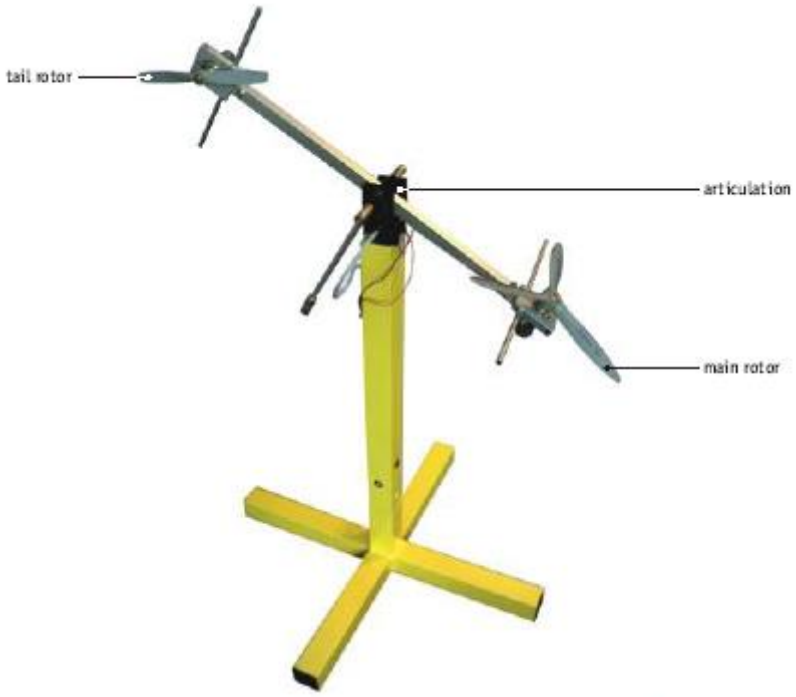
Two Rotor
Aero-dynamical System
(TRAS)

Windows 7/10 x86 x64
USB2 version

User's Manual



www.inteco.com.pl

Table of contents

1. Introduction

The **T**wo **R**otor **A**ero-dynamical **S**ystem (TRAS) is a laboratory set-up designed for control experiments. In certain aspects its behaviour resembles that of a helicopter. From the control point of view it exemplifies a high order non-linear system with significant cross-couplings. 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 RTDAC/USB2 I/O board configured in the Xilinx® technology. The software operates in real time under MS Windows®7 x86 or x64 using MATLAB®, Simulink and RTW toolboxes. Real-time is supported by the RT-CON toolbox from INTECO.

Control experiments are programmed and executed in real-time in the MATLAB/Simulink environment. Thus it is strongly recommended to a user to be familiar with the RTW toolbox. One has to know how to use the attached models and how to create his own models.

The approach to control problems corresponding to the TRAS proposed in this manual involves some theoretical knowledge of laws of physics and some heuristic dependencies difficult to be expressed in analytical form.

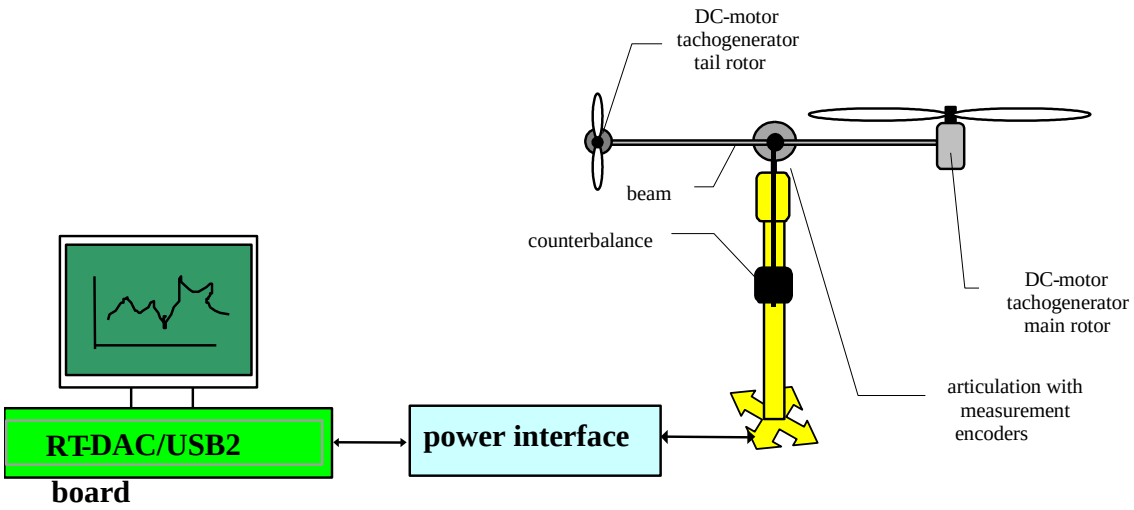


Fig. 1.1 The laboratory set-up: helicopter-like system

A schematic diagram of the laboratory set-up is shown in Fig. 1.1.The TRAS consists of a beam pivoted on its base in such a way that it can rotate freely both in the horizontal and vertical planes. At both ends of the beam there are rotors (the main and tail rotors) driven by DC motors. A counterbalance arm with a weight at its end is fixed to the beam at the pivot. The state of the beam is described by four process variables: horizontal and vertical angles measured by position sensors fitted at the pivot, and two corresponding angular velocities. Two additional state variables are the angular velocities of the rotors, measured by tacho-generators coupled with the driving DC motors.

In a casual helicopter the aerodynamic force is controlled by changing the angle of attack. The laboratory set-up from Fig. 1.1 is so constructed that the angle of attack is fixed. The aerodynamic force is controlled by varying the speed of rotors. Therefore, the control inputs are the supply voltages of the DC motors. A change in the voltage value results in a change of the rotation speed of the propeller which results in a change of the corresponding position of the beam. Significant cross-couplings are observed between the actions of the rotors: each rotor influences both position angles. Designing of stabilising controllers for such a system is based on decoupling. For a decoupled system an independent control input can be applied for each coordinate of the system.

An IBM-PC compatible computer can be used for real-time control of TRAS. The computer must be supplied with an interface board (RTDAC/USB2). Fig. 2.5 Measurements of the beam motion shows details of the hardware configuration of the control system for TRAS.

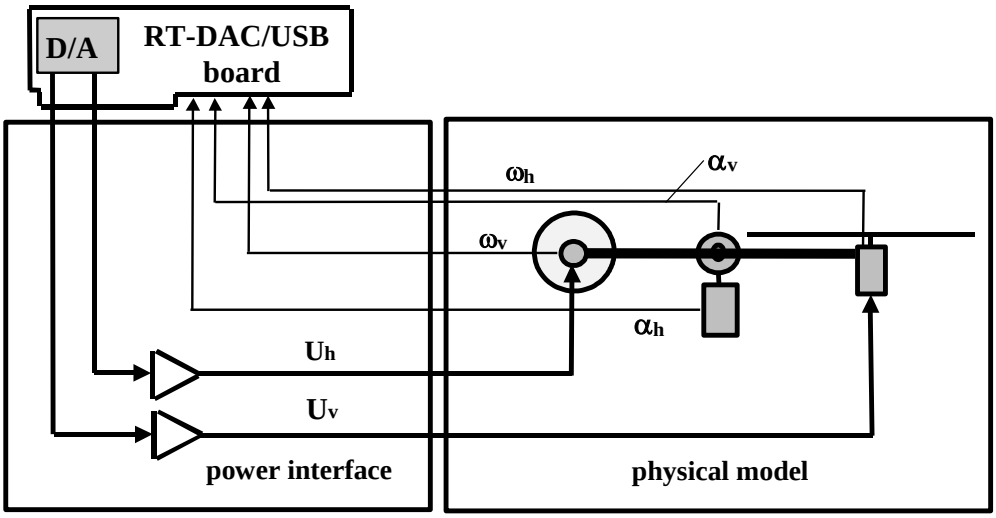


Fig. 1.2 Hardware configuration of the TRAS

The control software for the TRAS is included in the *TRAS toolbox*. This toolbox uses the RTW toolbox from MATLAB and the RT-CON toolbox from INTECO.

