ISSN 0280–5316
ISRN LUTFD2/TFRT--5566--SE

Dynamic Pictures as a Learning Tool in Control

Cesar A. Mendoza Serrano

Department of Automatic Control
Lund Institute of Technology
September 1996

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

Document name
MASTER THESIS

Date of issue
September 1996

Document Number
ISRN LUTFD2/TFRT--5566--SE

Author(s)
Cesar A. Mendoza Serrano

Supervisor
Björn Wittenmark and Mikael Johansson

Sponsoring organisation

Title and subtitle
Dynamic pictures as a learning tool in control

Abstract

The purpose of this master thesis is to create a computer program that presents some theoretical aspects of control theory using dynamic pictures, which show directly the influence of different parameters on a control system. It creates an interactive environment aided by virtual processes that use computer animation and real time simulation to enhance the learning process.

All the implementation has been done using Matlab and its graphical user interface. The use of Simulink was avoided in order to make faster animations.

Key words
education, computer controlled systems, dynamic pictures

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
44

Recipient's notes

Security classification

Contents

1. Introduction

In education it is important to present the theory in a friendly and easy way to understand. Both teaching and learning benefit from an environment where the student can participate actively in the learning process. This environment may be created by using sound, text, virtual processes, graphics, executable code, dynamic pictures, animation of algorithms and other multimedia tools.

The purpose of this master thesis is to create a computer program that presents some theoretical aspects of control theory using *dynamic pictures*, which show directly the influence of different parameters on a control system.

The program creates educational modules that help the student to understand better the control theory. It develops an interactive environment aided by "virtual processes" that use computer animation and real time simulation. With this, the learning process in control is enhanced.

All the implementation has been done using Matlab and its graphical user interface. The use of Simulink was avoided in order to make faster animations.

All the modules built include options like *Theory* and *Hints* that give the student a fast introduction to the control aspects treated in the module. To obtain information about how the module works the user is given the option *Help.*

This work is organized as follows:

In the Chapter of *Programming tools* there is an explanation of the main elements of the standard Matlab Graphical User Interface that were used.

The Chapter *Modules, animation and on-line simulation* deals with the methodology used to carry out the animation and the on-line simulation. It also shows the basic structure of how the modules were built.

The PD-control Chapter illustrates the concept of proportional and derivative control. Concepts of sample data systems are also analyzed. The control of a double integrator is used for this purpose.

The interaction between the inputs and the outputs of a multivariable process are given in the Chapter *Decoupling.* The process is analyzed in open-loop and in closed-loop using a PID-controller. The advantage to introduce a decoupler is also illustrated.

The Chapter *Controllability* uses a two-tanks system to explain the concepts of controllable and uncontrollable subspaces. It also shows a comparison between automatic and manual control. Constraints in the control signal due to physical limitations are treated. For this, the Bang-bang control is used.

State feedback design with and without observers are seen in Chapter *Robot mechanism process.* The response of the system due to different inputs can be analyzed.

A slalom game is used to show the behavior of typical systems like static, integrator, double integrator and non-minimum phase systems. This is seen in Chapter *Slalom.*

1.1 How to start

The package is started by giving the command *ccs2.* It was developed using Matlab Version 4.2c in a computer Solaris 2.

Acknowledgments

During my stay at the Department of Automatic Control in Lund, Sweden I had the opportunity to work in a friendly and nice environment. In particular I would like to thank Björn Wittenmark who has given me an invaluable guidance in this work. I would also want to thank Mikael Johansson for all his patience and help with Matlab.

Many thanks to Universidad Nacional Autonoma de Mexico (UNAM), which sponsored my graduate studies in England and to Lund Institute of Technology for the grant given to carry out this project.

I am also grateful to Professor Karl-Johan Åström and Dr. Martin Clark for giving me the opportunity of doing my Master Thesis project at the Department of Automatic Control of Lund.

Furthermore I would like to express my sincere thanks to all the people in the Department: Professors, technical staff, secretaries, PhD students, master thesis students and friends that made my staying in Sweden one of my most enjoyable experiences.

Finally, I want to thank my parents for all the support and love given from home.

2. Programming tools

The main tool used for this master thesis is the Matlab Graphical User Interface (GUI). The elements of the standard Matlab GUI are *controls* and *menus*. This elements were combined with the Matlab graphical objects to built each of the modules.

2.1 Objects

A Matlab object can be of three styles:

1. **Controls:** They are the push-bottoms, sliders, editable text, frames, radio buttons, pop-up menus, static text.
   Their function is that when manipulated with the mouse, they cause an action to be executed. For instance, pushing a bottom will perform an action specified by its *CallBack*.
   They are created by writing in Matlab:

   h = uicontrol(gcf,'Style','PropertyName1','PropertyValue1,...
   'CallBack','action');

   They are often known as *uicontrols*.

2. **Menus:**. Menus consist of a menu bar at the top of the active figure. Usually they are placed in the same window than the uicontrols. They can have another submenus and they let the user to choose among options for specific actions.
   They are created by writing in Matlab:

   h = uimenu(gcf,'label','MenuName',...'CallBack','action')

   They are also known as *uimenus*.

3. **Graphic Objects:** These are standard graphic objects in Matlab. They can be *axes, line, plot, patch, text*. These are the objects used to animate the pictures in the modules.

2.2 Handles

All Matlab objects have a handle. A handle is the identifier of an object. A handle contains the properties of a specific control, menu or graphic object. These properties can be, for example, color, visibility, position, interruptible, UserData, CallBack, etc. A handle can be specified by:

```
handle1=uicontrol(gcf,'Style','push','CallBack','action');
handle2=uimenu(gcf,'label','Option1','CallBack','action');
handle3=plot(NaN,NaN,'color',[0 0 0]);
```

In the last example *handle1, handle2, handle3* are the handles of a control object, a uimenu object or a graphical object.

2.3 About object properties

Object properties are the basic and most required tool for on-line simulation and animation. To create or to draw an object or a picture takes a lot of time, so in order to save time, the pictures and objects are created in a first stage and when the simulation is required, it is just necessary to change the properties. For example,

```
flow1=patch([.15 .15 .45 .45],[.2 .6 .6 .2],'g');
```

This code creates a green rectangle and it is assigned to the handle "flow1". The vectors specify the size and the position of the rectangle. If these vectors are changed then the size and the position are changed. To do this, we use the Matlab code:

```
set(flow1,'XData'[.15 .15 .45 .45],'YData',[.3 .7 .7 .3],'b');
```

This code moves the rectangle vertically 0.1 units and change to a blue color. The shape is not changed. Among the most important properties used are: *CallBack* and *UserData.*

CallBack

The CallBack property specifies the action Matlab executes when the user activates a user interface control (uicontrol). Each pushbotton, slider, menu, etc. has a CallBack property. It is defined as follows:

```
... 'Callback','action'...
```

Where the *action* can be a Matlab command, a variable name or an m-function. When a control object is activated, Matlab passes the *action* string specified in the *Callback* to the *eval* function and *executes it as if it had been typed in the command line.*

UserData

The UserData property lets the user define a matrix and associate it with an object. In this master thesis the "UserData" was mainly used to associate the handles to the main figure window, where the uicontrols and uimenus were defined. It is defined as follows:

```
... 'UserData','[handle1, handle2, ...] ,...
```

The main advantage of storing handles in a "UserData" is that the handles and variables exist as long as the figure window remains open. Another advantage is that can open aditional figure windows without confusing the variables and handles.

3. Modules, animation and on-line simulation

One of the most important parts of this master thesis is to introduce several control ideas making use of animated pictures together with on-line simulation of control processes.

3.1 Modules

The programming methodology used in all the modules is the one suggested by Math-Works Inc. which consists in programming each module using a single m-function. To do this and to carry out the actions of each m-function we made use of *Call-Backs*.

The basic structure of a module is:

```
function ctrbity(operation)
if nargin == 0,
   operation = 'init'
end;
if strcmp(operation,'init')
   % Create the user interface (sliders, pushbuttons, etc,.)
   % Create the different plots of the system.
   % Create the picture to be animated.
   % Store variables in UserData
elseif strcmp(operation,'action1')
   % Load variables from UserData
   % Execute action 1
   ⋮
elseif strcmp(operation,'actionN')
   % Load variables from UserData
   % Execute action n
elseif strcmp(operation,'calculations1')
   % Load variables from UserData
   % Carry out on-line simulation and animation
   ⋮
elseif strcmp(operation,'calculationsN')
   % Load variables from UserData
   % Carry out on-line simulation and animation
elseif strcmp(operation,'close')
   % Close windows of the graphical user interface
end;
```

In this sample structure of a module, *ctrbity* is an m-function. It can be called from other m-functions (for instance, a Main Menu). The string *operation* has the value of the current *CallBack* to be executed. If the m-function does not have any argument (i.e. the m-function is called just like *ctrbity*) then the condition "nargin==0" is meet and the string "init" is assigned to

the variable "operation". With this, the *initialization stage* is carried out. The user interface with all the *uicontrols* and *uimenus* are created. The *graphic objects* like "plots","lines","patches" are also created.

Once the graphical user interface and the pictures are created in the *initialization stage*, the module is waiting the user activates one of the *uicontrols* or *uimenus* to execute an action. The action to be executed will depend in the string associated with the *CallBack* of the *uicontrol* or *uimenu* activated.

When an action is executed, the variables and handles are loaded so they can be used or their properties can be changed. This is very useful when a *uicontrol* or *uimenu* gives an option to change the values of several parameters that are going to be used in the simulation or when some characteristics of the picture to be animated (for instance, position of the picture, axes limits, text, lines, etc,) need to be changed.

Finally, we have the stage in which the animation and the on-line simulation is done. Several *actions* can take place before this stage, which we have called *calculations.*

3.2 Animation and on-line simulation

All the figures and pictures that are going to be animated, as it was stated before, are created in the *initialization stage.* Each figure or picture is built by making a collection of graphical objects. To carry out the animations, the properties of these objects are changed in an intelligent way.

In all the modules, the state difference equations of the system are obtained and iterated in a loop. In each iteration, the properties of the graphical objects that form the figure are changed in relation to the state of the process. After several iterations, these changes will look like the figure is moving. Also, in each iteration the "data property" of the plots is increased with a new point so they will look like plotting on-line.

