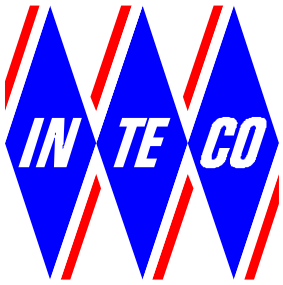
PENDULUM-CART SYSTEM

MATLAB 6.5 and 7 (R14 SP2) version

User's Manual



www.inteco.com.pl

Contents

1. Introduction

In this manual you will find the following sections: modelling and control problems, identification, PID control and Rule-based control. When you use the manual for the first time, you can start experiments from the Pendulum Control Window. However, before exploring the examples, you need to know if the system model is compatible with reality. The default parameters are stored in the model but it cannot be guaranteed that they are well identified. It is recommended to pass through the identification procedure.

In this manual you find the model of the cart-pendulum system. It contains several dedicated blocks added to obtain the best compatibility with the real laboratory system. The model is equipped with the set of default parameter values. The parameters can be divided into three groups: constant (not to be identified), well identified and poorly identified. For example, in every produced set the pendulum and the cart are identical. Their masses are thus assumed to be constant parameters. A different case is with the moment of inertia of the rotating pendulum. It is slightly different in different sets. The reason is simple: the pendulum is assembled by the user in his laboratory and so small differences in dimensions of the assembled set may occur. Unfortunately, in a real system there usually is a group of poorly identified parameters, characterised by random behaviour. In the pendulum-cart system, parameters related to the cart friction belong to this group.

The identification procedures generate experiments to obtain friction parameters to be used in the *Friction* block of the model. We must note that the cart friction function vs. cart velocity is an approximation of the real friction force. It is not sufficient to identify it once. It must be verified and usually tuned by signals exciting the simulation model and the real system. Verification and tuning is an obligatory step at the end of identification. If time responses of the model and the real system are close to each other, as far as shape and amplitude of signals are concerned, then we can trust in our modelling.

The Pendulum-Cart System consists of:

- Pentium or AMD based personal computer equipped with RTDAC4/PCI I/O board,
- pendulum and cart mechanical unit,
- power interface,
- control software

1.1. Hardware requirements

The following basic configuration of your PC is required:

- RTDAC4/PCIc data acquisition board for controlling the system. The board contains FPGA equipped with dedicated logic design;
- pendulum and cart set-up;
- power interface and wiring allowing electrical connections to the pendulum set.

1.2. Software requirements

For development of the project and automatic building of the real-time program it is required that the following software is properly installed on the PC:

- MS Windows 98/NT 4.0 (with Service Pack 5) / 2000 or Windows XP.
- MATLAB version 6.5 or 7 (R14 SP2) with Simulink, RTW and RTWT toolboxes (not included),
- Real Time Workshop to generate the code.
- Real Time Windows Target toolbox.
- 32 bit compiler MS Visual C++ if Matlab 6.5 version is used.

The Pendulum1 toolbox supports Matlab 6.5 and Matlab 7 (R14 SP2). MS Visual C++ compiler is requested for Matlab 6.5. If Matlab 7 is used the built-in Open Watcom compiler is applied.

- Device drivers to handle communication with the RTDAC4/PCIc data acquisition board. The Pendulum toolbox includes specialized drivers for the Pendulum-Cart System,

1.3. Hardware installation

Hardware installation is described in *Assemling and Installatiojn* manual.

1.4. Software installation

One must insert the installation CD and proceed step by step following displayed commands.

2. System description

One of the simplest problems in robotics is that of controlling the position of a single link using a steering force applied at the end. Pole-balancing systems are impressive demonstration models of missile stabilisation problems. The crane used at shipping ports is an another example of nonlinear electromechanical systems having a complex dynamic behaviour and creating challenging control problems. Mathematically both are just a pendulum in stable or unstable position.

The pendulum-cart setup consists of a pole mounted on a cart in such a way that the pole can swing free only in the vertical plane. The cart is driven by a DC flat motor. To swing and to balance the pole the cart is pushed back and forth on a rail of limited length.

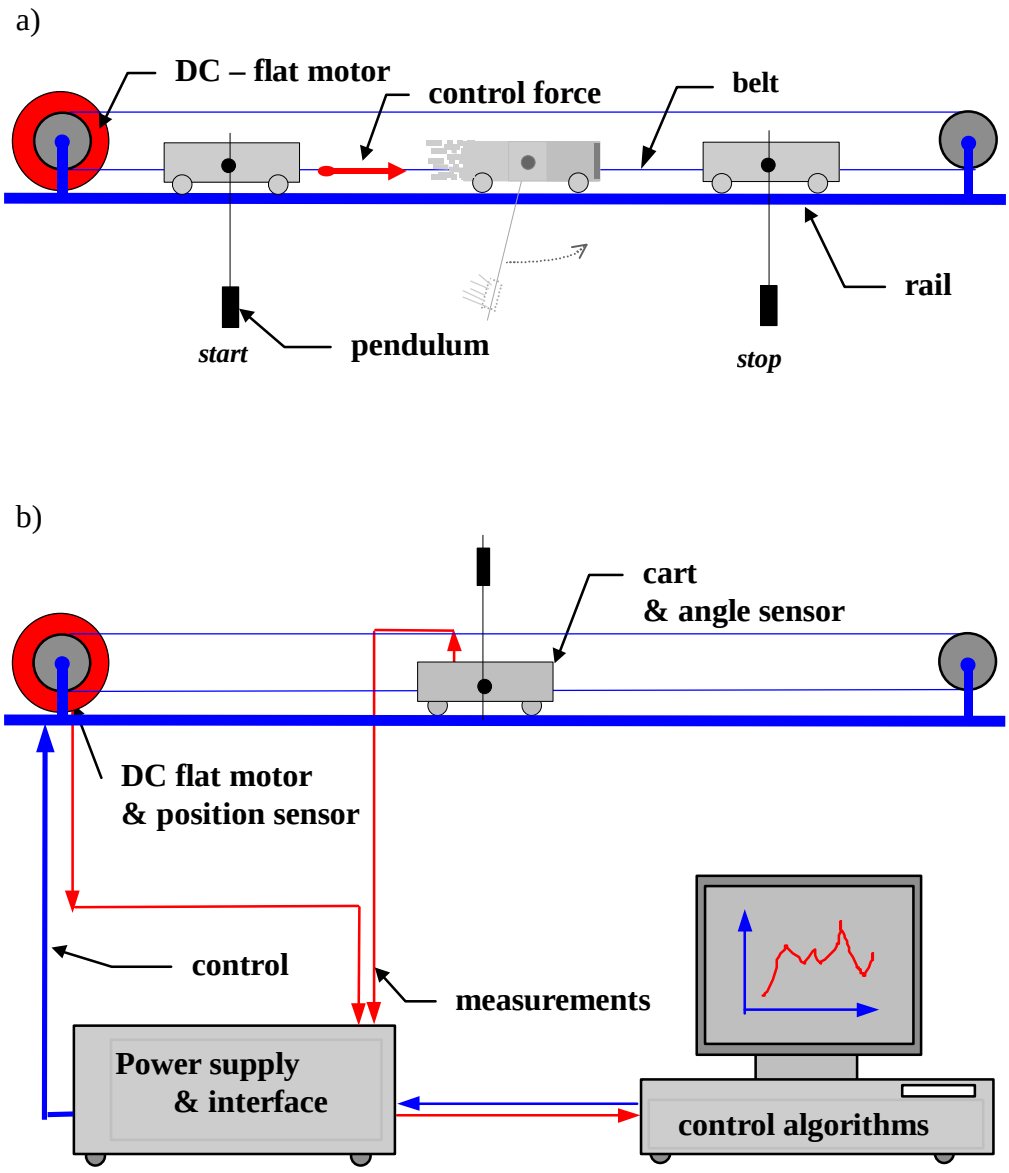


Fig. 2.1 a) The pendulum-cart setup, b) pendulum control system.

The vertical stationary positions of the pendulum (upright and down) are equilibrium positions when no force is being applied. In the upright position a small deviation from it results in an unstable motion. Generally the pendulum control problem is to bring the pole to one of the equilibrium positions. Preferably to do so as fast as possible, with few oscillations, and without letting the angle and velocity become too large. After the desired position is reached, we would like to keep the system in this state despite of random perturbations. Manual control of the cart-pole system is possible only for simple tasks e.g. for moving the cart from one place on the rail to another. For more complicated tasks (like stabilisation of the pole in the upper position) a feedback control system must be implemented (Fig. 2.1).

The purpose of the inverted pendulum control algorithm is to apply a sequence of forces of constrained magnitude to the cart such that the pole starts to swing with an increasing amplitude and the cart does not override the ends of the rail. The pole is swung up to achieve a vicinity of its upright position. Once this has been accomplish, the controller is maintaining the pole vertical and is bringing the cart back to the centre of the rail. Therefore two independent control algorithms are implemented for this purpose:

SWINGING ALGORITHM, AND STABILISING ALGORITHM.

Only one of two control algorithms is active in each control zone. These zones are shown in Fig. 2.2.

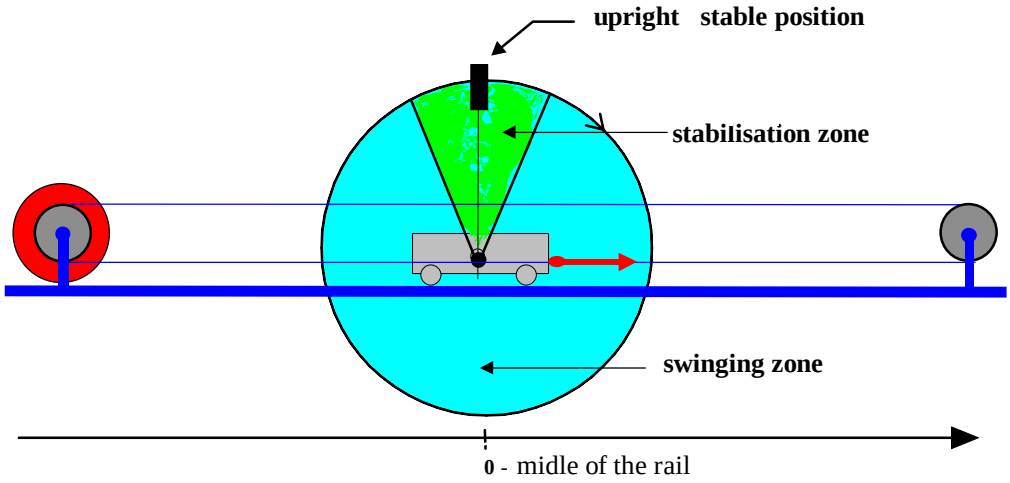


Fig. 2.2 Activity zones of two control algorithms

The swinging control algorithm is a heuristic one, based on energy rules. The algorithm steers the pole up increasing its total energy. There is a trade-off between two tasks: to swing the pendulum to the upright position and to centre the cart on the rail. Due to the presence of disturbances and parameter uncertainties a robust behaviour is more important than the optimal character of control strategy. The switching moments are calculated according to a simple rule. The characteristic feature of control is its “bang-bang” character. Swinging up the pole may result in overdriving the upper unstable equilibrium point. To achieve a “soft” landing in the vicinity of the upright position (“stabilising zone” in Fig. 2.2) a routine called the “soft landing arbiter” checks whether the kinetic energy of pole minus energy loss due to friction forces is sufficient to rise up the

centre of gravity of the pole to its upright position. If the condition is satisfied then control is set to zero. Now the "bang-bang" character of control is finished. After the pole has entered the stabilisation zone the system can be treated as linear. The control is switched to the stabilising algorithm. Due to the limited length of the rail a routine called “length control” is introduced to reinforce centring of the cart and prevent overrunning the edges of the rail. The rule is very simple. When the position given by the parameter "length" is reached, then the maximal force is applied to the cart steering it back away from this position.

3. Computer control

A block diagram of a computer-controlled process is given in Fig. 3.1.

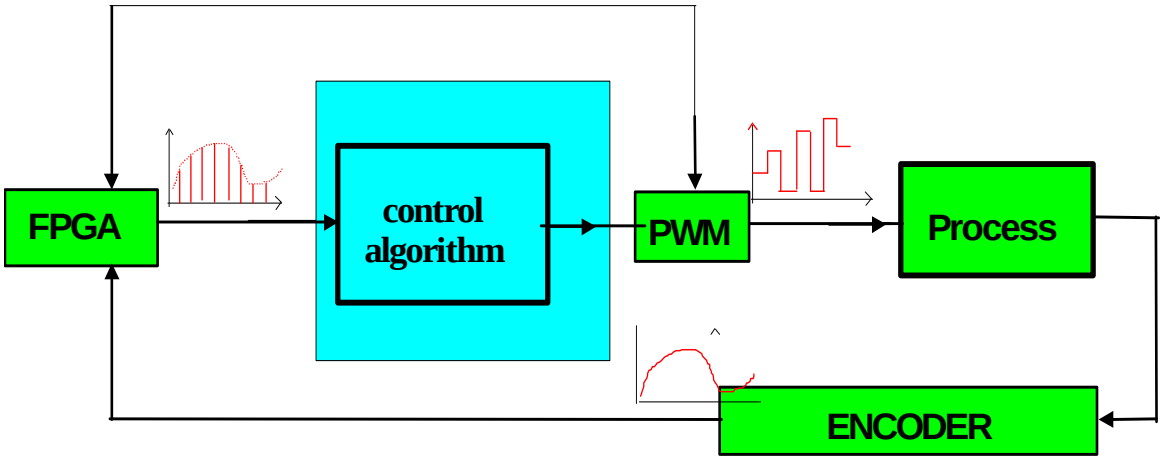


Fig. 3.1 Computer controlled process.

The system contains six blocks: the process, sensors (ENCODER), FPGA logic and PWM generator, control algorithm, and a clock. The operation of the converters and of the control algorithm is controlled by the software clock. The time between successive conversions of the signal to the digital form is called the sampling period (T_0). The clock supplies a pulse every T_0 seconds, and the PWM converter sends a signal to the computer every time an timer event arrives. The control algorithm computes the value of control variable and sends it as number to the FPGA logic. It is assumed that the FPGA holds the signal constant over the sampling period. Periodic sampling is normally used. It is possible to use different sampling periods.

An application of the general digital control system schema for pendulum control is given in a block diagram form in Fig. 3.2. Two process states are measured: the cart position x_1 and the pendulum angle x_2. Process states are measured as continuous signals and digitalized by optical encoders (sensors S_1 S_2). The reference input (desired value of the cart position x_1^d) can be generated in a digital form using an desired position generator. The software timer is used to supply interrupts for the system: The basic clock activates periodic sampling of optical decoders outputs, synchronises the computation of controller outputs (u) and periodic PWM generation.

The pendulum-cart system is controlled in real-time. The term "real-time" is often used but seldom defined. One possible definition is [3]:

"Real-time is the operating mode of a computer system in which the programs for the processing of data arriving from the outside are permanently ready, so that their results will be available within predetermined periods of time; the arrival times of the data can be randomly distributed or be already determined depending on the different applications."

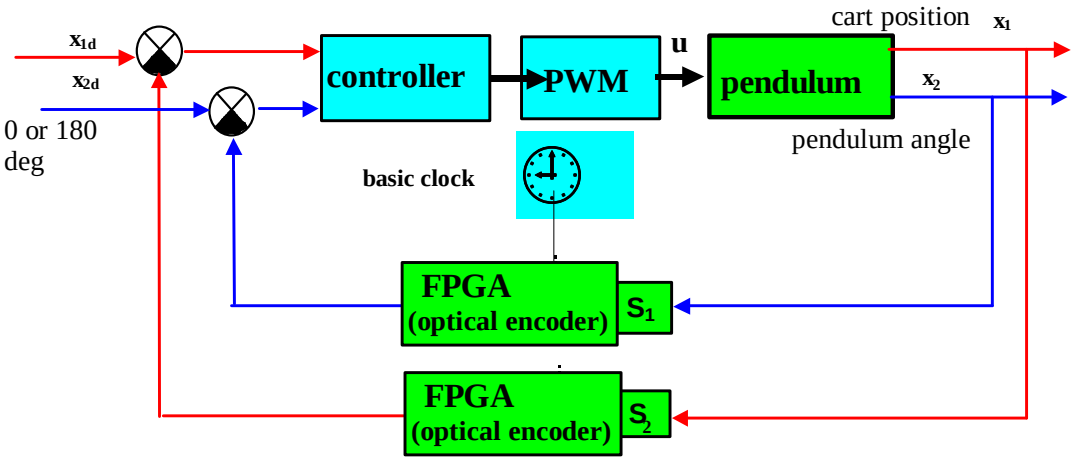


Fig. 3.2 Digital control of the pendulum-cart system (basic block diagram).

Through the board the PC reads the position of cart and the angle of pendulum in a digitalized form and also controls the DC motor. The sensors attached to the pendulum-cart setup, in combination with appropriate FPGA blocks, detect the angle of the pendulum and the position of the cart. The pendulum-cart setup utilises two optical encoders as angle and position detectors (Fig. 3.3).

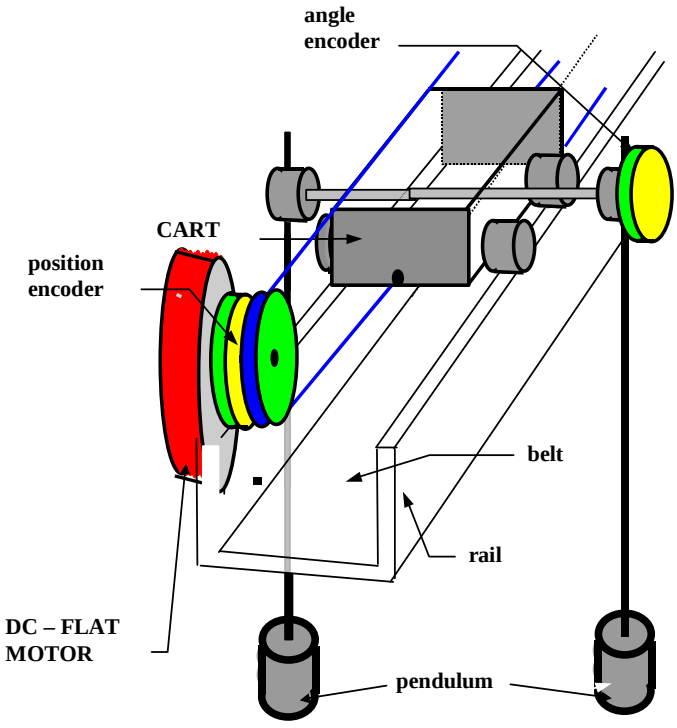


Fig. 3.3 Mechanical part and sensors of pendulum-cart setup

The first one is installed on the pendulum axis, the second on the DC motor axis. An optical encoder basically consists of a light source for emitting light, a light-receiving device, and a rotary code disk with slits (Fig. 3.4).

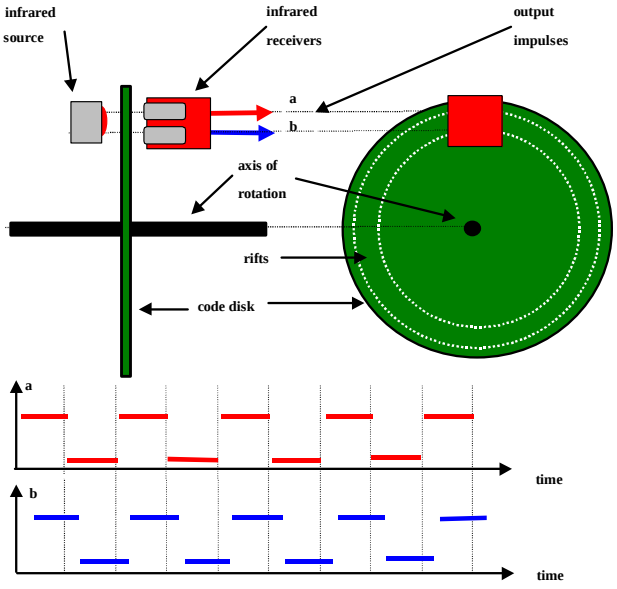


Fig. 3.4 Sensor of the position

The rotary disc is placed between the light source and the light-receiving device. The optical encoder obtains the number of pulses proportional to the angle of rotation of the disk. Accumulated output pulses of the encoder give the value of the rotation position of the disk by using a counter. The pendulum-cart setup uses incremental encoders. illustrates the principle of determining the direction of rotation for an incremental optical encoder. The light beams emitted by two light sources (A and B) go through two rings of slits on the disc. The slits have a phase difference, so that the electric outputs of the receivers (A and B) are rectangular waves with some phase difference. The sign of the phase difference allows determining the direction of rotation.

The following point should be noted when using an incremental encoder. After each experiment or power switching one should move the cart to the centre of the rail before starting the next experiment. This operation will guarantee setting zero values of the position.

The control signal flows from the computer through the PWM generator of the data acquisition board. PWM generates the control signal for the DC flat motor.

4. Algorithms of control – general remarks

The pendulum-cart system can illustrate several complex and nontrivial problems of control theory. The following characterisation is essential for this system:

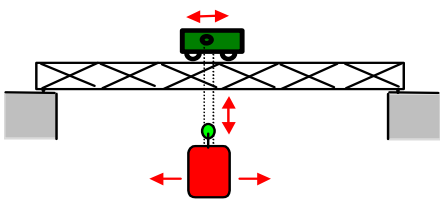
- four state variables and one control variable
- the dynamics is described by one ordinary differential equation (ode) of the fourth order or by four ordinary differential equations of the first order.
- there is an infinite number of equilibrium points for the uncontrolled system; there is also an infinite number of equilibrium points for the system with linear feedback.
- the system is non-linear
- the control is bounded
- the length of the rail is limited and the cart position is bounded

For our practicals we choose three equilibrium points:

- The first point is stable - the cart at a point close to the left end of the rail and the pendulum in its down position
- The second point is also stable - the cart at a point close to the right end of the rail and the pendulum in its down position
- The third point is unstable - the cart at the centre of the rail and the pendulum in its upright position.