TRAS Toolbox is a collection of M-functions, MDL-models and C-code DLL-files that extends the MATLAB environment in order to solve TRAS modelling, design and control problems. The integrated software supports all phases of a control system development:

- on-line process identification,
- control system modelling, design and simulation,
- real-time implementation of control algorithms.

TRAS Toolbox is intended to provide a user with a variety of software tools enabling:

- on-line information flow between the process and the MATLAB environment,
- real-time control experiments using demo algorithms,

- development, simulation and application of user-defined control algorithms.

1.2 Hardware and software requirements.

TRAS Toolbox is distributed on a CD-ROM. It contains the software and TRAS *User's Manual*. The *Installation Manual* is distributed in a printed form.

Hardware

Hardware installation is described in the *Installation* manual. It consists of:

- TRAS Mechanical Unit,
- Power interface and wiring allowing electrical connections to the TRAS set,
- RTDAC/USB2 I/O board. The board contains FPGA equipped with dedicated logic,
- Pentium or AMD based personal computer.

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

The TCP/IP protocol must be installed in the computer system,

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

⇨ **Real-time is supported by the RT-CON toolbox from INTECO (included in TRAS Toolbox and transparent for a user).**

Manuals:

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

⇨ **The experiments and corresponding to them 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 these given in the manual.**

1.3 FEATURES of TRAS

- A highly nonlinear MIMO system ideal for illustrating complex control algorithms.
- The system can be easily installed.
- 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.

Application note

The documentation assumes that the user has a basic experience with MATLAB, Simulink, and RTW toolboxes from *MathWorks Inc.*

1.4 Software installation

Insert the installation CD and proceed step by step following displayed commands.

2. Starting and testing procedures

The TRAS system is an “open” type. It means that a user can design and solve any TRAS control problem on the basis of the attached hardware and software. The software includes device drivers compatible with RTW toolbox. It is assumed that a user is familiarised with MATLAB tools especially with RTW toolbox. Therefore we do not include the detailed description of this tool.

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

Open Matlab

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

then type:

Tras

and the *TRAS Control Window* opens (see Fig. 2.1)

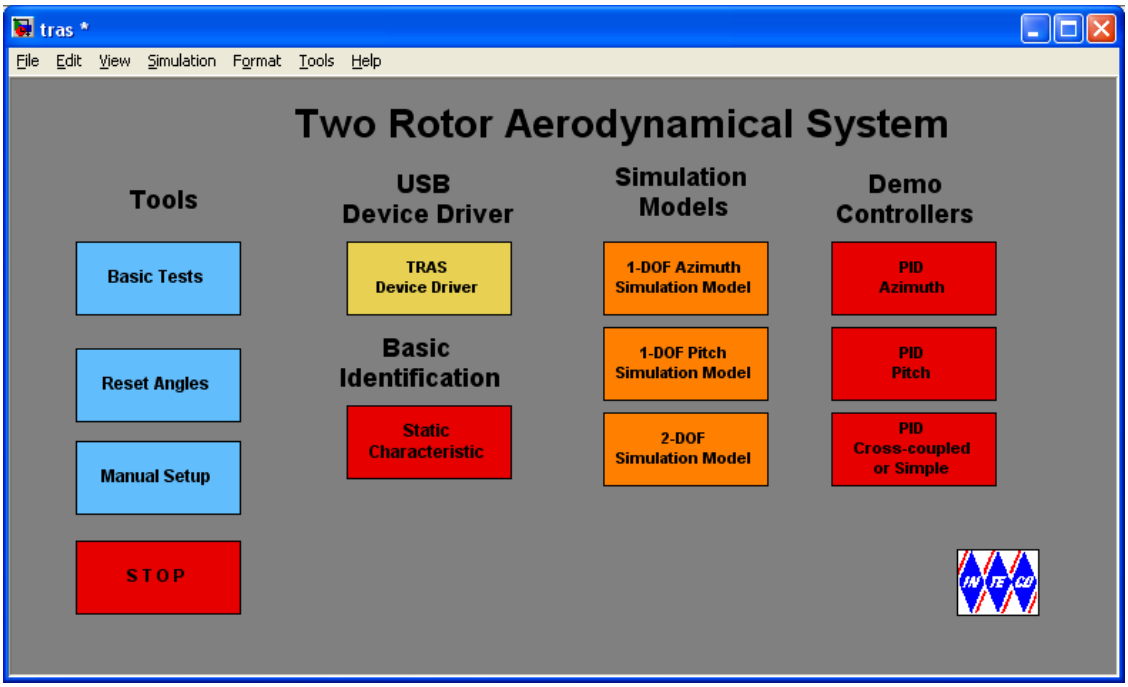


Fig. 2.1 TRAS Control Window

TRAS Control Window contains testing tools, drivers, models and demo applications..

2.2 Testing and troubleshooting

This section explains how to perform the tests. One can check if mechanical assembling and wiring has been done correctly. The tests have to be performed obligatorily after assembling the system. They are also necessary if an incorrect operation of the system happens. Due to the tests sources of the system fails can be tracked. The tests have been designed to validate the existence and sequence of measurements and controls. They do not relate to accuracy of the signals.

At the beginning one has to be sure that all signals are transmitted and transferred in a proper way. The following steps are applied:

- Double click the *Basic Tests* button. The *Basic Test* window appears (Fig. 2.2)

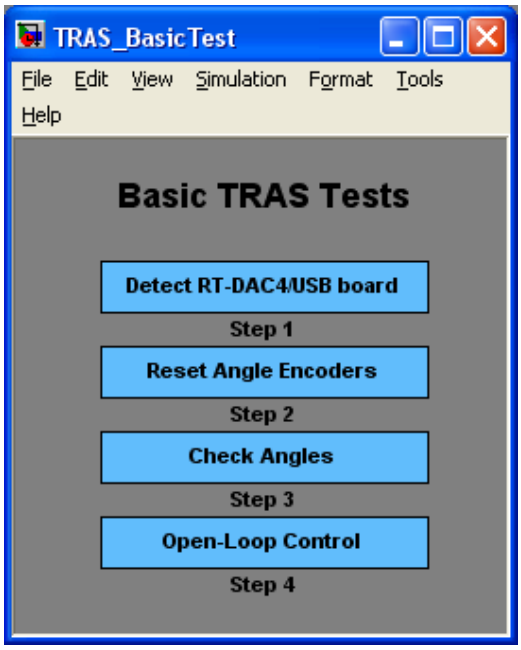


Fig. 2.2 The *Basic Tests* window

The experiment may be stopped in any time. Double click on the *Stop* block in the *TRAS Control Window* or somewhere else. If you wish to stop the visualisation process click once on the *Stop* bar in the *Simulation* menu.

The first step in the Modular Servo System testing is to check if the RTDAC/USB2 measuring and control board is installed properly.

- Double click the *Detect RTDAC/USB board* button. One of the messages shown in Fig. 2.3 opens. If the board has been correctly installed the left window is displayed.

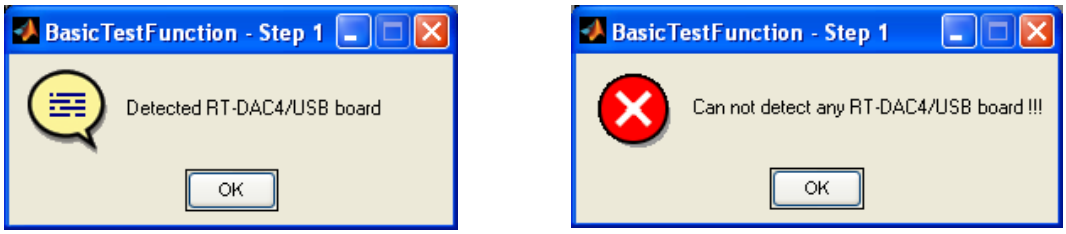


Fig. 2.3 Result of the step 1

If the board is not detected then check whether the board has been powered. The boards are checked very precisely before sending to a customer. In principle, a wrong assembling is the only reason of no detecting the board.

The next step consists in resetting the encoders. It means that the initial position of the beam is stored in the memory.

- Double click the *Reset Angles* button. When Fig. 2.4 opens, move the *TRAS* system to the origin position and then click the *Yes* option. The encoders reset and zero positions of the beam are going to be remembered so long as an measurement error occurs.