In some of the modules it is necessary to introduce a disturbance or to change the reference when the process is still being executed. To do that, in each loop the Matlab function *drawnow* has to be introduced. When *drawnow* is found, Matlab suspends the CallBack (i.e. go out from the loop) and check whether a uicontrol has been activated or not. If a uicontrol has been activated, then Matlab determines whether the object whose CallBack is suspended is interruptible. If it is interruptible, Matlab executes the other CallBack associated with the interrupting event and then return to the next instruction after *drawnow.*

Two modules of this master thesis make use of the keyboard. To do this, we set-on the "interruptible property" of the figure in the *initialization stage.* The "KeyPressFcn property" of the figure is activated when the user press a character in the keyboard, so when this is done, a CallBack, associated to this property , is executed. This CallBack gets the character that was pressed by using the "CurrentCharacter property" of the figure and it can be assigned to a variable to be used in the module.

It was found that to make use of the "KeyPressFcn property" the user has to click with the mouse on the corresponding figure. The user must be careful not to click in a uicontrol since the "KeyPressFcn property" will not be activated.

4. PD-control

4.1 Goal

The purpose of the PD-control module is to illustrate the concept of proportional and derivative control. Further, hidden oscillations or intersampling ripple can be studied.

4.2 Theory

One tool in process control is the PD-controller. Here the control signal is a combination of the P-part that is proportional to the error and the D-part that is the derivative of the error.

The proportional part increases the speed of response but gives a much larger transient overshoot. The derivative part increases the damping of the system and generally improves the stability of a system.

The PD-controller exhibits an anticipatory response since the derivative part improves the correction depending on the rate of change of the error.

In general the control signal can be described as:

$$u(t) = K\left[e(t) + T_d\frac{de(t)}{dt}\right] \tag{4.1}$$

Where K is the proportional gain and T_d is the constant of the derivative control and it is called the derivative time.

In this module, a double integrator process is controlled. For this process, the discrete-time algorithm for the PD-controller becomes:

$$u(k) = K\left[e(k) - T_d\dot{y}(k)\right] \tag{4.2}$$

Where $\dot{y}$ is the velocity which is also fed back. With this, since the controlled process is of second order, the PD-controller then corresponds to a state feedback controller.

In digital closed-loop control systems, sometimes, they seem to have an output response without oscillations,when in reality the output of the continuous process may have oscillations. The reason for this is because sometimes such oscillations (called *hidden oscillations*, or *intersample ripple*) cannot be seen at the sampling points. This intersample ripple is determined by the dynamics of the open-loop system since we have a closed loop system just in the sampling instants.

There are two main cases of intersample ripple. One is because observability, in the open-loop system, is lost when sampling causes cancellation of unstable modes and the sampling period matches the frequency of these modes. Just changing the sampling period will make the frequencies not to match and the oscillations will be seen at the output.

The other case of intersample ripple is caused by oscillations in the control signal due to the fact that the poorly damped zeros of the open-loop are cancelled by the controller.

4.3 Features of the Module

To illustrate in a simple way how the discrete-time closed-loop control system cannot, sometimes, detect the oscillations of the continuous process, the double integrator process was chosen for this purpose. The double integrator has the following transfer function:

$$G(s) = \frac{1}{s^2} \tag{4.3}$$

and the following sampled state representation:

$$x(kh+h) = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix} x(kh) + \begin{pmatrix} \frac{h^2}{2} \\ h \end{pmatrix} u(kh) \tag{4.4}$$

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

The closed-loop system with the PD-controller is:

$$x(kh+h) = \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix} x(kh) + \begin{pmatrix} \frac{h^2}{2} \\ h \end{pmatrix} K\left[e(k) - T_D\dot{y}(k)\right] \tag{4.5}$$

and the input-output model when the sampling period $h = 1$ is:

$$y(k) = \frac{0.5K(q+1)}{(q-1)(q-1+T_DK)+0.5K(q+1)} u_{ref}(k) \tag{4.6}$$

The module has the following features:

1) Continuous-time simulation of a double integrator process controlled by a PD-controller.
2) Discrete-time simulation of a double integrator process controlled by a PD-controller.
3) The possibility to change the values of the parameters K (gain) and T_d of the PD-controller.
4) The possibility to change the sampling period of the sampled-data controller.
5) The plots of the behavior of the continuous system (continuous line) together with the discretized system (small circles).
6) The plot of the control signal.
7) The closed loop poles and zeros of the system in the discrete plane.
8) An erasemode option, in which the user can choose to erase the plots each time a parameter of the PD-controller or the sampling period are changed or to choose to keep the plots.

4.4 Programming aspects

As it was stated before, the use of Simulink, for simple systems, was avoided. For that purpose, an m-function (dblint.m) was built. The general syntax for that function is given:

[t, tk, x, xk, u] = dblint (k,td,h)

where:

Input parameters:

k : gain for the PD-controller

td: constant parameter for the derivative part of the PD-controller

h : sampling period

Output parameters:

t : continuous time of the process.

tk: discrete-time of the process.

x : continuous state vector for the double integrator.

xk: discrete-time state vector for the double integrator.

u : control signal (the Matlab function "stairs" is used afterwards to obtain the hold of the control signal).

The function *dblint.m* simulates, both, the continuous closed-loop control system and the discrete-time closed-loop control system. Although both are simulated in discrete-time (i.e. iterating the differential equation in a loop) the continuous simulation is simulated faster than the discrete-time simulation since it is placed in a inner loop for the simulation purposes.

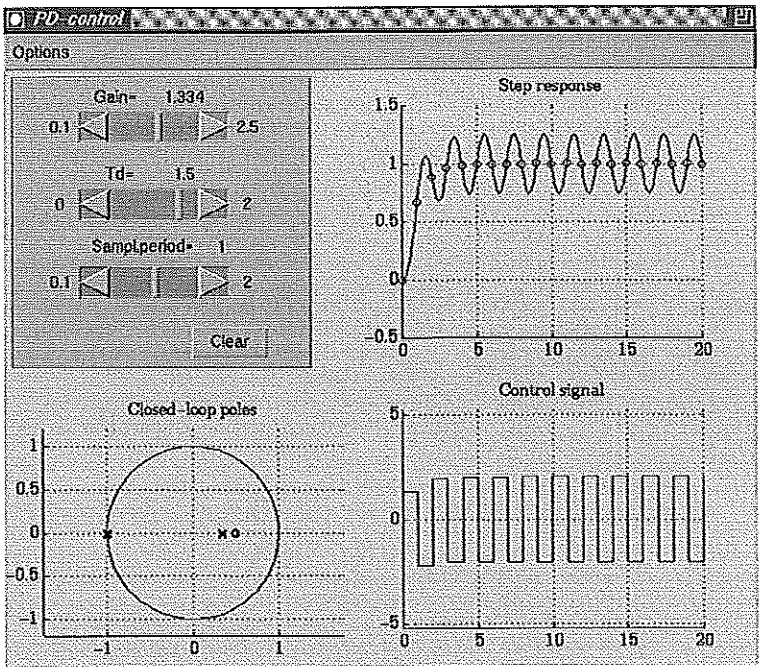

Figure 4.1 PD-control module

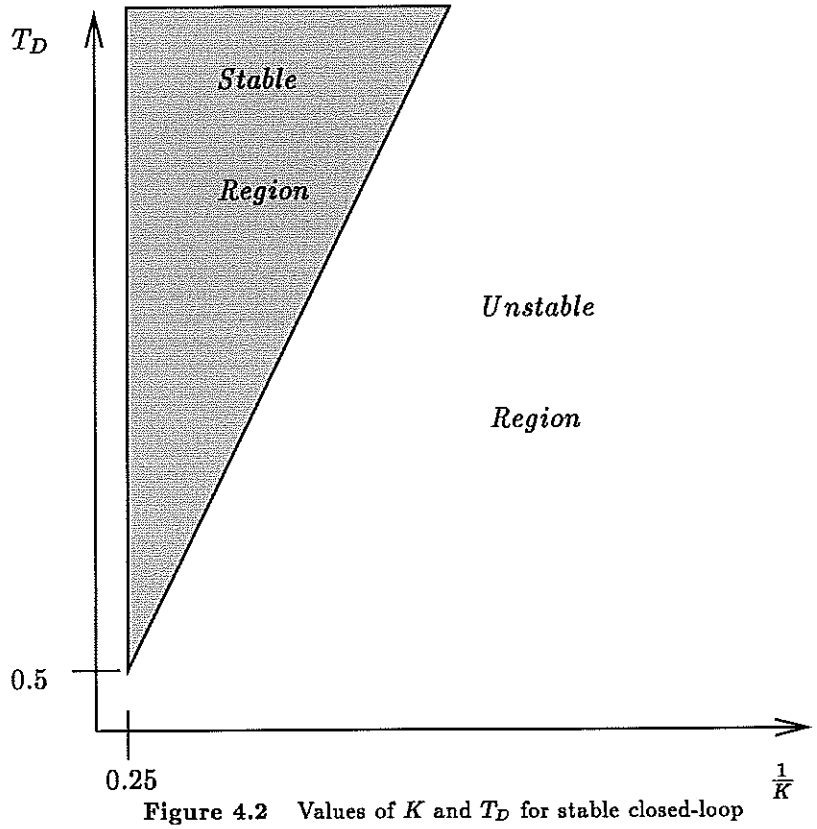

Figure 4.2 Values of K and T_D for stable closed-loop

4.5 Example

Consider the double integrator described by (4.4). The process is controlled by a PD-controller and using gain K=1.339 and Td= 1.5. The sampling period is chosen to be 1 second.

The step response plot shows the continuous response with full line and the discrete-time response with circles, representing the sampling instants. As it can be seen, the discrete-time system behaves like a first order system at the sampling instants while the continuous-time system has an oscillatory response.

This shows clearly that we can have oscillations in the system that are not seen at the sampling points. However, the oscillation can be detected in the control signal. Notice that this type of oscillation will not be detected in the output if the sampling period is changed, since the design of the controller is the same.

Further, if we consider (4.6) it is easy to see that the system is of second order and it will be stable if $K > 0$, $T_D > 0.5$, and $T_D K < 2$ (See Figure 4.2). This is seen in the module by changing the values in the slider for K and T_D. When the closed-loop poles are exactly on the unit circle, that will mean that we are on the limit for the conditions for stability and then we will have oscillations.

5. Decoupling

5.1 Goal

The purpose of the decoupling module is to illustrate:

1. The interaction between the inputs and the outputs of a multivariable *open-loop* process with and without decoupler.
2. The interaction between the inputs and the outputs of a multivariable *closed-loop* process with and without decoupler.
3. The advantage to tune controllers when the process has a decoupler included.

