

3DCrane

User's Manual

PCI version



www.inteco.com.pl

COPYRIGHT NOTICE

© Inteco Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of Inteco Ltd.

ACKNOWLEDGEMENTS

Inteco Ltd acknowledges all trademarks.
IBM, IBM - PC are registered trademarks of International Business Machines.
MICROSOFT, WINDOWS are registered trademarks of Microsoft Corporation.
MATLAB, Simulink and RTW are registered trademarks of Mathworks Inc.

Contents

1. INTRODUCTION AND GENERAL DESCRIPTION	5
1.1. Product overview	7
1.2. Requirements	8
2. SOFTWARE INSTALLATION.....	9
2.1. Installation	9
2.2. Uninstallation.....	11
3. STARTING, TESTING AND STOPPING PROCEDURES	13
3.1. Starting procedure.....	13
3.2. Testing and troubleshooting.....	13
3.3. Stopping procedure.....	18
4. MAIN CONTROL WINDOW	19
4.1. Tools	19
4.2. Drivers	26
4.3. Demo Controllers.....	28
5. YOUR FIRST REAL-TIME CONTROL EXPERIMENT.....	32
5.1. Real-time experiment.....	37
5.1.1. Data processing	37
5.2. Simulation.....	40
5.3. PID control of load position.....	43
6. CASE STUDY.....	46
7. PROTOTYPING YOUR OWN CONTROLLER IN RTWT ENVIRONMENT	61
7.1. Creating a model.....	62
7.2. Code generation and the build process	64
8. MATHEMATICAL MODEL OF THE 3DCRANE.....	67
8.1. Basic relationships	68
8.2. Simplified model with three control forces.....	69
8.3. Complete nonlinear model with constant pendulum length and two control forces	71
8.4. Complete nonlinear model with varying pendulum length and three control forces	72
9. DESCRIPTION OF THE CRANE3D CLASS PROPERTIES	74
9.1. BaseAddress	75
9.2. BitstreamVersion	75
9.3. Encoder.....	76
9.4. PWM.....	76
9.5. PWMPrescaler.....	76
9.6. ResetEncoder	77
9.7. RailLimit.....	77
9.8. RailLimitFlag.....	77
9.9. RailLimitSwitch.....	78
9.10. ResetSwitchFlag	78
9.11. Therm	78
9.12. ThermFlag	78
9.13. Time.....	79
9.14. Quick reference table.....	79

3DCrane

The industrial crane model controlled from PC

A tool for control education and research

1. Introduction and general description

The 3DCrane is a non-linear electromechanical system having a complex dynamic behaviour and creating challenging control problems. The system is controlled from a PC. Therefore it is delivered with hardware and software which can be easily mounted and installed in a laboratory. You obtain the mechanical unit with power supply and interface to a PC and the dedicated A/D, D/A board configured in the Xilinx[®] technology. The software operates under MS Windows[®] NT using MATLAB[®] and RTWT toolbox package.

Besides the hardware and the related software you obtain the *User's Manual*. The manual

- shows step-by-step how to design and generate your own real-time controller in MATLAB[®]/Simulink[®] environment,
- contains the library of ready-to use real-time controllers,
- includes the set of preprogrammed experiments.

The **3DCrane setup** (Fig. 1.1) consists of a payload hanging on a pendulum-like lift-line wound by a motor mounted on a cart.

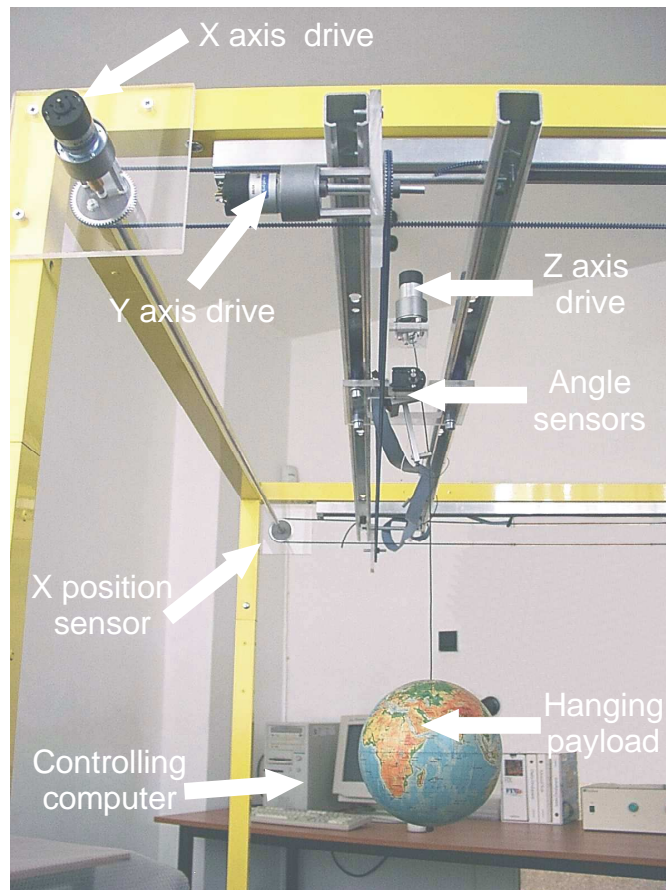


Fig. 1.1 The 3DCrane setup

The payload is lifted and lowered in the z direction. Both the rail and the cart are capable of horizontal motion in the x direction. The cart is capable of horizontal motion along the rail in the y direction. Therefore the payload attached to the end of the lift-line can move freely in 3 dimensions. The 3DCrane is driven by three DC motors.

There are five identical measuring encoders measuring five state variables: the cart coordinates on the horizontal plane, the lift-line length, and two deviation angles of the payload. The encoders measure movements with a high resolution equal to 4096 pulses per rotation (ppr). These encoders together with the specialised mechanical solution create a unique measurement unit. The deviation of the load is measured with a high accuracy equal to 0.0015 rad.

The power interface amplifies the control signals which are transmitted from the PC to the DC motors. It also converts the encoders pulse signals to the digital 16-bit form to be read by the PC.

The PC equipped with the RT-DAC/PCI multipurpose digital I/O board communicates with the power interface board. The whole logic necessary to activate and read the encoder signals and to generate the appropriate sequence of pulses of PWM to control the DC motors is configured in the Xilinx[®] chip of the RT-DAC/PCI board. All functions of the board are accessed from the 3DCrane Toolbox which operates directly in the MATLAB[®]/Simulink[®] environment.

KEY FEATURES

- Three-dimensional laboratory model of industrial crane.
- The model can be tailored according to user's size requirements.
- A highly nonlinear MIMO system.
- The system can be easily installed.
- There are high-resolution sensors – unique 2D angle measuring unit.
- The set-up is fully integrated with MATLAB[®]/Simulink[®] and operates in real-time in MS Windows.
- Real-time control algorithms can be rapidly prototyped. No C code programming is required.
- The software includes complete dynamic models.
- The *User's Manual*, library of basic controllers and a number of pre-programmed experiments familiarise the user with the system in a fast way.
- 3DCrane is ideal for illustrating complex nonlinear control algorithms.

1.1. Product overview

The crane is delivered in partially mounted form. The mounting frame makes a support and a flexible construction of the system. If it is fixed to the walls and the floor then the crane becomes a rigid construction. The construction is available in user-defined sizes. The dimensions are from the range: length from 1 m to 1.5 m, width from 1 m to 1.5 m. The standard size is length/width/high: 1.0 x 1.0 x 1.0 m.

SETUP COMPONENTS

hardware

- mechanical unit

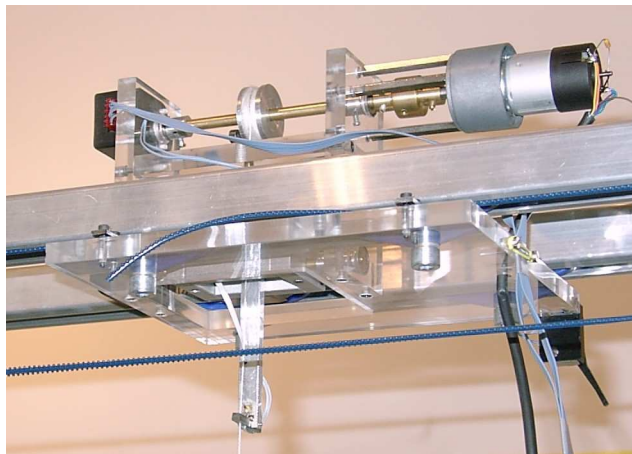


Fig. 1.2 Cart and 2D angle measuring unit

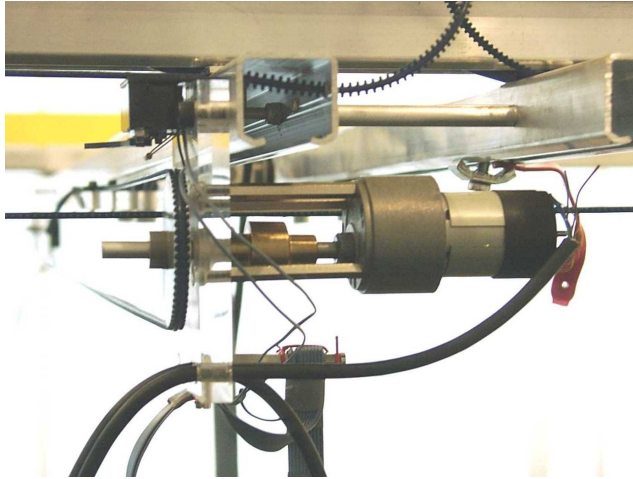


Fig. 1.3 X axis drive

- interface and Power Interface Unit
- RT-DAC/PCI I/O board (the PWM control logic is stored in a XILINX chip)

software

- 3DCrane Control/Simulation Toolbox operating in MATLAB[®]/Simulink[®] environment
- manuals
 - *Installation Manual*
 - *User's Manual*

1.2. Requirements

- Pentium or AMD based personal computer
- Microsoft Windows XP/7 x 86
- MATLAB version 32 bit with Simulink, RTW and RTWT toolboxes (not included)
- or
- Microsoft Windows XP/7 x 64
- MATLAB version 64 bit with Simulink, RTW and RTWT toolboxes (not included),



Details of the required software are available at:

http://www.inteco.com.pl/support/Software_requirements.pdf

2. Software Installation

2.1. Installation



The installation program has to be started by a system administrator.

To start the installation program insert the CD-ROM into the drive, and run *manager.exe* placed in the main directory. From the INTECO Software Manager application window, select 3DCrane Toolbox Installation. You will see the dialogue window (Fig. 2.1).

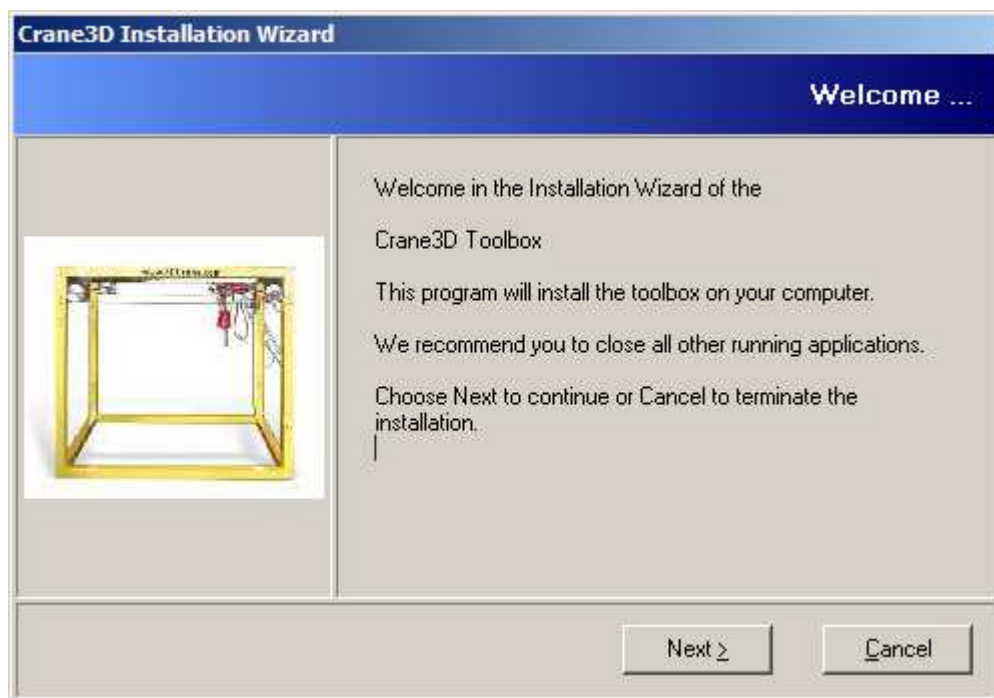


Fig 2.1

If you select the *Next* option), you will see the licence information. Read the licence agreement carefully. If you accept the licence terms click the *Next* button.

The window which follows asks you for your name and the name of your company (Fig. 2.1). If you want to install the software for your company you must check the *My Company* radio button.



Fig. 2.1

If you click the *Next* button the information you entered before will be displayed. Click *Next* to continue. You will see an important dialogue window containing your current MATLAB settings (Fig. 2.2). You will be informed about the version of your Windows system.

You must select the appropriate version of the MATLAB software installed. Next, point out the location of the matlabroot directory. For this purpose click the *Browse* button. You will see the dialogue box with drives and directory list displayed (see Fig. 2.3).

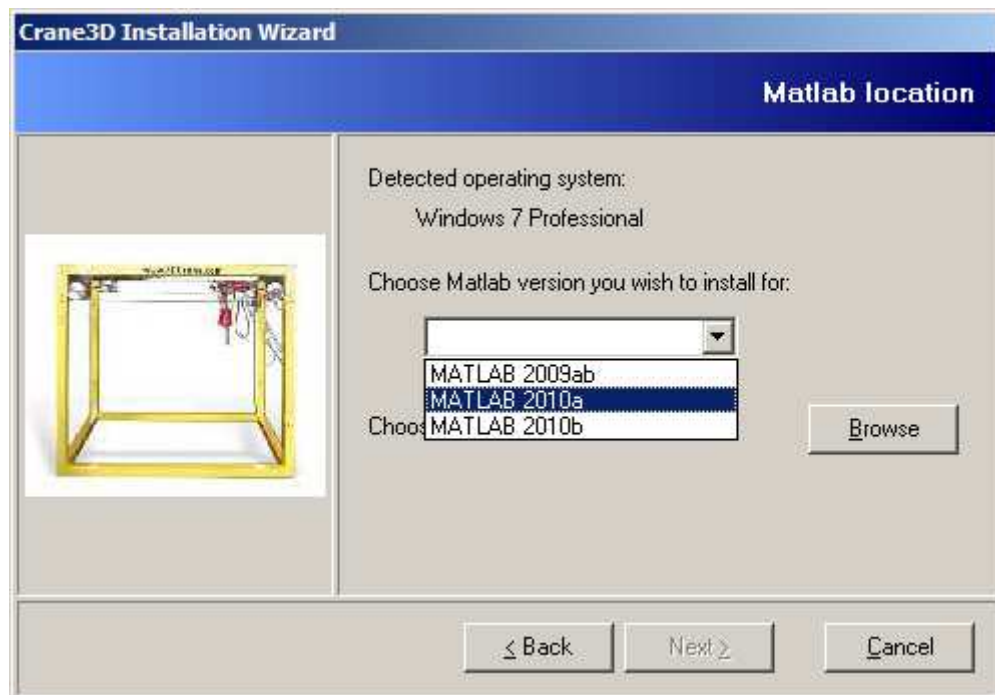


Fig. 2.2

Browse your system directories and select the appropriate directory. The program automatically detects it and closes the dialogue window. The selected location will be displayed in the *Installation setting* window.

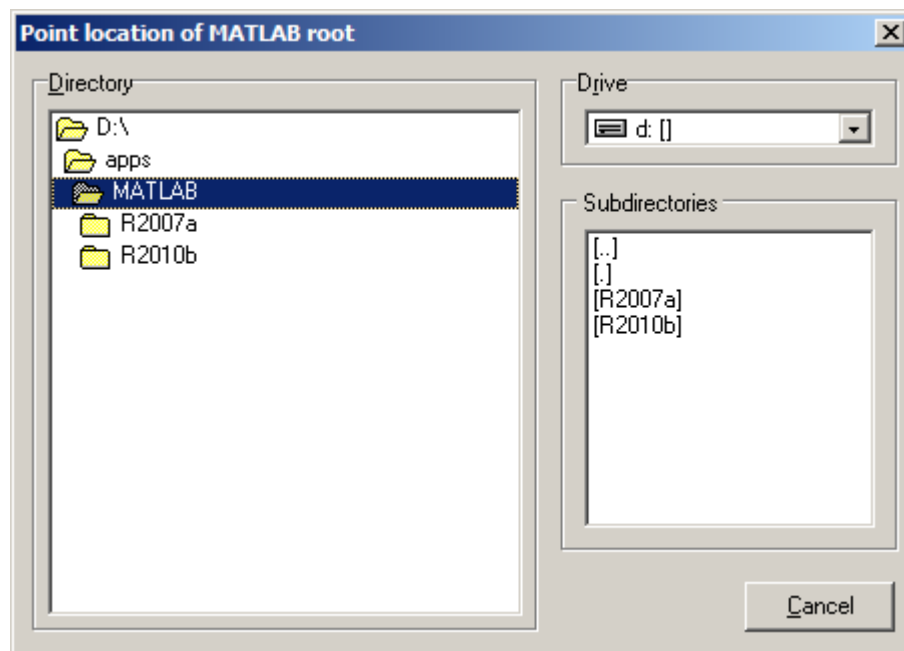


Fig. 2.3

If you have completed installation settings click the *Next* button to start the installation procedure. You will see the progress window, containing information about installation progress and names of files currently being copied.

The installation program installs the Windows NT kernel-mode device driver. This action is performed only if the driver does not exist in the system. The application will copy the driver to your system and modify system settings. Next you will be asked to restart your system. Click the *Yes* button to restart your system.

2.2. Uninstallation



The uninstallation program has to be started by a system administrator.

To start the uninstallation program insert the CD-ROM into the drive, and run *manager.exe* placed in the main directory. From the INTECO Software Manager application window select 3DCrane Toolbox Uninstallation. In the opened window click the *UNINSTALL* button to start the uninstallation program, or choose *EXIT* to quit.

If you choose the *UNINSTALL* option you will see the dialogue window (see Fig. 2.4) containing information about the installed components. If you want to remove an item select it on the list and click the *Uninstall* button, otherwise click the *Exit* button. The list box contains all 3DCrane Toolboxes installed on your computer. On the left side of the item the check box is visible. To uninstall an item the check box must be selected.

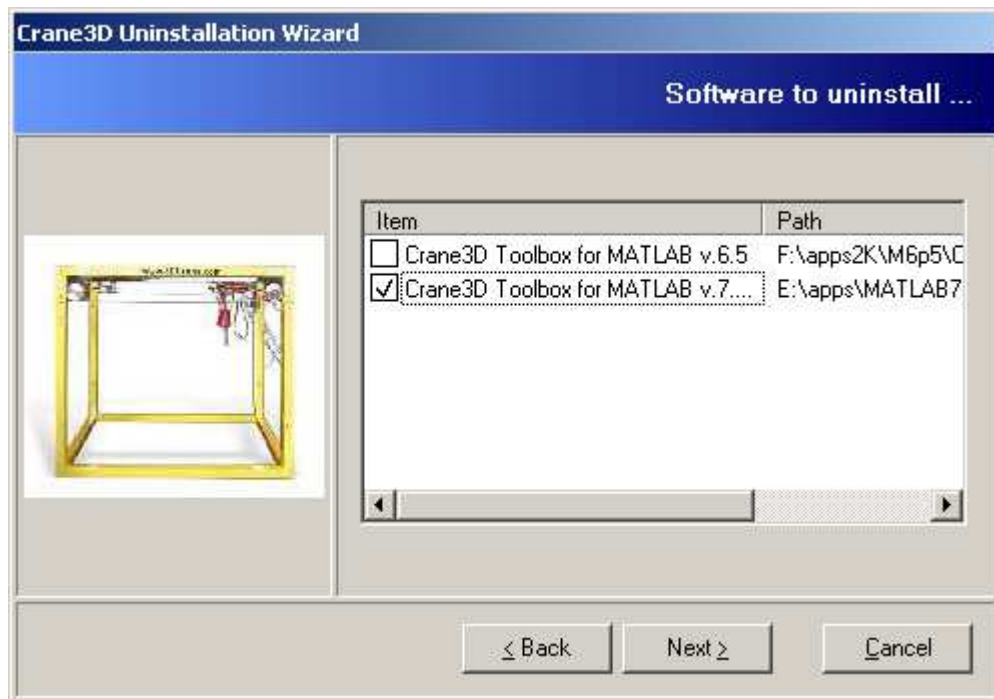


Fig. 2.4

Click the *Next* button.

3. Starting, testing and stopping procedures

3.1. Starting procedure

Combine the acquisition board, Control Interface and 3DCrane together. In the MS-WINDOWS environment invoke MATLAB by double clicking on the MATLAB icon. The MATLAB command window opens. Then simply type:

cr

MATLAB brings up the window 3DCrane *Main Control Window* (see Fig. 3.1). Pushbuttons indicate an action that executes callback routines when the user selects a menu item.

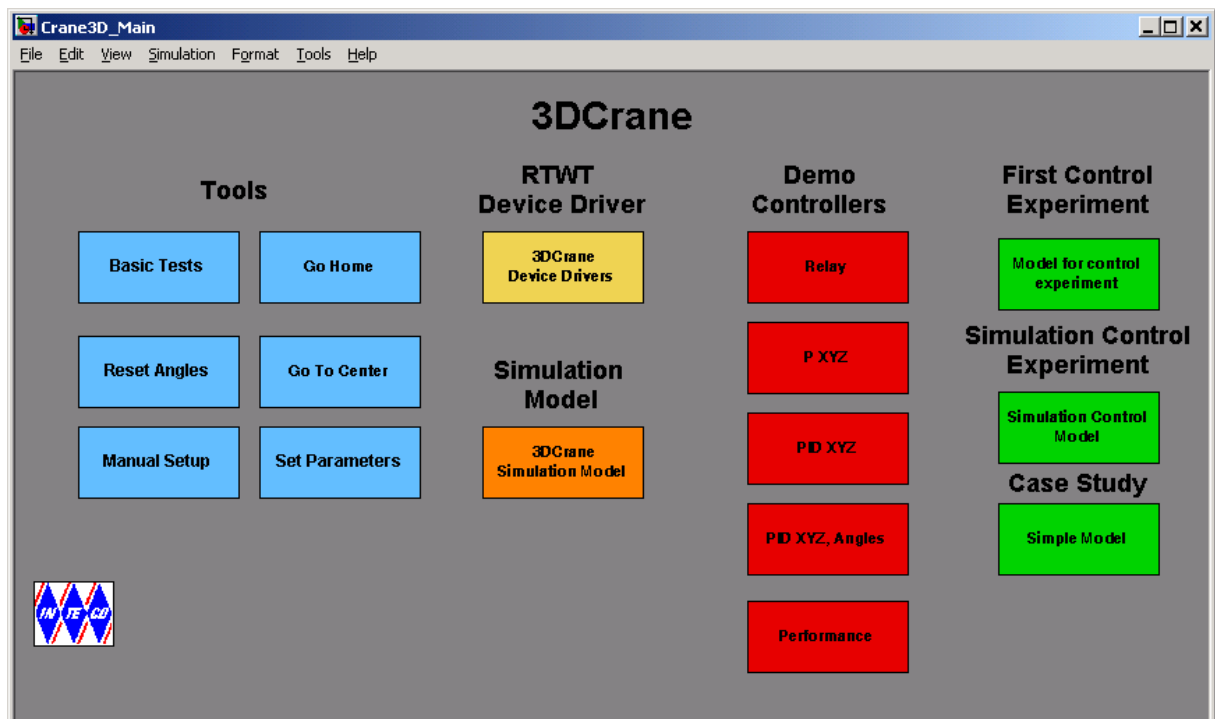


Fig. 3.1 The *Main Control Window* of the 3DCrane system

The *Main Control Window* contains testing tools, drivers, models and demo applications. Also a case study is included.

You can see a number of pushbuttons ready to use.

3.2. Testing and troubleshooting

This section explains how to perform the tests. These tests allow checking if mechanical assembling and wiring has been done correctly. The tests have to be performed obligatorily after assembling the system. They are also necessary after an incorrect operation of the system. The tests are helpful to look for causes of errors when the system fails. The tests have been designed to validate the existence and sequence of measurements and controls. They do not relate to accuracy of the signals.

In this manual some terms are used which describe the location of the cart. Fig. 3.2 defines these terms.

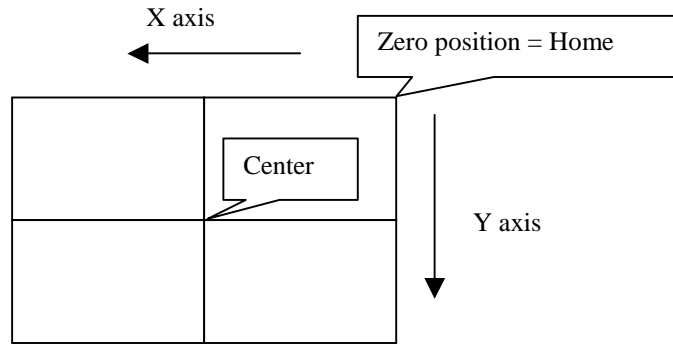


Fig. 3.2 View of the workspace of 3DCrane

First, you have to be aware that all signals are transferred in a proper way. Eleven checking steps are applied.



Before starting the test move the cart and the rail manually to an arbitrary position different from the zero position (Fig. 3.2).

- Double click the *Basic Tests* button. The following window appears (Fig. 3.3):

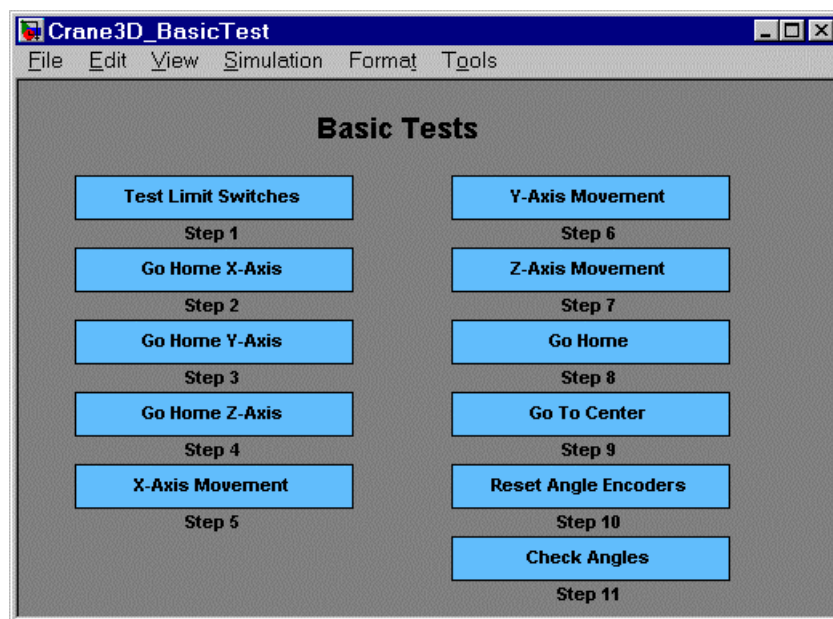


Fig. 3.3 The *Basic Tests* window

The first step in testing the crane is to check the proper operation of the limit switches. There are three switches applied to stop the moving parts of the system and to secure the system against destruction if the cart or the rail approaches the limits.

- Double click the *Test limit switches* button. The window presented in Fig. 3.4 opens:

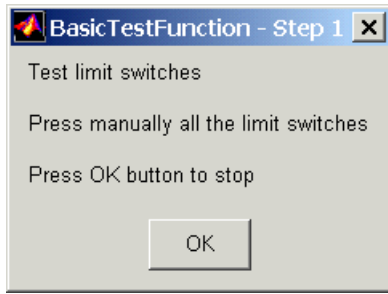


Fig. 3.4 *Test limit switches* window



Fig. 3.5 *Switch detected* window

Then close manually one by one all switches related to x, y and z-axes. After each closing you hear a sound signal. If you switch on the x-axis limit switch then the window presented in Fig. 3.5 appears. This means that the switch works properly. Close the window – click the *OK* button. When a switch is not detected please check connection of the appropriate cables to the undetected switch.

Next, you can check if the cart, rail and payload move in the right direction and if the system stops at the desired limit position. The system is moved in the chosen direction until it reaches the zero position (at this point the switch limit must be active).