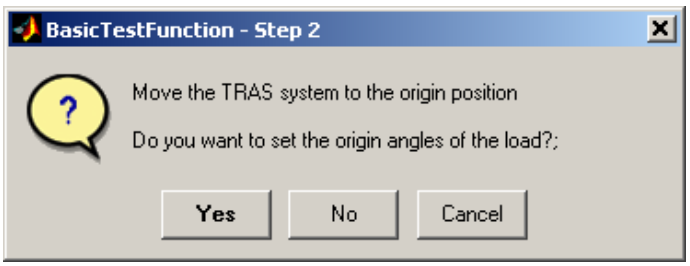


Fig. 2.4 The *Reset Angles* window

Double click the *Check Angles* button. When the window opens click *Yes,* then, move by hand the beam of the TRAS in all directions and observe measurements on the screen (see Fig. 2.5).

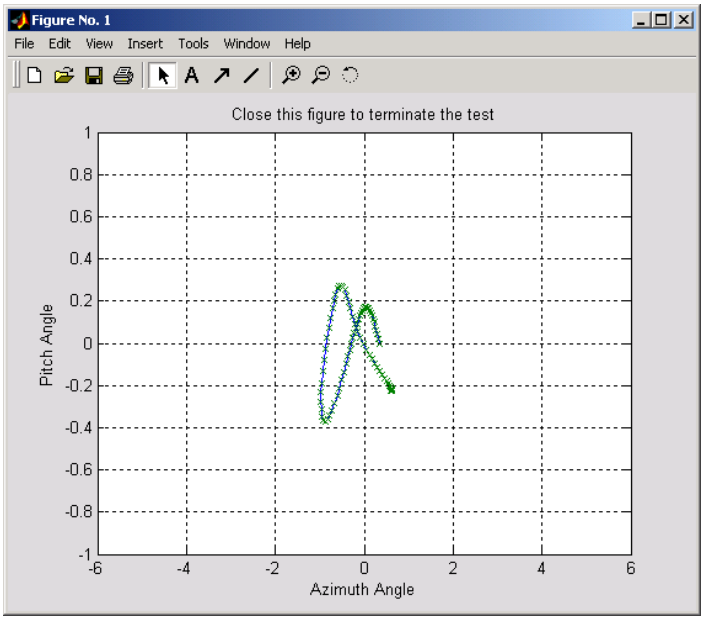


Fig. 2.5 Measurements of the beam motion

In the next step one checks if the main and tail motors work properly.

Double click the *Open loop control* button. When Fig. 2.6 opens one can to set the control inputs to the main and tail motor. The vertical axis corresponds to the main motor and the horizontal axis corresponds to the tail motor. When you locate the mouse pointer at [0 0.5] and click, then the control equal to 0.5 is set for the main motor. And if you click at [0.5 0] the control 0.5 is set for the tail motor. Using the mouse, click and slowly drug a rectangle. The motors rotate with respect to the mouse pointer location (the intersection of the green and red lines in Fig. 2.6). The red ends of the blue lines show the rotational velocities of the propellers. If the rectangle movement of the mouse is finished a picture similar to that given in Fig. 2.6 should be visible.

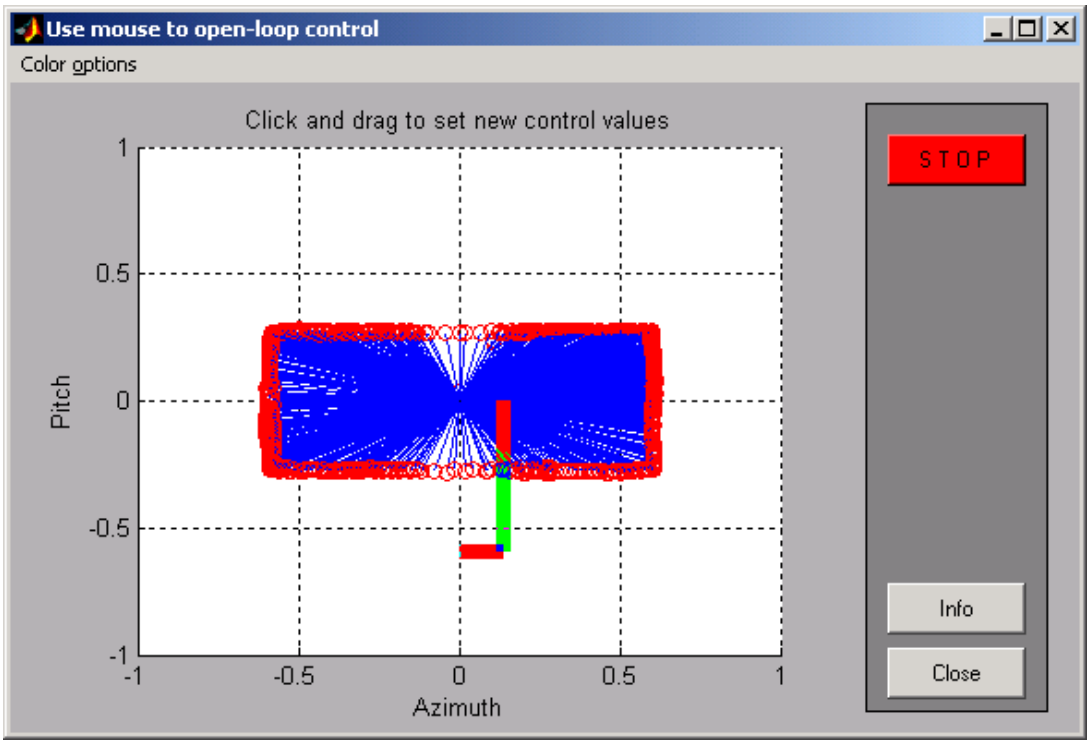


Fig. 2.6 Motors control and checking of tacho-generators

Troubleshooting

Message or faulty action	Solution
Board not detected	Check connecting of the board. Check if power switch is *ON*
Angles measurements failed	Check the Enc socket and wiring
Propellers do not rotate	Check M socket, Mains and ON switch
Velocities are not measured	Check T socket and wiring

3. TRAS Control Window

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

TRAS Control Window shown in Fig. 2.1 contains four groups of the menu items:

- Tools - Basic Test, Manual Setup, Reset Encoders and Stop Experiment,
- Drivers - USB Device Driver,
- Simulation Models: Pitch , Azimuth and 2-DOF model,
- Identification - Steady State Characteristics,
- Demo Controllers – PID azimuth, PID pitch and cross-coupled PID controller

3.2 Basic test

The *Basic Test* tool was described in the previous section.

3.3 TRAS Manual Setup

The *TRAS Manual Setup* program gives access to the basic parameters of the laboratory Two Rotor Aerodynamical System setup.

To run the *Manual Setup* in Windows XP click the *ManualSetup* button. If you are using Windows 7 do not use this button. Open windows explorer, find the *ManualSetup.exe* file in the matlabroot/toolbox/Tras/ManualSetup directory and double click this file.

The most important data transferred from the RTDAC/USB2 board and the measurements of the TRAS may be shown. Moreover, the control signals may be set.

The application contains four frames (see Fig. 3.1):

- RTDAC/USB2 board,
- Encoders,
- Control and
- Tacho.

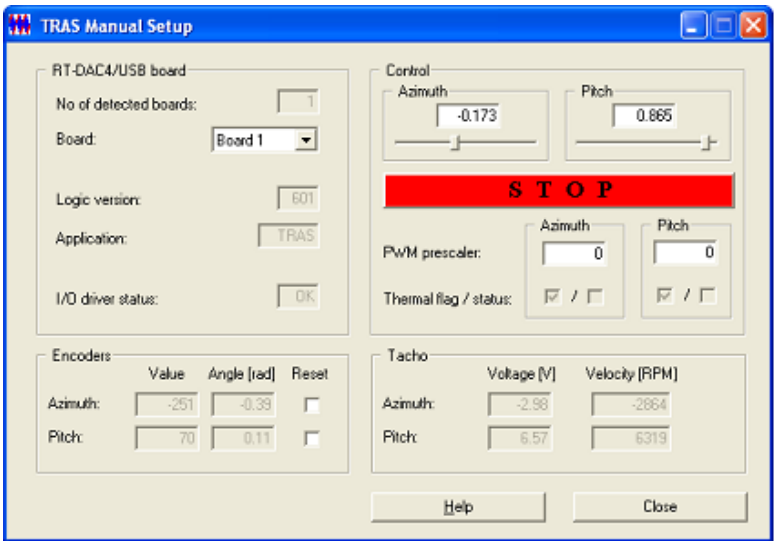


Fig. 3.1 View of the *TRAS Manual Setup* window

All the data accessible from the *TRAS Manual Setup* program are updated 10 times per second.

