ISSN 0280–5316
ISRN LUTFD2/TFRT--5590--SE

Digital Control of a Gantry Crane with Unconventional Sampling

Niklas Lagerblad

Department of Automatic Control
Lund Institute of Technology
January 1998

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

Document name
MASTER THESIS

Date of issue
January 1998

Document Number
ISRN LUTFD2/TFRT--5590--SE

Author(s)
Niklas Lagerblad

Supervisors
Karl Johan Åström, LTH and Pedro Albertos, Valencia

Sponsoring organisation

Title and subtitle
Digital Control of a Gantry Crane with Unconventional Sampling. (Digital reglering av en traverskran med okonventionell sampling).

Abstract

The purpose of this project is to control a gantry crane with conventional position measurement and a crudely quantified angle measurement. The problem formulation is presented together with design of a controller and a parameter estimator. The closed loop system has been validated by simulation. The control obtained has good performance, even with large model deviations. Algorithms for parameter estimation with crudely quantified measurements are also developed and possibilities for adaptive control are discussed.

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
76

Recipient's notes

Security classification

CONTENTS

INTRODUCTION

ABSTRACT

The purpose of this project is to control a gantry crane with conventional position measurement and a crudely quantified angle measurement. The problem formulation is presented together with design of a controller and a parameter estimator. The closed loop system has been validated by simulation. The control obtained has good performance, even with large model deviations. Algorithms for parameter estimation with crudely quantified measurements are also developed and possibilities for adaptive control are discussed.

THE BASIC PROBLEM:

The process is a gantry crane of fairly well known properties. From physical laws the crane is modeled, and the models behavior combined with measurements form the real system are used in the control.

Figure 1 A real world gantry crane in its typical surroundings

The goal is to be able to move the cart of the crane with the load hanging underneath as fast as possible to a new position minimizing oscillations. Large oscillations with high acceleration of the cargo may cause great damage both to the cargo and to the crane, and is therefore highly undesirable. This means that we compromise between speed, maximum signals to actuators and the maximum swing angle allowed to assure stability.

Two different variables can be measured. One is the position of the cart, and the other is the angle between the hanging wire and the vertical. The measurement of the

cart position is continuous, but the angle measurement is discrete. It is obtained from optical sensors that are positioned on a half-circle as illustrated in fig Figure 2.

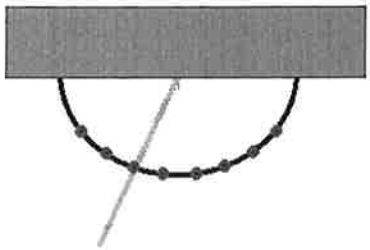


Figure 2 The halfmoon shaped disc with sensors

This implies that we get a short pulse from one of the sensors when the wire passes it, the samples may arrive at any instant of time. The sampling is therefore not periodic. This unconventional sampling leads to interesting theoretical problems. The photo-sensors are to prefer before continuous sensors because of their much cheaper price, and better reliability.

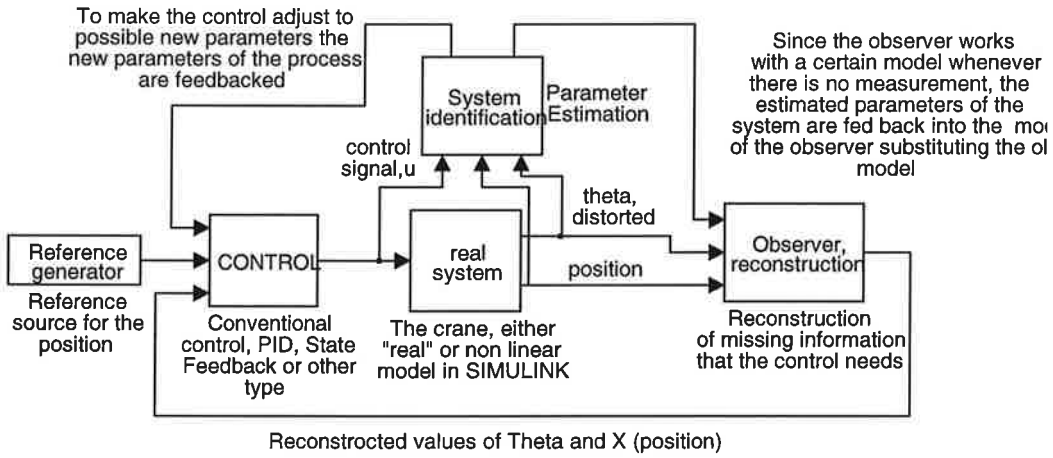


Figure 3 An overview of the control system with system identification and adaptive control

A reasonably robust control is made, so that it can handle certain model errors. The missing data is estimated by an observer for the control. Also, as in figure 3, the goal is to estimate the system parameters, and finally feed these back into the control and observer and gradually improve their function.

THE STRUCTURE OF THE PROJECT

The way I have structured the work in the project is as follows, and gives a base for the order in which the different part appear in this report.

First "attack" :

- Getting started with the functions in MATLAB
- Getting started with the functions of SIMULINK under MATLAB
- Make a functioning animation of the cart-pendulum system

Modeling the system :

- Make a working model of the nonlinear system in SIMULINK
- Examine the model and prove its correctness
- Make a functioning animation for the crane in SIMULINK
- Linearize the system around (0,0,0,0) and examine the behaviour of the model.

Control with all measurements :

- Find a good working control strategy to work with for the case of full information
- Make one control with the measurements available
- Make an observer for the two velocity states that cannot be directly measured
- Try state feedback with the new knowledge
- Make an estimator of the process parameters, and introduce this in the control, making it self adjusting

Control with the sensors instead of continuous output :

- Model the sensors in a half moon shaped disc
- Try the already existing control with the scarce measurements
- Try the existing observer policy with the scarce measurements
- Introduce model errors to see the effect upon stability
- Make a time variant Kalman filter to observe the states
- Close the control loop and test the behavior, even with big model errors

Sampling the control and reconstruction devices :

- Sample the control and examine the behaviour
- Sample the observer and possible parameter estimator

Make an adaptive implementation of the system in SIMULINK :

- Make an estimator of the process parameters with the scarse measurements
- Make a self adjusting mechanism to make the control even less sensetive to the model errors (partly acheived)

MODELING THE SYSTEM:

The first step towards a functioning controller is to create a model of the system that is to be controlled. A non linear model is derived from physical principles. It is linearized and simplified. Control design is then derived from the linear model. Finally the linear controller is simulated together with the non linear model. The idea of linearization is shown in Figure 4.

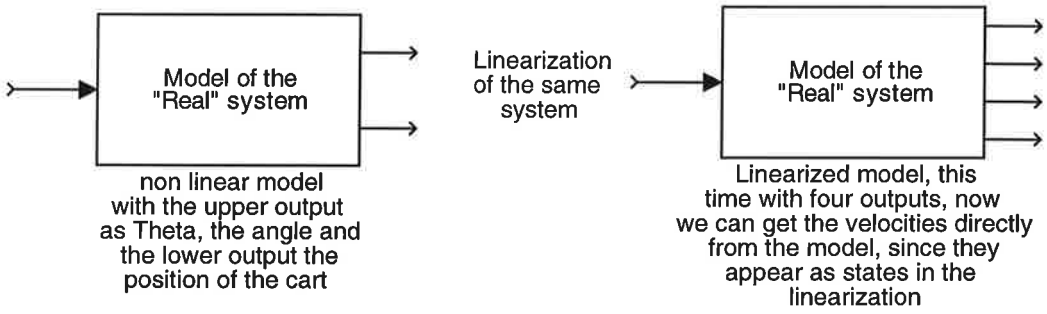


Figure 4 Illustration of the relation between systems

OVERVIEW OF PROCESS

First of all we analyze the system to examine its behavior , and find equations that describes its interesting parts. Our process is the gantry crane moving along a horizontal beam as illustrated in Figure 5. The moving part consists of a sliding trolley placed on top of the beam, a wire hanging down from this trolley and in the lower end of the wire the cargo is placed. Basically we have two parts, one is the trolley, that we move with the force F. The other is the wire/cargo that together form a pendulum system, The pendulum is only affected by the trolleys movements.

NONLINEAR MODELING

When taking a first intuitive look at the system we conclude that this is a process of the 4:th order. This makes sense when we look at the dependencies between our signals. The input signal will be F, the force that acts on the trolley. This force is

proportional to the acceleration of the cart, according to the law of acceleration of bodies (eq.1). Derived twice it yields the position of the trolley.

$$F = m \cdot a \qquad \textit{(eq. 1)}$$

The acceleration of the cart will be proportional to the applied force. There may be more terms involved also, but this relationship will have the highest order, that is 2-nd order, since the acceleration is the second derivative of the displacement of the trolley.

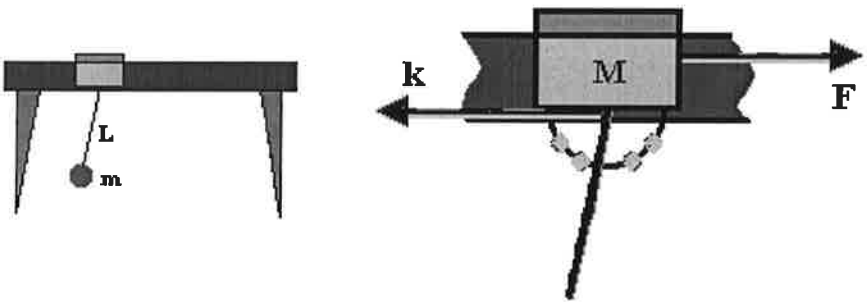


Figure 5 A simple view of the crane

Then we look at the pendulum. A pendulum that is hanging from a fixed point in the ceiling and that we release from a point ($\text{abs}\theta > 0$), will swing freely and always around its vertical. But, the force that acts inwards on the cargo, towards the vertical, is the "horizontal" component of the gravitation times the mass, which is $F = m \cdot sin(\theta)$. Thus, a constant acceleration in the opposite direction is the only thing that will give a constant displacement of the θ-angle because of the two forces canceling each other in equilibrium. So, the angle, θ, is approximately proportional to the acceleration of the cart. We obtain the angle acceleration , $\ddot{\theta}$, when we derive two times more.

We can say that to get the pendulums reaction to the force applied to the cart, we have to take into account the dynamics of the pendulum, 2:nd order. The "way to get there" for the force, is to first pass through the dynamics of another 2:nd order system. As we shall see there are more interactions between the systems, but this simple explanation gives an intuitive feeling for the relations between the different states of our process.

PHYSICAL PROPERTIES AND ASSUMPTIONS

To create a model for the system, some assumptions have been made to simplify the calculations. These makes the calculations easier, and do not impede the function of the control. However, when simplifications have been introduced in the calculations, the reasonability must be investigated in each "real-world" case before applying the

results. In this project the following assumptions have been made:

- We can ignore the mass of the wire and take into consideration only the masses of the cargo and the cart
- There is no drag present, that is, no friction caused by wind
- There is no friction in the fixation mechanism of the pendulum wire, it moves freely
- There can be only movement in one plane parallel to the crane, or the possible movement of the cart
- The actuator is linear in its actions. This means that the force is applied proportionally to the control signal within the range of control
- The friction of the wheels of the cart is linear

EQUATIONS

First of all we concentrate on the forces of cart. The forces acting horizontally are: first the force F that is applied as the actuator force, the drag that the friction coefficient multiplied with the velocity of the cart produces and finally the horizontal component of the force, T, with which the pendulum (cargo) acts upon the cart. These form the right side (eq.2) and the left side of the same equation is the mass of the cart multiplied with its acceleration.

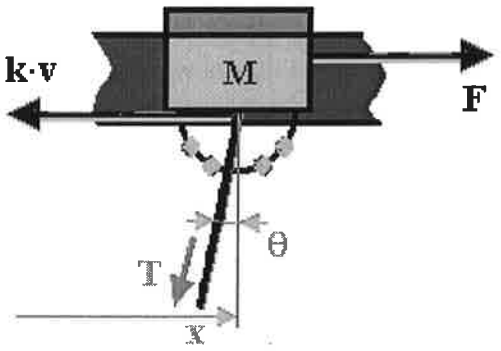


Figure 6 The forces acting upon the cart

$$M\ddot{x} = F + T\sin\theta - k\dot{x} \qquad \text{(eq. 2)}$$

Now we need to model the relations for the pendulum as well. In the other end of the wire shown above the force T is acting in the opposite direction seen from the cargo. Therefore we look at each one of the components of that force and its counterparts. From the same relation F=m*a we also get the formulas (eq. 3a) and (eq. 3b).The horizontal force component acts on the left/right acceleration of the cargo and the vertical component on the correspondent vertical acceleration, naturally.

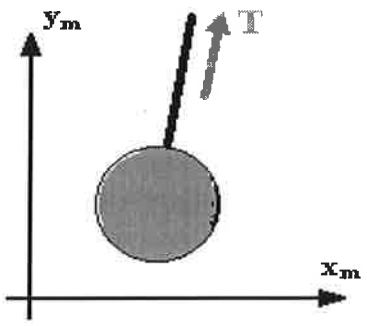


Figure 7.The acting forces of the pendulum mass

$$m\ddot{x}_m = -T\sin\theta \quad \text{(eq. 3a.)} \qquad m\ddot{y}_m = T\cos\theta - mg \quad \text{(eq. 3b)}$$

$$x_m = x + L\sin\theta \quad \text{(eq. 4a)} \qquad y_m = -L\cos\theta \quad \text{(eq. 4b)}$$

The two lower relationships (eq. 4a) and (eq. 4b) are the direct relation between the positions of the cargo and the cart and the angle, θ. We then use these equations, first using the equations (eq. 3) to replace the horizontal and vertical accelerations $\ddot{x}_m, \ddot{y}_m$ with the position of the cart, x, so that we are working with the minimal set of variables in our final equations. As described in Albertos et al (2) we obtain by implicit derivation the following 2:nd order differential equations from the equations 1 - 4.

$$\ddot{\theta} = \frac{-(F - kx)\cos\theta - (mL\sin\theta\cos\theta)\dot{\theta}^2 - g\overline{M}\sin\theta}{\overline{M}L - mL\cos^2\theta} \quad \text{(eq. 5)}$$

(4)

$$\overline{M}\ddot{x} = F - k\dot{x} - (mL\cos\theta)\ddot{\theta} + (mL\sin\theta)\dot{\theta}^2 \quad \text{(eq. 6)}$$

where $\overline{M}$ is the total mass of the system, and ignoring the mass of the wire itself this gives:

$$\overline{M} = M + m \quad \text{(eq. 7)}$$

In the following we will now see how we can use these formulas to linearize the system around a chosen operating point of x ,θ and their respective derivatives.

LINEARIZATION

To linearize the equations given above we first of all have to determine what type of system we are working with. This is a fourth order system with one input and two outputs. We therefore conclude that the state-space representation will have four states, two outputs and one single input according to the model. Further we choose the states to be such that they all have physical interpretations, that is, we let the states be the position x, the angle theta and their derivatives, respectively. Then we can from the equations (eq. 5) and (eq. 6) derive the following expressions for each one of the states.

EQUATIONS TO LINEARIZE

$$\frac{\partial x_1}{\partial t} = \frac{kx_3 \cos x_2 - ml(sinx_2 \cos x_2)x_1^{\,2} - g\overline{M}sinx_2}{\overline{M}l - ml\cos^2 x_2} - \frac{F\cos x_2}{\overline{M}l - ml\cos^2 x_2} = \dot{\theta} \qquad \text{(eq. 8a)}$$

$$\frac{\partial x_2}{\partial t} = x_1 = \dot{\theta} \qquad \text{(eq. 8b)}$$

$$\frac{\partial x_3}{\partial t} = \frac{-kx_3 - ml\dot{x}_1 \cos x_2 + mlx_1^2 sinx_2}{\overline{M}} + \frac{F}{\overline{M}} = \dot{x}_{pos} \qquad \text{(eq. 8c)}$$

$$\frac{\partial x_4}{\partial t} = x_3 = \dot{x}_{pos} \qquad \text{(eq. 8d)}$$

where : $\mathrm{x} = \{\dot{\theta}\ \theta\ \dot{x}_{pos}\ \mathrm{x}_{\mathrm{pos}}\}^{\mathrm{T}}$ and $\overline{M} = M + m$

Remark : In the third equation the double derivative of the angle theta, as $\dot{x}_1$, appears, but to save room I let the reader imagine the third equation in its full length.

CHOICE OF OPERATING-POINT

Now, we choose linearize our system to be true in and around $\mathrm{x}_{\mathrm{o}} = \{0\ \ 0\ \ 0\ \ 0\}^{\mathrm{T}}$. This choice is the most logical because it correspond to the pendulum hanging down still, without moving. The initial value of the position does not matter due to the pure integrator. However, choosing it to be 0 simplifies the linearization considerably. The velocity of the cart, we linearize around zero as well, because of the equal chances for the cart to begin moving in each direction.

JACOBIAN

One way to linearize a system is to take the partial derivatives of the system functions with respect to each state variable. The linearization simply is the making a plane in n-space with the gradients of the non linear functions in the operating point. This is the same as a Taylor expansion where the second and higher order terms are omitted. The result is a matrix that is the so called Jacobian that we use together with the working point values to form the matrix A. This will be used in our simulations as the internal behavior of the system.

$$A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \frac{\partial f_1}{\partial x_4} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \frac{\partial f_2}{\partial x_4} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \frac{\partial f_3}{\partial x_4} \\ \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} & \frac{\partial f_4}{\partial x_3} & \frac{\partial f_4}{\partial x_4} \end{bmatrix} \qquad B = \begin{bmatrix} \frac{\partial f_1}{\partial F} \\ \frac{\partial f_2}{\partial F} \\ \frac{\partial f_3}{\partial F} \\ \frac{\partial f_4}{\partial F} \end{bmatrix}$$

After a rather boring line of paperwork, these partial derivatives falls out as functions of the other state variables, we substitute our operating point into the equations:

$$A = \begin{bmatrix} 0 & -g\frac{\bar{M}}{ML} & \frac{k}{ML} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \frac{mg}{M} & -\frac{k}{M} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} -\frac{1}{ML} \\ 0 \\ \frac{1}{M} \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \text{(eq. 9)}$$

$$\frac{dx}{dt} = Ax + Bu \;;\quad y = Cx + Du \qquad \text{where u is the input, F} \qquad \text{(eq. 10)}$$

The two matrices C and D in equations 9 and 10 describes the connection between the internal dynamics of the system and the outputs. Since the position and angle are states internally represented, we measure these through the second and fourth states. The matrix C lets us configure the outputs of a system.

As long as we don't introduce disturbances in the system this simple model (eq.10) serves us well. From this linear state-space model we then derive the two transfer functions, one for each of the two outputs, since we only have one input. This gives us the transfer functions in equation 11. The pole in zero comes from the pure integrating characteristics of the x-position state, and is canceled by the double derivative zeroes of the theta-output. (This means that the cart-position state is unobservable from the theta output).

$$Y(s) = \begin{bmatrix} \theta(s) \\ X(s) \end{bmatrix} = \frac{1}{MLs^4 + kLs^3 + \bar{M}gs^2 + kgs} \begin{bmatrix} -s^2 \\ Ls^2 + g \end{bmatrix} F(s) \qquad \text{(eq. 11)}$$

SIMPLIFIED MODEL

Until now we have been working with a "full model" of the system. The model will now be simplified by neglecting the Tsinθ term of equation 2. This implies that there is no coupling back from the wire to the cart. So, taking away this term, which can be done if M » m, leaves us with the linear differential equation:

$$M\ddot{x} + k\dot{x} = F \qquad \textit{(eq. 12)}$$

which after Laplace transformation gives the transfer function

$$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + ks} \qquad \textit{(eq. 13)}$$

After this simplification, let us see if we can do the same with the pendulum part. Putting $\cos\theta = 1$; $\sin\theta = \theta$ and $\dot{\theta}^2 = 0$ and substituting $F - k\dot{x} = M\ddot{x}$ into the equation (eq.5) simplifies it considerably (eq.16).

$$\ddot{\theta} = \frac{-(F - k\dot{x})\cos\theta - (mL\sin\theta\cos\theta)\dot{\theta}^2 - g\bar{M}\sin\theta}{\bar{M}L - mL\cos^2\theta} \approx \frac{-M\ddot{x} - g\bar{M}\theta}{ML} \qquad \textit{(eq. 14)}$$

$$\ddot{\theta} + \frac{g\bar{M}}{ML}\theta = -\frac{\ddot{x}}{L} \Rightarrow \frac{\theta(s)}{X(s)} = \frac{-s^2/L}{s^2 + \frac{g\bar{M}}{LM}} = \left(\equiv \{M >> m\} \approx \frac{-s^2/L}{s^2 + g/L} \right) = \frac{1}{L}\frac{-s^2}{s^2 + \omega^2} \qquad \textit{(eq. 15)}$$

What we have achieved here is to de-couple the system and part it into smaller parts. From the earlier and more complex model a much easier handled model is derived (Figure 8 The new simplified model). This will be the model we use to build the control.