There are two essential components of virtually any controller: a mechanism for knowledge representation and a mechanism for decision-making. With conventional controllers knowledge is represented by a mathematical model (such as a set of differential equations) and decision-making is based on optimisation.
When the system is complex or less precisely known it becomes difficult to characterise the knowledge adequately with conventional means. One must find alternative ways to encode the available knowledge about the system. It can result in the development of intelligent controllers. However, one can also build a control algorithm without any model. This is called a „black box" technique.
The closed-loop responses with different controllers are more or less sensitive to variation of parameters of the model. For example, the time-optimal controller requires a very accurately identified model. The LQ and PID controllers are less sensitive. In most cases, however, precise identification of dynamics of the laboratory model is essential for the generation of an appropriate control algorithm.

Three common problems of pendulum control are presented in Fig. 4.1:



crane control (C)

swing-up control (SU) and self erecting control (SE)

Fig. 4.1 Control problems

A human or a computer may operate as a controller. A human operator may be fairly successful in the (C) problem, but completely fails in the next two problems. The SU and SE require a fast operating controller. Only a computer can manage such fast and complex tasks.
In this section the brief review of computer control algorithms is given. The detailed description of the rule-based controller is given in section 8.

The following control algorithms can be introduced:

- rule-based (SU)
- LQ (C and SE)
- PID (C)
- fuzzy(C)
- time-optimal (SU and C)

Remark. The user can take advantage of the flexibility of the structure to create his own algorithms. He is only limited to a number of input/output signals defined in a given LQ, PID or Fuzzy structure.

4.1. Rule-based control

The simple swinging up formula has the form:

$$u = \text{sign}[x_4(|x_2| - \pi/2)]$$

The variables are defined in section 7. It results in the normalised ±1 value of the control u. The control value is proportionally decreased by the parameter *Max.* taken from the range [0, 1]. To avoid rotation of the pole the total energy of the pole is calculated in each numerical step. If this energy is sufficient to steer the pole to its upright position then the control is set to zero. If the parameter T_s is greater then 0 then the compensation of static friction is introduced:

$$u = u_{computed} + \text{sign}(u_{computed})\,T_s$$

Then the limit of the cart position is checked. If the cart reaches this limit then the maximum value of control (e.g. ±1) is generated with the sign opposite to the sign of cart position.

4.2. LQ control

The linear-quadratic controller has the form:

$$u = -(K_1\varepsilon_1 + K_2\varepsilon_2 + K_3\varepsilon_3 + K_4\varepsilon_4)$$

where:

ε_1 = desired position of the cart – measured position of the cart,

ε_2 = desired angel of the pendulum – measured angel of the pendulum,

ε_3 = desired velocity of the cart – observed velocity of the cart,

ε_4 = desired angular velocity of the pendulum – observed angular velocity of the pendulum.

$K_1,...,K_4$ are positive constants.

The optimal feedback gain vector $K = [K_1,...,K_4]$ is calculated to minimize the quadratic cost function.

4.3. PID control

The PID controller has the form:

$$u = K_p\varepsilon_1(k) + K_i\sum_{k=0}^{n}\varepsilon_1(k) + K_d\left[\varepsilon_1(k) - \varepsilon_1(k-1)\right]$$

where:

ε_1 = desired position of the cart - measured position of the cart

K_p is gain coefficient

K_i is integration gain

K_d is derivative gain

5. Pendulum Control Window

The Pendulum-Crane system is an “open” type. That means that a user can design and solve any pendulum control problem on the base of presented hardware and software. Software includes device drivers compatible with RTWT MathWorks toolbox. It is assumed that a user is familiarised with MATLAB tools especially with RTWT toolbox. Therefore we do not include the detailed description of this tool.

The user has a rapid access to all basic functions of the Pendulum-Cart System from the *Pendulum Control Window*. It includes: identification, drivers, simulation model and application example.

In the Matlab Command Window type

p1

and the *Pendulum Control Window* shown in Fig. 5.1 appears:

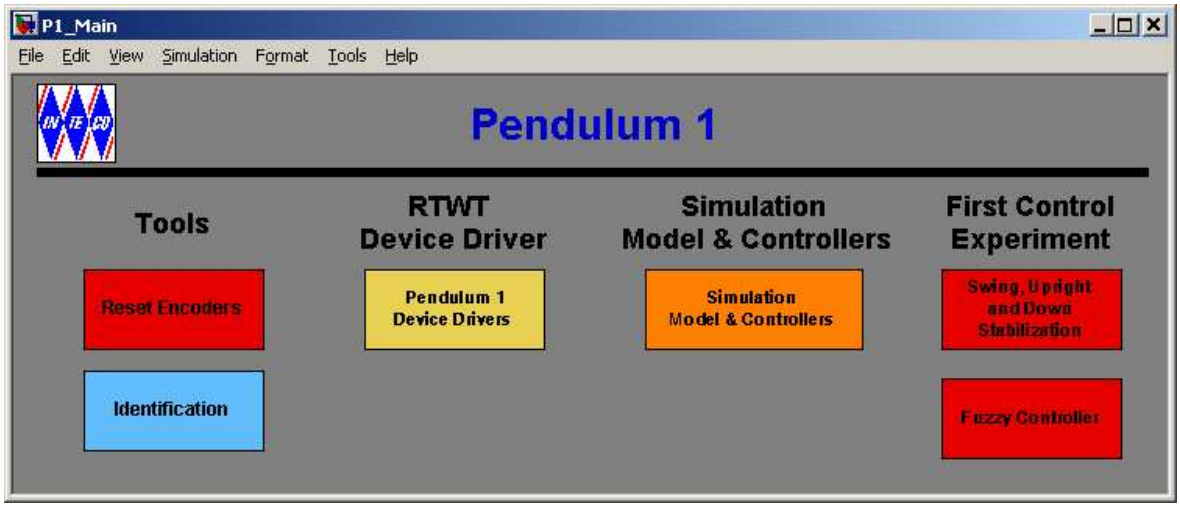


Fig. 5.1 *Pendulum Control Window*

The respective buttons in the TOOLS column perform the following tasks:

Reset Encoders - set two measurement encoders to zero.

☞	**Remember to reset encoders bringing the cart to the left end of the rail.**

Identification - opens the window from which you can perform system identification.

The main driver is located in the *RTWT Device Driver* column. The driver is a software go-between for the real time MATLAB environment and the RT-DAC/PCI acquisition board.

This driver serves to control and measure signals. Click the *Pendulum 1Device Drivers* button and the driver window opens (Fig. 5.2).

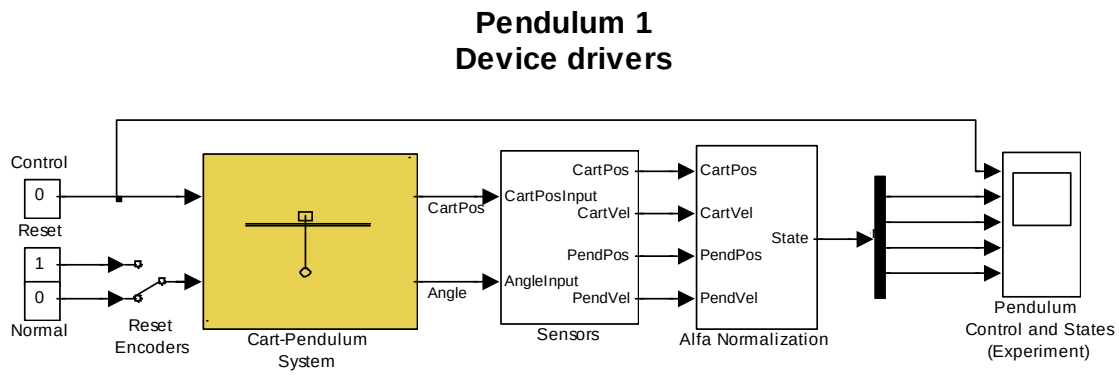


Fig. 5.2 Pendulum System Dedicated Driver

The proper driver is included in the *Cart-Pendulum System* block. The *Sensors* block calculates velocities of the pendulum and cart. The next block normalises the pendulum angle. The driver has two inputs: PWM - DC motor control and *Reset Encoders*. There are 4 outputs of the driver: *Cart Position, Cart Velocity, Pendulum Position and Pendulum Velocity.*

To build your own application you can copy this driver to a new model.

	**Do not make any changes inside the original driver. They can be done only inside its copy!!!**

The simulation model of the system is located in the *Simulation Model & Controllers* column. Click the *Pendulum Simulation Model* button and the model window opens (Fig. 5.3)

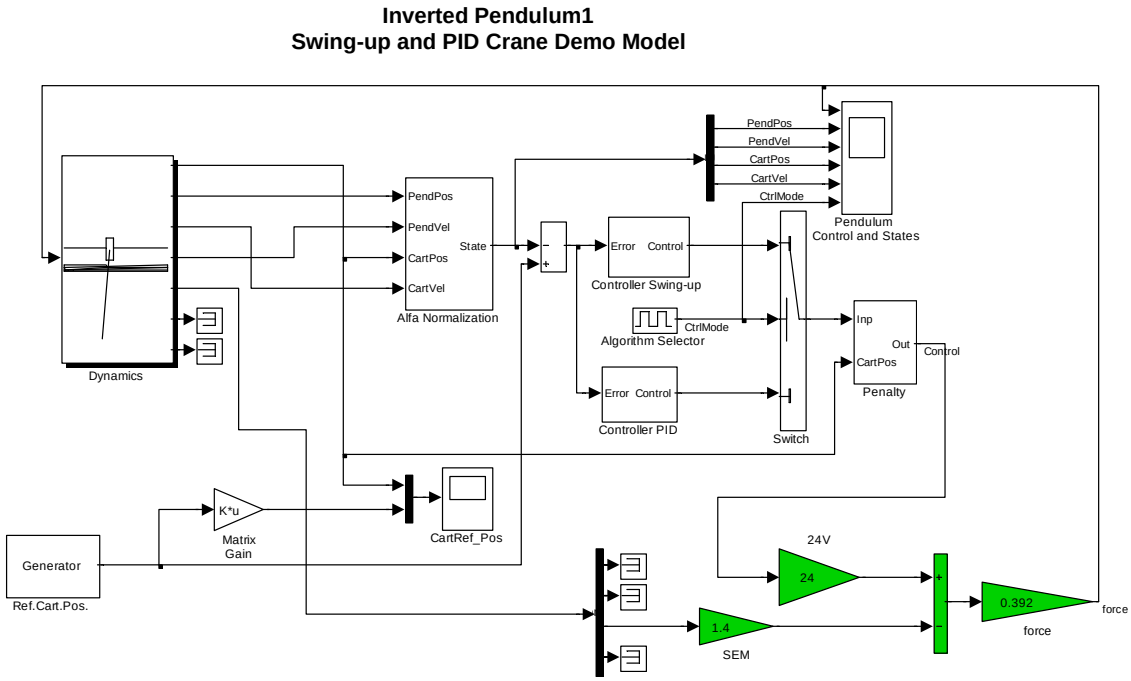


Fig. 5.3 Demo simulation model

This model is ready to simulate the swing-up task or the PID control in the crane mode. Dynamics of the pendulum-cart system is included in the *Dynamics* block. The details of the *Dynamics* block you can find in section 9.

Swing, Upright and Down Stabilisation button opens the Simulink model to perform real-time execution.

The *Fuzzy Controller* button allows to perform simulation and real-time experiment using fuzzy controller.

6. Starting and stopping procedures

	**Remember do not work in a directory belonging to MATLABPath. MATLAB protects its own directories and you are not permitted to build a model there.**

The Pendulum-cart system works in the RTWT mode. Due to this fact before starting the first demo in the real-time mode you have to check if your MATLAB configuration and compiler installation is properly defined.

	**Build and run an example of real-time application. Real-time Windows Target includes the model rtvdp.mdl.** **In the MATLAB Command Window, type** **rtvdp** **Next, build and run the real-time model.** **For details refer to the Real-Time Windows Target help, section Installation and Configuration.**

6.1. Starting procedure

If the system is properly mounted and wired to PC and power supply we are ready to start demos. There are the default settings in the model. You can apply the demo using these settings. How to change values of the settings will be described in section 8. The demo swings up the pendulum, stabilise it in the upright position and keeps the cart in the centre of the rail. After several seconds the pendulum goes down and the algorithm of down stabilisation begins. These two actions are repeated forever.

In the Pendulum Control Window click *Swing, Upright and Down Stabilisation* button and the model shown in Fig. 6.1 opens.

Build the model and after success perform the following steps:

- Bring the cart to the left end of the rail. The pendulum has to hang down motionless.
- Double click the *Reset Encoders* switch.
- Bring back the cart to the centre of the rail and click *Simulation/Connect to target* menu option and next click *Start real-time code*.

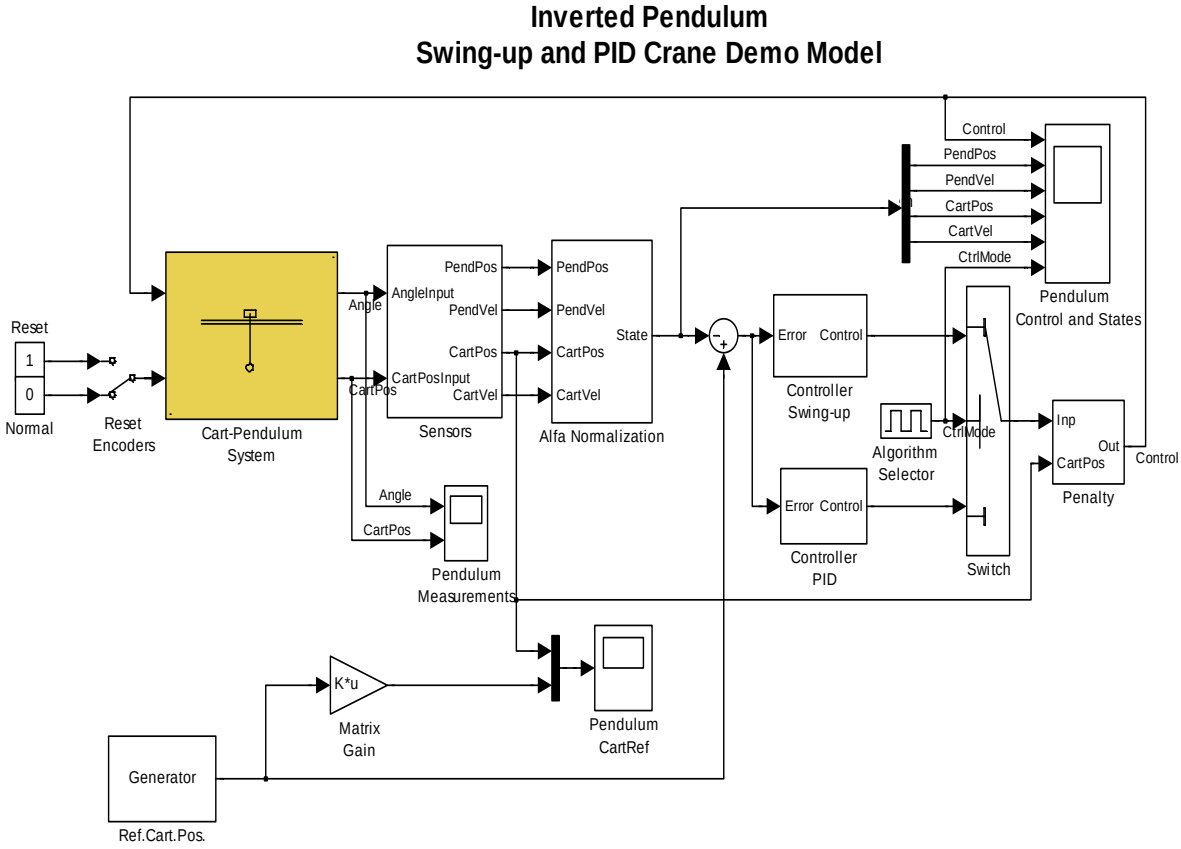


Fig. 6.1 Swing-up and PID crane demo model for an experiment in real time

To stop experiment click the *Simulation/StopRealTimeCode* menu option.

Fig. 6.2 presents a typical control experiment in real time. One can notice two modes of operation. The pendulum is swung-up and after is kept stabilized in the upright position. The cart moves back and forth on the rail. Finally it is maintained in the middle of the rail.

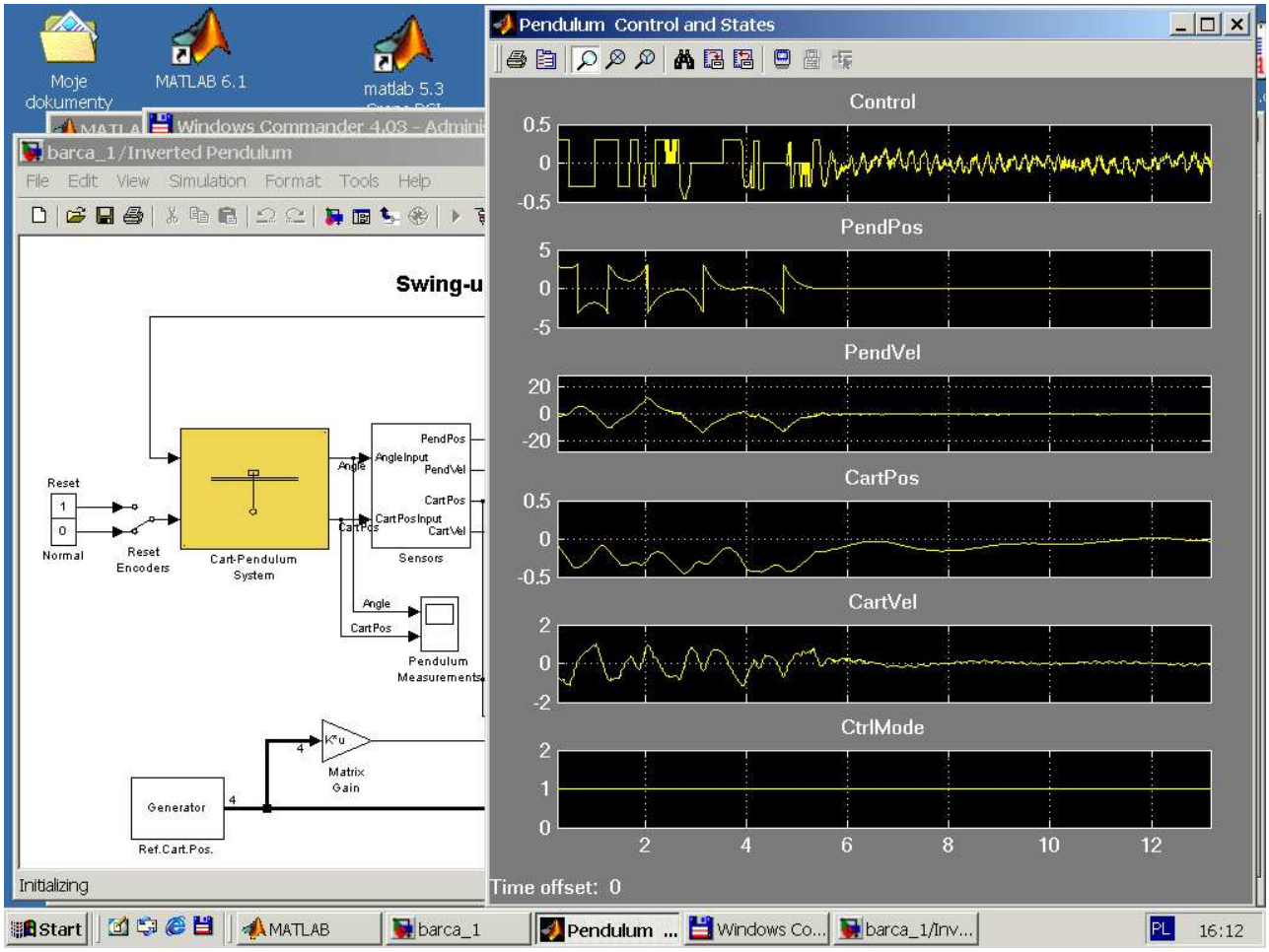


Fig. 6.2 The real time experiment window

There are two algorithms operating in this demo: *Controller Swing-up* and *Controller PID*. They are periodically selected in time by *Algorithm Selector*.

REMARK

The practical may be stopped any time by pressing the red Emergency Switch. It is useful in the case of unexpected behaviour of the system.

7. Prototyping an Own Controller in RTWT Environment

In this section the method of building your own controller is described. The *Real Time Windows Target* (RTWT) toolbox is used. How to use the *Pendulum* software is shown in sections 6 and 11. Here we introduce the reader how to proceed in the RTWT environment.

Only correct configuration of the MATLAB, Simulink, RTW, RTWT and C compiler guaranties the proper operation of the system.