***RTDAC/USB2 board* frame**

The *RTDAC/USB2 board* frame presents the main parameters of the USB board.

No of detected boards

Reads the number of detected RTDAC/USB2 boards. If the number is equal to zero it means that the software has detected none of the RTDAC/USB2 board. When more then one board is detected the *Board* list must be used to select the board that communicates with the program.

Board

Contains the list applied to the selected board currently used by the program. The list contains a single entry for each RTDAC/USB2 board installed in the computer. A new selection executed at the list automatically changes values of the remaining parameters.

Logic version

The number of the configuration logic of the on-board FPGA chip. A logic version corresponds to the configuration of the RTDAC/USB2 boards defined by this logic.

Application

The name of the application the board is dedicated for. The name contains four characters.

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 the other case the driver HAS TO BE INSTALLED.

***Encoders* frame**

The state of the encoder channels is given in the *Encoder* frame. The encoders are applied to measure the azimuth and pitch angles.

Azimuth, Pitch

The values of the encoder counters, the angles expressed in radians and the encoder reset flags are listed in the *Azimuth* and *Pitch* rows.

Value

The values of the encoder counters are given in the respective columns. The values are 16-bit integer numbers. When an encoder remains in the reset state the corresponding value is equal to zero.

Angle [rad]

The angular positions of the encoders expressed in radians are given in the respective columns. If the encoder remains in the reset state the corresponding angle is equal to zero.

Reset

When the checkbox is selected the corresponding encoder remains in the reset state. The checkbox has to be unselected to allow the encoder to count the position.

***Control* frame**

The *Control* frame allows to change the control signals. DC drives are controlled by PWM signals.

Azimuth and Pitch edit fields and sliders

The control edit boxes and the sliders are applied to set a new control values of the corresponding DC drives. The control value may vary from –1.0 to 1.0.

STOP

The pushbutton is applied to switch off the control signals. If it is pressed then both the azimuth and pitch control values are set to zero.

Azimuth and Pitch PWM prescaler

The divider of the PWM reference signal is given. The frequency of the corresponding PWM control is equal to:

$$f_{pwm} = \frac{40}{(1 + PWMprescaler)} [KHz]$$

Azimuth and Pitch Thermal flag / status

The thermal flags and the thermal statuses of the power amplifiers. If the thermal status box is checked the corresponding power interface is overheated. If the power interface is overheated and the corresponding thermal flag is set the RTDAC/USB2 board switches off the PWM control signal corresponded to the overheated power amplifier.

***Tacho* frame**

The *Tacho* frame displays two measured analog signals generated by the tacho-generators. The voltages and the corresponding velocities of the propellers are displayed.

Azimuth and Pitch Voltage [V]

Displays the voltage at the outputs of the tacho-generators.

Azimuth and Pitch Velocity [RPM]

Displays the velocity of the propellers. The velocities are calculated based on the corresponding voltages and are given in RPM.

3.4 USB Device Driver

The driver is a software go-between for the real-time MATLAB environment and the RTDAC/USB2 I/O board. The control and measurements are transferred. Click the *TRAS Device Driver* button and the driver window opens (Fig. 3.2).

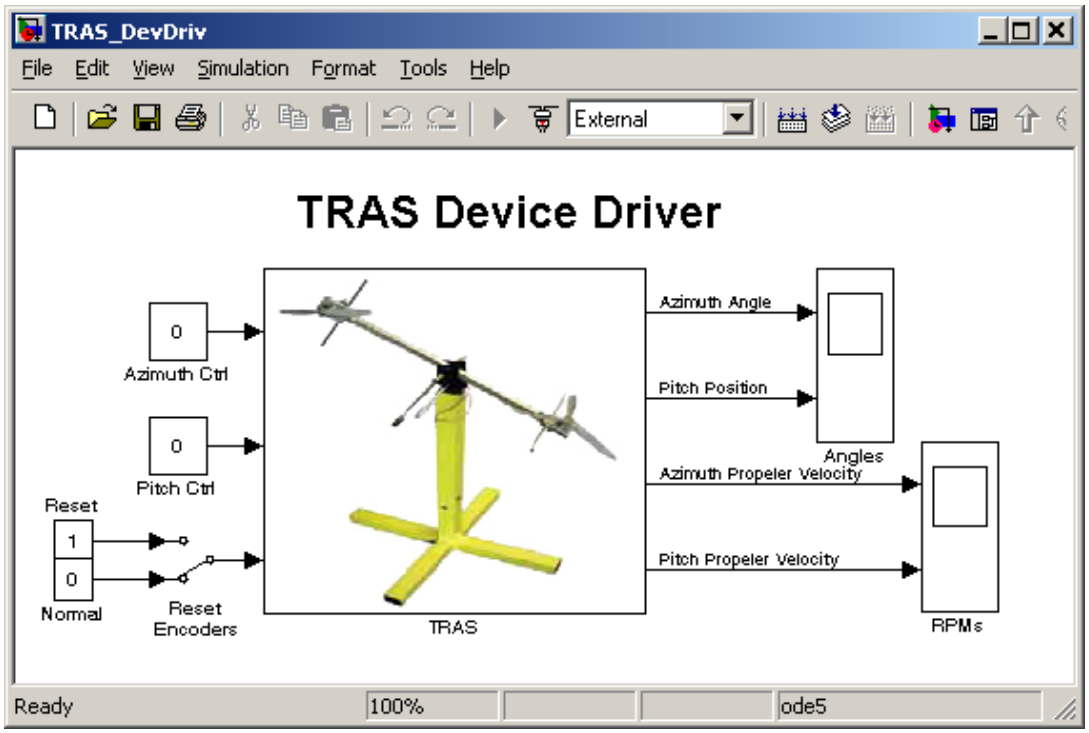


Fig. 3.2 USB2 Device Driver

When one wants to build his own application one can copy this driver to a new model. The *Reset Encoder* input can be used in the real-time mode only.

Do not do any changes inside the original driver. They can be introduced only inside its copy!!! Make a copy of the installation CD

The device driver has two inputs: control $u(t) \subset [-1+1]$ and signal *Reset*. If signal *Reset* changes to one the encoders are reset and do not work. If signal

Reset is equal to zero encoders normally work. It is important that *Reset* switch works only if the real-time code is executed. It means that changing the state of the switch, when real time mode is not running, is not effective. However when switching occurs while the real time is running, the encoder resets and starts measure when the switch returns to the zero (normal) position.

The details of the device driver are depicted in Fig. 3.3. The driver uses functions which communicates directly with a logic stored at the RTDAC/USB2 board.

