Reaction Wheel Pendulum

User's Manual

www.inteco.com.pl

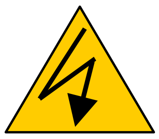
NOTES

SAFETY OF THE EQUIPMENT

The equipment, when used in accordance with the supplied instructions, within the parameter set for its mechanical and electrical performance, should not cause any danger to health or safety if normal engineering applications are observed.

If, in specific cases, circumstances exist in which a potential hazard may be brought about by careless or improper use, these will be pointed out and the necessary precautions emphasised.

Some National Directives require to indicate on our equipment certain warnings that require attention by the user. These have been indicated in the specified way by labels. The meaning of any labels that may be fixed to the equipment instrument are explained in this manual.


Risk of electric shock

PRODUCT IMPROVEMENTS

The Producer reserves a right to improve design and performance of the product without prior notice.

All major changes are incorporated into up-dated editions of manuals and this manual is believed to be correct at the time of printing. However, some product changes which do not affect the capability of the equipment, may not be included until it is necessary to incorporate other significant changes.

ELECTROMAGNETIC COMPATIBILITY

This equipment, when operated in accordance with the supplied documentation, does not cause electromagnetic disturbance outside its immediate electromagnetic environment.

COPYRIGHT NOTICE

ACKNOWLEDGEMENTS

CONTENTS

1. INTRODUCTION AND GENERAL DESCRIPTION

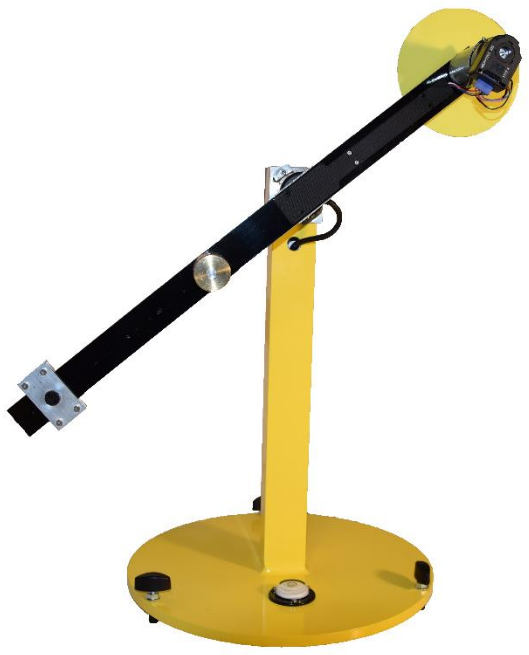


Fig. 1.1 Reaction Wheel Pendulum

Reaction Wheel Pendulum (**RWP**) shown in Fig. 1.1 is an inverted pendulum type device designed for control education and research. From a control theory point of view RWP is an unstable, nonlinear system to some extent similar to the Inverted Pendulum. To swing up and stabilise the system in an upright position DC motor drives a reaction wheel (a type of flywheel) forward and backward. The appropriate direction and torque produced by the DC motor is calculated by a feedback control algorithm. The control algorithm is designed, adjusted and run on single PC with MATLAB/Simulink environment using Model Based Design technique.

The RWP is equipped with power interface and multipurpose analog and digital I/O board.

The system serves for testing and verifying in practice linear and nonlinear control algorithms. The control algorithms are designed and prepared in the MATLAB/Simulink environment and run as Real-Time procedure on the same PC.

This manual describes:

- the system,
- the mathematical models and theory related to control experiments,
- the real-time experiments,

- ♦ how to use the library of ready-to-use real-time controllers,
- ♦ how to design and apply step-by-step an own controller in the MATLAB/Simulink environment.

It is assumed that a user has got an experience with MATLAB and Simulink from MathWorks Inc.

1.1 SYSTEM COMPONENTS

To use the **RWP** system the following software and hardware components are required:

Hardware

- **RWP** - the hardware unit shown in Fig. 1.1,
- a power interface,
- RTDAC/USB - an input/output board.

Software

- Microsoft Windows W7/W10x64 and MATLAB 64 bit with Simulink, and RTW (Simulink Coder) toolboxes (not included),
- Third party compiler MS Visual C++ depending on Matlab's version
- Details at: https://www.mathworks.com/support/sysreq/previous_releases.html

and

- CD-ROM or USB stick including the **RWP** software and the e-manuals.

⇨	**Details of the required software are available at:** **http://www.inteco.com.pl/support/Software_requirements.pdf**

Manuals:

- *Installation Manual*
- *User's Manual*

⇨	**The experiments and corresponding measurements have been conducted by the use of the standard INTECO systems. Every new system manufactured and developed by INTECO can be slightly different to those standard devices. It explains why a user can obtain results that are not identical to those in the manual.**

1.2 SYSTEM OVERVIEW

The schematic diagram of the **RWP** system is shown in Fig. 1.2. The device is equipped with PWM controlled DC motor which drives a flywheel. Any change in rotational speed of reaction wheel influences rotational speed of the pendulum rod accordingly to conservation of angular momentum. The center of mass of the rod can be adjusted with counterweight thus affecting dynamics of the RWP. When center of the mass is set above pivot point RWP becomes inverted pendulum - an unstable system which requires active control to maintain upright position. In case of center of the mass under pivot point RWP becomes stable, oscillating system. Third option is setting center of the mass in the axis of rotation making the RWP fully balanced and stable in every point. The angle of the rod and flywheel are measured with incremental encoders. To transfer power and data signals from flywheel encoder slip ring is used. The advantage of this structure is limitless movement with no hazard in destroying wiring.

RWP consists of the following elements:

- The mechanical unit,
- The PWM controlled DC motor,
- Two incremental encoders, one mounted at the DC motor axis to provide flywheel angle information (1024 imp/rpm quadrature), second mounted at pendulum axis (5000 imp/rpm quadrature) to provide information regarding pendulum angle,
- Two counterweights, one fixed and one adjustable to change pendulum characteristic,
- The power interface box containing a power supply, power amplifiers and signals conditioning unit.

,

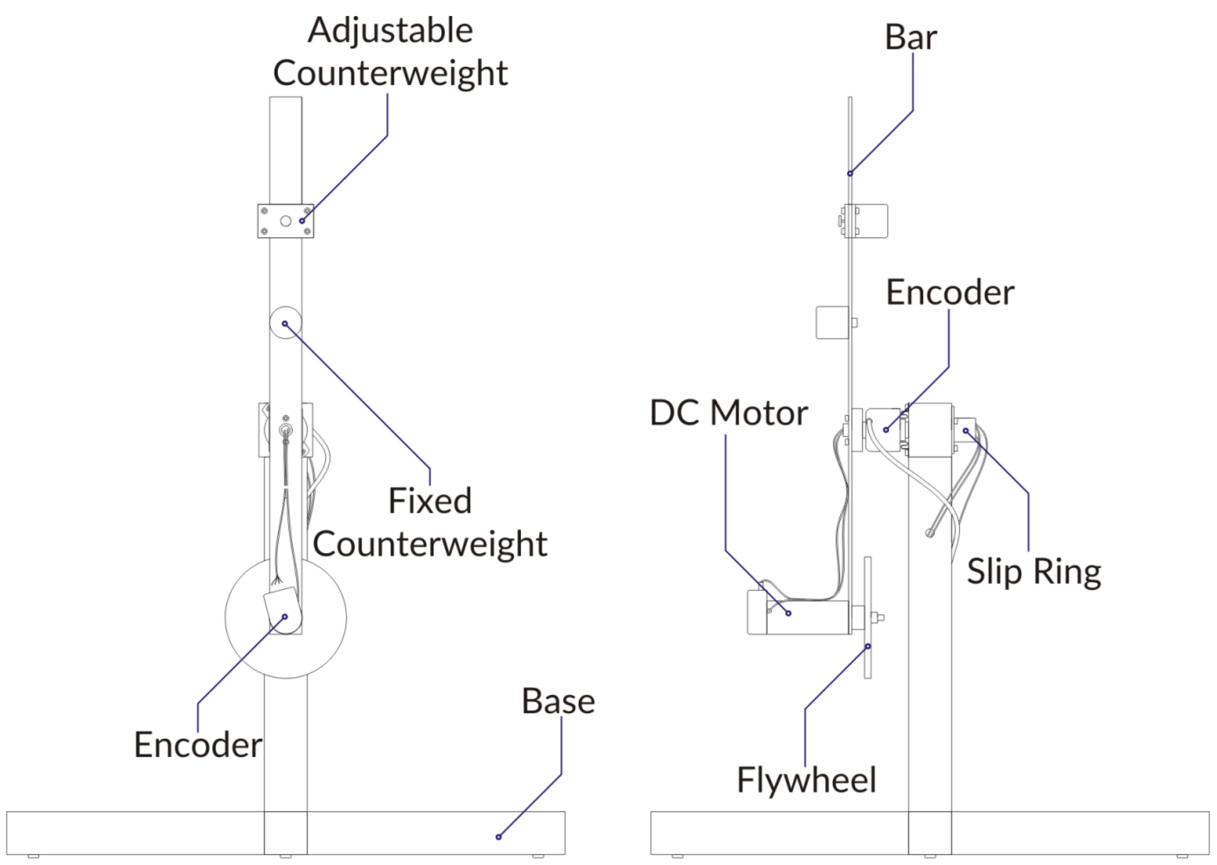


Fig. 1.2 RWP system block diagram

1.3 COUNTERWEIGHT MODIFICATION

To change reaction pendulum dynamical properties counterweight can be adjusted. Position of the counterweight can be chosen freely.

To change position perform three steps, please refer to figure below (no tool needed):

1. Untighten hand screw to loose counterweight.
2. Choose new position of the counterweight.
3. Fix counterweight in new position by tightening the hand screw.

Chapter 4.3 describes procedures to identify physical parameters of the pendulum after counterweight adjustment.

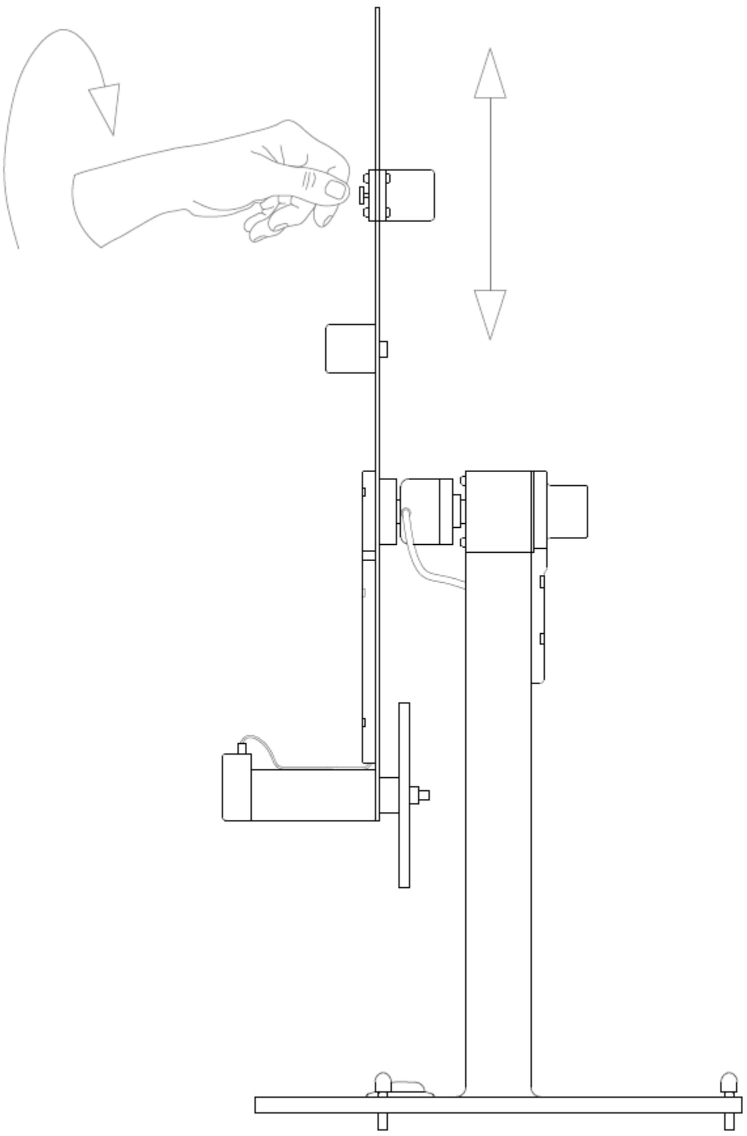


Figure 1 The counterweight position adjustement

1.4 SOFTWARE INSTALLATION

1.4.1. CD

Insert the installation CD and follow the displayed commands.

