Implementation of a Control Strategy for an Inverted Pendulum

Gustafsson, Kjell; Bernhardsson, Bo

1988

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

CODEN: LUTFD2/(TFRT-7405)/1–029/(1988)

Implementation of a Control Strategy for an Inverted Pendulum

Kjell Gustafsson
Bo Bernhardsson

Department of Automatic Control
Lund Institute of Technology
November 1988

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

Document name
Report

Date of issue
November 1988

Document Number
CODEN: LUTFD2/(TFRT-7405)/1–029/(1988)

Author(s)
Kjell Gustafsson, Bo Bernhardsson

Supervisor
Per Hagander, Karl Johan Åström

Sponsoring organisation

Title and subtitle
Implementation of a Control Strategy for an Inverted Pendulum

Abstract

This report describes a program that swings up an inverted pendulum and then controls it in upright position. The pendulum is mounted on a moving cart, and the cart is positioned using a hydraulic cylinder.

The program was written as a project in the graduate course "Design of Control Systems" at the Department of Automatic Control 1988.

Key words
Commissioning ; Unstable Nonlinear plant; Robotics; Real-time control; Saturation; Engineering ingenuity

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
29

Recipient's notes

Security classification

1. Introduction

This report describes a program that swings up an inverted pendulum and then controls it in upright position. The pendulum is mounted on a moving cart, and the cart is positioned using a hydraulic cylinder.

Several different controllers have previously been designed for the system. Cascaded PID:s, and state feedback together with Kalman filtering, are two strategies that have been tried successfully. The design described here is quite similar to the state feedback design, but is implemented as a computer program in contrast to the previous analogue controllers. This makes it easy to include logic to swing the pendulum up from downright to upright position. The program also includes code to drop the pendulum from upright position and then catch it again after it has made one turn. Both clockwise and anti clockwise turns are possible. To facilitate usage, a calibration mode has been implemented.

The program utilizes basically the same sensors and actuators as previous analogue designs. Some antialiasing filters had to be added and also an external amplifier to make use of full signal amplitude to the actuator.

The program was written as a project in the graduate course "Design of Control Systems" at the Department of Automatic Control 1988.

2. A Nonlinear Process Model

The inverted pendulum is shown in Figure 1. It is standing on a cart, which can be moved using a hydraulic cylinder. The input to the system is a voltage u fed to a hydraulic valve. The position of the hydraulic valve determines the differential pressure over the hydraulic cylinder. A pressure large enough to overcome static friction will make the cylinder and the cart move. The force acting on the cart is denoted f.

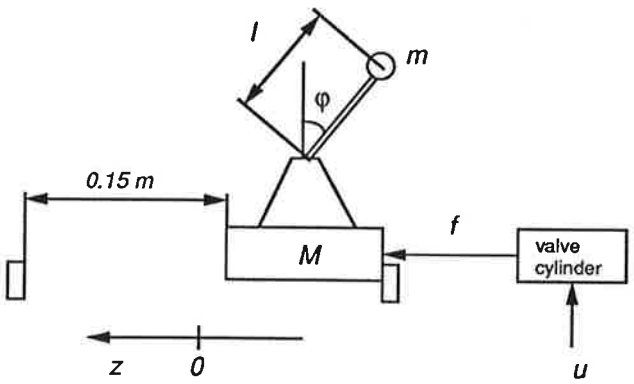

Figure 1. Pendulum and cart

In the modeling we will regard the system as two subsystems: the first consisting of the cart and the pendulum, and the second consisting of the hydraulic valve and cylinder.

Modeling the cart and the pendulum

The kinetic energy T of the cart and pendulum can be written

$$\begin{aligned} T &= \frac{M}{2}\dot{z}^2 + \frac{m}{2}\left((\dot{z} - \dot{\varphi} l\cos\varphi)^2 + (\dot{\varphi} l \sin\varphi)^2\right) \\ &= \frac{(M+m)}{2}\dot{z}^2 - m\dot{z}\dot{\varphi} l\cos\varphi + \frac{m}{2}\dot{\varphi}^2 l^2 \end{aligned}$$

while the potential energy V is given by

$$V = mgl\cos\varphi.$$

Form the Lagrangian as

$$L = T - V.$$

In the generalized coordinates (z, φ), Lagrange's equations for the system are

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{z}}\right) - \frac{\partial L}{\partial z} &= f \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi}}\right) - \frac{\partial L}{\partial \varphi} &= 0. \end{aligned}$$

These equations can be solved for $(\ddot{z}, \ddot{\varphi})$,

$$\begin{aligned} \ddot{z} &= \frac{lf - ml^2\dot{\varphi}^2\sin\varphi + mlg\sin\varphi\cos\varphi}{Ml + ml(1-\cos^2\varphi)} \\ \ddot{\varphi} &= \frac{f\cos\varphi - ml\dot{\varphi}^2\cos\varphi\sin\varphi + (M+m)g\sin\varphi}{Ml + ml(1-\cos^2\varphi)} \end{aligned} \tag{1}$$

For the cart and pendulum at the Department of Automatic Control in Lund, the different physical constants have the following values:

$$\begin{aligned} M &= 19 \text{ kg} \\ m &= 0.25 \text{ kg} \\ l &= 0.20 \text{ m} \\ g &= 9.81 \text{ m/s}^2 \end{aligned}$$

Modeling the hydraulic valve and cylinder

The input signal to the valve is a voltage u. This voltage determines the position of the valve, giving rise to a force $k_f u$ acting on the cylinder. There is also a static friction force f_f, as well as a viscous damping $k_v\dot{z}$. Together these forces add up to the force f acting on the cart. Force balance gives

$$f = k_f u - f_f - k_d\dot{z} \tag{2}$$

The viscous damping is large and for a constant voltage u the cylinder will soon reach constant velocity. In Figure 2 the steady state velocity is plotted as a function of input voltage.

When the cart is moving with constant speed, $f = 0$, then from (2)

$$\dot{z} = \frac{k_f}{k_d}u - \frac{1}{k_d}f_f$$

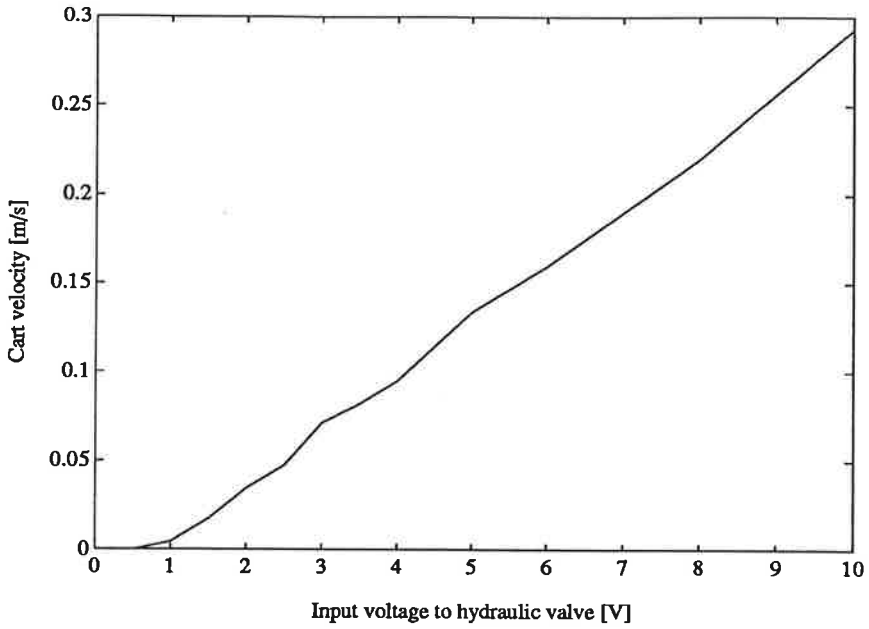

Figure 2. Steady state cart velocity as a function of the control signal u.