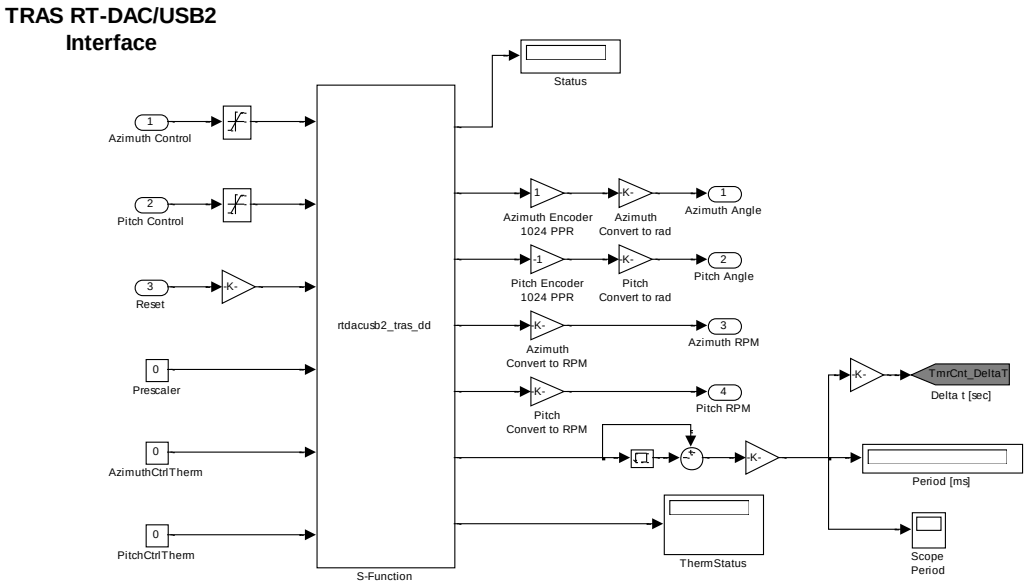
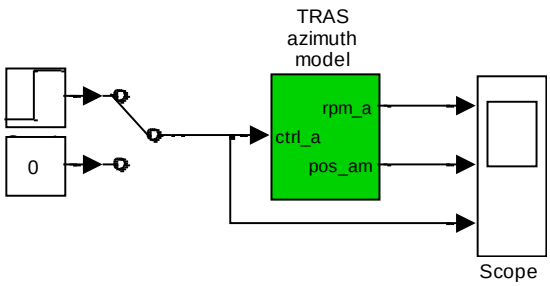


Fig. 3.3 Interior of the USB2 device driver

3.5 Simulation Models

There are three simulation models available for the *TRAS* system. The first one is a 1-DOF (degree of freedom) azimuth model. This model simulate behaviour of the system in the horizontal plane only. Click the *1-DOF Azimuth Simulation Model* button to open the model shown in Fig. 3.4. Next, click the subsystem block to see details of the model.



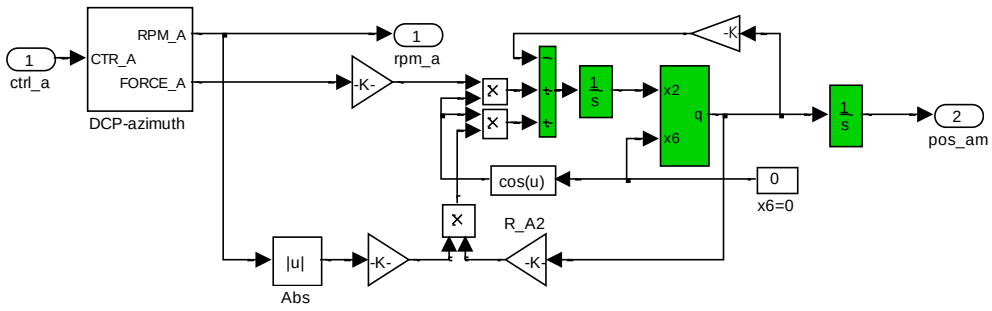


Fig. 3.4 The *Azimuth Simulation model* and its interior

A 1-DOF pitch is the second model. It describes behaviour of the system in the vertical plane. Click the *1-DOF Pitch Simulation Model* button and click the subsystem block to see the 1-DOF pitch model and its interior (see Fig. 3.5)

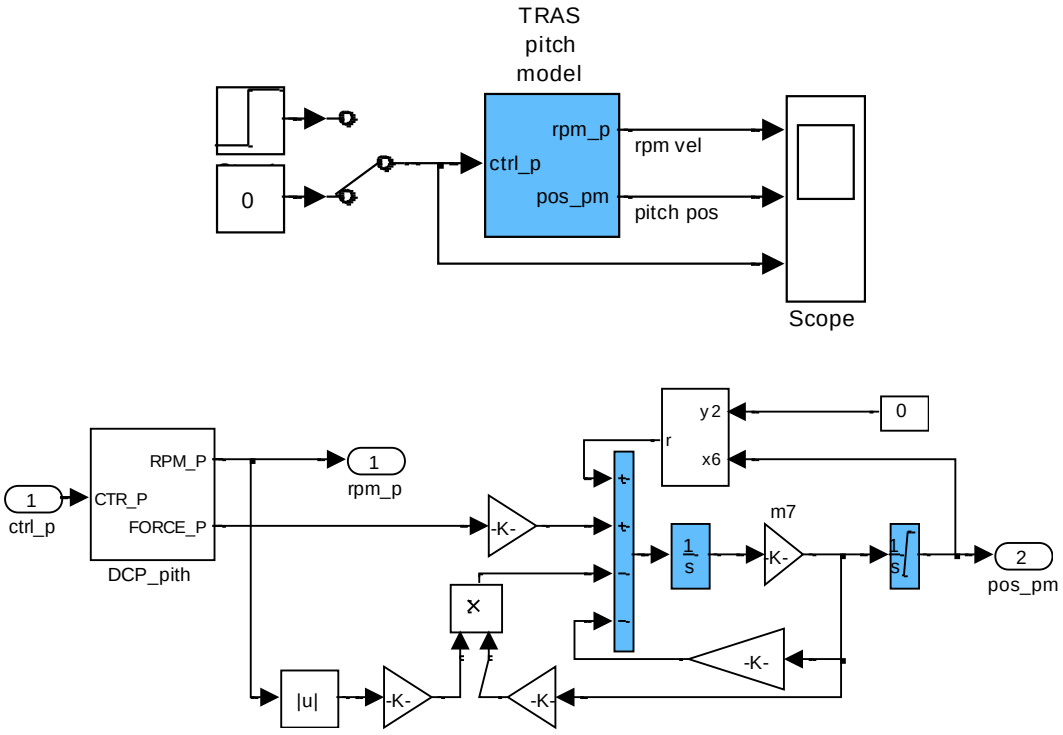


Fig. 3.5 The *Pitch Simulation* model and its interior

The third one is the complete simulation model. It describes movements in both planes with an interaction between the pitch and azimuth axes. Click the *2-DOF Simulation Model* button and the subsystem block to see the model and its interior (see Fig. 3.6)