- Double click the *Go Home X-axis (Y-axis and Z-axis)* button and observe the behaviour of the system. The window (Fig. 3.6) opens. You can interrupt the motion clicking the *OK* button.

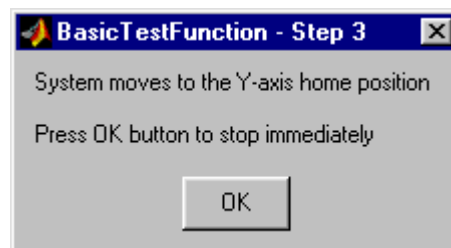


Fig. 3.6 *Go Home Y-axis* window

After performing tests along three directions the system is stopped at the zero position. The encoders of X,Y and Z-axes are automatically reset to zero value.

If motion in a given direction is not observed check wires and plugs related to that direction.

The next three steps perform the change of the system position from the initial position to the initial + 0.3 [m] position along a selected direction.

- Double click the *X-axis (Y-axis and Z-axis) Movement* button. The window (Fig. 3.7) opens where you can stop the motion clicking the *OK* button.

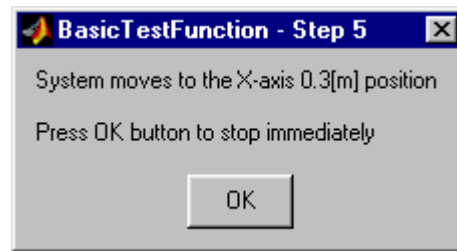


Fig. 3.7 X-axis movement window

Click the *OK* button and the plot of the movements appears (Fig. 3.8).

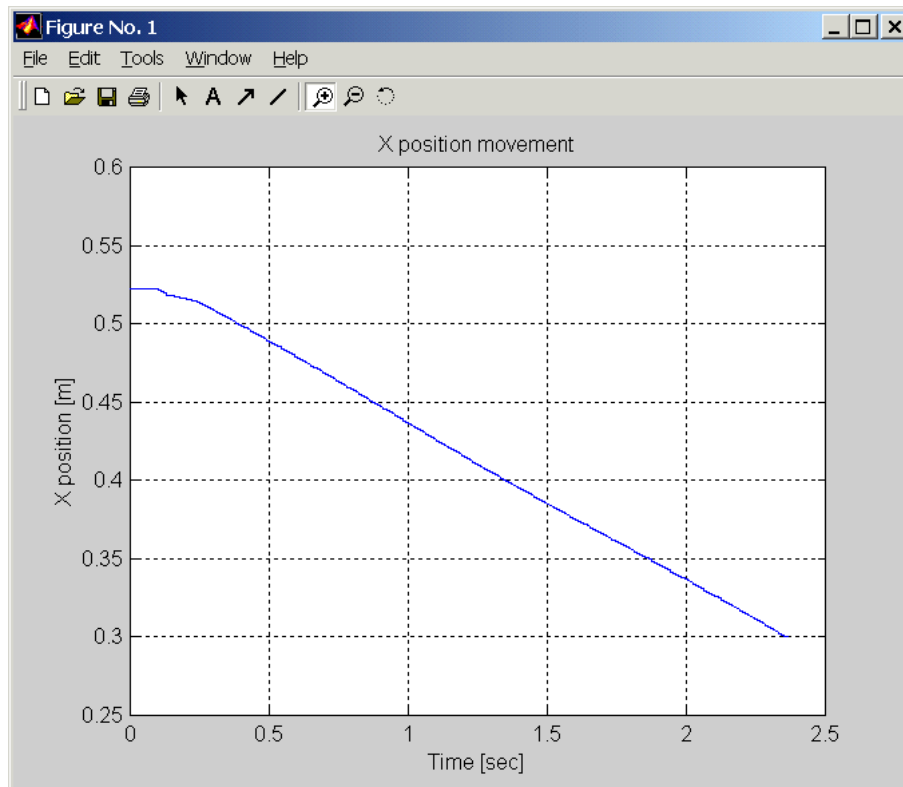


Fig. 3.8 Plot of the X-axis test movement

- In the next step double click the *Go To Center* button. The system moves to position in the center of the physical system workspace. Workspace boundaries are limited by sizes of the laboratory set and fixed in the program. The window presented in Fig. 3.9 opens



Fig. 3.9 Go to Center window

After clicking the *OK* button the plot of the movement is displayed (Fig. 3.10)

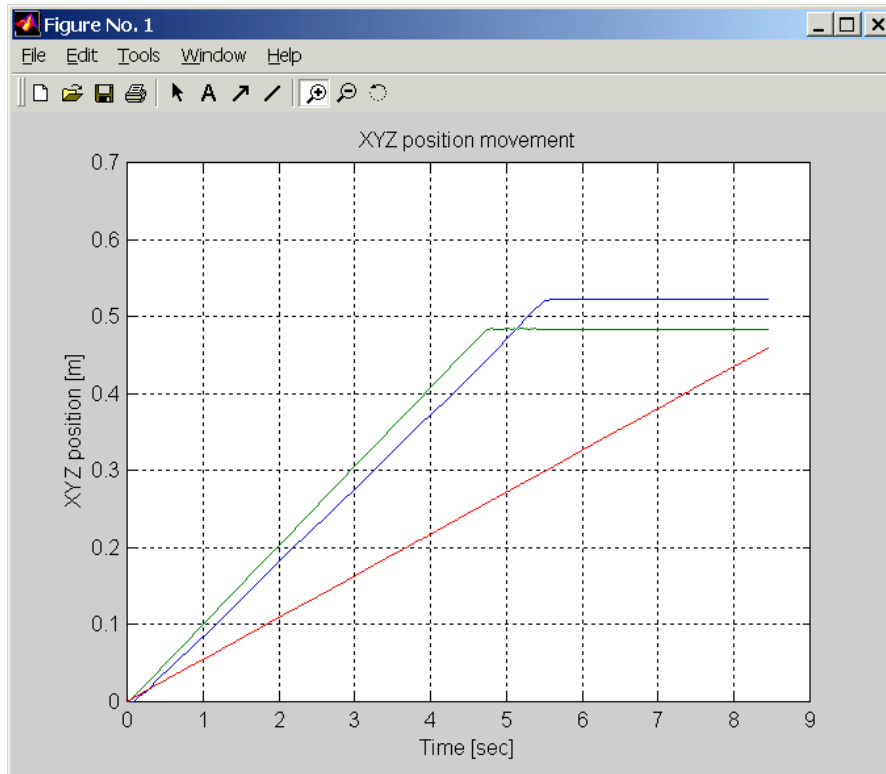


Fig. 3.10 Plot of the movement to the center

Notice that the center point was not exactly reached. This is due to the open loop control mode. The control signal is switched off when the system exceeds the center point.

Two next check steps are related to angle measurements.

- Double click the *Reset Angle Encoders* button. The window shown in Fig. 3.11 opens.

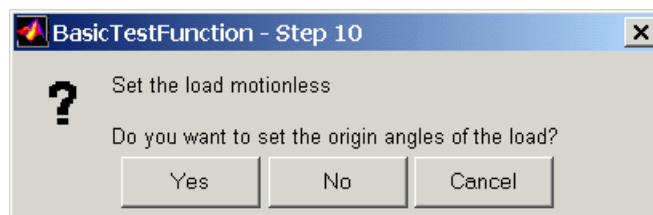


Fig. 3.11 *Reset Angle Encoders* window

Now you must set the load motionless and click *Yes*. The angle encoders are reset and the zero position is memorised by the system.

- To check if angle measurements are correct double clicks the *Check Angles* button. The window presented in Fig. 3.12 opens. Then manually move the load to a non-zero position and push it. After that click *OK* button and observe the motion on the screen (Fig. 3.13)

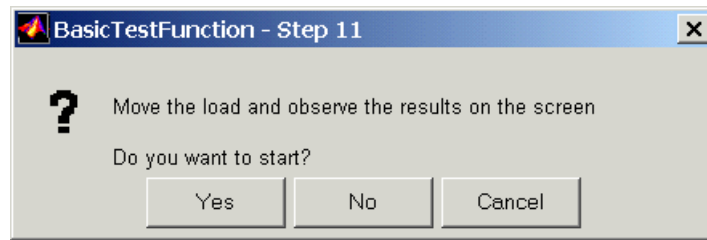


Fig. 3.12 Angle measurement observation window

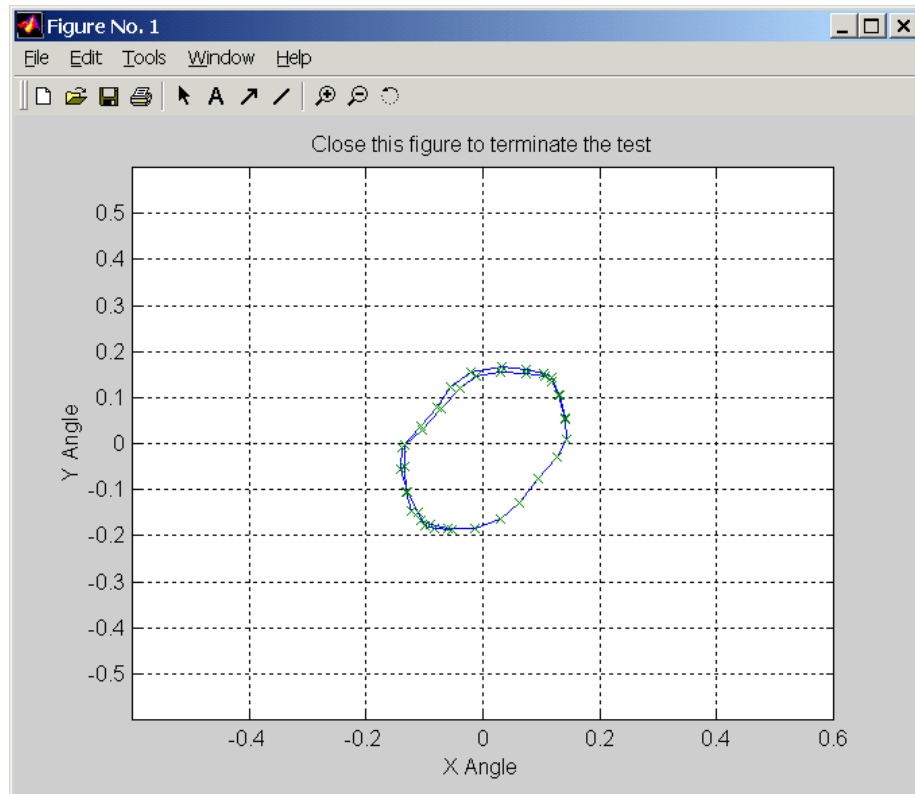


Fig. 3.13 Plot of the angles trajectory

3.3. Stopping procedure

The system is equipped with the hardware stop pushbutton. It cuts off the transfer of control signals to the crane. The pushbutton does not terminate the real-time process running in the background. Therefore to stop the task you have to use *Simulation/Stop* from the pull-down menus in the model window.

4. Main Control Window

The user has a rapid access to all basic functions of the 3DCrane control system from the *Main Control Window*. It includes tests, drivers, models and application examples.

The *Main Control Window* presented in Fig. 3.1 contains four groups of the menu items:

- Tools
- Drivers
- Demo Controllers
- Experiments

4.1. Tools

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

Basic Tests - checks the measurements and control.

Go Home – moves the crane to the zero position, resets the encoders and sets control signals to zero. This button is frequently used before starting an experiment. When the *Go Home* procedure is finished we can be sure that the values of all measured signals have been set to zero.

Reset Angles – resets the angle measuring encoders in a fixed position. If you stop the payload manually, perform the *Reset Angles* operation to be sure that the payload angles measured by the encoders show zeros.

Go to Center - moves the crane to the center of the crane workspace and switches off the control. Remember that the zero position of the crane is in the corner of the XY plane. Most experiments cannot be started from the zero point. *Go to Center* allows the crane quickly move to the center.

Set Parameters - enables the user to change the default values of *Rail limits*, *Base Address* and *Z displacement*. The default value of the Base Address may cause a conflict with other devices installed in the computer. One has to be ensured that his computer configuration is free from address conflicts.

The user can also need to adjust the crane workspace dimensions to his requirements. Fig. 4.1 presents the window where such changes can be done. The user has to type numerical values into these editable text boxes.

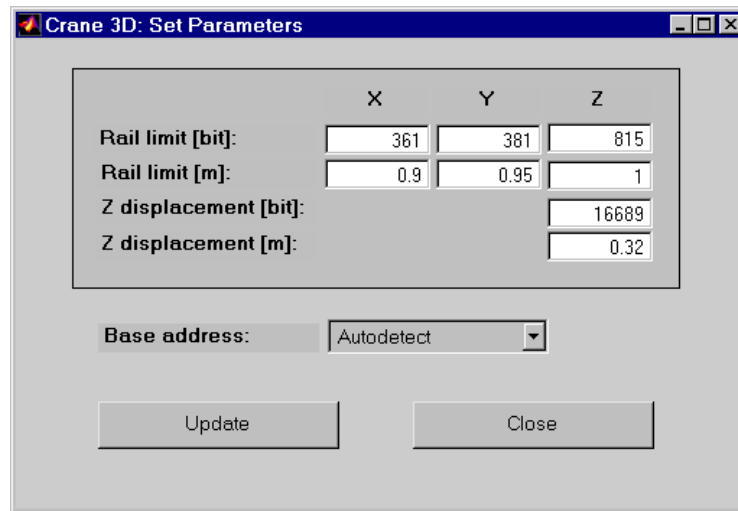


Fig. 4.1 *Set Parameters* window



All introduced modifications are written to the configuration file. Please be careful with introducing them. Writing the rail limit values exceeding the real rail limits may result in damage of the system elements.

Manual Setup – The **3DCrane Manual Setup** program gives access to the basic parameters of the laboratory 3-dimensional crane setup. The most important data transferred from the RT-DAC4/PCI board and the measurements of the crane as well as status signals and flags may be shown. Moreover the control signals of three DC drives may be set.

Double click the *Manual Setup* button and the screen presented in Fig. 4.2 opens.

The application contains five frames:

- The **RT-DAC4/PCI board** frame presents the main parameters of the PCI board.
- The **Control** frame allows to change the control signals.
- The X, Y and Z positions are given in the **X, Y and Z positions** frame.
- The **X and Y angles** frame contains the angle measurements.
- The **Status and flags** frame displays state of the status signals and flag values.

All the data presented by the **3D Crane Manual Setup** program are updated 20 times per second.

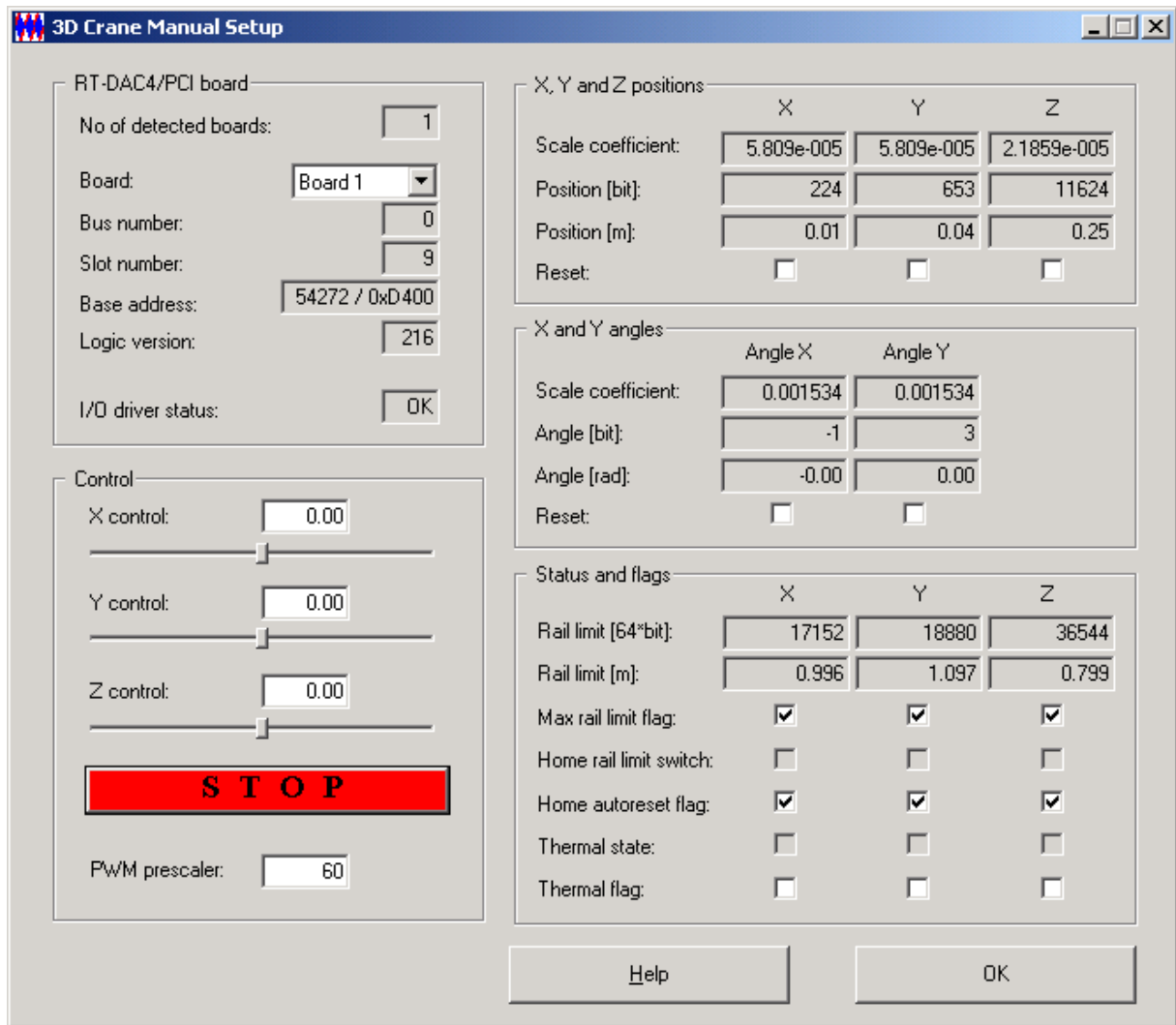


Fig. 4.2 The *Manual Setup* window

RT-DAC4/PCI board

The frame contains the parameters of the RT-DAC4/PCI boards detected by the computer.

No of detected boards

The number of detected RT-DAC4/PCI boards. If the number is equal to zero it means that the software has not detect any RT-DAC4/PCI board. When more then one board is detected the *Board* list must be used to select the board that communicates with the program.

Board

The list applied to select the board currently used by the program. The list contains a single entry for each RT-DAC4/PCI board installed in the computer. A new selection executed at the list automatically changes values of the remaining parameters within the frame. If more then one RT-DAC4/PCI board is detected the selection at the list must point to the board applied to control the 3D crane system. Otherwise the program is not able to operate in a proper way.

Bus number

The number of the PCI bus where the current RT-DAC4/PCI board is plugged-in. The parameter may be useful to distinguish boards, when more than one board is used and the computer system contains more than a single PCI bus.

Slot number

The number of the PCI slot where the current RT-DAC4/PCI board is plugged-in. The parameter may be useful to distinguish boards, when more than one board is used.

Base address

The base address of the current RT-DAC4/PCI board. The RT-DAC4/PCI board occupies 256 bytes of the I/O address space of the microprocessor. The base address is equal to the beginning of the occupied I/O range. The I/O space is assigned to the board by the computer system and may differ from computer to computer.

The base address is given in the decimal and hexadecimal forms.

Logic version

The number of the configuration logic of the on-board FPGA chip. A logic version corresponds to the configuration of the RT-DAC4/PCI board defined by this logic and depends on the version of the 3D crane model.

I/O driver status

The status of the driver that allows the access to the I/O address space of the microprocessor. The status has to be *OK* string. In other case the 3D crane software HAS TO BE REINSTALLED.

Control

The frame allows to set the control signals of three DC drives.

X control, Y control, Z control

The control signals of the X, Y and Z DC drives may be set by entering a new value into the corresponding edit field or by dragging the corresponding slider. The control values may vary from -1.0 to +1.0. The value of -1, 0.0 and +1 mean respectively: the maximum control in a given direction, zero control and the maximum control in the opposite direction to that defined by -1. If a new control value is entered in an edit field the corresponding slider changes respectively its position. If a slider is moved the value in the corresponding edit field changes as well.

S T O P

The pushbutton is applied to switch off all the control signals. When pressed all the control values are set to zero.

PWM prescaler

The control signals are generated as PWM waves. The PWM prescaler sets the divider of the PWM reference signal. The frequency of the PWM controls is equal to:

$$F_{PWM} = 20000/1023/(1+PWMPrescalerr) [kHz]$$

This parameter sets the frequency of all PWM control signals. The parameter may vary from 0 to 63. It causes the changes of the frequency of the PWM control signals from 19.55kHz to 305Hz.

X, Y and Z positions

The frame presents data related to the X, Y and Z axis positions. The X position is the rail position. The Y position is related to the position of the cart. The position of the load is denoted as Z. All position measurements are performed by the incremental encoders. There are the following four fields associated with each axis.

Scale coefficient

The value applied to calculate the position expressed in meters. The value read from the encoder counter is multiplied by the corresponding scale coefficient to obtain the position in meters.

Position [bit]

The value read from the corresponding encoder counter.

Position [m]

The position expressed in meters. The value read from the encoder counter is multiplied by the corresponding scale coefficient to obtain the position in meters.

Reset

The checkbox applied to reset the corresponding encoder counter. If the box is checked the related position is set to zero. The box has to be unchecked to allow position measurements.

As the incremental encoders are not able to detect an origin position the origin of the system has to be set by limit switches (see the description of the *Home autoreset flag*) or has to be set in a programming manner by a user. The **Reset** checkboxes are applied to set the origin position (zero position) in a programming way.

X and Y angles

The frame presents data related to the X and Y axis angles of the load. The angle measurements are performed by the incremental encoders. There are the following four fields associated with each angle.



Angle X and angle Y denote the angle deviations in X and Y direction, respectively. They are denoted by α and β in the mathematical model (see Fig. 8.1).

Scale coefficient

The value applied to calculate the angle expressed in radians. The value read from the encoder counter is multiplied by the corresponding scale coefficient to obtain the angle in radians.

Angle [bit]

The value read from the corresponding encoder counter.

Angle [rad]

The angle expressed in radians. The value read from the encoder counter is multiplied by the corresponding scale coefficient to obtain the position in radians.

Reset

The checkbox applied to reset the corresponding encoder counter. If the box is checked the related angle is set to zero. The box has to be unchecked to allow angle measurements.

As the incremental encoders are not able to detect an origin position the origin of the system has to be set in programming manner by a user. The angle reset checkboxes should be checked when the load remains motionless in the downright position.

Status and flags

The frame presents status data and flags related to the X, Y and Z axis. There are seven fields associated with each axis.

*Rail limit [64*bit]*

The RT-DAC4/PCI board is able to automatically turn off the control signal if the 3D crane system is going outside the operating range. The field defines the maximum value of the corresponding encoder determining the maximum accessible position. If the encoder reaches this position the RT-DAC4/PCI board is able to stop the generation of the DC control signals moving the system outside the operating range.

Rail limit [m]

The field defines the maximum accessible position expressed in meters. If the system reaches this position the RT-DAC4/PCI board is able to stop the generation of the DC control signals moving the system outside the operating range. The maximum position expressed in meters is obtained as the result of the multiplication of 64, maximum position expressed in bits and the corresponding scale coefficient.

Max rail limit flag

If the checkbox is selected the RT-DAC4/PCI board turns off the control when the system is going to move outside the operating range. It is recommended to keep this checkboxes selected all the time.

Home rail limit switch

The boxes that present the state of the home limit switches. If a limit switch is pressed the corresponding box is selected.

Home autoreset flag

The flag that causes automatic reset of the encoder counter when the corresponding home limit switch is pressed. If the checkbox is unchecked (the flag is inactive) the state of the limit switch does not influence the state of the encoder counter.

Therm state

The signal that presents the state of the thermal flag of the power interface. The system contains three power amplifiers for the DC drives. If the power amplifier is overheated the corresponding box is checked.

Therm flag

The flag that causes to turn off the control if the power amplifier is overheated. If the flag is unchecked, the power amplifier is overheated and the temperature increases the amplifier itself turns off the control signal. It is recommended to keep this checkboxes selected all the time.

Click the *Go to Center* button and you can see the changes in the *Manual Setup* window (see Fig. 4.3). Note that the X position, Y position and Z position have changed their values and become the center positions in the crane workspace.

The screenshot shows the '3D Crane Manual Setup' window with the following sections and values:

- RT-DAC4/PCI board**
 - No of detected boards: 1
 - Board: Board 1 (dropdown)
 - Bus number: 0
 - Slot number: 9
 - Base address: 54272 / 0xD400
 - Logic version: 216
 - I/O driver status: OK
- Control**
 - X control: 0.00 (slider)
 - Y control: 0.00 (slider)
 - Z control: 0.00 (slider)
 - STOP** (red button)
 - PWM prescaler: 60
- X, Y and Z positions**

	X	Y	Z
Scale coefficient:	5.809e-005	5.809e-005	2.1859e-005
Position [bit]:	4552	9600	18618
Position [m]:	0.26	0.56	0.41
Reset:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- X and Y angles**

	Angle X	Angle Y
Scale coefficient:	0.001534	0.001534
Angle [bit]:	0	0
Angle [rad]:	0.00	0.00
Reset:	<input type="checkbox"/>	<input type="checkbox"/>
- Status and flags**

	X	Y	Z
Rail limit [64*bit]:	8576	18880	36544
Rail limit [m]:	0.498	1.097	0.799
Max rail limit flag:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Home rail limit switch:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Home autoreset flag:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Thermal state:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Thermal flag:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- Buttons:** Help, OK

Fig. 4.3 *Manual Setup* window after the *Go to Center* action

4.2. Drivers

The main driver is located in the *RTWT Device Driver* column. The driver is a software go-between for the real crane MATLAB environment and the RT-DAC/PCI acquisition board. This driver serves to the control and measurement signals. Click the *3Dcrane Device Drivers* button and the driver window opens (Fig. 4.4)

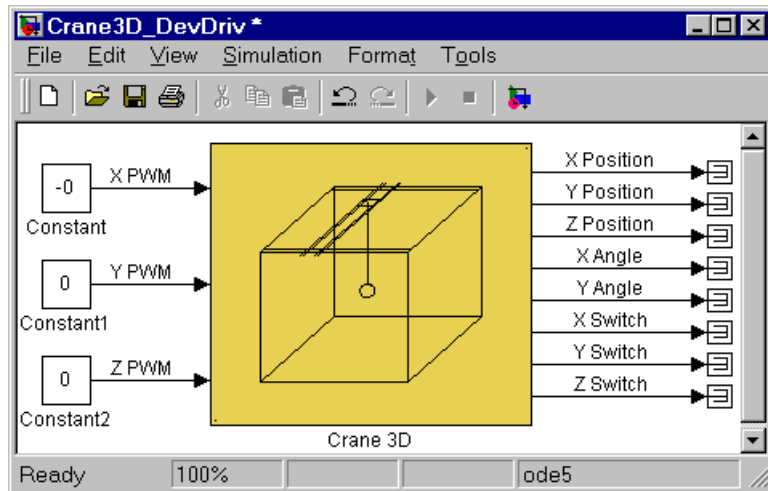


Fig. 4.4 3DCrane Device Drivers

The driver has three PWM inputs (DC motor controls) for the X,Y and Z-axes. There are 10 outputs of the driver: X position, Y position, Z position, two angles (see section 8) and additionally three safety switches. According to a pre-programmed logic the internal XILINX program of the RT-DAC/PCI board can use the switches to stop the DC motors.

When one wants to build his own application one can copy this driver to a new model.



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

Simulation Model – the simulation model of the crane is located under this button. Its external is identical as the model given in the *3Dcrane Device Drivers* except the lack of the safety switches (see Fig. 4.5). These switches are not used in the simulation mode. The simulation model is used for many purposes: identification, controllers design, etc.