The slope of the curve in Figure 2 determines the ratio k_f/k_d. Moreover, using the voltage u_0 that just barely makes the cart move, the friction force can be determined as $f_f = k_f u_0$.

By assuming that the mass of the cart is large compared to the pendulum mass, i.e. $M \gg m$, the force f can be approximated as $M\ddot{z}$. Then

$$M\ddot{z} + k_d\dot{z} = k_f u - f_f \tag{3}$$

The time constant of this system is given by M/k_d, and can easily be estimated by inspecting the step response of the system.

For our system the time constant was determined as 10 milliseconds, and hence

$$k_d = 1900 \text{ kg/s}$$
$$k_f = 59 \text{ N/V}$$
$$f_f = 41 \text{ N}$$

The derived model depends on the hydraulic oil pressure. For the plots and figures presented here it was 45 kg/cm^2.

Limitations

When $|u| \approx 10$ V the hydraulic valve saturates. This will lead to a limitation in cart velocity at approximately 0.3 m/s. Also the position of the cart is limited. The cart can only move ± 7.5 cm from the center position on the track.

Measured signals

The cart and pendulum is equipped with sensors for cart position (z), cart velocity ($\dot{z}$) and pendulum angle (φ). The output form the sensors are given

in the unit V. The conversion from SI-units to voltage is defined by the sensor constants K_z, K_v, and K_φ, where

$$K_z = 144 \text{ V/m}$$
$$K_v = 23.2 \text{ Vs/m}$$
$$K_\varphi = 14.0 \text{ V/rad}$$

Simnon model

The nonlinear model was coded as a Simnon model, to be used for simulation. The limitation in cart position was modeled as two well damped springs with large spring constants. A listing of the Simnon code is given in Appendix A.

3. A Linear Process Model

The pendulum and cart are to be controlled about $\varphi = 0$ and $z = 0$. To facilitate the controller design we will linearize the model about this point.

The angular velocity $\dot{\varphi}$ will normally be small, and the quadratic terms in (1) can be discarded. Moreover, for small φ, $\cos\varphi$ and $\sin\varphi$ can be approximated as 1 and φ respectively. Then (1) can be written

$$\begin{aligned} \ddot{z} &= \frac{f}{M} + \frac{m}{M} g\varphi \\ \ddot{\varphi} &= \frac{f}{Ml} + \frac{M+m}{Ml} g\varphi \end{aligned} \tag{4}$$

From (4) it can be seen that the approximation done in (3) is justified if $M \gg m$. The dynamics of (3) is very fast (time constant $\approx$ 10 ms) compared to the dynamics of the rest of the system. Therefore we choose to discard it. Then

$$\dot{z} = \frac{k_f}{k_d} u - \frac{1}{k_d} f_f$$

and if also the friction force is neglected the hydraulic valve and cylinder can be viewed as a constant gain from u to $\dot{z}$. Using this approximation, and the fact that $M \gg m$, turns (4) into

$$\begin{aligned} \dot{z} &= \frac{k_f}{k_d} u \\ \ddot{\varphi} &= \frac{1}{Ml} f + \frac{g}{l} \varphi \end{aligned} \tag{5}$$

The signal from the position sensor as well as the angle sensor are given in the unit V. The conversion from SI-units to volts is defined by the sensor constants K_z and K_φ. When writing (5) on state space form we would like to scale the states such that they also have the unit V. This avoids some signal scaling, and can be done by introducing the states as

$$\begin{aligned} x_1 &= K_\varphi \varphi \\ x_2 &= \frac{1}{\omega_0}(\dot{x}_1 - p\alpha u) \qquad p = \frac{\omega_0^2 K_\varphi}{g K_v}, \quad \alpha = K_v \frac{k_f}{k_d}, \quad \omega_0^2 = \frac{g}{l} \\ x_3 &= K_z z \end{aligned}$$

This gives the state space form

$$\frac{d}{dt}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 & \omega_0 & 0 \\ \omega_0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} p \\ 0 \\ q \end{pmatrix}\alpha u \qquad q = \frac{K_z}{K_v}$$

with ω_0, p, q, and α having the values

$$\begin{aligned} \omega_0 &= 7.0 \text{ rad/s} \\ p &= 3.0 \text{ s}^{-1} \\ q &= 6.2 \text{ s}^{-1} \\ \alpha &= 0.72 \end{aligned}$$

The parameter α is the steady state gain from u to the velocity $\dot{z}$ measured in the unit V, i.e. $K_v\dot{z}$.

4. Controller Design

Controller structure

A state feedback controller with a Kalman filter is used to control the pendulum around the upright position. The control signal is formed as a linear combination of cart position, pendulum angle, and pendulum angular velocity. The cart position is measured directly, while the pendulum angle and the angular velocity are reconstructed in the Kalman filter. It is possible to measure the pendulum angle directly, but the signal from the sensor includes an unknown offset. This offset is removed by the Kalman filter, and the reconstructed signal is used in the state feedback.

In the previous section it was concluded that by neglecting friction and some fast dynamics the transfer function from control signal u to cart velocity $\dot{z}$ could be viewed as a constant gain. To further justify this view some of the previous designs have included an inner feedback loop around the hydraulic cylinder (see Figure 3). Ideally, one would like the loop gain in this loop to be large, to make the transfer function approximately equal to unity in the frequency band of interest. As seen in the previous section the static gain from u to the measured velocity $K_v\dot{z}$ is quite moderate ($\alpha = 0.72$), and the gain in the controller needs to be large in order to achieve large loop gain. Previous designs have had the gain equal 1, which gives a quite small loop gain. An attempt was made to raise the gain in the controller, but that lead to limit cycles oscillations.

Since the loop gain in the inner feedback loop is quite low (0.72) its justification may be questioned. Still, it was decided to keep it, one large reason being lazyness. The main effect of the inner loop is a change in gain from the control signal u to the measured velocity $K_v\dot{z}$

$$\tilde{\alpha} = \frac{\alpha}{1+\alpha} = 0.42,$$

but it also changes the dynamics. If it was to be omitted the state feedback controller would have to be redesigned.

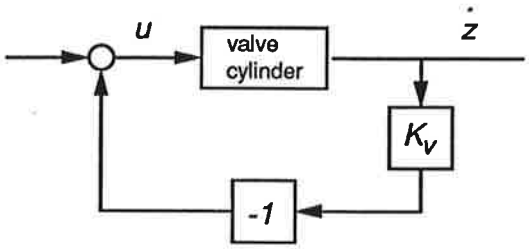

Figure 3. Inner feedback loop

Kalman Filter

An extra state x_4 is introduced as the unknown offset in the angle measurement. The state x_3 can be measured and is completely decoupled from the rest of the states and is therefore not included in the estimator. The Kalman filter equations are then

$$\frac{d}{dt}\begin{pmatrix}\hat{x}_1\\ \hat{x}_2\\ \hat{x}_4\end{pmatrix} = \begin{pmatrix}0 & \omega_0 & 0\\ \omega_0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix}\hat{x}_1\\ \hat{x}_2\\ \hat{x}_4\end{pmatrix} + \begin{pmatrix}p\\ 0\\ 0\end{pmatrix}\tilde{\alpha}u + \begin{pmatrix}k_1\\ k_2\\ k_4\end{pmatrix}(x_1 - \hat{x}_1 - \hat{x}_4)$$

$$\hat{y} = \begin{pmatrix}1 & 0 & 1\end{pmatrix}\begin{pmatrix}\hat{x}_1\\ \hat{x}_2\\ \hat{x}_4\end{pmatrix}$$

Note that $\tilde{\alpha}u \approx K_v\dot{z}$ (or if no inner loop is present, $\alpha u \approx K_v\dot{z}$). The quantity $K_v\dot{z}$ will differ from $\tilde{\alpha}u$ due to e.g. saturation, friction, and unmodeled dynamics. Therefore, the Kalman filter states will better reflect reality, if using $K_v\dot{z}$ instead of $\tilde{\alpha}u$.