To build the system that operates in the real-time mode the user has to:

- create a Simulink model of the control system which consists of the *Cart-Pendulum System* block and other blocks chosen from the Simulink library,
- build the executable file under RTWT (see the pop-up menus in Fig. 7.1)

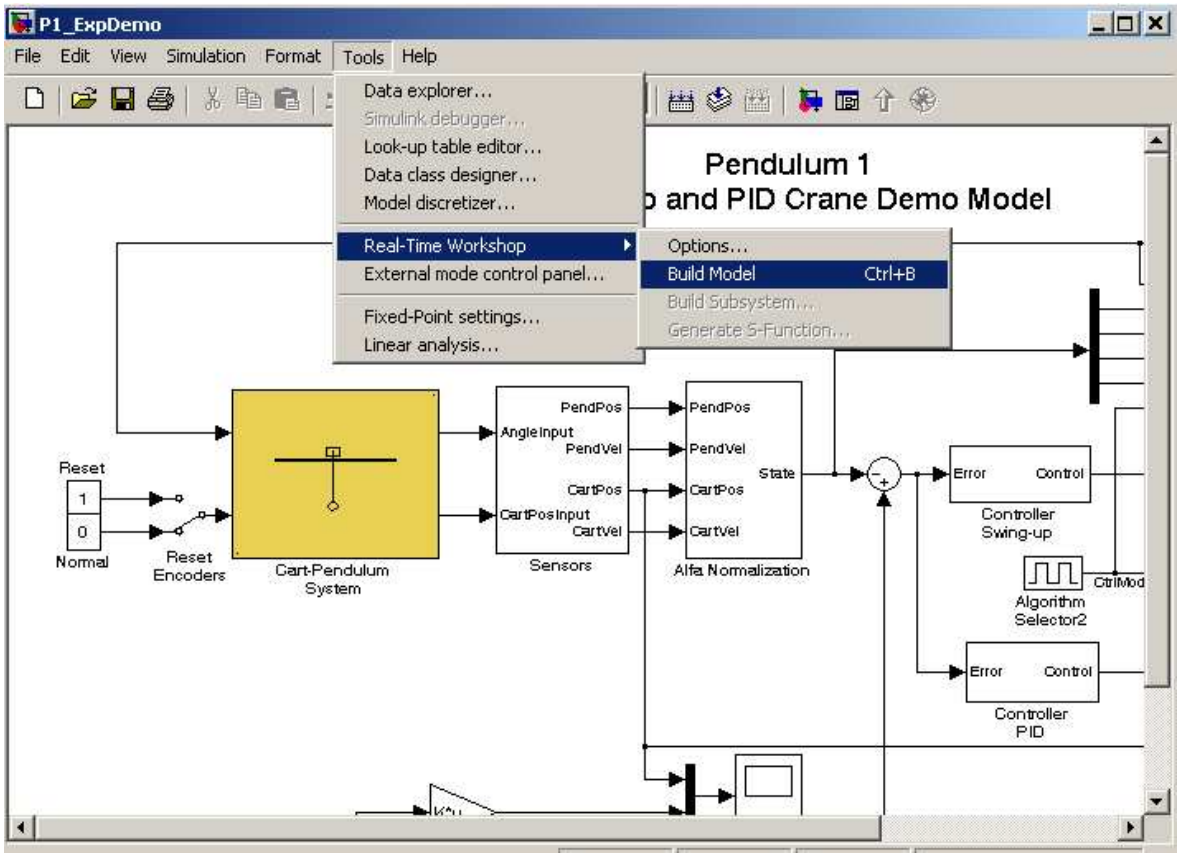


Fig. 7.1 Creating the executable file under RTWT

- start the real-time code to run from the *Simulation/Connect to target* and the *Simulation/Start real-time code* pull-down menus.

7.1. Creating a model

The simplest way to create a Simulink model for the pendulum system is to use one of the models included in the *Pendulum Control Window* as a template. For example, the *P1_ExpDemo* can be saved as the *MySystem.mdl* Simulink diagram. The *MySystem* Simulink model is shown in Fig. 7.2 .

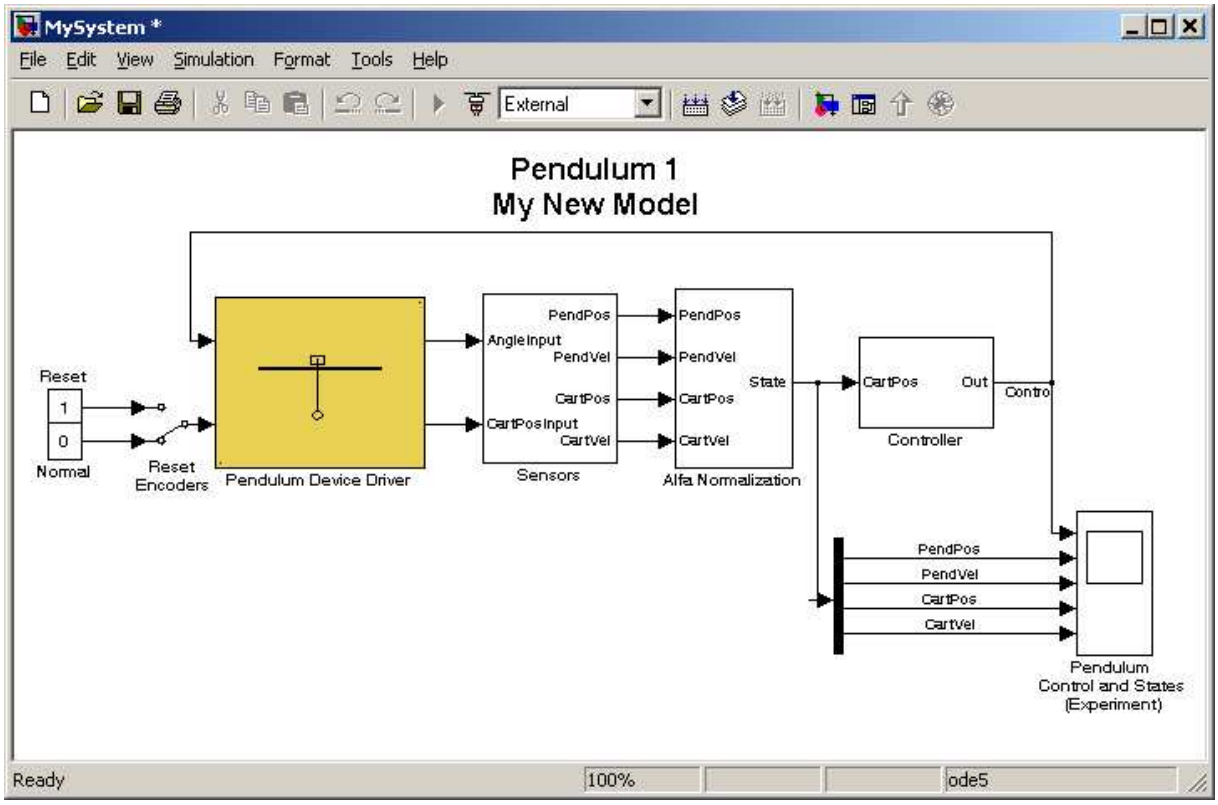


Fig. 7.2 The *MySystem* Simulink model

Now, you can modify the model. You have absolute freedom to develop your own controller. Remember, don't delete the *Pendulum Device Driver* block. This is necessary to support the data communication with the PCI I/O board.

Though it is not obligatory, we recommend you to leave the scope. You need a scope to watch how the system runs. Other blocks in the window are not necessary for a new project.

Creating your own model on the basis of the old example ensures that all internal options of the model are set properly. These options are required to proceed with compiling and linking in a proper way. To put the *Pendulum Device Driver* into the real-time code a special makefile is required. This file is included to the *Pendulum Toolbox* software.

You can use the most of the blocks from the Simulink library. However, some of them cannot be used (see *MathWorks* references manual for details).

The scope block properties are important for appropriate data acquisition and watching how the system runs. The *Scope* block properties are defined in the Scope property window (see Fig. 7.3). This window opens after the selection of the *Scope/Properties* tab. You can gather measurement data to the *Matlab Workspace* marking the *Save data to workspace* checkbox. The data is placed under *Variable name*. The variable format can be set as structure or matrix. The default *Sampling Decimation* parameter value is set to 1. This

means that each measured point is plotted and saved. Often we choose the *Decimation* parameter value equal to 5 or 10. A good choice to get enough points to describe the signal behaviour and to save the computer memory.

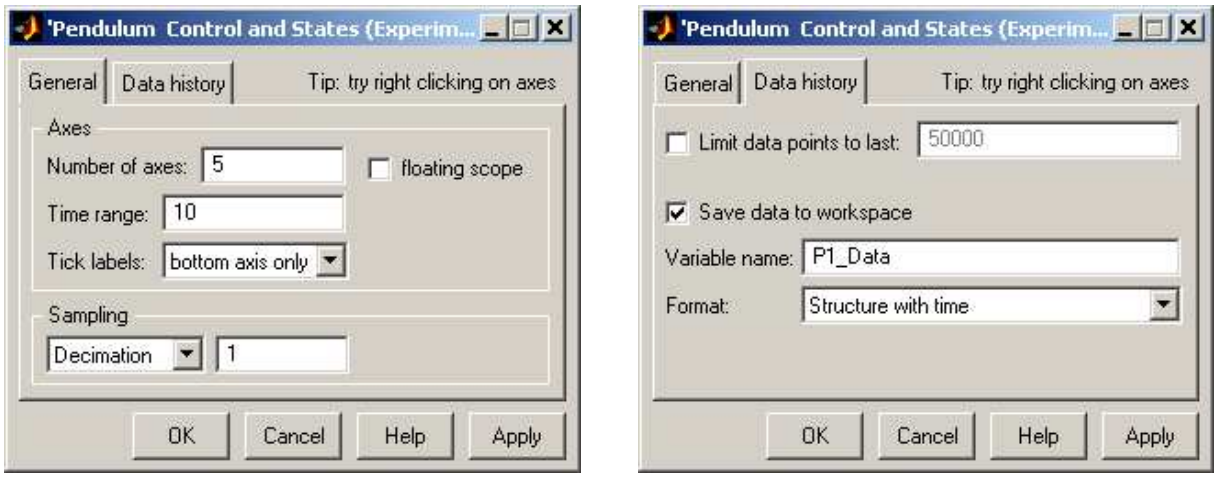


Fig. 7.3 Setting the parameters of the *Scope* block

When the Simulink model is ready, click the *Tools/External Mode Control Panel* option and click the *Signal Triggering* button. The window given in Fig. 7.4 opens. Select *XT Scope*, set *Source* as manual, set *Duration* equal to the number of samples you intend to collect, and finally close the window.

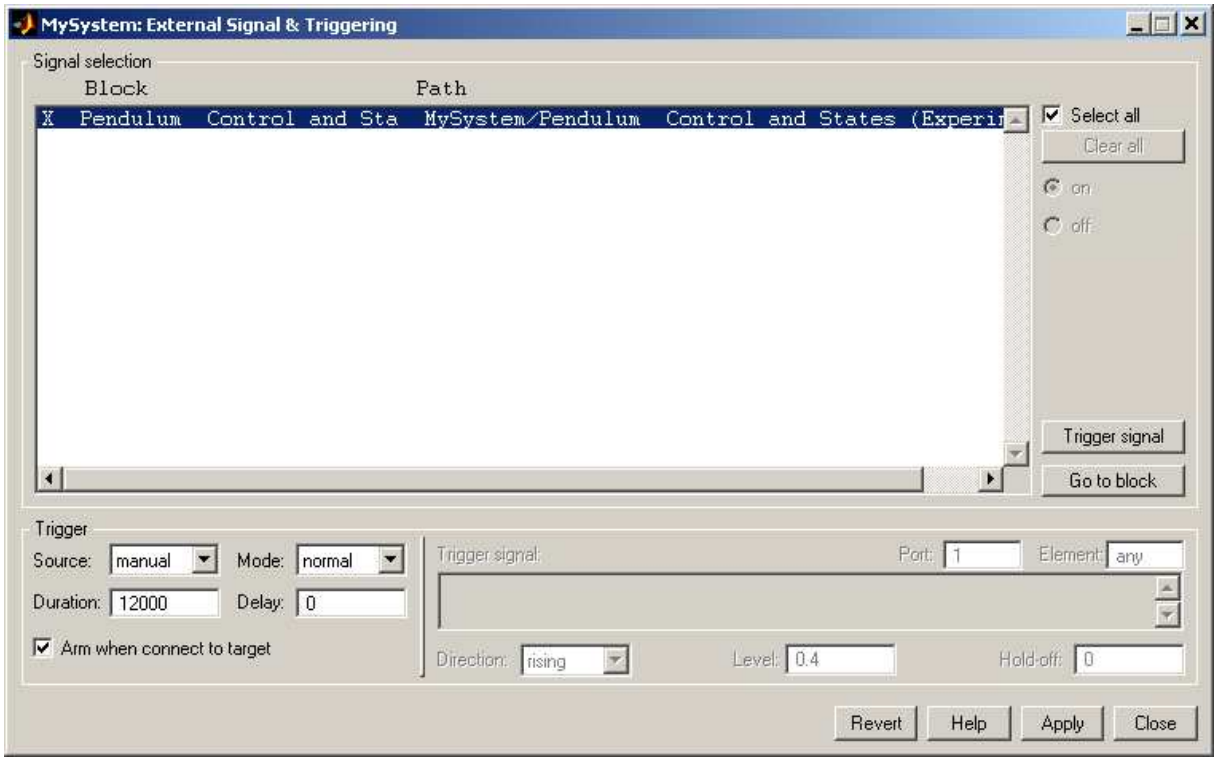


Fig. 7.4 *External Signal & Triggering* window

7.2. Code generation and build process

Once a model of the system has been created the code for the real-time mode can be generated, compiled, linked and downloaded into the target processor.

The code is generated by the use of Target Language Compiler (TLC) (see description of the *Simulink Target Language*). The makefile is used to build and download object files to the target hardware automatically.

First, you have to specify the simulation parameters of your Simulink model in the *Simulation parameters* dialog box (Fig. 7.5). The *Real-Time Workshop* and *Solver* tabs contain critical parameters.

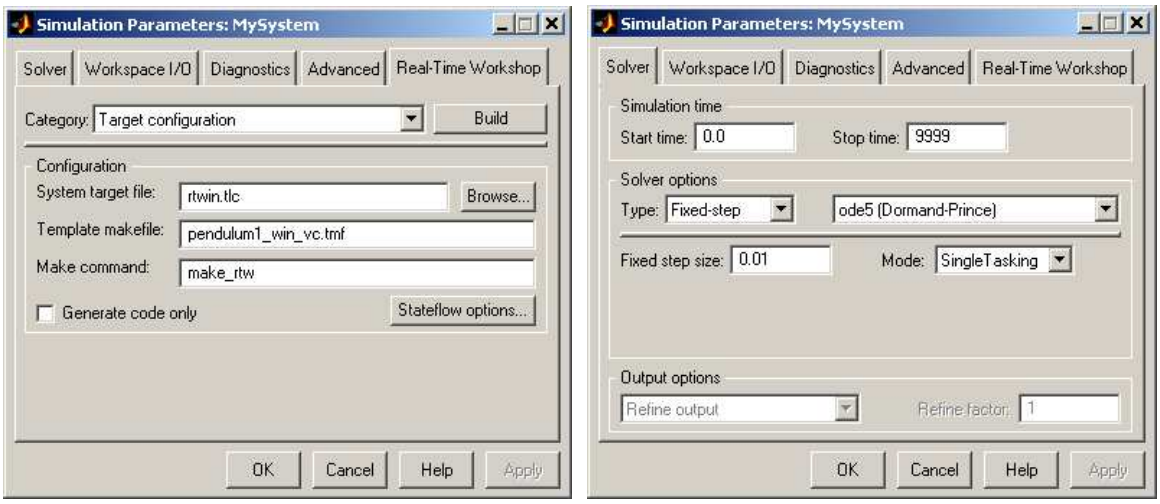


Fig. 7.5 Real-Time Workshop tags of the *Simulation parameters* menu option

The system target file name is *rtwin.tlc*. It manages the code generation process. The *pendulum1_win_vc.tmf* template makefile is responsible for C code generation using the Visual C/C++ compiler.
The *Solver* tab allows you to set the simulation parameters. Several parameters and options are available in the window. The *Fixed-step size* editable text box is set to 0.01 (this is the sampling period in seconds).

The *Fixed-step* solver is obligatory for real-time applications. If you use an arbitrary block from the discrete Simulink library or a block from the drivers' library remember that different sampling periods must have a common divider.

If the Matlab 7.0.4 version is used a third party compiler is not requested. The built-in Open Watcom compiler is used to creating real-time executable code for RTWT.
The *Configuration parameters* page for MATLAB 7.04 is shown in Fig. 7.6. Notice that *rtwin.tmf* template makefile is used. This file is default one for RTWT building process.

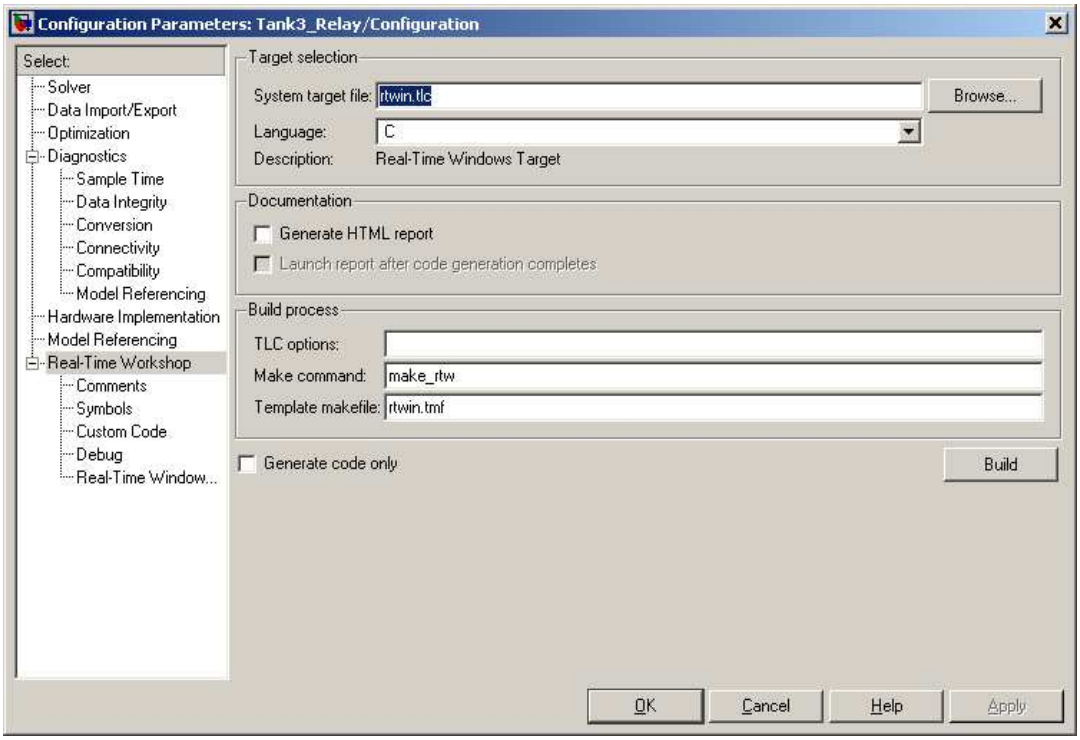


Fig. 7.6 *Configuration parameters* page for MATLAB ver. 7.04

The *Start time* has to be set to 0. The solver method has to be selected. In our example the fifth-order integration method – *ode5* is chosen. The *Stop time* field defines the length of the experiment. This value may be set to a large. Each experiment can be terminated by pressing the *Stop real-time code* button.

If all the parameters are set properly you can start the real-time executable building process. For this purpose press the *Build* push button on the Real Time Workshop page (Fig. 7.5), or simply "CRTL + B". Successful compilation and linking processes generate the following message:
Model MyModel.rtd successfully created
Successful completion of Real-Time Workshop build procedure for model: MyModel

Otherwise, an error message is displayed in the MATLAB Command Window.

8. Model and parameters

8.1. Mathematical model

Consider the system depicted in Fig. 8.1.

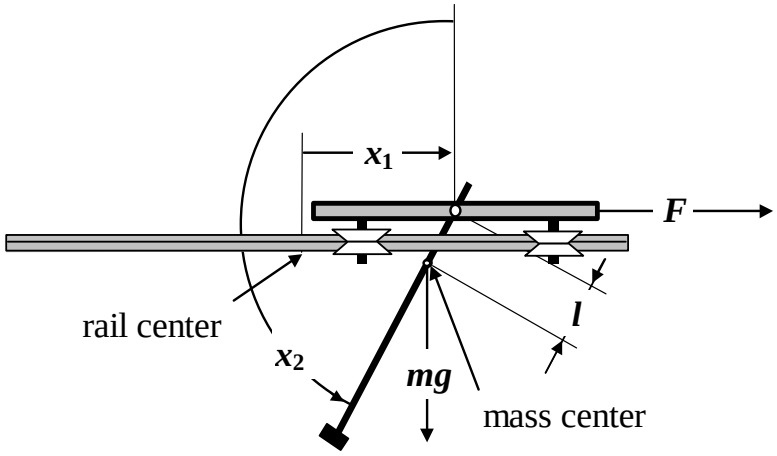


Fig. 8.1. Pendulum on a cart system

A pendulum rotates in a vertical plane around an axis located on a cart. The cart can move along a horizontal rail, lying in the plane of rotation. The state of the system is a vector x = col(x_1, x_2, x_3, x_4) where x_1 is the cart position, x_2 is the angle between the upward direction and the pendulum, measured counterclockwise (x_2 = 0 for the upright position of the pendulum), x_3 is the cart velocity, and x_4 is the pendulum angular velocity. A control force F, parallel to the rail, is applied to the cart. It is produced by a DC flat motor controlled by a pulse width modulation (PWM) voltage signal u, and $F = p_1u + p_2x_3$. The system control u takes values in the interval [−0.5, 0.5]. The total mass of the pendulum and cart is denoted by m. l is the distance from the axis of rotation of the pendulum to the center of mass of the system. J_p is the moment of inertia of the pendulum with respect to its axis on the cart. The cart friction is compound of two forces: the static friction compensated outside the model and the viscous friction proportional to the cart velocity, $f_c x_3$. There is also a friction torque in the angular motion of the pendulum, proportional to the angular velocity, $f_p x_4$. The state equations are as follows

$$\dot{x}_1 = x_3,$$

$$\dot{x}_2 = x_4$$

$$\dot{x}_3 = \frac{a_1 w_1(x,u) + w_2(x)\cos x_2}{d(x)}$$

$$\dot{x}_4 = \frac{w_1(x,u)\cos x_2 + a_2 w_2(x)}{d(x)} \qquad (7.1)$$

$$|u(t)| \le u_{\max} \tag{7.2}$$

where

$$w_1(x,u) = k_1 u - x_4^2 \sin x_2 - k_2 x_3 \tag{7.3}$$

$$w_2(x) = g \sin x_2 - k_3 x_4 \tag{7.4}$$

$$d(x) = b - \cos^2 x_2 \tag{7.5}$$

$$a_1 = \frac{J_p}{ml}, \quad a_2 = \frac{1}{l}, \quad b = a_1 a_2 = \frac{J_p}{ml^2} \tag{7.6}$$

$$k_1 = \frac{p_1}{ml}, \quad k_2 = \frac{f_c - p_2}{ml}, \quad k_3 = \frac{f_p}{ml}. \tag{7.7}$$

All parameters of the original model are given in Table 1.

8.2. Calculation of the inertia moment

The figure depicted below is used to illustrate calculations of the inertia moment.