1.4.2. USB STICK

Plug USB stick and run *manager.exe* application located in the main catalogue.

2. STARTING AND TESTING PROCEDURES

It is assumed that RWP toolbox installation was successful.

Invoke MATLAB by double clicking on the MATLAB icon. The MATLAB command window opens.

If the MATLAB R2018 or newer is used run command *rehash toolbox*, close Matlab and open again.

Then type:

```
rpendulum
```

MATLAB brings up the RWP *Control Window* (see Fig. 2.1). The user has an instant access to all basic functions of the device control and simulation systems from the RWP *Control Window*.

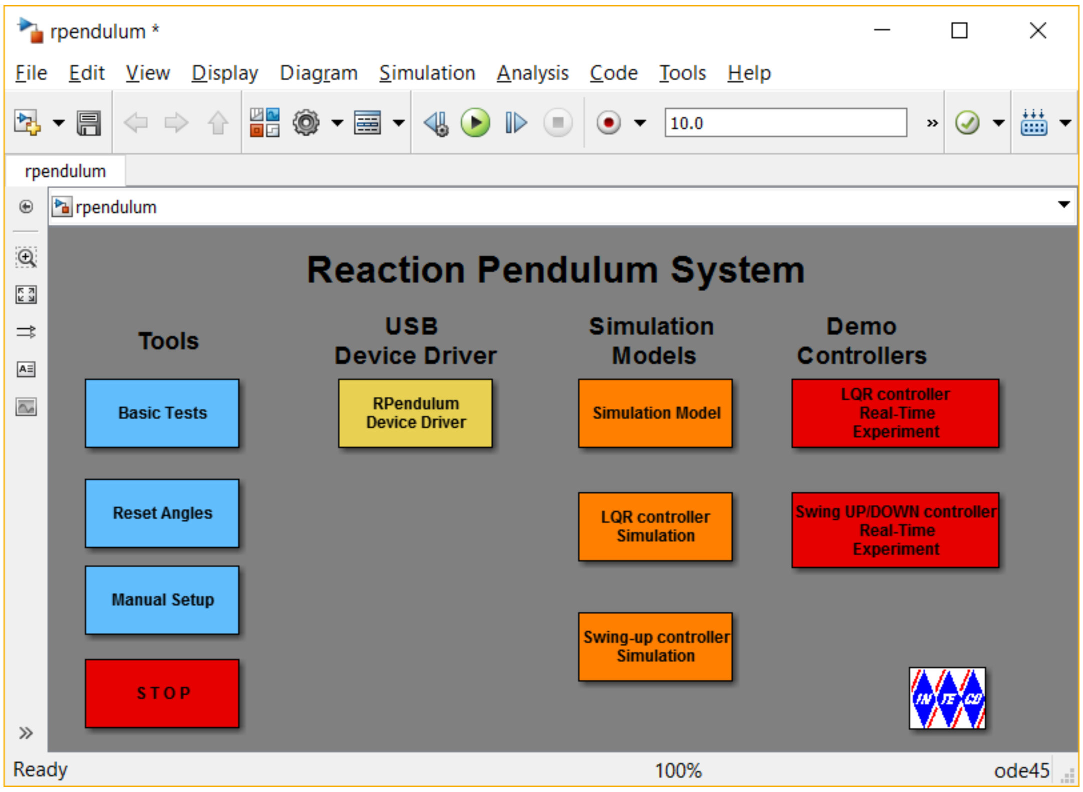


Fig. 2.1 Control Window of the *RWP* system

The *RWP Control Window* contains testing tools, drivers, models and demo applications. See section 3 for the detailed description.

3. RWP CONTROL WINDOW

3.1 BASIC TESTS

This section explains how to perform the basic tests. The tests have to be performed obligatorily before start of the real-time experiments. They are also necessary if any incorrect operation of the system was detected. The tests have been designed to validate the existence and sequence of measurements and controls.

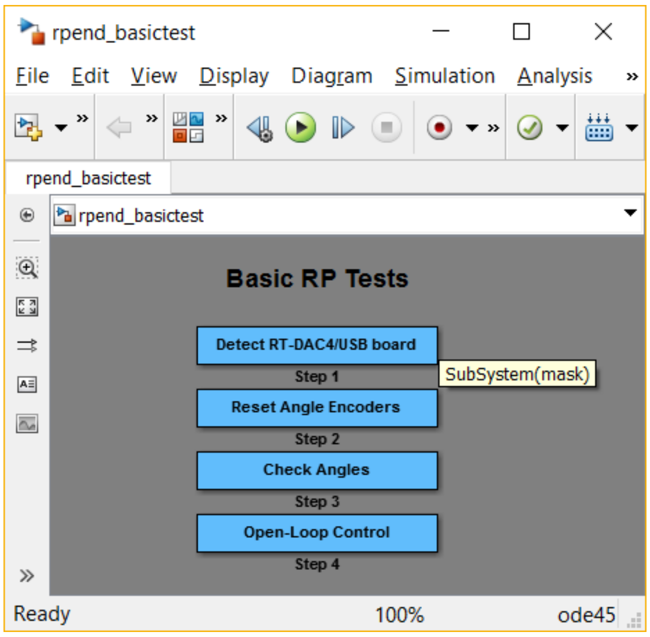


Fig. 3.1 *Basic Tests* window

1. *Detect RT-DAC board* button checks if control and measurement board is detected and visible in MATLAB. **Further tests cannot be performed if card detection fails.**
2. *Reset Encoders* button

   current values of the angular positions are set to zero.
3. *Check Angles* button

   allow validation of both signals from encoders.
4. *Open-Loop Control* button

   allow validation of control signal.

3.2 DEVICE DRIVER

The driver integrates the MATLAB/Simulink environment and RWP transforming and transmitting the measurement and control signals from/to the RWP system. To build a new application it is recommended to save a copy of the model with device driver (to preserve proper configuration). After clicking the Rpendulum *Device Driver* button the window shown in Fig. 3.2 opens.

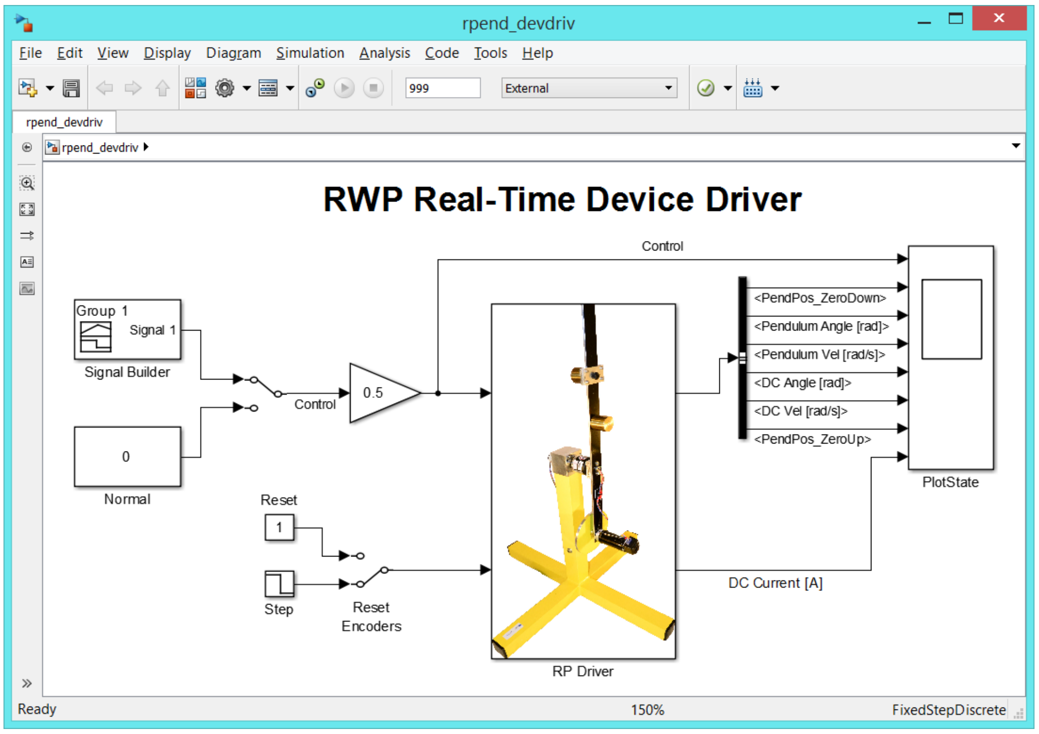


Fig. 3.2 *RWP Device Driver*

The driver has one PWM input for controlling the DC motor of the RWP. The control signal is scaled to [-1;1] corresponding to [-100%; 100%] of pulse width (sign can be interpreted as direction). The second input resets the incremental encoders which measure the angles of the pendulum and flywheel. The outputs are organised in *Bus* to allow selection of any signal by its name. Signals in the *Bus* are as follows: the angular position of the flywheel φ, the angular velocity of the flywheel $\dot{\varphi}$, the angular position of the pendulum θ, the angular velocity of the pendulum $\dot{\theta}$.

⇨ **Do not introduce any changes inside the original driver. They should be done only inside its copy.**

3.3 SIMULATION MODELS

Simulation model is introduced to familiarize a user with RWP system operation and for templates for developing and testing the user-defined control algorithms. In Fig. 3.3 the block diagram of nonlinear model of RWP system is shown. Three simulations are available to illustrate different scenarios. One is simulation model in open loop control. This model can be used to familiarize user with RWP system behaviour and as a starting point for user own applications. Second and third simulations shows closed loop control in, respectively, stabilisation in upright position and swing-up control. Each simulation utilises same simulation model presented in Fig. 3.3. This model is Simulink representation of mathematical model from chapter 4.

The model has PWM input signal for the DC motors. The output signals represents state of the system (state variables $x = [\varphi, \dot{\varphi}, \theta, \dot{\theta}]^T$). Detailed explanation can be found in the in chapter 4.

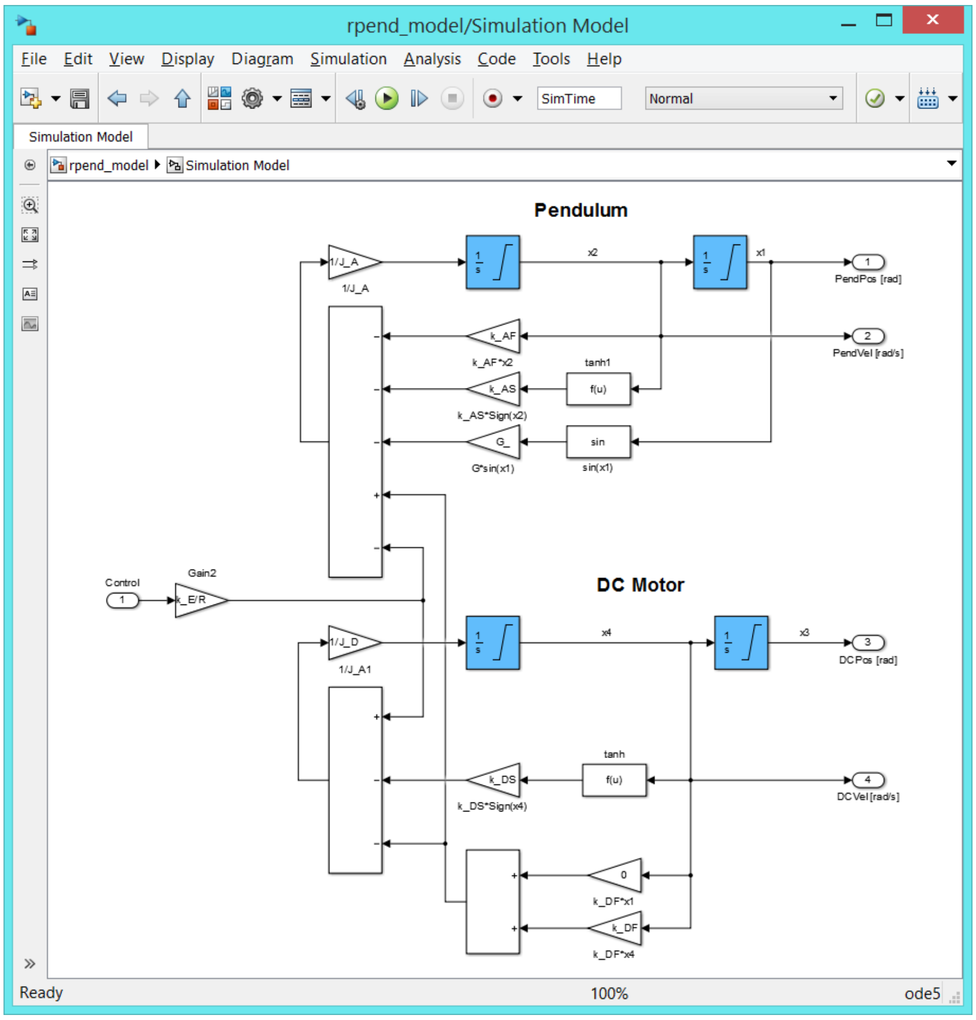


Fig. 3.3 Nonlinear simulation model

⇨	**Remember the RWP system in the inverted pendulum mode is structurally unstable and must run with a controller to get a stable behaviour.**
⇨	**Choose *Fixed step* solver options and set *Fixed-step size* equal to 0.01.**

3.4 DEMO CONTROLLERS

Demo controllers allow performing real-time experiments.