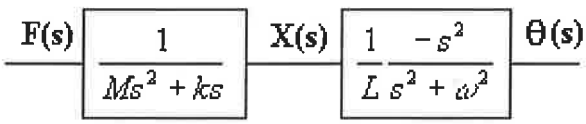


Figure 8 The new simplified model

If we compare the newly found transfer functions with the ones we had before, we find these to be closely related. The position output will not be the same of course, since we have taken away the feedback link that it had from the pendulum, but if we join the transfer functions in the figure above we get this result :

$$\frac{\theta(s)}{X(s)}\frac{X(s)}{F(s)} = \frac{1}{L}\frac{-s^2}{s^2+\omega^2}\cdot\frac{1}{Ms^2+ks} = \frac{1}{L}\frac{-s^2}{s^2+\frac{g\bar{M}}{LM}}\cdot\frac{1}{Ms^2+ks} = \frac{-s^2}{MLs^4+kLs^3+g\bar{M}s^2+\frac{kg\bar{M}}{M}s}$$

(eq. 16)

This is almost the same as in equation 11, except for the last term in the denominator. However, this will be approximately k*g, since (M+m)/M is close to 1 for reasonably small m. The new matrix A also get a slightly different shape (eq. 17):

$$A=\begin{bmatrix} 0 & -g\frac{\bar{M}}{ML} & \frac{k}{ML} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -\frac{k}{M} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad B=\begin{bmatrix} -\frac{1}{ML} \\ 0 \\ \frac{1}{M} \\ 0 \end{bmatrix} \quad C=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D=\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \textit{(eq. 17)}$$

As we see, there is only one change in the matrix above, and that is in the element $A_{3,2}$ which is the position that let the angle affect the position equations. We will use this approximation in the chapter of the control. First, however, we need to discretize the system in order to get a base to work with later on when we implement the discrete control.

ACCURACY OF LINEARIZATION AND SIMPLIFICATIONS

But, when we make and use a linear approximations of our system, and simplifications like the one above, what do we loose ? When making model simplifications the loss of accuracy has to be weighed carefully before using them.

Linearization implies that the Taylor approximations first term is taken and used instead of the non linear behavior. Since we use only this term, the error will be that of the second and higher order terms.

Generally when we simplify, the error depends on what and where we simplify. When we simplified our crane model, we concentrated on the upper part of the crane and forgot about the pendulum for a while.

Skipping the Tsinθ term in eq.2 means that we approximate it with 0. Linearizing instead replaces the term with Tθ. How much can we simplify the calculations and still have a useful model? T will depend on m, $\sin\theta \approx \theta$ and the whole term will be divided by M. Under the assumptions that M » m and the angle theta is very small, the term is small, and our new model will behave rather well.

$$\ddot{\theta} = \frac{-(F - k\dot{x})\cos\theta - (mL\sin\theta\cos\theta)\dot{\theta}^2 - g\overline{M}\sin\theta}{\overline{M}L - mL\cos^2\theta} \qquad \text{(eq. 18)}$$

Now we see from the equation above (also eq. 5) that the rest of the system offers a far more complex group of internal connections. If we examine the way the Taylor expansions affects the terms of the formula, we see that the first term in the numerator is directly depending on the acceleration of the cart (F-k*dx/dt) times a cos term. Quitting the higher order terms of the Taylor expansion, the approximation will be good only close to the linearization point. F will be affecting the linearized model more than the non-linear model since the maximum value of the cosine is 1. This means that a control made for the linear model, will not go unstable because of this. To do a really thorough stability analysis of the losses by linearization we need to apply some more elaborate mathematics and calculations, and I prefer to leave the discussion at this.

SAMPLING

Basically, when we sample a system we "discretize" it. This is equal to find the discrete model that corresponds to a continuous system that is sampled with zero order hold circuits (normal D-A and A-D converters with sample and hold function). See figure 10 for the function of sample and hold circuits.

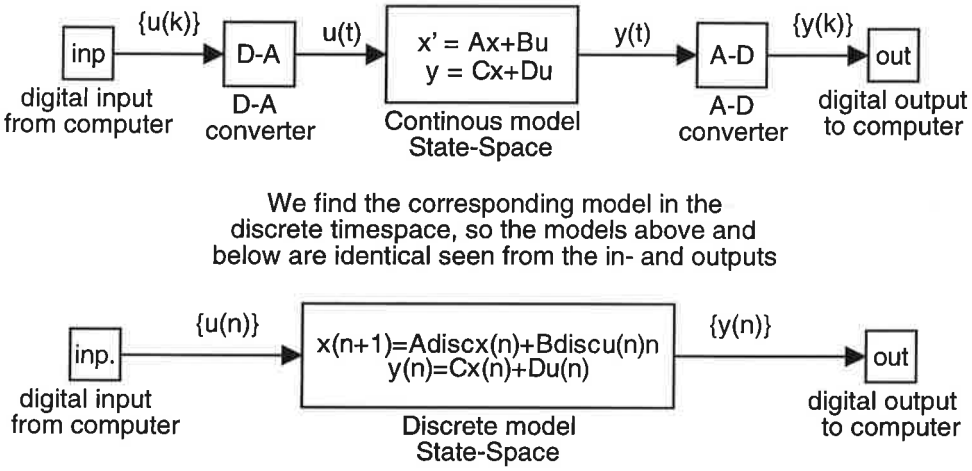


Figure 9 The relation between a discretized model and the corresponding sampled continuous system

How can we do this ? In the literature we find various ways. For the theory one may consult the textbook of Computer Controlled Systems, Åström and Wittenmark (6) chapter 3. What we want to find are the two matrices Φ and Γ that are the discrete versions of our old matrices A and B. The relation between these matrices can be expressed as

$$\begin{cases} A_{disc} = \Phi = e^{Ah} \\ B_{disc} = \Gamma = \int_0^h e^{Ah} dsB \end{cases} \qquad \textit{(eq. 19)}$$

And to find these there are various ways, see Computer Controlled Systems. Since MATLAB offers the function c2d (continuous to discrete) the easiest way is to take real values of the coefficients of the system, M, m, L and k, put them into the continuous matrix given in equations 9. I have chosen to put this discretization into a MATLAB

function, since I want to use the discrete matrices in many places in the calculations.

```
function [A,B,C,D] = makedisc(M,m,L,k,Ts)
% [A,B,C,D]=makedisc(M,m,L,k) takes the parameter values of
%
%    M = mass of cart
%    m = mass of pendulum
%    L = length of pendulum
%    k = friction coefficient of cart
%    Ts = Sample Time
%
%as arguments and produces a zero order hold discrete state space
%model of the linearized model of the cart-pendulum system.
%A,B,C,D are the matrices that represents the
%system:
%                x(n+1)=Ax(n)+Bu(n)
%                y(n)=Cx(n)+Du(n)
%
A=[0    -9.82*(M+m)/(M*L)  k/(M*L)     0;
   1         0                0         0;
   0      m*9.82/M          -k/M        0;
   0         0                1         0];
B=[-1/(M*L)  0  (1/(M+m))*(1+m/M)  0]';
C=[0 1 0 0;0 0 0 1];
D=[0 0]';
[A,B]=c2d(A,B,Ts);
end
```

The matrices C and D are the same for the two representations. This says that the states still have the same physical interpretations, position, angle and velocities. The only difference is that we will now obtain the same values in the sampling instants, and in between these we will have the last calculated value (in the last sampling instant) as output value (figure 10).

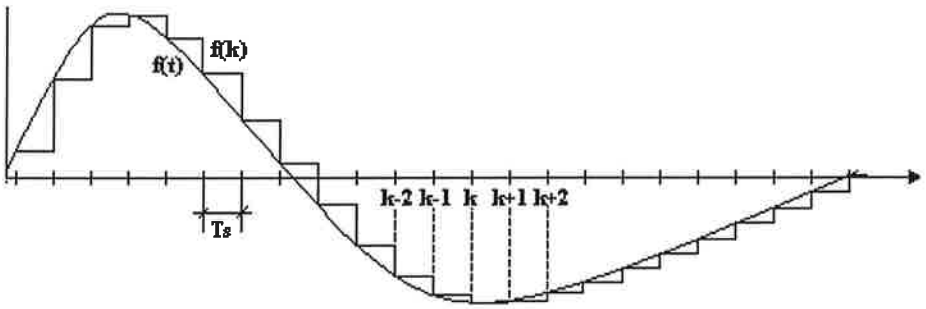


Figure 10 The principles of sampling. Comparison between the continuous signal and its sampled version

SIMULATION AND ANIMATION:

In this project simulation was a very important part of the work. To give the reader an insight into the methods used, I now want to give a small introduction to the simulation package SIMULINK. This and MATLAB have been the cornerstones of all the work. All the results that can be read in this report, figures and curves, come from simulation with these program packages. However, I presume that the reader already have some knowledge about MATLAB, so I restrict myself to a short overview of SIMULINK. For a more thorough description, see the SIMULINK-manual (1).

SHORT DESCRIPTION OF SIMULINK

SIMULINK is a program simulating dynamic systems as an extension to MATLAB. It makes use of the already existing MATLAB functions and procedures, and provides the user with a graphical user interface. The user enters the entire system in the form of a block diagram, and is therefore rather simple and intuitive to use for the first time user. It also has the advantage that the images from SIMULINK can be easily exported for presentations and paper (like this).

SIMULINK has two phases of use. First one defines the model to SIMULINK so that every syntactic detail of every partial system. Then these are connected as lines between boxes in the SIMULINK screen. One also has the possibility of creating subsystems that are transparent to the user of the top layer. This makes it possible to build and test small parts at the time, and the later connect these to test the system in its entirety.

The second phase is then the simulation and analysis of the system that has been entered. If we are dealing with more complex systems these two phases are often carried out in an iterative mode. And also, often in small pieces, as a consequence of parting the problem, and the system into subsystems, each one tested alone before connecting the whole system.

To start the SIMULINK application we first start MATLAB and from the command prompt we start SIMULINK, as seen in figure 11.

Figure 11 The MATLAB window from where we start SIMULINK

Then MATLAB opens this window, where we can first create a new file, an empty window, where we start to create our system from the block library with the built-in blocks that SIMULINK offers (Figure 12 The SIMULINK startup window.).

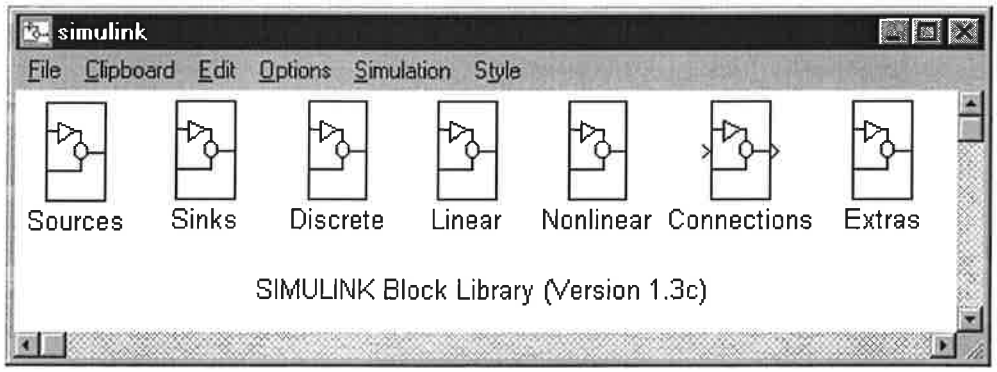


Figure 12 The SIMULINK startup window.

In the normal case we can build our system upon the multitude of already existing building blocks, and construct linear systems, non-linear systems, discrete systems etc. etc. A simple example of the typical use of the very intuitive methods that SIMULINK offers we see in figure 13. Here we are using the existing building blocks of continuous transfer function, PID regulator (which can be unmasked to see all its parts) sum, signal generator and graph drawing of the output signal.

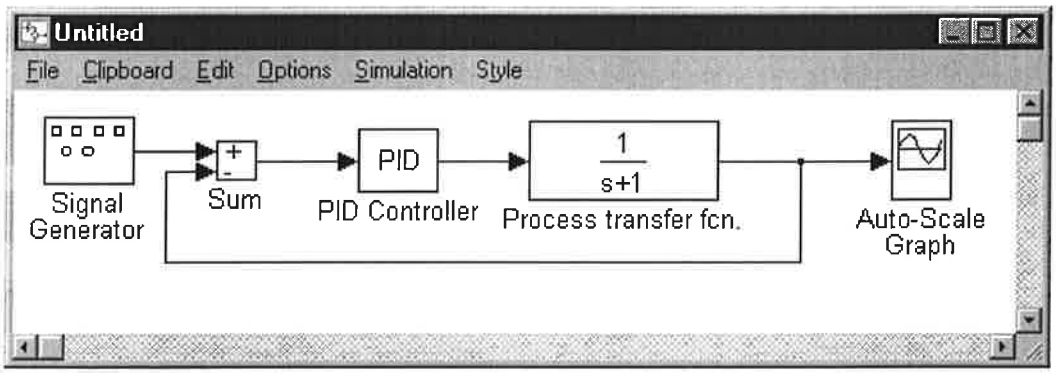


Figure 13 The SIMULINK window of a simple control system model.

Figure 14 The symbols for the MATLAB- and S-function blocks

In some occasions one finds oneself short of tools even in SIMULINK. In our specific case this happens when we want to implement the non regularities with the sampling periods and the parameter estimation. For this SIMULINK and MATLAB offers two tools. One can call two types of functions from a SIMULINK scheme, MATLAB-functions and S-functions (figure 14). These we find as building blocks under the "Nonlinear" box in the opening window.

MATLAB FUNCTION

The MATLAB-function is the usual function that we use when working with MATLAB. For instance, all the normal functions that we can use in MATLAB are MATLAB-functions, such as sin, cos, type, why, c2d etc. We can also define our own functions. For the simulation of the sensors in our model I found it to be easiest to model with a MATLAB-function. See "Modeling the sensors". Here is an example of this type of functions.

```
function [out1,out2] = myfunc(input1,input2)
% example of how a MATLAB function looks. In the function header
% all the in- and output variables are given, this is the
% "normal" programming mode that we can use. From SIMULINK we can
% call "homemade" functions as well as already existing functions
% In this case we have two inputs and two outputs. Note that it is
% important to give the number of outputs to SIMULINK in the dialog
% box of "MATLAB function"

out1=(input1+input2)/2; %out1 is the arithmetic mean value of the two inputs
out2=input1*input2;     %out2 is the product of the two inputs

end             %When we reach "end" the results are returned to the outputs
```

And to call this function from SIMULINK the only thing we do is to place a MATLAB-function symbol in our model, double click on it and fill in the dialog box as below. The file containing the function must be in the same directory or in the MATLAB-root-directory.

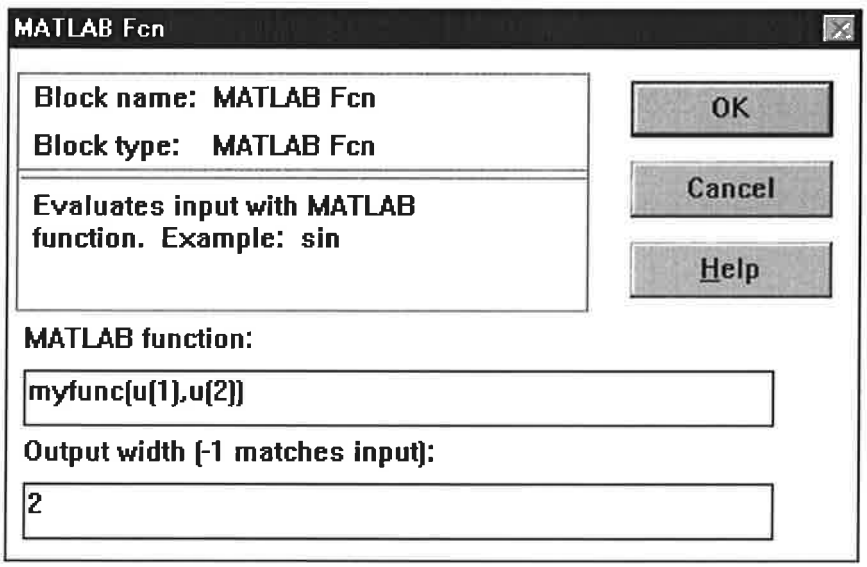


Figure 15 The dialog box of the MATLAB-function block

S-FUNCTION

The S-function is the type of functions that SIMULINK uses for all the construction blocks that already exists. It is a type of MATLAB-function that is used by SIMULINK in a more direct manner as callback function. In every iteration step, SIMULINK calls the function to ask for the different characteristics of the function. With the call, SIMULINK sends a flag that defines what it is that it wants to know. There are 7 different flags that can be sent. Below I have put the example given in the MATLAB file "dsfunc.m". Here we see how the S-function works when we use it to implement a discrete system. We have all the parts of the function that must be present for SIMULINK, and also we pass one parameter more into the structure. Since we want to be able to change the sample period from the SIMULINK model, we pass Ts, into our function.

The function is only called once with flag=0, the initialization flag, so in the case 0 we handle all the initialization code. Then in the continuation it is called with mainly flag=1 (or flag=2 for the discrete case) and flag=3. Flag=1 means that SIMULINK wants to know the new values of the states in the model. This code may be called many times during an iteration when the system is continuous. At the end of each iteration the function is called with the flag=3. SIMULINK then wants to know the system output.

```
function [sys, x0] = dsfunc(t,x,u,flag,Ts)
%DSFUNC An example M-file for defining a discrete system.
%       [SYS,X0] = DSFUNC(T,X,U,FLAG) returns depending on FLAG certain system
%       values SYS, given time point, T, current state vector, X, and
%       current input vector, U. FLAG is used to indicate the type of output
%       to be returned in SYS:
%
%       FLAG    SYS     DESCRIPTION
%       ----    ---     -----------
%       1       DX      state derivatives, dX/dT (empty matrix for discrete).
%       2       DS      discrete states, X(n+1).
%       3       Y       system outputs.
%       4       TNEXT   next time interval for update  (only discrete systems).
%       5       R       return the values of its root-functions.
%
%       To find out system characteristics the DSFUNC can be called with no
%       right hand arguments (or a FLAG value of zero). DSFUNC then returns
%       a vector of system sizes, SIZES=DSFUNC,  which
%       contains the sizes of the state vector and other parameters:
%
%               SIZES(1) number of continuous states
%               SIZES(2) number of discrete states.
%               SIZES(3) number of outputs
%               SIZES(4) number of inputs
%               SIZES(5) number of roots that the system has.
%               SIZES(6) set to 1 if the system has direct feed-through of
%                        its inputs (used for systems within systems).

%       Copyright (c) 1990-94 by The MathWorks, Inc.
%       Andrew Grace 11-12-90.

% Here is an example of how to create a set of discrete equations:
%       x(n+1) = Ax(n) + Bu(n)
%       y(n)   = Cx(n) + Du(n)



% Generate a discrete linear system:
A=[-1.3839   -0.5097 ; 1.0000         0];
B=[-2.5559         0 ; 0    4.2382];
C=[      0    2.0761 ; 0    7.7891];
D=[   -0.8141   -2.9334 ; 1.2426         0];

sample = Ts;          % Sample time
ns = 0;               % Number of continuous states
nd = length(A);       % Number of discrete states
[no,ni] = size(D);    % Number of inputs     and outputs.
```

```
% Linear Systems Description
if  abs(flag)==2                                          %calculate the new states
       sample_hit = (abs(rem(t,sample)) < sample/1e6);  %if there is a sample-hit
       if sample_hit
              xn=A*x(1:nd)+B*u;
              y=C*x(1:nd)+D*u;
              sys=[xn;y];
       else
              sys = x;
       end
elseif flag == 3                                          %tells the output this period
       dx=x;
       sys=dx(nd+1:nd+no);
elseif flag ==4,                                          %next sample
       sys=ceil(t/sample+sample/1e8)*sample;
elseif flag == 0,
       sys=[ns,nd+no,no,ni,0,1]; x0 = ones(nd+no,1);      %initialization
end
```

To use this procedure, we simply put the symbol of a S-function in our model, and fill its dialog box (figure 16) with the data needed. This is the name of the function-file and the user-defined in-parameters, in our case only the parameter Ts. The other in-parameters, t, x, u..... are always present in an S-function. For further information see the SIMULINK manuals. Below we see how the dialog box looks like in our case (Ts=0.05s).