5.2 Theory

In many processes it is necessary to consider several control loops at the same time. This is the case for systems in which there is an interaction between the different control loops. This means that if an input is changed then several outputs may be affected. If the interaction or coupling between the control loops of the system is strong then it will be necessary to design the controllers for several loops at the same time.

There are some ways to judge if there is a strong coupling between the different parts of the process; there is also methods to pair inputs and outputs [5]. We also have methods to eliminate the coupling in a process. This last point is considered in this module.

Coupled systems

For simplicity consider a system with two inputs signals and two outputs signals (see Figure 5.1).

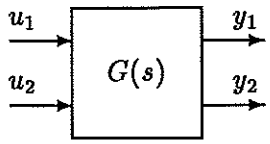

Figure 5.1 Multivariable process (Open-Loop)

The interaction between the variables of the system, can easily been seen in Figure 5.2. Then, the dynamics of the system may be represented using the Laplace transforms of the outputs and the inputs as follows:

$$\begin{pmatrix} Y_1(s) \\ Y_2(s) \end{pmatrix} = \begin{pmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{pmatrix} \begin{pmatrix} U_1(s) \\ U_2(s) \end{pmatrix} \tag{5.1}$$

Where G_{ij} is the transfer function from input j to output i. In this case Equation 5.1 shows how each input affects all the outputs. The matrix of all G_{ij} is often called *transfer function matrix* of the system.

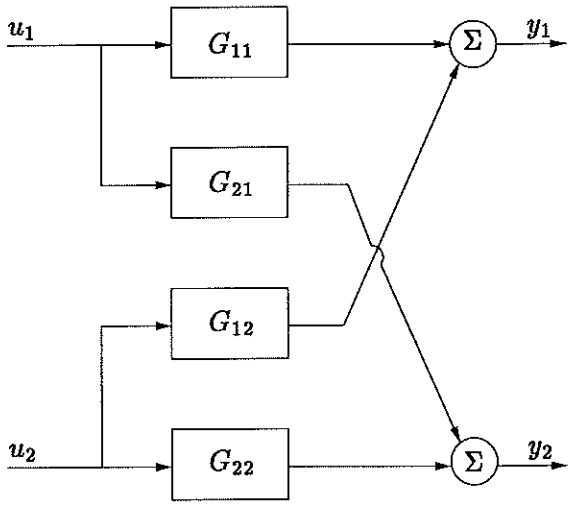

Figure 5.2 Coupled system in open-loop

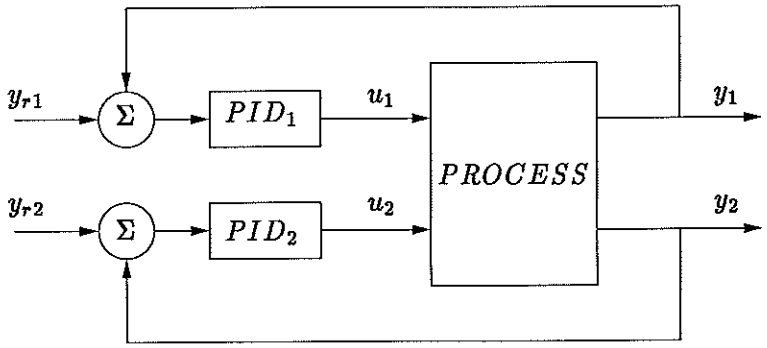

Figure 5.3 Coupled system with P I D controllers

Now if a PID controller is implemented for each output of the process, (see Figure 5.3), the tuning may not be easy since each controller affects both outputs.

Obviously, the inputs to the system will be the outputs of the control signal, which are represented by:

$$\begin{aligned} U_1(s) &= G_{c1}(Y_{r1}(s) - Y_1(s)) \qquad \text{Loop 1} \\ U_2(s) &= G_{c2}(Y_{r2}(s) - Y_2(s)) \qquad \text{Loop 2} \end{aligned} \tag{5.2}$$

In a more general sense, if the system is described as:

$$Y(s) = G_o(s)U(s) \tag{5.3}$$

where:

$Y(s)$: is the set of all the outputs.

$U(s)$: is the set of all the control signals.

$G_o(s)$: is the transfer function matrix.

and the controller signal as:

$$U(s) = G_r(s)(Y_r(s) - Y(s))$$

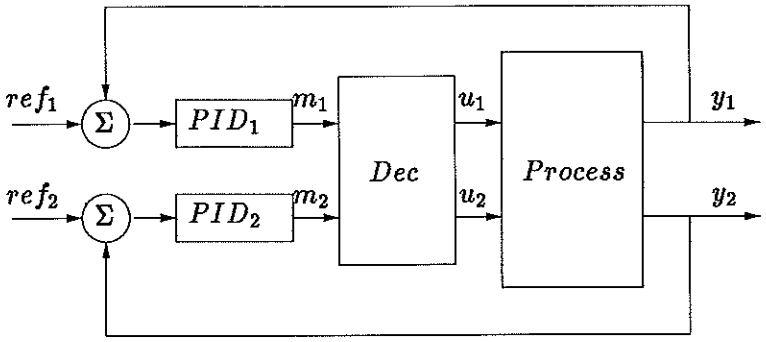

Figure 5.4 Decoupled process with P I D-controllers

Combining, the last two expressions and isolating $Y(s)$ we obtain

$$Y(s) = G_{cl}(s)Y_r(s)$$

where:

$G_{cl}(s)$: is the closed loop transfer function matrix defined by:

$$G_{cl}(s) = (I + G_o(s)G_r(s))^{-1}G_o(s)G_r(s)$$

If the characteristic equation (given by the denominator of $G_{cl}(s)$) has its roots in the left half plane then the system is stable.

However, due to the interaction, the closed loop system may be unstable even if each loop separately is stable, which makes it more difficult to tune the loops. (This can be seen in the macro if the user tries to tune the PID for the coupled system).

Decoupled systems

One way to improve the control of a coupled multivariable system is to introduce a decoupler. The decoupler is just another transfer function matrix $D(s)$ whose objective is such that the total transfer function matrix $T(s)$ has a decoupled structure (ideally a decoupled structure is a diagonal transfer function matrix).

The coupled system of (5.3) is modified when the decoupler is introduced. Then we have

$$Y(s) = G(s)D(s)M(s) = T(s)M(s) \tag{5.4}$$

where

$G(s)$= Original transfer function matrix.

$D(s)$= Decoupler transfer function matrix.

$T(s)$= Transfer function matrix of the decoupled process.

M(s)= Set of all the inputs to the decoupler (they can be the output of the controller).

The decoupling matrix is obtained from (5.4) as:

$$D(s) = G(s)^{-1}T(s) \tag{5.5}$$

This kind of decoupling is known as *dynamic decoupling* but it is possible to only look at the static part of the decoupler by choosing:

$$D(0) = G^{-1}(0)T(0)$$

This type of decoupler is known as *static decoupling.* In this master thesis only the dynamic decoupling was considered.

After introducing the decoupler, the interaction between the loops is, ideally, eliminated so the tuning is easier and the risk to get a unstable system is decreased.

5.3 Features of the Module

The multivariable process used for the simulations of this module has the following transfer function matrix

$$G(s) = \begin{pmatrix} \frac{1}{s+1} & \frac{2}{s+3} \\ \frac{1}{s+1} & \frac{1}{s+1} \end{pmatrix} \tag{5.6}$$

and the transfer function matrix of the *dynamic decoupler* according to (5.5) is

$$D(s) = \begin{pmatrix} 3 & \frac{-6(s+1)}{s+3} \\ -3 & 3 \end{pmatrix}$$

This gives:

$$T(s) = G(s)D(s) = \begin{pmatrix} \frac{3(1-s)}{(s+1)(s+3)} & 0 \\ 0 & \frac{3(1-s)}{(s+1)(s+3)} \end{pmatrix} \tag{5.7}$$

The module has the following features:

1) On-line simulation of an open-loop multivariable *coupled* system.

2) On-line simulation of an open-loop multivariable *decoupled* system.

3) On-line simulation of a *coupled* multivariable system with PID-controller.

4) On-line simulation of a *decoupled* multivariable system with PID-controller.

5) Dynamic pictures of each simulation showing the configuration of each process.

6) On-line plots for the outputs and the control signal.

7) The possibility to change the input reference of the process on-line. With this feature it is easy to notice how a change in the reference can affect the output of the process when this is coupled, decoupled or it's using a controller.

8) The ability to tune the PID-controller by changing the parameters.

5.4 Programming aspects

The simulation for the coupled process is based in (5.1). In the same way, the simulation for the decoupled process is based in (5.3). Thus, for the coupled process we have that the transfer matrix function is implemented by:

```
gnum11 =[1];
gden11 =[1 1];
gnum12 =[2];
gden12 =[1 3];
gnum21 =[1];
gden21 =[1 1];
gnum22 =[1];
gden22 =[1 1];
```

Where $gnum_{ij}$ and $gden_{ij}$ represent the numerator and denominator of G_{ij} of the transfer matrix function. To carry out the simulation, it was necessary to get the discrete-time state space representation. For that, the Matlab functions "tf2ss.m" and "c2d.m" were used. Then, we have for the state space representation:

```
[A11,B11,C11,D11] =tf2ss(gnum11,gden11);
[A12,B12,C12,D12] =tf2ss(gnum12,gden12);
[A21,B21,C21,D21] =tf2ss(gnum21,gden21);
[A22,B22,C22,D22] =tf2ss(gnum22,gden22);
```

A sampling period of T=0.1 second was chosen for the discrete-time state space representation:

```
[Phi1,Gamma1] = c2d(A11,B11,T);
[Phi2,Gamma2] = c2d(A12,B12,T);
[Phi3,Gamma3] = c2d(A21,B21,T);
[Phi4,Gamma4] = c2d(A22,B22,T);
```

Once the discrete-time state representation for each G_{ij} was obtained and according to (5.1), the difference equations were iterated in a loop in order to simulate the process.

For the decoupled process, the same was done for the matrices $G(s)$ and $D(s)$, but now the inputs to $G(s)$ are the outputs of $D(s)$.