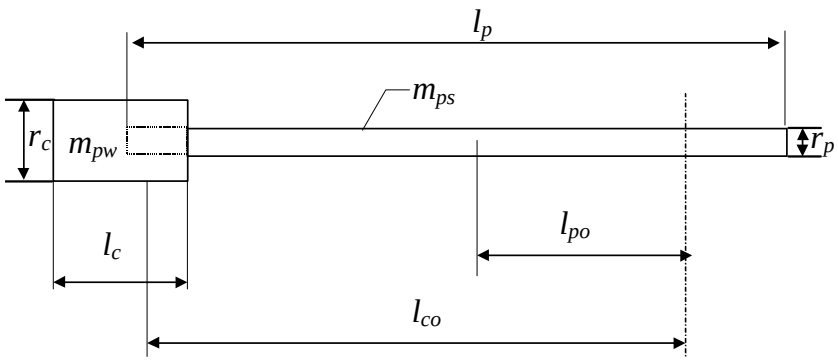


Fig. 8.2 View of the pendulum

We use the following notation

m_{pw}	-	mass of the load [kg],
m_{ps}	-	mass of the pole [kg],
l_p	-	length of the pole [m],
l_{po}	-	distance between centre of the pole mass and the pendulum rotation axis [m],

l_{pwo} - distance between centre of the load mass and the pendulum rotation axis [m],

l_c - length of load [m],

l_{co} - distance between centre of the load mass and the pendulum rotation axis [m],

r_p - radius of the pole [m],

r_c - radius of the load [m].

m_c - equivalent mass of the cart, pulleys and DC rotor

Moment of inertia of pendulum related to the pendulum rotation axis:

$$Jp = \frac{1}{12} m_{pw} l_c^2 + \frac{1}{4} m_{pw} r_c^2 + m_{pw} l_{co}^2 + \frac{1}{12} m_{ps} l_p^2 + \frac{1}{4} m_{pw} r_p^2 + m_{ps} l_{po}^2$$

J can be expressed by J_p .

$$J = Jp - l^2 \left(m_c + m_p \right)$$

The distance from the centre of mass to the axis of rotation equals

$$l = \frac{l_{po} m_{ps} + l_{pwo} m_{pw}}{m_c + m_{ps} + m_{pw}}$$

Table 1. Parameters of the original model

Name	Description	Unit
m	Equivalent mass of cart and pendulum	0.872 [kg]
l	distance from axis of rotation to center of mass of system	0.011 [m]
f_c	dynamic cart friction coefficient	0.5 [Ns/m]
f_s	static friction of the cart	1.203 [N]
f_p	rotational friction coefficient	$6.65 \cdot 10^{-5}$ [Nms/rad]
J_p	moment of inertia of pendulum with respect to axis of rotation	0.00292 [kgm^2]
g	Gravity	9.81 [m/s^2]
p_1	control force to PWM signal ratio	9.4 [N]
p_2	control force to cart velocity ratio	–0.548 [Ns/m]
$u_{\max}$	maximum value of PWM signal	0.5
m_c	Equivalent cart mass	0.768 [kg]
m_{ps}	pole mass	0.038 [kg]
m_{pw}	load mass	0.014 [kg]
R_l	rail length	1.8 [m]
l_p	length of pole	0.5[m]
l_{po}	distance between centre of pole mass and rotation axis	0.107[m]
l_c	length of load	0.03 [m]
l_{pwo}	distance between centre of load mass and rotation axis	0.354 [m]
T	pendulum period	1.17 [s]
J	moment of inertia related to the mass centre	0.00282 [kg·m^2]

8.3. The simulation model in Simulink

The simulation model of the real pendulum-cart system is presented in Fig. 8.3. The *Dynamics* block is taken from Fig. 5.3 (Simulation model).

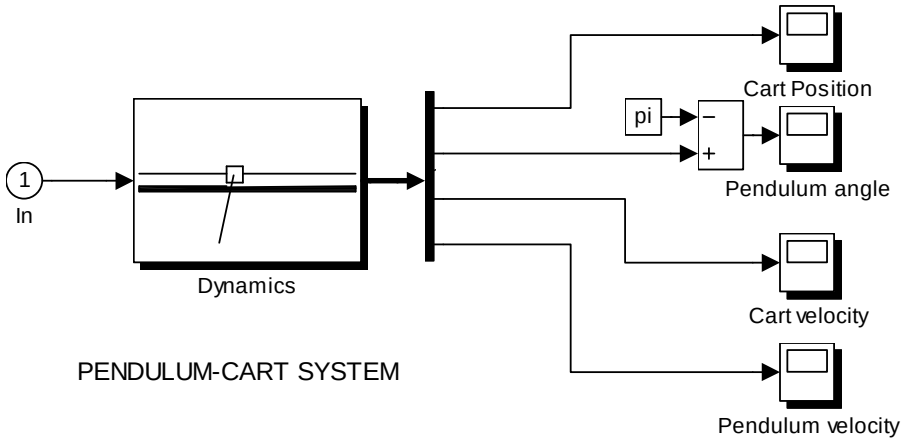


Fig. 8.3 Pendulum-cart model

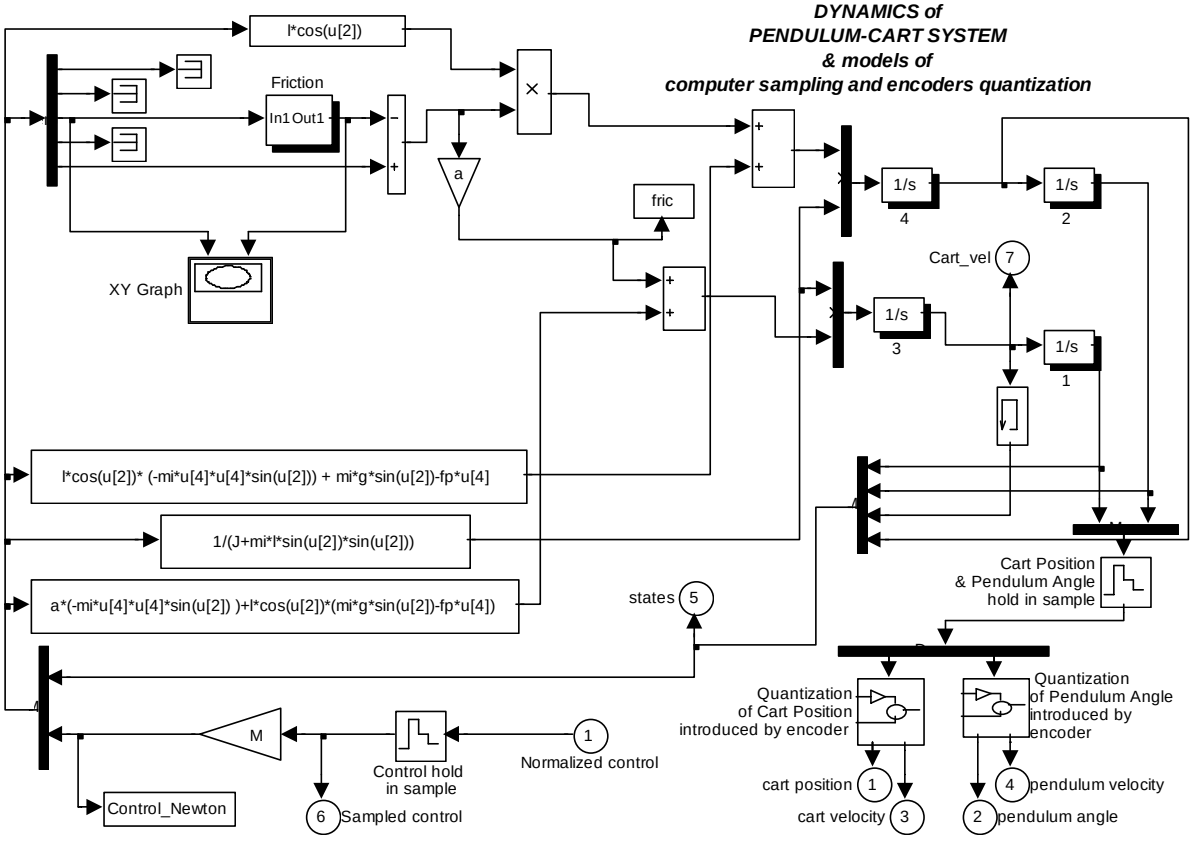
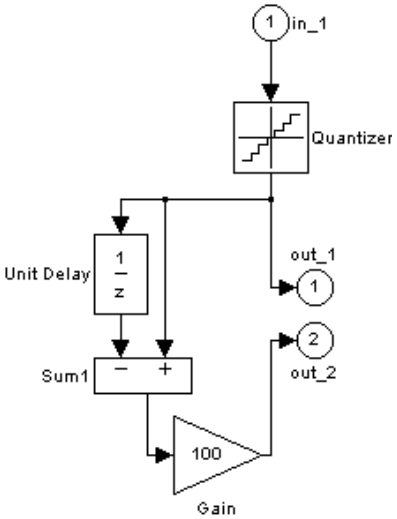


Fig. 8.4 Interior of the *Dynamics* block

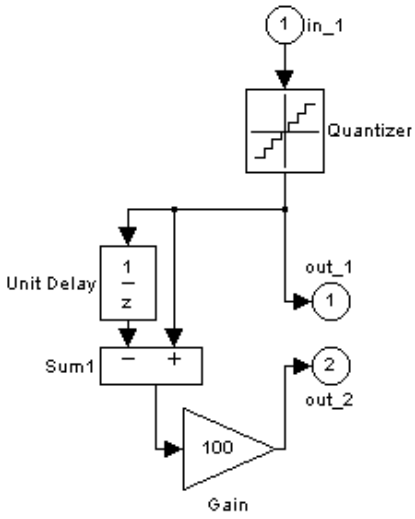
The cart-pendulum dynamics is modelled in the masked block *Dynamics*. The unmasked block *Dynamics* is given in Fig. 8.4. It contains several specialised blocks from the SIMULINK library like: *Dead zone* and *Saturation*, and the *Friction* block dedicated for pendulum-cart system.

Two memory blocks are introduced to avoid the generation of algebraic loops. There are three *Dead zone* blocks to make the model sensitive to small values of velocities and control.
Notice, that the fifth order Runge Kutta integration method with the constant step 0.01 [s] is used.

In this model the *Dynamics* block is additionally equipped with the model of the computer sampling and the model of *Cart Position & Pendulum angle* quantization. The interiors of the quantization blocks are given in Fig. 8.5. In the real pendulum-cart system two state variables: the pendulum angle and cart position are measured. There are two encoders mounted on the pendulum and motor shafts. Both encoders have identical resolution 4096 increments per one revolution. The angle of the pendulum is measured with the resolution equal to 0.001534 rad ($2\pi/4096$) and the cart position with the resolution of 0.038 mm (one revolution of the motor encoder corresponds to 0.235 m of the cart travel. While the cart travels from one end of the rail to the other end the encoder makes four full revolutions. The computer sampling period has also a significant influence on the measured and control signals. The sampling period is fixed to 0.01 s. There are several reasons to choose this value. The sampling period 0.01 s means the control frequency 100 Hz. It is sufficient for the system with the natural frequency about 1 Hz. Further diminishing of the sampling period is not recommended. It must be a trade off between sampling and quantization. Quantization is defined by a construction of sensing devices – the encoders.



Quantization 5.7053e-5 m
of Cart Position
introduced by encoder



Quantization 0.001534 rad
of Pendulum Angle
introduced by encoder

Fig. 8.5 Quantization blocks

The *sample hold* and *quantization* blocks are added to make modelling compatible with the real system conditions.

The cart friction T_c in the model is a non-linear function of the cart velocity x_3,

9. Identification

To start identification click *Identification* button in the *Pendulum Main Window*. The window shown in Fig. 9.1 opens.
The pendulum parameters are stored in the *p1_parameters.m* file. This file is located in the M directory of Pendulum1 Toolbox. You can modify this file during the identification process. This file contains all necessary calculations for the simulation model. You can call this file typing *p1_parameters* command in Matlab Command Window or you can click the button *Display parameters*.

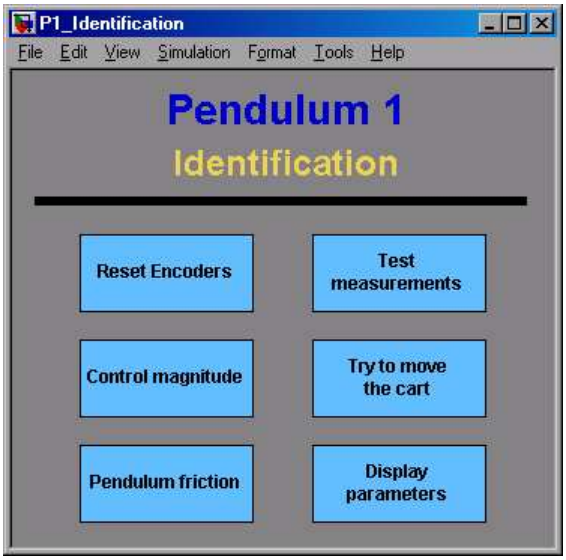


Fig. 9.1 **Pendulum identification window**

To get important information about identification process click on the *Help* button.

9.1. Checking measurement signals

Set the cart in the centre of the rail and set the pendulum motionless. Start MATLAB and type *p1_test* command. The window in Fig. 9.2 opens.
Next move the cart to the left approximately 10 cm. In the opened window click the *Get Position* button and read a cart position. This position must be negative. Move the cart to the right from the centre of the rail approximately 10 cm and click again *Get Position*. Read the next cart position. The position must be positive.
Swing by hand the pendulum to the left and click *Get Position*. The position must be positive. Next swing the pendulum to the right and click *Get Position*. The position must be negative. In this way you can check measuring operation of the encoders.

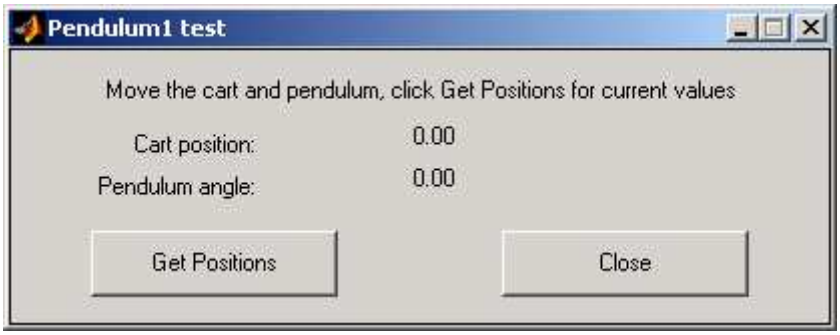


Fig. 9.2 Encoders check window

If measurements are correct click *Close* button.

9.2. Control magnitude identification

Mount the dynamometer to the cart as is shown in Fig. 9.3.

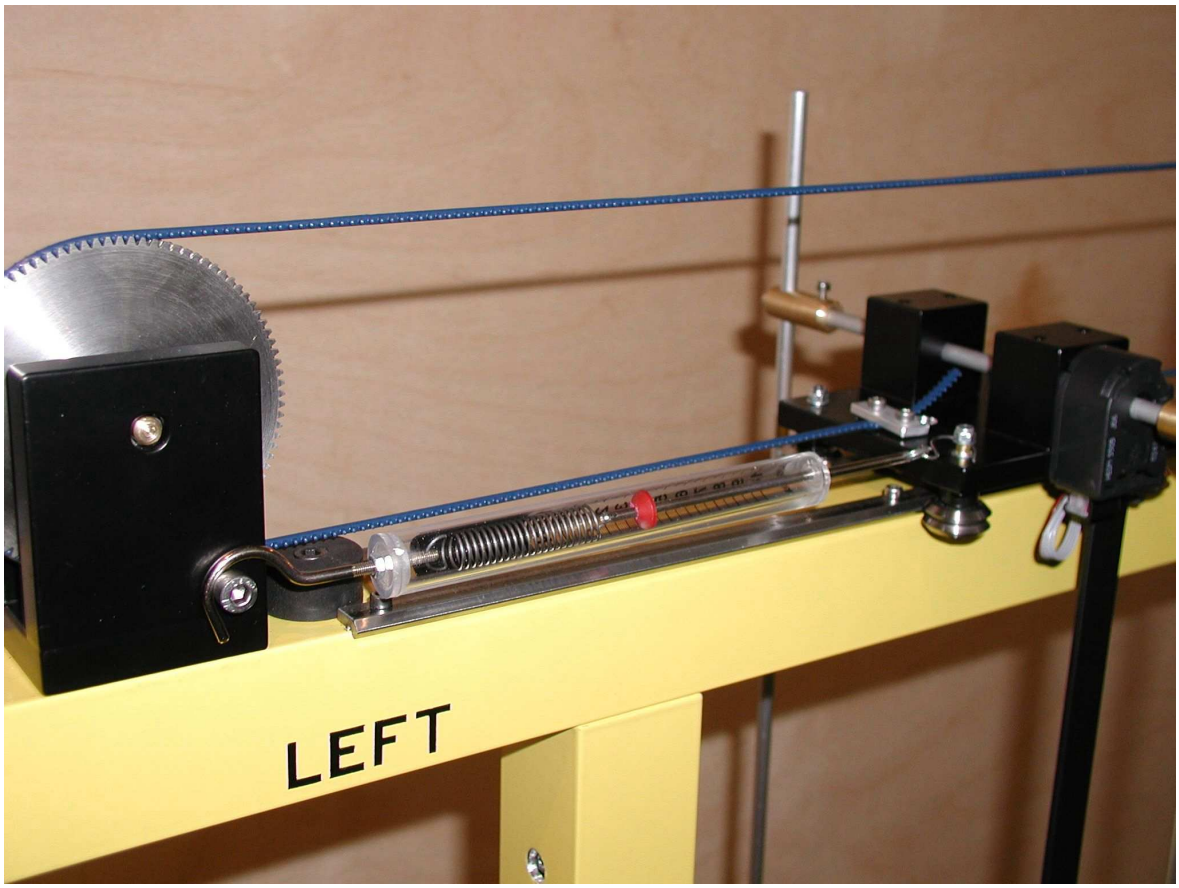


Fig. 9.3 Mounting the dynamometer

Click the *Control Magnitude* button (see Fig. 9.4) and set the control value of PWM duty cycle. Then observe the static force on the dynamometer scale. Write down results of this experiment. Repeat this step several times changing the PWM duty cycle.

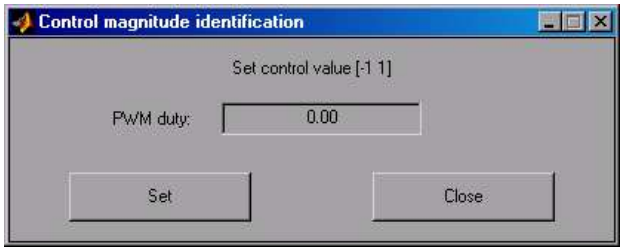


Fig. 9.4 PWM duty cycle setting

Finally you can calculate static force vs. control using (for example) *polyfit* function.

9.3. Minimum force needed to move the cart

We must know the minimum force f_s needed to move the cart. To run the identification procedure click *Try to move the cart* button in the *Identification* window.

Make sure that the control power button is pressed and give the positive answer in the active dialog window. Next Fig. 9.5 appears.

Set the cart at the left side and press OK. The algorithm starts.

Fig. 9.5 Information about cart motion

When the window (see Fig. 9.5) is active the cart is moving automatically form the left to the right side. The control force is slowly increased to detect the cart movement. Finally the cart achieves the right side of the rail. When you see Fig. 9.6 set the cart by hand at the right side and click *OK*.

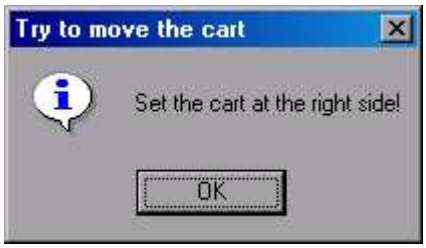


Fig. 9.6 Cart location

The experiment runs and the cart motion in the opposite direction is observed.

Fig. 9.7 Information about cart motion

When the procedure is finished the plot is displayed (Fig. 9.8). Bars in the figure shows the forces required to move the cart at different cart locations on the rail.

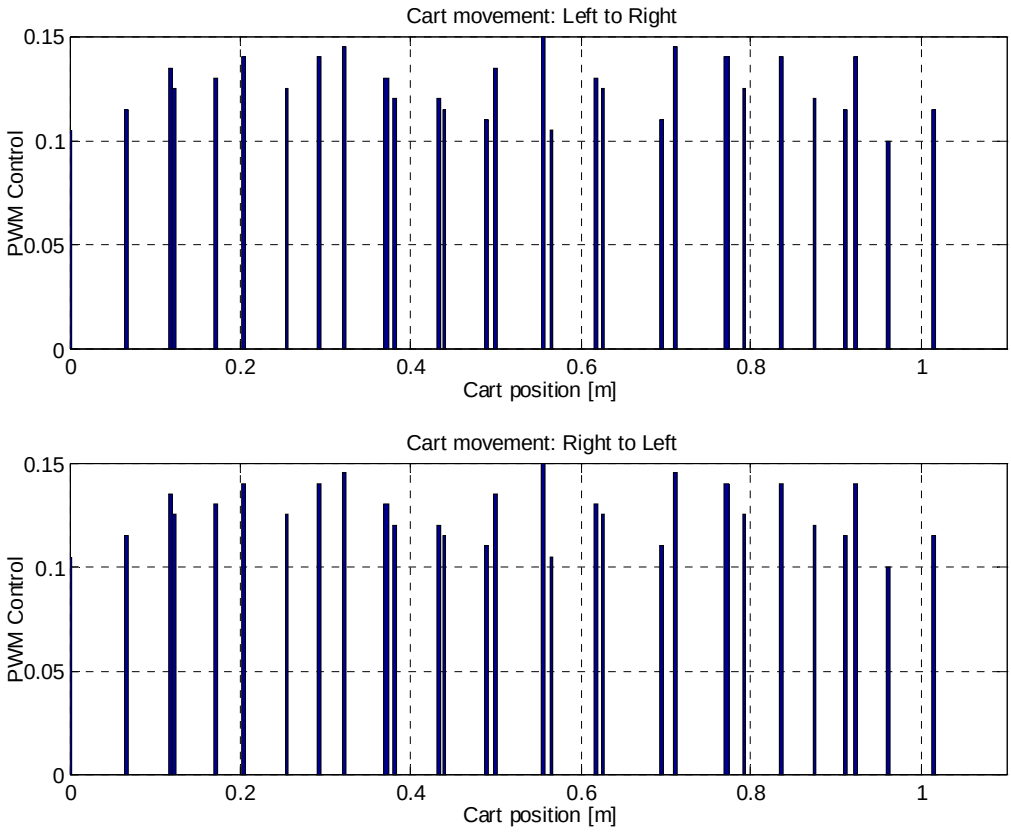


Fig. 9.8 Minimal force detection

In this case the medium value of the control signal (PWM duty cycle) is equal to 0.127.