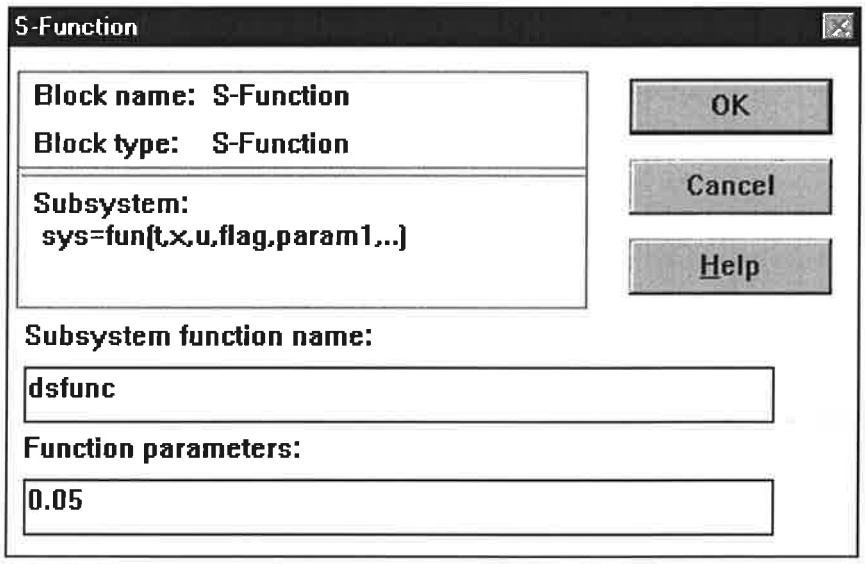


Figure 16 The S-function dialog box in SIMULINK

THE CRANE MODEL

We first model the given parts of the crane. In SIMULINK this looks like the diagram in Figure 17 What we would have in a Black Box model systems' given parts. If there were a real crane to investigate, these are the parts the user and the control would "see". The mathematical model in the "Real System" box, the sensor-block, that gives the control the discrete output of the crane, and an animation block that enables the viewer to see the system behavior.

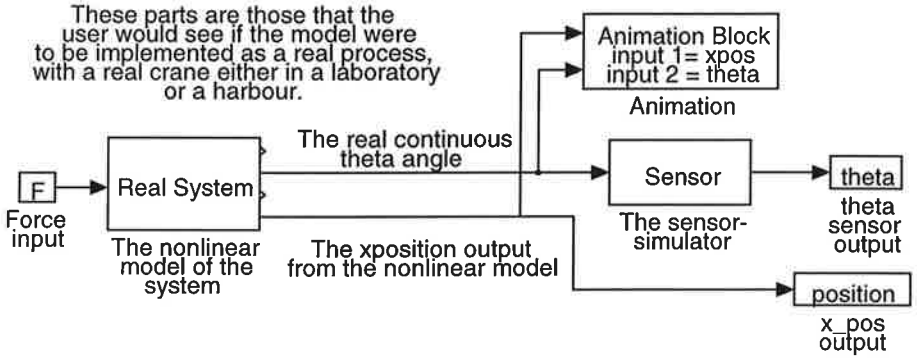


Figure 17 What we would have in a Black Box model systems' given parts

MODELING THE NONLINEAR SYSTEM

The nonlinear system in this mode was made by the already existing elements of SIMULINK using the derivation, integration and MATLAB-function elements. As we can see in figure 17 the entrance, "control signal", goes into the two multiplexers together with the feedback loops from the different "states".

The upper part, with its MATLAB-function and its multiplexer models the differential equation 5 that we found when modeling the crane. It is from this equation we get the values of the angles THETA and THETADOT. The lower part corresponds to the other differential equation 6, which deals mainly with the cart variables.

THE OUTPUTS OF THE "REAL" SYSTEM

However, in our dream world of the model above where we have access to all the state variables directly does not exactly correspond to the assumptions that we have made in our model. We cannot measure the velocities directly, nor can we measure the angle THETA continuously. The only variable that we have assumed full knowledge of at any instant is "x".

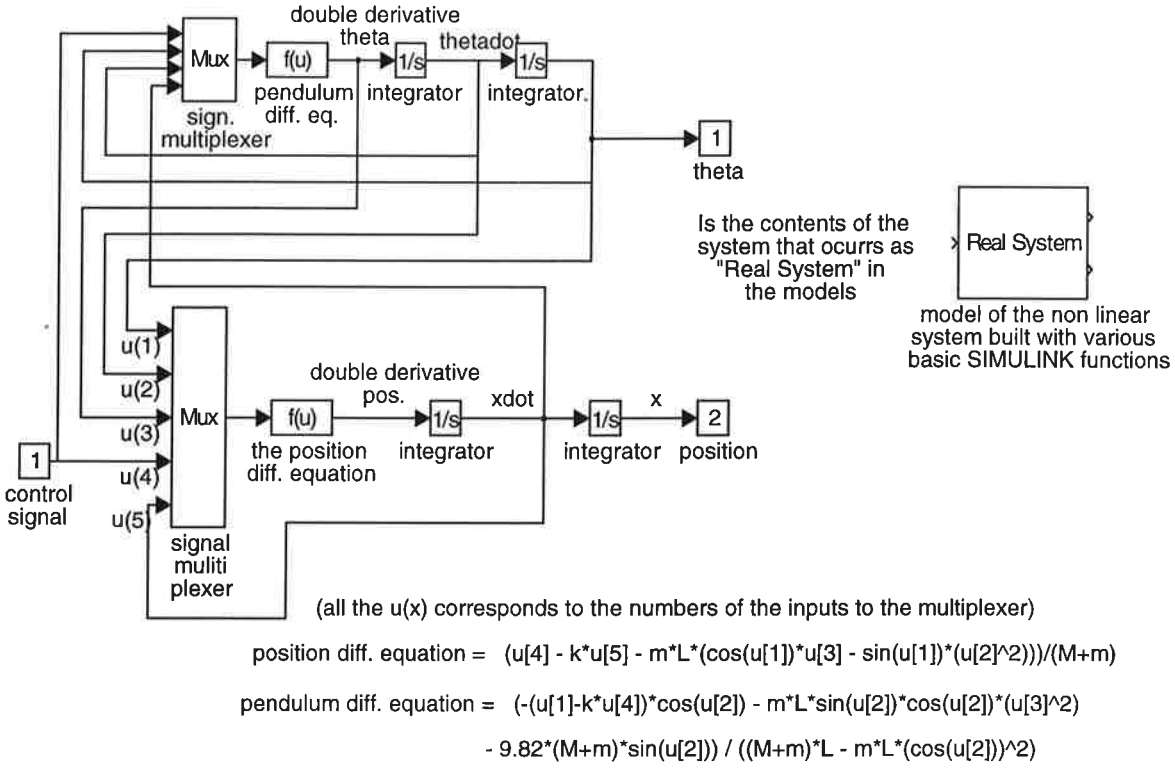


Figure 18. The internal representation of the non linear system

Modeling the angle-sensors

To model the output behavior of θ we have to do this in a separate MATLAB function. This we have to do because the assumed sample time of all the discrete parts of the system is Ts and we want to model events that occur in and during only split parts of a whole sampling period.

In our case the events that may occur are the trigging of the sensors that we have placed along the rim of a half-moon shaped piece next to the wire on the bottom side of the trolley. These may fire at any instant, when passed by the wire. Therefore we introduce the block "sensor", and in that block we introduce the MATLAB function "MakeOut3" that takes the two latest θ-outputs (these will come much more often than the sampling period, Ts) from the continuous non-linear process described above and compares them with the set of sensors that we have entered in the form of a vector.

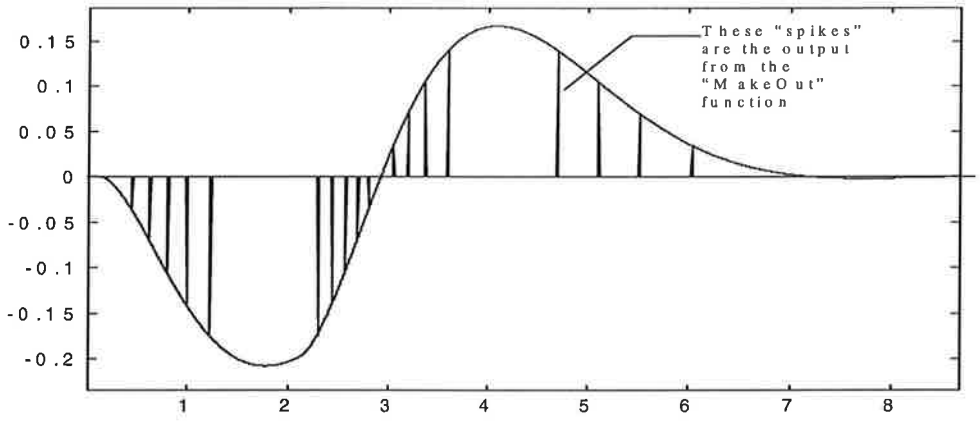


Figure 19 The function of the Sensor Block in the SIMULINK model.

For instance, {-10 -6 -2 2 6 10} means that we have placed the sensors at -10, -6.....etc. degrees from the vertical. The algorithm goes through the list of sensors, the vector, and checks to see if the wire has passed any of them. That is, it checks to see if the two successive values are on different sides of the "sensor-value".

The output of the algorithm is zero if no sensor has been passed. If one has been passed, then the output will be the corresponding value of that sensor. The two inputs thnew and thold are the values of θ, the actual value and the one that was actual 1/10 Ts ago. That is we have two inputs where the only difference is that one comes with a short time delay. In figure 19 we see how the output of the sensor block looks like. The sinus-shaped curve is the continuous angle measurement.

MAKEOUT, to simulate the Sensors from the continuous input.

```
function [out]=makeout(thnew,thold,angles);
%Makeout3 makes the calculation for
%the output Theta. Interrupted.
salange=0;
 for I=1:max(size(angles)),
    if((salange==0)&(((thnew-angles(I))*(angles(I)-thold)) >0))
     salange= angles(I);
    end
 end
 out=salange;
end
```

THE ANIMATION OF THE CRANE

To make the animation of the crane I have also used the earlier described S-functions. As we have seen these are normally used for implementing blocks of run time functions of MATLAB in SIMULINK. Here the initialization case (flag=0) is used to call a function that draws a picture. During the simulation, the call of (flag=2) another routine that erases the old picture and paints a new one is employed. When the whole simulation has been carried out we can choose to get the whole sequence played again in a much faster pace. This requires that there are variables created in the MATLAB workspace that contains the necessary data, time and output values.

Figure 20 The typical view of the Animation Block

CONTROL

Now we have a model of the crane. Next step is to construct a good control according to the demand of small swing-angle and fast operation.

First we assume full access to both outputs as continuous signals. For the control there are different possibilities. Any type of control that leaves us with asymptothic stability would work because sooner or later we would reach the reference positions. However, we do want the crane to be fast, and only swing moderately.

Once established a control, the case with the missing or scarce data on the outputs is handled. Then we continue with adding the difficulties one after another as we go for the final goal, to have the tools for an adaptive control handling the crane with the scarce measurements of the swing angle.

The steps to reach this will be the following :

- Linear Continuous Control with access to both output signal continuously.
- Linear Continuous Control with the scarce measurements of theta
- Discretization of the Linear Control w/ scarce measurement
- Estimating parameters of the system.
- Updating the dynamics of the observers and control by feedback of parameters.
- Investigating the possibility of reference signal shaping for optimization

LINEAR CONTROL WITH ALL DATA KNOWN

The basic control problem has been solved with good performance of the crane by a Korean team Lee, Cho and Cho (4), by employing a similar model to the simplified model presented in the end of the modeling chapter, and a sort of double loop PID control. Their control bases itself upon the continuous availability of theta and x, and also on an already existing velocity servo of which they have determined the characteristics, and that they later in the paper use to form their control. I have decided to follow their guidelines and build my control from their findings. For the basic theory the reader may consult either Åström and Wittenmark (6) or D'Azzo and Houpis (7).

POSITION CONTROL

We part the design into two parts, first of all we make a position control. Later this will be a "black box" for the second part of the system, the anti swing control. Figure 21 shows how the inner control loop looks.

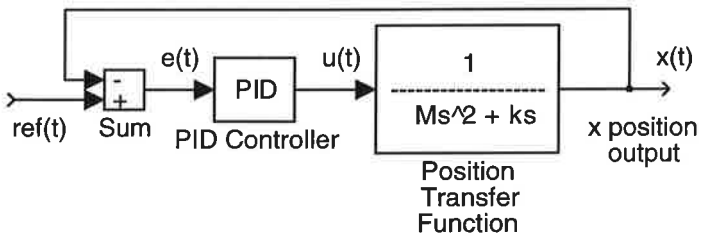


Figure 21 The position control loop structure

I use the simplified model that separates the crane into two systems. We use the modeled behavior of the upper part of the system as described before to design the position control. We have the cart with the mass M, the applied force F and finally the friction coefficient k.

$$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + ks} \qquad \text{(eq. 20)}$$

Together with a normal PID control with the control law :

$$G_{PID}(s) = (P + \frac{I}{s} + Ds) \qquad \text{(eq. 21)}$$

The transfer function for the cart (m<<M) becomes :

$$\frac{X_{ref}(s)}{X(s)} = \frac{\frac{P + I/s + Ds}{Ms^2 + ks}}{1 + \frac{(P + I/s + Ds)}{Ms^2 + ks}} = \frac{Ds^2 + Ps + I}{Ms^3 + (k + D)s^2 + Ps + I} \qquad \text{(eq. 22)}$$

As we can see here, with this control (eq.21) and a second order system (eq.20), we can place poles exactly the way we want since the terms in the denominator contain enough values of P,I or D. In our case the control can be considered as a pole placement problem (eq.23).

The pure integrating part of the models denominator (eq.20) comes from the physical fact that the cart stays in the place where it stopped. If we push in one direction it will never come back, unless pushing equally in the other direction that is. The other pole situated in -k/M is always very slow and proceeds from the inertia of the moving cart. As we see, with reasonable masses of more than one or two kilograms the pole will be placed in $-0.5 < -k/M < 0$.

In the design of the control we want to get good tracking of the reference signal in the low frequency region , and that in the higher frequency regions we oppress possible modeling errors and sensor noise. Choosing the parameters to be P=930, I=450, D=60 when we have the parameters M=2 and k=.5 the transfer function becomes :

$$\frac{X_{ref}(s)}{X(s)} = \frac{60s^2 + 930s + 450}{2s^3 + 60{,}5s^2 + 930s + 450} = \frac{30(s + 0.5)(s + 15)}{(s + 0.499)(s^2 + 29.75s + 450.13)} \qquad \text{(eq. 23)}$$

Which gives us poles in -15+15i , -15-15i and -0.499 and zeros in -15 and -0.5. This follows more or less the pole placements of the Korean team and I decided to go with these numbers. As we can see in the bode plot below, we obtain good suppression of eventual errors in the high frequency region, and very nice model-following in the low frequency region, and thus we achieve the goals that we put on the control.

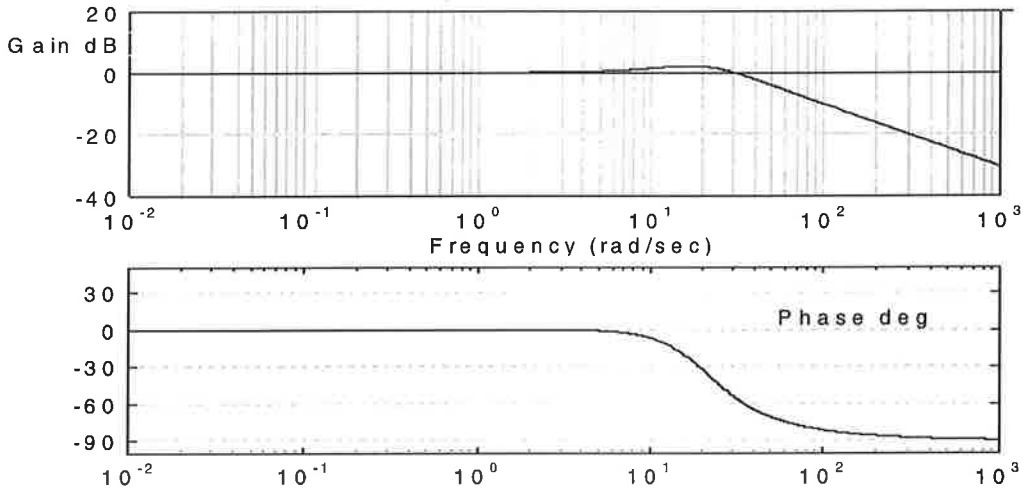


Figure 22 Bode diagrams for closed loop of the position control

ANTI SWING CONTROL

Now, the next step is to make the control for the pendulum. We take our functioning position control system and put it into an other PID control loop. We consider both the closed loop position control system and the transfer function of the pendulum part of the system, and try to find a functioning control for this part as well. When we look at the model in figure 23 we may find it a bit odd that our control is situated on the feedback loop of the pendulum loop control. Why?

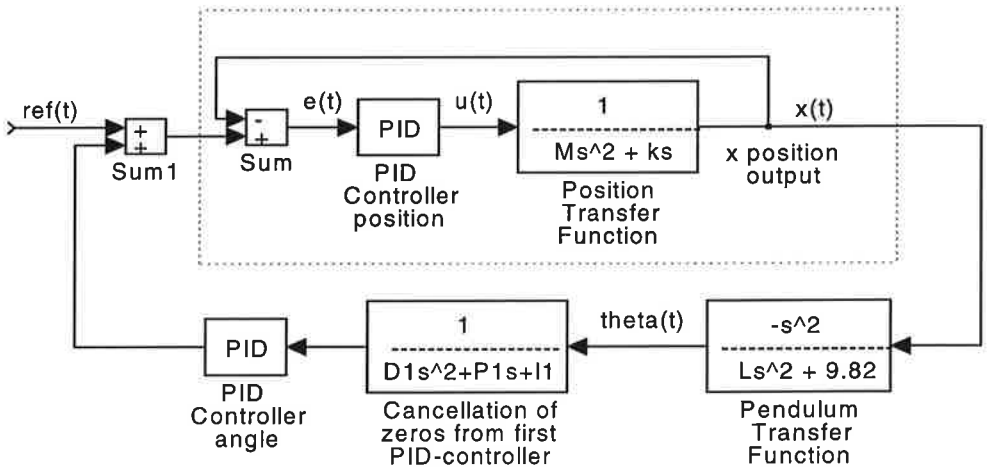


Figure 23 The SIMULINK mode with the anti-swing control addition.

We have to think a bit about what objectives that we have with our control. First of all we want to affect the position of the cart, this objective then competes with the demand for a small swing angle. Or rather, the acceleration of the cart will directly relate

to the swing angle.

As we can se in the figure the two inputs to the adder "Sum1" are just these goals for the position control. This is where we have a conflict. Both we and the anti-swing control put up rules for the position control to follow. Sometimes we contradict each other with loss of efficiency as result.

Would the goal-angle have been other than zero, this would be done with another adder in between the theta-output of the crane and the control. Just like we use the error of the position (y(t)-ref(t)) to control the position, we would control the swing-angle. The theta angle output is the error in our case. The transfer function from position to swing angle is :

$$\frac{\theta(s)}{X(s)} = \frac{1}{L} \cdot \frac{-s^2}{s^2 + \frac{g}{L}} \qquad \textit{(eq. 24)}$$

and therefore, together with eq. 23 we get the open loop transfer function from the position reference to the theta output. Now we take this transfer function and combine it with a PID control and decide the value of our proportional feedback of the error, given the relation between the P and D,I terms in the controller. We decide the value with the root locus method since MATLAB offers excellent tools for this.

$$\frac{\theta(s)}{X_{ref}(s)} = \frac{X(s)}{X_{ref}(s)} \cdot \frac{\theta(s)}{X(s)} = \frac{1}{L} \cdot \frac{-s^2}{s^2 + \frac{g}{L}} \cdot \frac{30(s+0.5)(s+15)}{(s+0.499)(s^2+29.75s+450.13)} =$$

$$= \frac{-30s^2(s+0.5)(s+15)}{L(s^2+\omega^2)(s+0.499)(s^2+29.75s+450.1)} \qquad \text{(eq. 25)}$$

When we design the control, we introduce one part that simply cancel the zeros of the transfer function above that comes from the inner loop PID control. Then we introduce one PID controller more in the chain. The complete open loop transfer function becomes eq.26.