The implementation of the PID-controller was done by using:

$$U(s) = K\left[bU_c(s) - Y(s) + \frac{1}{sT_I}e(s) - \frac{T_D s}{1+\frac{T_D s}{N}}Y(s)\right] \tag{5.8}$$

This algorithm includes a weighting b on the reference signal. If there is no weighting (i.e. $b = 1$), then an overshoot in the step response can appear. This can be avoided by setting $b \leq 1$. The algorithm includes a filter since a pure derivative should not be implemented. The filter is influenced by the parameter N, which is the derivative gain at high frequencies. N is called the maximum derivative gain and it is in the range of $3 - 20$.

The controllers were designed independently for each loop. The Matlab code is:

```
%INITIAL CONDITIONS
for k=1:tf
    ⋮
    e1=ref-y1(k)
    e2=ref2-y2(k)
    P(k)=Gain*(b2*ref-y1(k));
    I(k)=Iold+((Gain*T)/ti)*e1old;
    D(k)=((td*Dold)/(td+N*T))-...
        ((Gain*td*N)/(td+N*T))*(y1(k)-y1old);
    m1=P(k)+I(k)+D(k); % Control signal
    e1old=e1;
    e2old=e2;
    y1old=y1(k);
    Iold=I(k);
    Dold=D(k);
    ⋮
end
```

where $m1$ is the control signal for one of the controllers. The control signal for the other controller $m2$ is implemented in identical way.

About the animation

As the animation consisted in changing the pictures for each system (coupled, decoupled, coupled with PID-controller, decoupled with PID-controller), it was only required to set the properties of the uicontrols before starting the simulation. The change of input reference on line is done with the Matlab function drawnow as explained in Chapter 3.

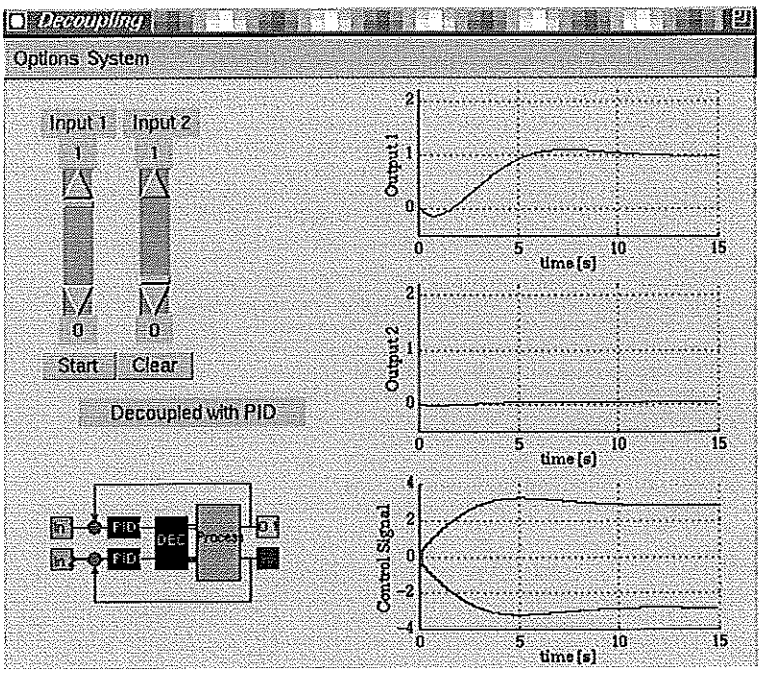

Figure 5.5 Decoupling module

5.5 Example

Consider the decoupled process of (5.6) using PI-controller (Td=0 in this case). The two controllers are tuned independently and the parameters for each are summarized in the following table:

	PI 1	PI 2
Kp	0.2	0.2
ti	0.6	0.6

If we set $u_1 = 1$ and $u_2 = 0$ and start the simulation then it is easy to see that the interaction between the control loops is eliminated. (See plots Figure 5.5) Then if the decoupler is eliminated it will be interaction between the loops. Notice that it is more difficult to tune the controller when there is no decoupler and the possibilities to obtain an unstable system are larger.

6. Controllability

6.1 Goal

The purpose of this module is to illustrate controllability using interactive dynamic pictures:

1) Concepts like controllable and uncontrollable subspaces.
2) The comparison between manual and automatic control.
3) The controllable subspace for constraint control signals

6.2 Theory

A system is *controllable* if there is always an input sequence *u(k)* that can *steer* a given *initial state*, $x(0)$, *to the origin* in a finite time.

A system is *reachable* if there is always an input sequence *u(k)* that can *steer* a given *initial state*, $x(0)$, *to any other state* in a finite time.

The set of all the states of a system, that can be reached using any possible control signal, is called the controllable subspace. To determine whether a state belongs to the controllable subspace or not it is necessary to analyze the controllability matrix.

The controllability matrix of the discrete-time state system:

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Cx(k) \end{aligned} \tag{6.1}$$

is defined as:

$$W_c = [\Gamma \quad \Phi\Gamma \quad \cdots \quad \Phi^{n-1}\Gamma] \tag{6.2}$$

The columns of the controllability matrix (6.2) span the states that can be reach from the origin. As reachability of a system is independent of the coordinates we can make a transformation to the controllability matrix (6.2) and find the controllable subspace.

There are many ways to determine if a system is controllable, one of them is to check if the rank of the controllability matrix is less than n, then the system is not controllable.

It is often necessary to find a suitable control signal that makes the system reaches certain desirable final state. To do that we proceed as follows:

$$x_1 = \Phi x_0 + \Gamma u_0$$

$$
\begin{aligned}
x_2 &= \Phi x_1 + \Gamma u_1 = \Phi^2 x_0 + \Phi\Gamma u_0 + \Gamma u_1 \\
&\vdots \\
x_n &= \Phi^n x_0 + \Phi^{n-1}\Gamma u_0 + \cdots + \Gamma u_{n-1} \\
&= A^n x_0 + \underbrace{\begin{bmatrix} \Gamma & \Phi\Gamma & \Phi^2\Gamma \cdots \Phi^{n-1}\Gamma \end{bmatrix}}_{W_c} \begin{bmatrix} u_{n-1} \\ \vdots \\ u_0 \end{bmatrix}
\end{aligned} \tag{6.3}
$$

Thus, we will reach a final desired state if $x_n = x_{desired}$ and from (6.3) the control signal required is:

$$
\begin{bmatrix} u_{n-1} \\ \vdots \\ u_0 \end{bmatrix} = W_c^{-1}\left(x_d - \Phi^n x_0\right) \tag{6.4}
$$

This method can give large control signals, and it is often that in control processes the control signal is constrained due to physical limitations. In this module we solve that problem using *bang-bang control.*

Bang-bang control

The basic idea of *bang-bang control* is to use only the maximum (one *bang*) and the minimum (other *bang*) control signals allowed for the limitations of the system. These control signals are going to be used to bring the states of the system to the desired final states by considering the phase plane trajectories of the system.

The desired final state belongs to one trajectory in the phase plane when the control signal is maximum and to another trajectory when the control signal is minimum.

Given the initial state it is possible to steer the state of a system to one of these trajectories using either the maximum or the minimum control signal. Once the trajectory is reached the value of the control signal is switched, so the behavior of the system will follow the phase plane trajectory. This trajectory is called *the switching line.*

6.3 Features of the Module

Description

To carry out the goals of this module, an example of two tanks was chosen. Each tank has a dynamics of a first order system with a given time constant for each of them.

The control objective is to reach a certain height (final state), for each tank, given an initial height (initial state) using a control signal.

The mathematical model used after being linearized (see programming aspects below for more information about the linearization) is:

$$
h(k+1) = \begin{pmatrix} -\frac{1}{\tau_1} & 0 \\ 0 & -\frac{1}{\tau_2} \end{pmatrix} h_1(k) + \begin{pmatrix} \frac{1}{\tau_1} \\ \frac{1}{\tau_2} \end{pmatrix} q_{in}(k) \tag{6.5}
$$

List of features:

1) Discrete-time simulation of a two tank process.
2) Interactive on-line animation.
3) Phase plane on-line plot for the heights of the system.
4) Time on-line plots for each height of the system.
5) Control signal on-line plot of the system.
6) The capability to change the initial condition (specified by an "x") and the final desired state (specified by an "o") by dragging the mouse. When this is done, the animation is updated.
7) The option to use automatic control or manual on-line control. For the automatic control the theory of bang-bang control was used. For the manual on-line control the user is given a slider that controls the inflow to the tanks. The user can also choose the keyboard to control the inflow.
8) For any initial state chosen the module has the capability to plot the subspace of the states that can be reached using finite or infinite energy control signal
9) The possibility to change the time constant so the animation is affected automatically.
10) The option to plot the switching line of bang-bang control for any final desired state and any time constant.

6.4 Programming aspects

About the model.

Since the purpose of the module is to make the concepts easy to understand it was necessary to work with a real system. Like most of the real processes, the two tank system has a non-linear dynamics.

The nonlinear model of the system is:

$$\dot{\mathbf{h}} = \begin{bmatrix} -\frac{a_1}{A_1}\sqrt{2gh_1} \\ -\frac{a_2}{A_2}\sqrt{2gh_2} \end{bmatrix} + \begin{bmatrix} \frac{1}{A_1} \\ \frac{1}{A_2} \end{bmatrix} q_{in} \tag{6.6}$$

Where:

a_i : is the sectional area of the outflow in the tanks.

A_i : is the sectional area of the tank.

g : is the gravity acceleration.

q_{in}: is the input flow to the tank.

h_i : is the liquid level.

As the module was supossed to simulate a first order linear system, it was neccesary to linearize the model. The linear model obtained is:

$$\dot{\mathbf{h}} = \begin{bmatrix} -\frac{a_1}{A_1}3.132 & 0 \\ 0 & -\frac{a_2}{A_2}3.132 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{A_1} \\ \frac{1}{A_2} \end{bmatrix} q_{in} \tag{6.7}$$

For this we set $g = 9.81\frac{m}{s^2}$. The linearization process was done around 0.5 (the height of the tanks is 1).

If $a_i = \frac{1}{3.132}$ then the linear system will be:

$$\dot{\mathbf{h}} = \begin{bmatrix} -\frac{1}{A_1} & 0 \\ 0 & -\frac{1}{A_2} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{A_1} \\ \frac{1}{A_2} \end{bmatrix} q_{in} \tag{6.8}$$

In this way, it is easy to see that for each tank we have a dynamics of a first order system in which A_i represents the time constant. In the module a change in the time constant will represent a change in the sectional area of the tank.

In the module, there is an option to change the time constant. This change is automatically seen in the picture.

About the on-line simulation and animation.