State feedback

The control signal is formed from to terms. One given by the state feedback and one given by the inner loop.

$$u = -L\begin{pmatrix}\hat{x}_1 & \hat{x}_2 & x_3\end{pmatrix}^T - K_v\dot{z}$$

In the state feedback, $\hat{x}_1$, $\hat{x}_2$ are estimated and x_3 measured directly.

Choosing L

Three poles are to be positioned. Two of them correspond to the pendulum while one is related to the cart position. The pendulum should be quite stiff in its upright position. This is required to be able to handle disturbances well (people tend to "hit" the pendulum to see if it falls). To achieve this the two pendulum modes need to be quite fast. On the other hand the requirements of the position mode are very modest. The cart should be kept at the center of the track, but there is no reason to waste control energy by making the bandwidth of this loop high.

Some experimentation clearly showed that the main limiting factors of the system are the saturation in cart velocity, the limitation in cart position, and

the sensor noise. The pole positions were in large decided by these limiting factors. After some tuning the pole locations were chosen as

$$s_{1,2} = -15 \pm 15i, \qquad s_3 = -0.36$$

During the tuning of the controller, much insight was gained by looking at eigenvector directions. These vectors make it possible to estimate how much the different modes relate. In particular it is easily seen that the slow mode corresponds to position control of the cart.

Choosing K

The trade off when choosing the Kalman gain is between noise magnification and problems with too slow convergence of estimates making the pendulum fall when connecting the state feedback controller. Clearly the angle measurement offset does not change in the same time scale as φ and $\dot{\varphi}$. Therefore the Kalman filter mode corresponding to the offset need not be as fast as the other two modes.

When this fact was realized and utilized the performance of the system improved considerably. This was especially clear when trying to catch the pendulum. Then it is very important that the Kalman filter soon gives accurate estimates of the different state variables, otherwise the catch will fail.

After much experimentation, the poles of the Kalman filter were placed at

$$s_{1,2} = -15 \pm 15i, \qquad s_3 = -1$$

The Kalman filter states were connected to the analogue outputs of the computer and an oscilloscope was used to monitor their transient and steady state behavior. This was an extremaly useful tool in the evaluation of different pole locations in the Kalman filter.

Discrete implementation

Since the controller is implemented as a computer program it has to be discretized. This was done by sampling the process (calculate Φ and Γ) and then calculating a L and K using translated continuous time poles (e^{sh}). The sampling time h was chosen as 3.5 milliseconds, which was the maximum speed possible on the IBM PC-AT given the calculations that had to be performed.

The discrete time state feedback and Kalman gain was

$$L = \begin{pmatrix} 26.0 & 55.1 & -1.20 \end{pmatrix}$$

$$K = \begin{pmatrix} 0.139 & 0.254 & -0.0304 \end{pmatrix}^T$$

5. Strategy to Swing Up and Catch the Pendulum

When swinging up the pendulum the goal is to give it enough energy to make it reach the upright position and then catch it there. To swing the pendulum up as fast as possible, one can calculate how to move the cart to input as much energy as possible to the pendulum at each time instance (c.f. computed torque methods from robot control). We have chosen a much simpler strategy. Our goal was not to swing the pendulum up in minimum time, but just to be able to do it with a simple strategy.

Swinging strategy

By moving the cart back and forth the pendulum will start to move. The amplitude of the oscillations can be increased by moving the cart in a certain pattern. Each time the pendulum passes the downright position the cart is moved in the opposite direction. The movements of the cart are large when the amplitude of the oscillation is small, and then they are gradually decreased as the pendulum comes closer to the upright position.

The described strategy is easy to implement. One only needs a position servo for the cart, and then it is just a question of choosing the right set-points. The position servo is implemented as a proportional controller (see Figure 4). The inner feedback loop around the hydraulic valve and cylinder was not used.

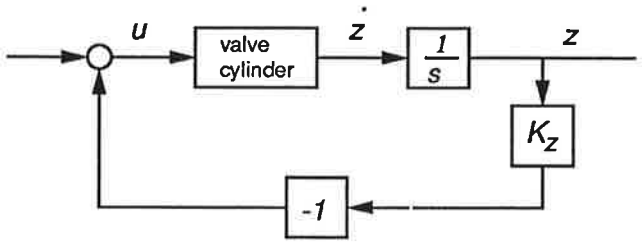

Figure 4. Cart position servo.

The gain in the proportional controller as well as how to choose set-points in relation to the amplitude of the oscillation was determined using simulations and experiments. A value of 2 for the gain was found appropriate, and the relation between the amplitude of the cart movements as a function of the angle of the pendulum was chosen as

$$z_{ref} = \text{sign}(\varphi)\left(0.3 + 4.4\left(\frac{\varphi_{min}}{\pi}\right)^2\left(3 - 2\frac{\varphi_{min}}{\pi}\right)\right) \qquad \text{[cm]}$$

with φ restricted to the interval $(-\pi, \pi]$. The variable φ_{min} is the minimum absolute value of the angle reached by the pendulum so far. The function is plotted in Figure 5.

Catching strategy

The used catching strategy is very simple. When the pendulum moves into a sector $\pm 4°$ around the upright position the state feedback controller is connected to the process.

To make the transfer bumpless the Kalman filter need to be initialized. At the connection the angle is measured and $\hat{x}_1$ is initialized with this value. It is possible to get an approximate value of the offset in the angular measurement by measuring the angle when holding the pendulum in upright position, see mode 9 below. This approximate value is used to initialize $\hat{x}_4$. The program contains a default value of the offset that was correct in August 1988.

A tricky state to initialize is $\hat{x}_2$, corresponding to the angular velocity. First it was tried just setting it to zero, but then it was very hard to succeed in catching the pendulum. A better strategy is to try to estimate the angular velocity by forming the difference between two succeeding angular measurements. Unfortunately the angular measurement is very noisy and the estimate

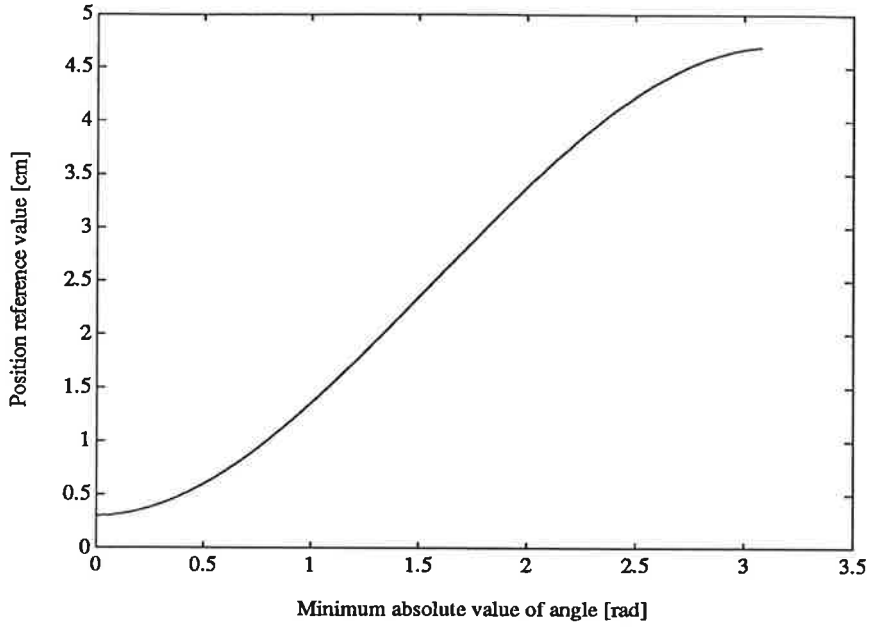

Figure 5. Relation between set-point to position servo and pendulum angle.

has poor quality. This is circumvented by filtering the differences with a first order filter, i.e.

$$x_{2est}(n+1) = \xi x_{2est}(n) + (1-\xi)\, K_\varphi\, (\varphi(n) - \varphi(n-1))$$

This estimate is then used to initialize $\hat{x}_2$. In the program we choose $\xi = 0.9$ which together with a 3.5 milliseconds long sampling interval corresponds to a filter with a time constant of 35 milliseconds.