9.4. Identification of the pendulum friction

Fix the cart at the left side of the rail using the spring as shown in Fig. 9.9.

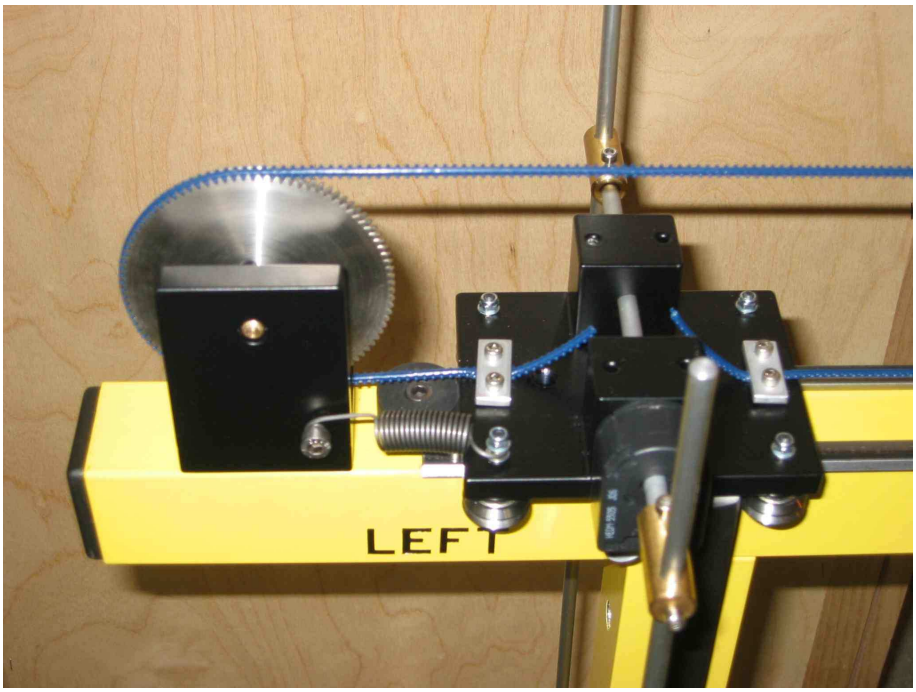


Fig. 9.9 Fixing the cart

When the cart is fixed click the *Pendulum friction* button placed in the *Identification* window.

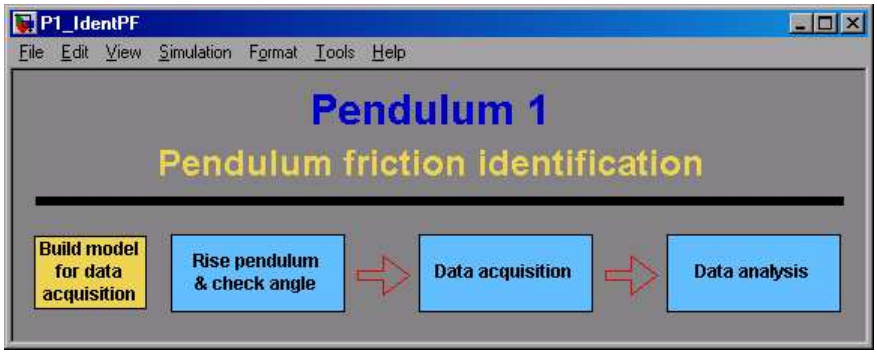


Fig. 9.10 Friction identification window

Click *Build model for data acquisition* button. Make sure that your current working directory is not one of directories in the MATLABPath.

Next raise the pendulum to $\pi \pm \pi/9$ (20 degrees up from the stable down position). If you click on the *Check Angle* button you see the current value of the pendulum angle.

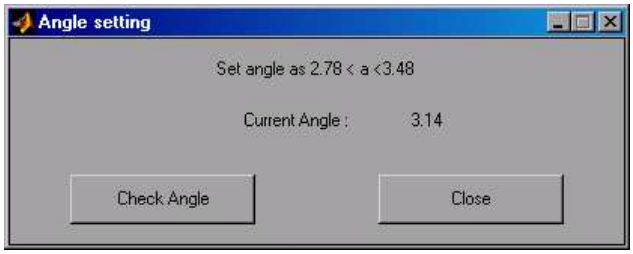


Fig. 9.11 Angle checking window

Click the *Close* button to continue identification. Then choose *Data acquisition* button to gather data. The pendulum is oscillating and data are being gathered. These data will be used to calculate friction coefficients. Finally choose *Data analysis* to perform the following tasks:

- calculation of the mean value of the pendulum period *T*,
- calculation of the pendulum friction (damping) coefficient,

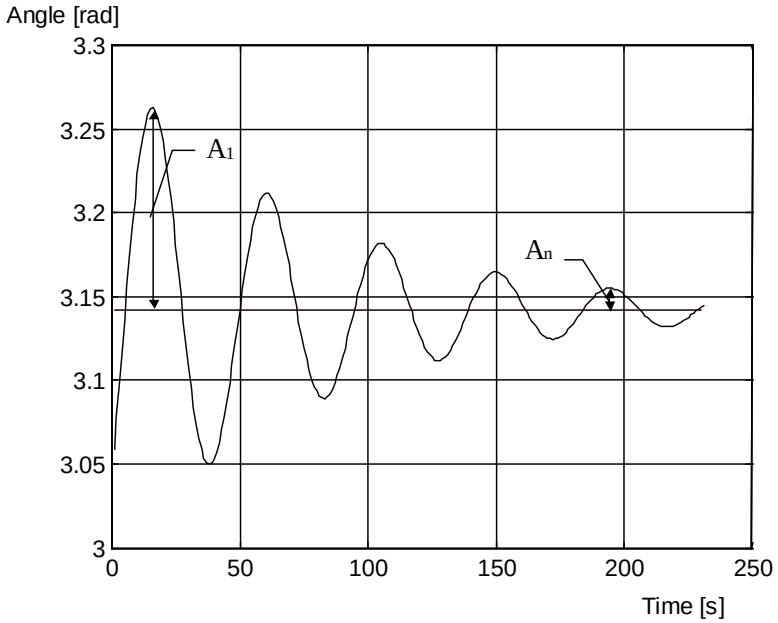


Fig. 9.12 Damped oscillations of the pendulum

The damping coefficient is calculated on the basis of local maximum values A_1, and A_n :

$$f_{pc} = \frac{2}{(2n-1)T}\log\left(\frac{A_1}{A_n}\right)$$

$$f_{pf} = J_p \cdot f_{pfc}$$

where :

f_p - dumping coefficient [kg·m^2/s]
J_p - moment of inertia of pendulum [kg·m^2]
n - number of local maximums upper the π value,
T - pendulum period [s],
A_1, A_n - local maximum values, after 1 oscillation and ‘n’ oscillations respectively

The plot of the most important data is automatically created (see Fig. 9.13).

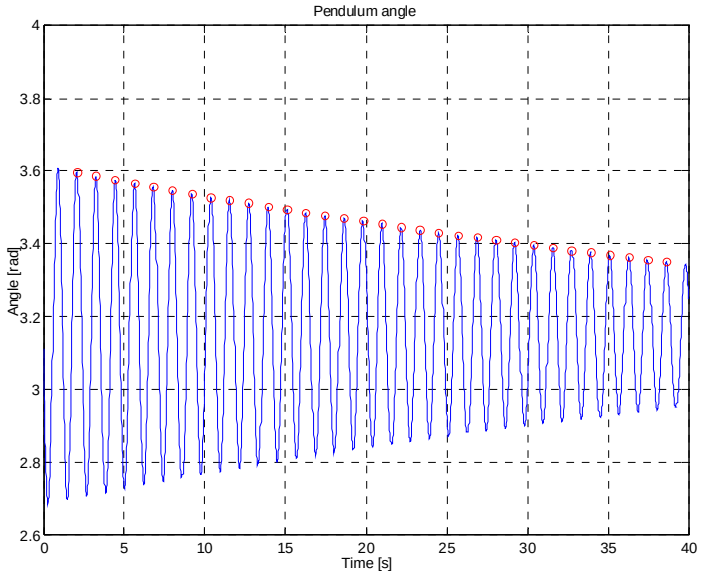


Fig. 9.13 Pendulum data plot

Finally, the results of data analysis are displayed in the Matlab Command Window:

```
Pendulum friction identification results:
Pendulum period = 1.177
Pendulum friction coefficient = 0.0227
```

9.4.1. Test of encoder quality

The detected amplitudes of oscillations are shown in Fig. 8.12. Due to a finite accuracy of the measuring device (encoder) the collected local maximum values are only distinguished after few oscillations. The encoder has the accuracy of 4096 [bits] per 2π [rad]. The minimum resolution thus is:

$$\Delta\varphi = \frac{2\pi}{4096} = 0.00153398 \ \text{[rad]}$$

9.5. Display parameters

Edit the *p1_parameters,m* file to store identified values. Next choose the *Display parameters* button to calculate and display parameters required for the Simulink model.

If you call the *p1_parameters.m* file from Matlab Command Window than all variables are loaded to the workspace.

10. Rule-based controller

The material presented in this chapter consists of two parts. The first corresponds to simulation of rule-based control. The second corresponds to practicals.

One can ask if it is possible that a human being can serve as a controller for the system? The answer is positive if a control algorithm is not too complex. Even such a control task as pole balancing by hand in its unstable upright position can be satisfactorily done by a human being. Unfortunately we are not able to balance the pendulum by pushing back and forth the cart. We simply fail and do not respond quickly enough to keep the pendulum in the unstable equilibrium point. However our experiments are not useless. We can gather knowledge about control algorithm, we can also build a reference model of the controller and simulate its behaviour. We can consider time and frequency responses of the system, generate additional characteristics such as potential or kinetics energy of the system etc. This information and our reasoning may result in a number of rules finally formed in an algorithm of control.

The rule-based control does not correspond to any of quality criteria as time-optimal, quadratic etc. It is rather a policy that achieves the control goal and is robust and insensitive to parameter changes and disturbances. In the pendulum-cart system when the control goal is defined as “swing-up and upright stabilization” we are considering the case when a trade off between two control actions is concerned. Namely, centring the cart and swinging-up the pendulum contradicts each other. To swing-up the pendulum the cart must be put in motion. Control rules applied to the cart centring may contradict raising-up the pendulum. Despite such effects the control goal has to be achieved finally. We can not guess control rules they are obtained by reasoning process. Therefore the rule-based approach is classified as an intelligent method. When conventional control algorithms fail intelligent methods seem to be more promising in solving complex control problems.

10.1. Simulation of rule-based control

Double click on the *Simulation Model & Controllers* button in the *Pendulum Control Window* to open the rule-based control model.
There are generally three control policies:

- swinging-up the pendulum and centring the cart to the middle of the rail
- stabilizing the pendulum in its upright position and stabilizing the cart in the middle of the rail
- limiting the cart position

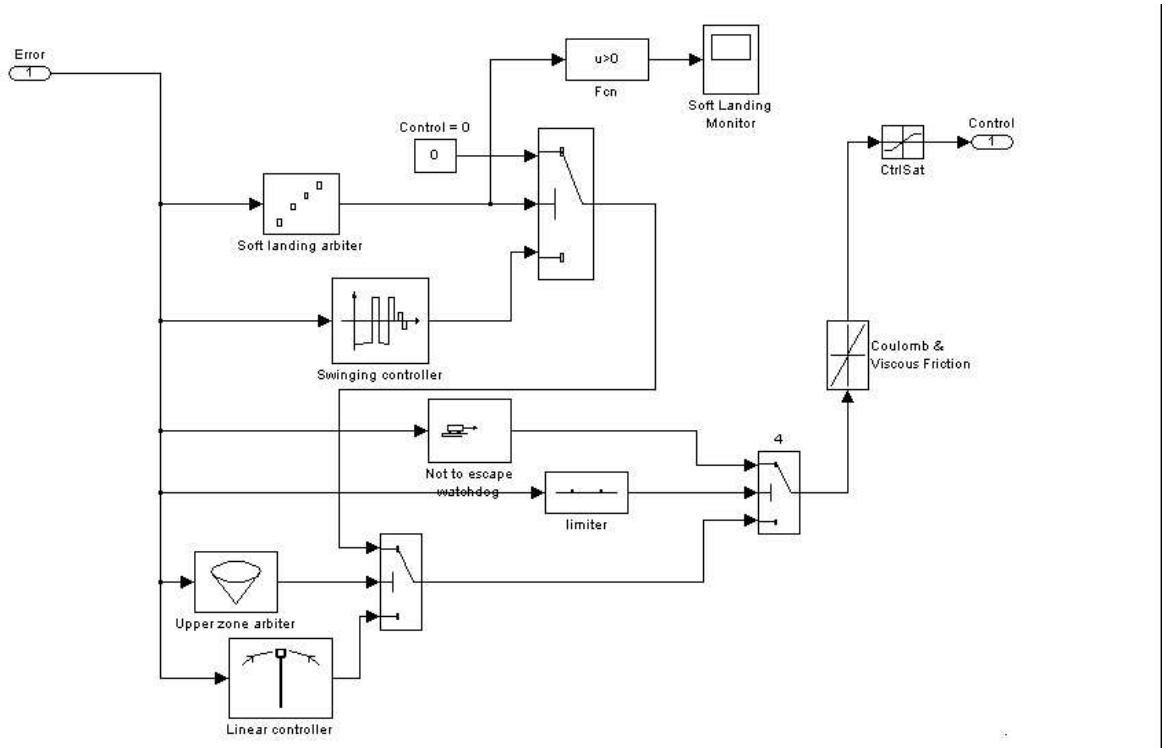


Fig. 10.1 The swing-up controller

Consider the control problem of steering the pendulum-cart system to a given target set **Z** in $\mathbf{R}^4$ with the use of admissible controls. In the problem of swinging up the pendulum to the upright position, the initial state can be any point in $\mathbf{R}^4$ and:

$\mathbf{Z} = \mathbf{Z}' = \{\text{col}(0,2i\pi,0,0): i = 0, +1, +2,...\}$.

Let us make some comments on the rule-based algorithm. The detailed rule base control algorithm has the **if...then...else** form. All the lines of the code are numbered from 1 to 18.

If	$\|x_2\|$-Stab.zone < 0	no.1
then	$u =: K_1x_1 + K_2x_2 + K_3x_3 + K_4x_4$	no.2
else	if $\{0.5\cdot(x_4)^2 + 9.81\cdot k\cdot R[\cos(x_2) - 1]\} > 0$	no.3
	then $u =: 0$	no.4
	then $u =: \text{sign}[x_4\cdot(\|x_2\| - \pi/2)]$	no.5
if	$u > 0$	no.6
then	$u =: U \cdot u + Friction$	no.7
else	$u =: U \cdot u - Friction$	no.8
if	$\|u\| > 1$	no.9
then	$u =: \text{sign}(u)$	no.10
if	$\|x_1\| - R_l/2 < 0$	no.11
then	$u =: \text{sign}(x_1)$	no.12

Table 2 Rule-based control algorithm

If the pendulum is in a vicinity of its upright position inside a stabilization cone (no.1) then a linear controller is applied (no.2). $K = K_1x_1 + K_2x_2 + K_3x_3 + K_4x_4$ is the feedback gain matrix obtained by solving an appropriate LQR problem off-line. The Simulink model of linear controller for the system is given in Fig. 10.2.

If the pendulum is outside the stabilization zone (condition no.1 not satisfied in the *Upper zone arbiter* block) it must be swung up. The algorithm checks if swinging up the pendulum does not result in overriding the upper unstable equilibrium point.

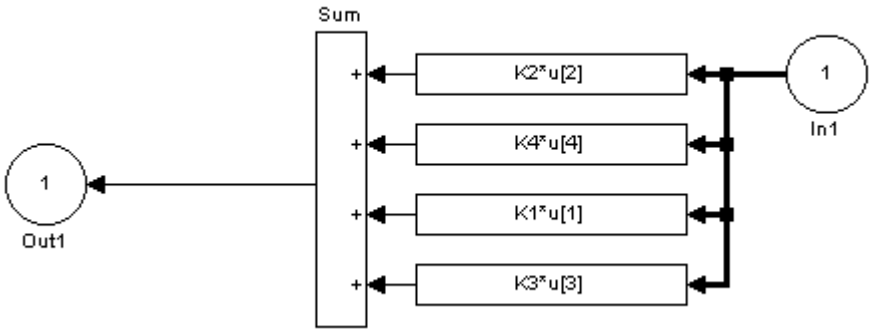


Fig. 10.2 Linear pendulum-cart controller

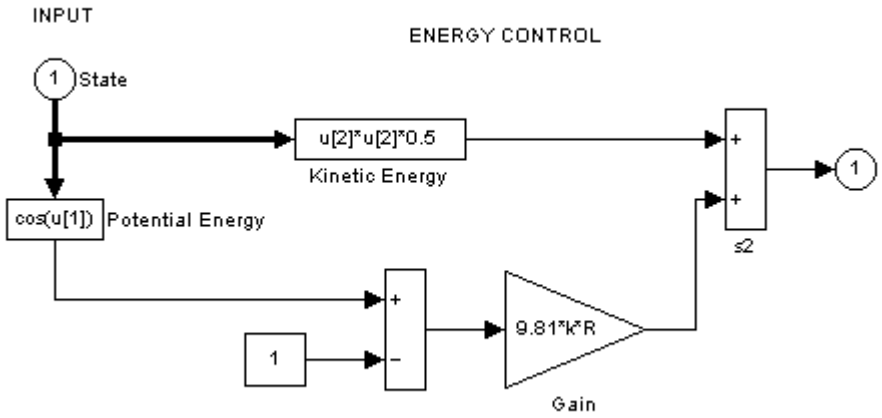


Fig. 10.3 Soft landing arbiter

To achieve a soft landing in the vicinity of the upright position, the algorithm checks whether the kinetic energy of the pendulum minus energy loss due to friction forces is sufficient to rise up the centre of gravity of the pendulum to its upright position. Look at condition no.3 in the *Soft landing arbiter* block (see Fig. 10.3). If the condition is satisfied then the control is set to zero (no.4).

The swinging rule has a very simple form (no. 5). It is "bang-bang" with switching moments related to the sign of the angular velocity and angle position, which points up, or down. The *Swinging controller* model is presented in Fig. 10.4.

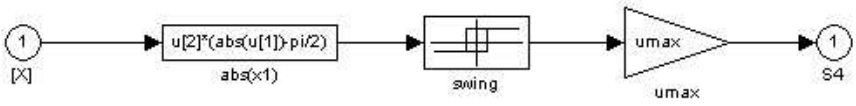


Fig. 10.4 Swinging controller

There are also a number of effects difficult for modelling. For example, the effects related to Coulomb friction of the cart. We can incorporate these effects straight into the control. If control is positive (no.6) we can make it "more positive" by adding an amount of control to overcome the Coulomb friction (no.7). Similarly, we make a negative control "more negative" (no.8). The *Coulomb Friction & Swinging Control Gain* block perform this action. Then the control is normalised to stay in the range [-1, +1] (no.9 and no.10). Usually the normalisation performed by the *Saturation* block is not active because the maximal control value is reduced by the *U* parameter. However usually we do not use the maximal force *M* generated by the DC motor. Thus in our disposal we have the parameter *U* to reduce this force proportionally, ($0 \leq U \leq 1$). This control signal is sampled and multiplied by *M* inside the *Dynamics* block. The real control magnitude *M* is equal to 12.8 N.

We use *Umax*=0.2 so we use only 20% of the DC motor power. We introduce the Coulomb friction compensation modelled by the *Friction* parameter (*Friction*=0.05).

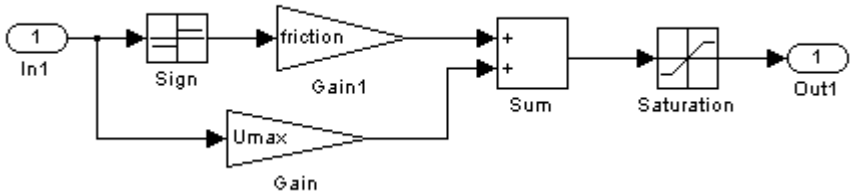


Fig. 10.5 *Coulomb Friction & Control Gain* controller

Finally, the *Limiter* block is added to reinforced centring of the cart and preventing it from overrunning the ends of the rail. The rule is very simple. If the range given by the "rail limit" parameter in the *limiter* block is exceeded (no.11), then the opposite to the direction of motion maximal steering force is applied to the cart (no.12).