What we do is to introduce one integrator in the complete open loop, cancel the zeros that came from the control of the inner loop, and replace these with one or two new ones. I have tried with various variants of placement of the zeros, to get various results. We could leave the control with just one integrator and a constant gain, leaving the zeros in -15 and -.5, but these will change with the characteristics of the position transfer function if we introduce an adaptive control in the inner loop. This dependency I want to avoid.

$$G_{\substack{\theta-open\\loop}}(s) = \frac{\theta(s)}{X_{ref}(s)} \cdot G_{\theta-control} = \frac{\theta(s)}{X_{ref}(s)} \cdot \frac{D_2 s^2 + P_2 s + I_2}{s} =$$

$$= \frac{-30s^2(s+0.5)(s+15)}{L(s^2+\omega^2)(s+0.499)(s^2+29.75s+450.1)} \cdot \frac{D_2 s^2 + P_2 s + I_2}{s(s+0.5)(s+15)} = \qquad \text{(eq. 26)}$$

$$= \frac{-30s^2}{L(s^2+\omega^2)(s+0.499)(s^2+29.75s+450.1)} \cdot \frac{D_2(s^2 + \frac{P_2}{D_2}s + \frac{I_2}{D_2})}{s}$$

We can also place only one zero, if we try with a zero in -20 together with the results from a root locus diagram to determine the derivative constant, D, we see that we obtain good results. This means that we leave the integrator term in the PID controller to 0 and use a pure PD-controller.

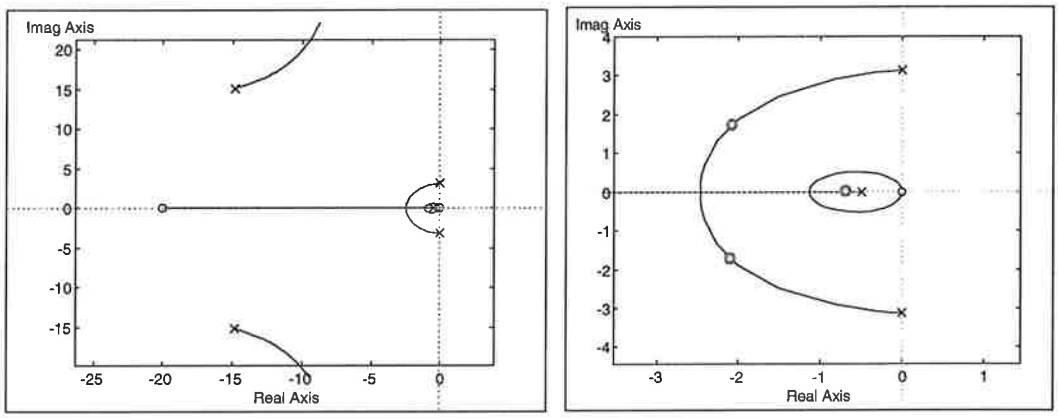


Figure 24 The results from the root locusprocedure. The diagram to the right is a zoom-in of the center of the image to the left.

In the figure to the right (figure 24 b) I have marked the place where we get a relative damping 0.7 in the root locus diagram (D > 0). This occurs with the D=3.1 for L=1m. The parameter L affects the function in two ways. First we can say that it is a constant amplification of 1/L, the optimal value of k varies with L directly. L also affects the behavior in the sense that with a smaller L the half circle in the root locus diagram to the right will become smaller, and so the optimal value of D. The model never will be unstable, for any value of D. However, very large values model errors might cause instability. The new and final open loop transfer function will have the following look :

$$G_{\substack{\theta-open\\loop}}(s) = \frac{-30s^2(s+0.5)(s+15)}{L(s^2+\omega^2)(s+0.499)(s^2+29.75s+450.1)} \cdot \frac{D_2(s+\frac{P_2}{D_2})}{(s+0.5)(s+15)} = \quad \text{(eq. 27)}$$

$$= \frac{-30s^2 \cdot 3.1 \cdot (s+20)}{L(s^2+\omega^2)(s+0.499)(s^2+29.75s+450.1)}$$

And with this result, the total transfer functions will be examined for stability from input to each one of the two outputs. We derive these as follows in equations 28-30:

$$\frac{X(s)}{X_{ref}(s)} = \frac{30(s^2+9.82)(s+0.5)(s+15)}{(s^2+9.82/1)(s+0.499)(s^2+29.75s+450.1)+30s^2 \cdot 3.1 \cdot (s+20)} = \quad \text{(eq. 28)}$$

$$= \frac{30(s^2+9.82)(s+0.5)(s+15)}{(s+0.6965)(s^2+25.36s+433.71)(s^2+4.1857s+7.3145)}$$

$$\frac{\theta(s)}{X_{ref}(s)} = \frac{-30s^2(s+0.5)(s+15)}{(s+0.6965)(s^2+25.36s+433.71)(s^2+4.1857s+7.3145)} \quad \text{(eq.29)}$$

$$G_{\substack{smoothing\\filter}}(s) = \frac{X_{ref}(s)}{X_{\substack{ref\\step}}(s)} = \frac{0.5}{s+0.5} \quad \text{(eq. 30)}$$

I have in my design used a filter to smoothen the input signal. The resulting block diagram for the finished control then looks like the finished control in figure 26 . I have chosen to use the slightly modified PID-controller that we see in equation 31. With this control we avoid deriving the reference signal. This helps us getting a smother control signal. The new control law does not affect stability of our system, but it helps using this form of the derivative part of the PID control when we later want to discretize the system. The discrete approximation of a derivative could give us problems otherwise. The PID now gives us the control :

$$U(s) = (P + \frac{I}{s})(X_{ref}(s) - X(s)) - DsX(s) \quad \text{(eq. 31)}$$

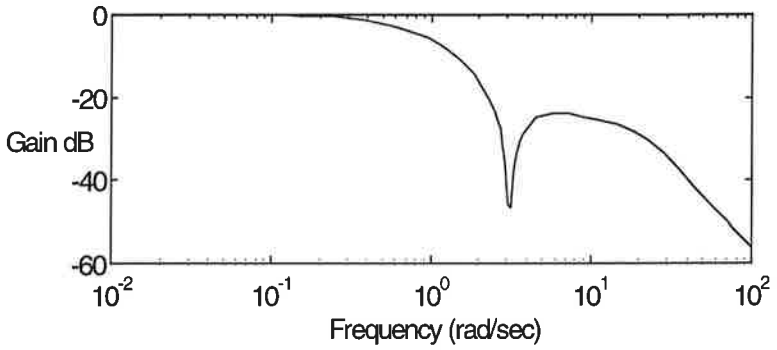


Figure 25 The Bode plot of the transfer function X(s)/Xref(s) (with the filter)

The Bode plot from position reference (unfiltered) Xref(s) to actual position X(s) is shown in figure 25. As we can see in the plot, we get an excellent command tracking in the low frequency region, and sufficient suppression of the high frequency regions. This assures a smooth control that helps the anti swing loop in its work. I have chosen to use a ramp-step signal as reference. This equals giving the crane a reference velocity during its displacement trajectory. The position follows the ramp part, that we choose to have an inclination of 1 m/s which has to be considered a rather high velocity for crane.

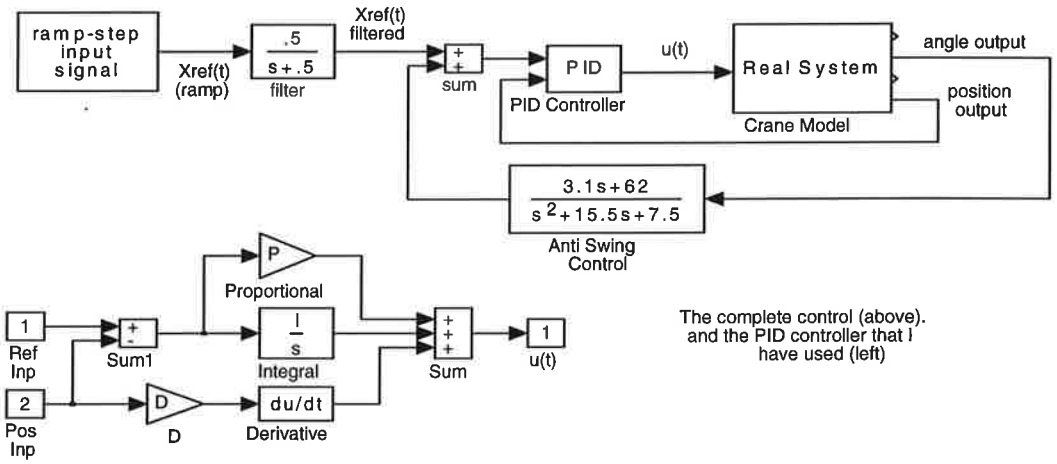


Figure 26 The finished control system for the continuous case.

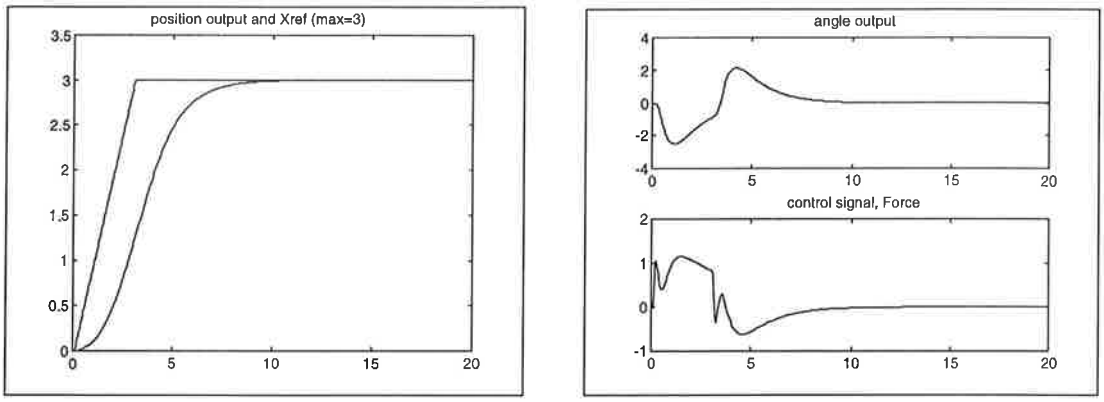


Figure 27 Signals for the control with a relatively small displacement, 3meters

As we see in figure 27 and 28 we get a maximum swing angle of approximately 2 degrees. The middle part of the position curve follows the ramp, that is the velocity is the same. The delay that the curve has compared to the ramp step are the acceleration and deceleration phases of the cart, in order to minimize the angle deviations from zero.

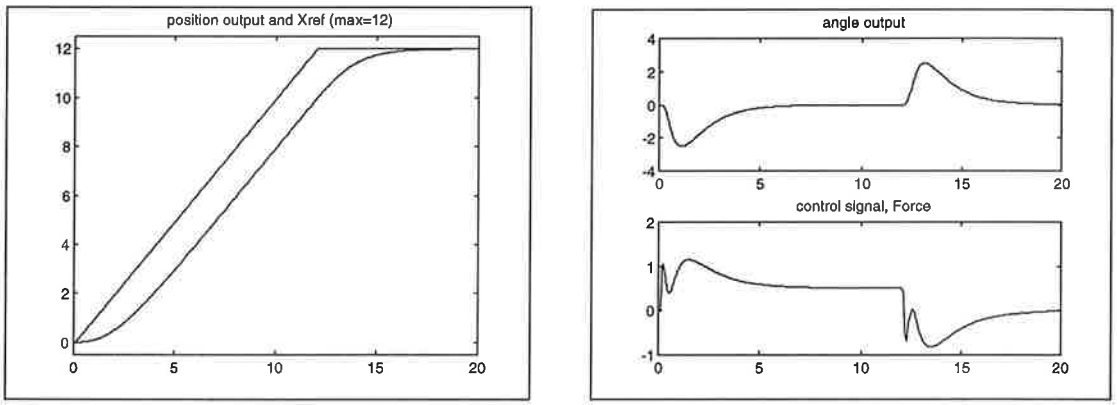


Figure 28 The same signals as in figure 27 for a larger displacement, 12 meters

We can say that the bigger the initial acceleration that we ask from the system, the faster we will get to the goal, but at the cost of a bigger swing angle. In figure 30 the following that the position control performs shows even clearer. As we see, the swing angle depends only on the initial acceleration and the final deceleration. As we remember from the modeling chapter the only outer force that affects the pendulum angle is the acceleration of the "hinge" point of the pendulum.

CONTROL WITH THE SENSORS FOR THE SWINGANGLE

To achieve the other objective of the control of the crane, the control with only the sensor swing angle output, we need to add one part to the block diagram in figure 29, an observer. Since we have no access to continuous measurements of the swing angle of the pendulum, and since the control that we just completed demands just this, we have to invent good guesses of the missing values when there are none present. The ordinary use that we have for an observer in control systems is to reconstruct the states that we can't measure, normally for a state feedback control.

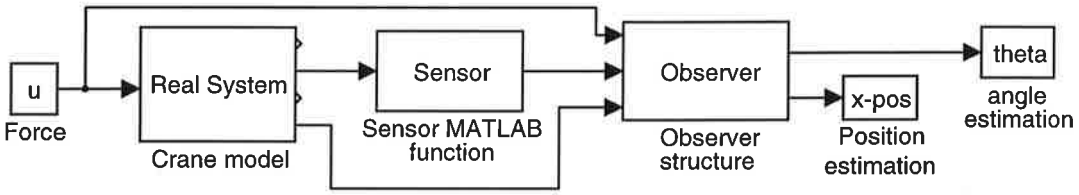


Figure 29 The structure that replaces the Crane Model with both outputs known

THE OBSERVER

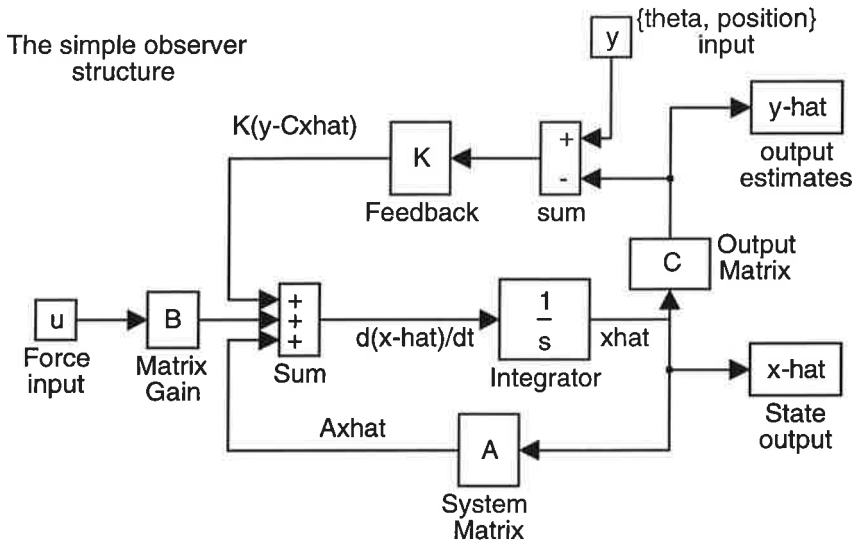
What is it that we have done in our control above ? Because of the derivative terms in our PID and PD controls respectively, we have already "reconstructed" the derivative states and performed state feedback with the coefficients of each one of the P,I and D terms. Now, we take away one state output more, and replace it with sporadic measurements. Can we reconstruct the three missing states from the position output? There are many places in the literature of basic automatic control where we can find the answer, in Åström and Wittenmark (6) we find the criterion in the observability matrix :

$$W_o = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \end{bmatrix} \qquad \text{(eq. 32)}$$

If and only if Rank(Wo)=n where n is the size of A we can find a vector K for our observer, so that the system in the observer is stable. In our case we do have Wo with the full rank, 4, and therefore we can observe the system from the position output. With access to both outputs the system is of course also observable, but in that case we wouldn't need an observer.

THE STRUCTURE

The observer is copy of the real system, in which we compare the "outputs" of the model and the real measurements of the process. The typical structure of a simple observer, that we are going to use is :



$$\frac{d\hat{x}}{dt} = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}) = (A - KC)\hat{x}(t) + Bu(t) + Ky(t) \quad \text{(eq. 33)}$$
$$\hat{y} = C\hat{x}$$

Figure 30 The structure of a "normal" observer and its equations

If we look closer at the structure, in the first representation, we see how the error between the existing outputs and the estimated value of the outputs is feed-backed into the system. But, this affects the dynamics of the system as well, as we see on the right side of the equality. So how do we choose the K to get a stable system, where the error of the estimation is stable and goes to zero? This is a classical problem in control theory, and the golden rule is to choose the observer dynamics so that it is faster than the observed system.

I have made slightly modified observers for the crane system, with good results as long as the model that we put into the observer is more or less correct, with reasonable values of K. In figures 31 and 32 we can se some of the results, first without model error, and then with model errors.

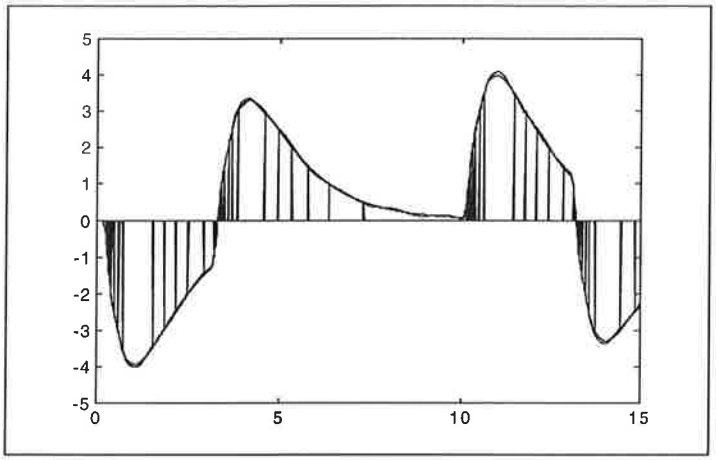


Figure 31 Good following of our observer, no model error

In our case the observer becomes a bit more complicated than in figure 30 since we deal with measurements that come every now and then. The observer works like this. The only signal that always is present is the in-signal to the system, so when there is no measurement available, we let our linear model A,B,C,D act in parallel with the real process. Whenever there is a measurement, we use this to correct the states in the observer. The code in our S-function looks like this (flag=1). K has been chosen to give a system slightly faster than the nominal model.

```
if u(2) ~= 0, %if insignal ;
    dx= A*x+B*u(1)+Kmatrix*(y-C*x);
else
    dx= A*x+B*u(1);
end;
```

Now, what happens with our model when there are no measurements? It continues calculating new guesses all the time to feed the control. When the model follows the process well, there are no problems as we see in figure 31, but if we introduce a model error we get results like in the following figures.

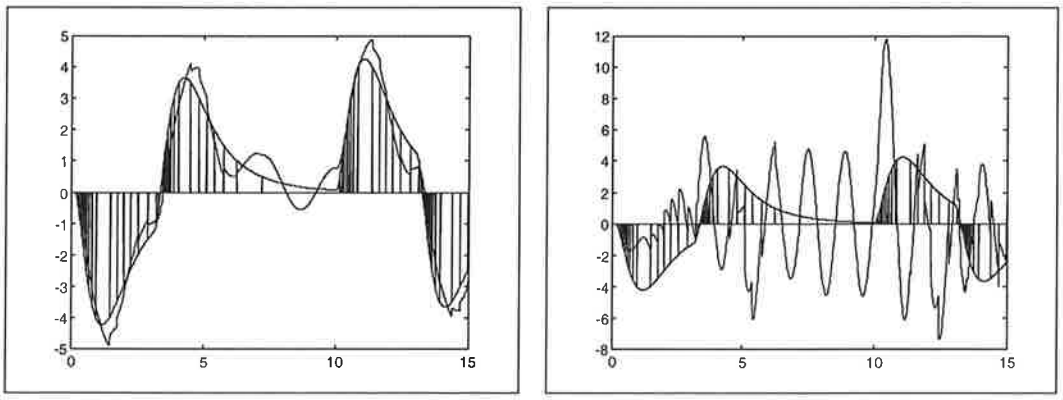


Figure 32 Here we see the same observer but with model errors, in the figure to the left the model has the values M=2,m=0.2,L=0.5and k=0.5 and in the figure to the right, M=1, m=0.2, L=0.5 and k=0.5.

Of course, with better choices of K we might get better results, but it is important to remember that our K is "penalizing" the errors that might occur, and the "penalty" always is the same. The problem is that the K is fixed, because of that the various model errors demand different settings of K. If we use a slightly distorted signal, from the observer outputs, we might get an augmented error in from the control loop as well. The two examples below in figure 33 show how the control becomes jumpy even with a rather small model error. Further down we will see what we can do about this.