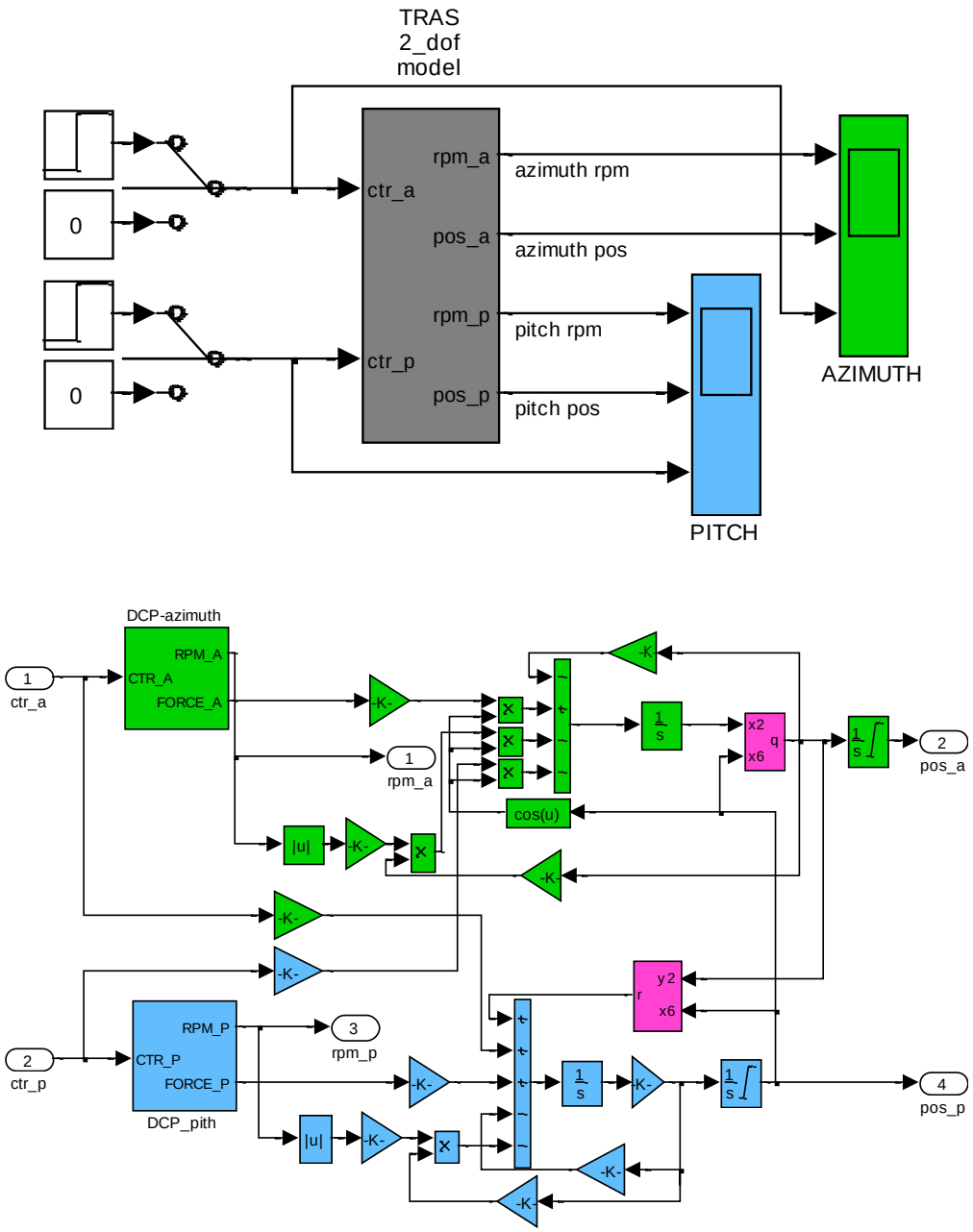


Fig. 3.6 The 2-DOF simulation model and its interior

4. Model and parameters

Modern methods of design and adaptation of real time controllers require high quality mathematical models of the system. For high order, non-linear cross-coupled systems classical modelling methods (based on Lagrange equations) are often very complicated. That is why a simpler approach is often used, which is based on block diagram representation of the system which is very suitable for the SIMULINK environment. The relations between the block diagram and mathematical model of the TRAS are explained in sections 4.2 – 4.5.

Fig. 4.1. shows an aero-dynamical system considered in this manual. At both ends of a beam, joined to its base with an articulation, there are two propellers driven by DC-motors. The articulated joint allows the beam to rotate in such a way that its ends move on spherical surfaces. There is a counter-weight fixed to the beam and it determines a stable equilibrium position. The system is balanced in such a way, that when the motors are switched off, the main rotor end of beam is lowered. The controls of the system are the motor supply voltages.

The measured signals are: position of the beam in the space that is two position angles and angular velocities of the rotors. Angular velocities of the beam are software reconstructed by differentiating and filtering measured position angles of the beam.

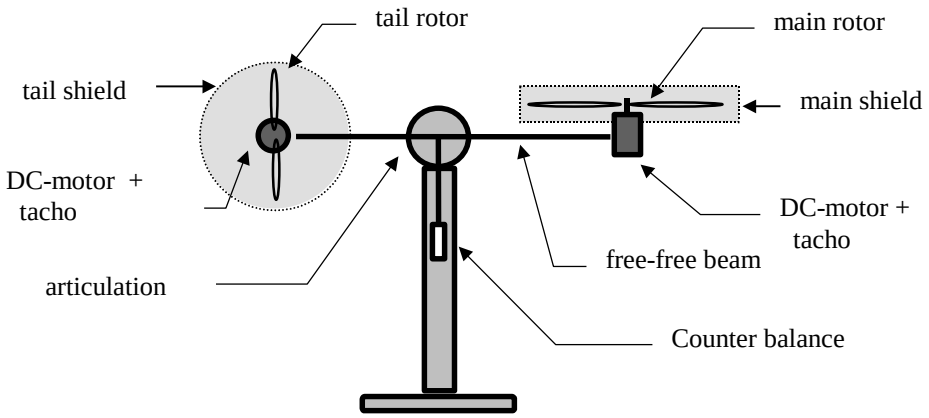


Fig. 4.1. Aero-dynamical model of TRAS

The block diagram of the TRAS model is shown in Fig. 4.2. The control voltages U_h and U_v are inputs to the DC-motors which drive the rotors (PWM mode).

A rotation of the propeller generates an angular momentum which, according to the law of conservation of angular momentum, must be compensated by the remaining body of the TRAS beam. This results in the interaction between two transfer functions, represented by the moment of inertia of the motors with propellers k_{hv} and k_{vh} (see Fig. 4.2). This interaction directly influences the velocities of the beam in both planes. The forces F_h and F_v

multiplied by the arm lengths $l_h(\alpha_v)$ and l_v are equal to the torques acting on the arm.

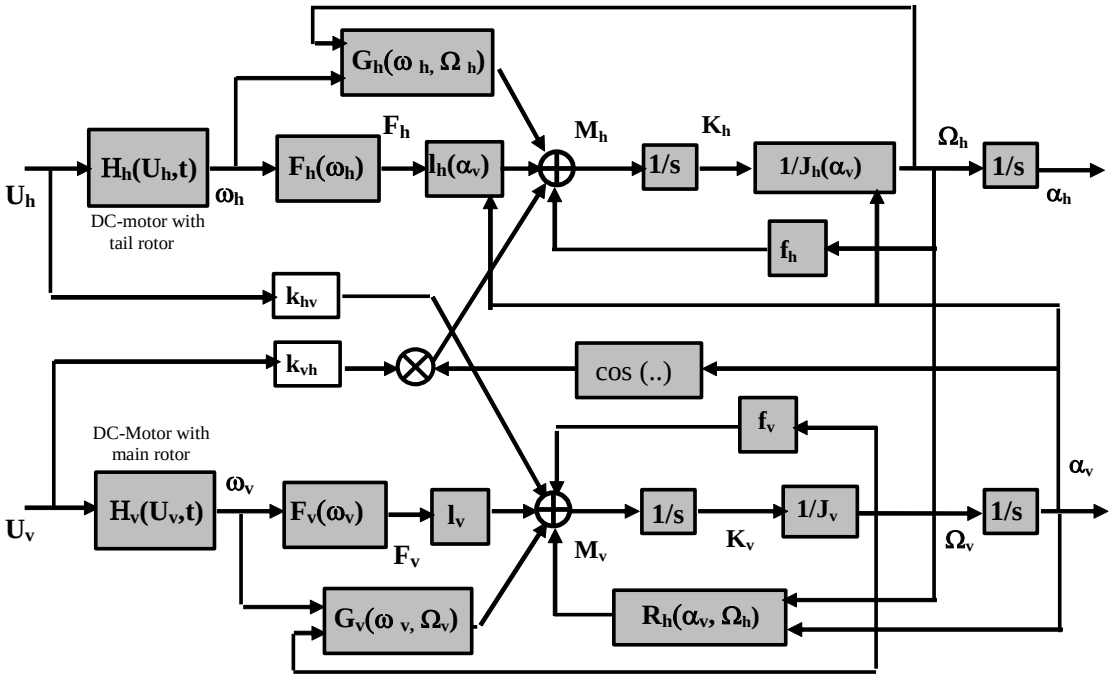


Fig. 4.2 Block diagram of the TRAS model

The following notation is used in Fig. 4.2:

α_h - horizontal position (azimuth position) of TRAS beam [rad];

Ω_h - angular velocity (azimuth velocity) of TRAS beam [rad/s];

U_h - horizontal DC-motor PWM control input ;

ω_h - rotational speed of tail rotor [rad/s] - non-linear function $\omega_h = H_h(U_h, t)$ [rad/s] ;

F_h - aerodynamic force from tail rotor - non-linear function $F_h = F_h(w_h)$ [N];

l_h - effective arm of aerodynamic force from tail rotor $l_h = l_h(a_v)$ [m];

J_h - non-linear function of moment of inertia with respect to vertical axis, $J_h = J_h(a_v)$ [kg m^2];

M_h - horizontal turning torque [Nm];

K_h - horizontal angular momentum [N m s];

f_h - friction coefficient in a horizontal plane [N m];

α_v - vertical position (pitch position) of TRAS beam [rad];

Ω_v - angular velocity (pitch velocity) of TRAS beam [rad/s];

U_v - vertical DC-motor PWM voltage control input;

ω_v - rotational speed of main rotor - non-linear function $\omega_v = H_v(U_v, t)$ [rad/s];

F_v - aerodynamic force from main rotor **-** non-linear function $F_v = F_v(\omega_v)$ [N];

l_v - arm of aerodynamic force from main rotor [m];

J_v - moment of inertia with respect to horizontal ax- [kg m^2];

M_v - vertical turning moment [Nm];

K_v - vertical angular momentum [Nms];

f_v - friction coefficient in a vertical plane [Nm];

R_v - vertical returning moment $R_h = f_{cf} + f_g = R_h(\alpha_v, \Omega_h)$ [Nm];

J_{hv} - vertical angular momentum from tail rotor [Nms];

J_{vh} - horizontal angular momentum from main rotor [Nms];

H_v - differential equation $\omega_v = H_v(U_v, t)$;

H_h - differential equation $\omega_h = H_h(U_h, t)$;

G_v - aerodynamical dumping torque from main rotor $G_v(\omega_v, \Omega_v)$;

G_h - aerodynamical dumping torque from tail rotor $G_h(\omega_h, \Omega_h)$.