The results of simulation are given in: Fig. 10.6, Fig. 10.7, Fig. 10.8, Fig. 10.9 and Fig. 10.10

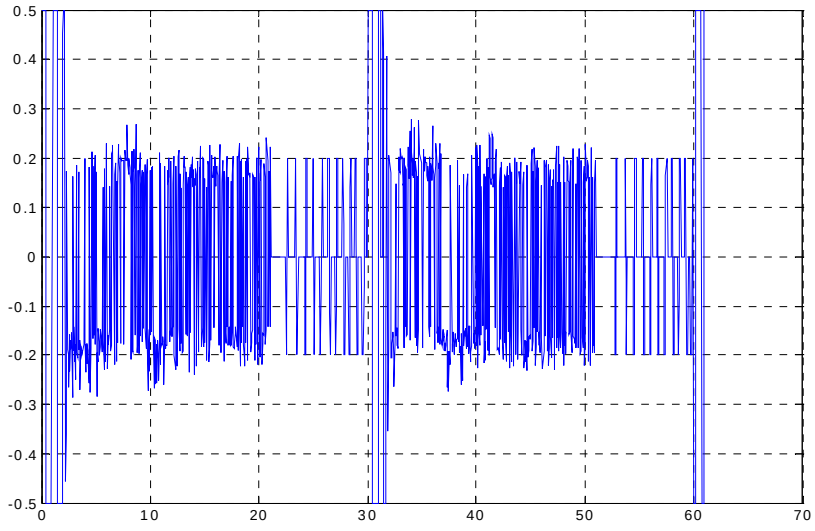


Fig. 10.6 Simulated control

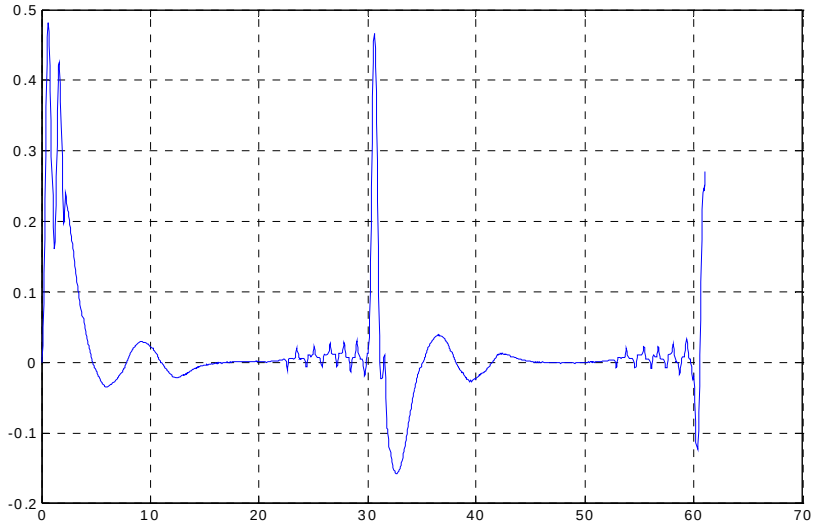


Fig. 10.7 Simulated cart position

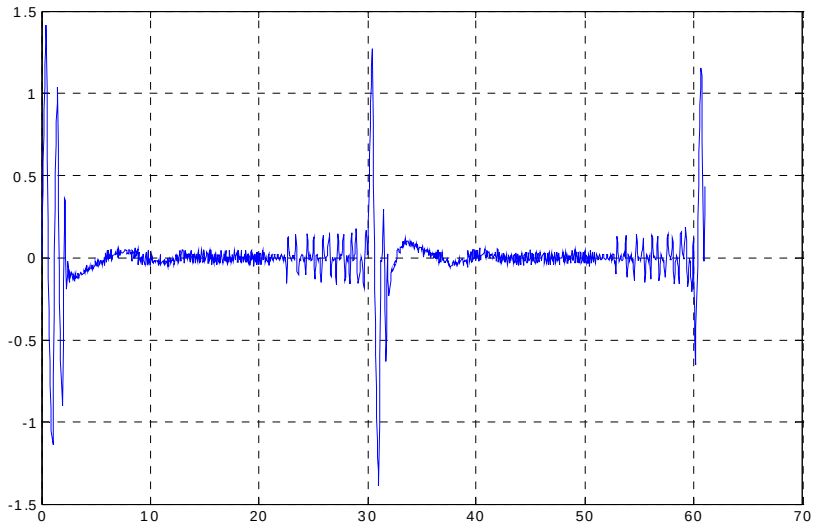


Fig. 10.8 Simulated cart velocity

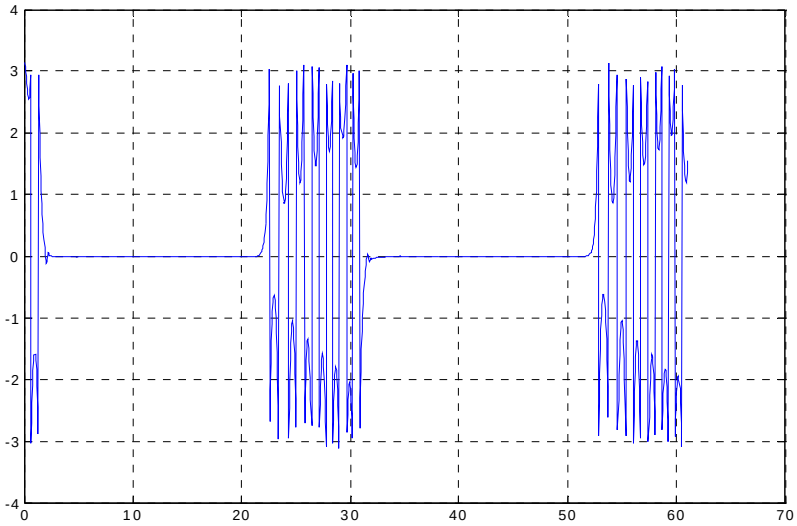


Fig. 10.9 Simulated pendulum angle

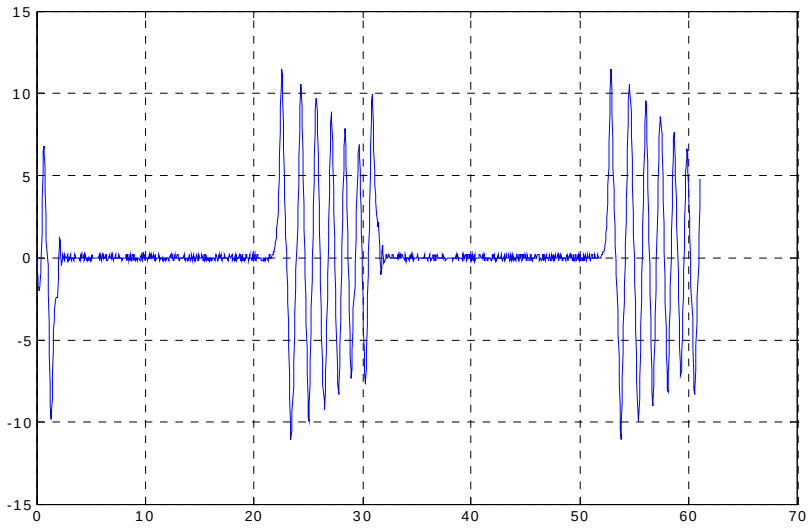


Fig. 10.10 Simulated pendulum angular velocity

10.2. Real-time control experiment

If we perform the similar experiment at the system we obtain the results shown in Fig. 10.11, Fig. 10.12, Fig. 10.13, Fig. 10.14 and Fig. 10.15.

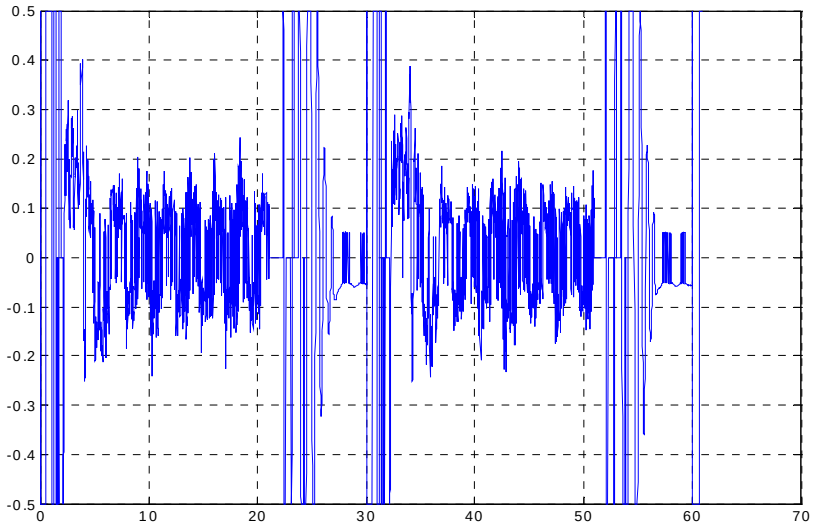


Fig. 10.11 Real-time control

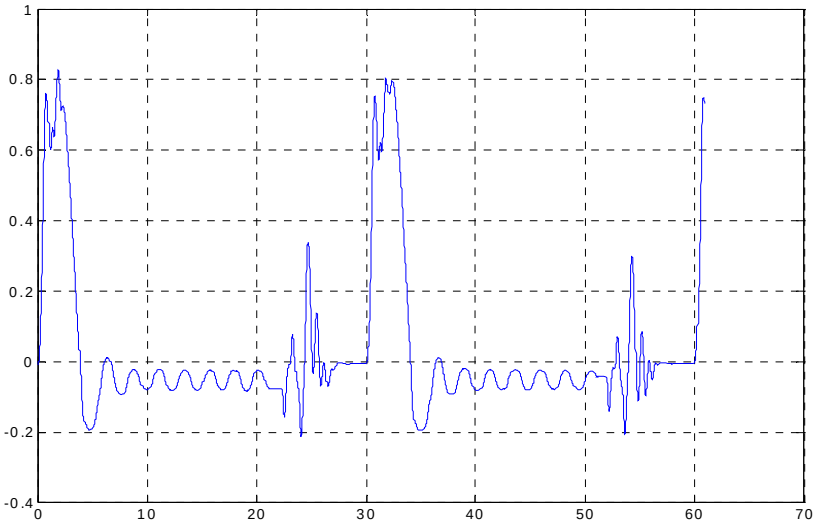


Fig. 10.12 Real-time cart position

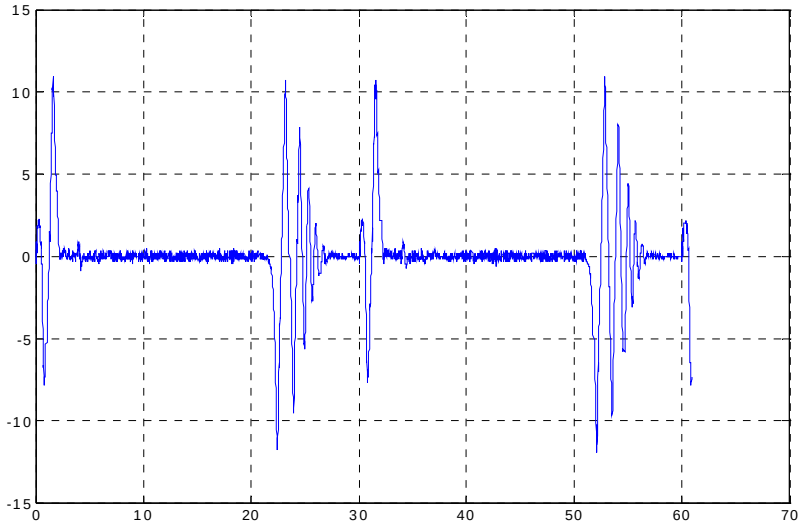


Fig. 10.13 Real-time cart velocity

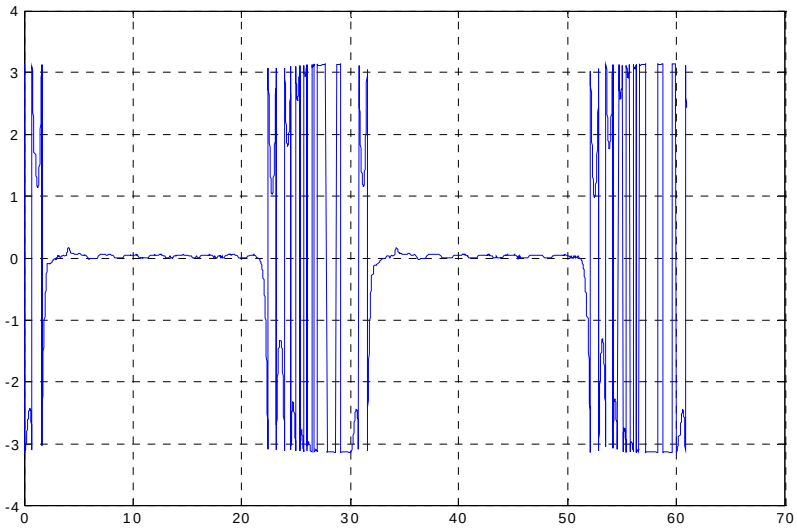


Fig. 10.14 Real-time pendulum angle

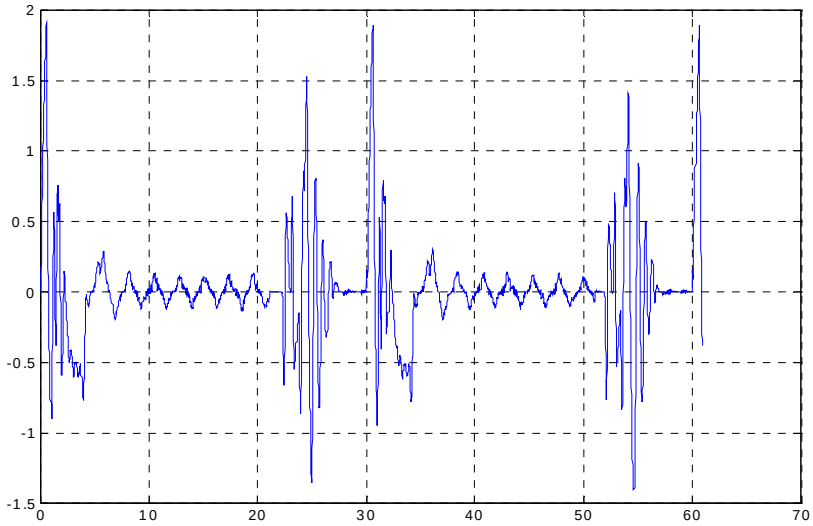


Fig. 10.15 Real-time pendulum angular velocity

We can tune the parameters of the model to obtain the model corresponding more accurately to real dynamics of the system.

11. Fuzzy Controller for Pendulum-Cart System

Any controller comprises two essential components: a mechanism for knowledge representation and a mechanism for decision-making. With conventional controllers a mathematical model (such as a set of differential equations or transfer function) represents knowledge and decision-making is based on optimisation.

When the system is complex or less precisely known it becomes difficult to characterize the knowledge adequately with conventional means. One must find alternative ways to encode the available knowledge about the system. It can result in the development of intelligent controllers. To this group among others belong fuzzy controllers.

The fuzzy controller *Crane Fuzzy Controller* of section 3 are presented in the simulation and real environments.

11.1. Fuzzy control

We assume that the reader is familiar with the *fuzzy set* terminology. Nevertheless we give a brief summary of definitions:

- A *fuzzy set A,* defined over some *universe of discourse,* is a mapping that universe in the interval [0, 1]. The process of creating this mapping is called *fuzzification*.
- A membership function is the curve that defines how true a given statement is for a given input value. The membership takes on the value of zero for no membership and one for full membership. Values in the range 0 to 1 represent partial membership to the given set.
- The *fuzzy set* is defined in terms of a pair $\{x, \mu(x)\}$ for all x in the *universe of discourse*. The *membership function* denoted by $\mu : F \rightarrow [0, 1]$ represent partial or graded membership to the given set *A*. For example the fuzzy set representing the term *fast* is depicted in Fig. 11.1.

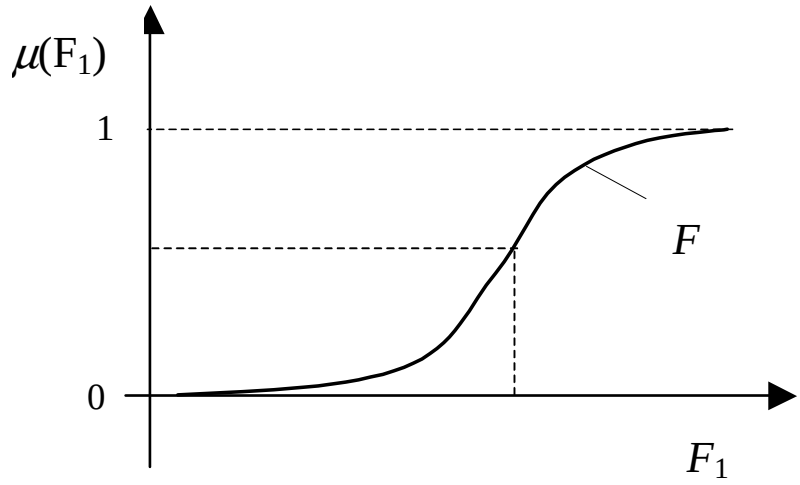


Fig. 11.1 Fuzzy set representing the term *fast*

The curve in Fig. 11.1 defines the transition from *not fast* to *fast*.

- A *linguistic variable* takes a linguistic term such as fast, slow, etc.
- Let *A* and *B* be fuzzy sets defined over the *X* universe of discourse. Then their intersection, $A \cap B$, is defined in terms of $\mu_A(x)$ and $\mu_B(x)$ as $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$
- Let *A* and *B* be fuzzy sets defined over the *X* universe of discourse. Then their union, $A \cup B$, is defined in terms of $\mu_A(x)$ and $\mu_B(x)$ as $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- Fuzzy reasoning is called the *inference mechanism*. Fuzzy rules take form:
  If x_A is *A* and / or x_B is B ... then y_Z is *Z*
  There are fuzzy relations over the Cartesian product of the *universe of discourses* of antecedent and consequent variables. For example, the rule *R*: If *x* is *A* then *y* is *Z*, is represented as a fuzzy relation defined as:
  $\mu_R(x, y) = \min(\mu_A(x), \mu_Z(y))$.
  - Interpreting an "If ... then ..." rule involves: evaluating the antecedent (involving *fuzzification)* and applying the result to the consequent (*implication*).
  - *Defuzzification* can be interpreted as obtaining a crisp single value of a possibility distribution. The typical defuzzification formula has the form:
  Mean Of Maximum = $\text{argmax}(\mu_Z(y))$

The following steps in the design of a fuzzy controller are essential:

- Selection of input/output variables
- Scaling the variables (if necessary)
- Definition of membership functions for all variables –fuzzification
- Development of the inference mechanism – fuzzy rules
- Selection of the defuzzification strategy
- Re-scaling the variables (if necessary).

The point of fuzzy logic is to map the input space to the output space, and the primary mechanism for doing this is the inference system which consists of "if and if - then" statements called rules. The "if" part of a rule is called the antecedent, while the "then" part of the rule is called the consequent. In fuzzy logic if the antecedent is true to some degree, then the consequent is true to the same degree. Multiple rules can be active for the same input value. It means that membership functions overlap in parts each other. A few rules can interpolate the *universe of discourse.*

In this section we discuss how to create and edit fuzzy inference systems by hand using the Fuzzy Logic Toolbox. There are two methods of building a fuzzy controller: interactive using the graphical tool and automatic using clustering and the adaptive neuro-fuzzy mechanism. There are also two types of fuzzy inference system: Mamdani and Sugeno. In this section we focus on the Mamdani inference system edited interactively by hand. This approach makes possible in a direct way to understand consequences of modifications being introduced into the inference mechanism and membership functions.

The variables are ordered in the following way: cart position, pendulum angle, cart velocity and pendulum angular velocity.

To start the simulation of fuzzy controller you must simply click on the *Fuzzy controller* button in the *Pendulum Control Window* and next the *Simulation* button (see Fig. 11.2). To start the real-time fuzzy experiment click on the *Fuzzy experiment* button. One can also open the fuzzy editor to visualize the inference fuzzy system (using the *Fuzzy editor* button).

The *Show FIS Message* button lists all warnings that occur during a real-time fuzzy experiment. A typical warning is generated by the fuzzy toolbox if the current input or output variable is outside the assumed *universe of discourse*. As far as simulation is concerned warnings do not stop the simulation run. They only slow down the execution of simulation by writing messages in the MATLAB Command Window. In the real-time mode warnings do not stop the run of the system, they are stored in the memory for further analysis of errors. When the experiment is finished we can read the list of warnings double clicking the *Show FIS Message* button.

11.1.1. How to run a simulation?

If we click the *Fuzzy controller* button in the *Pendulum Control Window* then the window given in Fig. 11.2 opens.

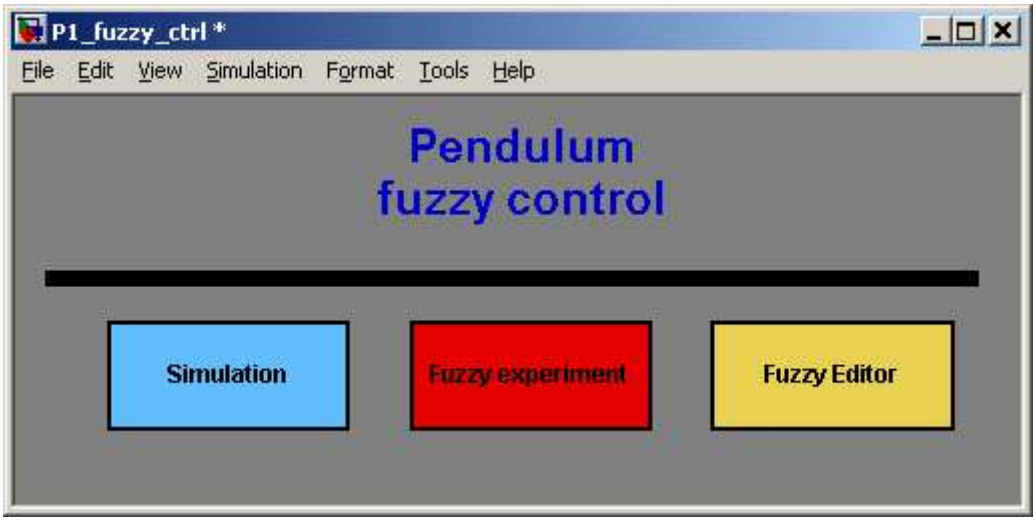


Fig. 11.2 Pendulum fuzzy control window

To perform simulation click the blue *Simulation* button and model given in Fig. 11.3 opens.