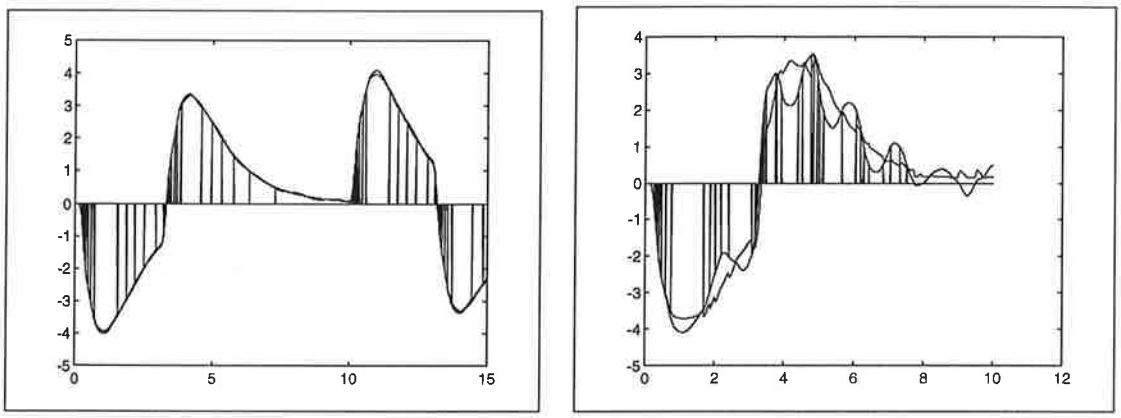


Figure 33 Examples of the observer outputs used for the control. In the first figure without errors, but in the second I have introduced an error M=1 instead of M=2 and here we see how the distortion makes the signals look very uneven.

In this observer we haven't made use of the fact that we actually know the value of the position all the time, and this is of course the next step that we would use in the design of the observer. The new code we can see below. We have to understand that the K is different for the two different cases. In the first, when there are two input values, theta and x, the K matrix has to be a 4x2 matrix, as well as the C matrix is a 2x4. In the other case, they are 4 element vectors, since we only have one measurement available. Then we change the C for what C would have been if we only had one output from the process, that is a vector [0 0 0 1], the lower row of C.

```
if u(2) ~= 0, %if insignal ;
    dx= A*x+B*u(1)+Kmatrix*([theta position]'-C*x);
else
    dx= A*x+B*u(1)+Kvector*(position-[0 0 0 1]*x);
end;
```

Now we only have to choose good values for the two different K How do we get them to play well together, and how do we get them to change with the frequency of the incoming sensor signals?

TIME VARIABLE KALMAN FILTER

To determine an optimal estimator, observer, of the states in our model, we can make use of the Kalman filter. The observer has the same structure as in our earlier model, but we determine the K from a dynamic model that tries to minimize the

variance of the error of the estimates. See Åström,Wittenmark (6) chapter 11.1-3 for the theory behind our estimator. The dynamics of such an observer becomes:

$$\begin{aligned}
\frac{d\hat{x}}{dt} &= A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}) \\
\frac{dP}{dt} &= AP + PA^T + R_e - PC^T R_x^{-1} CP \\
K &= PC^T R_x^{-1} \\
\hat{y} &= C\hat{x}
\end{aligned} \qquad \text{(eq. 34a-d)}$$

What has to be done here is to choose wisely values of the two matrices Rx and Re. But in our case, since we still have the two possible cases, we make two parallel editions of the Kalman filter above. What the two branches will have in common is the P-matrix, and when there comes a measurement of the angle we let this affect the values of this matrix. Since there is less certainty in the system when there is no measurement available, the P-matrix is growing until the next measurement, when it jumps down to the lower level that the upper branch provides.

THE MATLAB SOLUTION

The MATLAB/SIMULINK code (S-function) for the Kalman filter-observer is shown below. Here we see in the case of flag==1 that we have two branches, either with or without the angle measurement. The values of Rx and Re are our possibilities to change the dynamics of the P-matrix. Re can be said to represent the built in uncertainty of the system, modeling errors, noise from within the process etc. while the Rx is the uncertainty of the measurements. In the Re elements we try to estimate how much model errors affect the model output. The other matrix Rx we put lower, whenever there comes a measurement, we let it affect the system much, since the uncertainty for the moment "has gone away".

In the flag==3 case, we give the function output to the rest of SIMULINK. In this case I have chosen to let the state vector be the output for the rest of our system. We may also just lave the two interesting states, theta and position, x, as outputs. Here we also calculate and save the new values of the two K-matrices in global variables.

We want to do this only once every iteration. If we had the K calculation in the same branch as the state and P-matrix iterations we would have the risk of inaccuracy in the state variable calculations. This because the K would change at the same time as the P, in every iteration step.

```
function  [sys, x0] = sfunobserv(t,x,u,flag,nstates,A,B,C,Ktest)
global Kmatrix Kmatcont Re Ry
%u(1)=forceinput,u(2)=thetaout,u(3)=xposition

if abs(flag) == 1,
% If FLAG==1, then SIMULINK is looking for the next state derivative, dx

P=zeros(nstates);
P(:)=x(nstates+1:nstates+nstates*nstates);

  if u(2) ~= 0, %the case where there is a    measurement available
```

```
    dx= A*x(1:nstates)+B*u(1)+Kmatrix*([u(2);u(3)]-C*x(1:nstates));
    dP= A*P+P*A'+Re-P*C'*inv(Ry)*C*P;
  else          %the case where we only know the position of the trolley
    dx= A*x(1:nstates)+B*u(1)+Kmatcont*(u(3)-[0 0 0 1]*x(1:nstates));
    dP= A*P + P*A' + Re - P*[0 0 0 1]'*inv(Ry(2,2))*[0 0 0 1]*P;
  end;

  sys = [dx;dP(:)]; %returning the new state vector and the P-matrix

elseif flag == 3,
  % If FLAG==3, then SIMULINK wants to know what the next output is.
  P=zeros(nstates);

  P(:)=x(nstates+1:nstates+nstates*nstates);
  sys =x(1:4);              %here we leave the state vector as output from the S-
function

  Kmatrix =  P*C'*inv(Ry);  %Calculate the new values of the K-matrices
  Kmatcont=P*[0 0 0 1]'/Ry(2,2);

elseif flag  == 0,          %initialization loop, where all initial values are given
```

The result that we obtain from this observer with the reasonable choices of Rx and Re, shows to be more stable than was the case with the ordinary fixed K observing policy. In the figures 39 and 40 we can see how our observer behaves in the control loop without and with various model errors.

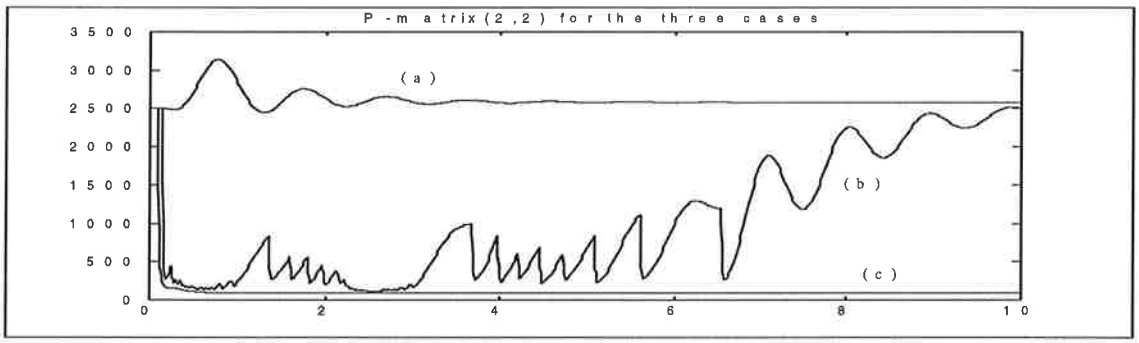


Figure 34 The P-matrix behavior for the three cases a) no values of theta at all c) full information, continuous measurement and b) our case with measurements every now and then

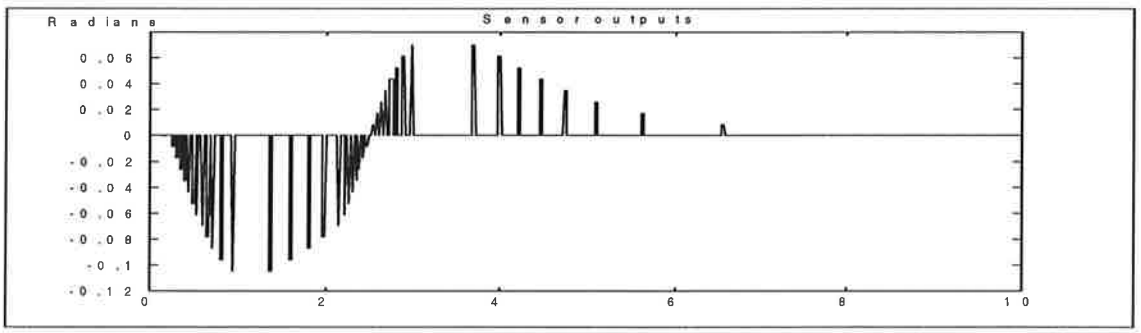


Figure 35 The corresponding angle sensor outputs for the figure 34 above

One important remark is that the P-matrix, and therefore also the K-matrices are not affected by the inputs to the system. In the equations 34a-d , the P-matrix system is a linear system where we put a certain initial value, and coefficients that tells the system (through the eigenvalues) how fast it will go to a certain position. However, in our case we are calculating the system matrix, P, with two different formulas, depending on the input signal. So, basically we are letting the system "uncertainty" depend on the time between two incoming measurements. In the two extremes, where we have either measurements all the time, continuously, or where we have no measurements at all, we obtain the values towards which the P-matrix elements would go in each case. In figure 34 and 35 our more realistic case makes these values "ping pong" between these two lines. The speed with which this happens, and also the final values are given by the choices of Rx and Re.

Note how the different cases correspond to the uncertainty of the angle measurement, as we have in figure 34 case "(a)", only position measurements. This case always follows the P matrix corresponding to the lower branch in the S-function. In the case "(c)" we have full, continuous information. This means that we always enter the upper branch, and calculate the P-matrix value of less uncertainty. This corresponds to lower values of its elements. Our "real" case we find in "(b)". Here we find that in the beginning of the control cycle, where we obtain many measurements, the P(2,2) follows the lower curve of figure 34 and after 5,8 seconds we clearly see how it takes off towards the upper curve. At 6,5 seconds however the last measurement comes in and forces down the P-matrix value towards the lower curve again. Then after this, when there are none, it finally goes up to the upper curve and rests up there. The other elements of matrix P behaves in very similar ways.

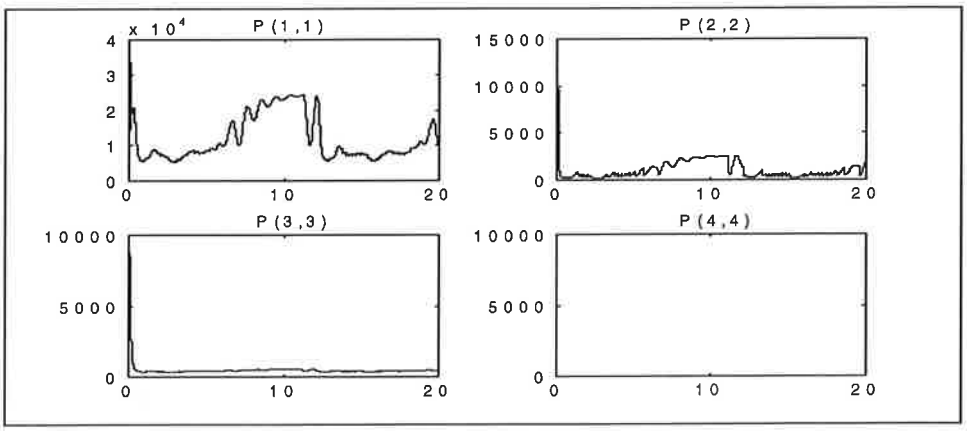


Figure 36 The diagonal elements of the P-matrix with a correct model. The resulting P values for P(3,3) and P(4,4) will always follow the lowest possible P-values since we assume continuous measurements from the position output.

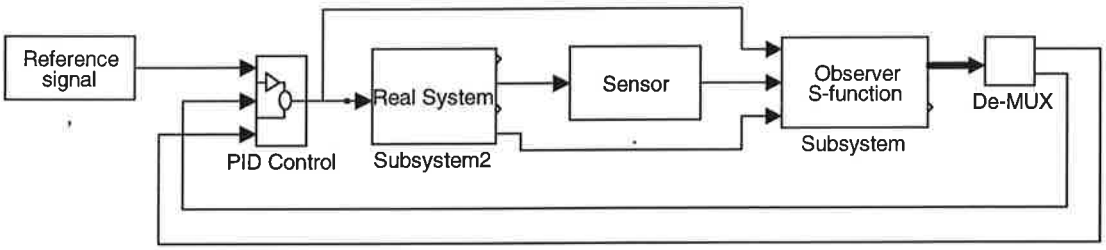


Figure 37 The new control structure with the Kalman Filter-Observer

The new structure of the controller with observer and PIDs will be the one of figure 37. Inside the Observer block I have put a S-function in which the new observer function reveals itself for the one who unmasks the block. Here I have made the P-matrix a part of the output as well, so we can have a look at how the curves changes when there are few measurements available.

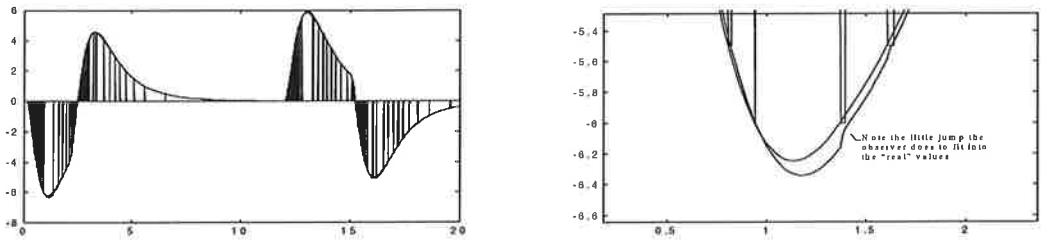


Figure 38 Examples of the function of the new observer. Only angle diagram, since we already may "trust" the position measurements.

With a considerable model error, we still get good control. If we try with the same errors as in the earlier example with the new observer policy we find the behavior to be much nicer.

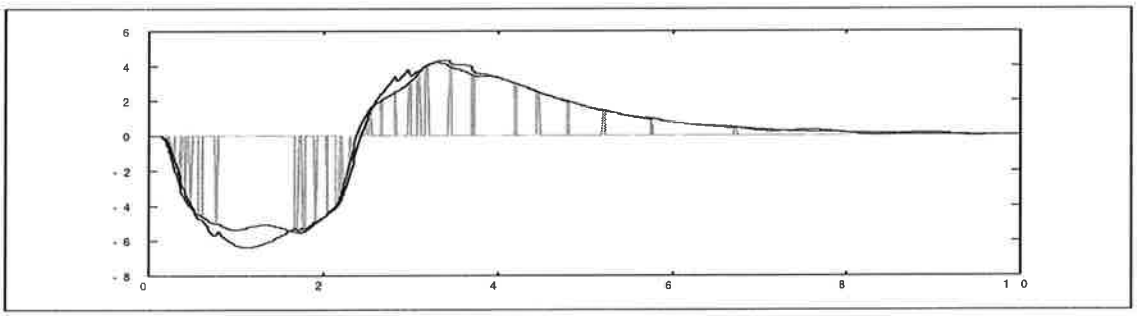


Figure 39 The observer-control with L=0,5 (normally 1)

As we can see in the figures 39 and 40, the behavior is much nicer of our new system, put together with the control from the beginning of this chapter. Depending on how we choose the Rx and Re matrices our P will have different final values, and therefore different behaviors regarding the different coefficients of our system, M, m, L and k. The figures above should be compared with the figure 32 where the same model error is introduced and makes the system unstable, with the earlier observer.

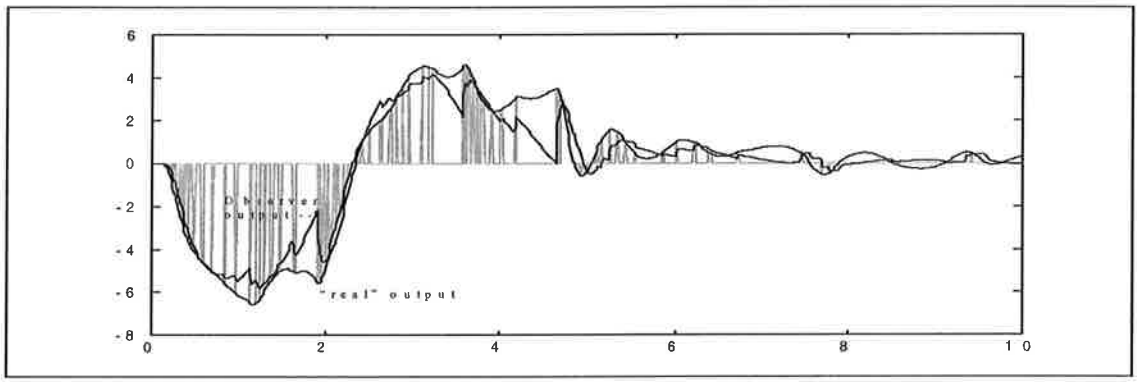


Figure 40 The observer-control system with M=1 (normally 2) and L=0,5 (normally 1)

SAMPLING

Now we consider ourselves ready for the discretization, sampling of the system that we have built so far. Our system consist of the position control loop, the outer swing-damping loop and the observer. We begin with the position control. Then we continue with the anti-swing control and finally we sample the observer.

CONTROL

In the inner loop we have a PID controller, and this includes a pure derivative action and since we cannot find a direct z-transform of a derivative we have to approximate. The outer loop we just find the corresponding discrete system, and replace the old one. First we have to decide what sampling frequency we are going to use in order to get the rest of our functions right.

Sampling frequency, 1/Ts

What we have to think about when we choose Ts is of course the dynamics of the system that we wish to control. A higher sampling rate means safer control, but in a

real time application the risk of overloading the computer. Every sampling period we have to calculate the outputs of the system. The own frequency of the pendulum is about

$$\left.\begin{array}{l}\omega_o = \sqrt{g/L} = \sqrt{9.82} = 3{,}1337\, rad/s \\ thumbrule \quad \omega_o h = 0{,}2 - 0{,}6\, rad\end{array}\right\} \Rightarrow h \approx 0{,}07s \qquad \text{(eq. 35)}$$

However, the fastest oscillating poles in the closed loop system are faster, and therefore I chose to go down to Ts=0,01 to be able to control for model errors and other higher frequency components. The poles of the control are situated in -15±15i so ω will give Ts=0,01s and we go with this :

$$\left.\begin{array}{l}\omega_o = \sqrt{15^2 + 15^2} = 15\sqrt{2} = 21{,}21\, rad/s \\ thumbrule \quad \omega_o h = 0{,}2 - 0{,}6\, rad\end{array}\right\} \Rightarrow h \approx 0{,}01s \qquad \text{(eq. 36)}$$

Position Control Loop

So, here we start from the PID-control given in equation 21 and find an appropriate discrete approximation of the transfer function for each one of the parts. Starting with the simplest, the proportional part we just leave it as it is since it is just a static gain. Then for the integral and derivative parts we make approximations according to the following model

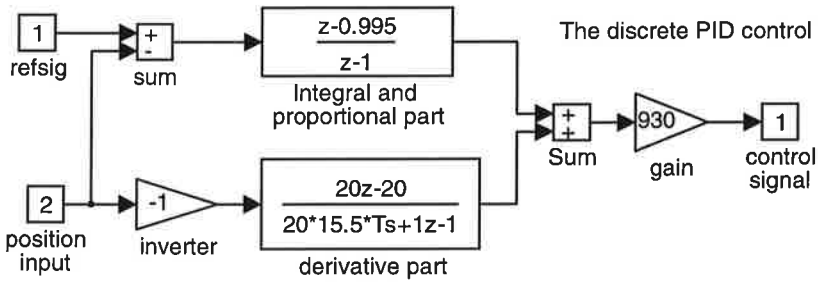


Figure 41 The discrete PID controller in the position control loop

Here we make use of a variation of the original PID control that we used before. First of all, when we approximate the derivative of our signal, we are not interested in the derivative of the reference signal, only that of the position of the cart, so we take the derivative of only this input. Furthermore we build in the filter function :

$$Td \cdot s \approx \frac{sTd}{1+\frac{sTd}{N}} \qquad \text{(eq. 37)}$$

Where the N marks the cutoff frequency of the filter, or rather N/Td. We choose N to be 20 and use the built in function for discretization of SIMULINK for the integral-proportional part and backward difference for the derivative part. Backward difference is when we replace all "s" in a continuous transfer function with the backward difference (z-1)/(z*Ts) The equations the we use then becomes :

$$U(s) = P\left((1+\frac{1}{sTi})(X_{ref}(s) - X(s)) - \frac{sTd}{1+\frac{sTd}{N}} X(s) \right) \qquad \text{(eq. 38)}$$

$$where \quad Ti = \frac{P}{I} \quad Td = \frac{D}{P}$$

And with the given values and backward difference on the derivative part the final equation is:

$$U(z) = P\left(\frac{z+\left(\frac{Ts}{Ti}-1\right)}{z-1}(X_{ref}(z) - X(z)) - \frac{N(z-1)}{\left(1+\frac{NTs}{Td}\right)z-1} \cdot X(z) \right) \qquad \text{(eq. 39)}$$

The first term within the parenthesis is the integral-proportional part, that I have transformed directly with the z-transform, and the second is the derivative term. The final result is what we saw above in figure 41.