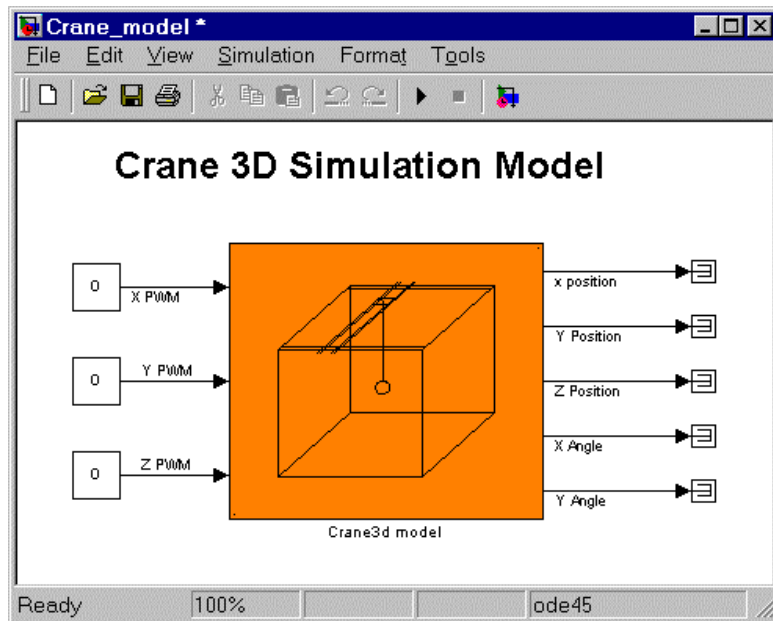


Fig. 4.5 3DCrane Simulation Model

In the mask presented in Fig. 4.6 you can introduce initial conditions for the model state variables. Additionally, by marking the checkbox you can use the model with constant length (see section 8 for details).

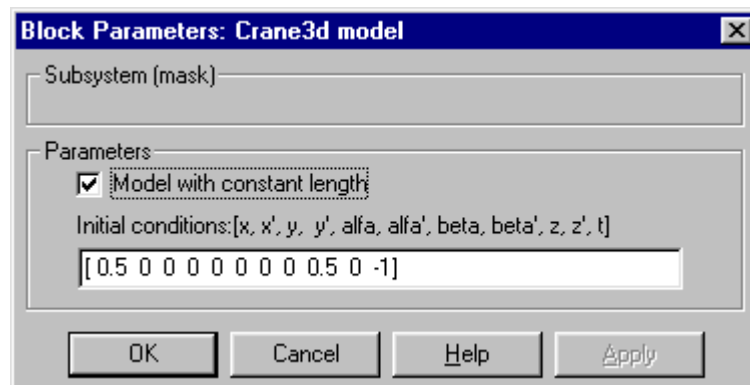


Fig. 4.6 Mask of the simulation model

The simulation model is running in the normal simulation mode.



Set solver options to: *Fixed step* and *Fixed-step size* equal to 0.001 at least. A greater value could result in the errors of solution.

The C source code of the *model3ddm.c* file is attached to the *DevDriv* directory.

4.3. Demo Controllers

In this column some examples of control systems are given. These demos can be used to familiarise the user with the crane system operation and allow creating the user-defined control systems. The examples must be rebuilt before using.

Due to similarity of the examples we focus our attention on one of them.

After clicking on the *Relay* button the model appears (Fig. 4.7):

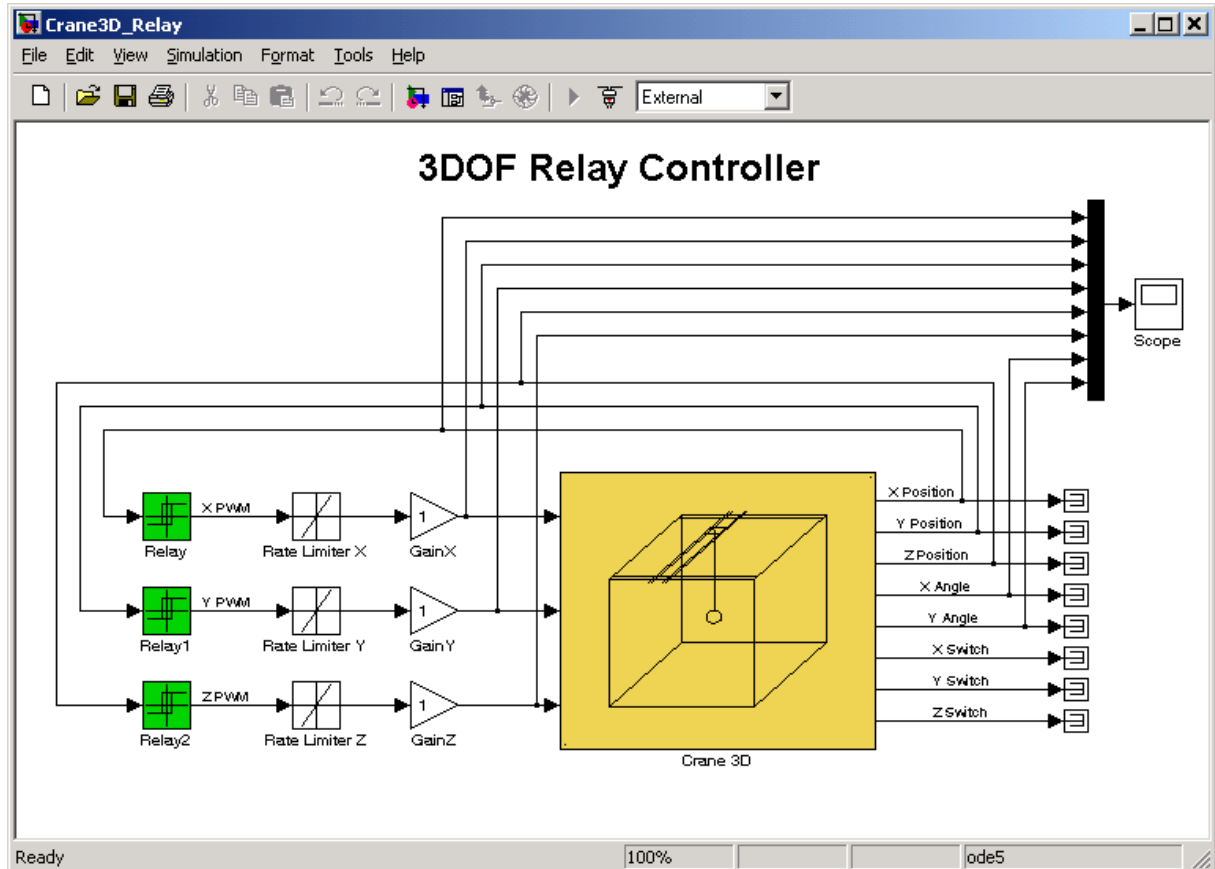


Fig. 4.7 Control system with the relay controllers

Notice that this model looks like a typical Simulink model. The device driver is applied in the same way as other blocks from the Simulink library. The only difference is that the model is used by Real Time Windows Target (RTWT) to create the executable library, which runs in the real-time mode.

The goal of the model is the relay control in the X-axis only. Therefore *GainY* and *GainZ* are set to zero.

- Look at the mask of the *Relay* block connected to the X PWM input (Fig. 4.8).

Note that the control generated by the controller has two values: $+0.5$ and -0.5 . The On/Off limits are 0.2 and 0.6. This means that the crane will move between these limits with the speed corresponding to the control value equal to 0.5.

- To choose the starting point inside the $[-0.5, 0.5]$ range go to *Main Control Window* and click the *Go to Center* button.

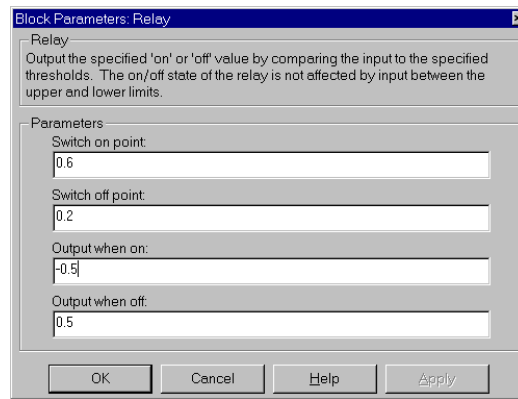


Fig. 4.8 X-PWM controller parameters

- Then, choose the *Tools* pull-down menus in the Simulink model window. The pop-up menus provide a choice between predefined items. Choose the *RTW Build* item. A successful compilation and linking process is finished with the following message:

*Model Crane3D_Relay.rtd successfully created
 ### Successful completion of Real-Time Workshop build procedure for model:
 Crane3D_Relay*

If any error occurs then the message corresponding to the error is displayed in the MATLAB command window.

- Next, click the *Tools/External Mode Control Panel* item and next click the *Signal Triggering* button. The window presented in Fig. 4.9 opens.
- Select *XT Scope* , set *Source* as the manual option, mark *Arm when connect to Target* option and close the window.
- Return to the model window and click the *Simulation/Connect to Target* option. Next, click the *Simulation/Start real-time code* item.

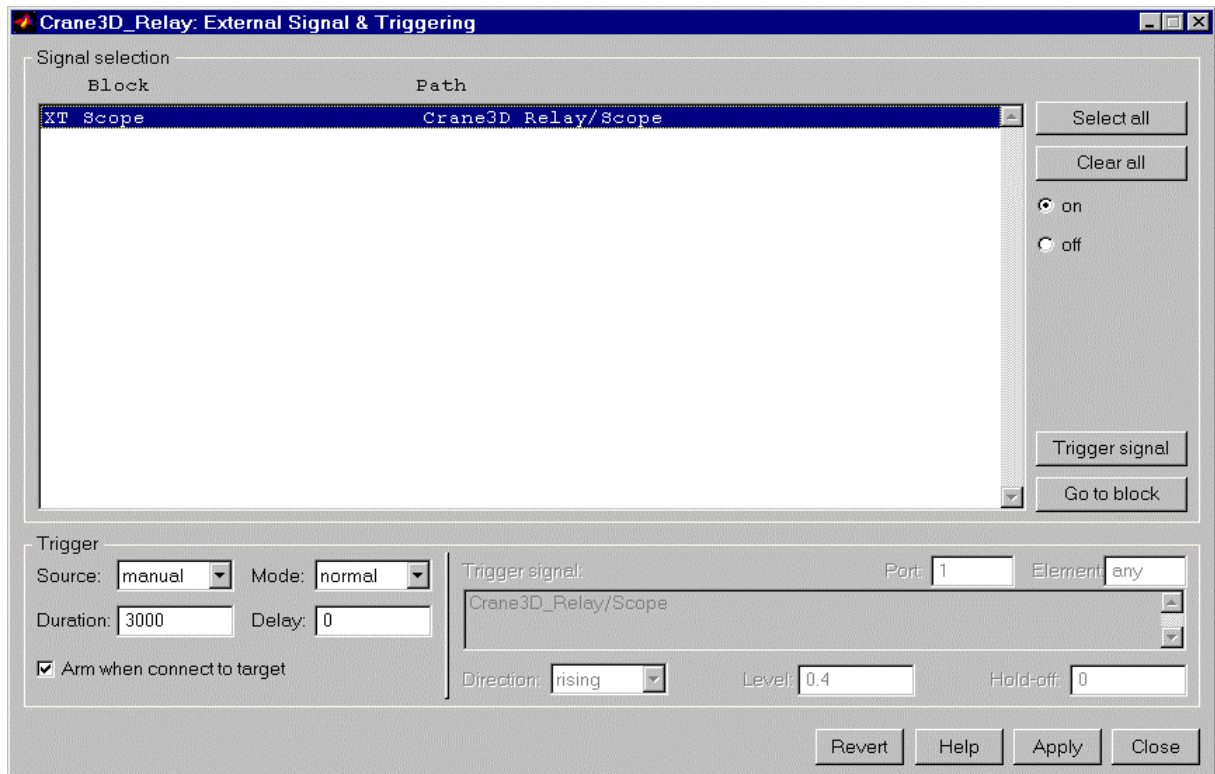


Fig. 4.9 External Signal & Triggering window

- Observe the plots in the scope and click *Stop Simulation* after some time.

Results of the example are presented in Fig. 4.10.

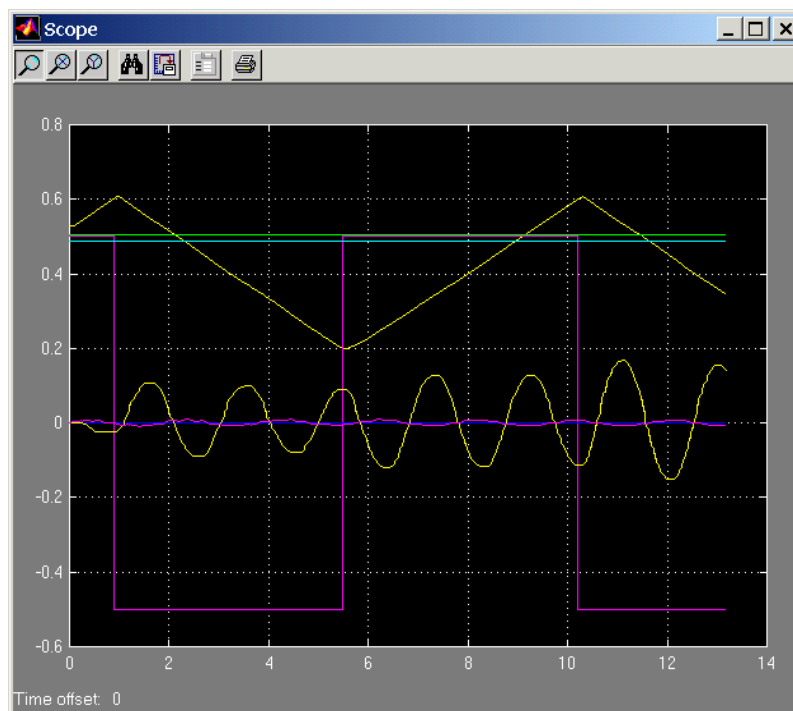


Fig. 4.10 Results of the relay controller demo experiment

The X position starts from 0.54 and changes between 0.2 and 0.6. The control (red line) is a square wave in the range $[-0.5, 0.5]$. The control switches when the X position reaches one of the limit values. Note that the X angle in the form of a sinusoidal curve is modulated by the control interacting with friction.

5. Your first real-time control experiment

We propose two experiments. In the first experiment only one control loop in the x direction is defined. In this case stabilisation of the angle of the payload is neglected. In the second experiment the stabilisation of the payload angle is added.

We begin from a simple real-time control experiment. A PID controller for the x position of the cart is built. The *Crane3D_first* model is illustrated in Fig. 5.1. To invoke it, click *Model for control experiment* button in the *Main Control Window*. In this case, the active control corresponding to the x cart position is all you need. The Simulink blocks included in the control are drawn with dropped shadows to distinguish active control loops from disabled loops. In our first experiment we use the *X position of the cart* PID controller activating only its P control part.

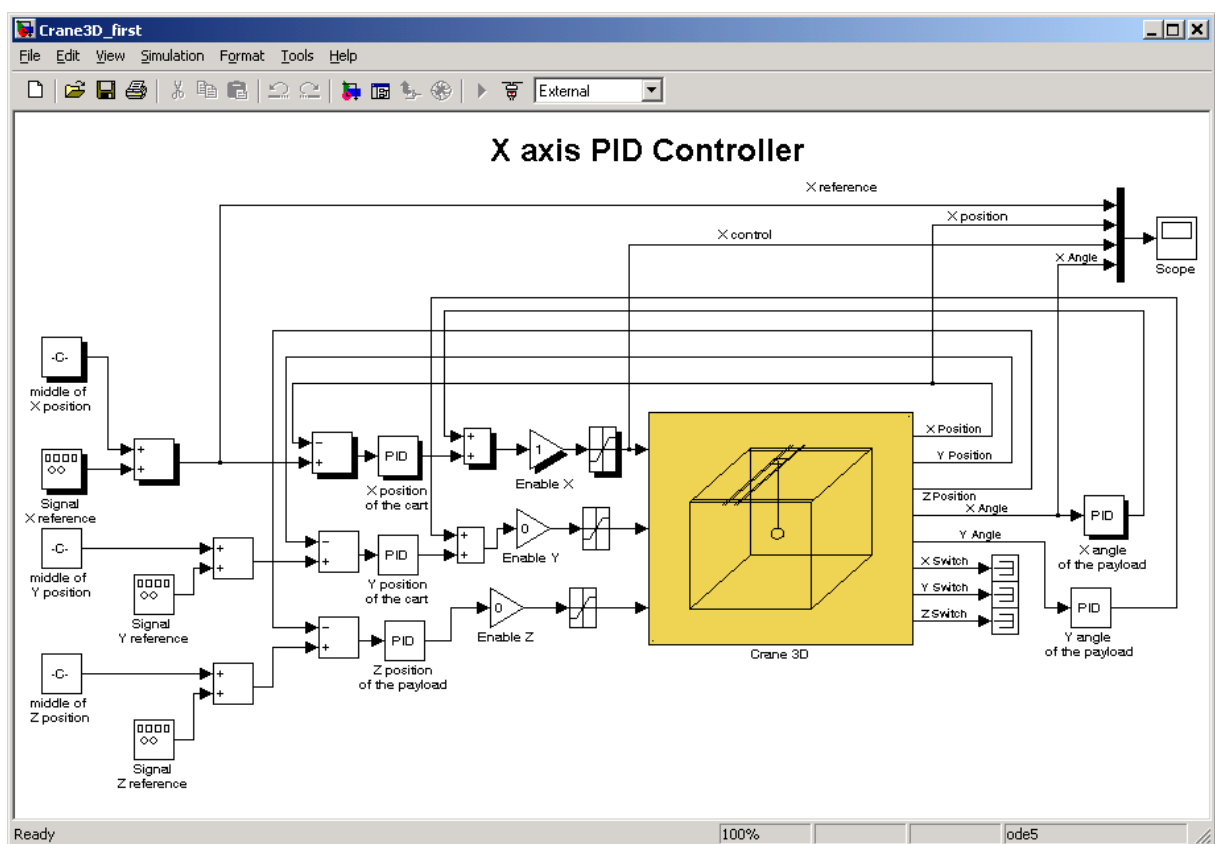


Fig. 5.1 Two PID controllers applied in a real time experiment. The first step – PID of *X position of the cart* is active. The second step – both controllers are active

We define a source of a desired x position signal as the *X reference/Generator* from the Simulink library (see Fig. 5.2)

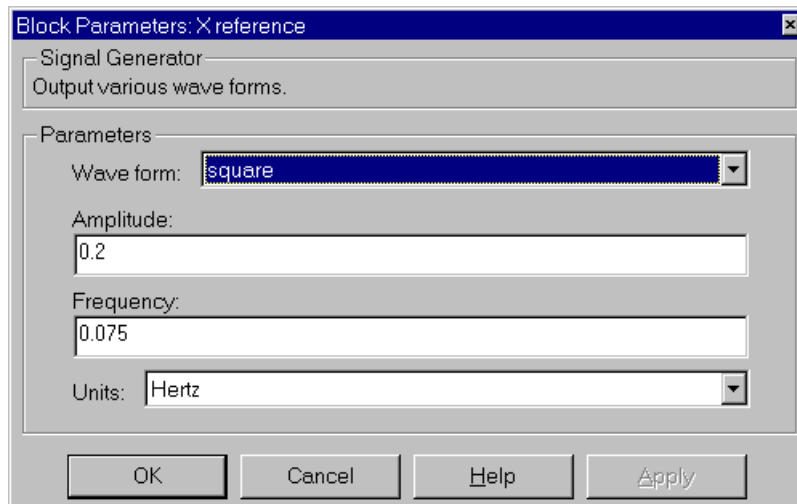


Fig. 5.2 Signal Generator set to become the square wave signal source

As usual, after performing the *GO HOME* and *GO TO CENTER* actions, we start an experiment from the middle of the *x*, *y* rails. Therefore the constant block *shift* is required.

Next, we define the signals to be used. These are *X-reference* – the desired *x* position value, *X-position*, *X-control* and *X-angle*. These signals, among others, are connected to the *Scope* block.

The properties of this block are defined below (see Fig. 5.3). This window opens after the selection of the *Scope/Properties* tab. Mark the *Save data to workspace* checkbox, define the *Variable name* as *EX1* and the *data format* as structure. This means the collected data within 30 seconds time range are saved to the workspace in the structure *EX1*. The sampling period set in the *Simulation Parameters* window (see Fig. 5.3) is equal to 0.01, *Sampling Decimation* is set to 10. Therefore, the size of *EX1* is equal to

$$30 \text{ s} / (0.01 \text{ s} \times 10) + 1 = 301$$

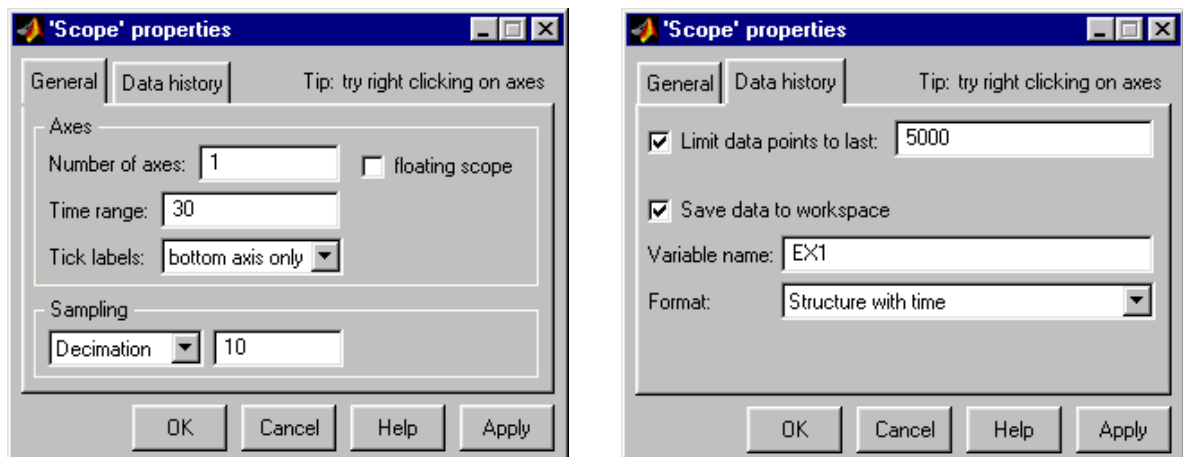


Fig. 5.3 Setting of the *Scope* block

Next, return to the *Main Control Window* and select the *Simulation/Parameters* item. In the *Solver* tab select *Fixed Step* and set *Stop time* equal to 30 (Fig. 5.4). The *Real-time Workshop* tab must be defined as in Fig. 5.5.

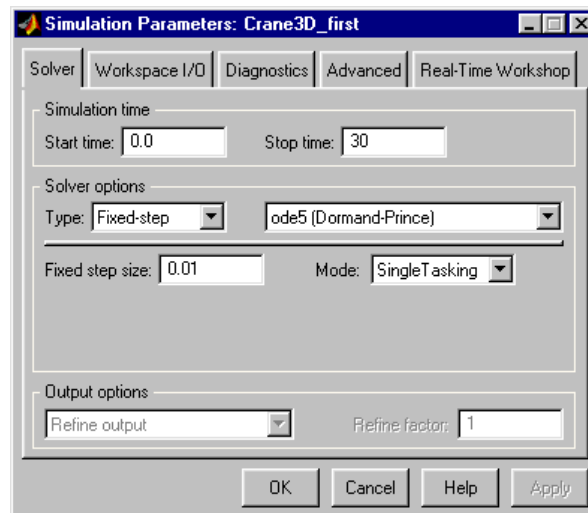


Fig. 5.4 The *Solver* tab

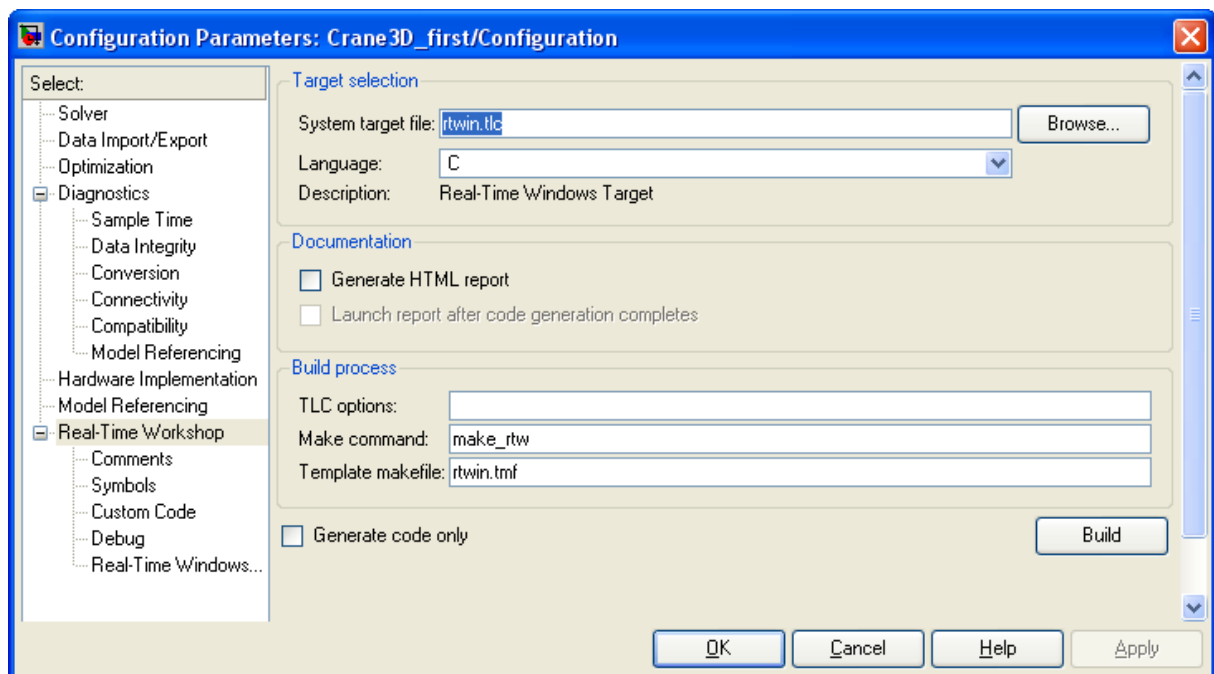


Fig. 5.5 The *Real-time Workshop* tab

The next step is to set the PID controllers. We set the *Proportional* part of the *X position of the cart* PID controller as indicated by the arrow in Fig. 5.6. The *X angle of the payload* PID controller remains inactive. The other controlling loops are disabled due to the *Gain* blocks set to zero (see Fig. 5.1).

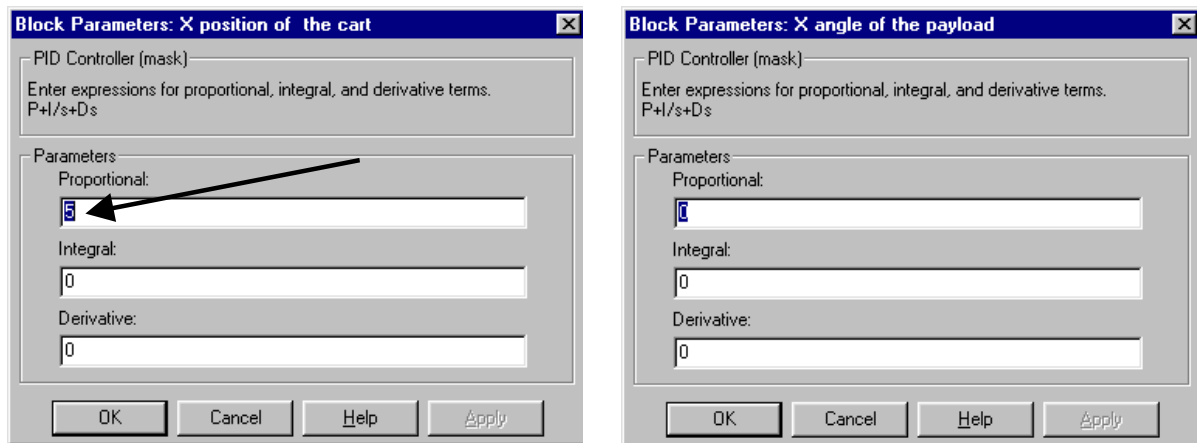


Fig. 5.6 Setting of the PID controllers

Mark *Simulation/External* item in the *Crane3D_first* model window (see Fig. 5.7).