Demo controller accomplishes the task of swing-up and stabilization of the RWP system in inverted pendulum mode as well as stabilization in stable point.

The model, presented in Fig. 3.4, contains the device driver, appropriate controllers (swing-up and LQR), logic to decide which control strategy use, scope to present data and switch to allow manual control.

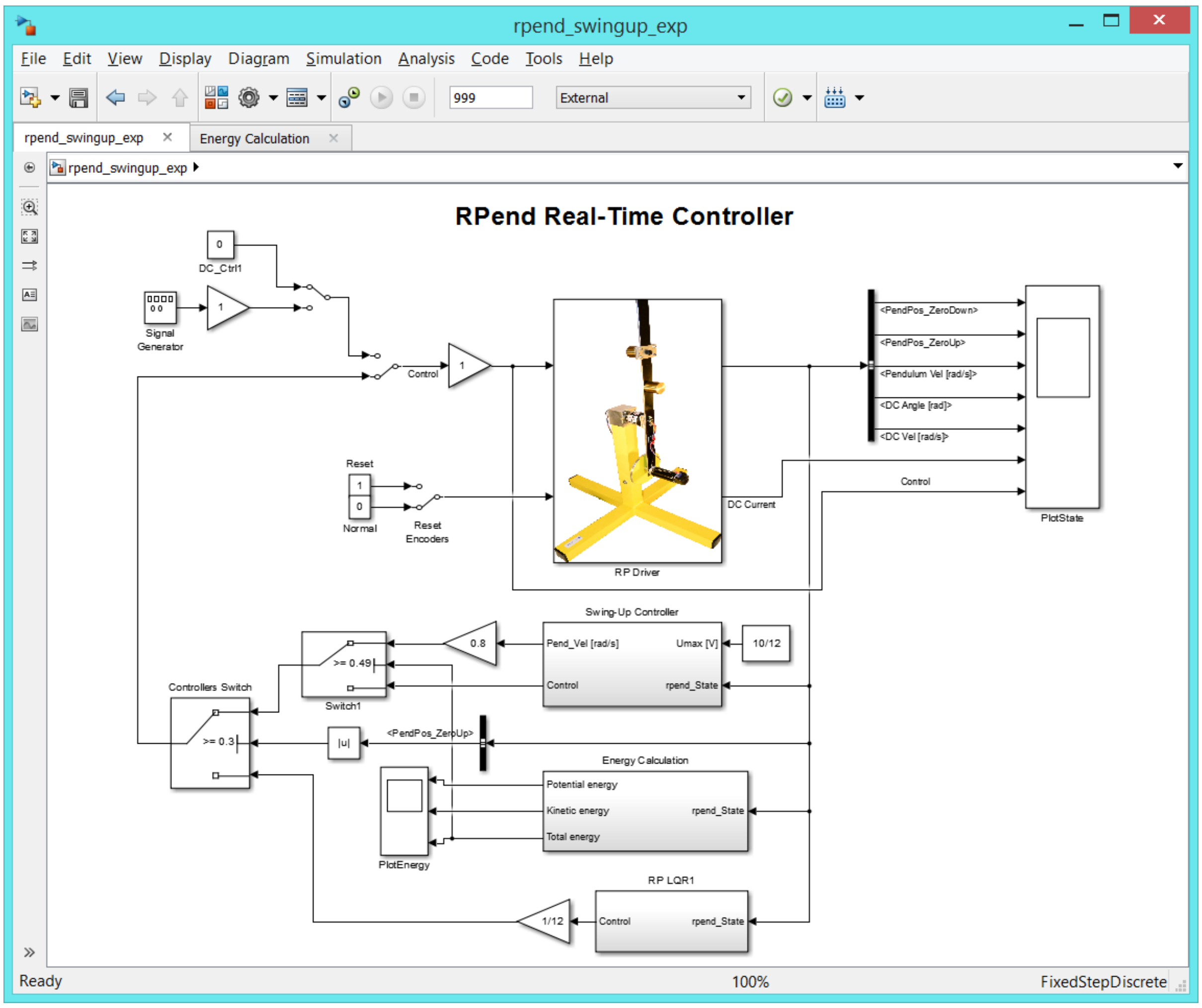


Fig. 3.4 Real-time demo model

The interior of each block is as follows:

- **LQ controller –** state feedback controller to stabilise pendulum in upright position.

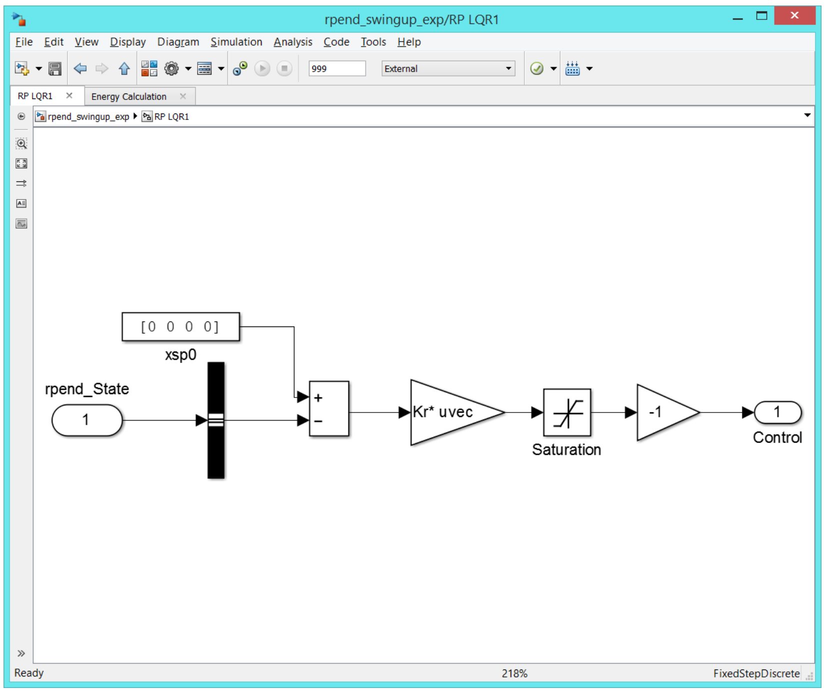


Fig. 3.5 LQ controller

- **Swing-up controller**– bang-bang controller to swing up pendulum to upright position.

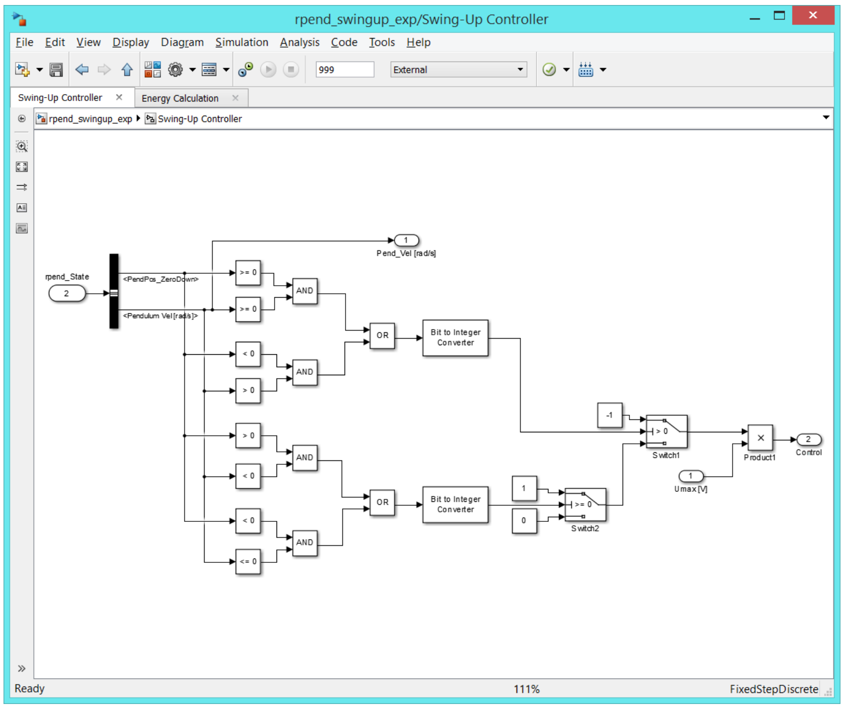


Fig. 3.6 Swing-up controller

- **Energy calculation-** performs calculation of mechanical energy of the system to decide about swing up control.

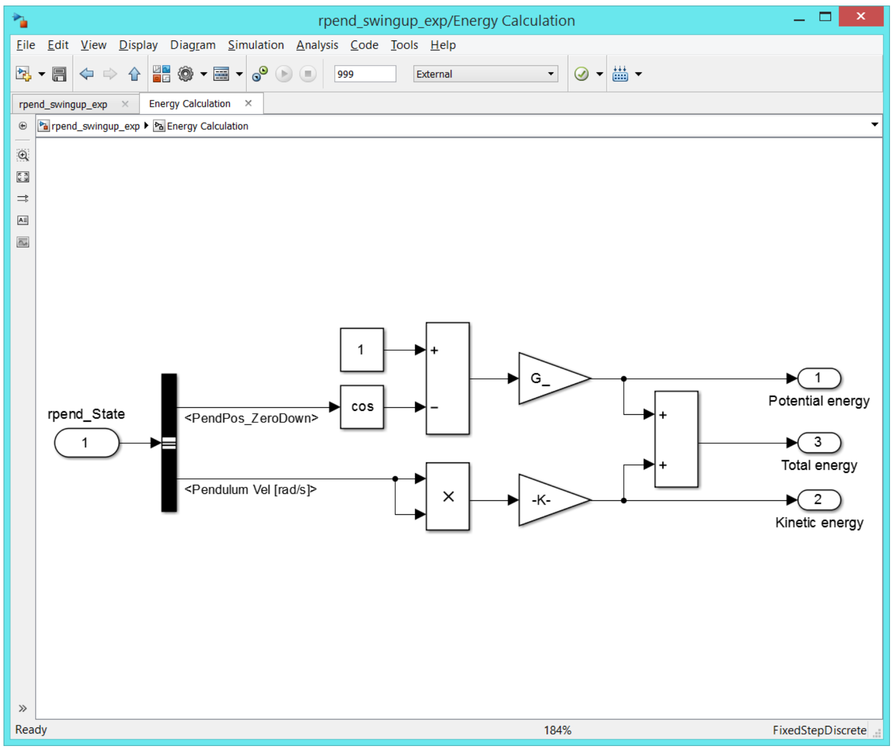


Fig. 3.7 Energy calculation

4. MATHEMATICAL MODEL OF THE RWP

Available after purchasing the system.

4.1 NONLINEAR MODEL

Available after purchasing the system.

4.2 LINEARIZED MODEL

Available after purchasing the system.

4.3 PENDULUM'S PARAMETERS IDENTIFICATION

Each time counterweight is adjusted to different position pendulum's moment of inertia and pendulum's center of mass changes. This section describes method for identification of those parameters based on simple experiments.

4.3.1. CENTER OF MASS IDENTIFICATION

This section describes method for identification of pendulum's center of mass based on simple experiment. The idea behind the experiment is to measure displacement of adjustable counterweight from position in which pendulum is balanced to desired position of counterweight.