R_h - moment of centrifugal force $R_h(\alpha_v, \Omega_h)$.

Controlling the system consists in stabilising the TRAS beam in an arbitrary (within practical limits) desired position (pitch and azimuth) or making it track a desired trajectory. Both goals may be achieved by means of appropriately chosen controllers. The user can select between two types of PID controllers and a state feedback controller (see section 6).

4.2 Non-linear model

Available after purchasing the system.

4.3 State equations

Available after purchasing the system.

4.4 Static characteristics

It is necessary to identify the following functions:

- Two non-linear input characteristics determining dependence of the DC-motor rotational speed on the input voltage (RPM characteristics): $\omega_v = H_v(U_v)$, $\omega_h = H_h(U_h)$

To measure the characteristics double click the *Static characteristics* button in *TRAS Control Window.* The window given in Fig. 4.3 opens. In this window one defines the minimal and maximal control values and a number of measured points. The control order can be set as: *Ascending, Descending* or *Reverse*. Also one can choose the pitch or azimuth static characteristic. Note, that the control signal is normalised and changes in the range

[-1, +1] what corresponding to the input voltage range *[-24V, +24V]* .

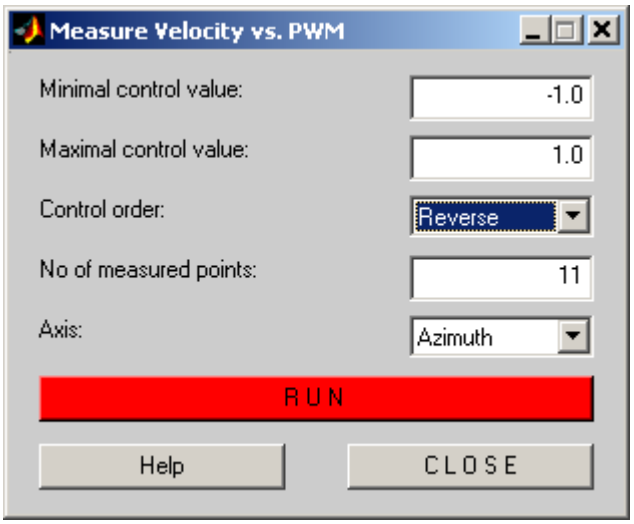


Fig. 4.3 Parameters of measurement of static characteristics

Choose *Azimuth* axis (tail rotor) and click the *Run* button. The constant value of control activates the DC motor so long as is required to obtain a steady state of the shaft angular velocity. Then, the velocity is measured and the control value is changed to the next constant value and DC motor is activated again. These steps are repeated to the end of the control range. This action should be repeated for pitch axis (main rotor) to obtained the both characteristics. Examples of the measured static characteristic for the main and tail rotors are shown in Fig. 4.4.

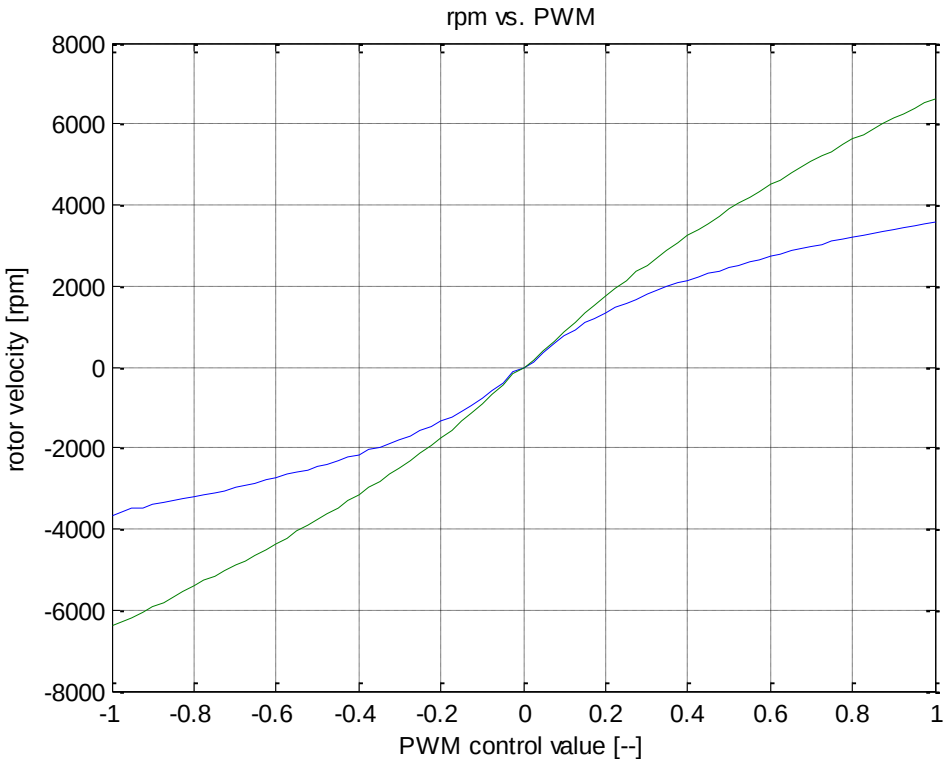


Fig. 4.4. Main and tail rotor static characteristics

If the characteristics is measured in *Reverse* mode (the control has been changed from –1 to +1 and reverse), there are two slightly different plots.

- Two non-linear characteristics determining dependence of the propeller thrust on DC- motor rotational speed (thrust characteristics): $F_h=F_h(\omega_h)$, $F_v=F_v(\omega_v)$.

The thrust static characteristics of the propellers should be measured in the case when the propellers were changed by a user. In this case a proper electronic balance (no delivered with the system) is necessary to measure the force created by rotational movements of the propellers. The characteristics included in the *TRAS Toolbox* and shown in this section were obtained by the manufacturer of the TRAS.

4.4.1 Main rotor thrust characteristics

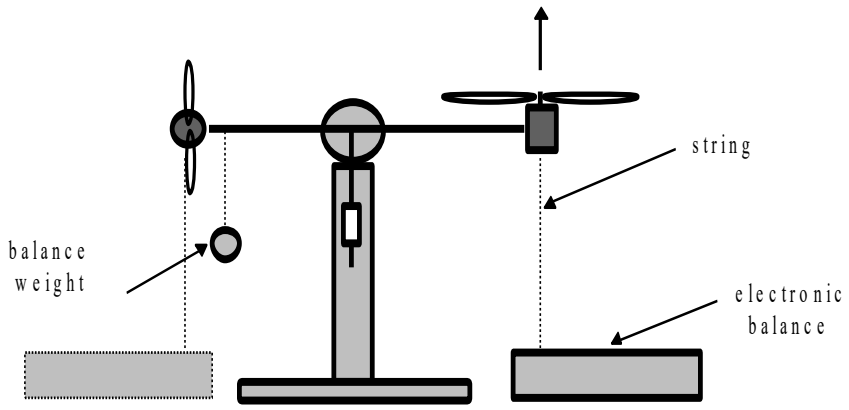


Fig. 4.5 Measuring of the main rotor thrust characteristics

To perform measurements correctly block the beam so that it could not rotate around the vertical axis, place the electronic balance under the beam in such a way that it is pulled by the propeller straight up. To balance the beam in the horizontal position attach a weight to the beam (as in Fig. 4.5) .