Anti Swing Control Loop

The anti swing control then becomes much simpler to discretize since the expression of the control here can be transformed directly by using tables. The function that we used before as anti swing control, and its z-transform are :

$$G_{\substack{antiswing \\ control}}(s) = \frac{3.1s+62}{s^2+15.5s+7.5} \xrightarrow[Ts=0.01\,sec.]{z-transform} H_{\substack{antiswing \\ control}}(z) = \frac{0.0317z-0.026}{z^2-1.86z+0.861} \qquad \text{(eq. 40)}$$

Here I have transformed the function with Ts=0,01 seconds. This shows itself to be sufficiently short sampling period for our control. Of course this number changes with

the characteristics of the system, i.e. with a larger crane, and lower own frequencies, we could do with a lower sampling rate. We see the finished discrete control in figure 42.

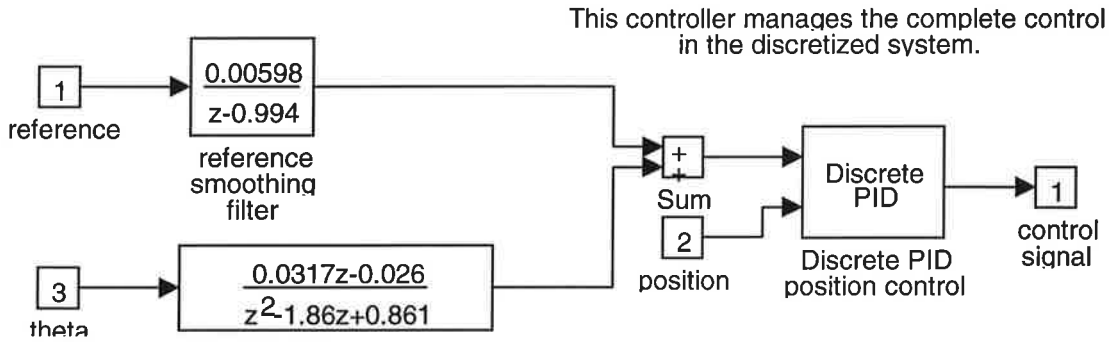


Figure 42 The new discrete control as described in the text. Note the filter on the input that is the z-transformed version of the filter G(s)=0,5/(s+0,5) that we had before. Ts=0,01.

DISCRETE TIME VARIABLE KALMAN FILTER

The final part of out discretization is then the observer that we made in a continuous variant before. The "z-transformation" of this is very easy. We have to introduce some modifications of the S-function that we had before. We are now in the discrete domain, so therefore we have to use the discrete cases of flags in the function. That is, we catch calls to flag=2 instead of flag=1 and give an output value as before on flag=3. The equations 34a-d change a bit as well, and the new system equations are described by equations 41 :

$$\begin{aligned} \hat{x}(k+1) &= \Phi\hat{x}(k) + \Gamma u(k) + K(y(k) - C\hat{x}(k)) \\ P(k+1) &= \Phi P(k)\Phi^T + R_e - K_{matrix}(k+1)CP\Phi^T \\ K(k+1) &= \Phi PC^T(CPC^T + R_y)^{-1} \\ \hat{y}(k) &= C\hat{x}(k) \end{aligned} \qquad \text{(eq. 41)}$$

I have introduced the Greek letters above for the system matrices according to the discrete part of the modeling chapter. We note that the calculation of K now helps us in the P matrix, so I have moved it up to the flag=2 call, and make sure that it is not called

unnecessarily many times every sampling interval. The new state-calculation part looks like this :

```
if abs(flag) == 2,
  % If FLAG==2, then SIMULINK is looking for the next state derivative, dx

  Kmatrix =  A*P*C'*inv(C*P*C'+Ry);
  Kmatcont=A*P*[0 0 0 1]'*inv([0 0 0 1]*P*[0 0 0 1]'+Ry(2,2));

     if u(2) ~= 0,
       dx= A*x(1:nstates)+B*u(1)+Kmatrix*([u(2);u(3)]-C*x(1:nstates));
       dP= A*P*A'+Re-Kmatrix*C*P*A';
     else
       dx= A*x(1:nstates)+B*u(1)+Kmatcont*(u(3)-[0 0 0 1]*x(1:nstates));
       dP= A*P*A' + Re - Kmatcont*[0 0 0 1]*P*A';
     end
     sys = [dx;dP(:)];

elseif flag == 3,
  % If FLAG==3, then SIMULINK wants to know what the next output is.

sys =[x(2),x(4)]';     %we return the outputs to SIMULINK
```

CONTROL W/ OBSERVER

Finally we put the pieces together as before to prove the function of our finished system. The total system as we have constructed it now looks like in figure 43. Compare the curves with the corresponding curves in continuous time.

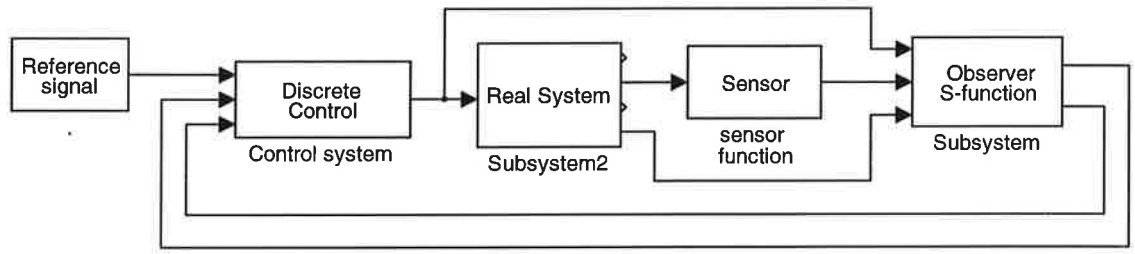


Figure 43 The complete discrete control system of the Gantry Crane

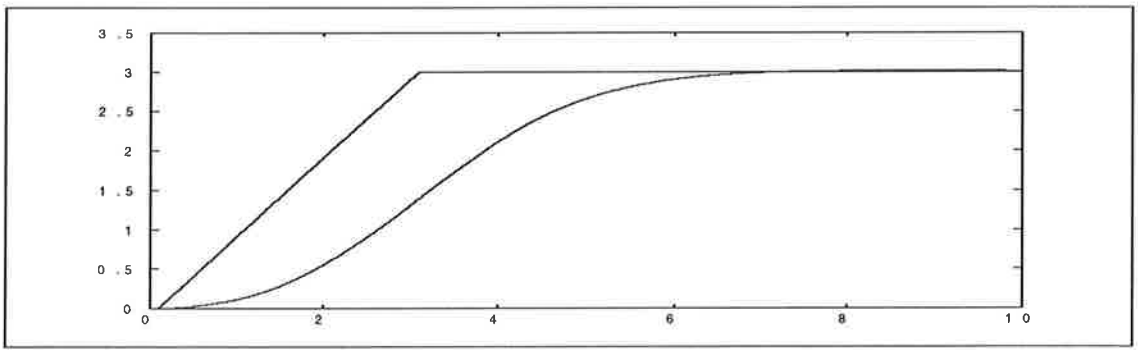


Figure 44 The model following characteristics are the same as in the continuous control

As we can see, the control is functioning very well with the new discrete version,

and if we compare with our results from the continuous control the curves here around look very much the same.

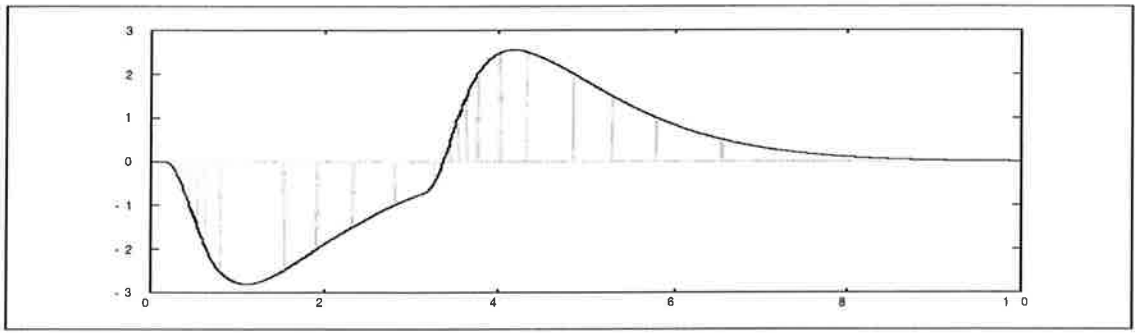


Figure 45And so are the anti swing capabilities of the control

I would like to make a little remark about the P-matrix however. In the continuous case, we understand that the longer time one sensor is giving a signal, two or three iterations in the SIMULINK model, the more is the P-matrix going to be affected by the lower P-matrix value-curve. In the discrete case, however, the probability that the sensor will give signal more than one sample period is very small, and therefore the reaction that we get from our observers P-matrix in the figure 46 is much sharper in the discrete case. In real life the first example should be the most common, since a normal wire may be likely to take more time in passing one sensor than one sample period.

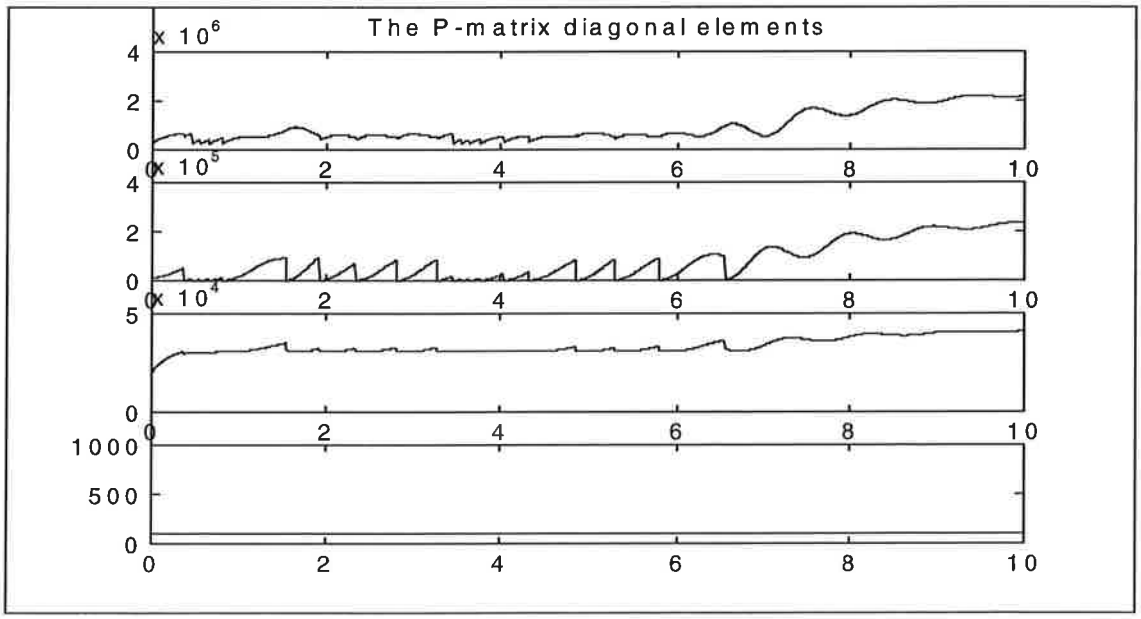


Figure 46 The P-matrix diagonal elements behave very similarly to those of the continuous case as well

PROCESS IDENTIFICATION

One way of making the control more robust to model errors is to make it self adjusting, or adaptive. This can be made in many different ways. The simplest is just to estimate the most important parameters of the system, recalculate the control design and start controlling with the new figures. First we treat the estimation, and then we look into the adaptation.

ESTIMATION

The different types of parameter estimation are mainly variations on the same theme. For different system behaviors we lay out a model, the systems most probable structure, and then fit the model into a series of measurements from the real system. Then we use different laws to penalize deviations from the model, to get the best model fit.

ESTIMATING THE POSITION CONTROL PARAMETERS

We are going to split the problem into the two simplified halves that we were dealing with in the other chapters, Cart and Pendulum. First we start with the estimation of the trolley process from the transfer function in (eq. 20).

To do this we employ an algorithm that minimizes the squared errors in our model with respect to the obtained measurements from the process. This is exactly the same thing as a least square fit to a curve in statistics that most people have done one time or another. The LS-approximation can be made recursive, which is perfect for our purposes.

The regular RLS estimator

The RLS (Recursive Least Square) estimator has a simple structure very similar to the Kalman filter/observer that we used for the reconstruction of the missing data in the earlier chapter. For deeper understanding of the mechanisms look in Jerry M.Mendel, "Lessons in Digital Estimation Theory" (8) where the subject is explained further. In short we can say that if we have a model of a system, we know of which degree it is but not the coefficients of the polynomial of the numerator and denominator of the transfer function, then we can write the function like this :

y(k)+a1y(k-1)+a2y(k-2) ..... amy(k-m)=b0u(k)+b1u(k-1)+b2u(k-2) ..... bnu(k-n)

And isolating the y(k) term on one side gives :

y(k)=-a1y(k-1)-a2y(k-2) ..... -amy(k-m)+b0u(k)+b1u(k-1) ..... bnu(k-n) (eq.42)

Then making a least square fit with each element corresponding to its element in y or u and like this gradually improving the guesses of the coefficient vector [A B] gives us an estimate of this vector that we later can use for various purposes. The recursive equations that we use are the following

$$\theta = \{b_0 \quad b_1 \quad b_2 \quad a_1 \quad a_2\}^T \quad \varphi = \{u(k) \quad u(k-1) \quad u(k-2) \quad -y(k-1) \quad -y(k-2)\}$$
$$\hat{\theta}(k) = \hat{\theta}(k-1) + K(k) \cdot \left(y(k) - \varphi^T(k)\hat{\theta}(k-1)\right)$$
$$K(k) = P(k)\varphi(k) = P(k-1)\varphi(t)(\lambda I + \varphi^T P(k-1)\varphi(k))^{-1} \qquad \text{(eq.43a-d)}$$
$$P(k) = (I - K(k)\varphi^T(k))P(k-1)/\lambda$$

The notation above is that of Åström Wittenmark, Adaptive Control p.53 (5) and includes exponential forgetting, that we set with the factor λ. Normal values are $0{,}9 < \lambda < 1{,}0$. Also we play with the formulas by giving the theta vector, the coefficient vector, good initial guesses, and by setting the P-matrix to high initial values for fast adjustment. If we look at the second line where our parameter vector is formed, we se that this is formed by the last value of the vector plus a factor K multiplied with the difference between the last real output value and the value that our latest estimated model gives. Compare this with the observer that we constructed in the former chapter and see the possibility of using this estimator to estimate the states and the parameters in the same algorithm. Like in the observer, the K matrix is what decides the importance that we give to the error to change the parameters. Therefore we start with big values in matrix P that gives a big K. The program listing is also very similar to that of the discrete Kalman filter/observer and can be read in appendix.

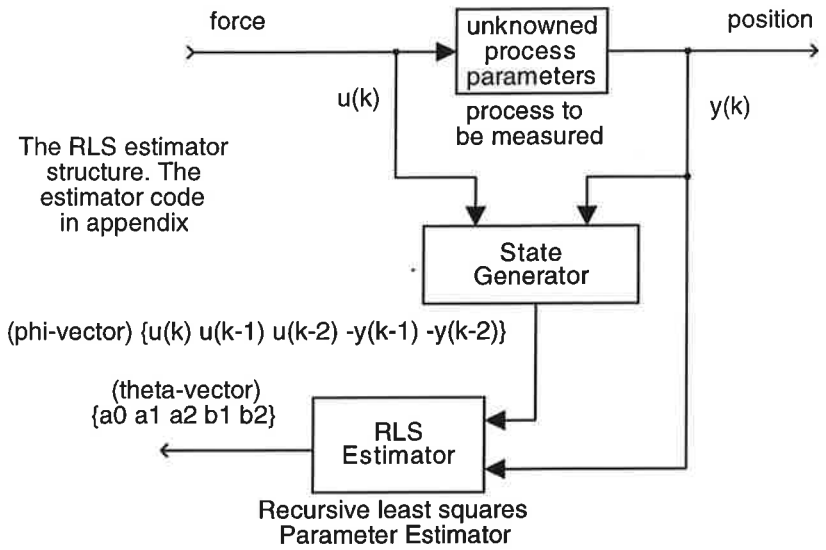


Figure 47 The RLS estimator structure in SIMULINK

The "state generator" is what I use to save the past values of u and y to form the phi vector. Then this, as well as the "fresh" value of y is fed into the parameter estimator. See the code in appendix, for further information. Below we can see the results. We obtain a very fast estimation of the right values with this estimator for this specific process. The true values of the parameter are given with dotted lines.

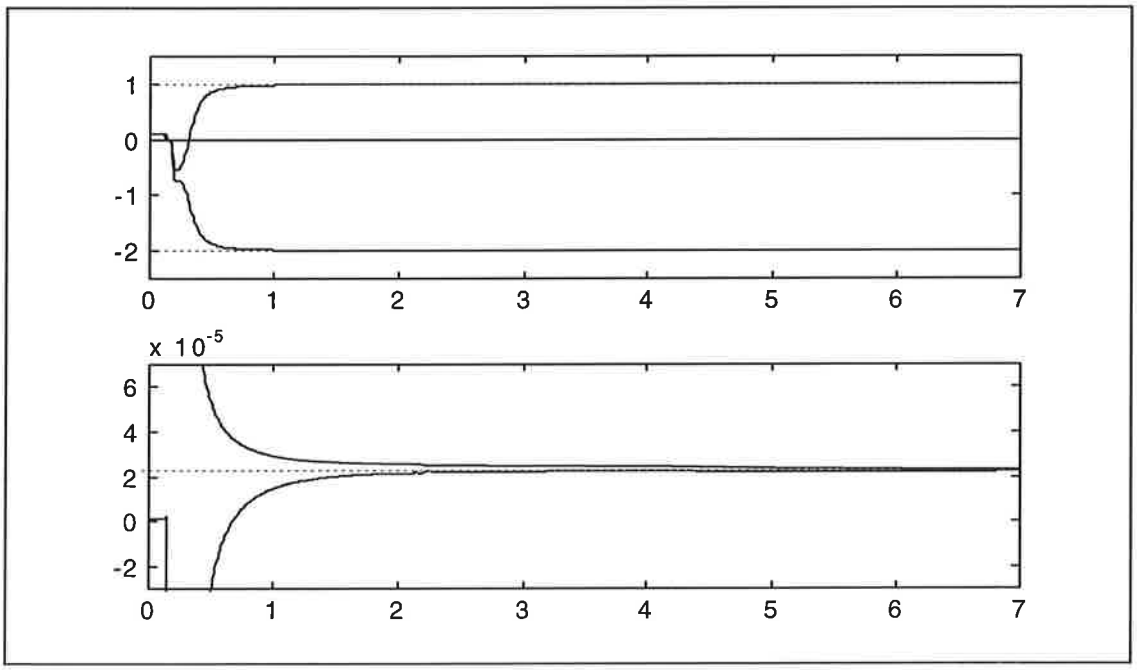


Figure 48 The parameter estimation results with the RLS estimator.

With the parameter values M=2, k=0,5 in the model from before 1/(Ms^2+ks) with the sampling period Ts = 0,01 , we get the parameters A = [-1.0000 -1.9975 0.9975] and B = 1e-4*[0.0000 0.2498 0.2496] and in a model H(z)=B(z)/A(z) as we discretize the model directly in MATLAB with the c2dm command. The zero in the numerator means that there is no second degree term in the numerator, b0 = 0. The latest input value, u(k), does not affect the output y(k).

We see after some experimentation that the constant before the numerator polynomial depends mainly on the M, and one possibility for making the control adaptive for the position control would be to just feedback this constant into an amplifier on the input to the system, and make the control for a double integrator, which is the case when we have a very big M and a small k.

When we want to estimate the second half of the process, we are dealing with another second order system, and from the estimation point of view there is no big difference between the two processes. However it may look a bit odd in the schematics in figure 49 that we are only using the outputs to estimate our system. We have to remember that in our simplified model, the position of the cart is the input to the pendulum process, that we are about to estimate now.