In order to carry out the simulation the state difference equation was obtained. All the variables to be used in the simulation, like time constants, initial and final conditions, are loaded and obtained from the UserData property of the figure. The convention used to store variable in the user data is:

...'UserData','[handles for simulation, handles for animation]',...

For the simulation and the animation the difference equations were iterated in a loop and the graphical objects were updated at the same time. (See Chapter 2).

About moving the initial and final state.

Moving an object requires actions to be executed when the mouse button is pressed down, when the pointer is moving while pressed down and when the mouse button is released. This is done by using the object properties ButtonDownFcn, WindowButtonMotionFcn and WindowButtonUpFcn.

The following is the procedure followed. First, the object to be moved has to be created in the *initialization stage* (see Chapter 3).

```
Xhandle=plot(0.25,0.25,'kx',EraseMode','Xor',...
    'ButtonDownFcn','ctrbity("move")'
```

The last Matlab code places in the (0.25,0.25) coordinates an x and the property 'EraseMode' is set to 'Xor', so when the object is moved then the last position is erased. When the pointer of the mouse is placed in the object and the left button is pressed the CallBack with the string "move" is executed.

The Matlab code for the CallBack "move" is:

```
elseif strcmp(operation,'move');
    set(figctrbity,...
```

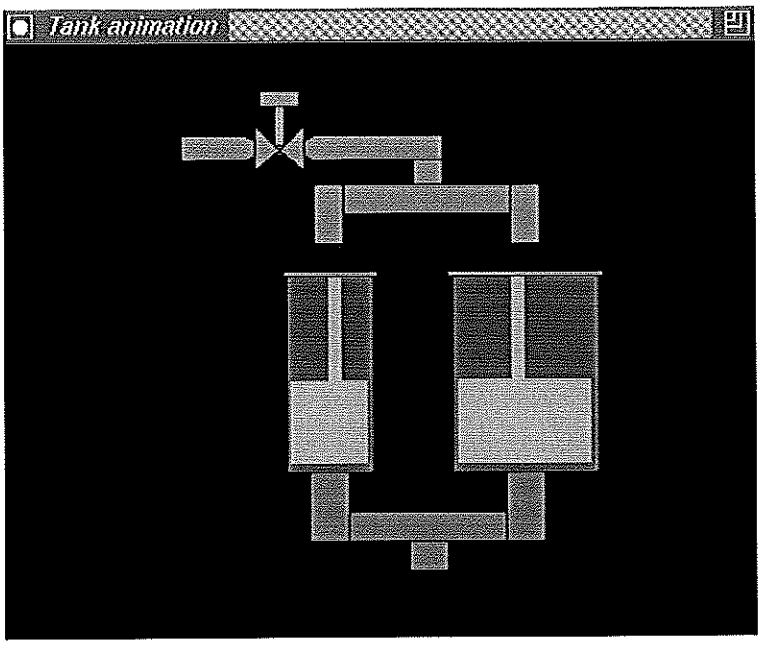

Figure 6.1 Animation

```
'WindowButtonMotionFcn','ctrbity("moving")',...
'WindowButtonUpFcn','ctrbity("moved")');
```

The last code sets two CallBacks. They are executed sequentially, not at the same time. The first to be executed is the one with the string "moving". This CallBack sets the x in the position of the mouse pointer making use of the "CurrentPoint" property. It will be executed as far as the button is pressed. When the button is released the next CallBack is executed. In this CallBack, the settings done in the CallBack with string "move" are cleared, thus the properties return to the original state.

Difficulties It was found that if two different objects to be moved are set with "EraseMode" as "xor" and if they belong to the same axes, a memory fragmentation will occur. To overcome this problem, one object was defined with "xor" erasemode property and the other one with "normal". The consequence of this is that moving one of this objects will not be as fast and as nice as the other one (this is the reason for which the user may have difficulties moving the *zero* that defines the final state.

About the manual control

The manual control of the module was done by using interruptible CallBacks. Since the simulation is done by iterating the difference equations inside a loop, a *drawnow* is placed in the loop (see Chapter 3) and the object that called the CallBack, that contains the simulation, is set as interruptible.

Thus, in each iteration Matlab checks if the user has moved the slider corresponding to the control signal, if this has been done then the new control signal will be updated and used for the next iteration.

If the user, after starting the simulation, clicks on the *green rectangle* where it is written "keyboard?" (or in any other part of the window that it is not a

uicontrol) then the keyboard will be activated. Once the keyboard is activated Matlab checks, in each iteration, if the user has moved the slider or activated a character in the keyboard (See Chapter 3).

Four keys are going to be identified. These are "j" which moves up the slider and "m" that moves down the slider about 1 %. The character "k" moves up the slider to the maximum value (one bang), and "j" moves down the slider to the minimum value (other bang).

About the automatic control: Bang-Bang control

The automatic control feature cannot be used if the system is uncontrollable. To verify this the controllability matrix is obtained and its rank is checked.

The automatic control of the system was done considering two cases: When the control signal is infinite and when it is finite. The first case was done by using (6.4) and the second case was done by using bang-bang control.

To apply bang-bang control the differential equations in (6.8) are solved. First considering the input $q_{in} = 0$ and then considering the $q_{in} = 1$. Once they are solved, they are put one state in function of the other (i.e $h_2(h_1)$) so the phase plane trajectories can be plotted. Then the equations that represent the two trajectories are:
When $q_{in} = 0$:

$$h_2(t) = h_2(0)e^{\frac{\tau_1}{\tau_2} ln \frac{h_1(t)}{h_1(0)}} \tag{6.9}$$

When $q_{in} = 1$:

$$h_2(t) = \big(h_2(0) - 1\big)e^{\frac{\tau_1}{\tau_2} ln \frac{h_1(t)-1}{h_1(0)-1}} + 1 \tag{6.10}$$

Only one section of both trajectories will conform the *switching line*. The final desired state, $h_i(t_f)$, must belong to the *switching line*. Thus by physical interpretation of the system, (6.9) plotted from 0 to $h_1(t_f)$ and (6.10) plotted from $h_1(t_f)$ to .999 will conform the *switching line*.

If the initial state lies on the *switching curve*, then we have:

$$q_{in} = \begin{cases} 1 & \text{if } h_1(t) < h_1(t_f) \\ 0 & \text{if } h_1(t) > h_1(t_f) \end{cases}$$

But, as in most cases, the initial state is not on the switching curve, then q_{in} must be chosen to move the system toward the switching curve. By inspection of the typical paths of the trajectories it is possible to determine the control law:

If $\tau_1 < \tau_2$ then:

$$q_{in} = \begin{cases} 0 \text{ if } & h_1(t) \geq h_1(t_f) \text{ and } & h_2(t) > h_2(t_f)e^{\frac{\tau_1}{\tau_2} ln \frac{h_1(t)}{h_1(t_f)}} \\ \text{or if } & h_1(t) < h_1(t_f) \text{ and } & h_2(t) > (h_2(t_f) - 1)e^{\frac{\tau_1}{\tau_2} ln(\frac{h_1(t)-1}{h_1(t_f)-1})} + 1 \\ 1 \text{ if } & h_1(t) \geq h_1(t_f) \text{ and } & h_2(t) < h_2(t_f)e^{\frac{\tau_1}{\tau_2} ln \frac{h_1(t)}{h_1(t_f)}} \\ \text{or if } & h_1(t) < h_1(t_f) \text{ and } & h_2(t) < (h_2(t_f) - 1)e^{\frac{\tau_1}{\tau_2} ln(\frac{h_1(t)-1}{h_1(t_f)-1})} + 1 \end{cases}$$

When $\tau_1 > \tau_2$ the control law is almost the same, but now the conditions are inverted. This is, if $q_{in} = 0$ then the conditions for $q_{in} = 1$ are applicable, and vice versa.

The last control law was used to apply the bang-bang control algorithm.

About detecting the final state

Problem The problem arises because the final state, sometimes, cannot be detected since it is not part of the sampling instants. One of the solutions that it could have been taken was to change the sampling period. However, the consequences of changing the sampling period are not the purpose of this module (for this, see Chapter 4). Other solution was to reduce the sampling period so the possibilities to detect the final state would be bigger. Here the problem is that the animation would have been very slow.

Solution The solution taken is based in the comparison between the final state and the current state. If the distance between them is less than a given range then the state is detected.

This range could have been taken arbitrarily, but sometimes the state is not detected because if the range is too big then a false state is detected or if the range is too small, then a big change between the states at t_1 and the states at t_2 might ignore the final state, since the final state can be between them.

For that, a dynamic range was set regarding the difference of change between the states. Part of the Matlab code is the following:

```
dx1=abs(x(1,k+1)-x(1,k))/2;
dx2=abs(x(1,k+1)-x(2,k))/2;
⋮
if (abs(x(1,k+1)-x1f)¡dx1 & abs(x(2,k+1)-x2f)¡dx2),
    % Final state detected
    ⋮
end;
```

Where "dx1" and "dx2" represent the dynamic range.

About the finite & infinite energy control subspace plots

The module includes an option to plot the subspace of the states that can be reached using finite or infinite control energy.

Uncontrollable system If the system is uncontrollable this option plots the states that, by coincidence with the dynamics of the system, can be reached.

To do this, if the finite control energy option is being used, it is just necessary to draw a *triangle* with corners in the origin, the initial conditions and (1,1). The shape of a *triangle* arises since one side of the figure will be the controllable subspace, which is a diagonal along the phase plane. The other side comes from simulating the system with the minimum input ($q_{in} = 0$) and the last side, that completes the figure of the triangle, comes from simulating the system with the maximum input ($q_{in} = 1$) (See Figure 6.3).

In case the option of "infinite gain" is being used, and if the initial conditions are in the diagonal line, which is the controllable subspace, then this same diagonal will be plotted. But if the initial conditions are not in the controllable subspace, then all the states that can be reached are in the set of all the parallels to the controllable subspace. The parallels start in the main diagonal and finish with the parallel to which the initial condition belongs to.

A figure considering all this parallels is plot using the command "patch" (See Figure 6.3).

Controllable system If the system is controllable and the finite gain control energy is being used then the bounds for the reachable subspace will be given for the plots of (6.9) and (6.10) since this equations are obtained using the maximum and the minimum input possibles, thus the dynamics of the system will not be able to go outside this limits.

If the system is using infinite gain control energy then, as it is a controllable system, the reachable subspace will be all the phase plane, therefore a "square" (with the current axis dimensions) is drawn using the command "patch".