Rotating the pendulum one turn

The pendulum entertainment show contains one extra item. The program includes code to make the pendulum swing one turn either clockwise or anti-clockwise. This is achieved by switching from pendulum angle control to pure position control of the cart. The cart is then moved either to the left or to the right to make the pendulum fall. After the pendulum having made one turn the angle control is connected again.

When switching to angle control again the same bumpless transfer strategy as when swinging up the pendulum is used. Moreover, by experimenting a little it was found how much to move the cart to make the pendulum exactly reach the upright position. This further improves the transfer to angle control.

6. Simulation of the Catching Strategy

The system was simulated to better understand its properties. The main objective was to find for which initial angles φ and initial angular velocities $\dot{\varphi}$, the controller would be able to catch the pendulum. The result of that simulation is shown in Figure 6. Each successful catch is marked with an 'x'.

From practical experiments it was noted that failed catches seemed to depend mainly on saturation in either cart velocity or cart position. This was

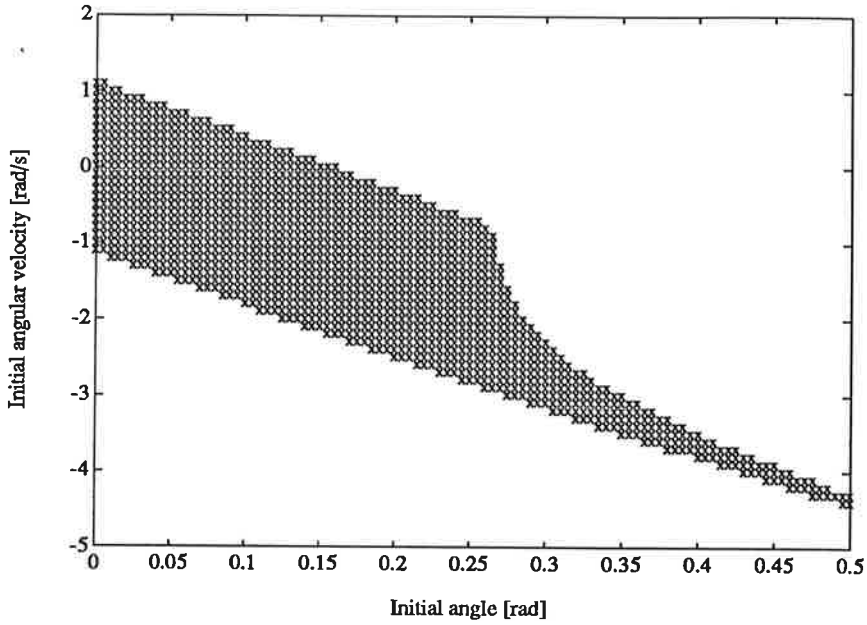

Figure 6. Successful catches as a function of initial angle φ and initial angular velocity $\dot{\varphi}$.

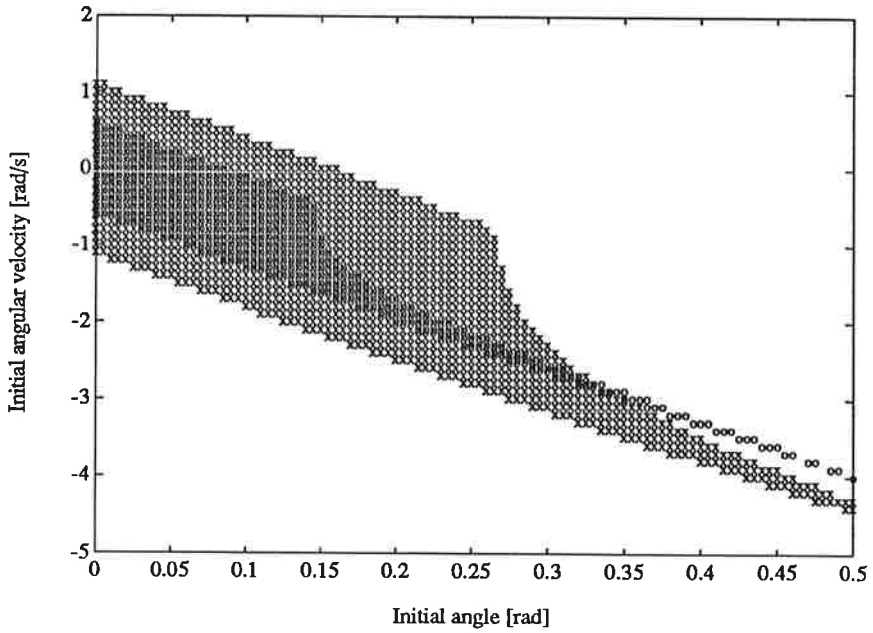

Figure 7. The influence of control signal saturation on the success of the catching strategy. The points marked with 'x' corresponds to u saturated at ± 10 V, and 'o' corresponds to u saturated at ± 5 V.

confirmed through simulation (see Figure 7 and Figure 8). Note that in Figure 7 there are initial values for which a catch is possible when the control signal is limited to $\pm 5V$, while it fails when the limitation is $\pm 10V$. This indicates that the controller could be improved.

The linear model is a very good approximation at the pendulum positions where a catch is possible. This can be seen in Figure 9 were the results

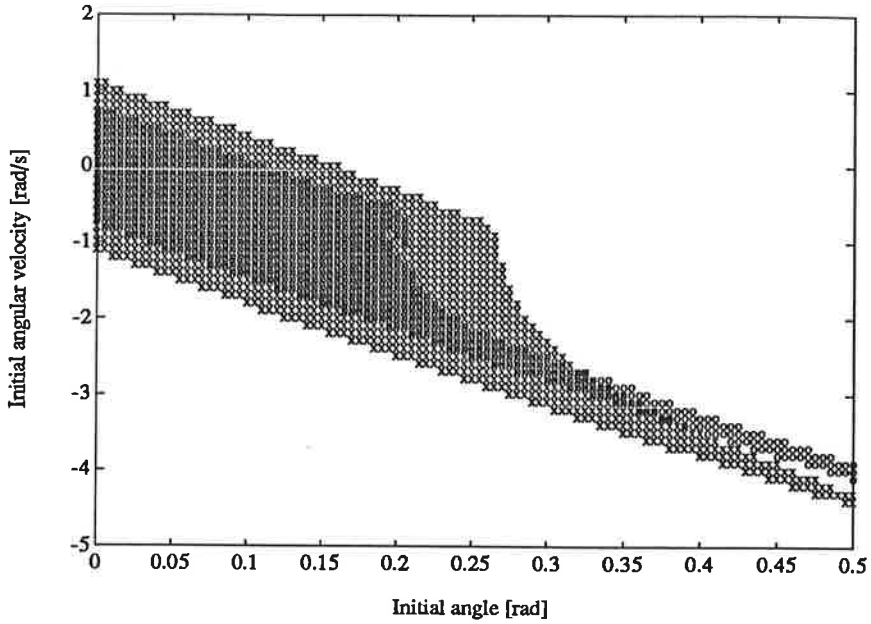

Figure 8. The influence of limited cart position on the success of the catching strategy. The points marked with 'x' corresponds to z limited to ± 7.5 cm, and 'o' corresponds to z limited to ± 3.5 cm.

from simulation using the linear model almost overlaps the results from the nonlinear model. Using the linear model it would probably be possible to calculate the region for successful catches analytically. No attempt to do this was done.