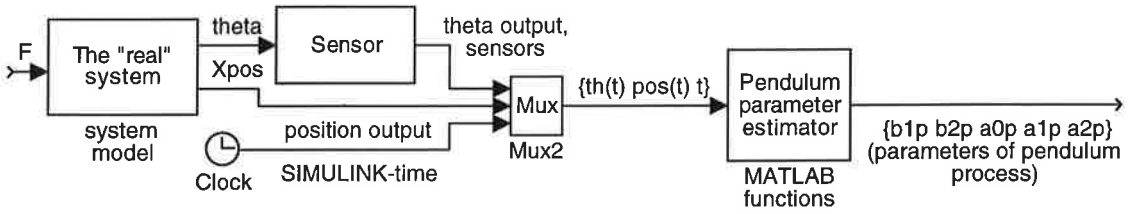


Figure 49 The structure of our pendulum parameter estimator

Discretizing our pendulum system model from before, G(s) = -s^2/(s^2+g/L), with L=1,0 and Ts=0,01 in MATLAB to see what we can expect from our estimator, gives B = [-1.0000 1.9995 -0.9995] and A = [1.0000 -1.9990 1.0000]. This means that now we are going to have to estimate the b0 term too. See the program listings for further information of how the RLS estimators function.

A modified semi on-line strategy

However, our biggest problem here is not how many parameters we will need to estimate, but rather how to gather sufficient information to be able to do the estimation. This problem has been tackled by P.Albertos, R.Sanchis and A.Sala at Universidad Politecnica de Valencia,Spain (2). They present an estimation strategy that mainly suggests that when there are no available data, the output of an adaptive observer is taken instead and being fed into the estimator, via well formed filters. To deal with data that occur in between two samples they propose to interpolate the measurement between the last sample and the coming sample to proportion the values that are fed to the estimator. Then the estimation can work fully on-line, and always with the freshest possible values to use for a control.

Now, considering a rather low data availability rate, given the sensors and continuous position measurements, I have chosen to use a modified version of their strategy. We will use a semi on-line strategy that provides us with as correct parameters as possible, but with the delay of one measurement. We assume that at time k we have earlier estimations of the parameters θ , P, K stored, the last 5 values of the angle-

sensors and a vector of timestamps, when they occured. Also, we need a vector that contains all the position output samples measured from the time for the first angle/time vector value. Parameters θ , P and K are the vectors/matrix that our normal RLS estimator uses, and remembers between the samples. So, whenever a sensor is passed by the wire, the algorithm is activated.

1. Recall the values of parameters θ , P and K, the position measurement vector and finally the angle-time vector and the position output vector saved from the last calculation.

2. Add the last measurement, th(k) to the angle-timestamp vector. We now have the vector with the last 5 measurements {th(k-4) th(k-3) ..... th(k)} , where k is the time for the last measurement.

3. With the vectors, perform the best possible interpolation of the angle as a function of time. In this case I have chosen to use the spline-function of MATLAB to mimic the behavior of the real system.

4. Now, sample the obtained function with the actual sample time and form a vector the is matched with the position output vector.

5. Run the RLS-algorithm described in the previous section from where it was stopped the last time, time(k-2), to time(k-1). Since spline approximates the derivatives and second derivatives in the end point with pure guesses, it is not advisable to use the approximation from time(k-1) to time(k).

6. Store the new values of parameters θ , P and K. Take away the values from time(k-4) to time(k-3) in the vectors and store theses as well.

7. When a new measurement comes start from position 1 with the new parameters θ , P and K, and the stored vectors.

As we see we still have a recursive strategy, so the calculations will always take a finite time. However it does demand rather much computational time since it is saving the work in batches for when a new measurement occurs. In semi code the function will have the following aspect. It always provides the latest calculated set of parameters and needs to be called every sample period.

```
Function [A,B]=CalculateParameters(y,x,time) ;
begin
static var theta, K, P , x, yvector(vector of 5), timestamp(vector of 5)
static var xvector(unlimited list), timevector(unlimited list) ;

Fill the xvector and timevector with the latest values
   if y == 0
      Make a 1 step shift in yvector and timestamp to make room for the latest sample
      Fill the last position of yvector and timestamp
      splineresult:=spline interpolation of yvector/timestamp over the timevector
abscissa.
      For i := TimeLastCalculation : sampletime :timestamp(size-1)
         [Theta,K,P]=RLSestimation(Theta,K,P,splineresult(i-2:i),xvector(i-2:i);
      end
```

```
   end
B=theta(1:NumeratorTerms) ;
A=theta(NumeratorTerms:size(theta)) ;
end
```

Worth mentioning is that the problem that the data from the sensors might come between two samples, or cover two or more sample periods is ignored in this algorithm. In the first case, where the sample comes between two samples we make sure that the value is put into the position of the next sample position. If many samples are covered when the wire passes, we can take the value of the middle sample position and use this. The most important thing about this in the algorithm is that the sample vector does not get filled by values from the same sensor at the same occation.

Results of pendulum process estimation

First of all we take a look at how good the approximation with spline functions is. As we see in figure 53, the approximation is very good. I refer to mathematical literature (10) in numerical methods for further explanation of the cubic spline-functions.

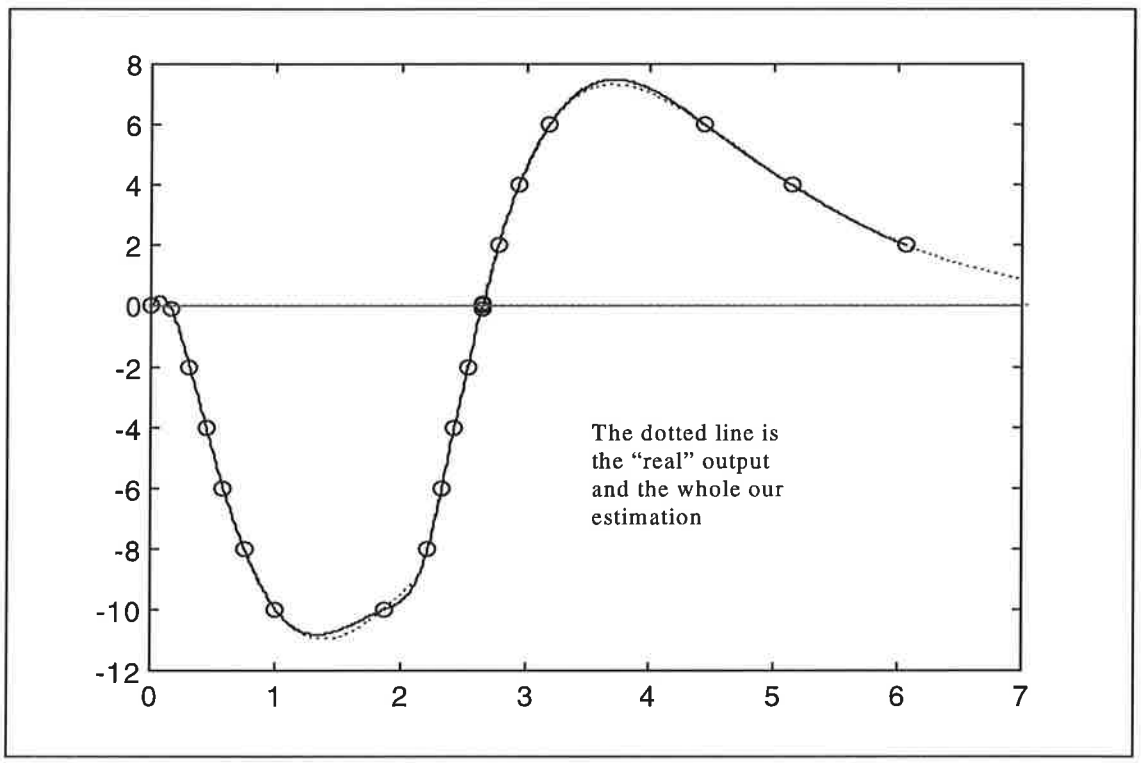


Figure 50 The spline approximation of our sensor-outputs, the circles are the sensor values

When we use the algorithm described above our estimator has an impossible task to perform when we have few measurements to estimate from. Therefore I have chosen

to use a good guess during the first four or five measurements, while the vectors are getting filled with meaningful information.

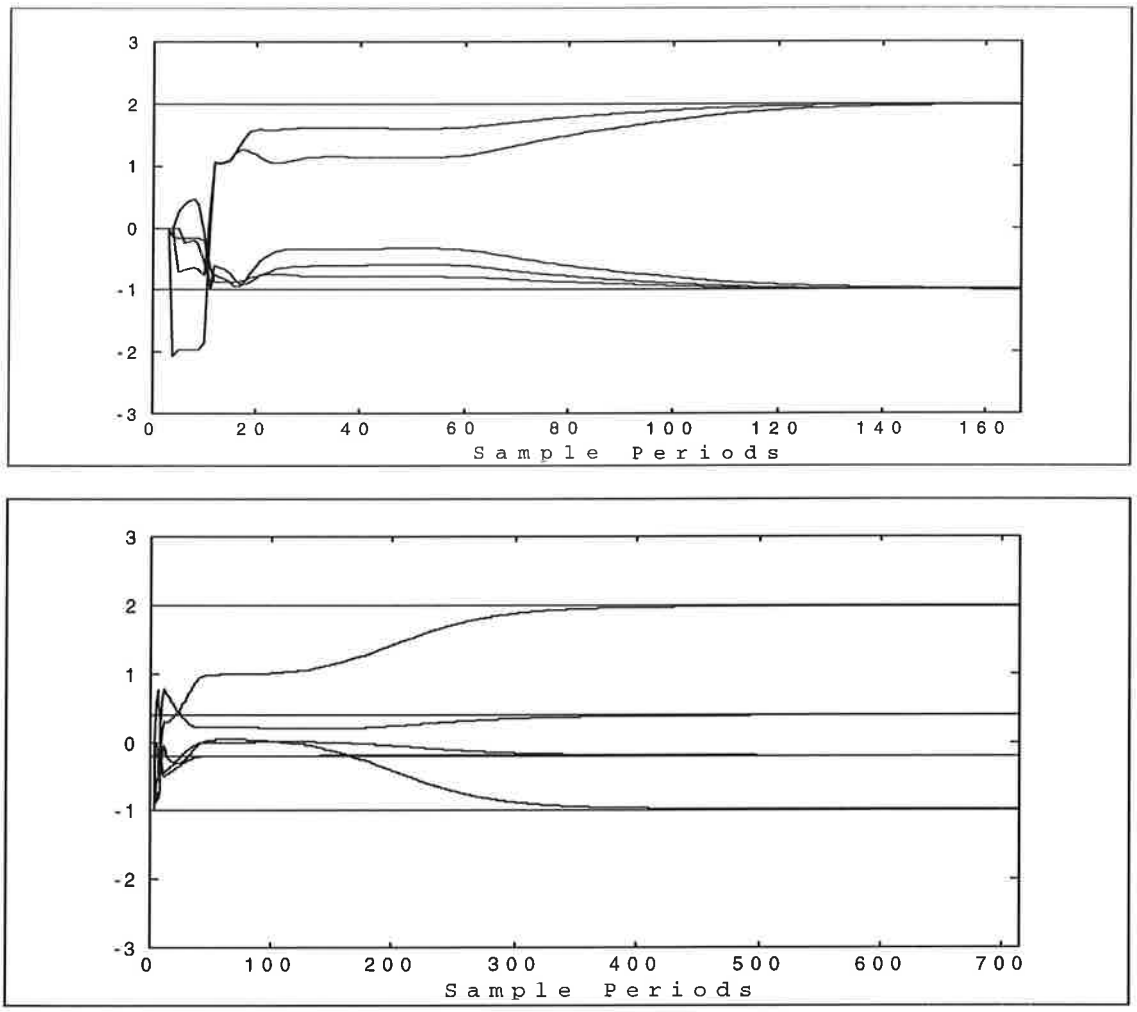


Figure 51 Estimation Result for a system with L=1 (upper image), and for L=5 (lower image), the horizontal lines are the precalculated "correct" values

An important observation that I made is in its place to mention. To get results from the pendulum parameter estimation, we have to make sure we get sufficient exitation of the system. In the examples above the old control from before was employed, but fed with a pure step input instead of the ramp-step that we used before to smoothen the control. This I had to do to get a signal sufficiently abrupt from the angle output. Our control from before is simply too smooth and nice to estimate the parameters well.

ADAPTATION, POSSIBILITIES

So, for what should we now use our estimated parameters? There may be many answers to this, ours could be to improve the control that we have made. As we saw in the control chapter the control works well with all the correct process parameters. That is, when the control is tuned after the true parameters of the system. As we recall there are four physical parameters of our system, the mass of the cart, M, the mass of the cargo, m, the friction coefficient of the cart against its rail, k, and the length of the wire, L.

THE ADAPTIVE POSITION CONTROL SYSTEM

We saw in the chapter before that we are now able to estimate the complete discrete transfer functions of our system. Some questions remains though. What happened to the parameter m in our model ? It just disappeared from our functions.

The truth is that that parameter plays a small role in the position transfer function part. With our assumption that the middle term of equation 2 can be eliminated if m is relatively small it will not appear at all in the transfer function. When we estimate the parameters of this part, we assume a second order system, and this proves to cover the possible deviations caused by the m. Since the estimates are stable and do not move hardly anything from the "model" estimates, we can conclude that this part is not very much affected by variations of m. Maybe we could add it to M as a sort of total mass of the system, but provided that $m << M$, it does not make a big difference.

If we want to improve the function of the position control, we have to do something. We want it to have a fixed behavior so that the anti swing control can act normally even when M, m or k changes. The first thing to try would be to make the control for the case of a pure double integrator, that means that k=0, and then try to estimate the M, that would be reduced to a constant in front of the position transfer function. This M would then be fed back as a multiplying factor on the input to the system.

The estimation of only one parameter can be done with Reduced Parameter RLS, that allows us to do just that. With a model of the continuous system and sampled data, we can isolate one or more parameters of the continuous system and estimate them.

A question that arises is if we really need to estimate and change the control for the mass M. I would say that very rarely the mass of the cart above, that is a part of the crane, and that will not change, need to be estimated. However, two other parameters, k and m, may change and since our parameter estimation from before gives us the whole truth about the system it may be useful to do the parameter estimation. Then pole-placement techniques or other methods can be used to make the control. The need for adaptive control has to be examined thoroughly. Maybe a control robust enough can handle the possible deviations in k and m.

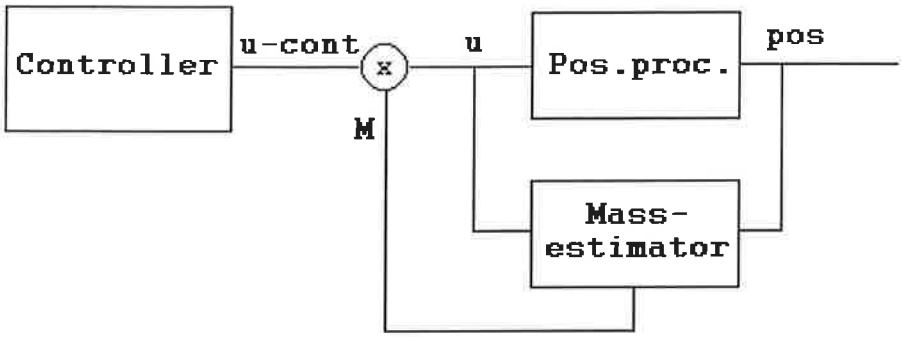


Figure 52 A simple possibility of adaptive control of our position control

THE ADAPTIVE ANTI SWING CONTROL SYSTEM

Now we look at the function of the anti-swing control. Here we have the case where a possible robust control could save us the trouble of the rather time consuming, but possible estimation and adaptation algorithm. We saw before that the adjustment and convergence of the estimator might be too slow for the control.

With more reasonable initial guesses of the process parameters from the beginning a better result is reached for control purposes. In the examples I used [0 0 0 0 0] as the initial values for the RLS algorithm, to show convergence. This can of course be improved with more realistic guesses. It seems to be more difficult for the control to handle shorter pendulum lengths, whereas longer pendulums does not cause as much problems. Therefore it is recommendable to start with initial guesses of too short pendulums.

This once again bring up the question of the necessity of adaptive control. We may experience problems caused by the parameter m. In the physical world, the mass of the cargo will affect the swing characteristics of the system. Imagine that in the moment of passing the vertical line where the swing angle is zero, here the mass has a velocity, and therefore contains a certain amount of movement energy (m*v*v)/2. Now, when the pendulum reaches the end position the potential energy that it contains now has to be the same. The height, alternatively swing angle, will always depend on the existing energy in the system. When we add energy from outside the system the m affects the height the mass will reach and therefore also the swing-angle. The most probable effect that it will have is as a constant in front of the transfer function, with little effect on the dynamics, the swing frequency characteristics.

The parameter L will have this effect as well, but it affects also very much the own frequency of the system. As we remember, the square root of the gravitation divided

by the square root of L is the own frequency of the undamped system that we have. I have tested the swing control with various values and found the control to be rather robust and indifferent for different values of L. However, the estimation of the system constant, the coefficient of the first term in the numerator of the transfer function, could be used to feed back an amplifier factor as in the case with M in the position control to help the anti swing control somewhat.

As we remember from the control chapter, when we made the pole placement of the anti swing control with the root locus method, that all values of P, the proportional feed back were stable. This means that our system will be stable for all lengths of the pendulum, but we also said that for higher frequencies we may have problems caused by model errors and lack of model following of the position control in the higher frequencies.

My suggestion is to improve the robustness of the control to avoid the necessity of adaptive strategies in the system. First when this shows to be difficult, the adaptive control should be contemplated. Our parameter estimations may however be useful for other things. As in the Albertos, Sanchis, Sala paper, a similar strategy is used for fault detection. The small mass m, of the cargo is continuously detected to discover fast changes at an early stage. A fast drop in the mass would mean a load drop, and is of course of great interest for the owner of the load if no one else.

Other uses of the parameter estimation would be similar applications where it is needed an adaptive algorithm and it is dealt with scarce measurements like in our application with the crane. Then the semi on-line algorithm described earlier may be employed as well if the real time demand is not to hard.

CONCLUSIONS

SUMMARY

In this project I have made the control of a gantry crane. First I modeled the crane and verified the model. Then I made a control of it assuming full knowledge of its variables. A discrete signal was introduced instead of the continuous outputs from the crane. This brought up the need for an observer. The observer was implemented in the model with a time variable Kalman filter. Good results were achieved with the original control and the new observer.

The system was sampled and the new discrete system verified. Finally tests were made with estimation of the system variables for possible adaptation algorithms in the control. The trolley part of the system, gives continuous measurement and is easy to estimate with a 2 degree polynomial function. The discrete output signal of the pendulum/wire was treated with a semi on-line estimation algorithm and is also estimated with a 2:nd order function.

CONCLUSIONS

The control shows a simple and understandable solution to the pendulum stabilization problem. It shows how ordinary PID control can be succesfully employed in such problems. The characteristics of the control are very good, and with the control simulated in this paper the cargo swings to a maximum of 2,5 degrees and moves with a speed of approximately 1 m/s for a crane model with wire-length 1 meter, cart weight 2 kilograms and cargo weight 0,2 kilograms.

A very important relation in the physical process is that the swing-angle and swing acceleration are directly proportional to the acceleration of the cart. The maximum speed of the crane is not that interesting in these aspects, only the acceleration. We have to take into account how the reference signal affects the control when it is designed. A ramp position reference causes static error of the signal.

The variables that we want to minimize are the acceleration of the cargo, the maximum swing angle and the time demand to reach goal position. During the control design, the physical properties were the main factors that I worked with. With the demonstrated control, the acceleration of the cart is made very smooth to minimize swing.

In the control the variables used are only the position and swing angle of the system. Velocities and accelerations are not directly used. However, in the PID loops we see that the signals are derived and integrated, and fed back. This means that we really are "observing" these states in the PID.

The missing data case, where we get information from sensors was solved with two parts, one ordinary least square parameter estimator and another with a modified algorithm. In this project the main and most interesting problem has been to solve the problem with the missing data control.

The developed algorithm for parameter estimation seems to work well, and can be generalized to transfer to other similar problems. In this case the cubic spline function approximated the swing behavior very well. Depending on the demand for low noise and correct model in the estimation approximations of varying degree may be employed. For other applications other approximating functions may need to be used with the same algorithm.

Before using these results and applying them to a real world situation, the validity of the assumptions made and necessity of adaptive control have to be carefully investigated.