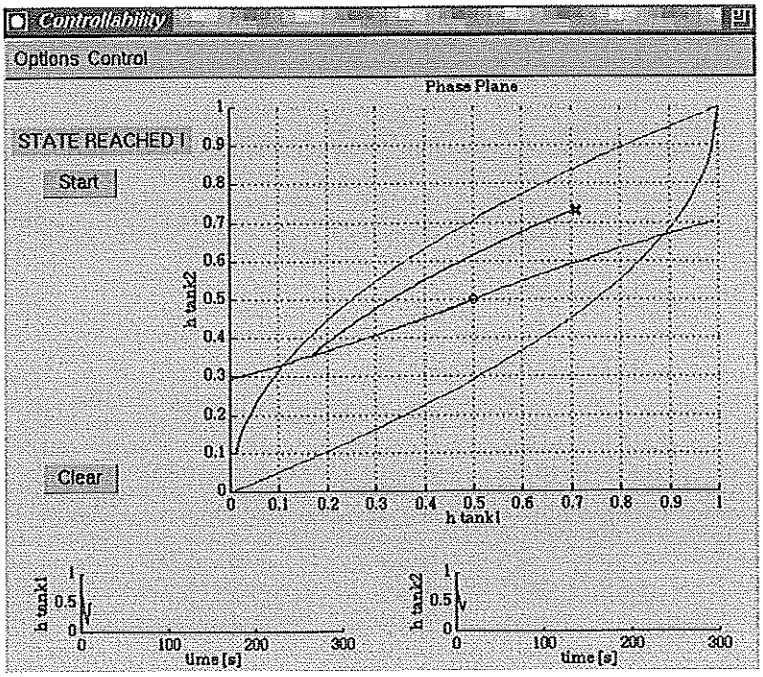

Figure 6.2 Reachable subspace, switching line and automatic control using bang-bang

6.5 Example

Example 1

Consider placing the initial conditions at $(0.71, 0.73)$ as it is shown in Figure 6.2. Set the time constants as $\tau_1 = 5$ and $\tau_2 = 10$, this will cause the system to be controllable since the controllability matrix will be of full rank. Observe the following:

1. Plot the reachable subspace using finite control energy by selecting "Control" in the menu and then "Plots". A curve with the form of an "eye" will appear. Notice that if the time constants get closer then the "eye" will tend to close. This means that the reachable subspace is smaller

since the system is approaching to be uncontrollable. When uncontrollability is reached the "eye" will be completely closed and a diagonal along the phase plane will be obtained.

If the user moves the desired final state outside this "eye", then the state will never be reached, however if the initial conditions are outside the "eye", but the desired final state is inside the "eye", it is possible to reach the final state since the "eye" represents the reachable subspace and it does not have relationship with the initial conditions.

2. Plot the *switching line* by selecting "Control" and then "Plots". A curve starting at $(0, 0.3)$ and ending at $(1, 0.7)$ is obtained (See Figure 6.2). Notice that the final state always belongs to this line. The part of the curve to the left of the final point "o" corresponds to the control input $q_{in} = 1$ and the the part to the right corresponds to $q_{in} = 0$. This may change for other time constants.

3. Use automatic control with finite control energy by selecting "Control" then "Automatic" and "Finite Gain". It is possible to see that the trajectory that starts at the initial conditions when it reaches the *switching line* the control input is switched and the dynamics of the system starts following the trajectory of the switching line. The bang-bang theory is used here. If the manual control is used, then it is easy to see that, the dynamics of the system will be always inside the "eye" since this is the only reachable subspace for finite control energy.

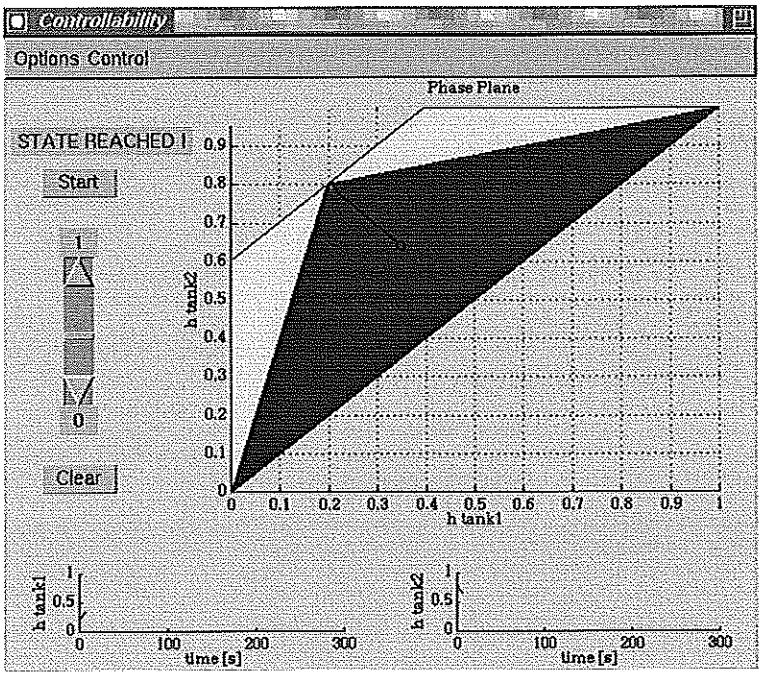

Figure 6.3 Uncontrollable system: Reachable states by coincidence with the dynamics of the system

Example 2

Consider now setting the time constants to be equal $\tau_1 = \tau_2 = 10$. This will cause the system to be uncontrollable, thus the only controllable subspace will be a diagonal along the phase plane. Now

1. Move the initial conditions to $(0.2, 0.8)$ and the final state to $(0.35, 0.65)$. Notice that this time is not possible to reach such a final state since it does not belong to the controllable subspace.

2. Use the menu option "Control", "Plots" and then "Subspace" to plot the reachable subspace using infinite and finite control gain. This time the plots represent the states that can be reached by coincidence with the dynamics of the system.

   If manual control is selected it is possible, *sometimes*, to drag the dynamics of the system to the final state. This may happen if the dynamics of the system, by coincidence, reach the final state. In the module, just push start, and this final state will be reached automatically, since the position of the slider has been set before for this purpose.

   Once the controllable subspace (the diagonal line) is reached the behavior of the system will always lie in this line.

7. Robot mechanism process

7.1 Goal

The purpose of the Robot module is to use dynamic pictures to illustrate:

1. State feedback design.
2. State feedback design using observers.

7.2 Theory

Consider the process to be controlled:

$$\frac{dx}{dt} = Ax + Bu \tag{7.1}$$

where u is the control vector, x the state vector, and A and B are constant matrices. The discrete-time space representation is:

$$x(kh + h) = \Phi x(kh) + \Gamma u(kh) \tag{7.2}$$

The control law that makes the system follow desired trajectories is given by

$$u(kh) = -Lx(kh) + l_c u_c \tag{7.3}$$

The design parameters of the system are the sampling period and the closed-loop poles in the characteristic equation. The problem is thus often stated as specifying the sampling period h, the natural frequency ω and the relative damping ζ. If all the states variables can be measured, the aim is to find a matrix L such that the matrix has prescribed eigenvalues, given by the specifications.

When the states cannot be measured, we can use the control law:

$$u(k) = -L\hat{x}(k) + l_c u_c \tag{7.4}$$

where $\hat{x}$ is obtained from the observer

$$\hat{x}(k+1) = \Phi\hat{x}(k) + \Gamma u(k) + K\left[y(k) - C\hat{x}(k)\right] \tag{7.5}$$

where 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.

7.3 Features of the Module

Description

To carry out the goals of this module a robot mechanism process was chosen. This system consists of a motor that drives a load consisting of two masses coupled with a spring constant, k. The moments of inertia are J_1 and J_2 and the damping in the spring is d. 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

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 & -1 \\ \alpha - 1 & -\beta_1 & \beta_1 \\ \alpha & \beta_2 & -\beta_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \gamma \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ \delta \\ 0 \end{bmatrix} v \tag{7.6}$$

$$y = \begin{bmatrix} 0 & 0 & \omega_0 \end{bmatrix} x \tag{7.7}$$

where:

$$\begin{aligned} \alpha &= J_1/(J_1 + J_2) \\ \beta_1 &= d/(J_1\omega_0) \\ \beta_2 &= d(J_2\omega_0) \\ \gamma &= K_I/(J_1\omega_0) \\ \delta &= 1/(J_1\omega_0) \end{aligned}$$

The states are

$$\begin{aligned} x_1 &= \rho_1 - \rho_2 \\ x_2 &= \omega_1/\omega_0 \\ x_3 &= \omega_2/\omega_0 \end{aligned}$$

The system is further described in [1] Chapter 9.

List of Features:

1. Discrete-time simulation of a robot mechanism process:
   (a) In open loop: Impulse and Step response.
   (b) With state feedback.
   (c) Output feedback (use of observers)
2. Interactive on-line animation.
3. On-line disturbances.

4. On-line plots for the output and the control signal.
5. Plot for the poles when the system is in open loop.
6. Plot for the closed loop poles of the system and of the observer (if working with Output feedback).
7. Beginners and Advance level option. In the beginners level there is no possibilities to change the parameters of the system while in the advance level the user can change the parameters.

7.4 Programming aspects

About the on-line simulation

All the variables used in the simulation, (i.e. the design parameters) are loaded and obtained from the UserData property of the figure. The convention used to store variables in the UserData is:

...'UserData','[[handles for simulation],[handles for animation]]',...

Notice that this time the handles are stored in vectors and each vector is stored in a matrix. This makes the access to each variable a bit more complex but it is a more organized way in the cases when working with several windows and figures.

In order to carry out the simulation the sampled version of (7.6) was used. The sampling of the system was done using the Matlab function: "c2d.m".