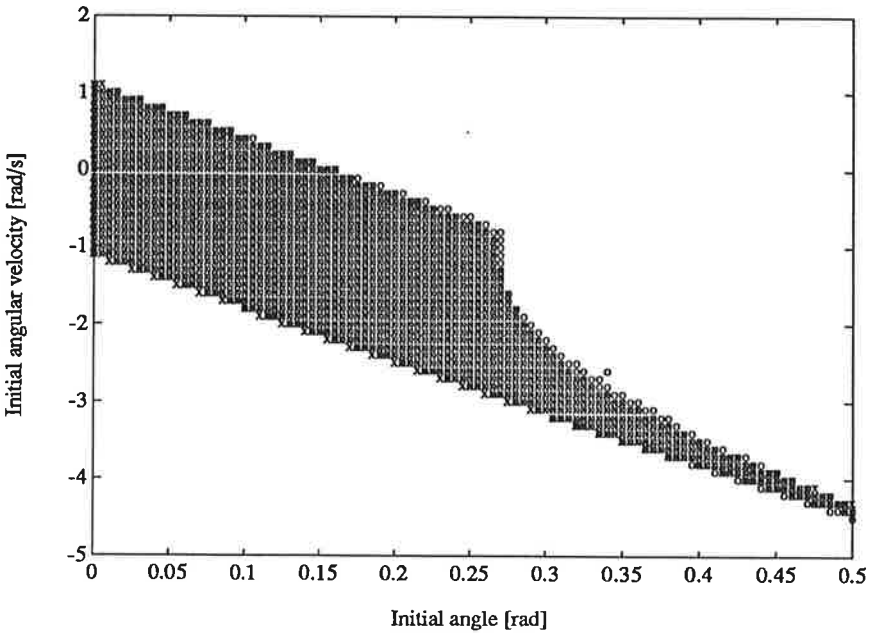

Figure 9. A comparison between the nonlinear and the linear pendulum model. Points marked with 'x' corresponds to the nonlinear model while 'o' corresponds to the linear.

The different Simnon models used in the simulations are listed in Ap-

pendix A.

7. Implementation

Hardware

To be able to swing up the pendulum some minor changes had to be done to the hardware used by previous designs.

With the angle sensor it is only possible to resolve an angle modulo $\pi/2$, and hence during a full 2π turn four different angles will give the same output from the sensor. This was handled by adding a new sensor consisting of a photodiode and a disc with black and white sectors. Using that setup it is possible to detect when the pendulum moves between different $\pi/2$ sectors. The program gets the angle measurement as well as the sector measurement and can easily determine the correct angle in the range $\pm\pi$.

All the sensors use a system with carrier frequency and thus the signals have a large high frequency component. Therefore to avoid aliasing the signals have to be filtered. First order RC-filters with a cut-off frequency of 35 Hz were used.

The signal range from the D/A-converters on the computer were ± 10V, while the actuator accepts a control signal in the range ± 15V. To use the full range an external amplifier was added. It has the gain 3/2 and therefore the control signal has to be attenuated a factor 2/3 internally.

Software

The program (a listing is given in Appendix B) uses a foreground-background scheduler due to Dag Brück, [1988]. It is written in C and assembly language and runs on the IBM PC-AT. The scheduler is optimized for speed.

The foreground process implements the controller. It also includes the logic for swinging up the pendulum and making it swing one turn. The controller is implemented as a finite state machine. Some state transitions are automatic and depend on different logical tests, while others have to be initiated from the keyboard. The only task for the background process is to read these state transitions from the keyboard.

The state machine has 11 different states. The program includes a variable `mode` which tells what state is currently active. The `mode`-variable is used in a large `case`-statement that constitutes the main part of the foreground process. The rest of the foreground process handles the angle measurement. The sensor handles a $\pi/2$ sector and some logic has to be added to deal with a full $\pm\pi$ range.

The states in the state machine are as follows:

0 **Idle.** The cart is kept at the middle of the track.

1 **Controlling.** The state feedback controller and the Kalman filter is running, keeping the pendulum at upright position. If failing in this mission the state machine moves to state 3.

2 **Swinging.** The pendulum is swung back and forth to eventually reach upright position. This is achieved using position control of the cart, and an alternating set-point sequence. This state also handles the transfer to controlling in upright position (state 1).

3 **Wait state.** After a failed catch this state is used to wait until the pendulum stops rotating past the upright position. Then the state machine moves on to state 4.

4 **Wait state.** Before trying to swing up the pendulum again (state 2) one needs to know how high the pendulum is currently swinging in order to determine the right set-point amplitude for the cart position control. That is done in this state and then control is transfered to state 2.

5 **Anti clockwise turn.** Start state to make the pendulum do an anti-clockwise turn. The cart position is measured to know how to move the cart to make the pendulum fall in a controlled way. Move on to state 7.

6 **Clockwise turn.** Start state to make the pendulum do an anticlockwise turn. The cart position is measured to know how to move the cart to make the pendulum fall in a controlled way. Move on to state 7.

7 **Falling.** Make the pendulum fall either clockwise or anti clockwise by moving the cart, and then move on to state 8.

8 **Turning.** State to wait for the pendulum to reach upright position again. Then the state machine moves to state 1 in the same way as the transition from 2 to 1. If the pendulum due to some reason would not reach upright position again, there is a time out that eventually will force the state machine to state 3.

9 **Calibration.** The offset in the angular measurement varies with time. Since it is used to initialize the Kalman filter when catching the pendulum it is important to know it as well as possible. In this state it is measured when controlling the pendulum in upright position and then stored to be used at subsequent catches. This state should only be used when already in state 1. After one sample the state machine moves back to state 1.

10 **Manual start.** If the offset in the angular measurement has changed substantially it may be impossible to swing up the pendulum automatically. Then this state can be used to start it manually. Stay in state 0 and then switch to state 10 while holding the pendulum in upright position. Then use the calibration state 9 to make the automatic swinging work properly.

11 **Stop.** Set output to zero and stop program execution.

A full listing of the program is given in Appendix B.

8. How to use the program

To use the program do as follows:

1 Find a diskette with the program. Copy `pendulum.exe` to the hard disc.

2 Start the program `pendulum`.

3 Follow the instructions on the screen on how to connect D/A and A/D. When done hit `RETURN`. When doing this the pendulum must be hanging straight down, otherwise some internal variables in the program will be initialized incorrectly.

4 The state machine in the program will now be in state 0. You can change to any other state by typing the state number and hit `RETURN`. To swing up the pendulum type 4 and hit `RETURN`.

5 If the program does not catch the pendulum it will try again. If it, after several tries, still has not managed to catch the pendulum you should go

back to the idle state by typing 0. Then try a manual start by holding the pendulum in upright position and switch to state 10.

6 When the pendulum stands upright calibrate the system using state 9. This will help the program to succeed in catching the pendulum in subsequent tries.

7 Now you are ready for demonstrations. Make the pendulum do turns by using state 5 and 6. Drop the pendulum using state 0. Swing it up again using state 4.

Note that although you have the possibility to make the state machine go to any state by typing its number on the keyboard, you should **only** use states 0, 4, 5, 6, 9, and 10. The states 1, 2, 3, 7, and 8 are internal and should not be accessed manually. If you do use them, internal states in the state machine may be set erroneously, and the program has to be restarted.