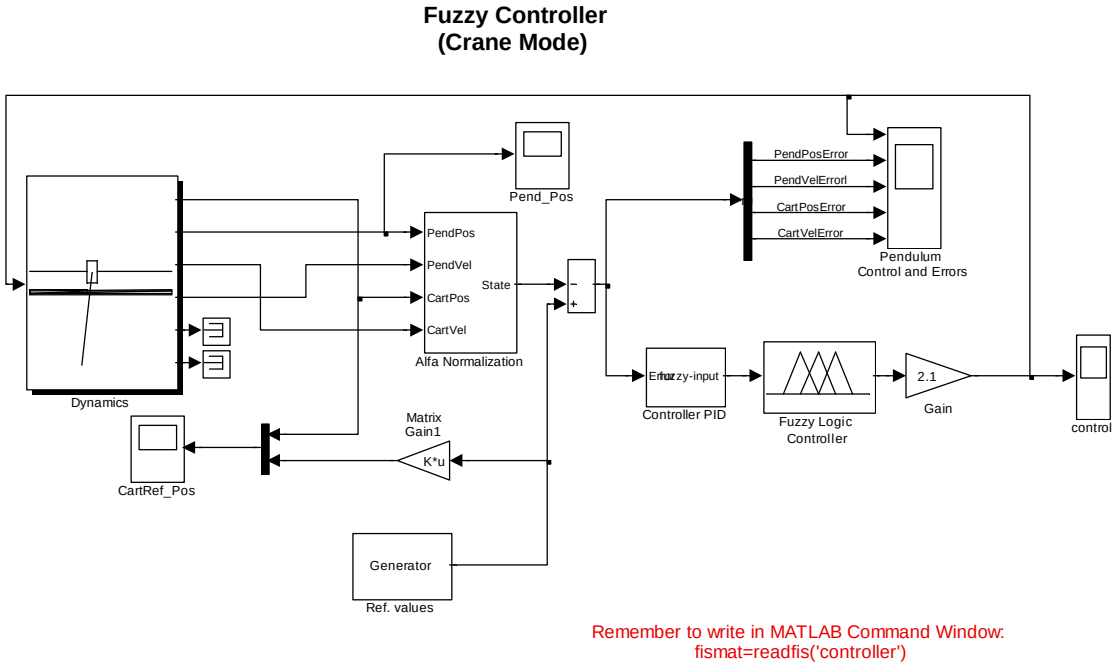


Fig. 11.3 Fuzzy simulation – the *p1_fuzzy_sim.mdl* crane controller model

Following the remark visible in the middle of the window we type:

```
>>fismat = readfis('controller');
```

at the MATLAB prompt. The *controller.fis* file (the so called FIS Matrix) is loaded from the disk to the *Fuzzy Logic Controller* block. If you double click the block its mask with the *fismat* name becomes visible (see Fig. 11.4).

Fig. 11.4 Mask of the *Fuzzy crane controller* block

The fuzzy controller is now available to be used in the *Fuzzy logic controller* block in a Simulink diagram. We start simulation as we usually proceed.

How to run the real-time experiment?

In the window given in Fig. 11.2 double click the red *Fuzzy Experiment* button to open the corresponding real-time model shown in Fig. 11.5.
Before starting experiment be sure that the hardware configuration is correct and that the *Reset encoder* action has been performed correctly.

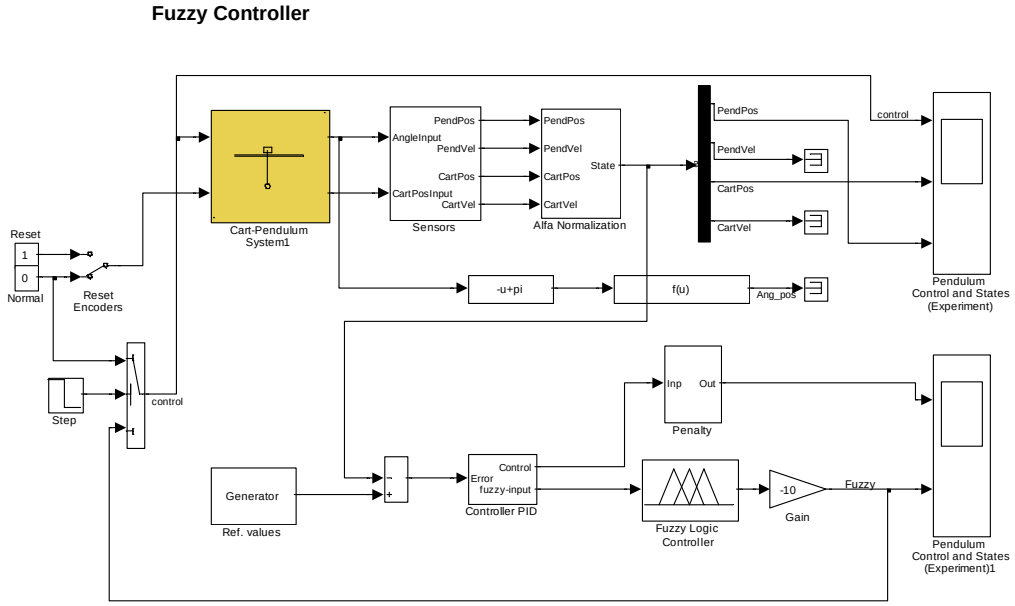


Fig. 11.5 Crane real-time controller

To run the experiment successfully we do not need to change the default settings. The parameter tuning is only required if the system responds wrongly. To create the real-time code of the given model build the model. Next, click the *Simulation/connect to target* and *Start real-time code* options to start the experiment.

11.2. Crane mode

The control problem is the following: steer the cart from an initial position on the rail to a given final position while dumping oscillations of the pendulum which occurs due to the cart motion.

The *FIS Editor controller* window is opened (Fig. 11.7). From this window you can go farther to watch and modify the fuzzy crane controller. We recommend you to use the editor. However, you can see or design a fuzzy controller without the help of the editor. This technique is presented alternatively in this section. Hence you can write:

```
>>a=readfis('controller');
```

It results in storing in the MATLAB Workspace under the *a* variable the *controller* fismatrix.

```
          name: 'controller'
          type: 'mamdani'
     andMethod: 'min'
      orMethod: 'max'
  defuzzMethod: 'centroid'
     impMethod: 'min'
     aggMethod: 'max'
         input: [1x4 struct]
        output: [1x1 struct]
          rule: [1x12 struct]
```

If you write:

```
» plotfis(a)
```

then the crane project is shown (given in Fig. 11.6). You can see the *controller* system with 4 inputs, 1 output, 12 rules. We incorporate an input variable into the project by defining rules corresponding to that variable. Otherwise the variable is left out. As one can notice the third input variable i.e. the pendulum velocity does not take part in building the fuzzy rules (see the dashed line in the crane project).

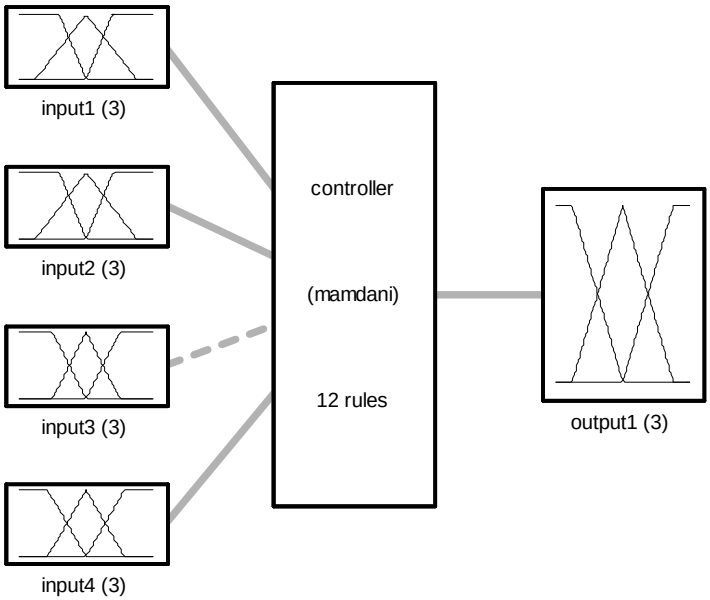


Fig. 11.6 Crane project

If you write:

```
» showfis(a)
```

the following message appears:

```
1.  Name            controller
2.  Type            mamdani
3.  Inputs/Outputs  [4 1]
4.  NumInputMFs     [3 3 3 3]
5.  NumOutputMFs    3
6.  NumRules        12
7.  AndMethod       min
8.  OrMethod        max
9.  ImpMethod       min
10. AggMethod       max
11. DefuzzMethod    centroid
12. InLabels        input1
13.                 input2
14.                 input3
15.                 input4
16. OutLabels       output1
17. InRange         [-5 5]
18.                 [-5 5]
19.                 [-4 4]
20.                 [-0.9 0.9]
21. OutRange        [-0.2 0.2]
22. InMFLabels      neg-cart
23.                 zer-cart
24.                 pos-cart
```

```
25.                   neg-pend
26.                   zer-pend
27.                   pos-pend
28.                   neg-pendvel
29.                   zer-pendvel
30.                   pos-pendvel
31.                   neg-cartvel
32.                   zer-cartvel
33.                   pos-cartvel
34. OutMFLabels       neg-control
35.                   zer-control
36.                   pos-control
37. InMFTypes         trapmf
38.                   trimf
39.                   trapmf
40.                   trapmf
41.                   trimf
42.                   trapmf
43.                   trapmf
44.                   trimf
45.                   trapmf
46.                   trapmf
47.                   trimf
48.                   trapmf
49. OutMFTypes        trapmf
50.                   trimf
51.                   trapmf
52. InMFParams        [-17.9 -17 -1.997 0]
53.                   [-3.75 0 3.75 0]
54.                   [-0.0265 1.96 16.97 17.9]
55.                   [-5 -5 -1.997 0]
56.                   [-3.75 0 3.75 0]
57.                   [0 1.997 5 5]
58.                   [-4 -4 -2 0]
59.                   [-2 0 2 0]
60.                   [0 2 4 4]
61.                   [-0.9 -0.9 -0.5143 0]
62.                   [-0.5143 0 0.5143 0]
63.                   [0 0.5143 0.9 0.9]
64. OutMFParams       [-0.2 -0.2 -0.15 0]
65.                   [-0.15 0 0.15 0]
66.                   [0 0.15 0.2 0.2]
67. Rule Antecedent   [1 1 0 0]
68.                   [2 2 0 0]
69.                   [3 3 0 0]
70.                   [1 2 0 0]
71.                   [3 2 0 0]
72.                   [2 1 0 0]
73.                   [2 3 0 0]
74.                   [1 3 0 0]
75.                   [3 1 0 0]
76.                   [0 0 0 1]
77.                   [0 0 0 2]
78.                   [0 0 0 3]
67. Rule Consequent   1
68.                   2
69.                   3
70.                   1
```

```
71.                  3
72.                  1
73.                  3
74.                  2
75.                  2
76.                  1
77.                  2
78.                  3
67. Rule Weigth      1
68.                  1
69.                  1
70.                  1
71.                  1
72.                  1
73.                  1
74.                  1
75.                  1
76.                  1
77.                  1
78.                  1
67. Rule Connection  1
68.                  1
69.                  1
70.                  1
71.                  1
72.                  1
73.                  1
74.                  1
75.                  1
76.                  1
77.                  1
78.                  1
```

If you double click the *controller.fis* file you obtain a similar data collected in a more compact form:

```
[System]
Name='controller'
Type='mamdani'
Version=2.0
NumInputs=4
NumOutputs=1
NumRules=12
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='input1'
Range=[-5 5]
NumMFs=3
MF1='neg-cart':'trapmf',[-17.9 -17 -1.997 0]
MF2='zer-cart':'trimf',[-3.75 0 3.75]
MF3='pos-cart':'trapmf',[-0.0265 1.96 16.97 17.9]

[Input2]
```

```
Name='input2'
Range=[-5 5]
NumMFs=3
MF1='neg-pend':'trapmf',[-5 -5 -1.9973544973545 0]
MF2='zer-pend':'trimf',[-3.75 0 3.75]
MF3='pos-pend':'trapmf',[0 1.9973544973545 5 5]

[Input3]
Name='input3'
Range=[-4 4]
NumMFs=3
MF1='neg-pendvel':'trapmf',[-4 -4 -2 0]
MF2='zer-pendvel':'trimf',[-2 0 2]
MF3='pos-pendvel':'trapmf',[0 2 4 4]

[Input4]
Name='input4'
Range=[-0.9 0.9]
NumMFs=3
MF1='neg-cartvel':'trapmf',[-0.9 -0.9 -0.5143 0]
MF2='zer-cartvel':'trimf',[-0.5143 0 0.5143]
MF3='pos-cartvel':'trapmf',[0 0.5143 0.9 0.9]

[Output1]
Name='output1'
Range=[-0.2 0.2]
NumMFs=3
MF1='neg-control':'trapmf',[-0.2 -0.2 -0.15 0]
MF2='zer-control':'trimf',[-0.15 0 0.15]
MF3='pos-control':'trapmf',[0 0.15 0.2 0.2]

[Rules]
1 1 0 0, 1 (1) : 1
2 2 0 0, 2 (1) : 1
3 3 0 0, 3 (1) : 1
1 2 0 0, 1 (1) : 1
3 2 0 0, 3 (1) : 1
2 1 0 0, 1 (1) : 1
2 3 0 0, 3 (1) : 1
1 3 0 0, 2 (1) : 1
3 1 0 0, 2 (1) : 1
0 0 0 1, 1 (1) : 1
0 0 0 2, 2 (1) : 1
0 0 0 3, 3 (1) : 1
```

Thus you obtain the entire description of the *controller* system. All names, shapes of membership functions, rules, etc. are visible in numerical forms. You can trace these data in the *Fis Editor* window (Fig. 11.7). You can edit membership functions or rules. You can view rules or surface of the controller.

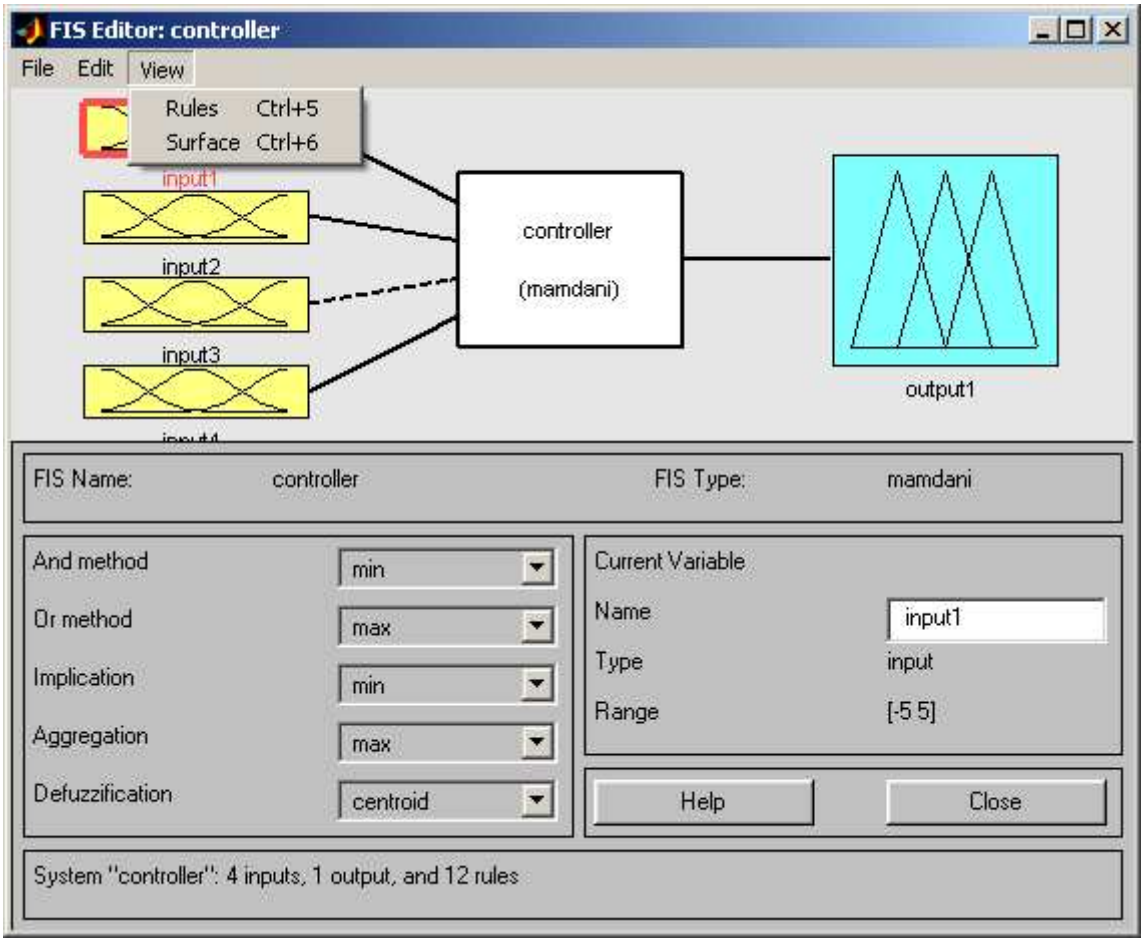
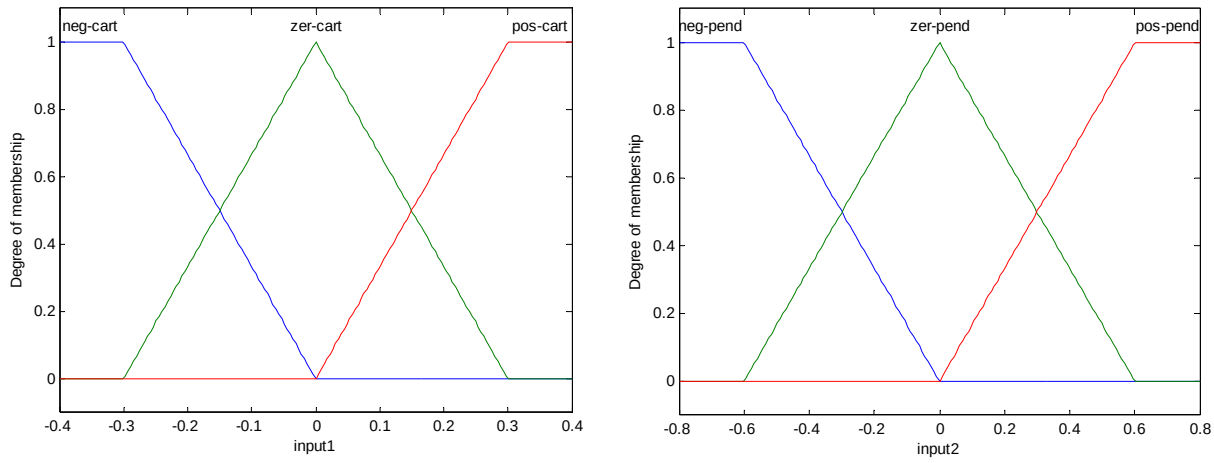


Fig. 11.7 The *Fis Editor*: *Controller* window

The membership functions are shown in Fig. 11.8. One can invoke these figures writing in MATLAB Command Window:

```
plotmf(a,'input',1)
```

This corresponds to the first input.



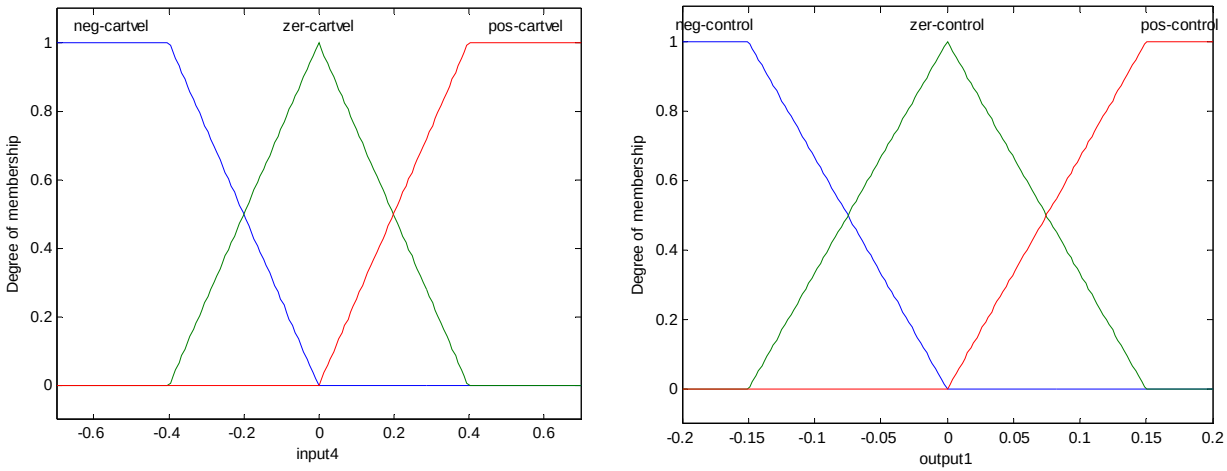


Fig. 11.8 Membership functions

The last picture is obtained by the command:

```
plotmf(a,'output',1).
```

We set them to be equally distributed in the universe of discourse. The rules are given in Fig. 11.9. The *If...then* inference mechanism is well visible in the verbose form. The index form of rules is an equivalent form of description. The index (1) in the first rule means that the weight of this rule is set equal to 1.The index form: 2 0 0 0, 7 (1) : 1 can be explained as follows: the 2nd input membership function for the first input variable **and** none "0" input membership function for the second input variable **and** none "0" input membership function for third input variable **and** none "0" input membership function for fourth input variable implies 7 output membership function for the output variable. The weight of this rule is set to one (1). The "and" logic quantificator is defined as ":1".