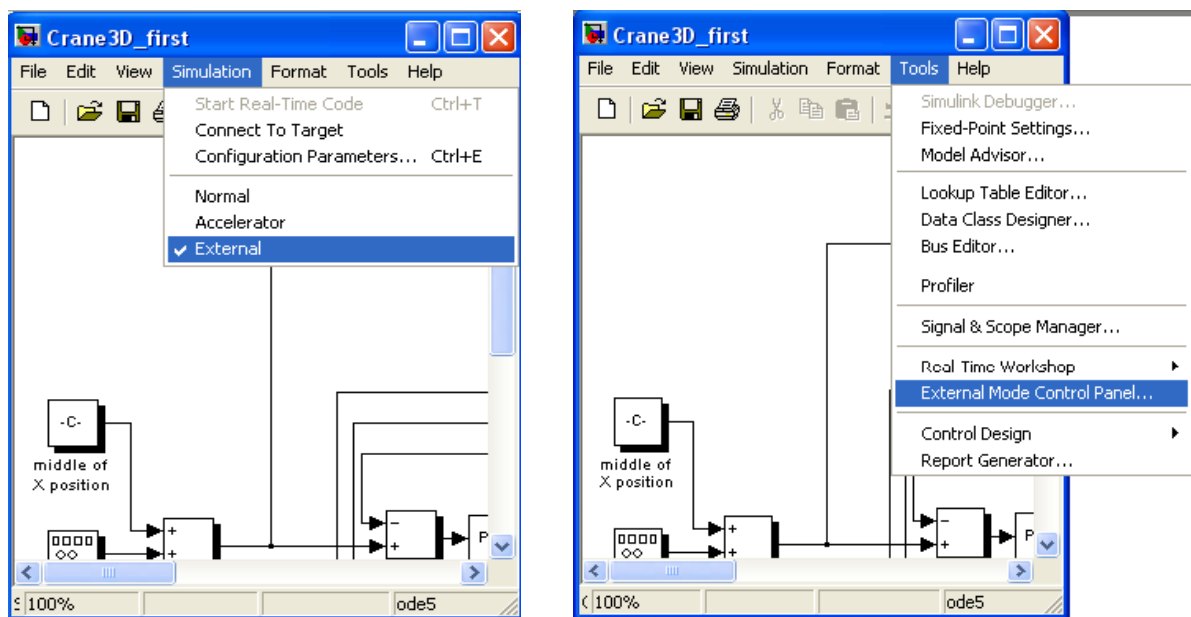


Fig. 5.7 External control mode

Next, invoke the *Tools/External mode control panel* item. The *External Mode Control Panel* window opens (see Fig. 5.8).

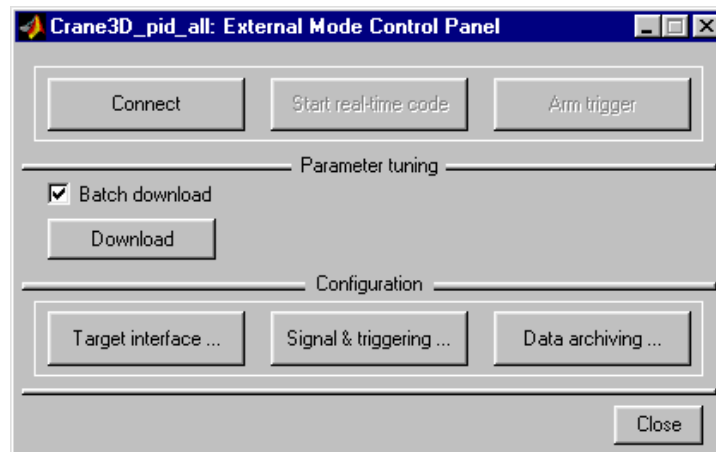


Fig. 5.8 Setting data acquisition in RTWT

By clicking on the *Signal & Triggering* button invoke the window shown in Fig. 5.9. In this window we define a triggering mode for marked blocks. In our case only one block exists – *XT Scope*. We mark *XT Scope*, set *Source* as the *manual* option, mark *Arm when connect to target* and close the window.

Now we can build the real-time model. To do it select the *Simulation /Parameters* item and then the *Real-Time Workshop* tab in the model window, and click the *Build* item. Successful compilation and linking process should be finished with the following message:

Model Crane3D_first.rtd successfully created

Successful completion of Real-Time Workshop build procedure for model: Crane3D_first

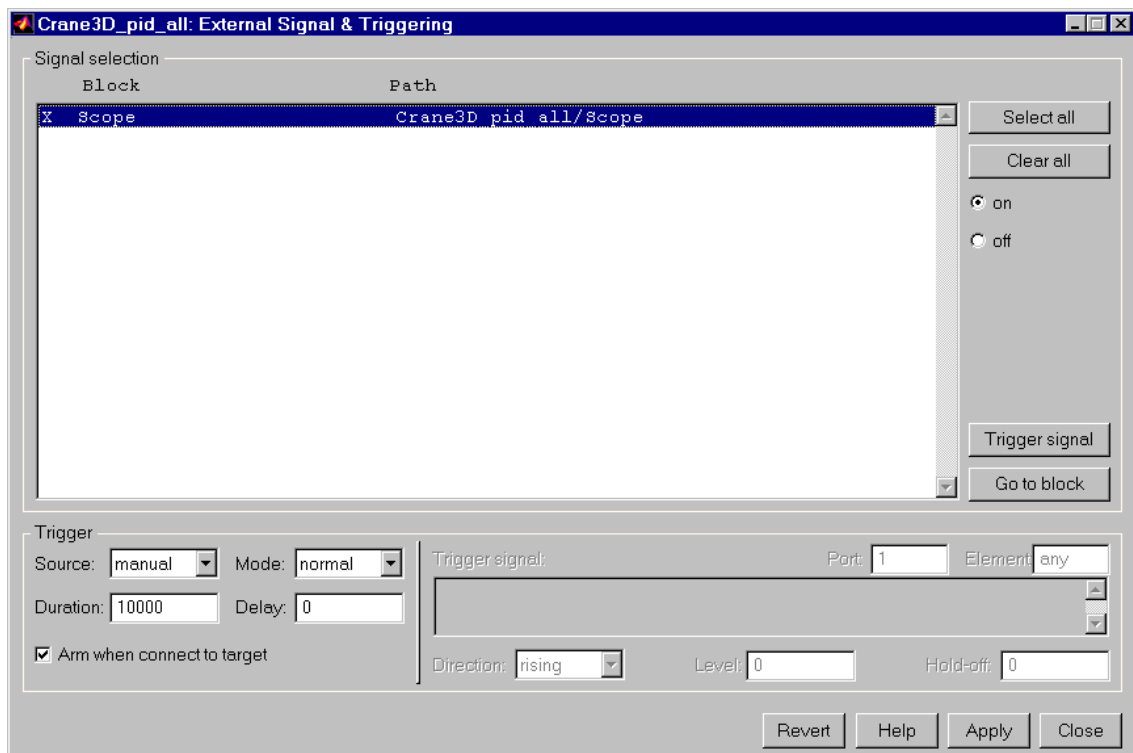


Fig. 5.9 Setting triggering of signals

5.1. Real-time experiment

Having prepared the controller model you can start the real-time experiment.

Two actions *GO HOME* and *GO TO CENTER* must be performed in the *Main Control Window* first. The crane is ready for the experiment. The cart is in the middle of the x, y plane, the payload is hanging down in its rest position. Open the *Scope* figure clicking on the *Scope* block.

Now return to the model window and click the *Simulation/Connect to Target* item. Next, click the *Simulation/Start real-time code* item. It activates the experiment lasting 30 seconds. Observe the cart motion in the x direction. The cart follows the desired square wave signal controlled by the P regulator. The payload oscillates freely, being uncontrolled. After 30 seconds the experiment stops. The history of the *EX1* variable is visible in the *Scope* (see Fig. 5.10). Notice the harmonic (uncontrolled) angle signal of the payload, the square wave generated by *Signal Generator* followed by the cart x position signal. The static error is due to the inadequate P control action. The control has the highest magnitude among other signals visible in the figure. When an abrupt change in the wave signal occurs then it results in the saturation of the control signal.



Fig. 5.10 Data visible in the scope during the experiment

5.1.1. Data processing

The results are saved to the workspace as a structure variable *EX1*. If you write the variable name in the MATLAB command window then you obtain the answer

```
EX1 =  
    time: [301x1 double]  
    signals: [1x1 struct]
```

blockName: 'Crane3D_pid_all/Scope'

This data can be plotted in many ways. For example use the following command

```
>> plot(EX1.time,EX1.signals.values(:,1:4))
```

You can repeat the experiment several times using different P parameter settings and including another P controller for the x angle. The parameters of the controllers for successive five experiments are given in Table 5-1.

Number of experiment	P of the cart position	P of the payload angle	Figure
1	2.5	–	Fig. 5.11
2	5	–	Fig. 5.11
3	2.5	1	Fig. 5.12
4	5	2	Fig. 5.12
5	5	4	Fig. 5.12

Table 5-1 Parameters for the experiments

The results of five experiments are presented in Fig. 5.11 and Fig. 5.12.

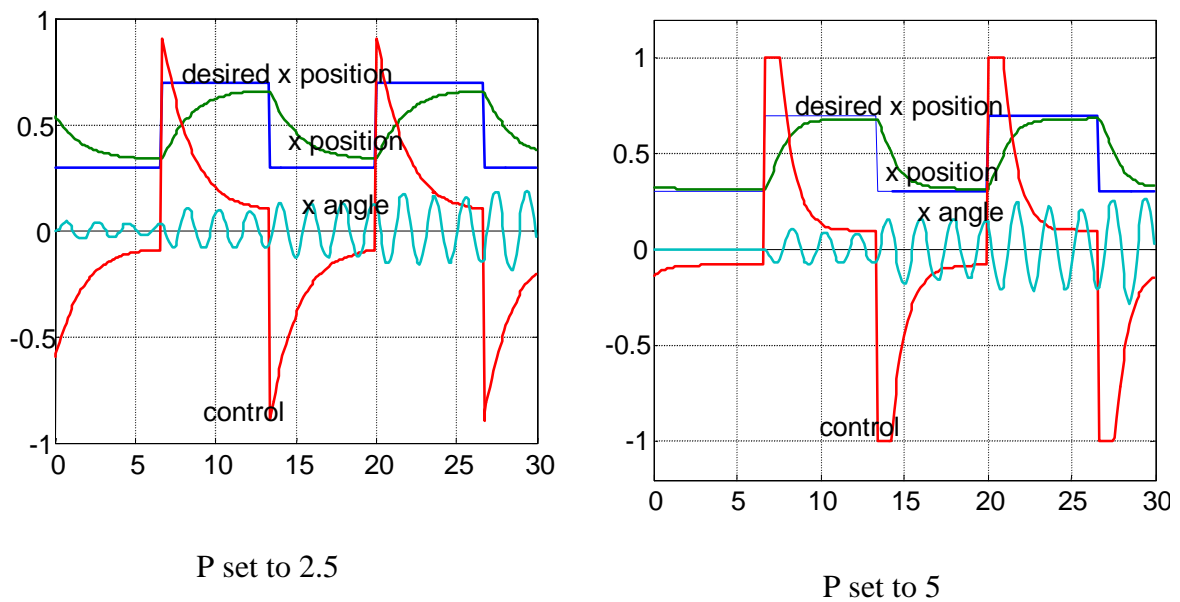
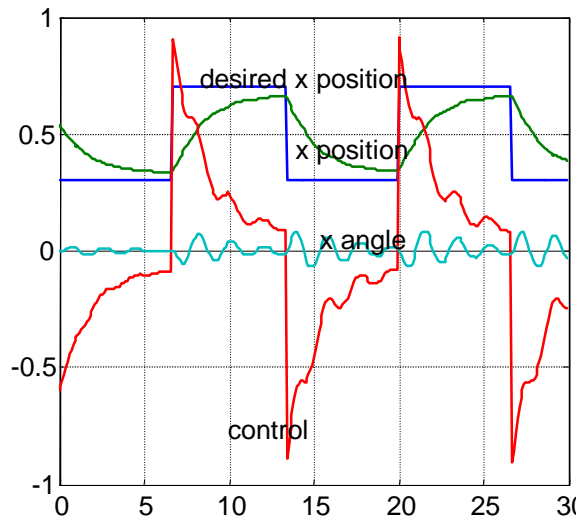
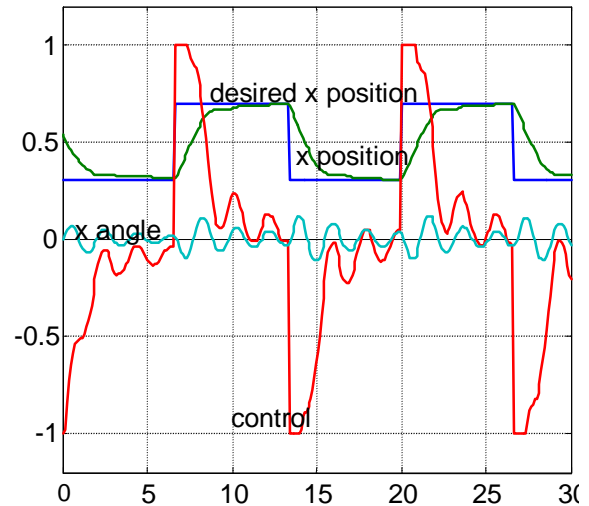


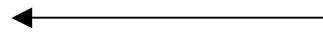
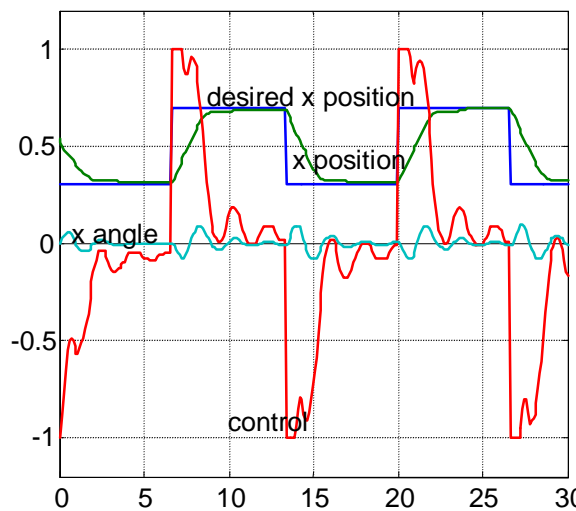
Fig. 5.11 Only one controller is active – desired x position of the cart is tracked



P of the cart position set to **2.5**
P of the payload angle set to **1**



P of the cart position set to **5**
P of the payload angle set to **2**



P of the cart position set to **5**
P of the payload angle set to **4**

Fig. 5.12 Two controllers – desired x position of the cart and x angle of the payload are tracked

The left-hand side of Fig. 5.11 shows that the P value set to 2.5 is too small for a proper x position tracking. The static error of x position is large. A higher gain P equal to 5 (the right-hand side of Fig. 5.11) reduces the static error but results in the saturation of control.

In Fig. 5.12 one can see similar results obtained for two active controllers. We focus our attention on the x angle control. The trade-off between two acting controllers is well visible in the upper left-hand picture of Fig. 5.12. One control signal serves for two control purposes: follow the desired value of the cart x position and simultaneously stabilises the payload in its hanging down position.

5.2. Simulation

We can repeat the real-time experiment from the previous section in a purely simulated form.

We invoke the *Crane3D_first_model* window. Notice two differences. The *Crane3D* real-time driver block has been replaced by the *Crane3D_model* simulation model block. The *External* mode of operation has been replaced by the *Normal* mode of operation (see Fig. 5.13). All other parts remain as in the real-time experiment.

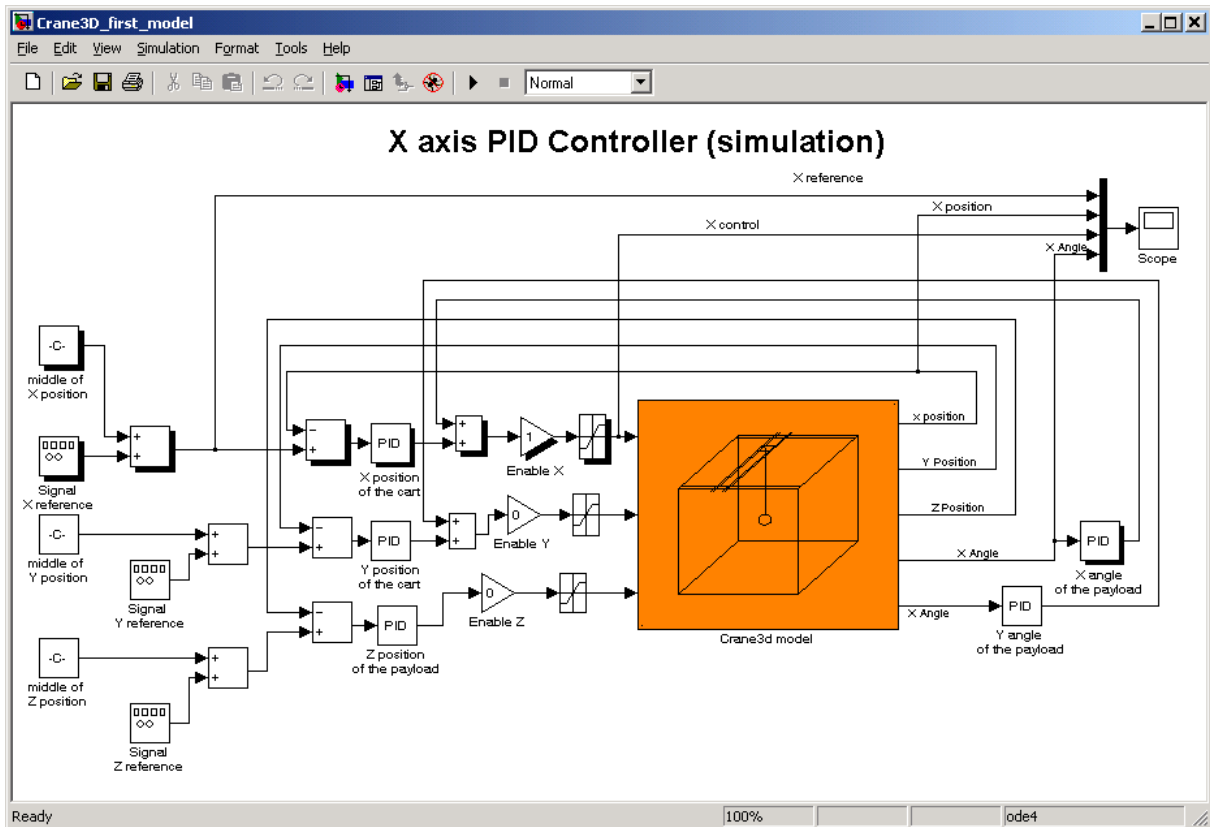


Fig. 5.13 Two PID controllers applied in a simulated experiment. The first step – PID of *X position of the cart* is active. The second step – both controllers are active

The interior of *Crane3D_model* includes the complete nonlinear model described in section 6.4. After clicking on *Crane3D_model* the mask given in Fig. 5.14 opens. You can set the initial values of 10 variables. You can mark the constant or varying pendulum length. In our example all initial conditions are set to zero except the *X position* set to 0.55 and *Z position* set to 0.5 and the last variable *t* set to -1. Setting *t* to -1 denotes that the source of time is the *RTWT* clock.

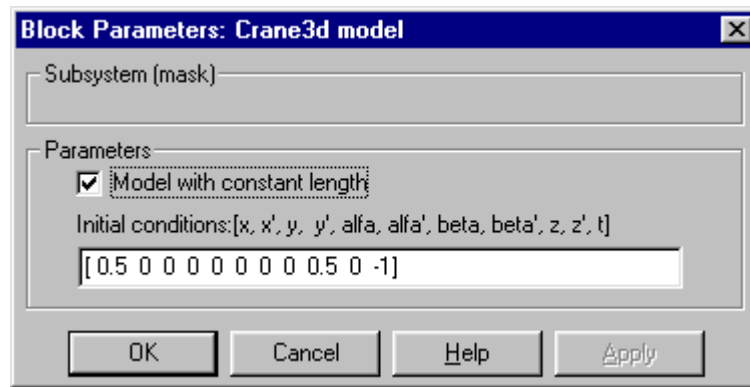


Fig. 5.14 Mask of *Crane3D_model*

If you look under the *Crane3D_model* mask you can see its interior (Fig. 5.15). *model3dddm* is the executable dll file. The scale factors are set to 34. These parameters relate to friction and tension of the belts.

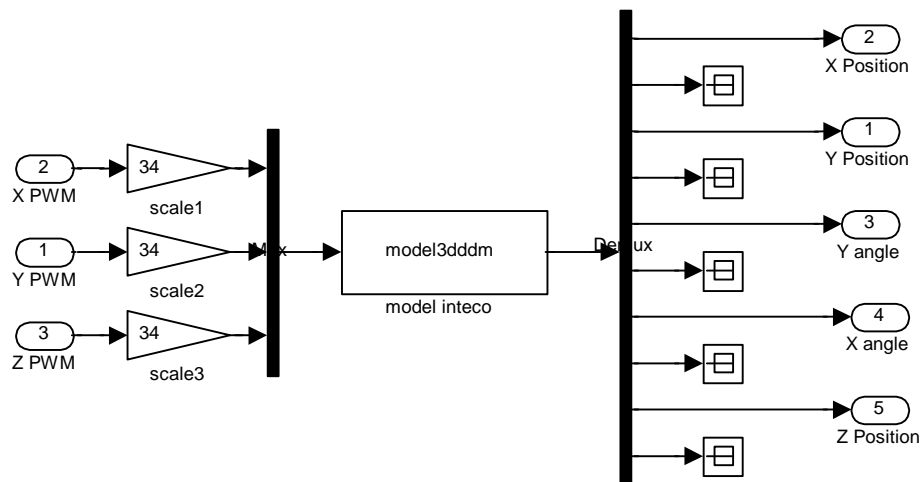


Fig. 5.15 The interior of *Crane3D_model*

In opposite to the real-time model it is not necessary to rebuild the model before running. Open *Simulation parameters* from the menu bar. Notice the solver options: *Fixed step* and *Fixed-step size* set to 0.001 in the editable text box. You can start a simulation from the menu bar. When the simulation is running you can watch the results in the scope (see Fig. 5.16). Similarly as in the real time experiment, the data are saved in the scope in the *EX1* structural variable.

Plotting four curves as functions of time (Fig. 5.16) is produced by:

```
>> plot(EX1.time, EX1.signals.values)
```

We perform simulations related to the experiments 2 and 5 from Table 5-1. The results are visible in Fig. 5.17. They are similar to the results of the real-time experiments. The model reflects characteristic features of the laboratory crane. The model compatibility to the real crane strongly depends on such parameters as the payload lift-line length, the static cart

friction, the belt tension, etc. These and other parameters are written into the C source code of the *model3ddm.c* file attached in the *DevDriv* directory. If you wish to modify this file please make a copy and introduce the new parameters in the copy. Remember to produce an executable .dll file afterwards.

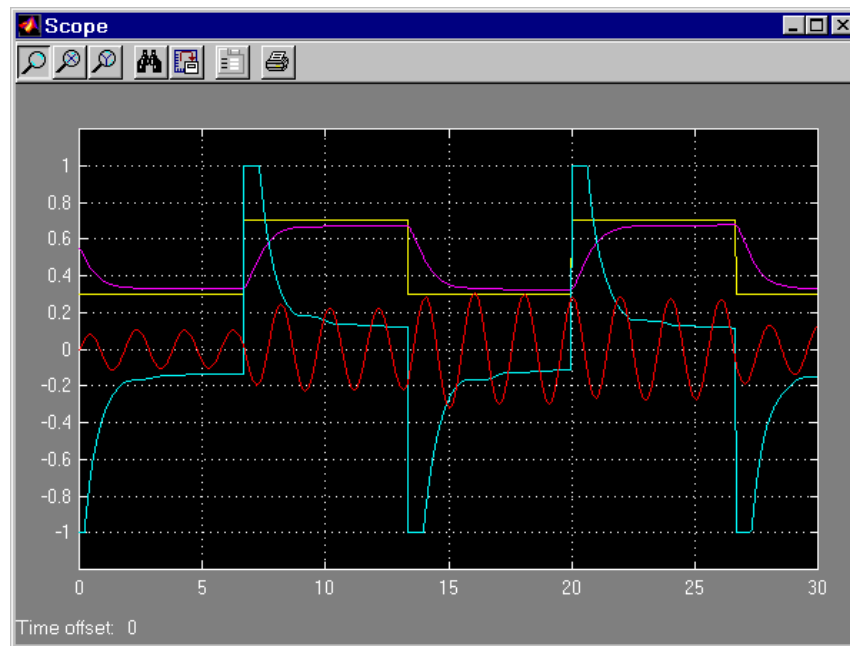
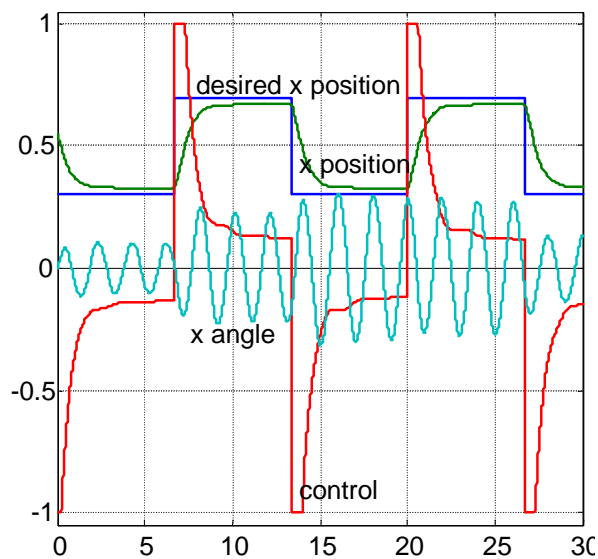
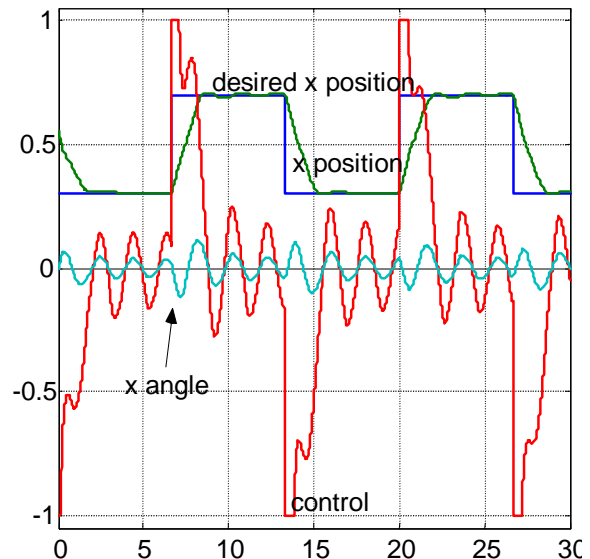


Fig. 5.16 Simulation data visible in the scope



P of the cart position set to **5**
P of the payload angle set to **0**

Only one controller is active – the desired x position of the cart is tracked



P of the cart position set to **5**
P of the payload angle set to **4**

Two controllers – the desired x position of the cart is tracked and the x angle of the payload is stabilised

Fig. 5.17 Simulation results

5.3. PID control of load position

In our experiment the cart is following a Lissajous curve. We invoke the *Crane3D_impres* model from the MATLAB command window (see Fig. 5.18). We put the cart into motion in two directions. The desired cart positions are generated as two sinusoidal signals. There are two generators identical in amplitudes equal to 0.2 m and different in frequencies. The *X* motion frequency is 0.4 rad/s (Fig. 5.19) and the *Y* motion frequency is 0.8 rad/s.

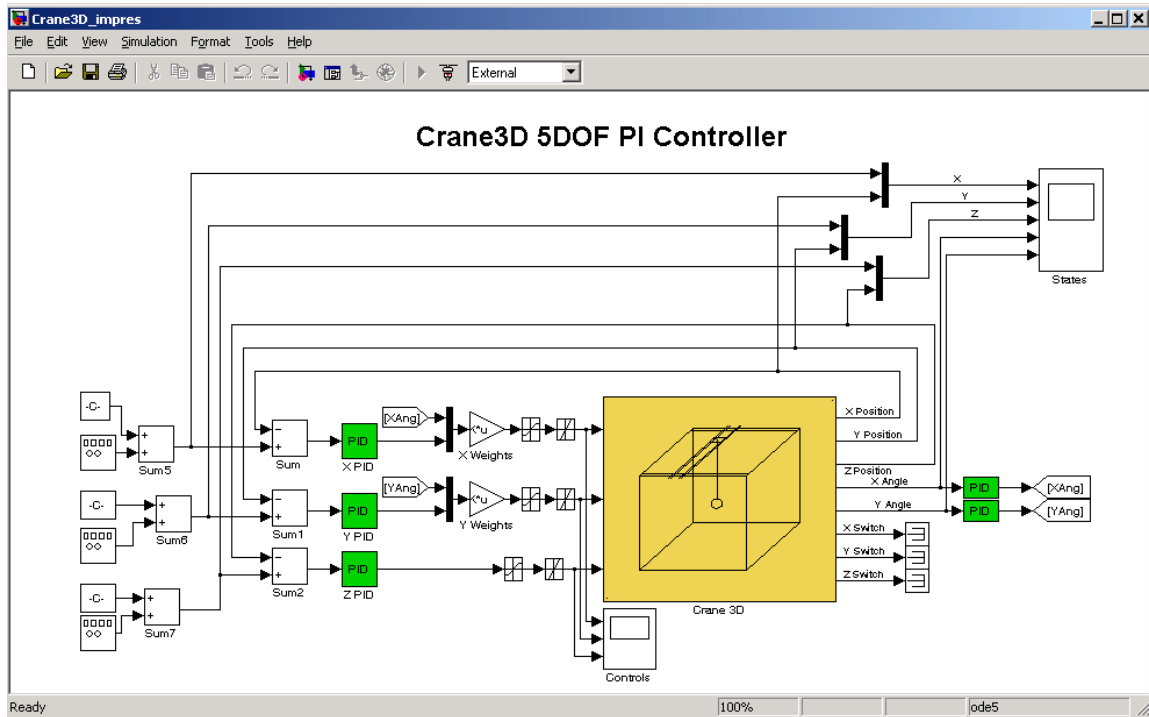


Fig. 5.18 The controller built for the cart to follow a Lissajous curve

We perform the experiment twice, first time without the *P* angle controllers and second time, with the *P* angle controllers set to 20 (see Fig. 5.20).