9. References

BRÜCK, DAG (1988): "A Foreground/Background Real-Time Scheduler for the IBM AT," CODEN: LUTFD2/TFRT-7393, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

10. Appendix A: Simnon models

Here follows a listing of the Simnon code used when simulating the system. Both the linear and the nonlinear model are included.

```
continuous system pendulum

" Filename: pendulum.t
" Author: Kjell Gustafsson

" This is a nonlinear model of an inverted pendulum on a moving cart. There
" is no damping in the movements of the pendulum. The cart is controlled by
" a hydraulic cylinder. The input signal is a voltage to the hydraulic valve
" which results in a force. There is also a static friction force and a
" viscous damping in the cylinder.
"
" The position of the cart is limited. This is modelled as two large well-
" damped springs at zmax and zmin respectively.

input  u              " input voltage to hydraulic valve [V]

output zm             " output from position sensor [V]
output vm             " output from velocity sensor [V]
output thm            " output from angle sensor [V]
output stopsim        " tells when the pendulum has been catched

state  z              " cart position [m]
state  v              " cart velocity [m/s]
state  th             " pendulum angle [rad] (th=0  pendulum standing upright)
state  w              " pendulum angular velocity [rad/s]

der    dz dv dth dw


" dynamic equations

dz  = v
dv  = (m2*l*st*(-l*w*w + g*ct) + l*ftot)/den
dth = w
dw  = (st*(-m2*l*ct*w*w  + (m1+m2)*g) + ftot*ct)/den

den = l*(m1+m2*(1-ct*ct))
st = sin(th)
ct = cos(th)

" calculating input force

ftot = fh + fzsat     " hydraulic force + force due to limitation in position

fh = flin - ffric
flin = kf*u - kvisc*v
ffric = if abs(v)<eps then ffric0 else fstat*sign(v)
ffric0 = if abs(flin)<fstat then flin else fstat*sign(flin)
fstat = 0.7*kf        " the cart starts moving when  u=0.7

fzsat = if z<zmin then k*(zmin-z)-d*v else if z>zmax then k*(zmax-z)-d*v else 0
d = 2*sqrt(k*m1)

" form output variables

zm = z*kz               " measured position
vm = v*kv               " measured velocity
thm = th*kth            " measured angle

" is the pendulum caught or dropped

dropped = abs(th)>dropang
caught = (abs(th)<catchang) and (abs(w)<catchw) and (abs(v)<catchv)
stopsim = dropped or caught
```

```
dropang : 1.5
catchang : 0.015
catchw : 0.12
catchv : 0.02

m1 : 19                 " cart mass [kg]
m2 : 0.25               " pendulum mass [kg]
l  : 0.2                " pendulum length [m]
g  : 9.81               " gravity constant [m/s^2]
k  : 1e6                " spring constant in z saturation

kz  : 144               " position sensor conversion constant [V/m]
kv  : 23.2              " velocity sensor conversion constant [Vs/m]
kth : 14.0              " angle sensor conversion constant [V/rad]

kf    : 59.2            " gain from valve voltage to force [N/V]
kvisc : 1900            " viscous damping coefficient [N/m/s]

zmin : -0.075           " minimum cart position
zmax :  0.075           " maximum cart position

eps : 0.005
end
```

```
continuous system pendulum

" Filename: linpend.t
" Author: Kjell Gustafsson

" This is a linearized model of an inverted pendulum on a moving cart.
" There is no damping in the movements of the pendulum. The input signal
" is the cart velocity.
"
" The position of the cart is limited. This is modelled as two large well-
" damped springs at xmax and xmin respectively.
"

input u             " cart velocity

output vm           " measured velocity, differs from u due to
                    " limited cart position
output stopsim      " tells when the pendulum has been catched

state  thm          " pendulum angle measured in [V], state x1 in report
state  x2           " state x2 in report
state  zm           " cart position measured in [V], state x3 in report

der    dthm dx2 dzm

initial

thm = initth*kth
x2 = (kth*initw-p*kv*initv)/omega

sort

" dynamic equations

dzm  = q*vm
dthm = wm
dx2  = omega*thm

vm = if z<zmin then 0 else if z>zmax then 0 else u  " real cart velocity
wm = omega*x2 + p*vm                                " measured angular velocity

z = zm/kz             " real position [m]
v = vm/kv             " real velocity [m/s]
th = thm/kth          " real angle [rad]
w = wm/kth            " real angular velocity [rad/s]
```

```
" is the pendulum caught or dropped

dropped = abs(th)>dropang
caught = (abs(th)<catchang) and (abs(w)<catchw) and (abs(v)<catchv)
stopsim = dropped or caught

dropang : 1.5
catchang : 0.01
catchw : 0.08
catchv : 0.02

omega : 7.0             " parameter in linearized state space model
p : 3.0                 " parameter in linearized state space model
q : 6.2                 " parameter in linearized state space model

kz  : 144               " position sensor conversion constant [V/m]
kv  : 23.2              " velocity sensor conversion constant [Vs/m]
kth : 14.0              " angle sensor conversion constant [V/rad]

zmin : -0.075           " minimum cart position
zmax :  0.075           " maximum cart position

initth : 0              " initial pendulum angle [rad]
initw  : 0              " initial pendulum angular velocity [rad/s]
initv  : 0              " initial cart velocity [m/s]
end
```

```
discrete system regul

" Filename: stfkal.t
" Author: Kjell Gustafsson

" Linear state feedback controller with Kalman filter. Used to control the
" inverted pendulum.

time t
tsamp ts

input zm                " measured position [V]
input thm               " measured pendulum angle [V]
input vm                " measured cart velocity [V]

output u                " control signal, voltage to hydraulic valve

state  x1  x2  x4       " Kalman filter states, see report
new    nx1 nx2 nx4

initial

x1 = kth*initth
x2 = (kth*initw-p*kv*initv)/omega

sort

err = thm - x1 - x4                     " prediction error

" Kalman filter equations

nx1 = fi11*x1 + fi12*x2 + g1*vm + k1*err
nx2 = fi12*x1 + fi11*x2 + g2*vm + k2*err
nx4 = x4 + k3*err

" control signal calculation

utemp = - l1*x1 - l2*x2 - l3*zm - lvm*vm
u = if utemp>ulim then ulim else if utemp<-ulim then -ulim else utemp

fi11 = cosh(omega*h)
fi12 = sinh(omega*h)
```

```
g1 = p/omega*fi12
g2 = p/omega*(fi11-1)

ts = t + h

ulim :  10              " control signal saturation

h    : 0.0035           " sampling period

kth : 14.0              " angle sensor conversion constant [V/rad]
kv  : 23.2              " velocity sensor conversion constant [Vs/m]

l1  : 26.0              " state feedback gains
l2  : 55.1
l3  : -1.20
lvm : 1                 " feedback gain in inner loop

k1  : 0.139             " Kalman filter gains
k2  : 0.254
k3  : -0.0304

p     : 3.0             " parameter in linearized state space model
omega : 7.0             " parameter in linearized state space model

initth : 0              " initial pendulum angle [rad]
initw  : 0              " initial pendulum angular velocity [rad/s]
initv  : 0              " initial cart velocity [m/s]

end
```

```
discrete system regul

" Filename: lstfkal.t
" Author: Kjell Gustafsson

" Linear state feedback controller with Kalman filter. Used to control the
" inverted pendulum. Equivalent to the controller in 'stfkal.t' but changed
" to fit together with the linear model.

time t
tsamp ts

input zm                " measured position [V]
input thm               " measured pendulum angle [V]
input vm                " measured cart velocity [V]

output u                " control signal, voltage to hydraulic valve

state  x1  x2  x4       " Kalman filter states, see report
new    nx1 nx2 nx4

initial

x1 = kth*initth
x2 = (kth*initw-p*kv*initv)/omega

sort

err = thm - x1 - x4

" Kalman filter equations

nx1 = fi11*x1 + fi12*x2 + g1*vm + k1*err
nx2 = fi12*x1 + fi11*x2 + g2*vm + k2*err
nx4 = x4 + k3*err

" control signal calculation

utemp = (- l1*x1 - l2*x2 - l3*zm)*gain
u = if utemp>ulim then ulim else if utemp<-ulim then -ulim else utemp
```

```
fi11 = cosh(omega*h)
fi12 = sinh(omega*h)
g1 = p/omega*fi12
g2 = p/omega*(fi11-1)

ts = t + h

ulim :  10              " control signal saturation

h    : 0.0035           " sampling period

kth : 14.0              " angle sensor conversion constant [V/rad]
kv  : 23.2              " velocity sensor conversion constant [Vs/m]

l1  : 26.0              " state feedback gains
l2  : 55.1
l3  : -1.20
lvm : 1                 " feedback gain in inner loop
gain : 0.42             " gain from u to real velocity

k1  : 0.139             " Kalman filter gains
k2  : 0.254
k3  : -0.0304

p     : 3.0             " parameter in linearized state space model
omega : 7.0             " parameter in linearized state space model

initth : 0              " initial pendulum angle [rad]
initw  : 0              " initial pendulum angular velocity [rad/s]
initv  : 0              " initial cart velocity [m/s]

end
```

```
connecting system conn

" Filename: stfkalco.t
" Author: Kjell Gustafsson

u[pendulum] = u[regul]
thm[regul] = thm[pendulum]
zm[regul] = zm[pendulum]
vm[regul] = vm[pendulum]
u[cterm] = stopsim[pendulum]

end
```