We sample (7.6) with two different sampling periods. We use a sampling period of $h = 0.1$ to simulate the continuous-time process. The other sampling period can be in the range $0.1 \leq h \leq 1$ and it is used for the implementation of the discrete-time state feedback with or without observer. Part of the main code for that is:

```
elseif strcmp(operation,'calculations');
% load handles for the simulation from the UserData
% Specify the continuous-time system
% Sample the system
   [phi,gamma]=c2d(A,B,h);
   [phic,gammac]=c2d(A,B,0.1)
% Design feedback and observer using phi and gamma
% Set Initial conditions for simulation and animation
% START SIMULATION
for k=1:(80/0.1)
   :
   t(k+1)=k*0.1;
   x(:,k+1)=phic*x(:,k)+gammac(:,1)*u(:,k)+gammac(:,2)*disturbance;
% if simulation is for Output Feedback
   if sum==h
      u(k+1)=-L*x(:,k)+ref*lc;
      sum=0;
   else
```

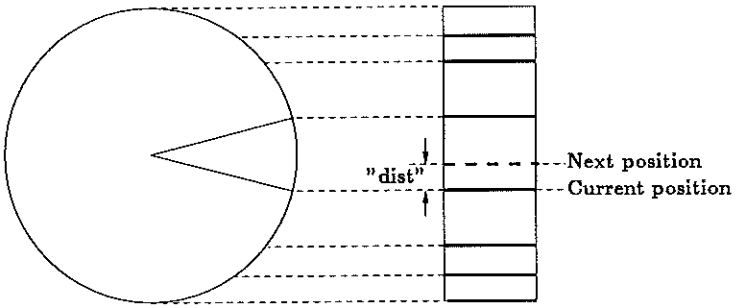

Figure 7.1 Wheel: "dist" represents the distance that a line is moved in each iteration.

```
        u(k+1)=u(k);
    end;
    sum=sum+0.1
    ⋮
    drawnow;
end
⋮
```

In the code, "phic" and "gammac" are used for the simulation of the continuous-time system, while "phi" and "gamma" are used for the design of the discrete-time feedback and observer. When $sum == h$ a new sample is taken and the control law for the observer is calculated otherwise the same control law is used for the continuous-time process.

About the Animation

For the the animation the difference equations were iterated in a loop and the graphical objects were updated in each iteration.

To give the sense of movement to the wheels, seven graphical objects with the shape of a line were drawn using "patch" (See Figure 7.1). The instruction "line" was not used since when "line" is iterated the window starts blinking making the simulation slower.

Each line drawn with "patch", is moved from its initial position to the initial position of the upper line by setting the "YData" property. Once the position of the upper line is reached, the line is set to its original initial position. The line is moved in each iteration a distance given by

$$dist = \frac{abs\big(hsin\omega(k)\big)}{Times} \tag{7.8}$$

Where $\omega(k)$ represents the angular velocity of the system and h is the time between samples of the system. $Times$ is a resolution factor. The larger $Times$ the better is the animation but the better the animation the slower the simulation. Finally, $dist$ is scaled for each line so all the lines will reach the upper line initial position at the same time.

About the impulse, step response and the disturbances

Impulse To introduce an impulse in the system we set $u(0) = 1$ as initial condition. Once, inside the simulation loop, we set $u(k) = 0$.

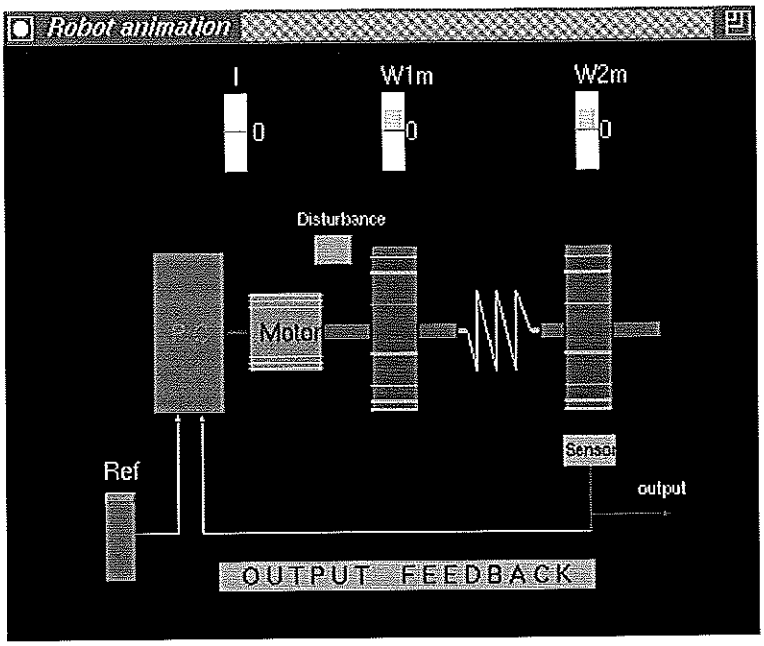

Figure 7.2 Robot Animation.

Step response and disturbance A *drawnow* is put in the iteration loop (See Chapter 2), so each time the user activates the *reference value*, using the slider in the animation window, or introduce a disturbance, using a push-bottom, the new value will be updated and used in the running simulation automatically. If a "unit step" is required then the pointer of the reference slider must be set in the top. The disturbance is set to be of height -10 and of duration of 0.1 time units.

The other parameters, like for instance h, ζ, α_1 and α_0, ω_m can only be updated when the process is not running.

About the State Feedback

Since the system is of third order, then three poles can be placed using the controller in (7.3). The parameter l_c is calculated such that the steady-state gain from u_c to y is unity. This is

$$l_c = \frac{1}{C(I - \Phi + \Gamma L)^{-1}\Gamma} \tag{7.9}$$

The desired poles can be specified by

$$(s^2 + 2\zeta_m\omega_m s + \omega_m^2)(s + \alpha_1\omega_m) \tag{7.10}$$

This characteristic equation is put in a state space representation and then discretized in order to be able to get the desired eigenvalues for the design. This is

```
[As,Bs,Cs,Ds]=tf2ss([0 0 0 1]);
[phis,gas]=c2d(As,Bs,h);
P=eig(phis);
```

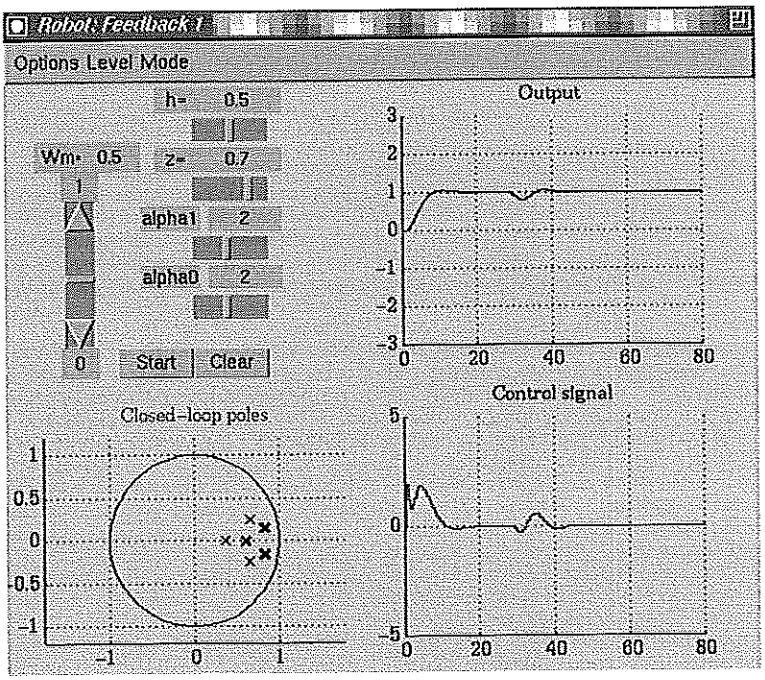

Figure 7.3 Robot mecanism module.

To obtain the gain L in (7.3) we use the Ackermann's formula using

L=acker(phi,gamma,P);

Where phi and gamma are Φ and Γ respectively. "P" is the set of the desired eigenvalues given by the specifications in (7.10).

About the Output Feedback

Here, we can only measure the output of the process. To get the other states we use (7.5). The eigenvalues of $\Phi - KC$ are chosen in the same way as the close-loop poles of the system but a factor α_0 further away from the origin. Thus the characteristic equation for the observer is

$$(s^2 + 2\zeta\alpha_0\omega_m s + (\alpha_0\omega_m)^2)(s + \alpha_0\alpha_1\omega_m) = 0 \tag{7.11}$$

This characteristic equation is put in the state representation and then discretized to obtain the eigenvalues of the design. To get the gain K we again use the Ackermmann's formula as before

K=acker(phio,gammao,P);

The term $-\alpha_0\alpha_1\omega_m$ is the continuous-time equivalence of the fastest pole of the closed-loop system when we are using the observer.

7.5 Examples

In this example it is required that the dominant modes of the robot mechanism have a natural frequency of $\omega_m = 5$ rad/s and a damping $\zeta_m = 0.7$). Set these

values by using the sliders obtained by selecting "Level" in the menu and then "Advance".

Consider the rule of thumb to get a sampling period of $h = 0.5$ for the process. Suppose that only the output is measurable, this means that we should set the system in "Output Feedback" mode. Set $\alpha_1 = 2$ and $\alpha_0 = 2$, recall that $-\alpha_0\alpha_1\omega_m$ is the continuous equivalence of the fastest pole of the closed loop system (see Figure 7.3).

The module will design the observer and the required gain to meet the specifications given. Push the button "Start" to see the simulation. It is possible to apply a disturbance to see how it affects the system with observer. Notice that the response of the system is the same than the response of the observer. The reason of this is because the system is simulated with the same initial conditions.

If we want to compare the same design without using observer, we just have to change the mode to "State Feedback" and the module will design everything again.

8. Slalom

8.1 Goal

The purpose of the Slalom module is to show the behavior of typical systems when these are being controlled manually and automatically.

8.2 Theory

The output response always depends on the dynamics of the system and on the input signal that it is controlling the process. To illustrate this, we have chosen some interesting systems like static, integrator, double integrator or non-minimum phase systems which can be represented by using their transfer function as follows: For a static system we have

$$Y(s) = KU(s) \tag{8.1}$$

for an integrator

$$Y(s) = \frac{1}{s}U(s) \tag{8.2}$$

for a double integrator

$$Y(s) = \frac{1}{s^2}U(s) \tag{8.3}$$

and for a non-minimum phase system

$$Y(s) = \frac{s-1}{s(s+1)}U(s) \tag{8.4}$$

In a static system the output will be the same as the input multiplied by a gain factor K.

The integrator is called like that since the output depends in the integral of the input. So, if a step input is given the output will be a ramp.

For a double integrator, the response depends in the double integral of the input which makes the control of such system more difficult. For instance, if a step input is given, the response will be a parabola. There are several systems that can be modelled as double integrators, for example, the attitude of a satellite or a ball and beam system.

We call a system nonminimum phase if it has zeros in the right half plane (or in the case of discrete-time systems if it has zeros outside the unit disc). The control of this systems is sometimes difficult due to the time delays they usually have.