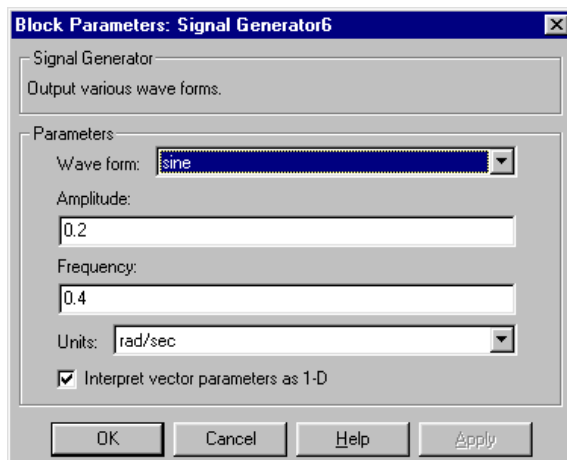


Fig. 5.19 Generator of the desired *X* position signal

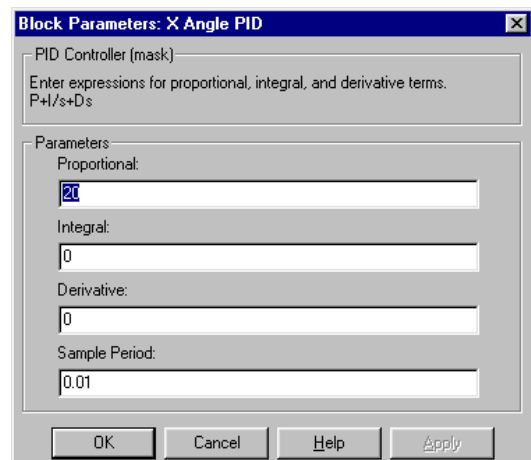


Fig. 5.20 *X* angle PID controller

The cart motion is shown in Fig. 5.21. The thick line represents the cart position in the $X Y$ plane (there is no movement in the Z direction). The respective controls and the payload angles are shown in Fig. 5.22 and Fig. 5.23.

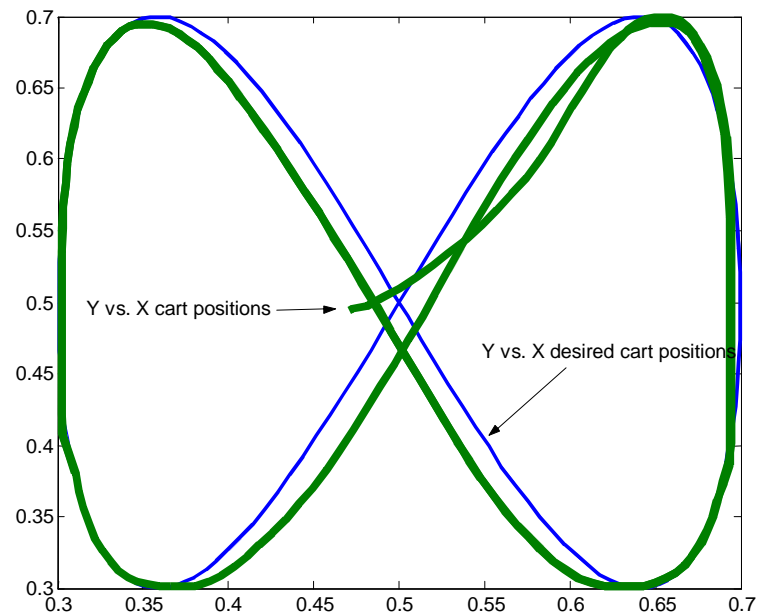


Fig. 5.21 The desired cart positions – thin line and the cart positions – thick line (in meters)

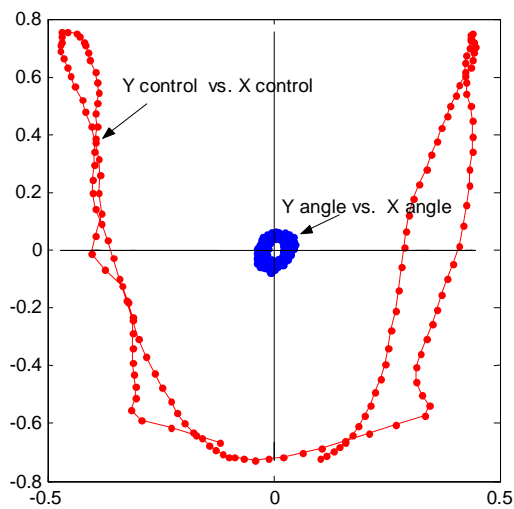


Fig. 5.22 The controls (normalised units) and the payload angles [rad] on the $X Y$ plane; the angles **without** control

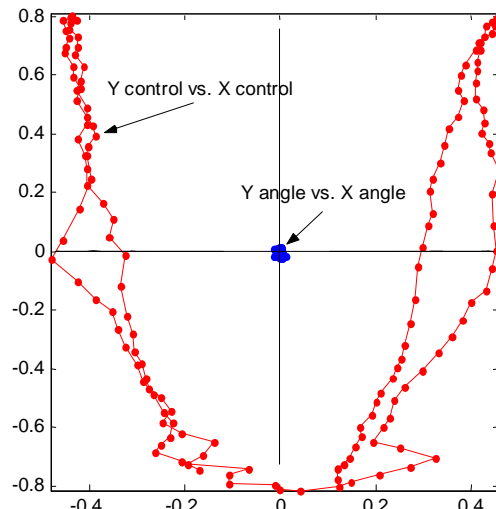


Fig. 5.23 The controls (normalised units) and the payload angles [rad] on the $X Y$ plane; the angles **with** control

Notice that the control curve in the $X Y$ plane for the case of uncontrolled deviations of the payload is smoother than that for the controlled deviations case. The payload deviations are zoomed in Fig. 5.24 and Fig. 5.25 (please notice the scales of deviations).

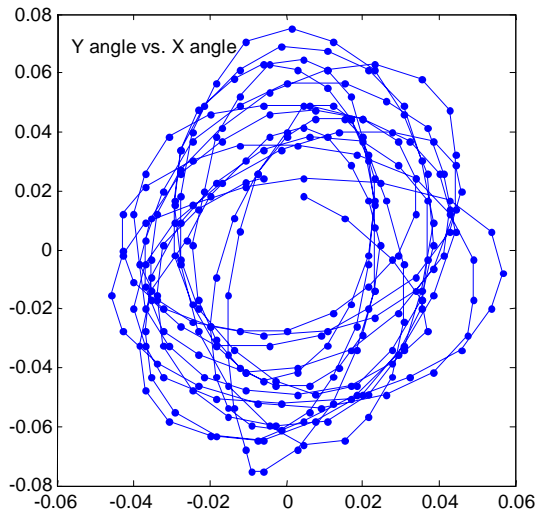


Fig. 5.24 Unstabilised deviations of the payload in the $X Y$ plane (in meters)

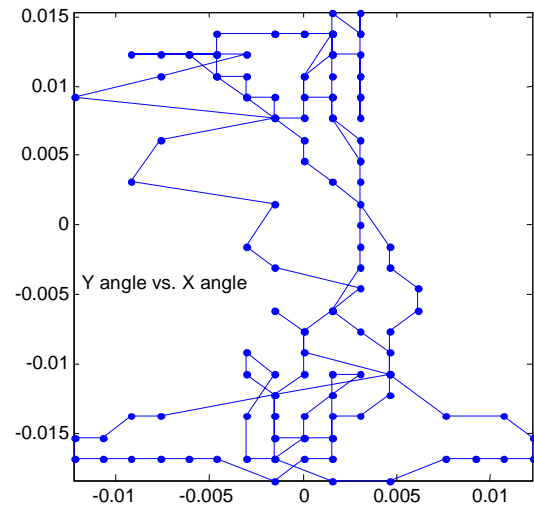


Fig. 5.25 Stabilised deviations of the payload in the $X Y$ plane (in meters)

The controls in the X and Y directions are shown in Fig. 5.26 and Fig. 5.27. Two cases of control: with and without stabilisation of angles are compared.

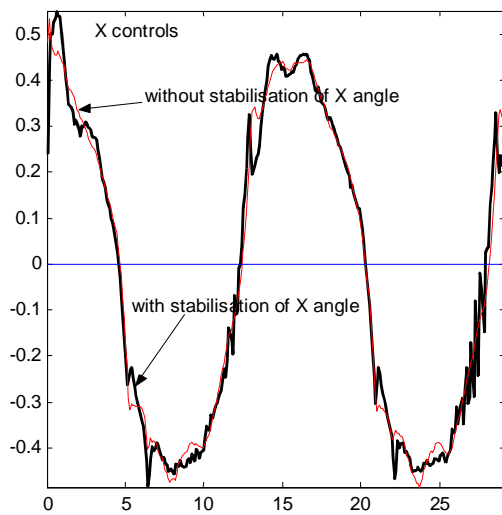


Fig. 5.26 The controls (normalised units) in the X direction vs. time (seconds)

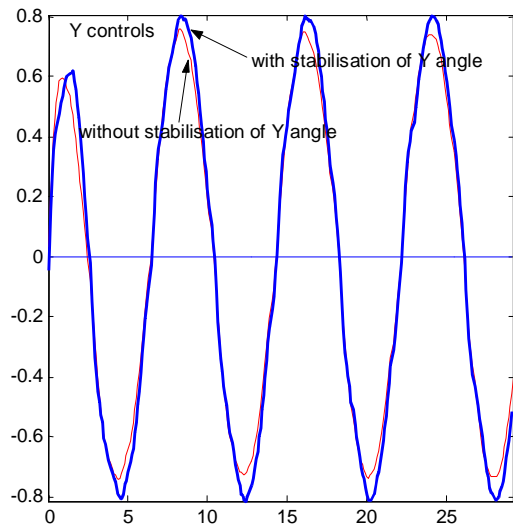


Fig. 5.27 The controls (normalised units) in the Y direction vs. time (seconds)

Notice that the controls related to the case of angles stabilisation (thick lines in the figures) share their actions between two tasks: tracking a desired position of the cart and stabilising the payload in its down position.

6. CASE STUDY

This section describes nine steps to be performed by the user for collecting data from the real 3DCrane system, identifying a simple crane model and identifying the z displacement. Finally, the optimisation of the 5 DOF PID controller based on the collected data yields an appropriate set of the PID controller's parameters.

All steps are described in detail below. After clicking on the *Simple Model* button the following window opens (see Fig. 6.1).

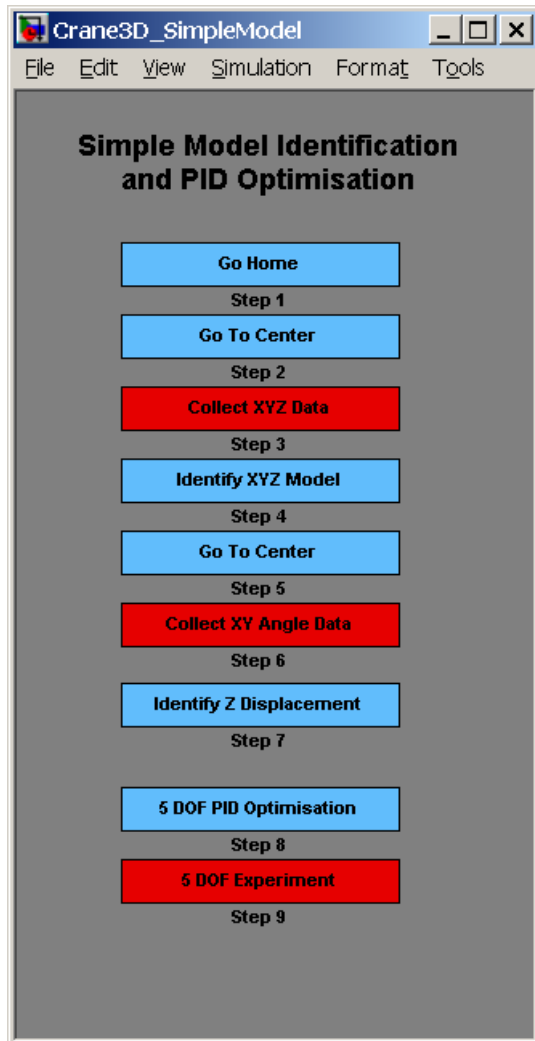


Fig. 6.1 Case study window

Step 1 and **Step 2** have been described in section 4.1. After performing these steps the crane is ready to start an experiment.

Step 3

This step collects the $X Y Z$ data. The cart is steered forth and back in the X and Y directions. The payload is lifted up and lowered. These actions run simultaneously due to operating controllers, as illustrated in Fig. 6.2. There are three identical *Relay* blocks used as controllers. You can see the control ranges by clicking on a block – see Fig. 6.3. The control values change from 0.5 to -0.5 (a half of the maximal excitations). The motion ranges are between 0.3 and 0.5 m.

Before starting the motion set the *Base Address* first. Click the *Set Base Address* button in the *3DCrane 3DOF Relay Controller* window – see Fig. 6.2. Next, choose *Tools* from the window menu bar. These pull-down menus execute callback routines when the user selects an individual menu item. Choose the *Real_Time Workshop* menu and the *Build Model* submenu. The model is rebuilt. Finally, choose *Simulation* from the window menu bar and click the *Connect to target* pull-down menu.

When the system is running observe the real motion of the crane and the plot in the scope – see Fig. 6.4.



The Matlab Optimisation Toolbox is required to perform step 4.

Step 4

Click the *Identify XYZ Model* button. The optimisation procedure starts. The plot illustrated in Fig. 6.5 appears.

The following simple linear model of the crane dynamics in the X -axis is assumed

$$G_x(s) = \frac{K_x}{s(T_x s + 1)}.$$

The *Identify XYZ Model* button calls the *CaseStudy_XYZIdent* m-file. This routine calls the *Crane3D_CSPenaltyXY* function. The function invokes simulation of the *Crane3D_CSMModelXY* Simulink model (see Fig. 6.6). If you wish to build your own optimisation procedure all source codes are available. The optimisation procedure (called inside *CaseStudy_XYZIdent*) uses the *fmincon* procedure from the MATLAB optimisation library, which finds the constrained minimum of the *Crane3D_CSPenaltyXY* function.

The iterative procedure based on *fminunc* results in matching real system time responses to the simple model responses.

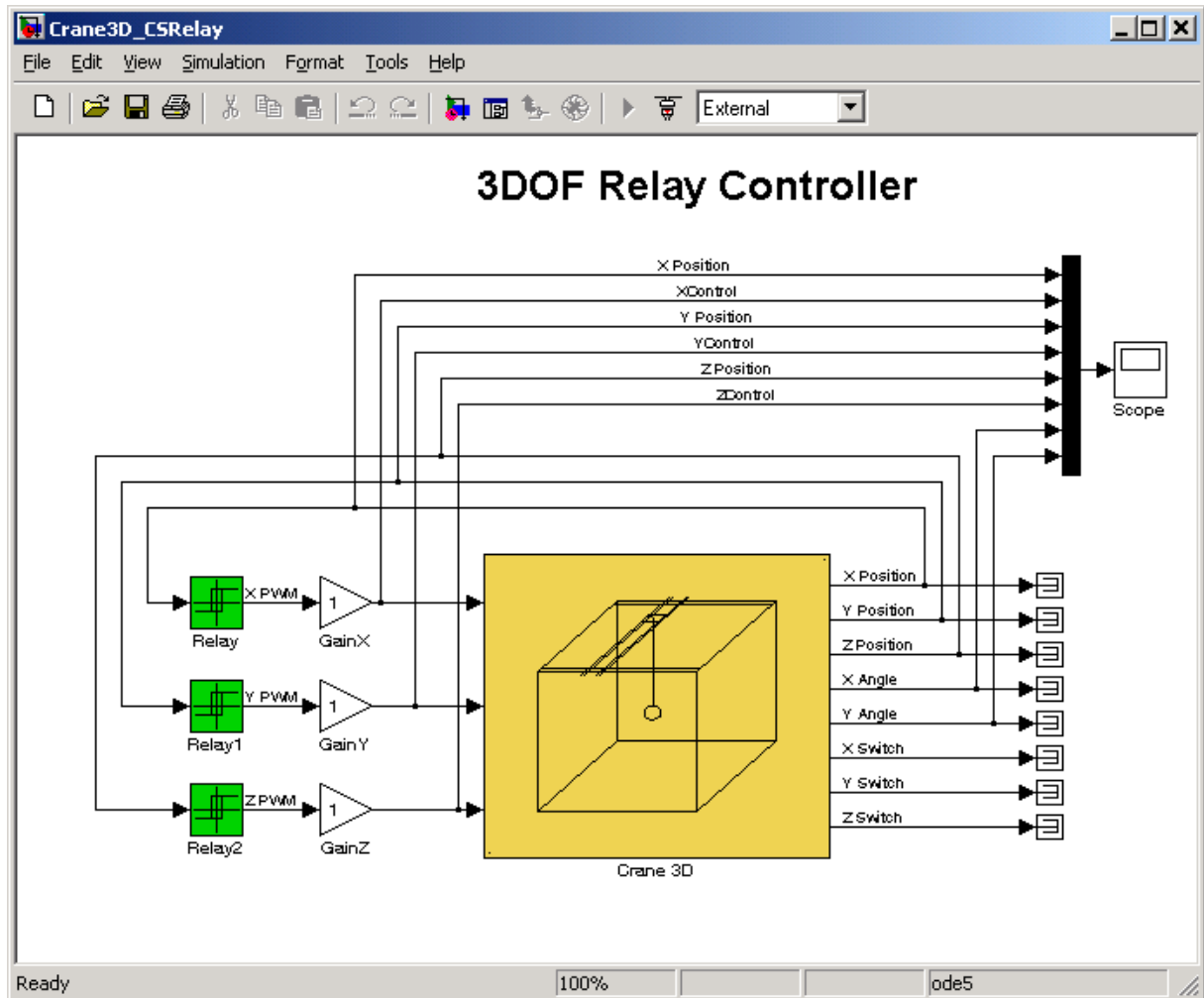


Fig. 6.2 Motion controllers in the X , Y , Z directions

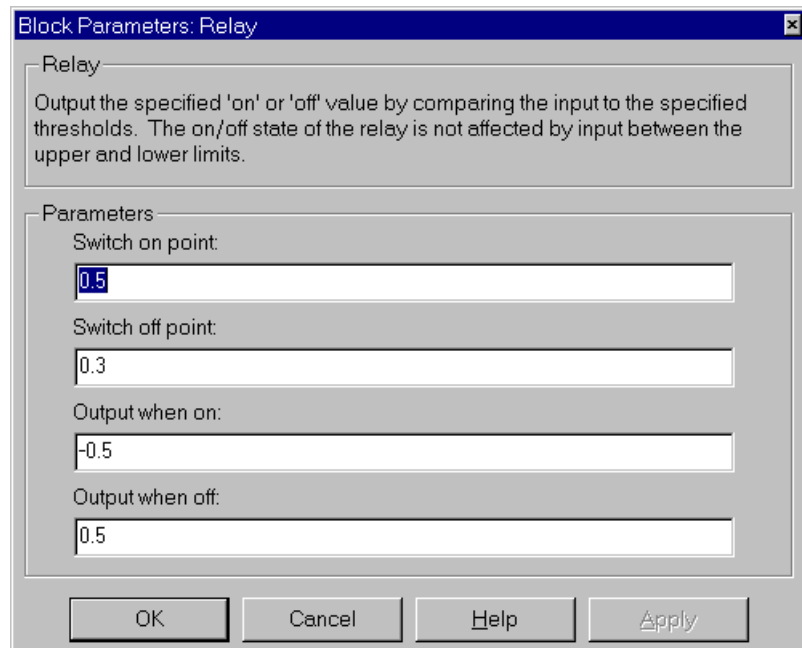


Fig. 6.3 Parameters of the *Relay* block

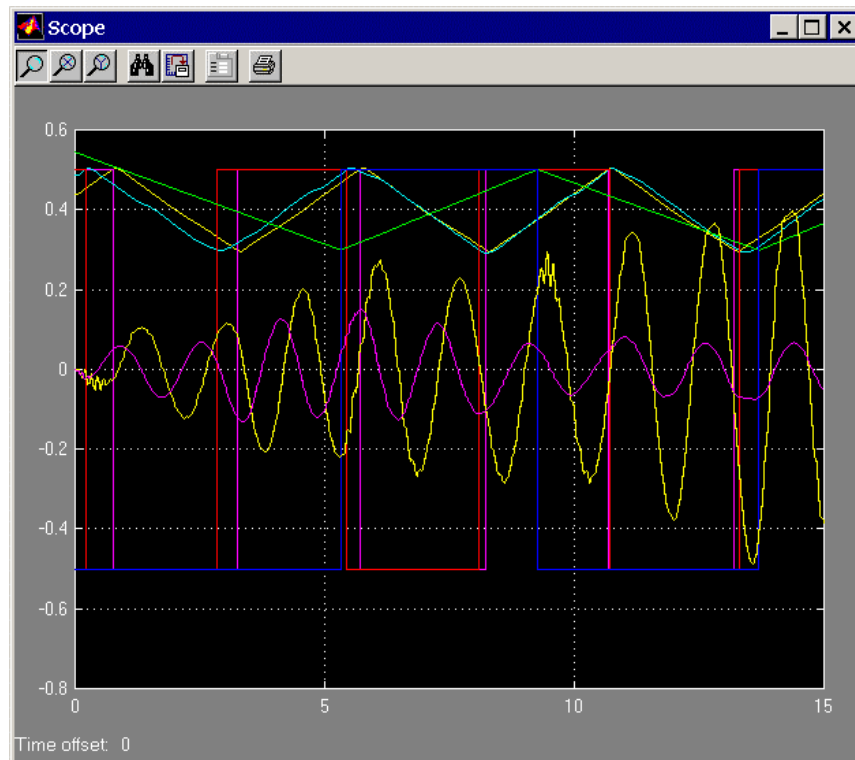


Fig. 6.4 Collected data visible in the *scope* figure

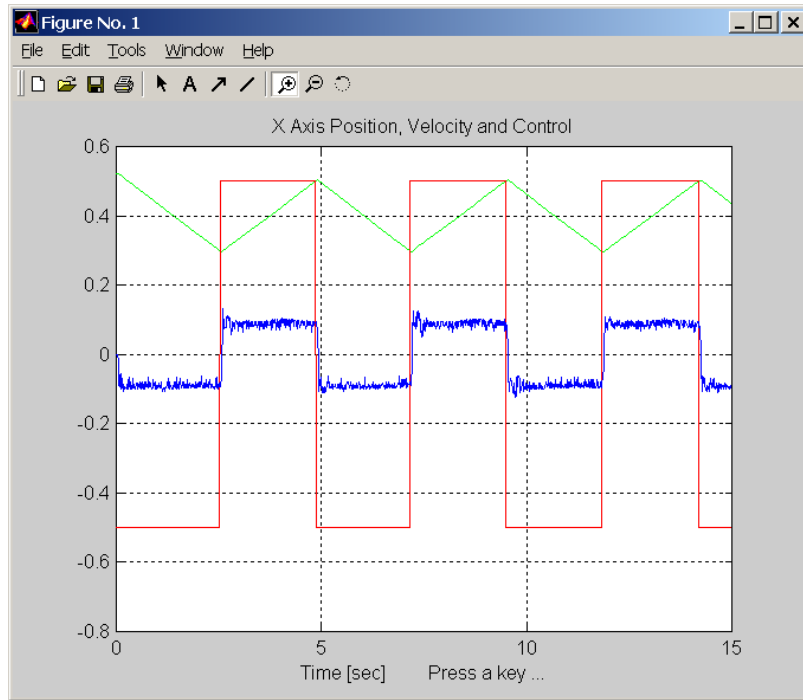


Fig. 6.5 The first plot of the collected data (measurements in the X axis)

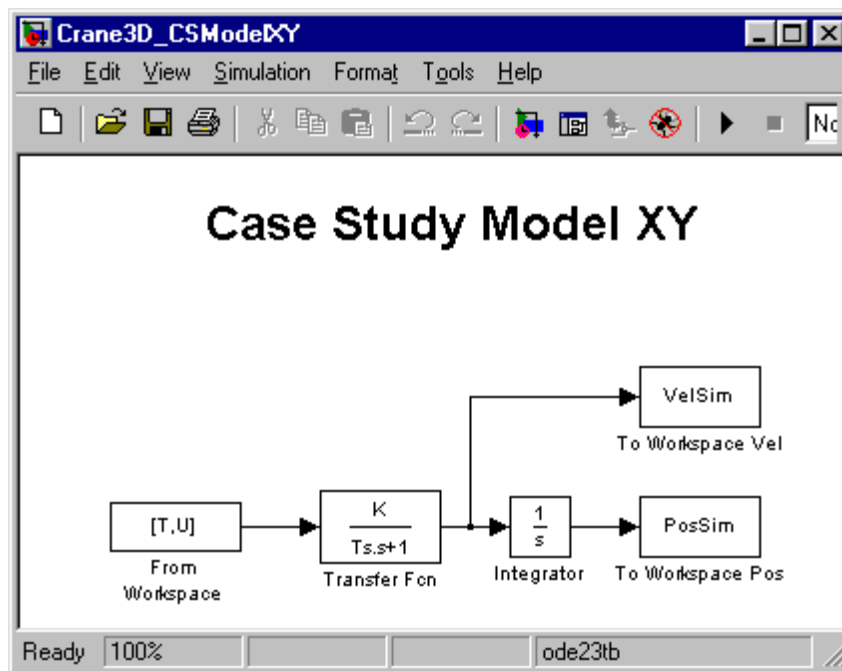


Fig. 6.6 Simple Dynamics in the X or Y directions

Finally, we obtain K_x and T_x parameters of the simple model. In this example they have the following numerical values

$$K_x = 0.173511, \quad T_x = 0.074388.$$

The final plots after optimisation (shown in Fig. 6.7) indicate that the model matching is successful.

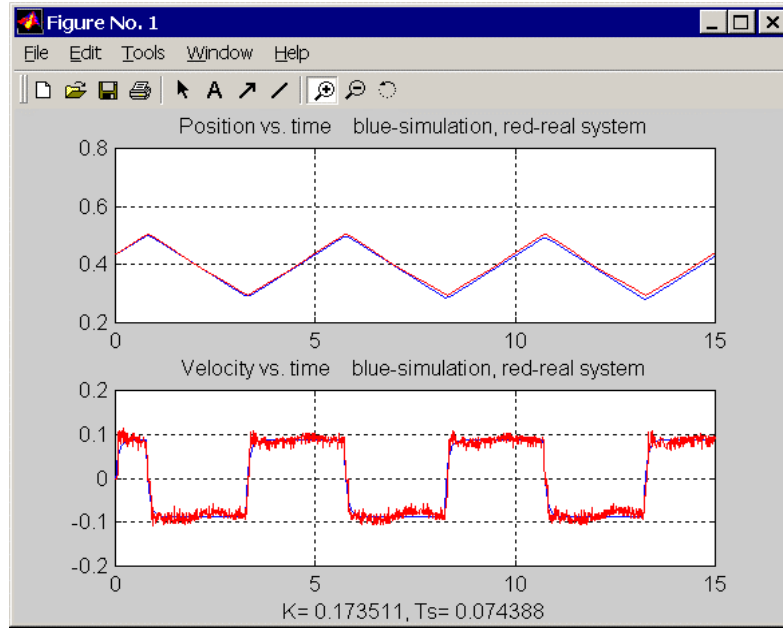


Fig. 6.7 The model results matched to the real data (measurements in the X axis)

After pressing a key again the program starts. The same procedure is repeated in the Y axis. A similar simple linear model of the crane dynamics in Y - axis is assumed

$$G_y(s) = \frac{K_y}{s(T_y s + 1)}.$$

Results obtained for the unmatched model (before optimisation) and the matched model (after optimisation) are visible in Fig. 6.8 and Fig. 6.9.