---

11. Appendix B: Program listing

Here follows a listing of the program. For a listing of the scheduler see [Brück, 1988].

```
/*
 *   PENDULUM.C
 *
 *   Program to swing up and control the inverted pendulum in upright position
 *   Authors : Bo Bernhardsson and Kjell Gustafsson
 *   Date : 880525
 *   Major revision : 880826
 *
 *   The program does not use any routines from the math library since
 *   since in realtime they does not seem to work properly together with
 *   the io-routines. The construction abs((int) x) is often used instead
 *   of fabs(x).
 */


#include <stdio.h>
#include "fb.h"
#include <math.h>

/* A/D and D/A converter channels */

#define   CH_ANG   0     /* pendulum angle */
#define   CH_COL   1     /* color of code disc, i.e. what sector */
#define   CH_POS   2     /* cart position */
#define   CH_VEL   3     /* cart velocity */
#define   CH_PROC  0     /* which process is running, used for timing */
#define   CH_U     1     /* control signal */


/* Other compile-time constants */

#define   UHIGH          2047.0          /* A/D and D/A converter limits */
#define   ULOW           (-UHIGH)
#define   GRAY           512             /* separates white and black */
#define   H              0.0035          /* sampling period in seconds */
#define   TICKS          (H/0.0005)      /* schedular ticks per sample */
#define   VHIGH          (0.89*UHIGH)    /* largest angle measurement */
#define   VLOW           (-0.92*UHIGH)   /* smallest angle measurement */
#define   STARTOFFSET    (-0.13*UHIGH)   /* angle measurment offset */
#define   OMEGA          7.0             /* coefficient in state model */
#define   P              3.01            /* coefficient in state model */
#define   DOWNSCALE      (22.0/33.0)     /* attenuation corresponding to */
                                         /* external amplifier */
#define   SWITCHANGLE    (0.1*UHIGH)     /* switch to angle control */
#define   FAILANGLE      (0.4*UHIGH)     /* failed angle control */
#define   TWOSEC         (2.0/H)         /* two seconds */
#define   ESTPOLE        0.90            /* estimation pole */


/* Global data, parameters */

float   K;                      /* gain in position servo */
float   l1, l2, l3;             /* state feedback gains */
float   k1, k2, k3;             /* kalman filter gains */

float   refmin, deltaref;       /* position set-point when swinging */

float   currminangle;           /* absolute value of minimum angle so far */
float   offset;                 /* offset in angle measurement */

float   prevangle;              /* angle at previous sample */
float   x2est;                  /* estimation to initialize Kalman filter */
```

```
int     sector;                 /* sector number, 0..3 */
int     seccol[4];              /* color of sector */
int     nextsec[4][2];          /* to what sector are we moving */

int mode;                       /* current mode */
int nsamp;                      /* used for timing */

int clockwise;                  /* clockwise or anticlockwise turn */
float turnposref;               /* reference position when making */
                                /* pendulum rotate one turn */

float u;                        /* control signal */
float x1hat, x2hat, x4hat;      /* kalman filter states */
float fi11, fi12, g1, g2;       /* parameters in state space model */


/*
 * input()
 *
 * Read a value with prompt and range checking.
 *
 */


float input(prompt, min, max)
  char *prompt;
  float min, max;
{
  float x;

again:
  printf("%s:   ", prompt);
  scanf("%f", &x);
  if (x < min || x > max) {
    printf("Value out of range (%f..%f)\n", min, max);
    goto again;
  }

  return x;
}



/*
 * init_param()
 *
 * Initialize parameters and controller coefficients
 *
 */

void init_param()
{
  /* position servo gain, state feedback gains, Kalman filter gains */

  K = 2;
  l1 = 26.0;
  l2 = 55.1;
  l3 = -1.20;
  k1 = 0.139;
  k2 = 0.254;
  k3 = -0.0304;

  /* position set-points when swinging */

  refmin = 90;
  deltaref = 2500;

  /* plant model parameters */
```

```
  fi11=cosh(OMEGA*H);
  fi12=sinh(OMEGA*H);
  g1=P/OMEGA*sinh(OMEGA*H);
  g2=P/OMEGA*(cosh(OMEGA*H)-1);

  offset = STARTOFFSET;
}



/*
 * init_sectorlogic()
 *
 * Initialize the sector logic
 *
 */

void init_sectorlogic()
{
  /* when the program starts the pendulum is hanging down, i.e. sector 0 */

  sector = 0;

  /* on the code disc sector 0 and 2 are black and sector 1 and 3 are white */

  seccol[0] = 1;
  seccol[1] = 0;
  seccol[2] = 1;
  seccol[3] = 0;

  /* sector number (first index) and angle sign (second index) */
  /* determine the next sector */

  nextsec[0][0] = 3;
  nextsec[1][0] = 2;
  nextsec[2][0] = 1;
  nextsec[3][0] = 0;
  nextsec[0][1] = 1;
  nextsec[1][1] = 0;
  nextsec[2][1] = 3;
  nextsec[3][1] = 2;
}



/*
 * foreground()
 *
 * This is the controller. It is implemented as a finite state machine
 * with the variable 'mode' determining the state.
 *
 */

#pragma check_stack-

void foreground()
{
  float ref;              /* current reference value */
  float sensorangle;      /* angle from sensor, corresponds to +-45 deg */
  float angle;            /* angle recalculated corresponding to +-180 deg */
  float correctangle;     /* angle +-180 deg, corrected with offset */
  float err;              /* Kalman filter residual */
  float v;                /* cart velocity */
  float temp;             /* temporary variable */

  int color;              /* color on code disc */
  int sign;               /* sign of sensorangle */


  /* output control signal calculated last sample */
```

```
DAout(CH_U, (int) u);


/* read sensorangle and color of code disc */

if (ADin(CH_COL)>GRAY)
  color = 1;                    /* black */
else
  color = 0;                    /* white */

sensorangle = ADin(CH_ANG);

if (sensorangle>0)
  sign = 1;
else
  sign = 0;


/* moving to new sector ? */

if (seccol[sector] != color)
  sector = nextsec[sector][sign];


/* convert sensorangle to range +-180 degrees */

switch (sector) {

  case 0 :
    /* down */
    if (sensorangle>offset)
      angle = 2*(VLOW - VHIGH) + sensorangle;
    else
      angle = 2*(VHIGH - VLOW) + sensorangle;
    break;