When pendulum is full balanced its center of mass is identical with axis of rotation. This can be described as

$$0 = \frac{m_{ex} r_{ex} + m_{Cadj} r_{Cadj_{balanced}}}{m_{ex} + m_{Cadj}}$$

where

- m_{ex} is mass of whole pendulum except adjustable counterweight, which can be expressed as $m_{ex} = m + m_{DC} + \ m_{Cfixed} + m_B$
- r_{ex} is center of mass of whole pendulum except adjustable counterweight,
- $r_{Cadj_{balanced}}$ is center of mass of adjustable counterweight when pendulum is balanced.

After adjustment of counterweight new center of mass *L* can be expressed by equation

$$L = \frac{m_{ex} r_{ex} + m_{Cadj} r_{Cadj}}{m_{ex} + m_{Cadj}}$$

by subtracting both equations wanted parameter *R* can be found as all masses are known.

$$L = \frac{m_{Cadj}(r_{Cadj_{balanced}} - r_{Cadj})}{m_{ex} + m_{Cadj}}$$

4.3.2. Moment of Inertia Identification

This section describes method for identification of pendulum's moment of inertia based on simple experiment. The idea behind the experiment is to measure oscillation period of pendulum and knowing that compound pendulum period depends on its moment of inertia obtain wanted parameter.

The experiment can be performed in MATLAB/Simulink environment with Real-Time model (with control signal switch off) to log the data or with a stopwatch. Move the rod by small angle and measure the oscillation period (as seen in below figure).

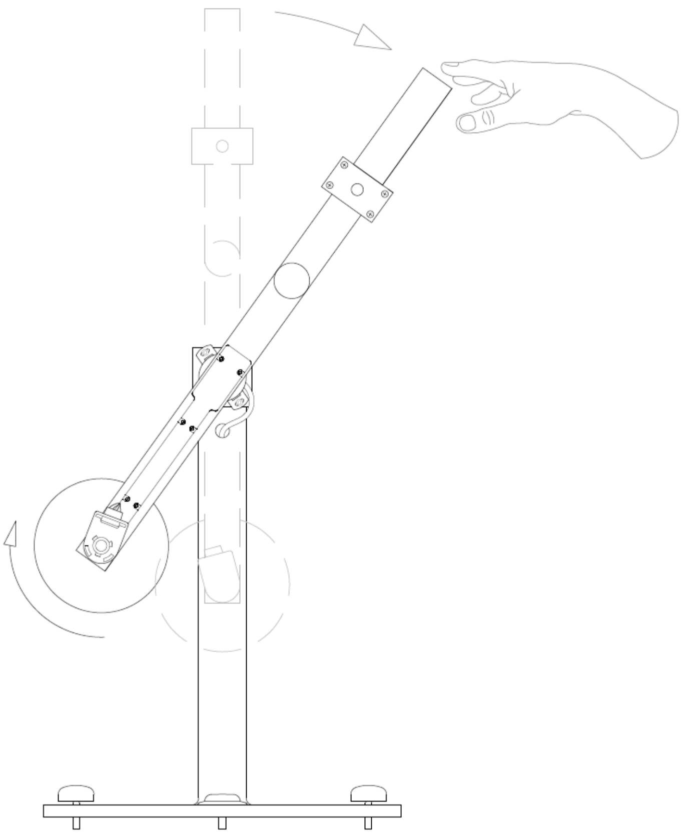


Figure 2 Swing the pendulum by small angle.

Knowing that

$$T = 2\pi\sqrt{\frac{J_\theta}{mgL}}$$

moment of inertia can be obtained

$$J_\theta = \frac{T^2 mgL}{4\pi^2}$$

Following script will perform all the necessary steps for the data provided in second line.

```
clear all;
load RWP_data

time        = StateData.time;
ctrl        = StateData.signals(1).values;
pendPosZD   = StateData.signals(2).values;
pendPosZU   = StateData.signals(3).values;
pendVel     = StateData.signals(4).values;
diskPos     = StateData.signals(5).values;
diskVel     = StateData.signals(6).values;

% Check the system response
plot(time, pendPosZD)
hold on, grid on
% Find period of damped oscillation, e.g. using findpeaks MATLAB function
% and averaging obtained results
[pks,locs] = findpeaks(pendPosZD);
plot(time(locs),pks,'-*')
[pks_,locs_] = findpeaks(-pendPosZD);
plot(time(locs_),-pks_,'-*')
period=mean([diff(time(locs))]);
if(exist('fit','file'))
    f=fit(time(locs),pks,'exp1');
end
% Calculate Moment of Inertia (in respect of rotation axis) using formula
% for physical pendulum period
m = 0.3; d = 0.1;
MomentOfInertia = (period / (2*pi))^2 * m * 9.81 * d;
% Using Stainer's theorem recalculate Moment of Inertia in respect of
% pendulum centroid
```

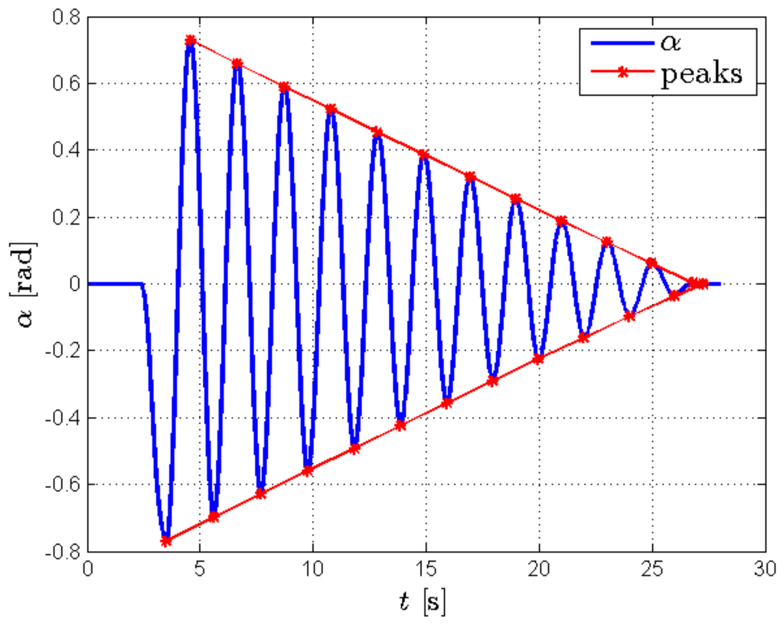


Fig. 4.1 Script result illustration

5. REAL-TIME CONTROL EXPERIMENTS DESIGN AND IMPLEMENTATION

In the following section a control experiment is described. The experiment consists of three phases: the design phase, simulation phase and real-time implementation phase.

5.1 DESIGN OF LINEAR CONTROLLER

The RWP system demo presented in this section deals with the control task: *swing-up the pendulum and stabilize it in the upper position* (an unstable steady-state point).

The design of the continuous LQ controller is shown below. For a very small value of the sampling time the response of the discrete system converges to the response of the corresponding continuous system. It is just our case.

The linearized dynamical model of the unstable transporter is described by the linear differential equations (**Błąd! Nie można odnaleźć źródła odwołania.**) at the unstable equilibrium point defined as: $x_0 = (0{,}0{,}0{,}0)^T, u_0 = 0$:

$$J(u) = \frac{1}{2}\int_0^\infty [x^T(\tau)\mathrm{Q}\mathrm{x}(\tau) + u^T(\tau)\mathrm{R}\mathrm{u}(\tau)]d\tau,$$

where :Q is a nonnegative definite matrix $Q \geq 0, Q = Q^T$,
R is a positive definite matrix $R > 0, R = R^T$,
and the pair (A, B) is controllable.

The weighting matrices Q and R are selected by a designer but they must satisfy the above conditions. This is most easily accomplished by choosing Q to be diagonal with all diagonal elements positive or zero.

The LQ optimal control u is given then by:

$$u = -Kx$$

the optimal state feedback matrix.

The optimal control problem is now defined as follows: find the gain K such that the feedback law (5.2) minimizes the cost function (6.2) subject to the state equation (6.1). The optimal feedback gain can be obtained by iterative solution of the associated matrix Riccati equation:

$$\mathrm{SA} + A^T S - \mathrm{SBR}^{-1} B^T S + Q = 0$$

To solve the LQ controller problem the *lqr* function can be used from the MATLAB Control System Toolbox. The synopsis of *lqr* is: *[K,S,E] = lqr(A,B,C,D,Q,R)*.

In our case the state equation matrices are as follows:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 3.47 & 0.13 & 0 & 0.052 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 6.23 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 2.44 \\ 0 \\ 291.8 \end{bmatrix}$$

matrix C is identity and D is zero matrix.

The weighting matrices *Q* and *R* have the following forms

$$Q = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad R = [1000],$$

After command $[\mathrm{K,S,E}] = \mathrm{lqr(A,B,C,D,Q,R)}$ the optimal gain matrix *K* for the considered parameters is calculated as

$$K = [0.0158 \quad 0.0514 \quad 4.1139 \quad 0.5304],$$

Recall now our task control:

***Swing- up the pendulum and stabilize it in the upper position* (an unstable steady-state point).**

The LQ control simulation experiments are performed for the following data:

- Setpoint: $x_r = (\pi, 0,0,0)$
- and starting point: $x_0 = (0,0,0,0)^T$,

5.2 SIMULATION

Using the second simulation model – *LQR Controller Simulation* (shown in Fig. 5.1) the simulation experiment with the LQ controller is performed.

Before running the model initialize parameters with *rpend_model_parameters.m* script.

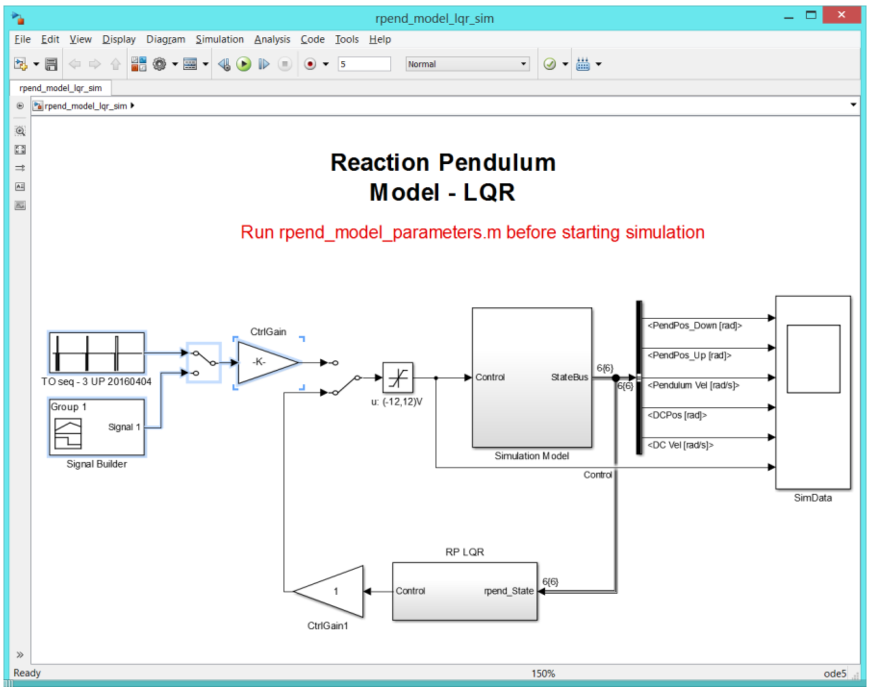


Fig. 5.1 LQ control – the simulation model

The initial state of the pendulum is equal to zero. The results of the simulation are shown in Fig. 5.2., Fig. 5.2. Fig. 5.2.