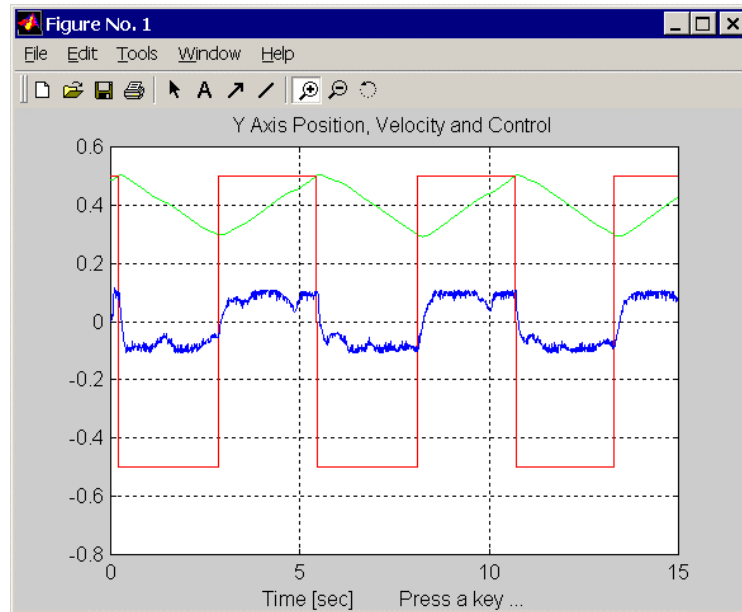


Fig. 6.8 The model results unmatched to the real data (measurements in the Y axis)

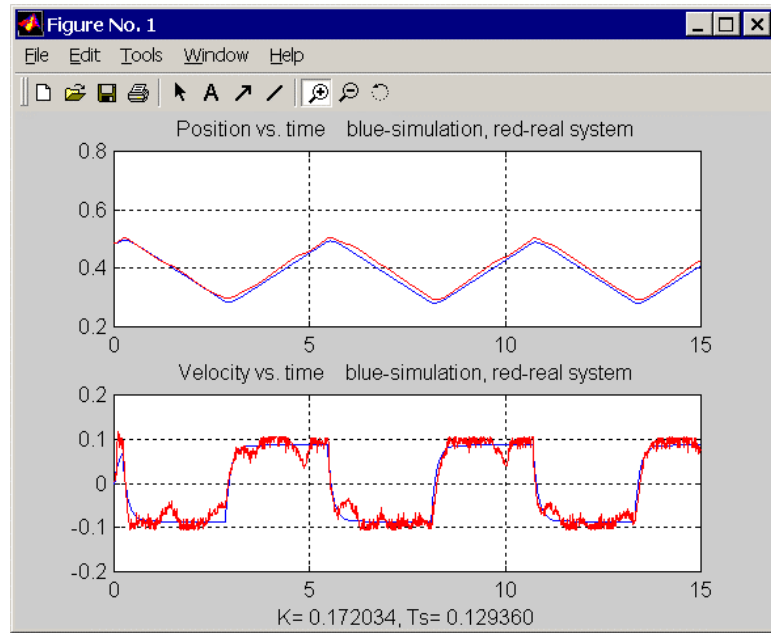


Fig. 6.9 The model results matched to the real data (measurements in the Y axis)

The following numerical values of K_y and T_y were obtains:

$$K_y = 0.172034, \quad T_y = 0.129360.$$

If we press a key again then the model optimisation in the Z direction starts (again the simple model is used). The initial stage before optimisation is visible in Fig. 6.10.

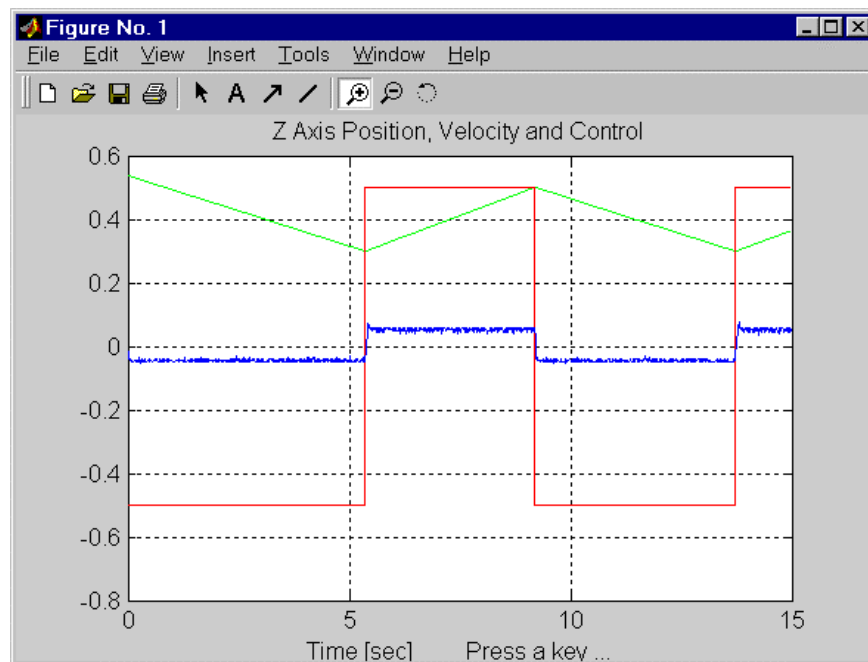


Fig. 6.10 The model results unmatched to the real data (measurements in the Z axis)

After optimisation the following picture is obtained (see Fig. 6.11)

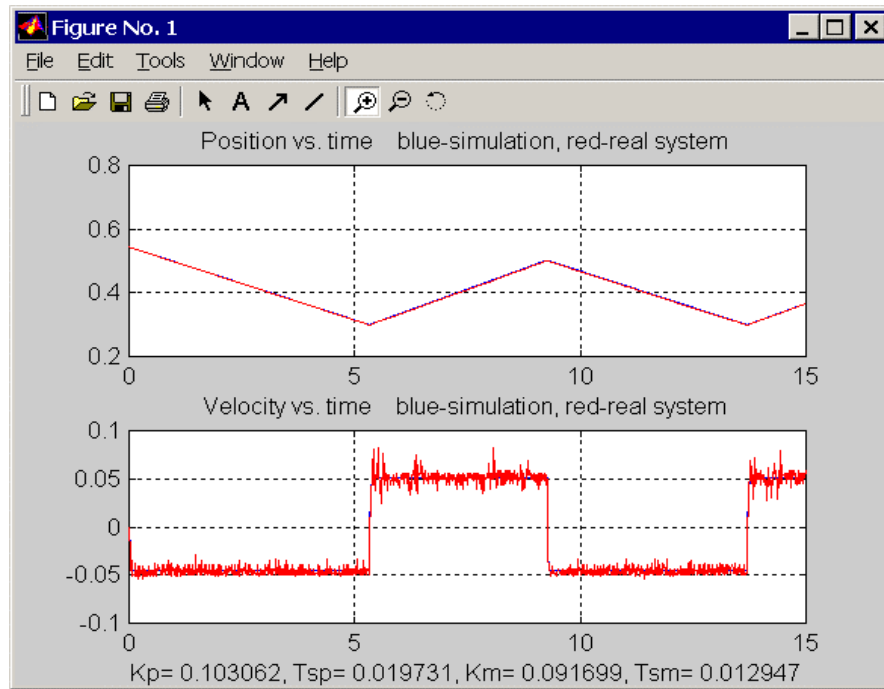


Fig. 6.11 The model results matched to the real data (measurements in the Z axis)

The model matching in the Z direction results in two models identical in form and different in parameters. We have $K_p = 0.103062$ and $T_{sp} = 0.019731$ when the payload is lifted and $K_m = 0.091699$ and $T_{sm} = 0.012947$ when the payload is lowered. The model shown in Fig. 6.12, different from that presented in Fig. 6.6, is used.

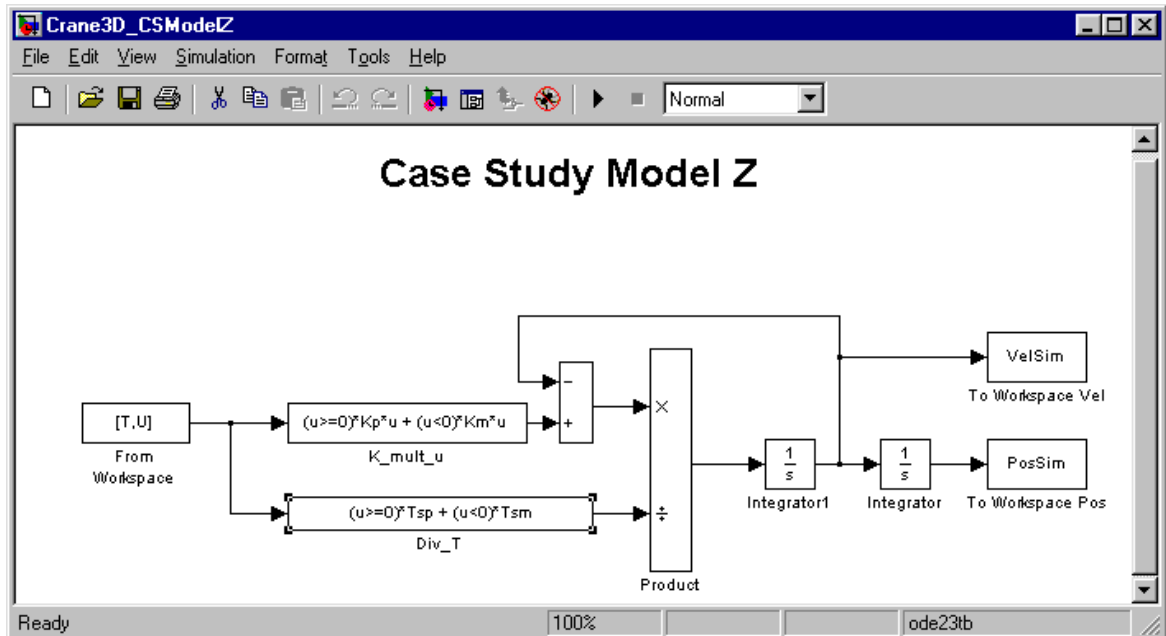


Fig. 6.12 Simple Dynamics in the Z direction

Step 5

Perform the *Go To Center* action. Set the payload motionless. **After that it is necessary to reset the angle encoders.** Go to the *Main Control Window* and reset the angle encoders.

Step 6

Click on the *Collect X Y Data* button. The corresponding controller denoted by *3DCrane Angle Excitation* opens – see Fig. 6.13. Click *Set Address* button and rebuild the model. Next choose successively *Connect to Target* and *Start real-time simulation* from the pull-down menu. The cart and the rail start to move and stop afterwards. This short (in time) motion gives an impulse to the payload to swing. The system collects angles data.

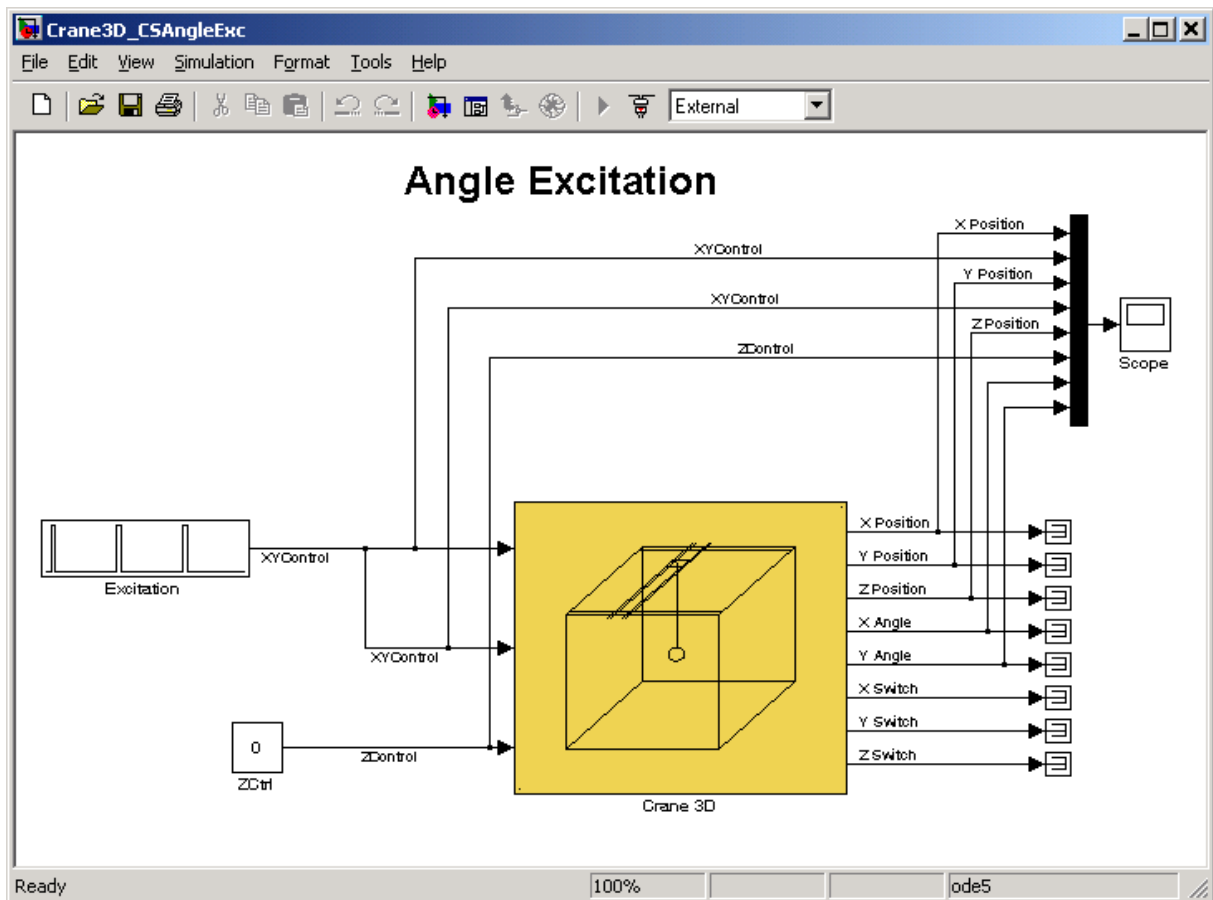


Fig. 6.13 3DCrane Angle Excitation window

The results of the experiment are presented in Fig. 6.14.

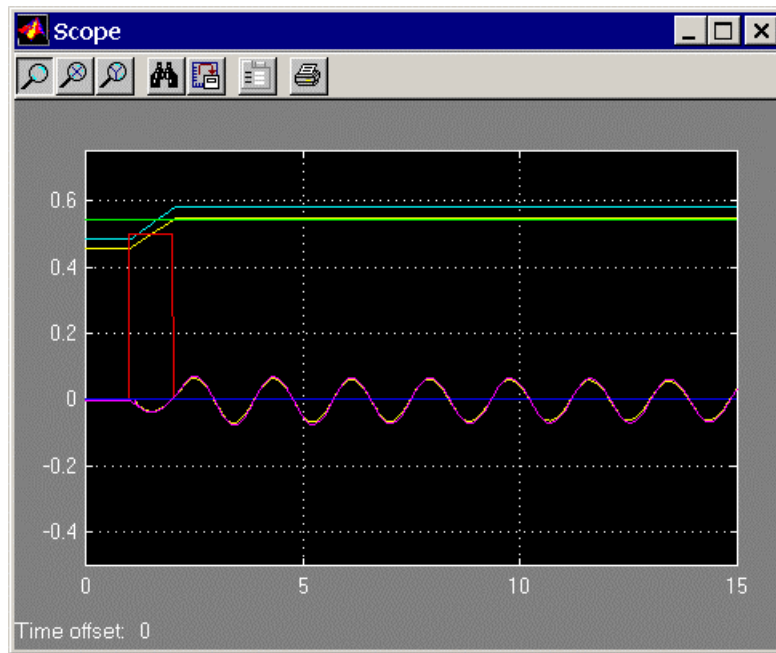


Fig. 6.14 Collected data visible in the *scope* figure

Step 7

Having collected data of angles trajectories you can start the identification of Z displacement. Click on the *Identify Z Displacement* button. The angle data are displayed (see Fig. 6.15)

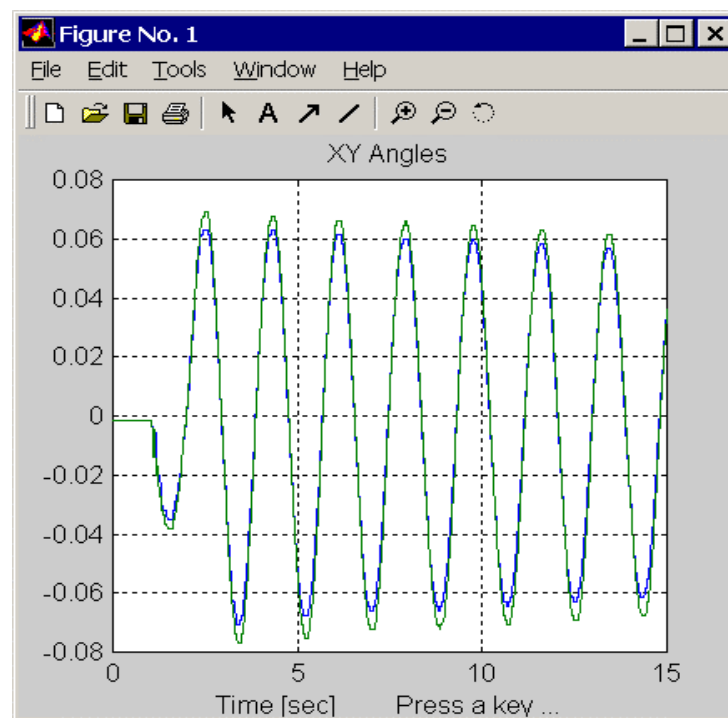


Fig. 6.15 The real data (measurements of *X Y* angles)

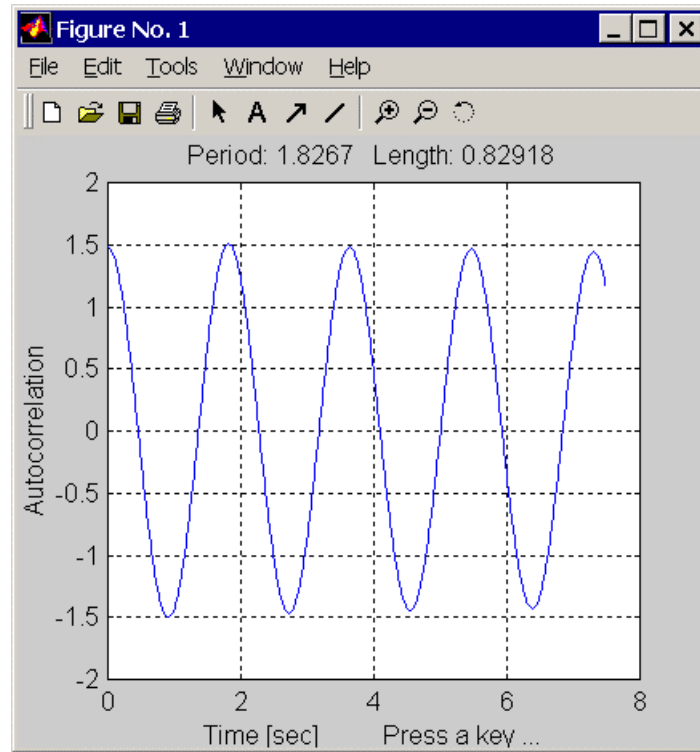


Fig. 6.16 The calculated period and length of the pendulum

The mathematical pendulum model is assumed. The period of oscillations T is calculated from the formula

$$T = 2\pi\sqrt{\frac{l}{g}}$$

where l is the mathematical pendulum length and g is the gravity constant. You obtain the values for the period and the length of the pendulum. They are: 1.826 s and 0.822 m in the presented example.

Step 8

We click on the *PID Optimisation* button to invoke the iterative procedure. This procedure tunes the parameters of the PI controller for each axis using the model given in Fig. 6.17. The performance index is equal to the sum of squares of the differences between the reference signal and the response of the system. The penalty function is also used (see *CaseStudy_penaltyPID.m* file for details).

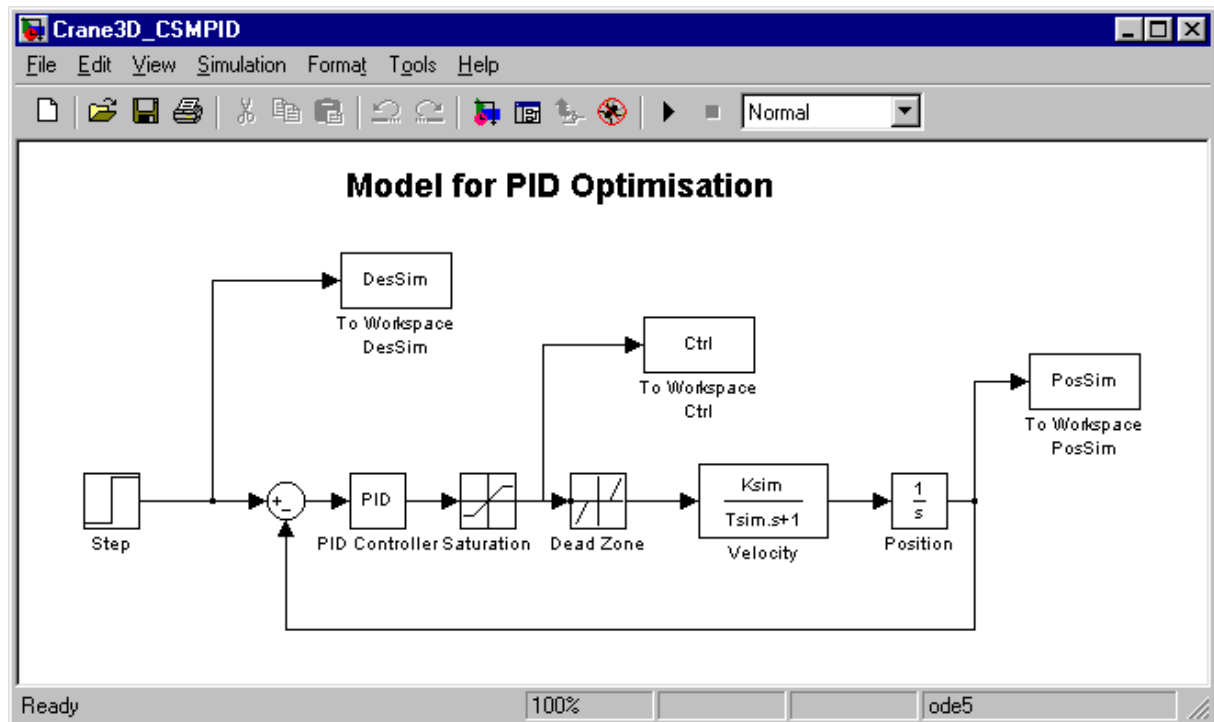


Fig. 6.17 Model for optimisation of the PID parameters

The optimisation runs. The temporary system responses to the iterative parameters K_p and K_i values are displayed in the consecutive three figures. The final results are visible in Fig. 6.18, Fig. 6.19 and Fig. 6.20, and displayed in the MATLAB command window.

In the presented example the PI optimisation results in:

```
[ Kpx Kix ]= [ 32.9328 8.82428e-007 ]
[ Kpy Kiy ]= [ 19.0999 4.74215e-006 ]
[ Kpz Kiz ]= [ 267.3002 3.414063e-007 ]
```

The proportional gains are large. This follows from the fact that the time constants in the previously identified models are very small. The models are thus close to non-dynamical systems. In such a case the closed-loop system requires very large K parameters for PI controllers to track the step reference signals.

Remember that the parameters tuned above relate only to the PI controllers of the desired X , Y and Z cart positions. They do not relate to the PID controllers of the X and Y angles.

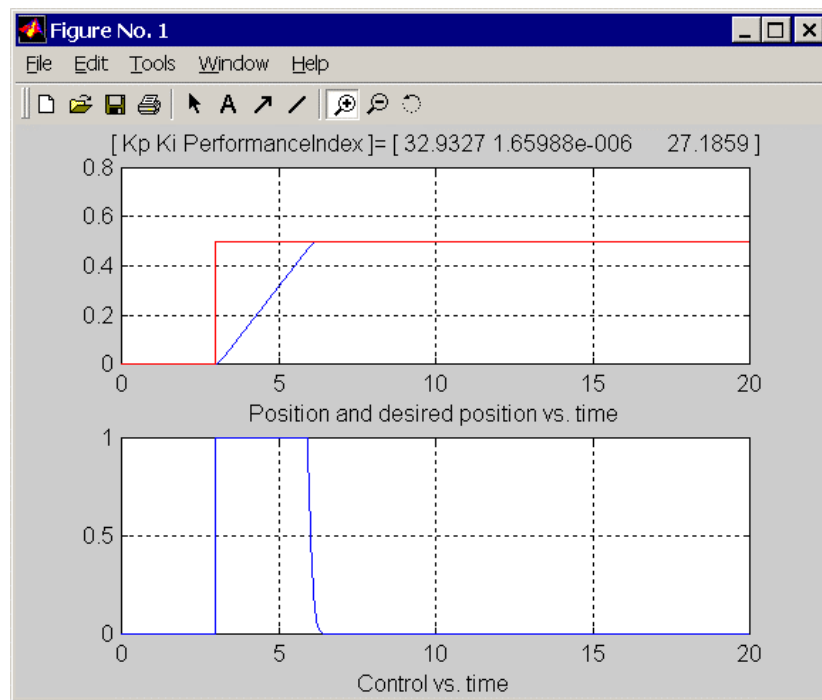


Fig. 6.18 The optimisation run window – the final result for the X direction

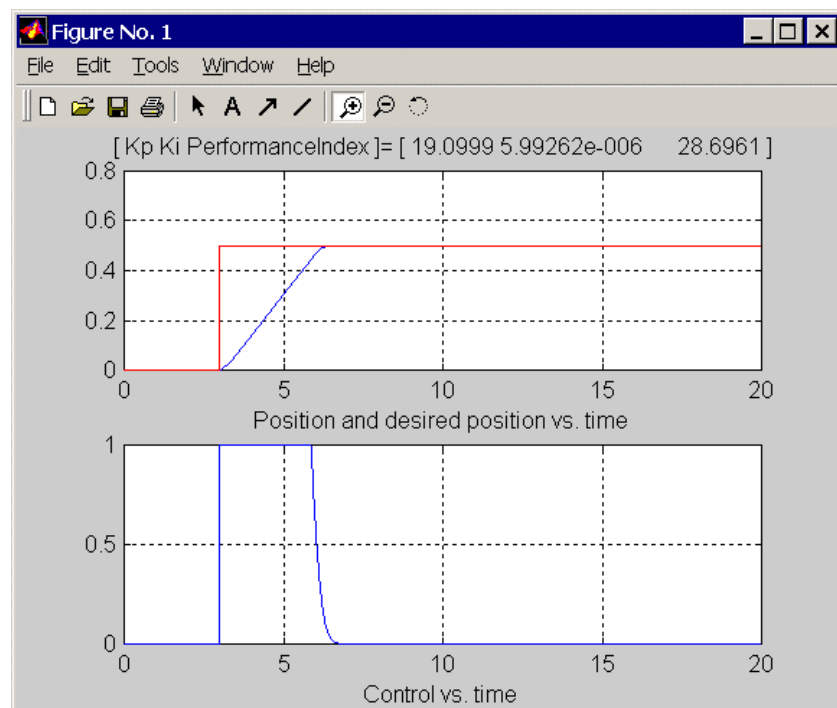


Fig. 6.19 The optimisation run window – the final result for the Y direction

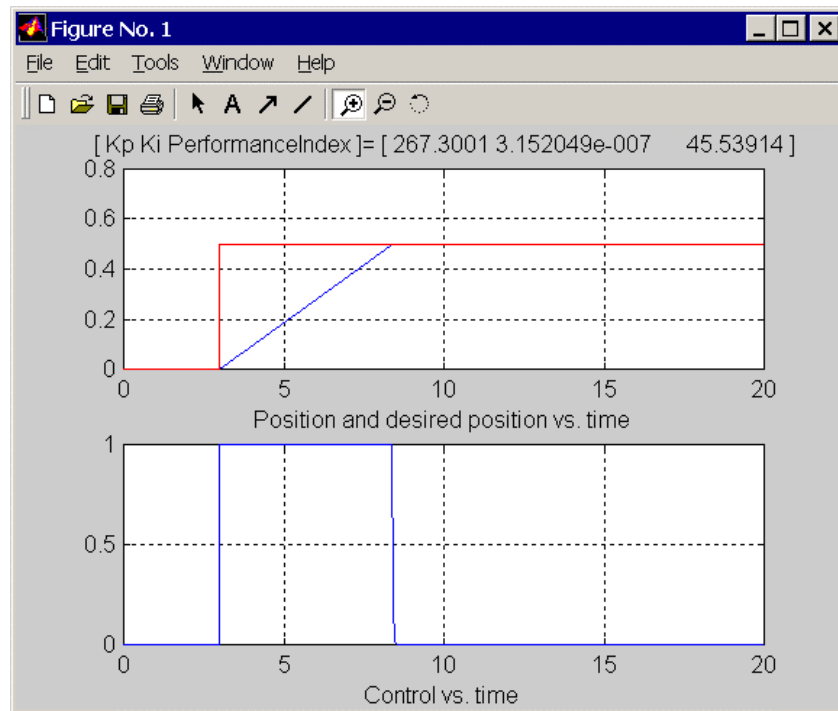


Fig. 6.20 The optimisation run window – the final result for the Z direction

When the optimisation is finished click *5DOF Experiment* button, invoke the *Crane3D_CSPID* model (see Fig. 6.21) and manually introduce the set of new parameters for the *X PID* and *Y PID* controllers.