REFERENCES:

1. The MATHWORKS Inc SIMULINK- Dynamic System Simulation Software. Handbook of SIMULINK with examples
2. P.Albertos, R.Sanchis and A.Sala (1995) Digital model parameter estimation with missing data, Departamento de Sistemas, Computadores y Automática, Universidad Politecnica de Valencia
3. Albertos, R. Sanchis and A. Sala (1994) Fault-detection via parameter estimation in continuous-time systems with random sampling (scarce measurements), Departamento de Sistemas, Computadores y Automática, Universidad Politecnica de Valencia, Proc. Of 4th Symposium Low Cost Control Sept. 1995 Buenos Aires, Argentina
4. Ho-Hoon Lee, Sung-Kun Cho, Jae-Sung Cho,(1997), A new Anti Swing Control Of Overhead Cranes, The proc. of ASI '97 Kyongju, Korea
5. Karl Johan Åström Björn Wittenmark. Adaptive Control, 2nd edition. Addison-Wesley Publishing Company Inc. (1995)
6. Karl Johan Åström, Björn Wittenmark, Computer controlled systems, theory and design. 2nd edition. Prentice Hall international Inc.(1990)
7. John J. D'Azzo & Constantine H. Houpis, Linear control system analysis and design, Conventional and modern. McGraw Hill Book Company. (1975)
8. Jerry M.Mendel Lessons in Digital Estimation Theory. Prentice Hall, Inc. (1987)
9. Johan Eker, Karl Johan Åström, A nonlinear Observer for the Inverted Pendulum
10. Ángel Valera, Master CAD/CAM, funciones spline (Spanish textbook) (1990)

APPENDIX

INITIALIZATION FILES :

The initialization file that were used before every simulation to set the important parameters of the system:

```
tid = [0 .49 .49 10 11 11 12 28 28 30]';
T=[0 .1 2.1 20]' ;
Inp=[0 0 10 10]' ;

M=2;m=.2;
L=1;
k=.5;
K=k;
g=9.82;
sat=50;
wo=sqrt(g/L);
Tperiod=2*pi/wo
Ts=.01
```

SYSTEM TRANSFORMATION FILES

System transformations were used to make the matrices and other parameters for the block diagrams in SIMULINK. Makecont and Makedisc were often used from the initialization file, to provide the SIMULINK models with data.

```
function [A,B,C,D] = makecont(M,m,L,k)
% [A,B,C,D]=makecont(M,m,L,k) takes the parameter
%values of
%   M=mass of cart
%   m=mass of pendulum
%   L=length of pendulum
%   k= frictioncoefficient of cart
%
%as arguments and produces the state space model
A=[0 -9.82*(M+m)/(M*L) k/(M*L) 0;
  1    0          0     0;
  0  m*9.82/M      -(k/M)   0;
  0    0          1     0];
B=[-1/(M*L) 0 1/M 0]';
C=[0 1 0 0;0 0 0 1];
D=[0 0]';
end
```

MAKEDISC, to make the discrete version

```
function [A,B,C,D] = makedisc(M,m,L,k,Ts)
% [A,B,C,D]=makedisc(M,m,L,k) takes the parameter
%values of
%
%  M=mass of cart
%  m=mass of pendulum
%  L=length of pendulum
%  k= frictioncoefficient of cart
%  Ts=SampleTime
%
A=[0 -9.82*(M+m)/(M*L) k/(M*L) 0;
  1    0          0    0;
  0  m*9.82/M -(k/(m+M))*(1+m/M) 0;
  0    0          1    0];
B=[-1/(M*L) 0 (1/(M+m))*(1+m/M) 0]';
C=[0 1 0 0;0 0 0 1];
D=[0 0]';
[A,B]=c2d(A,B,Ts);
end
```

SENSOR SIMULATION

This little simple program takes the new value and the last value of theta and checks if a "sensor" has been passed. It gives no output if that is not the case. The last input (k-1) I provide through a delay of one time unit. In the simulation this has been set to Ts/10 to pretend a continuous time-space.

MAKEOUT, to simulate the Sensors from the continuous input.

```
function [out]=makeout(thnew,thold,angles);
%Makeout3 makes the calculation for
%the output Theta. Interrupted.
salange=0;
 for I=1:max(size(angles)),
    if((salange==0)&(((thnew-angles(I))*(angles(I)-thold)) >0))
     salange= angles(I);
    end
 end
 out=salange;
end
```

ANIMATION FUNCTIONS

The animation consists of three functions. The main function is called anim is a S-function where we use that the flag=0 is only called once, and therefore the intialization is performed then. In anim.m we see that in case of flag=0 the initialization procedure anim0 is called. Then on any flag call the function animast is called. Animast erases the old picture and draws the new picture in the same window. The programs are

modifications of MathWorks own demonstration files for animation with MATLAB.

```
function [ret,x0] = anim(t,x,u,flag,offon)
if flag==0,
  if offon==1; anim0;end;
   x0 = []; ret = [0 0 0 2 0 0];
elseif (flag==2)
   [flag,fig] = figflag('Crane Visualization',1);
   if offon==1,
     if flag,
       ud = get(fig,'UserData');
       animast(t,ud,u);
     end;
   end;
   ret = [];
else
   ret = [];
end;
% end anima

function anim0

CurrentBlock = get_param;
sys = get_param(CurrentBlock,'Parent');

TimeClock = 0;
%RefSignal = str2num(get_param([sys '/' RefBlock],'Value'));
XCart = 0; %u(2);
theta = 0; %u(1);
XDelta = 0.3;
PDelta = 0.1;
FigureName = 'Crane Visualization';
XPendTop = XCart;    % + 10*sin(Theta);    % Will be
YPendTop = 0;        %10*cos(Theta);        % Will be 10
XPendBottom = XCart- 5*sin(theta);
YPendBottom = -1*cos(theta);
PDcosT = PDelta*cos(theta);       % Will be 0.2
PDsinT = -PDelta*sin(theta);      % Will be zero

Figures = get(0,'Chil');
for INDEX=1:length(Figures),
   if strcmp(get(Figures(INDEX),'Type'),'figure'),
     if strcmp(get(Figures(INDEX),'Name'),FigureName),
       Fig = Figures(INDEX);
       set(0,'CurrentF',Fig);
       FigUD = get(Fig,'UserData');
       %Cart = FigUD(1);
       %Pend = FigUD(2);
        %TimeField = FigUD(3);
        %set(TimeField,'String',num2str(TimeClock));

      % Time to exit the routine
       return
     end % if right name
   end % if ifigure
end % for INDEX
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Due to *return* above, rest of code only executed if creating animation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Fig = figure('Unit','pixel','Pos',[100 100 800 450],'Name',FigureName);
axes('Unit','pixel','Pos',[50 100 700 300],'CLim',[1 64], ...
    'Xlim',[-3 13],'Ylim',[-4 2],'Vis','on');%,'off');
Cart = surface('XData',ones(2,1)*[XCart-XDelta XCart+XDelta], ...
   'YData',[.3 .3 ; 0 0],'ZData',zeros(2),'CData',ones(2),'Erase','xor');
Pend = surface('XData',[XPendTop-PDcosT XPendTop+PDcosT; ...
               XPendBottom-PDcosT XPendBottom+PDcosT], ...
   'YData',[YPendTop-PDsinT YPendTop+PDsinT; YPendBottom-PDsinT YPendBottom+PDsinT], ...
   'ZData',zeros(2),'CData',11*ones(2),'Erase','xor');
uicontrol(Fig,'Style','text','Unit','pixel','Pos',[300 35 100 25], ...
   'Horiz','right','String','Time: ');
TimeField = uicontrol(Fig,'Style','text','Unit','pixel', ...
   'Pos',[400 35 100 25],'Horiz','left','String',num2str(TimeClock));

uicontrol(Fig,'Style','push','Pos',[40 40 70 20],'String','Playback', ...
   'Call', ...
   ['ud = get(gcf,''UserData'');' ...
    'if exist(''t'')==1,' ...
    '  for i=1:length(t),' ...
    '    testast(t(i),ud,y(i,:));' ...
    '  end,' ...
    'else,' ...
    '  disp(''Must run simulation first.''),' ...
    'end']);
set(Fig,'UserData',[Cart Pend TimeField],'NextPlot','replace');
drawnow
% end anim0

function animast(time,ud,u)
%PENDSETS Animation for the inverted pendulum demo.
%      PENDSETS(TIME,UD,U) uses set to position the graphic objects
%      for the inverted pendulum demo. UD contains a vector
%      of handles [Cart Pend TimeField SlideControl RefMark].
%      Copyright (c) 1990-94 by The MathWorks, Inc.

XDelta = .3;
PDelta = 0.1;
XPendTop = u(1); % + 10*sin(u(2));
YPendTop = 0 ; %10*cos(u(2));
XPendBottom = u(1)+ 5*sin(u(2));
YPendBottom = -1*cos(u(2));
PDcosT = PDelta*cos(u(2));
PDsinT = -PDelta*sin(u(2));
set(ud(1),'XData',ones(2,1)*[u(1)-XDelta u(1)+XDelta]);
%get(ud(1),'XData')
%set(ud(1),'XData',[u(1) u(1)*2]);
set(ud(2),'XData', ...
   [XPendTop-PDcosT XPendTop+PDcosT; XPendBottom-PDcosT XPendBottom+PDcosT], ...
   'YData',[YPendTop+PDsinT YPendTop-PDsinT;YPendBottom+PDsinT YPendBottom-PDsinT]);
set(ud(3),'String',num2str(time));
% Force plot to be drawn
pause(0)
drawnow
% end animast
```

OBSERVER FUNCTIONS

In the control two unconventional observers were used, one continuous, and later the discrete version of this. Below we can see how the continuous version works.

OBSERVER CONTINUOUS

```
function  [sys, x0] = sfunobserv(t,x,u,flag,nstates,A,B,C,Ktest)

global Kmatrix Kmatcont Re Ry

%u(1)=forceinput,u(2)=thetaout,u(3)=xposition


if abs(flag) == 1,          % If FLAG==1, then SIMULINK is looking for the next state derivative, dx

  P=zeros(nstates);
  P(:)=x(nstates+1:nstates+nstates*nstates);
  if u(2) ~= 0,
    dx= A*x(1:nstates)+B*u(1)+Kmatrix*([u(2);u(3)]-C*x(1:nstates));
    dP= A*P+P*A'+Re-P*C'*inv(Ry)*C*P;
  else
    dx= A*x(1:nstates)+B*u(1)+Kmatcont*(u(3)-[0 0 0 1]*x(1:nstates));
    dP= A*P + P*A' + Re - P*[0 0 0 1]'*inv(Ry(2,2))*[0 0 0 1]*P;
   end;
sys = [dx;dP(:)];

elseif flag == 3,  % If FLAG==3, then SIMULINK wants to know what the next output is.
  P=zeros(nstates);
  P(:)=x(nstates+1:nstates+nstates*nstates);
  sys = [x(2),x(4)]';

  Kmatrix =  P*C'*inv(Ry);
  Kmatcont=P*[0 0 0 1]'/Ry(2,2);

elseif flag  == 0, % sizes(1) = number of continuous states
  sizes(1) = nstates+nstates*nstates; % sizes(2)=discrete states
  sizes(2) = 0;      % sizes(3) = number of system outputs (length of output y)
  sizes(3) = 20;      % sizes(4) = number of system inputs (length of input u)
  sizes(4) = 3;
  sizes(5) = 0;
  sizes(6) = 0;

  Kmatrix=[0 0;0 0;0 0;0 0];
  Re=[20000 0 0 0;0 200 0 0;0 0 1000 0;0 0 0 100];
  Ry=[10 0 ;
        0  1 ];
  P=[ 2.5e4  0  0  0;
        0  2500 0  0;
        0  0  100  0;
        0  0  0  100];
  x0 = [zeros(nstates,1);P(:)];
  sys = sizes';
else
  sys = [];
end    % if abs(flag) == ...
```

In the discrete version the only part that I changed was the case of flag=1 that was changed to the corresponding discrete flag=2 call. In the code the following changes were made.

```
if abs(flag) == 2,    % If FLAG==2, then SIMULINK is looking for the next state derivative, dx

 P=zeros(nstates);
 P(:)=x(nstates+1:nstates+nstates*nstates);

 Kmatrix =  A*P*C'*inv(C*P*C'+Ry);
 Kmatcont=A*P*[0 0 0 1]'*inv([0 0 0 1]*P*[0 0 0 1]'+Ry(2,2));

   if u(2) ~= 0,
    dx= A*x(1:nstates)+B*u(1)+Kmatrix*([u(2);u(3)]-C*x(1:nstates));
    dP= A*P*A'+Re-Kmatrix*C*P*A';
   else
    dx= A*x(1:nstates)+B*u(1)+Kmatcont*(u(3)-[0 0 0 1]*x(1:nstates));
    dP= A*P*A' + Re - Kmatcont*[0 0 0 1]*P*A';
   end

sys = [dx;dP(:)];
```

POSITION SYSTEM RLS-ESTIMATOR

In the system an ordinary RLS estimator was used. This I employed to find out the true characteristics of the cart system. This algorithm is discrete and was therefore used with the sampled system. It can however be used together with any system

```
function  [sys, x0, str, ts]  = rlsests(t,x,u,flag,nstates,lambda,dt)
%RLSESTS S-Function to perform system identification.
%       This function performs parameter estimation using the Recursive Least Squares
%       Parameter Estimation Algorithm with Exponential Data Weighting
%
%     The input arguments are
%
%                nstates:                the number of states in the states vector
%                lambda:                 the exponential data weighting factor
%                dt:                     how often to sample points (secs)
%
%       The RLS estimator is defined by the following equations:
%
%                                  1     P(k-2) * phi(k-1) * [y(k) - phi(k-1)'theta(k-1)]
%               theta[k] = theta[k-1] + ------ * ------------------------------------------------
%                                lambda       lambda + phi(k-1)' * P(k-2) * phi(k-1)
%
%                        1      P(k-2) * phi(k-1) * phi(k-1)' * P(k-2)
%               P(k-1) = ------ * ---------------------------------------
%                      lambda    lambda + phi(k-1)' * P(k-2) * phi(k-1)
%
%       where:
%
%                theta:          the parameter estimates
%                phi:            the state vector
%                P:              the covariance       matrix
```

```
%                lambda:          the exponential data weighting factor
%

if abs(flag) == 2  % flag = 2 --> real time hit
  % sample hit, return the next discrete states, which are the
  % next parameter estimates

  theta = x(1:nstates);                                  % parameter estimates
  P = zeros(nstates,nstates);                            % get covariance matrix
  P(:) = x(nstates+1:nstates+nstates*nstates);
  yk = u(nstates + 1);                                   % system output
  phi = u(1:nstates);                                    % state vector
  est_err = yk - phi' * theta;                           % estimation error
  den = lambda + phi' * P * phi;                 % lambda + phi' * P * phi
  theta_new = theta + P * phi * (est_err / den);         % new parameter estimates
  Pnew = (P - P * phi * phi' * P / den) / lambda;        % new covariance
  sys = [theta_new', Pnew(:)']';                         % return them

elseif flag == 4    % flag = 4 --> Return next sample hit

  sys = [];

elseif flag == 0    % flag = 0 --> Return sizes of parameters and initial conditions

  sys(1) = 0;                                    % 0 continuous states
  sys(2) = nstates+nstates*nstates;  % enough discrete states to hold the estimates
                                                 % and the covariance matrix
  sys(3) = nstates;                          % nstate estimate outputs
  sys(4) = nstates+1;                            % nstate+1 (regression vector + system output)
                                                 % inputs
  sys(5) = 0;                                    % 0 roots
  sys(6) = 0;                                    % no direct feedthrough
  sys(7) = 1;                                    % 1 sample time

  % initialize the covariance matrix and initial estimates
  P = eye(nstates, nstates) * 1e6;
  x0 = [[.1 .1 0 0], P(:)']';        %initial guess for a too light system,
                                     %so that the first control signals
                                     %will not be too big
  ts = [dt, 0];

elseif flag == 3                   % flag = 3 --> Return outputs, only at sample hits
    sys(:) = x(1:nstates);
else
    sys = [];
 end
```

PENDULUM PARAMETER ESTIMATION

In the estimation of the pendulum parameters a semi on-line recursive algorithm was employed. The code is divided into two parts. In the function recest the collection of data is performed. When there is a measurement, the vectors are prepared and sent to the other part which is the RLS estimator that takes care of the batch-process of estimating recursively all the approximated data inbetween two measurements.

```
PENDULUM ESTIMATION SEMI RECURSIVE ALGORITHM
function [thetaut]=recest(yin,uin, Ts, t,LengthGuess)
global maxtime numeratoren denominatoren tidsraknare yvector
global posvector timevector minsens lasttimevalue thetatheta postimevect

%collect all values of x in one vector
%and all the values of theta=/=0
if(t==0)
 maxtime=0;
 timevector=[0];
 posvector=[];
 yvector=[0];
 lasttimevalue=0;
 minsens=110;
 postimevect=[];
 thetatheta=[0 0 0 0 0]';
 tidsraknare=0;
 [numeratoren, denominatoren] = c2dm([-1 0 0],[LengthGuess 0 9.82],Ts,'zoh');

end
nuevovalor=0;

%make sure SIMULINK is not jumping backwards when we calculate

maxtime=max(maxtime,t);

% to get the last possible value of the sensors

if(yin~=0)&(minsens>abs(yin))
  minsens=abs(yin);
end

%create the value-time vector ,nyvekt, fill with zero if zero is passed

lastsensout= yvector(max(size(yvector)));
lasttime= timevector(max(size(timevector)));

if (maxtime==t) & (yin ~= 0) & ((yin ~= lastsensout)|(abs(lasttimevalue-t) > 2*Ts) )
    if (abs(yin)==minsens) & (abs(yvector(max(size(yvector))))==minsens)
      fyllvarde=(yin+lastsensout)/2;
      yvector=[yvector; fyllvarde];
      timevector=[timevector;(t+lasttime)/2];
      if fyllvarde ~= 0    %if we are only "touching" the "0" fill one more position
        yvector=[yvector; fyllvarde];
        timevector=[timevector;t-.1*(t-lasttime)];
      end
      yvector=[yvector; yin];
      timevector=[timevector;t];
      nuevovalor=1;
      lasttimevalue=t;
    else
      yvector=[yvector; yin];
      timevector=[timevector;t];
      nuevovalor=1;
      lasttimevalue=t;
    end
end
```

```
%if there are no new values coming in assume angle = 0;

if(yin==0)&((t-lasttimevalue)>=1.0)&(abs(yvector(max(size(yvector))))==minsens)
      yvector=[yvector; 0];
      timevector=[timevector;t];
      nuevovalor=1;
      lasttimevalue=t;
end

%make the time-vector against the position inputs

if (t>=tidsraknare)&(t==maxtime)
 posvector=[posvector uin];
 postimevect=[postimevect t];
 tidsraknare=tidsraknare+Ts;
end


%spline part to approximate curves and fake a regular sampling
if (nuevovalor ==1)& (rem(max(size(yvector)),4)==0)   %Make estimation every 4:th input
 xi = 0:Ts:t;
 yi = spline(timevector,yvector,xi);
 yi(1)=0;yi(2)=0;yi(3)=0;
 posi = spline(postimevect,posvector,xi);
                        %plot(timevector,yvector*180/pi,'o',xi,yi*180/pi)
                        %uncomment to see the spline approximation
 if(size(yi)==size(posi))
  thetatheta=estpen(yi',posi',LengthGuess,Ts); %RLS-estimation of the vector.
 end
elseif<=4   %give an initial guess to system
 thetatheta=[-denominatoren(2:3) numeratoren]';
end

thetaut=thetatheta;
```

The second part is mainly the same as the ordinary RLS algorithm that we used in the S-function before. Now we cannot make use of such a function, since we only want to calculate the new estimations in case of new data. This takes the last data points and uses them for the estimation.

PENDULUM RLS-ESTIMATION ALGORITHM

```
function [thetaut]=estpen(yin,uin,Lguess,Ts)
%flopcount=flops;

 lambda=.999;
 P = eye(5, 5) * 1e6;
 theta = [0 0 0 0 0]';
 nstates=2;

 [num, den] = c2dm([-1 0 0],[Lguess 0 9.82],Ts,'zoh');
 theta2guess=[-den(2:3) num]';
 thetaguess=theta2guess;

thetavekt=[];


 yin = [0; 0; yin];
 uin = [0; 0; uin];


for r= 3:max(size(yin))

 yk = yin(r);
 phi = [yin(r-1) yin(r-2) uin(r) uin(r-1) uin(r-2)]';

 est_err = yk - phi' * theta;

 den = lambda + phi' * P * phi;
 theta = theta + P * phi * (est_err / den);
 P = (P - P * phi * phi' * P / den) / lambda;
thetavekt=[thetavekt theta];
end

%flopcount=flops-flopcount
hold off;
figure;plot(thetavekt');hold on;
plot([1 max(size(thetavekt))],[theta2guess theta2guess]);
axis([ 0 max(size(thetavekt)) -3 3])
thetaut=theta;
```