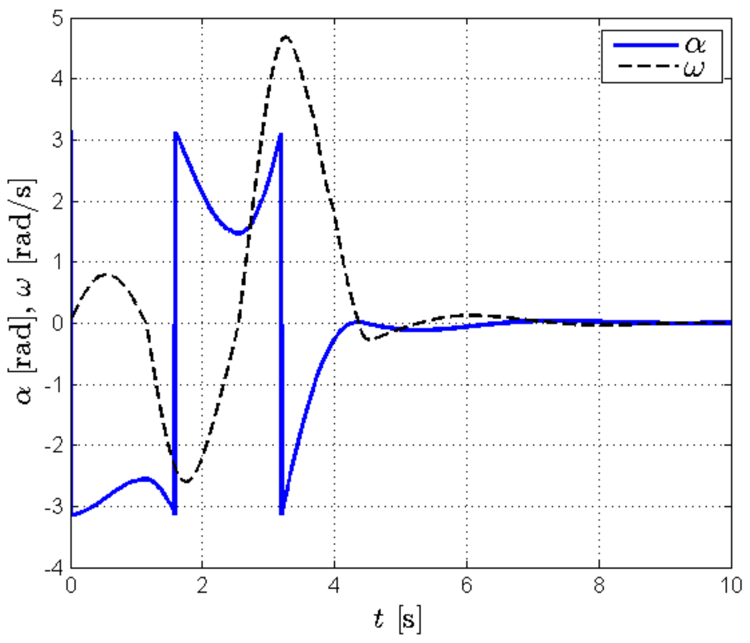


Fig. 5.2 LQ control: the nonlinear simulation model. Angle and angular velocity of the pendulum

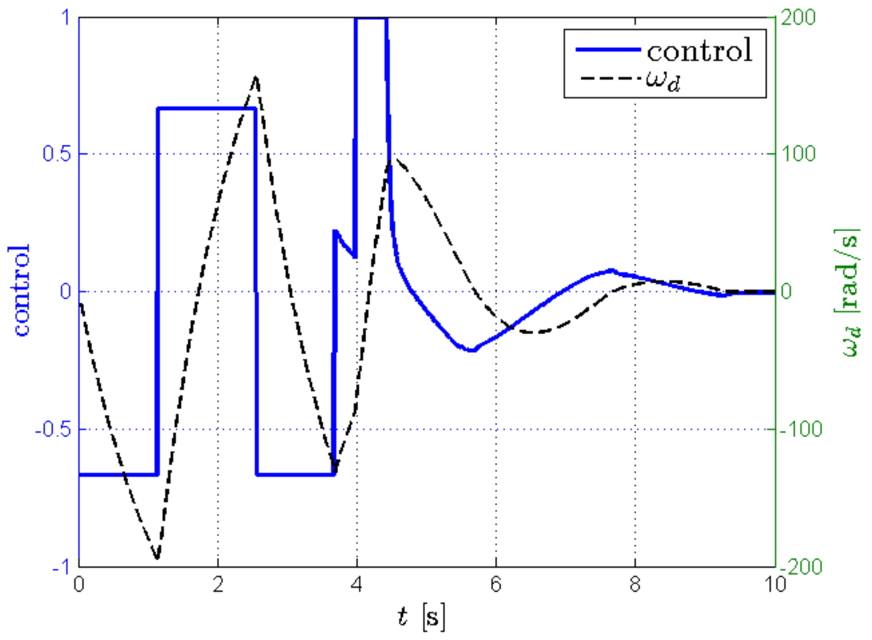


Fig. 5.3 LQ control: the nonlinear simulation model. Control signal and flywheel velocity.

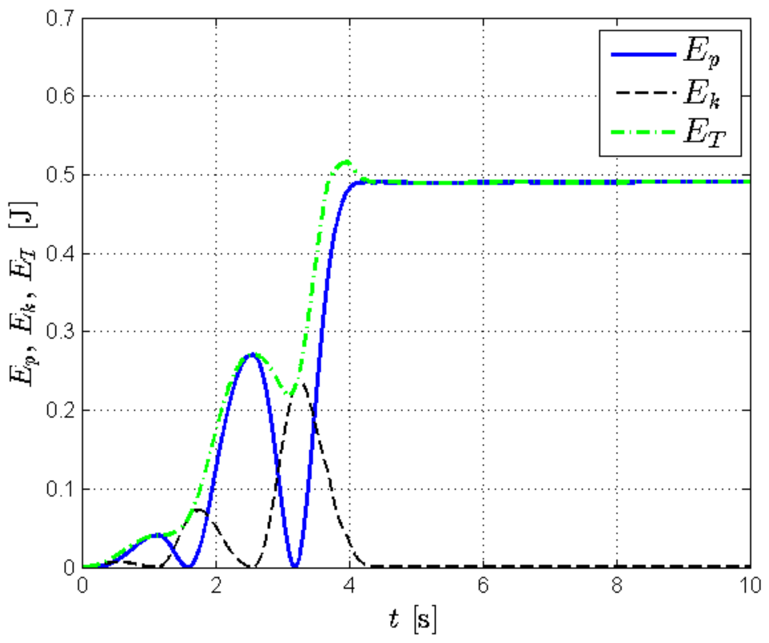


Fig. 5.4 LQ control: the nonlinear simulation model. Potential, Kinetic and Total Energy of the System

5.3 REAL-TIME EXPERIMENTS- IMPLEMENTATION

In this section the preprogramed examples of the RWP control system are illustrated. This demo is used to familiarize a user with the RWP system operation and help to create user-defined control algorithms.

5.3.1. EXAMPLE 1 – SWING UP AND STABILISATION IN UPPER POSITION

After clicking on the *Swing UP/DOWN controller Real-Time Experiment* button the model of the real-time controlled RWP system appears (Fig. 5.5).

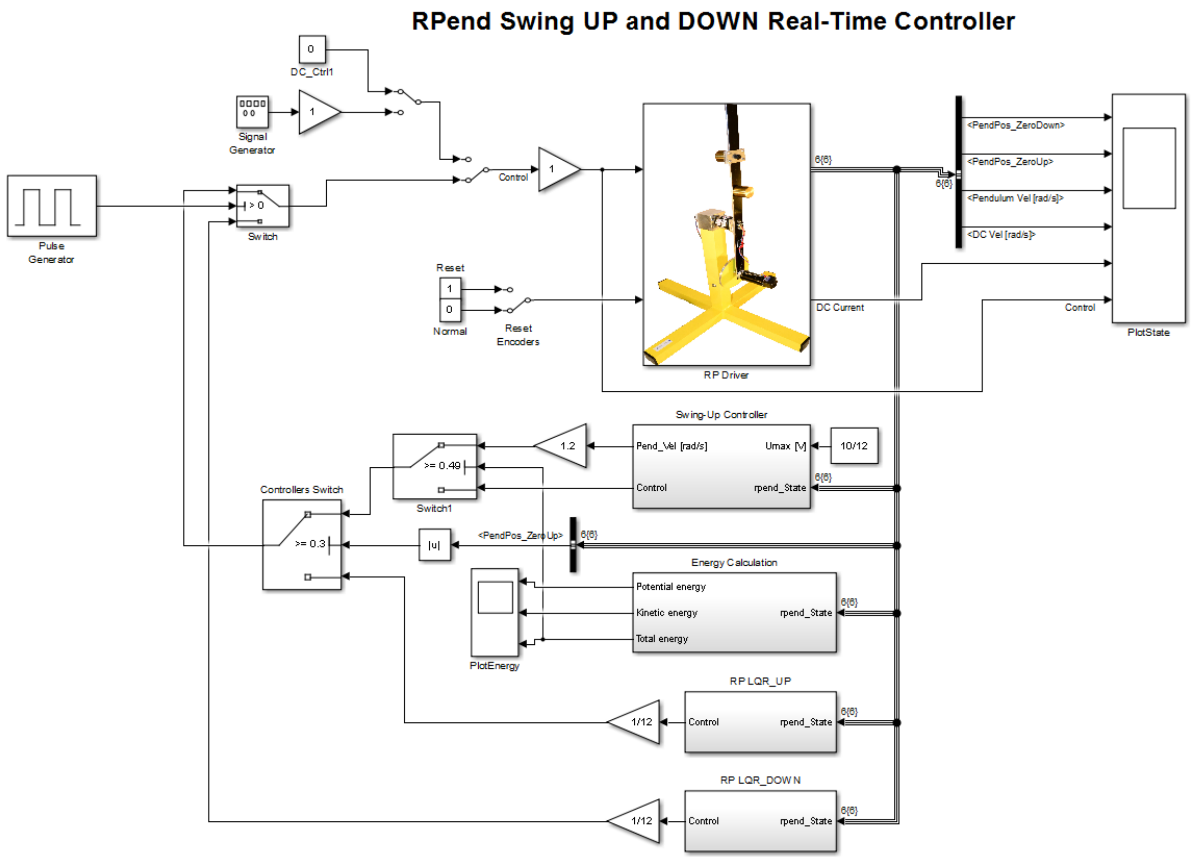


Fig. 5.5 Control system with the LQ controller

Note that this is a typical Simulink model. The device driver shown in Fig. 3.2 is applied in the same way as other blocks from the Simulink library.

To start experiment click the *Connect To Target* button. Next click the *OK* button to activate real-time application you PC computer.

Results of the experiment are shown in Fig. 5.6, Fig. 5.7. and Fig. 5.7.In the experiment pendulum is swin up and stabilized (blue plot). Dashed plot shows pendulum velocity.

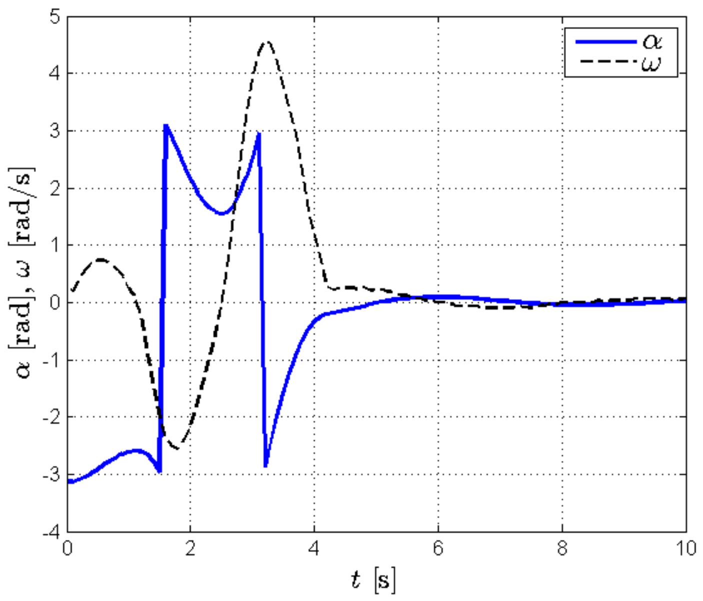


Fig. 5.6 State variables: α-blue and ω- black.

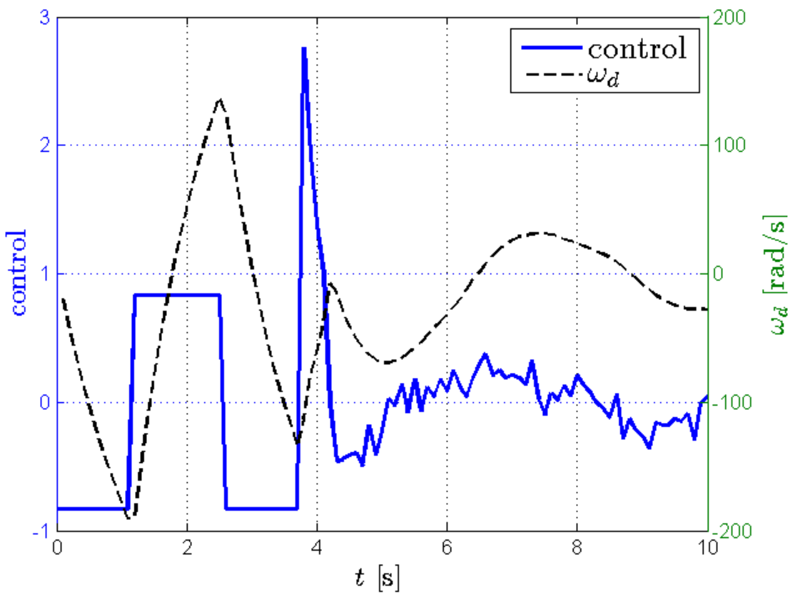


Fig. 5.7 Control signal - blue and disk velocity - black

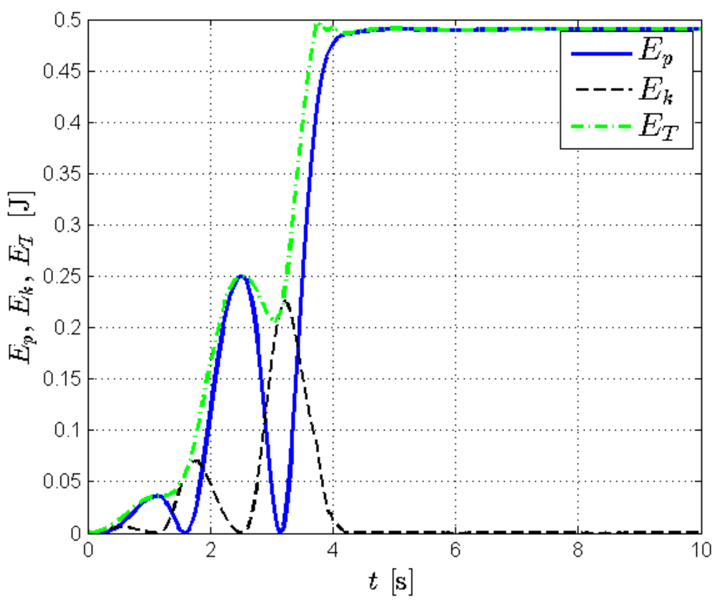


Fig. 5.8 Potential, Kinetic and Total Energy of the System

5.3.2. EXAMPLE 2 –PENDULUM DAMPING FROM UNSTABLE EQUILIBRIUM

Same Simulink Model as in experiment 1 may be used to obtain following result. Demo Controller is set to cyclically switch between swinging up and stabilise in pendulum up position to damping and stabilise in pendulum down postion.