The *PID* controllers of the *X* and *Y* angles have the default K_p values equal to 20. You may change and enlarge these values up to your requirements. However, do remember that this is a trade-off between tracking the desired cart trajectory and the payload stabilisation. Too high K_p values may result in an unstable behaviour of the crane.

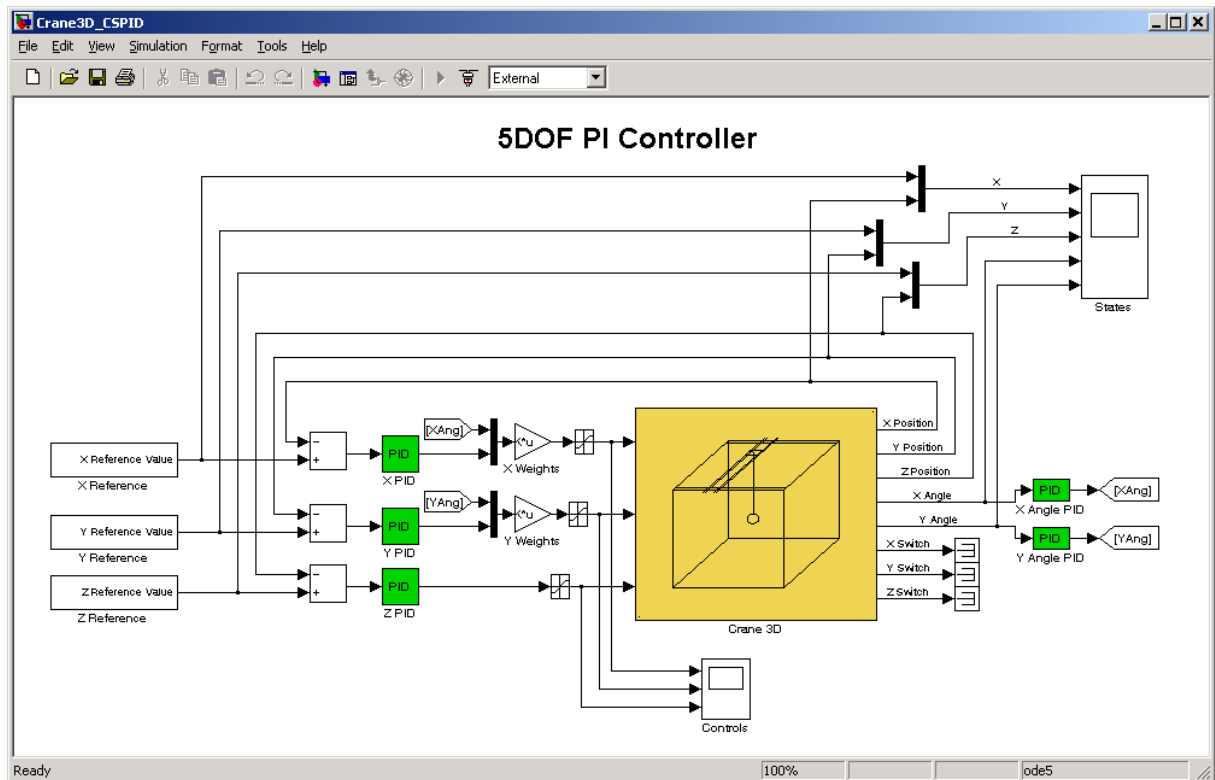


Fig. 6.21 The Case Study 5DOF PID controller window

Rebuild the model, then open two scopes and start the real-time experiment. The results of the experiment are visible in the scopes (see Fig. 6.22 and Fig. 6.23).

The cart is tracking the reference signals in the form of square waves. Abrupt changes in the cart movement result in oscillations of the payload. These oscillations are immediately damped due to the P angle controllers operation.

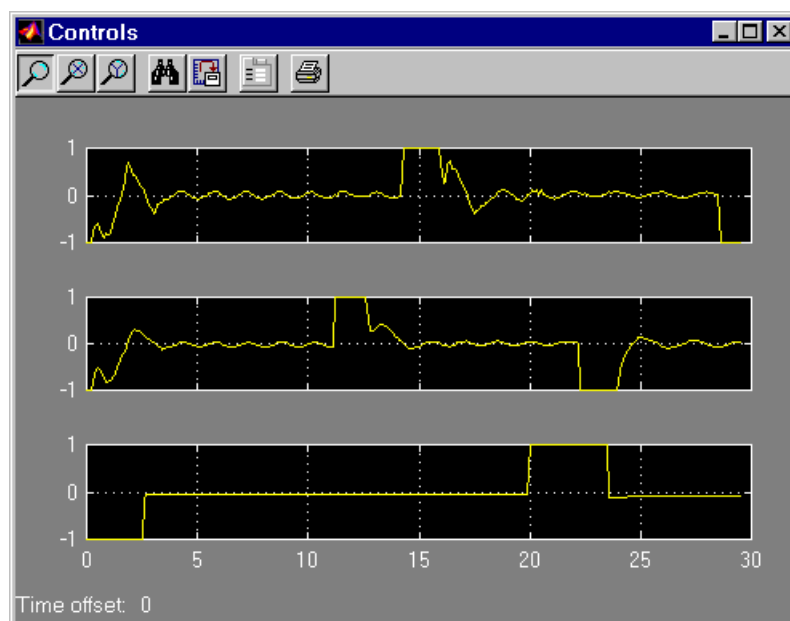


Fig. 6.22 The X, Y and Z controls visible in the *Controls* scope

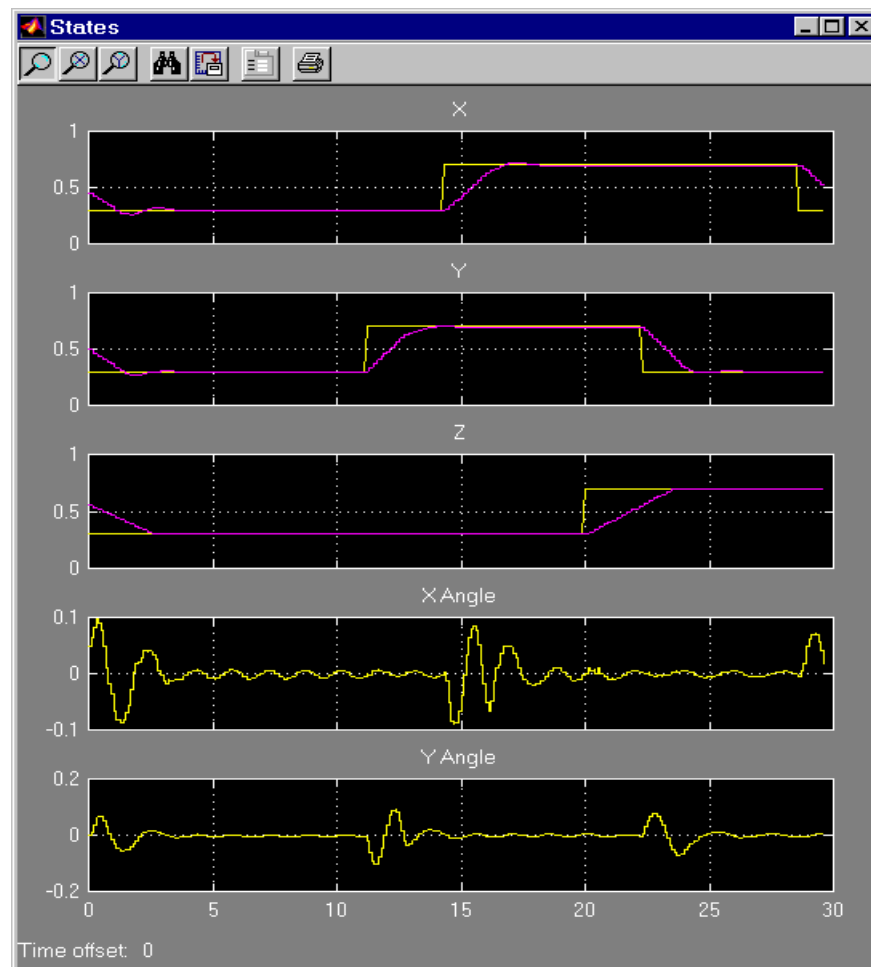


Fig. 6.23 The X, Y, Z positions and X, Y angles visible in the *States* scope

7. Prototyping your own controller in RTWT environment

In this section the process of building your own control system is described. The *Real Time Windows Target* (RTWT) toolbox is used. An example how to use the Crane3D software is shown later in section 5.3. In this section we give indications how to proceed in the RTWT environment.



Before start, test your MATLAB configuration and compiler installation by building and running an example of real-time application. Real-time Windows Target includes the model `rtvdp.mdl`. Running this model will test the installation by running Real-Time Workshop, your third-party C compiler, Real-Time Windows Target, and the Real-Time Windows Target kernel. 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.

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 *3DCrane Device Driver* and other blocks chosen from the Simulink library,
- build the executable file under RTWT (see the pop-up menus in Fig. 5.1)

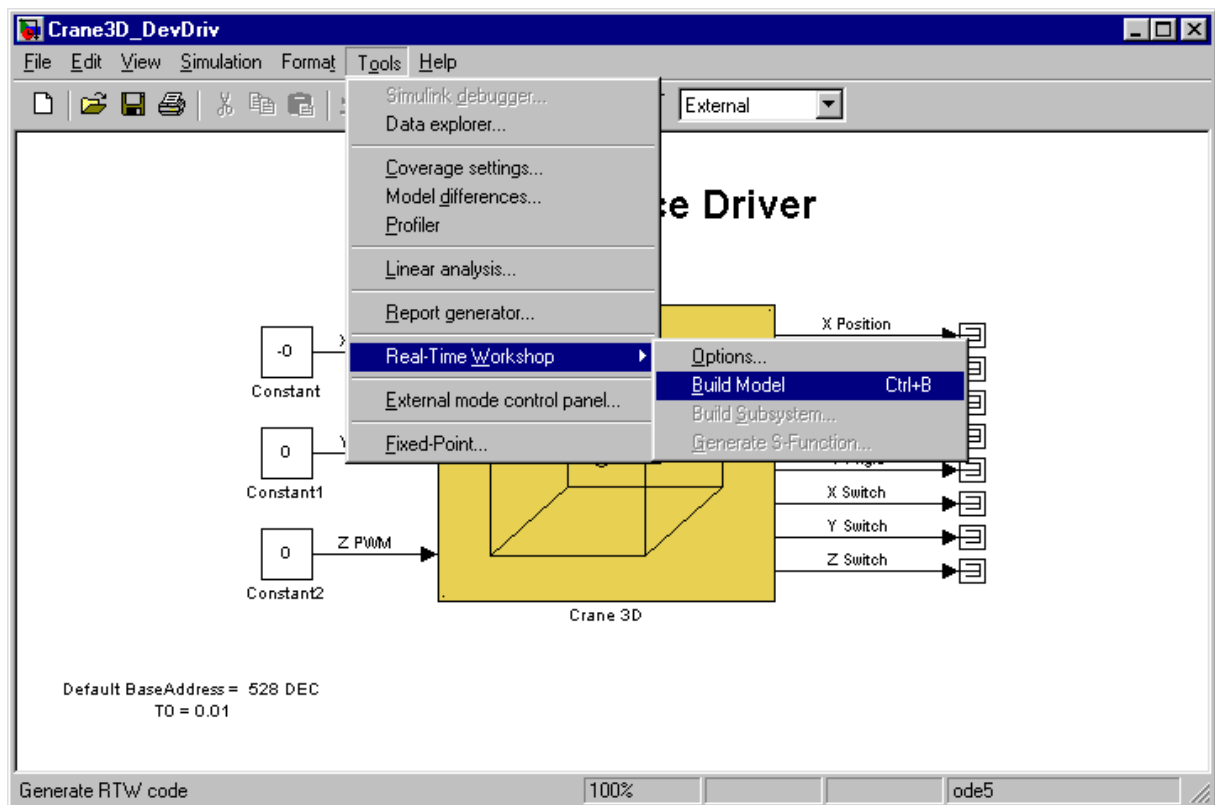


Fig. 7.1 Creating the executable file under RTWT

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

7.1. Creating a model

The simplest way to create a Simulink model of the control system is to use one of the models included in the *Main Control Window* as a template. For example, click on the *Pxyz* button and save it as *MySystem.mdl* name. The *MySystem* Simulink model is shown in Fig. 7.2.

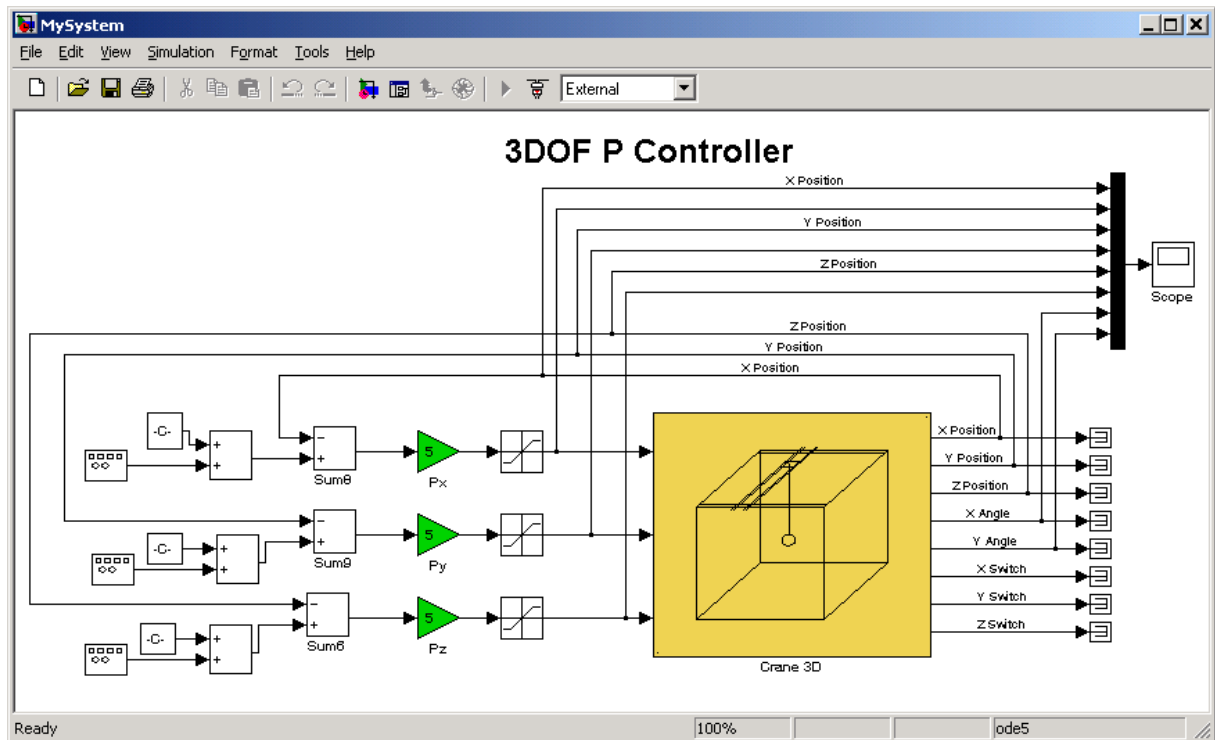


Fig. 7.2 The *MySystem* Simulink model

Now, you can modify the model. You have absolute freedom to develop your own controller. Remember to leave the *3DCrane* driver model and the *Set Base Address* button in the window. This is necessary to activate transferring the base address of the I/O card to the model as a MATLAB workspace variable.

Though it is not obligatory, we recommend you to live the multiplexer with the scope and the control saturation blocks. You need a scope to watch how the system runs. You also need the saturation blocks to constraint the controls to match the maximal PWM signals sent to the DC motors. The saturation blocks are built in the *Crane 3D* driver block. They limit currents to DC motors for safety reasons. However they are not visible for the user who may amaze at the saturation of controls. Other blocks remaining in the window are not necessary for our new project.

Creating your own model on the basis of an 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 *3DCrane Device Driver* into the real-time code a special make-file is required. This file is included to the *3DCrane* software.

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

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. This is a good choice to get enough points to describe the signal behaviour and to save the computer memory.

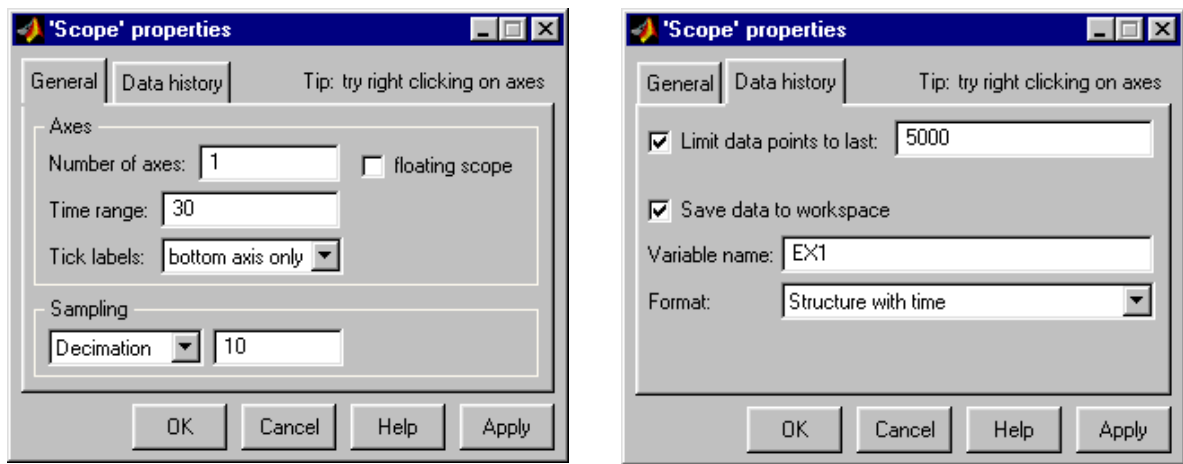


Fig. 7.3 Setting the parameters of the *Scope* block

When the Simulink model is ready, click the *Tools/External Mode Control Panel* option and next click the *Signal Triggering* button. The window presented 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 close the window.

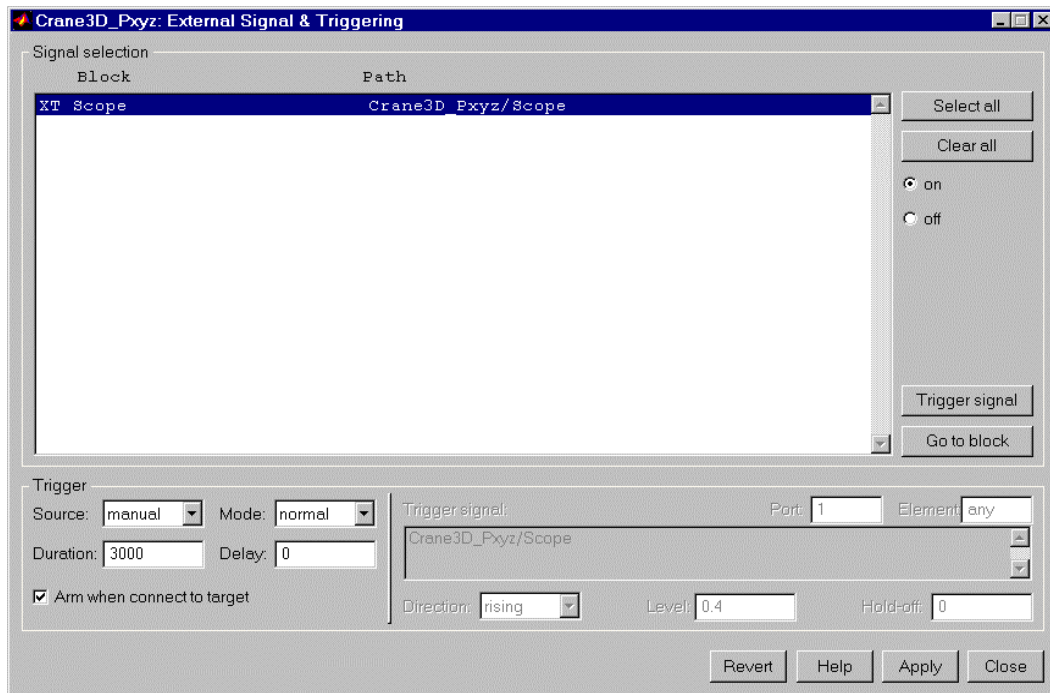


Fig. 7.4 *External Signal & Triggering* window

7.2. Code generation and the build process

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

The code is generated by the use of Target Language Compiler (TLC) (see description of the Simulink Target Language). The make-file 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. The RTW page appears when you select the *RTW* tab (Fig. 7.5). The *RTW* page allows you to set the real-time build options and then to start the building process of the *RTW.DLL* executable file.

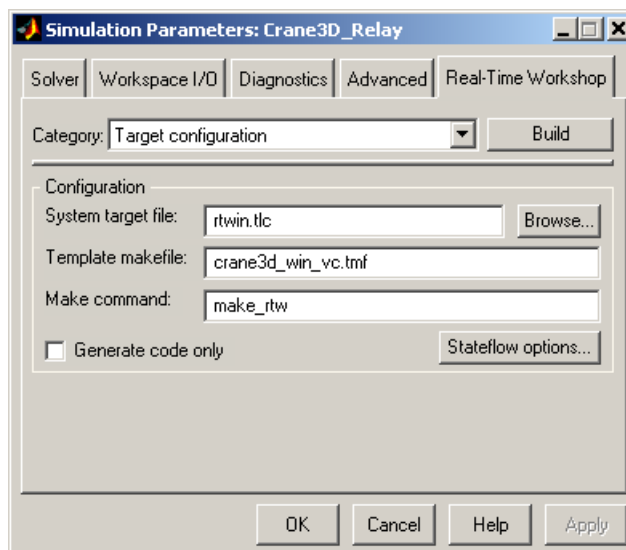



Fig. 7.5 RTW page of the *Simulation parameters* dialog box (Matlab 6.5)

The system target file name is *rtwin.tlc*. It manages the code generation process. The *crane3d_win_vc.tmf* template makefile is responsible for C code generation using the Visual C/C++ compiler.

There are three options which have to be properly marked as shown in Fig. 7.5:

- *Inline parameters* - not used when building a real-time program,
- *Retain.rtw* file - if marked, auxiliary information is stored in the file (with .rtw extension),
- *Generate code only* - if marked, a code is generated but compilation is not performed.

The *Solver* page appears when you select the *Solver* tab (Fig. 7.7). The *Solver* page 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 driver library remember that different sampling periods must have a common divider.**

If the Matlab 7 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 is shown in Fig. 7.6. Notice that *rtwin.tmf* template makefile is used. This file is default one for RTWT building process.

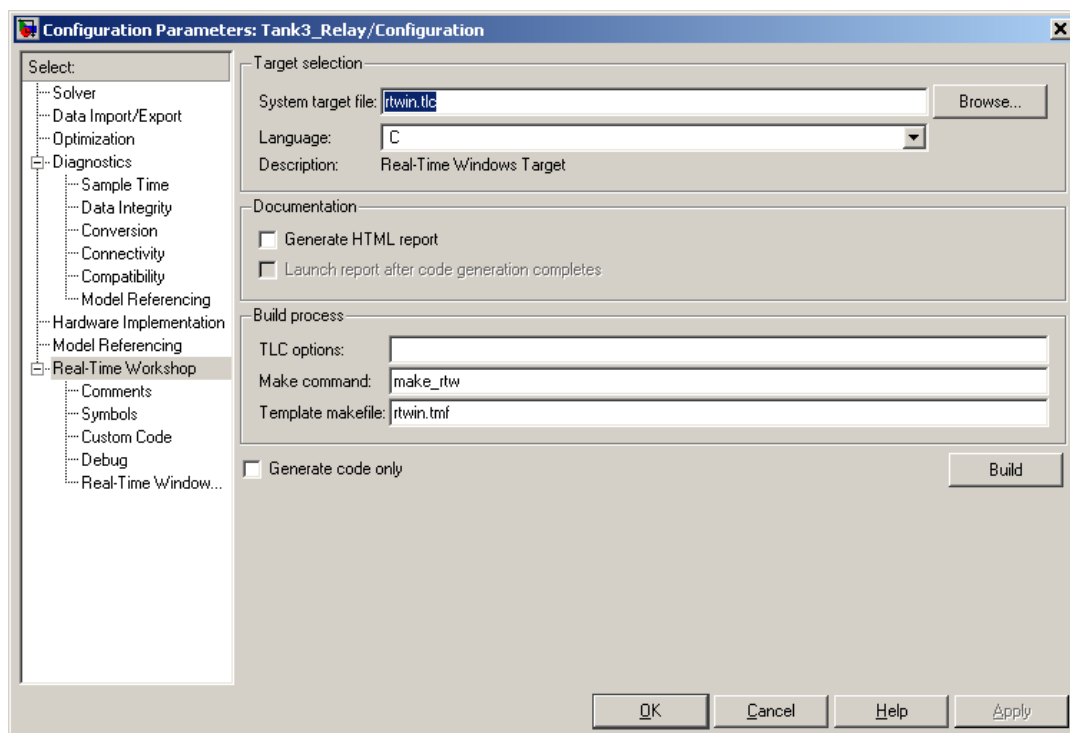


Fig. 7.6 *Configuration parameters* page

The *Start time* has to be set to 0. The solver has to be selected. In our example the fifth-order integration method – *ode5* is chosen.

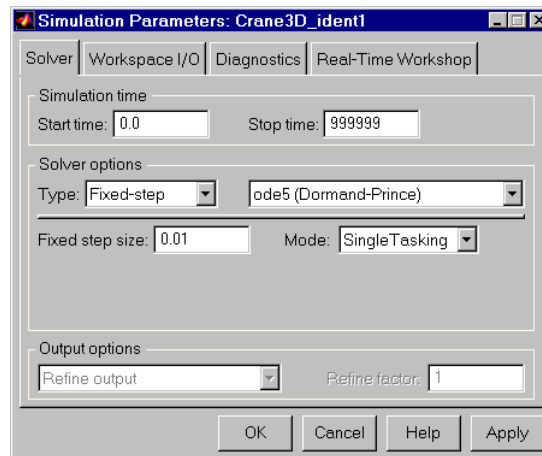


Fig. 7.7 Simulation parameters

If all parameters are set properly you can start the DLL executable building process. For this purpose press the *Build* push button on the RTW page (Fig. 7.5).

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.



Before starting the experiment set the initial position of the cart, the rail and the payload in a safe zone. The *Go Home* and *Go To Center* buttons are applied to fulfil these tasks.

8. Mathematical model of the 3DCrane

The schematic diagram of the crane is given in Fig. 8.1.