  case 1 :
    /* left */
    angle = 2*VLOW - sensorangle;
    break;

  case 2 :
    /* up */
    angle = sensorangle;
    break;

  case 3 :
    /* right */
    angle = 2*VHIGH - sensorangle;
    break;
}
correctangle = angle - offset;


/* finite state machine */

switch (mode) {

  case 0 :
    /* idle */
    currminangle = 4*UHIGH;

    /* cart position control at center of track */
    u = DOWNSCALE*K*(0.0 - ADin(CH_POS));
    prevangle = 0;
    break;

  case 1 :
    /* controlling */
    v = ADin(CH_VEL);
```

```
  /* Kalman filter */
  err = angle - x1hat - x4hat;
  temp  = fi11*x1hat + fi12*x2hat + g1*v + k1*err;
  x2hat = fi12*x1hat + fi11*x2hat + g2*v + k2*err;
  x1hat = temp;
  x4hat = x4hat + k3*err;

  /* state feedback and inner feedback loop */
  u = -DOWNSCALE*(l1*x1hat + l2*x2hat + l3*ADin(CH_POS) + v);

  /* have we dropped the pendulum? */
  if (abs((int) x1hat)>FAILANGLE)
    mode = 3;
  break;

case 2 :
  /* swinging */

  /* calculate set-point for cart position  */
  if (abs((int) correctangle)<currminangle)
    currminangle = abs((int) correctangle);
  temp = currminangle/(4*UHIGH);
  ref = refmin+temp*temp*(1.5-temp)*deltaref;
  if (correctangle<0)
    ref = -ref;

  u = DOWNSCALE*K*(ref - ADin(CH_POS));

  /* estimate angular velocity to be able to initialize Kalman filter */
  if (sector==2)
    x2est = ESTPOLE*x2est+(1.0-ESTPOLE)*(angle-prevangle)/H;
  else
    x2est = 0.0;

  /* time to try to catch the pendulum? */
  if (abs((int) currminangle)<SWITCHANGLE) {
    mode = 1;
    x1hat = correctangle;
    x2hat = (x2est - P*ADin(CH_VEL))/OMEGA;
    x4hat = offset;
  }

  prevangle = angle;
  break;

case 3 :
  /* wait state after failed catch */

  /* cart position control at center of track */
  u = DOWNSCALE*K*(0.0 - ADin(CH_POS));

  nsamp = nsamp+1;

  /* reset timer each time the pendulum passes upright position */
  if (abs((int) correctangle)<SWITCHANGLE)
    nsamp = 0;

  /* pendulum no longer rotating, just swinging? */
  if (nsamp>TWOSEC) {
    nsamp = 0;
    mode = 4;
    currminangle = 4*UHIGH;
  }
  break;

case 4 :
  /* estimate currminangle */

  /* cart position control at center of track */
  u = DOWNSCALE*K*(0.0 - ADin(CH_POS));
```

```
  nsamp = nsamp+1;

  if (abs((int) correctangle)<currminangle)
    currminangle = abs((int) correctangle);

  if (nsamp>TWOSEC) {
    nsamp = 0;
    mode = 2;
    currminangle = 1.1*currminangle;
  }
  break;

case 5 :
  /* transition state to get position before anti clockwise turn */
  turnposref = ADin(CH_POS);
  mode = 7;
  clockwise = 0;
  break;

case 6 :
  /* transition state to get position before clockwise turn */
  turnposref = ADin(CH_POS);
  mode = 7;
  clockwise = 1;
  break;

case 7 :
  /* make pendulum fall  */
  if (clockwise==1)
    u = DOWNSCALE*K*(turnposref + 200 - ADin(CH_POS));
  else
    u = DOWNSCALE*K*(turnposref - 220 - ADin(CH_POS));

  /* have the pendulum fallen yet? */
  if (abs((int) correctangle)>FAILANGLE)
    mode = 8;
  break;

case 8 :
  /* turn */

  /* cart position control at position determined by pendulum angle */
  if (clockwise==1)
    if (correctangle>0)
      u = DOWNSCALE*K*(turnposref + 200 - ADin(CH_POS));
    else
      u = DOWNSCALE*K*(turnposref - ADin(CH_POS));
  else
    if (correctangle>0)
      u = DOWNSCALE*K*(turnposref - ADin(CH_POS));
    else
      u = DOWNSCALE*K*(turnposref - 220 - ADin(CH_POS));

  /* estimate angular velocity to be able to initialize Kalman filter */
  if (sector==2)
    x2est = ESTPOLE*x2est+(1.0-ESTPOLE)*(angle-prevangle)/H;
  else
    x2est = 0.0;

  /* time to try to catch the pendulum? */
  if (abs((int) correctangle)<SWITCHANGLE) {
    mode = 1;
    x1hat = correctangle;
    x2hat = (x2est - P*ADin(CH_VEL))/OMEGA;
    x4hat = offset;
    nsamp = 0;
  }

  /* time out, did the turn fail? */
  nsamp = nsamp+1;
  if (nsamp>TWOSEC) {
```

```
        mode = 3;
        nsamp = 0;
      }

      prevangle = angle;
      break;

    case 9 :
      /* calibration state, one sample */
      offset = x4hat;
      mode = 1;
      break;

    case 10 :
      /* manual start */
      mode = 1;
      x1hat = angle - offset;
      x2hat = 0;
      x4hat = offset;
      break;

    case 11 :
      /* stopping */
      u = 0;
      break;
  }

  /* limit control signal */

  if (u < ULOW)
    u = ULOW;
  else if (u > UHIGH)
    u = UHIGH;

  /* output signal to be able to time the controller, the background */
  /* process outputs zero at the same channel */

  DAout(CH_PROC, 1000);
}

#pragma check_stack+


/*
 * main()
 *
 * Main program.
 *
 */


main()
{
  printf("*********************************\n");
  printf("*      BoB's and Kjell's        *\n");
  printf("*      PENDULUM-SWINGER         *\n");
  printf("*    May 1988, August 1988      *\n");
  printf("*                               *\n");
  printf("*********************************\n");
  printf("\n\n");
  printf("Connect the A/D and D/A as follows:\n");
  printf("Inputs: A/D 0 - pendulum angle\n");
  printf("        A/D 1 - code disc\n");
  printf("        A/D 2 - cart position\n");
  printf("        A/D 3 - cart velocity\n");
  printf("Output: D/A 1 - control signal\n\n");
  printf("Hit return when all connections are done.");

  while (getchar() != '\n')
    /* wait */;
```

```
  printf("\n\nOptions:\n");
  printf("Mode  0 - Idle                    \n");
  printf("Mode  1 - Controlling             \n");
  printf("Mode  2 - Swinging                \n");
  printf("Mode  3 - Prepare swinging (1)    \n");
  printf("Mode  4 - Prepare swinging (2)    \n");
  printf("Mode  5 - Anti clockwise turn     \n");
  printf("Mode  6 - Clockwise turn          \n");
  printf("Mode  7 - Make pendulum fall      \n");
  printf("Mode  8 - Catch turning pendulum \n");
  printf("Mode  9 - Calibrate               \n");
  printf("Mode 10 - Manual start            \n");
  printf("Mode 11 - Stop                   \n\n");
  printf("During normal operation use only mode 0, 3, 5, 6, 9, 10, and 11\n\n");

  /* Initialize */

  u = 0;
  init_param();
  init_sectorlogic();
  nsamp = 0;
  mode = 0;

  schedule(foreground, (int) TICKS);

  while (mode != 11) {
    mode = (int) input("Mode", 0.0, 11.0);
  };

  reset();
}
```