Results of the experiment are shown in Fig. 5.6, Fig. 5.7. and Fig. 5.7.In the experiment pendulum is damped and stabilized (blue plot). Dashed plot shows pendulum velocity.

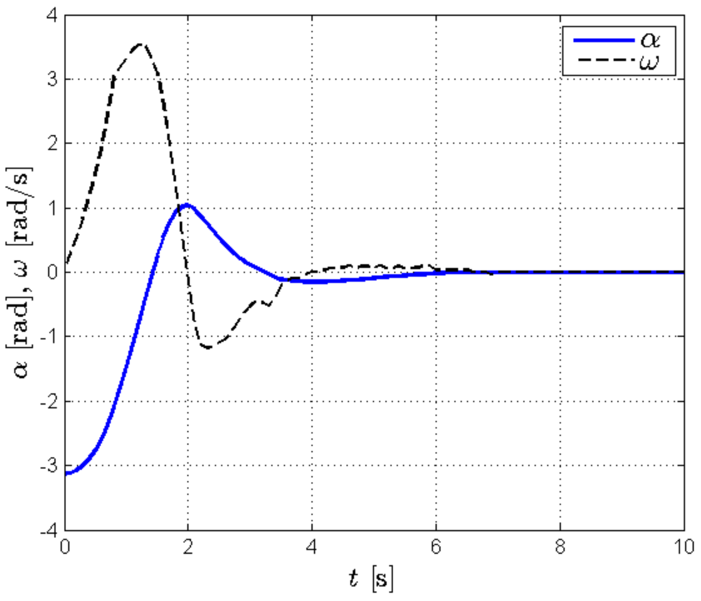


Fig. 5.9 State variables: α-blue and ω- black.

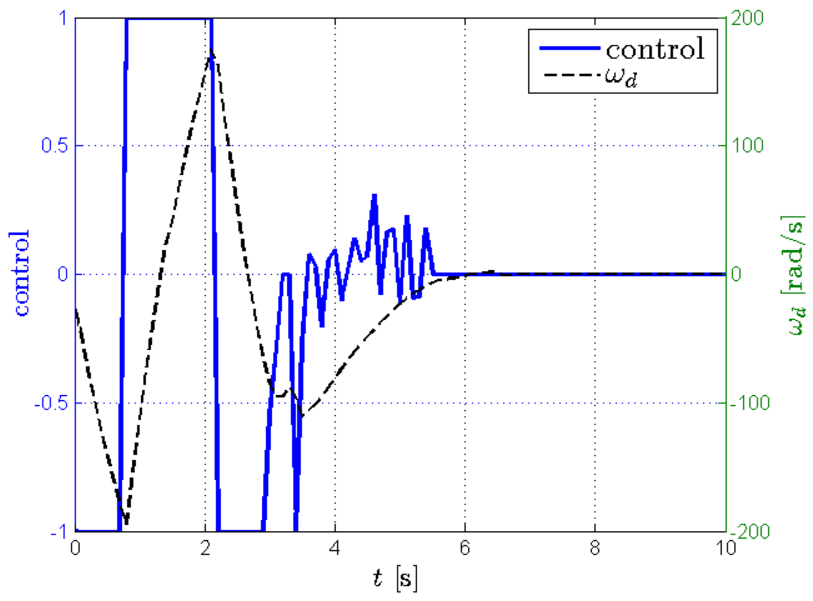


Fig. 5.10 Control signal - blue and disk velocity - black

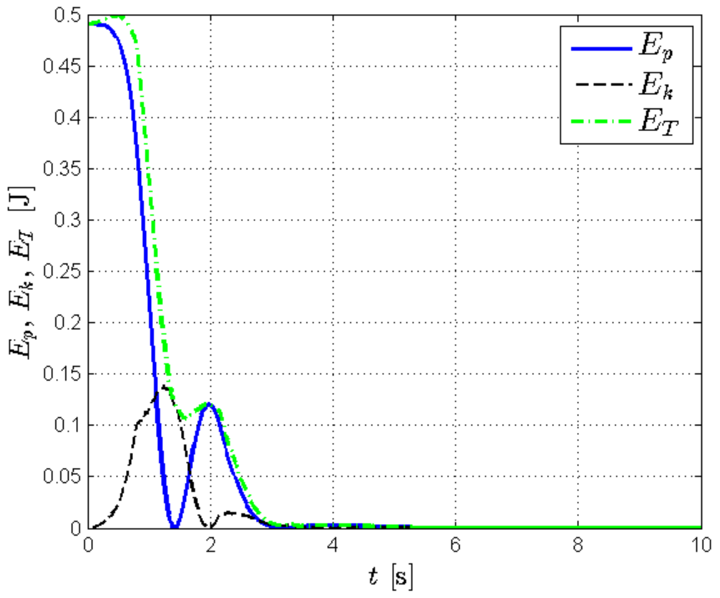


Fig. 5.11 Potential, Kinetic and Total Energy of the System

6. PROTOTYPING AN OWN CONTROLLER IN MATLAB/SIMULINK ENVIRONMENT

In this section the method of building your own controller is described. The Simulink Coder (formerly Real-Time Workshop) toolbox is used. Section 3 shows how to use the *RWP* software. Here we introduce the reader how to proceed in the Simulink Coder environment.

⇨

Before starting, test your MATLAB configuration and compiler installation by building and running an example of a real-time application. MSS toolbox includes the real-time model of PC speaker.

The model pc_speaker_xxx.mdl does not have any I/O blocks so that you can run this model regardless of the I/O boards in your computer. Running this model will test the installation by running Real-Time Workshop, and your third-party C compiler.

In the MATLAB command window, type

pc_speaker_vc

Build and run this real-time model as follows:

- From the *Tools* menu, point to *Real-Time Workshop*, and then click *Build Model*.

The MATLAB command window displays the following messages.

```
### Starting Real-Time Workshop build procedure for model:
pc_speaker_vc
### Generating code into build directory: C:\apps\MATLAB\R2006a\work\
pc_speaker_vc_RTCON
### Invoking Target Language Compiler on pc_speaker_vc.rtw. . .
.
.. . .
### Created RT-CON executable: pc_speaker_vc.dll
### Successful completion of Real-Time Workshop build procedure for model:
pc_speaker_vc
```

- From the *Simulation* menu, click *External*, and then click *Connect to target*.

The MATLAB command window displays the following message.

```
Model pc_speaker_vc loaded
```

- From *Simulation* menu, click *Start real-time code*.

The *Dual* scope window displays the output signals and you hear the speaker sound.

.

6.1 CREATING A MODEL

The simplest way to create a Simulink model for the RWP system is to use the *Device Driver* as a template. Save this model as the *MySystem.slx* Simulink diagram. The *MySystem* Simulink model is shown in Fig. 6.1.

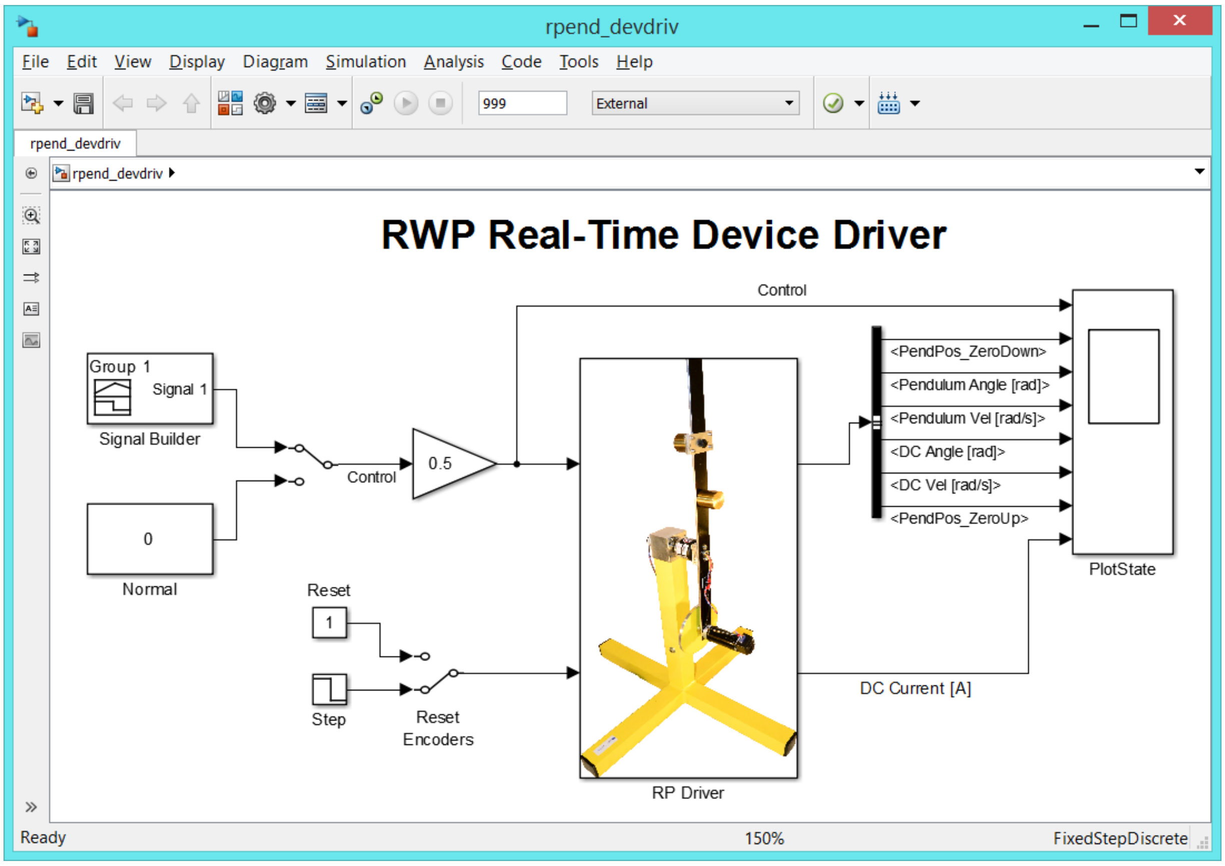


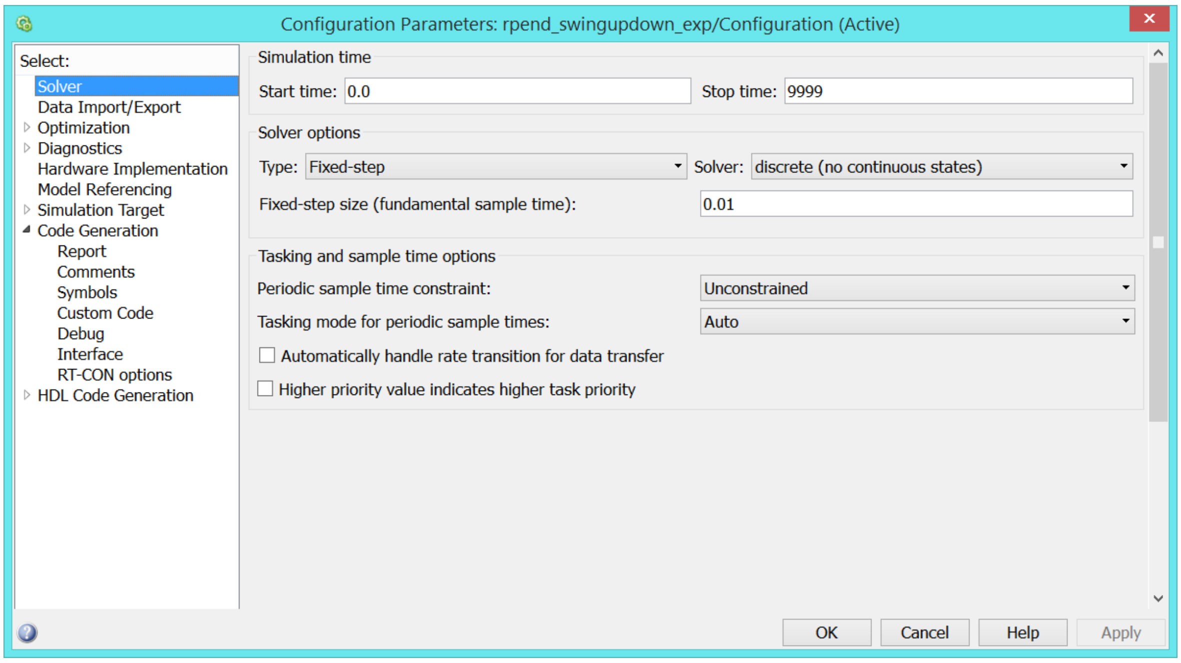
Fig. 6.1 The *MySystem* Simulink model

Now, you can modify the model. You have absolute freedom to develop your own controller. Remember to do not delete the *RP Driver* block. Though it is not obligatory, we recommend you to leave the scope (*PlotState* block in Fig. 6.1). You need a scope to watch how the system runs. Other blocks in the window are not necessary for a new project.