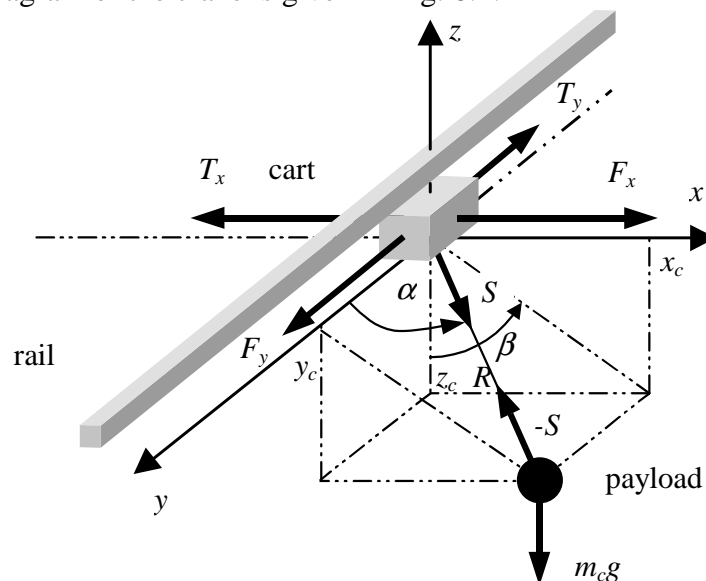


Fig. 8.1 3DCrane system: coordinates and forces

There are five measured quantities:

- x_w (not marked in Fig. 8.1) denotes the distance of the rail with the cart from the center of the construction frame;
- y_w (not marked in Fig. 8.1) denotes the distance of the cart from the center of the rail;
- R denotes the length of the lift-line;
- α denotes the angle between the y axis and the lift-line;
- β denotes the angle between the negative direction on the z axis and the projection of the lift-line onto the xz plane.

Denote also:

m_c	-	mass of the payload
m_w	-	mass of the cart
m_s	-	mass of the moving rail
x_c, y_c, z_c	-	coordinates of the payload
S	-	reaction force in the lift-line acting on the cart
F_x	-	force driving the rail with cart
F_y	-	force driving the cart along the rail
F_R	-	force controlling the length of the lift-line
T_x, T_y, T_R	-	friction forces.

8.1. Basic relationships

An important element in the construction of mathematical model is the appropriate choice of the system of coordinates. The Cartesian system, although simple in interpretation and determining the position in space in a unique way in both directions, is not convenient for the description of the dynamics of rotational motion. The spherical system has therefore been adopted. The position of the payload is described by two angles, α and β , shown in Fig. 8.1. A drawback of the spherical system of coordinates is that for every point on the y-axis, the corresponding value of β is not uniquely determined. However, the points on the y-axis are not attainable in real crane systems.

The following symbols are used in the sequel

$$\begin{aligned}\mu_1 &= \frac{m_c}{m_w}, & \mu_2 &= \frac{m_c}{m_w + m_s} \\ u_1 &= \frac{F_y}{m_w}, & u_2 &= \frac{F_x}{m_w + m_c}, & u_3 &= \frac{F_R}{m_c} \\ T_1 &= \frac{T_y}{m_w}, & T_2 &= \frac{T_x}{m_w + m_c}, & T_3 &= \frac{T_R}{m_c} \\ N_1 &= u_1 - T_1, & N_2 &= u_2 - T_2, & N_3 &= u_3 - T_3 \\ s &= \frac{S}{m_c}.\end{aligned}$$

The position of the payload is described by the equalities

$$x_c = x_w + R \sin \alpha \sin \beta \quad (1)$$

$$y_c = y_w + R \cos \alpha \quad (2)$$

$$z_c = -R \sin \alpha \cos \beta. \quad (3)$$

The dynamics of the crane is given by the equations (Fig. 8.1)

$$m_c \ddot{x}_c = -S_x \quad (4)$$

$$m_c \ddot{y}_c = -S_y \quad (5)$$

$$m_c \ddot{z}_c = -S_z - m_c g \quad (6)$$

$$(m_w + m_s) \ddot{x}_w = F_x - T_x + S_x \quad (7)$$

$$m_w \ddot{y}_w = F_y - T_y + S_y \quad (8)$$

where S_x , S_y and S_z are the components of the vector S

$$S_x = S \sin \alpha \sin \beta \quad (9)$$

$$S_y = S \cos \alpha \quad (10)$$

$$S_z = -S \sin \alpha \cos \beta. \quad (11)$$

It is assumed that the lift-line is always stretched, that is,

$$S_x(x_c - x_w) + S_y(y_c - y_w) + S_z z_c > 0. \quad (12)$$

In the case where the payload is lifted and lowered with the use of the control force F_R , S (4) - (12) should be replaced as follows

$$S = F_R - T_R \quad (13)$$

8.2. Simplified model with three control forces

Assume that the deviation of the payload from the z -axis is small. Then

$$\cos \alpha = \cos\left(\frac{\pi}{2} + \Delta\alpha\right) \cong -\Delta\alpha \quad (14)$$

$$\sin \alpha = \sin\left(\frac{\pi}{2} + \Delta\alpha\right) \cong 1 \quad (15)$$

$$\cos \beta \cong 1 \quad (16)$$

$$\sin \beta \cong \Delta\beta. \quad (17)$$

Equations (9) - (11) take the form

$$S_x = S\Delta\beta \quad (18)$$

$$S_y = -S\Delta\alpha \quad (19)$$

$$S_z = -S. \quad (20)$$

Substituting (18) - (20) and (13) in (4) - (8) we obtain

$$\ddot{x}_c = -(u_3 - T_3)\Delta\beta \quad (21)$$

$$\ddot{y}_c = (u_3 - T_3)\Delta\alpha \quad (22)$$

$$\ddot{z}_c = u_3 - T_3 - g \quad (23)$$

$$\ddot{x}_w = u_2 - T_2 + (u_3 - T_3)\mu_2\Delta\beta. \quad (24)$$

$$\ddot{y}_w = u_1 - T_1 - (u_3 - T_3)\mu_1\Delta\alpha \quad (25)$$

With the simplification (14) - (17), the position of the payload satisfies

$$x_c = x_w + R\Delta\beta \quad (26)$$

$$y_c = y_w - R\Delta\alpha \quad (27)$$

$$z_c = -R. \quad (28)$$

The acceleration of the payload is given by

$$\ddot{x}_c = \ddot{x}_w + \ddot{R}\Delta\beta + 2\dot{R}\dot{\Delta\beta} + R\ddot{\Delta\beta} \quad (29)$$

$$\ddot{y}_c = \ddot{y}_w - \ddot{R}\Delta\alpha - 2\dot{R}\Delta\dot{\alpha} - R\Delta\ddot{\alpha} \quad (30)$$

$$\ddot{z}_c = -\ddot{R}. \quad (31)$$

After the substitution of equalities (29) - (31) to (21) - (25), five equations with five unknowns $\Delta\ddot{\alpha}$, $\Delta\ddot{\beta}$, \ddot{x}_w , \ddot{y}_w and \ddot{R} are obtained. The solution of this set of equations with respect to the second derivatives and the introduction of new variables

$$\begin{aligned} x_1 &= y_w & x_6 &= \dot{x}_5 = \Delta\dot{\alpha} \\ x_2 &= \dot{x}_1 = \dot{y}_w & x_7 &= \Delta\dot{\beta} \\ x_3 &= x_w & x_8 &= \dot{x}_7 = \Delta\dot{\beta} \\ x_4 &= \dot{x}_3 = \dot{x}_w & x_9 &= R \\ x_5 &= \Delta\alpha & x_{10} &= \dot{x}_9 = \dot{R} \end{aligned}$$

leads to the final, simplified system of state equations for the 3D crane

$$\dot{x}_1 = x_2 \quad (32)$$

$$\dot{x}_2 = N_1 - \mu_1 x_5 N_3 \quad (33)$$

$$\dot{x}_3 = x_4 \quad (34)$$

$$\dot{x}_4 = N_2 + \mu_2 x_7 N_3 \quad (35)$$

$$\dot{x}_5 = x_6 \quad (36)$$

$$\dot{x}_6 = (N_1 - \mu_1 x_5 N_3 - g x_5 - 2x_6 x_{10})/x_9 \quad (37)$$

$$\dot{x}_7 = x_8 \quad (38)$$

$$\dot{x}_8 = -(N_2 + \mu_2 x_7 N_3 + g x_7 + 2x_8 x_{10})/x_9 \quad (39)$$

$$\dot{x}_9 = x_{10} \quad (40)$$

$$\dot{x}_{10} = -N_3 + g. \quad (41)$$

The proposed simplification results in a partial separation of the equations of the crane. The equations which describe the motion of the crane along the y-axis, that is, the equations for x_1, x_2, x_5, x_6 , are not connected with the equations for the variables x_3, x_4, x_7, x_8 , describing the motion along the x-axis. The crane can be thus treated as two independent subsystems. This separation is partial, as both these subsystems depend on x_9 – the length of the lift-line R . A complete separation takes place if $x_9 = R = \text{const}$ and T_x, T_y are separated. Note that cranes are always controlled in such a way that the swinging of the payload is suppressed, and so the deviation angles are small. The simplified model is then adequate.

8.3. Complete nonlinear model with constant pendulum length and two control forces

We will now derive the crane equations without the simplifications of equations (14) - (17), but assuming that the length of the lift-line R is constant and there is no control force F_R . Putting formulas (9) - (11) into (4) - (8), we obtain

$$\ddot{x}_c = -s \sin \alpha \sin \beta \quad (42)$$

$$\ddot{y}_c = -s \cos \alpha \Rightarrow s = -\frac{\ddot{y}_c}{\cos \alpha} \quad (43)$$

$$\ddot{z}_c = s \sin \alpha \cos \beta - g \quad (44)$$

$$\ddot{x}_w = u_2 - T_2 + s \mu_2 \sin \alpha \sin \beta. \quad (45)$$

$$\ddot{y}_w = u_1 - T_1 + s \mu_1 \cos \alpha \quad (46)$$

The position of the payload is given by (1) – (3). Its acceleration satisfies

$$\ddot{x}_c = \ddot{x}_w + R(\ddot{\alpha} \cos \alpha \sin \beta + \ddot{\beta} \sin \alpha \cos \beta - \quad (47)$$

$$-(\dot{\alpha}^2 + \dot{\beta}^2) \sin \alpha \sin \beta + 2\dot{\alpha}\dot{\beta} \cos \alpha \cos \beta) \quad (48)$$

$$\ddot{y}_c = \ddot{y}_w - R(\ddot{\alpha} \sin \alpha + \ddot{\alpha}^2 \cos \alpha) \quad (49)$$

$$\ddot{z}_c = -R(\ddot{\alpha} \cos \alpha \cos \beta - \ddot{\beta} \sin \alpha \sin \beta - \quad (49)$$

$$-(\dot{\alpha}^2 + \dot{\beta}^2) \sin \alpha \cos \beta - 2\dot{\alpha}\dot{\beta} \cos \alpha \sin \beta).$$

After the elimination of s , that is, the substitution of (32) into equations (33) – (36) we get a set of four equations

$$\ddot{x}_c = \ddot{y}_c \operatorname{tg} \alpha \sin \beta \quad (50)$$

$$\ddot{z}_c = -\ddot{y}_c \operatorname{tg} \alpha \cos \beta - g \quad (51)$$

$$\ddot{x}_w = u_2 - T_2 - \mu_2 \ddot{y}_c \operatorname{tg} \alpha \sin \beta. \quad (52)$$

$$\ddot{y}_w = u_1 - T_1 - \mu_1 \ddot{y}_c \quad (53)$$

Using equations (47) - (49) in (50) - (53) we arrive at a system of four equations with four unknowns $\ddot{x}_w, \ddot{y}_w, \ddot{\alpha}$ and $\ddot{\beta}$. The solution of this system with respect to the second derivatives and the introduction of new variables gives the final description in the form of nonlinear state equations. We introduce the notations

$$\begin{aligned} x_1 &= y_w & x_5 &= \alpha \\ x_2 &= \dot{x}_1 = \dot{y}_w & x_6 &= \dot{x}_5 = \dot{\alpha} \\ x_3 &= x_w & x_7 &= \beta \\ x_4 &= \dot{x}_3 = \dot{x}_w & x_8 &= \dot{x}_7 = \dot{\beta} \end{aligned}$$

$$s_n \equiv \sin x_n$$

$$c_n \equiv \cos x_n$$

$$A = 1 + \mu_1 c_5^2 + \mu_2 s_5^2 s_7^2$$

$$B = 1 + \mu_1$$

$$V_1 = \mu_1 R c_5 (x_6^2 + s_5^2 x_8^2) + \mu_1 g c_5 s_5 c_7$$

$$V_2 = \mu_2 R s_7 (s_5^3 x_8^2 + s_5 x_6^2) + \mu_2 g s_5^2 c_7 s_7$$

$$V_3 = B g c_5 c_7 + B R c_5 s_5 x_8^2 + (\mu_1 - \mu_2 s_7^2) R c_5 s_5 x_6^2$$

$$V_4 = \mu_2 R s_5 c_7 s_7 (x_6^2 + s_5^2 x_8^2) + (B + (\mu_2 - \mu_1) s_5^2) g s_7 + 2 R A c_5 x_6 x_8.$$

Finally, we obtain eight state equations describing the dynamics of the crane with constant pendulum length

$$\dot{x}_1 = x_2 \quad (54)$$

$$\dot{x}_2 = ((1 + \mu_2 s_5^2 s_7^2) N_1 - \mu_1 c_5 s_5 s_7 N_2 + V_1) / A \quad (55)$$

$$\dot{x}_3 = x_4 \quad (56)$$

$$\dot{x}_4 = (-\mu_2 c_5 s_5 s_7 N_1 + (1 + \mu_1 c_5^2) N_2 + V_2) / A \quad (57)$$

$$\dot{x}_5 = x_6 \quad (58)$$

$$\dot{x}_6 = ((1 + \mu_2 s_7^2) s_5 N_1 - B c_5 s_7 N_2 + V_3) / (R A) \quad (59)$$

$$\dot{x}_7 = x_8 \quad (60)$$

$$\dot{x}_8 = (\mu_2 c_5 s_5 c_7 s_7 N_1 - (B - \mu_1 s_5^2) c_7 N_2 - V_4) / (R A s_5) \quad (61)$$

The expressions: A , R and $\sin x_5$ are greater than zero. Therefore the model is free from singularities.

8.4. Complete nonlinear model with varying pendulum length and three control forces

In order to derive the crane equations without the simplifications given in equations (14) - (17), formulas (9) - (11) and (13) are substituted into (4) - (8) which gives

$$\ddot{x}_c = -(u_3 - T_3) \sin \alpha \sin \beta \quad (62)$$

$$\ddot{y}_c = -(u_3 - T_3) \cos \alpha \quad (63)$$

$$\ddot{z}_c = (u_3 - T_3) \sin \alpha \cos \beta - g \quad (64)$$

$$\ddot{x}_w = u_2 - T_2 + \mu_2 (u_3 - T_3) \sin \alpha \sin \beta. \quad (65)$$

$$\ddot{y}_w = u_1 - T_1 + \mu_1 (u_3 - T_3) \cos \alpha \quad (66)$$

The equations (1) - (3) are differentiated twice with taking into account that the length of the lift-line $R(t)$ varies in time due to the action of the control force F_R . Proceeding as in section 6.2, the complete system of nonlinear state equations for the crane controlled by means of three forces is obtained

$$\begin{aligned}
\ddot{x}_c &= \ddot{x}_w + (\ddot{R} - R\dot{\alpha}^2 - R\dot{\beta}^2) \sin \alpha \sin \beta + 2R\dot{\alpha}\dot{\beta} \cos \alpha \cos \beta + \\
&+ (2\dot{R}\dot{\alpha} + R\ddot{\alpha}) \cos \alpha \sin \beta + (2\dot{R}\dot{\beta} + R\ddot{\beta}) \sin \alpha \cos \beta \\
\ddot{y}_c &= \ddot{y}_w + (\ddot{R} - R\dot{\alpha}^2) \cos \alpha - (2\dot{R}\dot{\alpha} + R\ddot{\alpha}) \sin \alpha \\
\ddot{z}_c &= (-\ddot{R} + R\dot{\alpha}^2 + R\dot{\beta}^2) \sin \alpha \cos \beta + 2R\dot{\alpha}\dot{\beta} \cos \alpha \sin \beta - \\
&+ (2\dot{R}\dot{\alpha} + R\ddot{\alpha}) \cos \alpha \cos \beta + (2\dot{R}\dot{\beta} + R\ddot{\beta}) \sin \alpha \sin \beta.
\end{aligned}$$

We denote

$$\begin{aligned}
x_1 &= y_w & x_6 &= \dot{x}_5 = \dot{\alpha} \\
x_2 &= \dot{x}_1 = \dot{y}_w & x_7 &= \beta \\
x_3 &= x_w & x_8 &= \dot{x}_7 = \dot{\beta} \\
x_4 &= \dot{x}_3 = \dot{x}_w & x_9 &= R \\
x_5 &= \alpha & x_{10} &= \dot{x}_9 = \dot{R} \\
s_n &\equiv \sin x_n \\
c_n &\equiv \cos x_n \\
V_5 &= c_5 s_5 x_8^2 x_9 - 2x_{10} x_6 + g c_5 c_7 \\
V_6 &= 2x_8 (c_5 x_6 x_9 + s_5 x_{10}) + g s_7 \\
V_7 &= s_5^2 x_8^2 x_9 + g s_5 c_7 + x_6^2 x_9.
\end{aligned}$$

Finally, we obtain ten equations describing the dynamics of the crane with varying pendulum length

$$\dot{x}_1 = x_2 \quad (67)$$

$$\dot{x}_2 = N_1 + \mu_1 c_5 N_3 \quad (68)$$

$$\dot{x}_3 = x_4 \quad (69)$$

$$\dot{x}_4 = N_2 + \mu_2 s_5 s_7 N_3 \quad (70)$$

$$\dot{x}_5 = x_6 \quad (71)$$

$$\dot{x}_6 = (s_5 N_1 - c_5 s_7 N_2 + (\mu_1 - \mu_2 s_7^2) c_5 s_5 N_3 + V_5) / x_9 \quad (72)$$

$$\dot{x}_7 = x_8 \quad (73)$$

$$\dot{x}_8 = -(c_7 N_2 + \mu_2 s_5 c_7 s_7 N_3 + V_6) / (s_5 x_9) \quad (74)$$

$$\dot{x}_9 = x_{10} \quad (75)$$

$$\dot{x}_{10} = -c_5 N_1 - s_5 s_7 N_2 - (1 + \mu_1 c_5^2 + \mu_2 s_5^2 s_7^2) N_3 + V_7. \quad (76)$$

The denominator in equation (74) includes $\sin x_5$. When the crane operates in its real range then $\sin x_5 \neq 0$.

9. Description of the Crane3D class properties

The *Crane3D* is a MATLAB class, which gives the access to all the features of the RT-DAC/PCI board equipped with the logic for the 3DCrane model. The RT-DAC/PCI board is an interface between the control software executed by a PC computer and the power-interface electronic of the 3DCrane model. The logic on the board contains the following blocks:

- incremental encoder registers – five 16-bit registers to measure the position of the incremental encoders. There are five identical encoders measuring five state quantities: two cart positions at the horizontal plane, the lift-line-length and two deviation angles of the payload;
- incremental encoder resets logic. The incremental encoders are able to generate different output waves when the encoder rotates clockwise and the counter clockwise. The encoders are not able to detect the reference (“zero”) position. To determine the “zero” position the incremental encoder registers can be set to zero from the computer program or an encoder register is reset when the corresponding limit switch to the encoder is reached;
- PWM generation block – generates three sets of signals. Each set contains the PWM output signal, the direction signal and the brake signal. The PWM prescaler determines the frequency of all the PWM waves. The PWM block logic can prevent the cart from motion outside the rail limits and the lift-line angles from lying outside the operating range. The operating ranges are detected twofold – by the limit switches and by three limit registers;
- power interface thermal flags – when the temperature of the power interface for the DC motors is too high the thermal flags can be used to disable the operation of the corresponding overheated DC motor.

All the parameters and measured variables from the RT-DAC/PCI board are accessible by appropriate methods of the *Crane3D* class.

The object of the *Crane3D* class is created by the command:

```
object_name = crane3d;
```

The *get* method is called to read a value of the property of the object:

```
property_value = get( object_name, 'property_name' );
```

The *set* method is called to set new value of the given property:

```
set( object_name, 'property_name', new_property_value );
```

The *display* method is applied to display the property values when the *object_name* is entered in the MATLAB command window.

This section describes all the properties of the *Crane3D* class. The description consists of the following fields:

Purpose	Provides short description of the property
Synopsis	Shows the format of the method calls
Description	Describes what the property does and the restrictions of is subjected to
Arguments	Describes arguments of the set method
See	Refers to other related properties
Examples	Provides examples how the property can be used

9.1. BaseAddress

Purpose: Read the base address of the RT-DAC/PCI board.

Synopsis: *BaseAddress* = *get(cr3, 'BaseAddress');*

Description: The base address of RT-DAC/PCI board is determined by OS. Each Crane3D object has to know the base address of the board. When a Crane3D object is created the base address is detected automatically. The detection procedure detects the base address of the first RT-DAC/PCI board plugged into the PCI slots.

Example: Create the Crane3D object:

```
cr3 = crane3d;
```

Display its properties by typing the command:

```
cr3
```

```
>>Type:      Crane3D Object
>>BaseAddress:  528
>>Bitstream ver.: x33
>>Encoder:      [ 65479  7661  20032  65533  65534 ][bit]
>>              [ -0.0022207[m] 0.29847[m] 0.38411[m] -0.004602[rad] -0.003068[rad] ]
>>Z displacement:  0.32[m]
>>PWM:          [ -0.062561  0.031281      -1 ]
>>PWMPrescaler:  60
>>RailLimit:     [ 361  381  815 ]*64[bit] <--> [23104  24384  52160 ][bit]
>>              [ 0.90013    0.95    1.0002 ][m]
>>RailLimitFlag: [ 1  1  1 ]
>>RailLimitSwitch: [ 0  1  1 ]
>>ResetSwitchFlag: [ 0  0  0 ]
>>Therm:         [ 1  1  1 ]
>>ThermFlag:     [ 1  1  1 ]
>>Time:          1.041 [sec]
```

Read the base address:

```
BA = get( cr3, 'BaseAddress' );
```

9.2. BitstreamVersion

Purpose: Read the version of the logic design for the RT-DAC/PCI board.

Synopsis: *Version* = *get(cr3, 'BitstreamVersion');*

Description: This property determines the version of the logic design of the RT-DAC/PCI board. The 3DCrane models may vary and the detection of the logic design version makes it possible to check if the logic design is compatible with the physical model.

9.3. Encoder

Purpose: Read the incremental encoder registers.

Synopsis: `enc = get(cr3, 'Encoder');`

Description: The property returns five digits. The first two measure the position of the cart. The third digit is used to measure the length of the lift-line and the last two measure the angles of the lift-line. The returned values can vary from 0 to 65535 (16-bit counters). When a register is reset the value is set to zero. When a rail limit flag is set it disables the movement outside the defined working range (rail limit). When a reset switch flag is set the encoder register is reset automatically when the appropriate switch is pressed. The incremental encoders generate 4096 or 2048 pulses per rotation. The values of the *Encoder* property should be converted into physical units.

See: *ResetEncoder, RailLimit, RailLimitFlag, ResetSwitchFlag*

9.4. PWM

Purpose: Set the parameters of the PWM waves.

Synopsis: `PWM = get(cr3, 'PWM');`
`set(cr3, 'PWM', NewPWM);`

Description: The property determines the duty cycle and direction of the PWM waves for three DC motors. The first two DC motors control the position of the cart and the last motor controls the length of the lift-line. The *PWM* and *NewPWM* variables are 1x3 vectors. Each element of these vectors determines the parameters of the PWM wave for one DC motor. The values of the elements of these vectors can vary from -1.0 to 1.0. The value -1.0 means the maximum control in one direction, the value 0.0 means zero control and the value 1.0 means the maximum control in the opposite direction.

The PWM wave is not generated if:

- a rail limit flag is set and the cart or lift-line are going to operate outside the working range,
- a therm flag is set and the power amplifier is overheated.

See: *RailLimit, RailLimitFlag, Therm, ThermFlag*

Example: `set(cr3, 'PWM', [-0.3 0.0 1.0]);`

9.5. PWMPrescaler

Purpose: Determine the frequency of the PWM waves.

Synopsis: `Prescaler = get(cr3, 'PWMPrescaler');`
`set(cr3, 'PWMPrescaler', NewPrescaler);`

Description: The prescaler value can vary from 0 to 63. The 0 value generates the maximum PWM frequency. The value 63 generates the minimum frequency.

See: *PWM*

9.6. ResetEncoder

Purpose: Reset the encoder counters.

Synopsis: *set(cr3, 'ResetEncoder', ResetFlags);*

Description: The property is used to reset the encoder registers. The *ResetFlags* is a 1x5 vector. Each element of this vector is responsible for one encoder register. If the element is equal to 1 the appropriate register is set to zero. If the element is equal to 0 the appropriate register remains unchanged.

See: *Encoder*

Example: To reset the first and fourth encoder registers execute the command:
set(cr3, 'ResetEncoder', [1 0 0 1 0]);

9.7. RailLimit

Purpose: Control the operating range of the 3D-crane system.

Synopsis: *Limit = get(cr3, 'RailLimit');*
set(cr3, 'RailLimit', NewLimit);

Description: The *Limit* and *NewLimit* variables are 1x3 vectors. The elements of these vectors define the operating range of the cart and the maximum length of the lift-line. If a flag defined by the *RailLimitFlag* property is set the corresponding to it PWM wave stops when the corresponding to it encoder register exceeds the limit.

See: *RailLimitFlag*

9.8. RailLimitFlag

Purpose: Set range of limit flags.

Synopsis: *LimitFlag = get(cr3, 'RailLimitFlag');*
set(cr3, 'RailLimitFlag', NewLimitFlag);

Description: The *RailLimitFlags* is a 1x3 vector. The first two elements control the operating range of the cart. The last element controls the maximum length of the lift-line. If the flag is set to 1 and the encoder register exceeds the range the DC motor corresponding to it stops. If the flag is set to 0 the motion continues in spite of the range limit exceeded in the encoder register.

See: *RailLimit, RailLimitSwitch*

9.9. RailLimitSwitch

Purpose: Read the state of limit switches.

Synopsis: *LimitSwitch = get(cr3, 'RailLimitSwitch');*

Description: Reads the state of three limit switches. Returns a 1x3 vector. If an element of this vector is equal to 0 it means that the switch has been pressed.

See: *RailLimit, RailLimitFlag*

9.10. ResetSwitchFlag

Purpose: Control the auto-reset of the encoder registers.

Synopsis: *ResetSwitchFlag = get(cr3, 'ResetSwitchFlag');*
set(cr3, 'ResetSwitchFlag', ResetSwitchFlag);

Description: The *ResetSwitchFlag* and *NewResetSwitchFlag* are 1x3 vectors. If an element of these vectors is equal to 1 the corresponding to it encoder register is automatically reset in the case when the corresponding to it limit switch is pressed.

See: *ResetEncoder, RailLimitSwitch*

9.11. Therm

Purpose: Read thermal flags of the power amplifiers.

Synopsis: *Therm = get(cr3, 'Therm');*

Description: Returns three thermal flags of three power amplifiers. When the temperature of a power amplifier is too high the appropriate flag is set to x.

See: *ThermFlag*

9.12. ThermFlag

Purpose: Control an automatic power down of the power amplifiers.

Synopsis: *ThermFlag = get(cr3, 'ThermFlag');*
set(cr3, 'ThermFlag', NewThermFlag);

Description: The *ThermFlag* and *NewThermFlag* are 1x3 vectors. If an element of these vectors is equal to 1 the DC motor corresponding to it is not excited by the PWM wave when it is overheated.

See: *Therm*

9.13. Time

Purpose: Return time information.

Synopsis: $T = \text{get}(cr3, 'Time');$

Description: The *Crane3D* object contains the time counter. When a *Crane3D* object is created the time counter is set to zero. Each reference to the *Time* property updates its value. The value is equal to the number of milliseconds past since the object was created.

9.14. Quick reference table

Property Name	Description
<i>BitstreamVersion</i>	Read the version of the logic design for the RT-DAC/PCI board
<i>Encoder</i>	Read the incremental encoder registers
<i>PWM</i>	Set the parameters of the PWM waves
<i>PWMPrescaler</i>	Determine the frequency of the PWM waves
<i>ResetEncoder</i>	Reset the encoder counters
<i>RailLimit</i>	Control the operating range of the 3DCrane system
<i>RailLimitFlag</i>	Set the range limit flags
<i>RailLimitSwitch</i>	Read the state of the limit switches
<i>ResetSwitchFlag</i>	Control the auto-reset of the encoder registers
<i>Therm</i>	Read the thermal flags of the power amplifiers
<i>ThermFlag</i>	Control the automatic power down of the power amplifiers
<i>Time</i>	Return time information