We can reach a desired state if we give to the system a right control sequence. In this module we aim to mimic a slalom track where the purpose is to pass through the gates by using manual control.

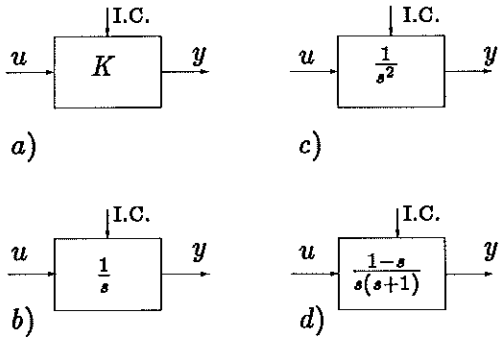

Figure 8.1 a) Static system b) Integrator system c) Double integrator system d)Non-minimum phase system.

8.3 Features of the Module

Description

To carry out the goal of this module a slalom game was chosen. The user is given a pair of skies that can be controlled manually or automatically. The objective is to make the skies pass through the gates that are distributed along a snow field. The dynamics of the skies may be represented by either a static, integrator, double integrator or non-minimum phase systems.

List of features

1. Discrete-time simulation of skier when this is represented by

   A *static* system.

   An *integrator* system.

   A *double integrator* system.

   A *non-minimum phase* system.

2. Automatic control option so the skier can follow a desired trajectory.
3. Manual control option so the user has the control of the skies. This can be done either using the slider or using the keyboard.
4. On-line plot of the input signal.
5. Animation.

8.4 Programming aspects

About the automatic control

To get the correct control sequences that make the skies pass through the gates we

1. Determine the continuous state space representation given the transfer function. This is,

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

2. Discretize the system to obtain Φ and Γ, since the aim was to do a discrete-time simulation of the skies (sampling period $h = 0.2$). This is,

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned}$$

3. Solve the state space difference equation for each system by using *z-transform* and considering as initial conditions the value of the states at the position of one gate and as $y(k)$ the position at the next gate. Thus, the *z-transforms* obtained for each system were solved several times but with different initial conditions. For this, we use

$$Y(z) = C(zI - \Phi)^{-1}[\Gamma u + zx(0)] \tag{8.5}$$

To solve (8.5) we consider a step input signal of size m, i.e.

$$u(z) = \frac{mz}{z-1} \tag{8.6}$$

Finally we determine m in (8.5) (Recall that y(k) is given by the position of the next gate). With this, the strategy to make the skies pass through the gates is to apply a step input of size m each time the skies were passing through a gate.

We use the Matlab functions "tf2ss", to obtain the state space representations, "c2d", to discretize the system and we solve by hand the *z-transform* for each system.

Static system For the static system we skip the first two steps, since the output will be the same as the input multiplied by a factor K. From (8.1) and recalling that we have chosen u to be a step of magnitude m we have

$$m = \frac{y(k)}{K}$$

Thus, the size of the step to go from one gate to another will be obtained by calculating m each time. As it was stated before, $y(k)$ is the position of the next gate.

Integrator system For the integrator system specified in (8.2), the difference equation obtained is:

$$\begin{aligned} x(k+1) &= x(k) + 0.2mu(k) \\ y(k) &= x(k) \end{aligned}$$

Solving for $y(k)$ using *z-transform* and isolating m we get:

$$m = \frac{y - x(0)}{0.2k}$$

The size of the step at each gate is m.

Double integrator system For the double integrator specified in (8.3) we obtained the following state space difference equation:

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 1 & 0 \\ 0.2 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0.2 \\ 0.02 \end{bmatrix} mu(k) \\ y(k) &= \begin{bmatrix} 0 & 1 \end{bmatrix} x(k) \end{aligned}$$

We determine $y(k)$ using *z-transform* and solving for m we get:

$$m = \frac{y(k) - 0.2kx_1(0) - x_2(0)}{0.02(k^2 - k + 1)}$$

The size of the step is m.

Non-minimum phase system For the non-minimum phase system specified in (8.4) we obtained the following state space difference equation :

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0.8187 & 0 \\ 0.1813 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0.1813 \\ 0.0187 \end{bmatrix} mu(k) \\ y(k) &= \begin{bmatrix} -1 & 1 \end{bmatrix} x(k) \end{aligned}$$

We determine $y(k)$ using (*z-transform*) and solving for m we get:

$$m = \frac{y(k) + 2x_1(0)(0.8187)^k - x_1(0) - x_2(0)}{0.2k + 2(0.8187^k) - 2}$$

About the manual control and the animation

The state space difference equations for each system (static, integrator, double integrator, nonminimum phase) were used for the simulaton. A *drawnow* was included so each time there is an iteration, Matlab checks if the slider, given to the user for the control of the skies, has been activated. If this has been activated the new value for the input control signal is updated, otherwise the simulation keeps on using the same value.

If the user, after starting the simulation, clicks on the *green rectangle* where it is written "keyboard?" then the keyboard will be activated. Once the keyboard is activated Matlab checks, in each iterarion, if the user has moved the slider or activated a character in the keyboard (See Chapter 3).

Six keys of the keyboard are identified. These are "m" which moves up the slider and "," that moves down the slider 1 %. The character "j" moves up and "k" moves down the slider 10 %. The character "u" moves up and "i" moves down the slider 20 %.

As in the last modules the picture for the animation was done in the *initialization stage* and then during the simulation the "position property" is changed according to the value of the states of the difference equation.

8.5 Examples

Example 1

Select the integrator system using the "System" option in the menu. Set the "Automatic" mode. Start the simulation. The trajectories of the skies are a set

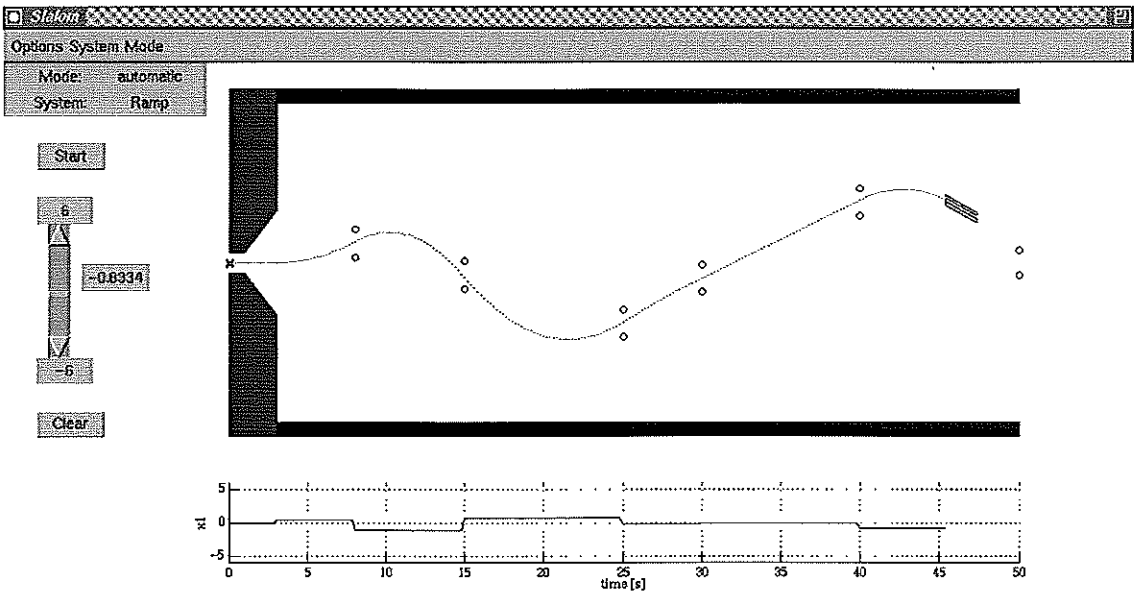

Figure 8.2 Slalom module when controlling the double integrator and using automatic mode.

of ramps. This happens since an integral system integrates the input signal, and in this case the input is a constant (step input) so the result is a ramp. If the "Manual" mode is chosen, the user is given a slider which can be used to control the skies. It is easy to see that it is more difficult to use "manual control" to make the skies pass through gates than using "automatic control". Therefore we can conclude that the analysis to find the right control sequence will be more efficient than trying to control a system whose dynamics are uncertain for a given input.

Example 2

Select the double integrator system and start the simulation. The trajectories of the skies are a set of parabolas. This is due to the input, which is a step, is integrated twice. Now select the "Manual" mode to control the skies (If you want to use the keybord remember to click in the green rectangle each time you push the start buttom). It is easily seen that it is more difficult to control the skies than an integrator. It might seem that even we give inputs to change the direction of the skies, these keep their direction for a while and then change. This is because when the value of the slider (or the variable store in the keyboard), sometimse does not change to negative inmediatelly and a positive input is still integrated twice.

9. Conclusions

In this master thesis a software that illustrates several control concepts is developed. The program creates an interactive environment where the user can change parameters and modify the models. These changes are directly visualized by using dynamic pictures.

The use of animated pictures showing theoretical concepts of control make easier the learning process, thus the program developed in this master thesis will hopefully give the students another view of the control aspects presented.

The different designs, the on-line simulation and the animations were done using Matlab and its graphical user interface. The use of Simulink was avoided so there was no need to declare variables in the Matlab workspace, with this, the variables are more protected against unwanted user action. However, the trade-off is that the better the simulation and the animated pictures the slower the program will be.

Other way widely used in this work to protect the local variables was done with the "UserData" property of the figures. This also allows the user work with other Matlab applications without confusing the variables. It would be desirable that for further development of this software the "UserData" keeps on being used.

Another suggested feature it would be to include this demo in the Internet such that it can be used with Netscape or Mosaic.

10. Bibliography

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

[2] MATLAB, *Reference Guide*, The MATHWORKS Inc. 1992.

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

[4] HAGLUND, HELENA, *Dynamic Pictures is Sampled Data Systems*, Master Thesis, Department of Automatic Control, Lund Institute of Technology, November 1995.

[5] ALLWRIGHT, JOHN, *Course notes for the course: Discrete Time Systems*, Imperial College, University of London, 1995.

[6] WITTENMARK, BJÖRN, ÅSTROM, KARL J. & BAY-JORGENSEN, STEN, *Process Control*, Lecture notes, Department of Automatic Control, Lund Institute of Technology, Sweden.

[7] BRYSON, ARTHUR & HO, YU-CHI, *Applied Optimal Control*, Ginn and Company, 1969.