Creating your own model on the basis of the old example ensures that all internal options of the model are set properly. These options are required to proceed with compiling and linking in a proper way. To put the *RWP Driver* into the real-time code a target language compiler and a template makefile are required. Those files are included in the system software.

You can use the most of the blocks from the Simulink library. However, some of blocks are not supported (see *MathWorks* references manual for details).

The Fig. 6.2 and Fig. 6.3 show the proper configuration of all parameters of the model to create a real-time working executable.



Configuration Parameters: rpend_swingupdown_exp/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface
  - RT-CON options
- HDL Code Generation

Simulation time

Start time: 0.0 Stop time: 9999

Solver options

Type: Fixed-step Solver: discrete (no continuous states)

Fixed-step size (fundamental sample time): 0.01

Tasking and sample time options

Periodic sample time constraint: Unconstrained

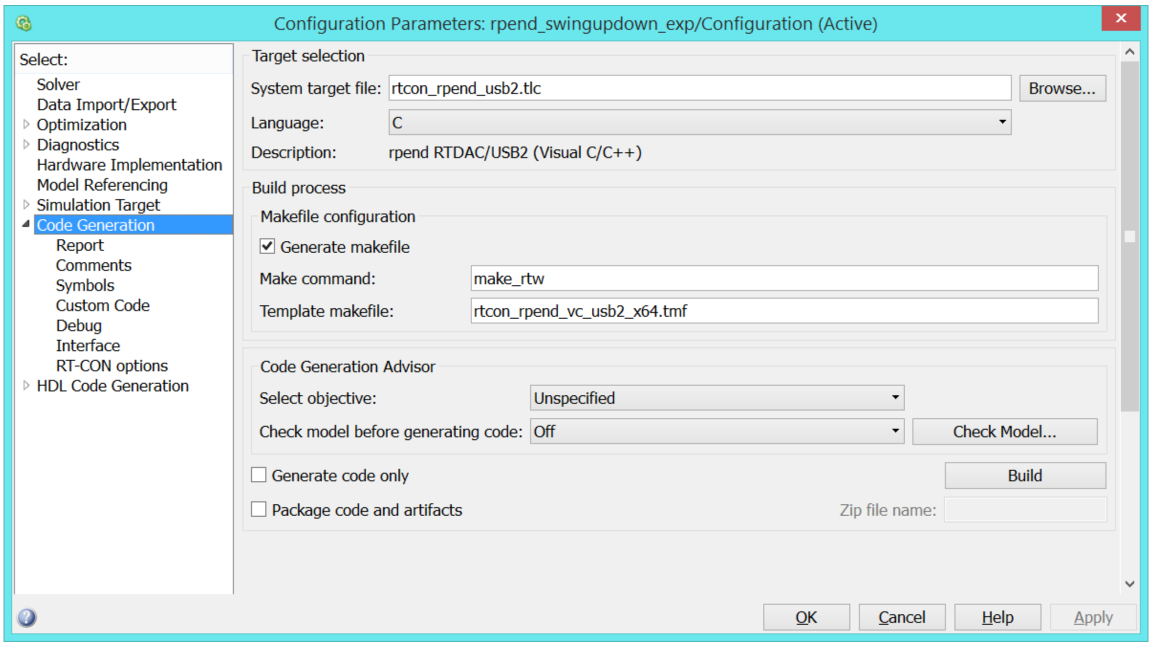
Tasking mode for periodic sample times: Auto

☐ Automatically handle rate transition for data transfer

☐ Higher priority value indicates higher task priority

Fig. 6.2 Solver tab

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's library remember, that different sampling periods must have a common divider.


Configuration Parameters: rpend_swingupdown_exp/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface
  - RT-CON options
- HDL Code Generation

Target selection

System target file: rtcon_rpend_usb2.tlc

Language: C

Description: rpend RTDAC/USB2 (Visual C/C++)

Build process

Makefile configuration

☑ Generate makefile

Make command: make_rtw

Template makefile: rtcon_rpend_vc_usb2_x64.tmf

Code Generation Advisor

Select objective: Unspecified

Check model before generating code: Off

☐ Generate code only

☐ Package code and artifacts

Zip file name:

Fig. 6.3 Configuration Parameters – *Code Generation*

7. REAL-TIME MODEL IN MATLAB VERSION R2019B OR NEWER

The previous section described creating and running real-time models for older versions of Matlab.

Since the R2019b version MathWorks has changed the look and the way of handling real-time models created in Simulink. This section describes how to compile (build) and run a real-time model for these newer versions of Matlab. The *Simulink Coder* and *RT-CON* toolboxes are used.

Suppose we have designed the My_System model shown below in Fig. 7.1. This model was created as a copy of any real-time model contained in the INTECO software.

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 *Device Driver* and other blocks chosen from the Simulink library,
- build the executable file compiling model,
- start the real-time code.

The description in this section contains only the differences from the previous versions of MATLAB. So reading the previous chapter carefully is necessary to understand the process of creating and running real time.

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 INTECO's software. as a template. For example, click any model and save it as *MySystem* name. The *MySystem* Simulink model is shown in Fig. 7.1.

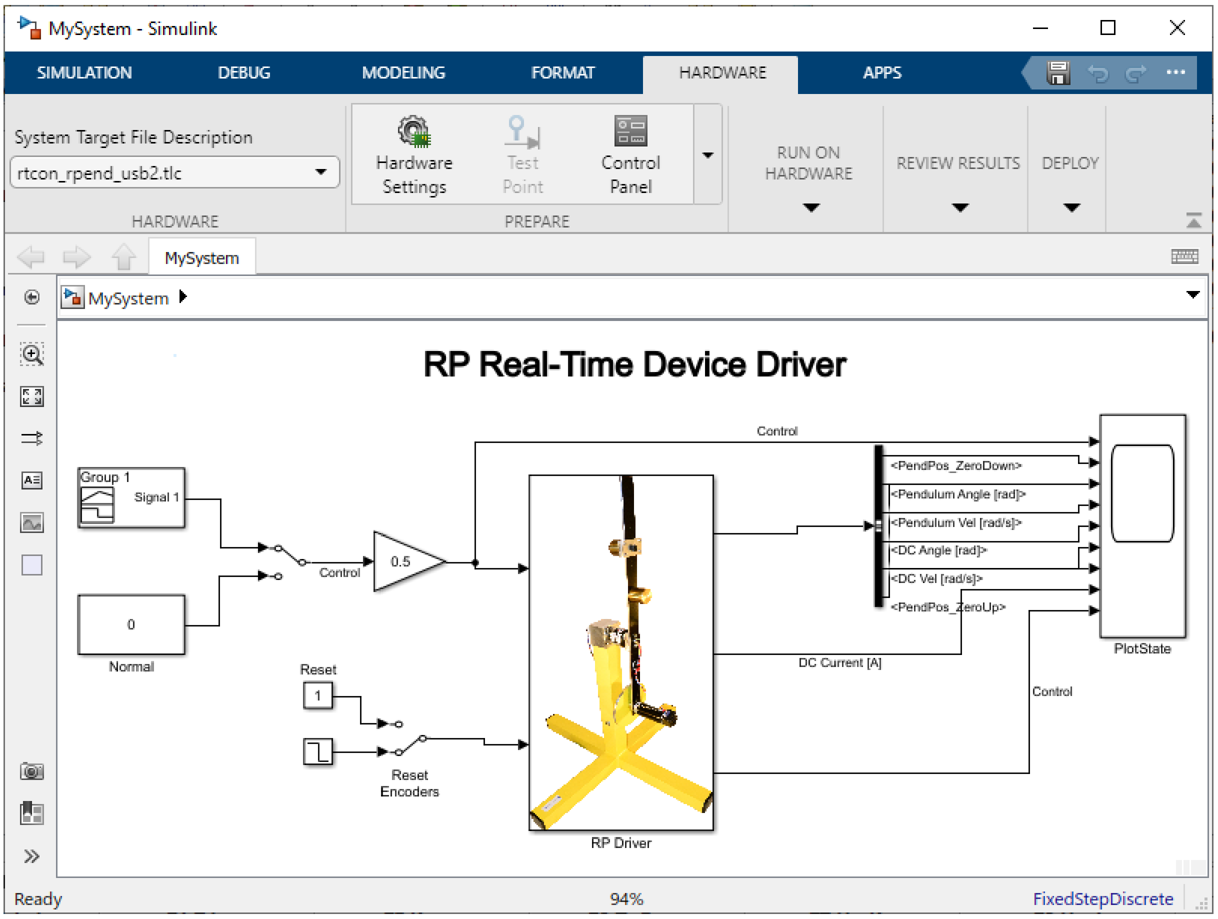


Fig. 7.1. My_System real-time model

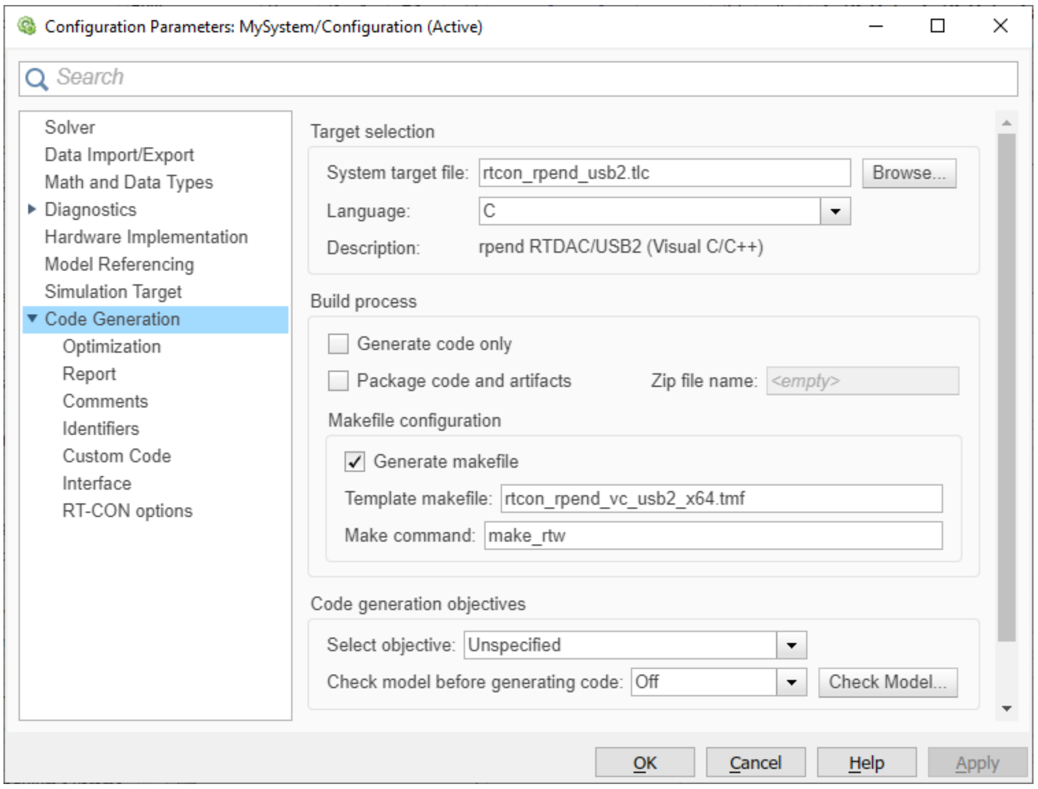
Now, you can modify the model. You get absolute freedom to develop your own controller. Remember to leave the *Device driver* block in the window. This is necessary to work in the real-time environment.

