by not giving it a callback. When this is done, the control still reacts when touched, although no action is taken place. It would be more appropriate to completely freeze the control, when it is not supposed to be touched. Another desirable feature would be to have the possibility to easily make a control blink for a specified time. This feature could be used to get the users attention to important information.

The function environment, recommended in [5], is very practical when there is only need for one m-file per module. The disadvantage is though, when Simulink is used for the simulations, that all variables that are used in Simulink has to be declared global in the workspace. This makes the program sensitive to unwanted user actions. For example, the variables can be changed from the work space, which can cause problems. There is another way of handling variables when using functions, that is called `UserData` matrix. This is supposed to be a better way of protecting the variables against unwanted actions, although it is more circumstantial to use. During this master thesis, the `UserData` matrix has though not been further investigated.

The result of this Master Thesis Project will hopefully be a good complement to [1] during the course in Digital Control. Hopefully it will give the students a broader perspective and a deeper understanding concerning computer controlled systems. This demo will probably constitute a very useful learning environment.

Acknowledgement

I would like to express my sincere thanks to my supervisor Björn Wittenmark for all the help and support he has given me during my work. I also want to thank Mikael Johansson who has helped me a lot with my problems concerning Matlab. Finally, I thank the rest of the staff at the Department of Automatic Control, Lund Institute of Technology.

18. References

[1] KARL J. ÅSTRÖM & BJÖRN WITTENMARK. *Computer Controlled systems, theory and design.* Prentice-Hall International Inc. 1990.

[2] KARL J. ÅSTRÖM & TORE HÄGGLUND. *PID Controllers: Theory, Design and Tuning.* Instrument Society for Measurement and Control, 1995

[3] PER PERSSON. *Towards Autonomous PID Control.* PhD thesis ISRN LUTFD2/TFRT–1037–SE, Department of Automatic Control, Lund Institute of Technology, April 1992.

[4] PETER JERRAM & MICHAEL GOSNEY. *Multimedia Powertools.* Verbum Inc. & The Gosney Company Inc. 1993.

[5] *MATLAB, Building a Graphical User Interface.* The MATHWORKS Inc. 1993.

[6] *SIMULINK, A Program for Simulating Dynamic Systems* The MATHWORKS Inc. 1992.