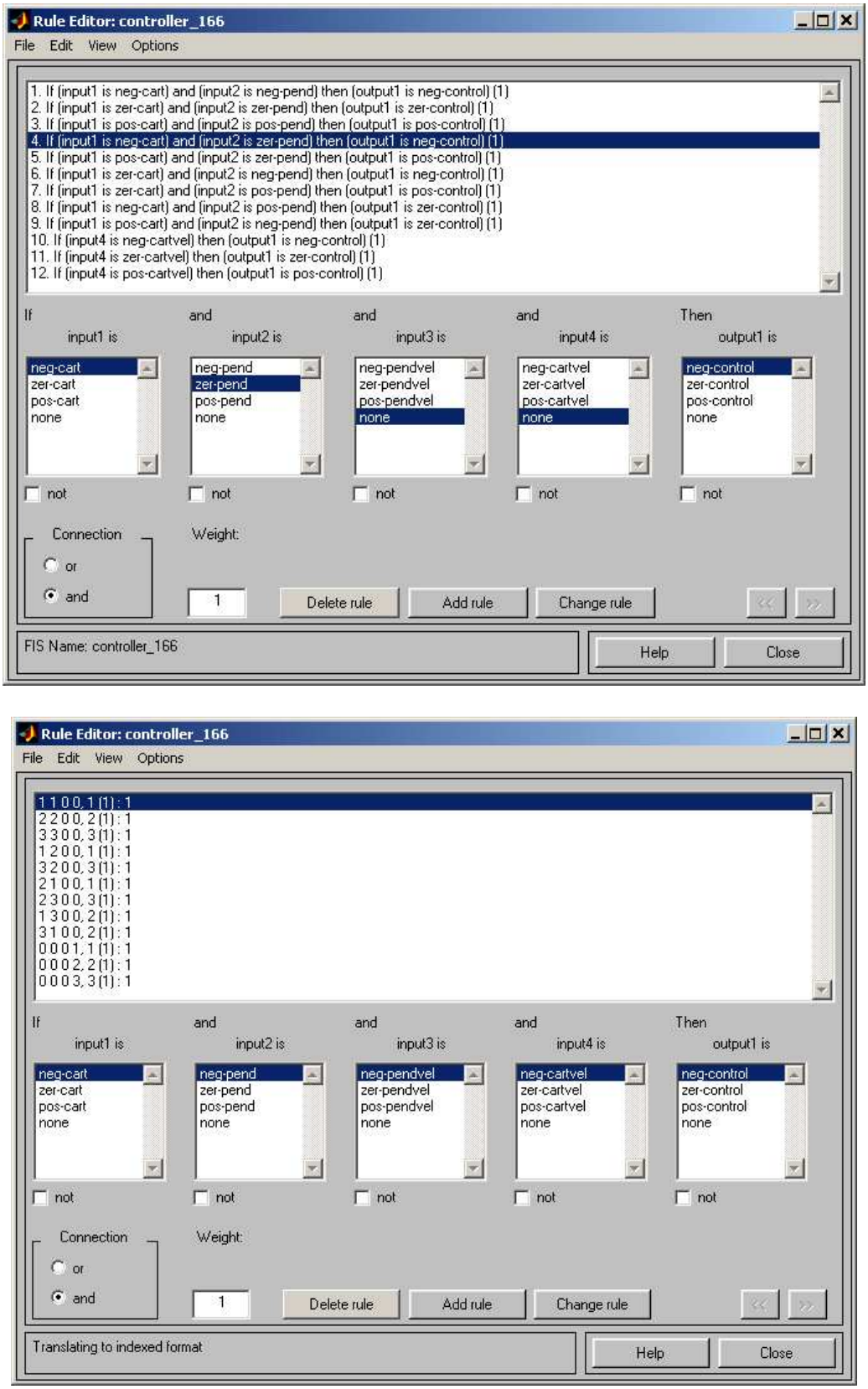


Fig. 11.9 Rules for the fuzzy crane controller: VERBOSE and INDEX forms

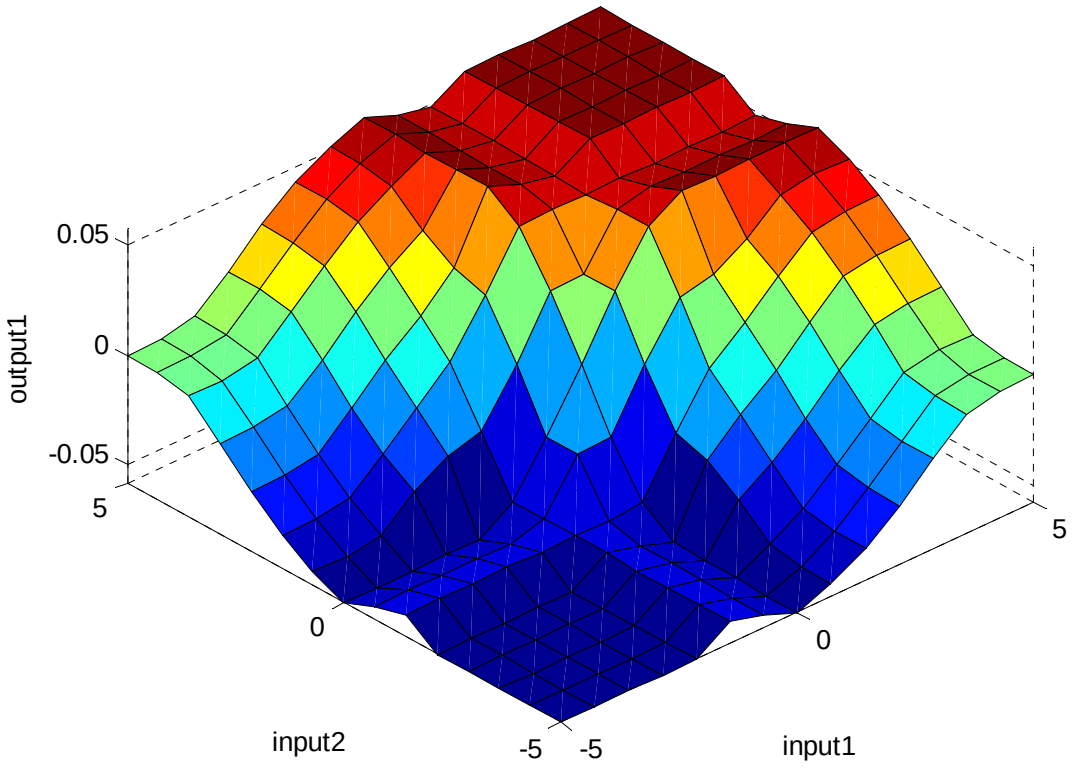


Fig. 11.10 Fuzzy surface

The *control* vs. *position* and *angle* surface is given in Fig. 11.10.

11.2.1. Simulation: example

The example demonstrates how the cart-pendulum system can be used as a crane control model. The control goal is to steer the cart from an initial position on the rail to a given final position while dumping oscillations of the pendulum. The oscillations occur due to the cart motion.
Double click the *Simulation* button shown in Fig. 11.2. The model depicted in Fig. 11.3 opens.

Remember to write:

```
>>fismat=readfis('controller');
```

at the *MATLAB* prompt. Check the desired position signal and the initial values of the state variables.

The simulation results of the crane mode are given in Fig. 11.11 and Fig. 11.12. Notice, that the cart tracks the desired position square-wave signal while the pendulum motion is dumped.

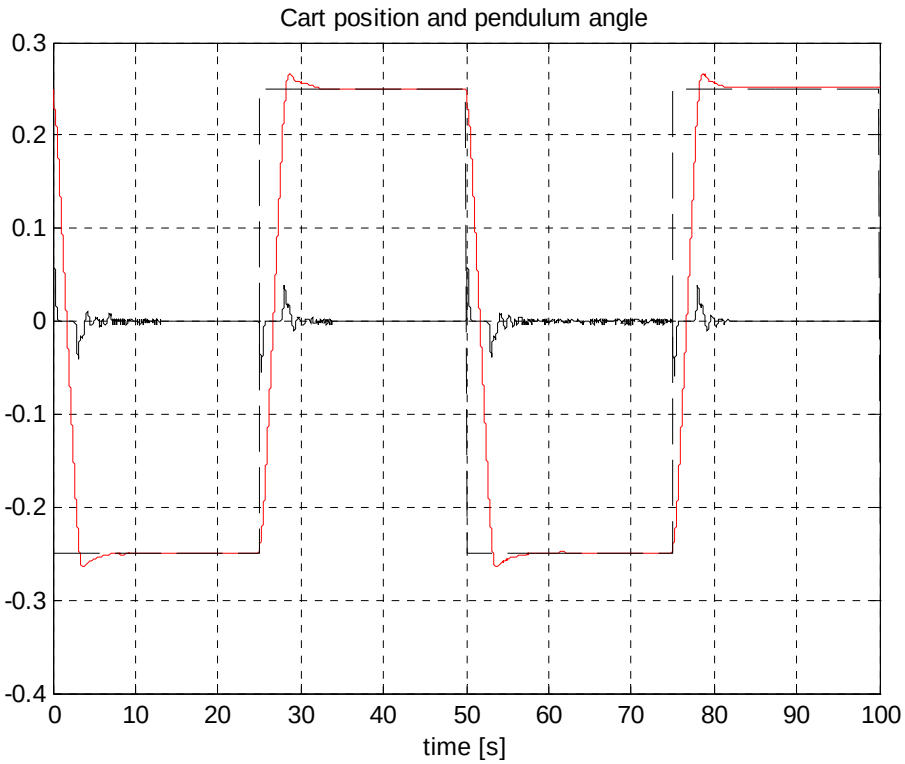


Fig. 11.11 Data stored during the simulation: the cart position (the red line), reference cart position and pendulum angle (the signal at a small vicinity of zero)

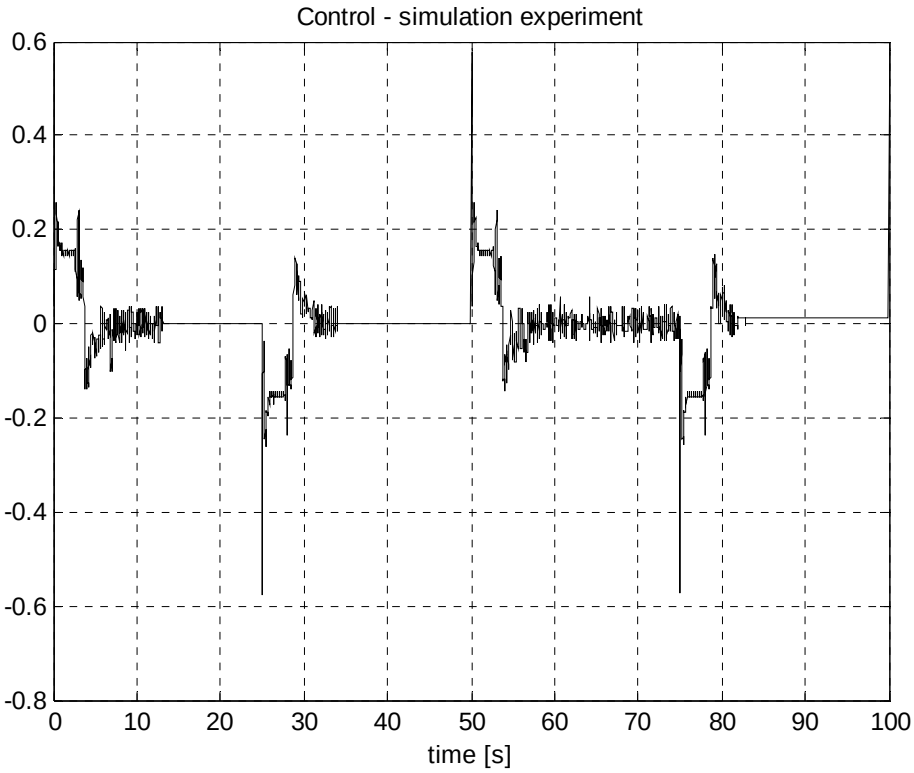


Fig. 11.12 Simulation – the control signal

11.2.2. Real-time experiment

After simulation we can go directly to the real-time experiment. Double click the *Fuzzy Experiment* button shown in Fig. 11.2. The real-time model opens (see Fig. 11.5). From that window you can start the real-time experiment. The goal of the fuzzy controller is to track the desired position signal generated by the *Ref. values* Simulink generator block. The *controller.fis* is the name of the fuzzy controller. In Fig. 11.13 are shown data measured during the real time experiment. These are: the cart position signal following the square wave desired position signal and the pendulum angle – the signal in a vicinity of the zero angle. **Remark.** The angle of the pendulum hanging down is defined as the zero value.

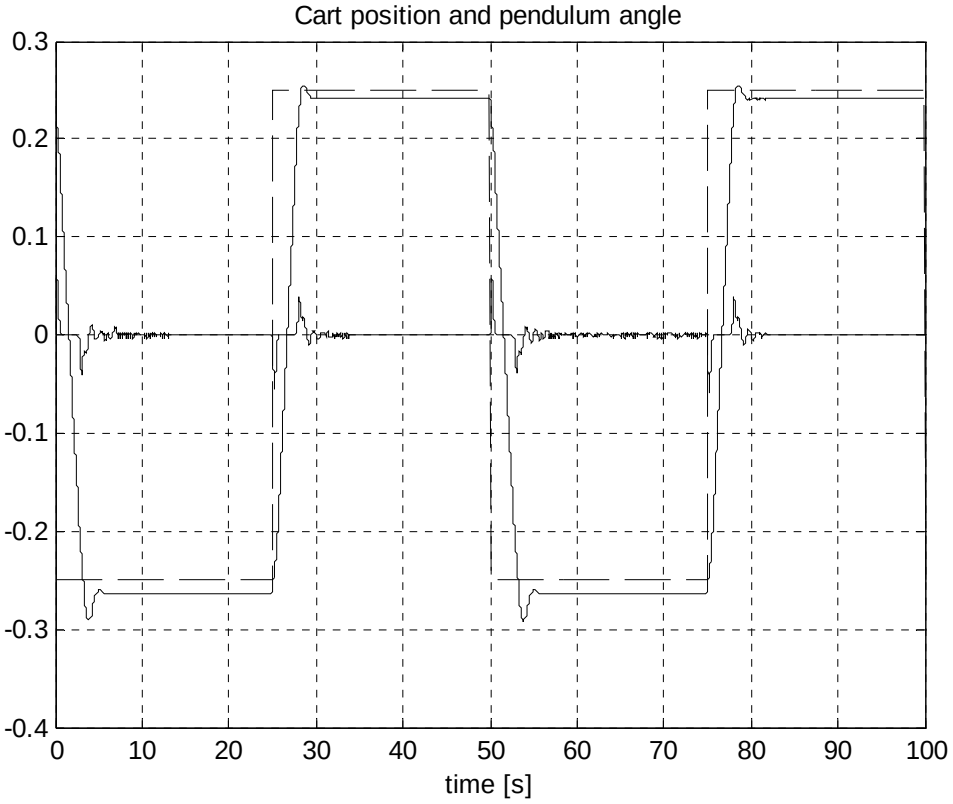


Fig. 11.13 Data stored during the real-time experiment: the cart position, reference cart position and pendulum angle

The fuzzy controller is tracking the square wave signal of the desired cart position. At the beginning the pendulum is hanging down. Due to the cart motion the pendulum starts to swing. The fuzzy controller dumps oscillation of the pendulum. Notice, the similarities between the simulation and real-time control signals.

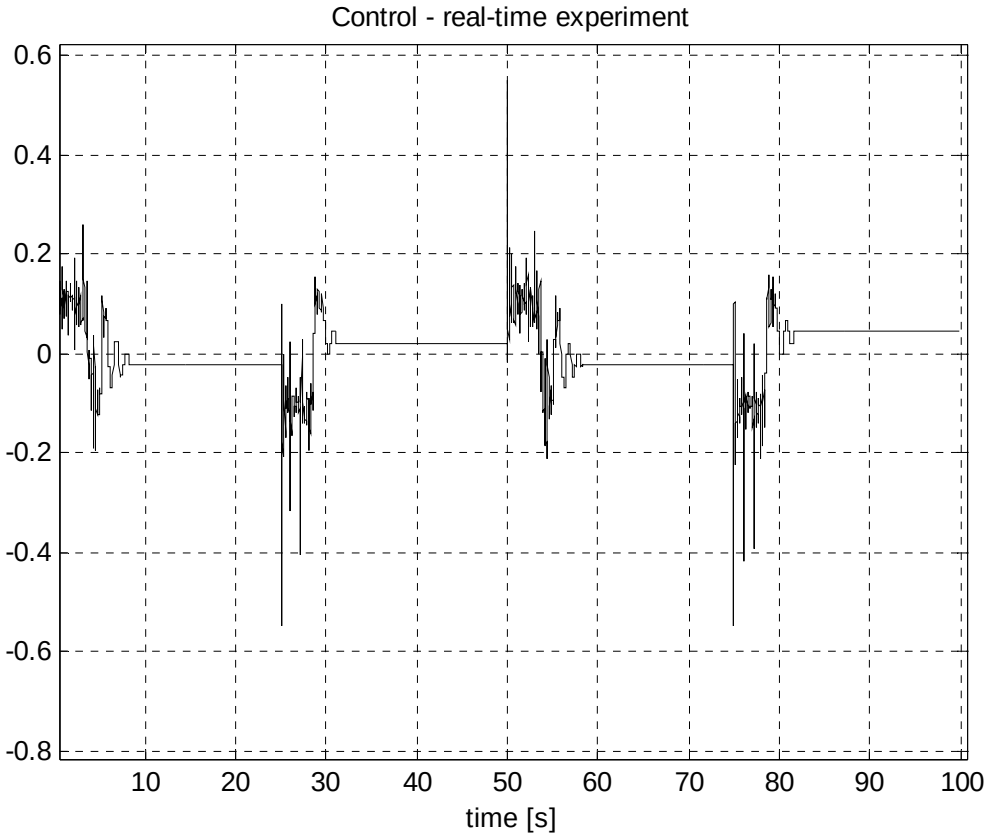


Fig. 11.14 Real-time experiment – the control signal

12. How to fulfil the compilation settings page

In

Fig. 12.1 and Fig. 12.2 the different *Simulation Parameters* pages are shown. Having the MATLAB version and the compiler version the user has to choose the appropriate *Simulation Parameters* page version.

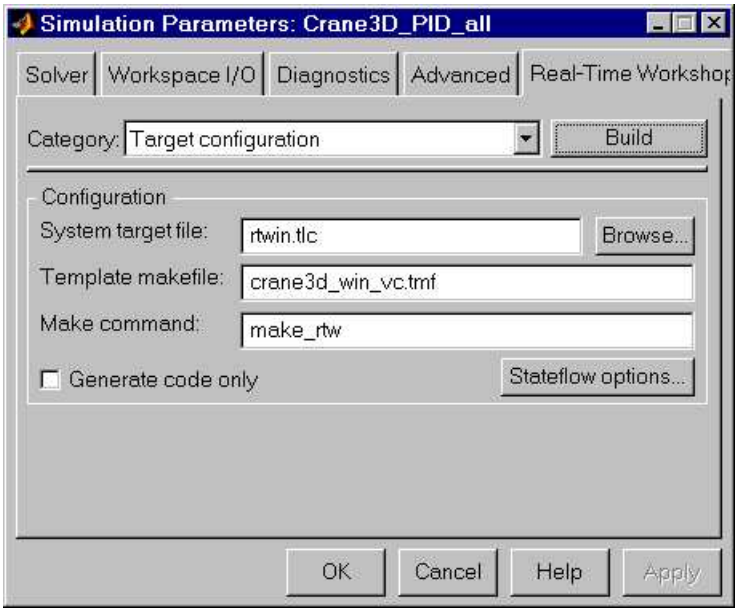


Fig. 12.1 *Simulation parameters* page for the MATLAB ver. 6.5 and Visual C++ compiler

In the case when Matlab version 7.0.4 is used the Open Watcom compiler is applied. This compiler is placed on Matlab installation CD and is installed automatically.
The appropriate *Configuration Parameters* page is shown in Fig. 12.2.

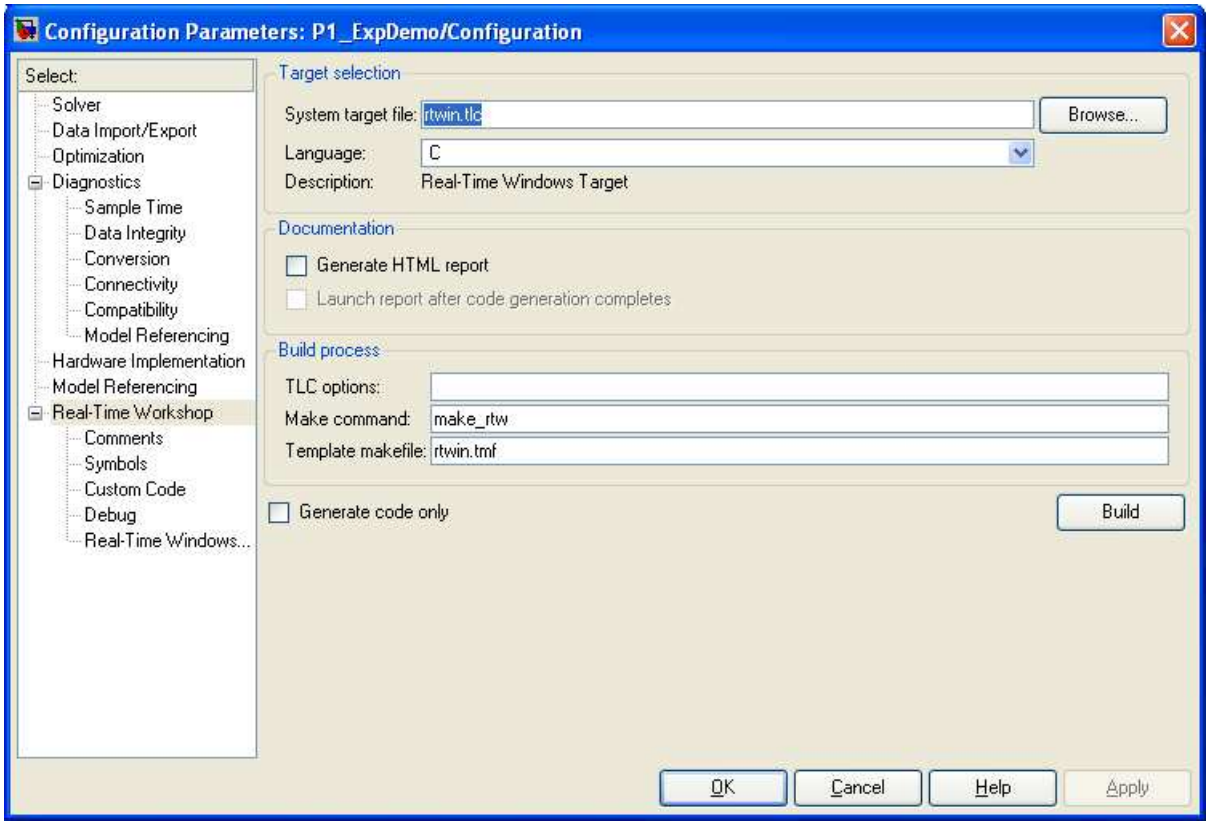


Fig. 12.2 *Configuration Parameters* page for the MATLAB ver. 7 (R14 SP2)