Though it is not obligatory, we recommend you to leave at least one scope. You need a scope to watch how the system runs.

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 compiling and linking in a proper way. See at Fig. 7.2 and Fig. 7.3. To build real-time code a special files shown in *System target file* and *Template file* sections are required. These files are included to the INTECO's software.

You can apply most of the blocks from the Simulink library. However, some of them cannot be used (see Simulink Coder reference manuals).

When the Simulink model is ready, click the *Hardware Settings* option and next click the *Code Generation* button. The window presented in Fig. 7.2 opens. Select *Interface* button to see the external mode options at Fig. 7.3. The system target file name is *rtcon_tras_USB2.tlc*. It manages the code generation process.



Target selection

System target file: rtcon_rpend_usb2.tlc

Language: C

Description: rpend RTDAC/USB2 (Visual C/C++)

Build process

☐ Generate code only

☐ Package code and artifacts Zip file name: <empty>

Makefile configuration

☑ Generate makefile

Template makefile: rtcon_rpend_vc_usb2_x64.tmf

Make command: make_rtw

Code generation objectives

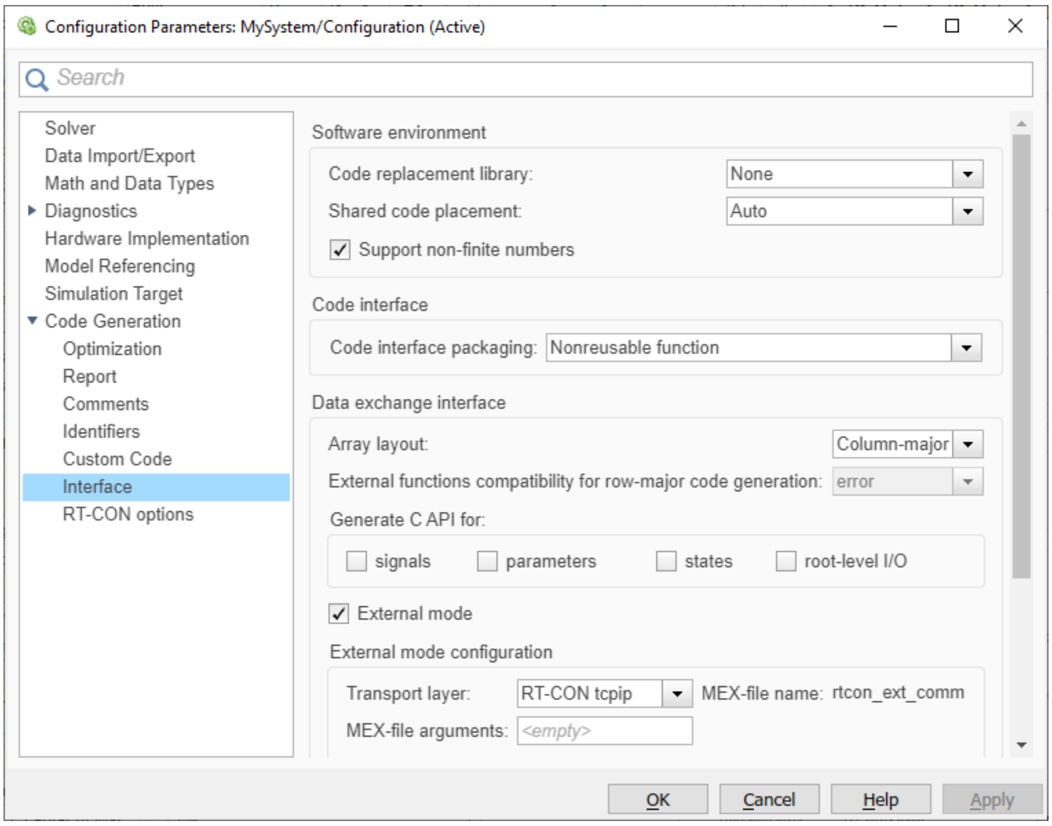
Select objective: Unspecified

Check model before generating code: Off

Fig. 7.2 Internal code generation options

The *rtcon_tras_vc_us2b.tmf* template make-file is devoted to C code generation using the Visual C++ compiler.

These files are examples and are of course different for different systems.



Configuration Parameters: MySystem/Configuration (Active)

Software environment

Code replacement library: None

Shared code placement: Auto

[✓] Support non-finite numbers

Code interface

Code interface packaging: Nonreusable function

Data exchange interface

Array layout: Column-major

External functions compatibility for row-major code generation: error

Generate C API for: [ ] signals [ ] parameters [ ] states [ ] root-level I/O

[✓] External mode

External mode configuration

Transport layer: RT-CON tcpip MEX-file name: rtcon_ext_comm

MEX-file arguments: <empty>

Fig. 7.3. Interface of the external mode options

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.

To compile (build) model click *Monitor&Tune* button and next *Build for Monitoring* option (see in Fig. 7.4). You can also use the keyboard by clicking CTRL+b keys. You will get the same effect in this way.

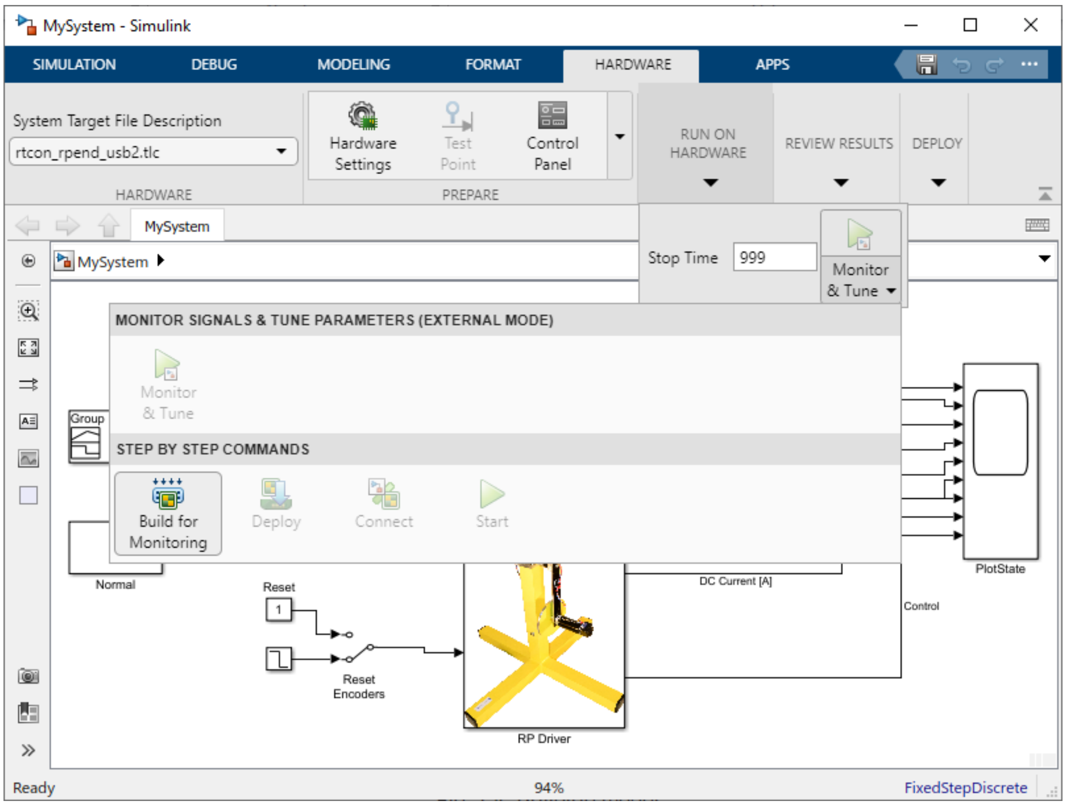


Fig. 7.4. Building model

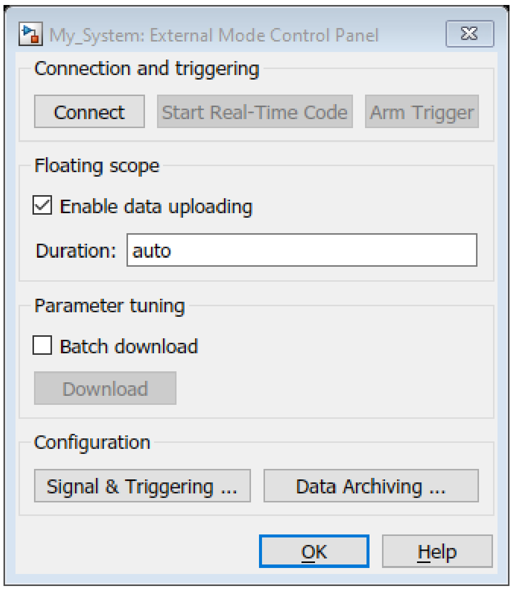
Successful compilation and linking processes generate the following message:

### Successful completion of build procedure for model: My_System

### Simulink cache artifacts for 'My_System' were created in '..My_System.slxc'. Build process completed successfully

Otherwise, an error massage is displayed in the *Diagnostic Viewer* window.

To run the real-time model click the *Control Panel* option and window shown in Fig. 7.5 will appear. Next click *Connect* button and real-time starts.



☑ Enable data uploading

Duration: auto

☐ Batch download

Fig. 7.5 Connect with target

Do not use option *Build Stand-Alone* shown in Fig. 7.6 for compilation .

.

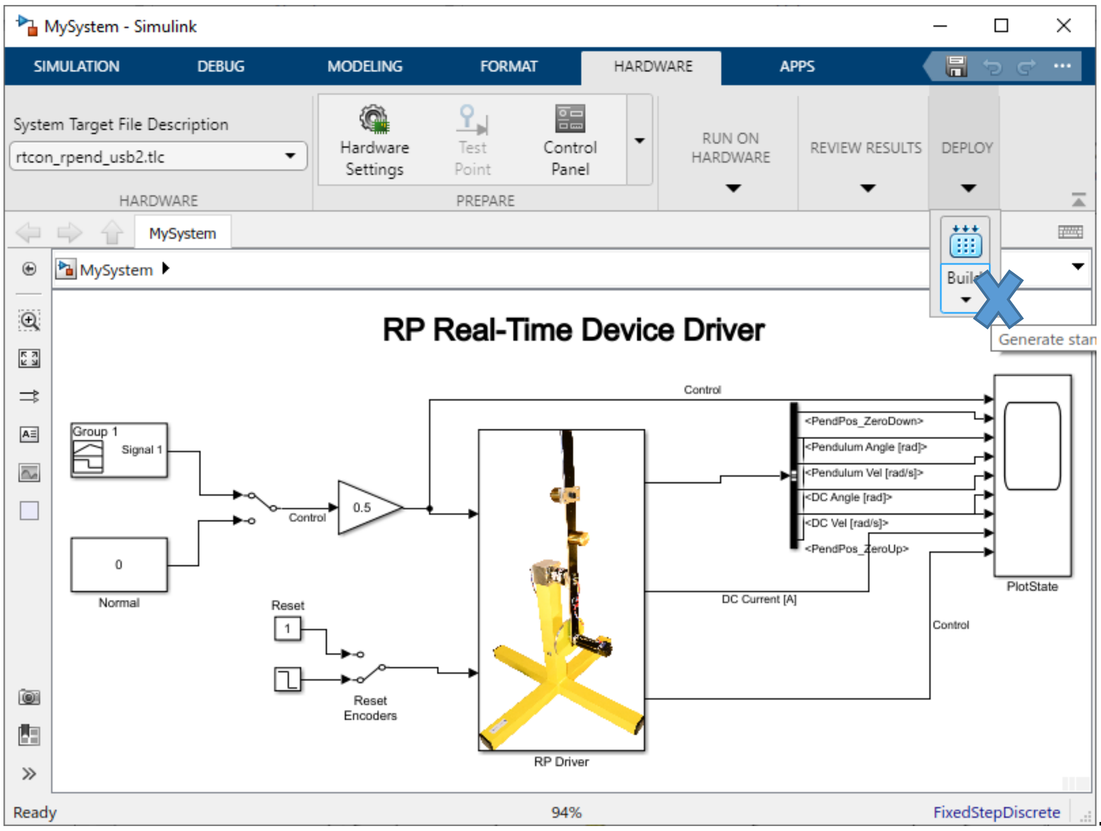


Fig. 7.6 Do not use this option !!