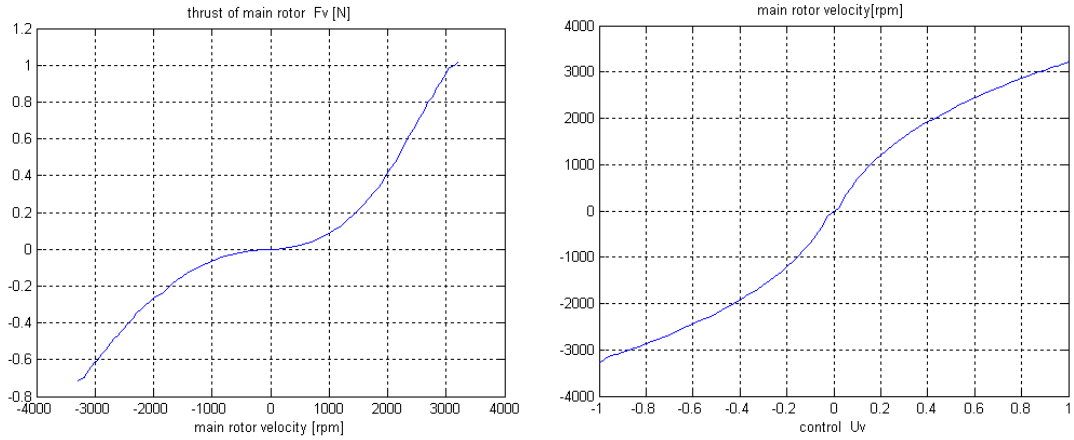


Fig. 4.6 Measured static thrust characteristics of the main rotor

For further applications the measured characteristics should be replaced by their polynomial approximations. For this purposes one can use the MATLAB ***polyfit.m*** function. An example is given in Fig.3.6. The obtained polynomials have the form:

$$\tilde{F}_v = -1.8\cdot10^{-18}\,\omega_v^5 - 7.8\cdot10^{-16}\,\omega_v^4 + 4.1\cdot10^{-11}\,\omega_v^3 + 2.7\cdot10^{-8}\,\omega_v^2 + 3.5\cdot10^{-5}\,\omega_v - 0.014$$

$$\tilde{\omega}_v = -5.2\cdot10^{3}\,U_v^7 - 1.1\cdot10^{2}\,U_v^6 + 1.1\cdot10^{4}\,U_v^5 + 1.3\cdot10^{2}\,U_v^4 - 9.2\cdot10^{3}\,U_v^3 - 31U_v^2 + 6.1\cdot10^{3}\,U_v - 4.5$$

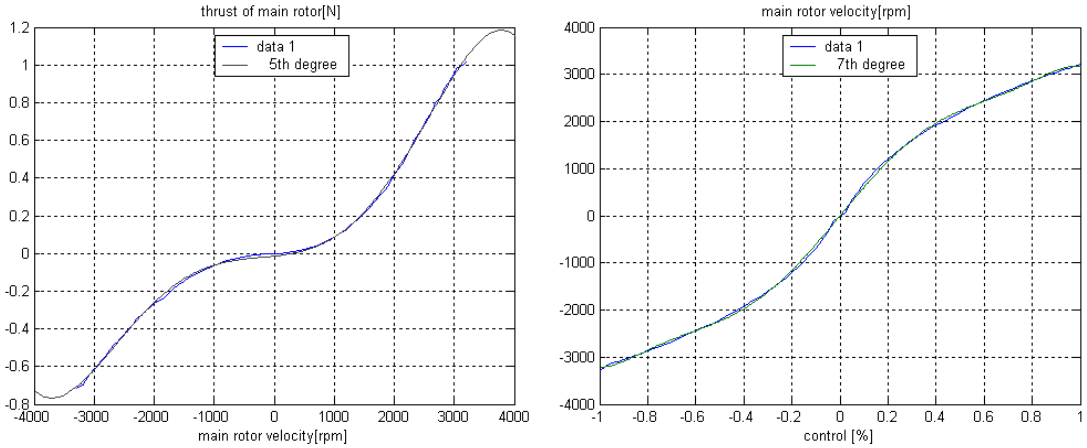


Fig. 4.7 Polynomial approximation of the main rotor characteristics

4.4.2 Tail rotor thrust characteristics

Fig. 4.8 shows laboratory set-up for measuring thrust of the tail rotor.

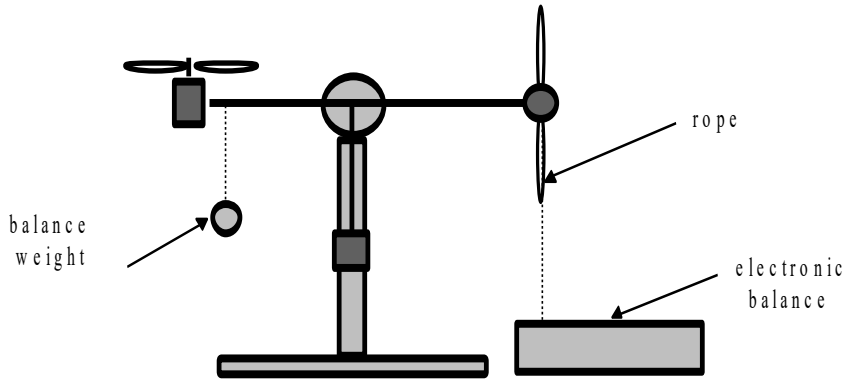


Fig. 4.8 Laboratory set-up for the tail rotor thrust characteristics

To measure the static thrust characteristics one should to rearrange the laboratory set-up as shown in Fig. 4.8 and the electronic balance should be used.

The measured by the producer of the TRAS thrust static characteristics of the tail motor are given in Fig. 4.9.

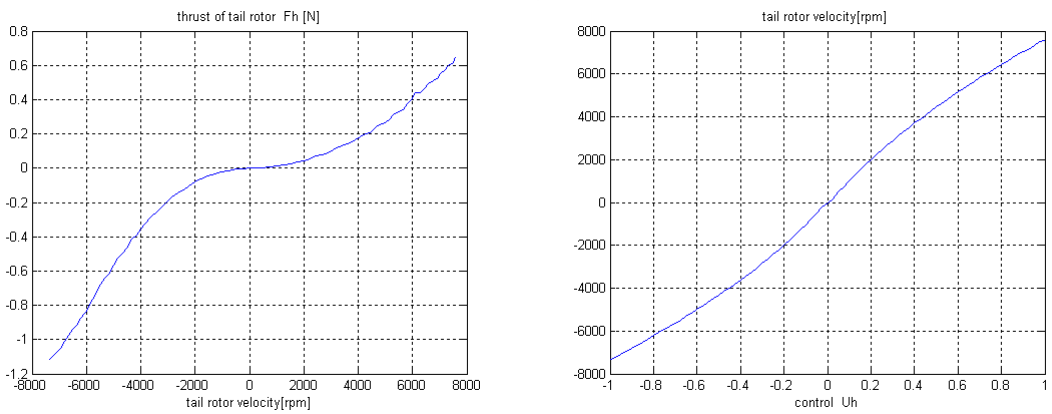


Fig. 4.9 Thrust characteristics measured for the tail rotor.

For further applications the characteristics can be replaced by their polynomial approximations. For this purposes one can use the MATLAB ***polyfit.m*** function. The obtained polynomials are as follows:

$$\tilde{F}_h = -2.6\cdot10^{-20}\omega_h^5 + 4.1\cdot10^{-17}\omega_h^4 + 3.2\cdot10^{-12}\omega_h^3 - 7.3\cdot10^{-9}\omega_h^2 + 2.1\cdot10^{-5}\omega_v + 0.0091$$

$$\tilde{\omega}_h = 2.2\cdot10^{3}U_h^5 - 1.7\cdot10^{2}U_v^4 - 4.5\cdot10^{3}U_v^3 + 3\cdot10^{2}U_v^2 + 9.8\cdot10^{3}U_v - 9.2$$

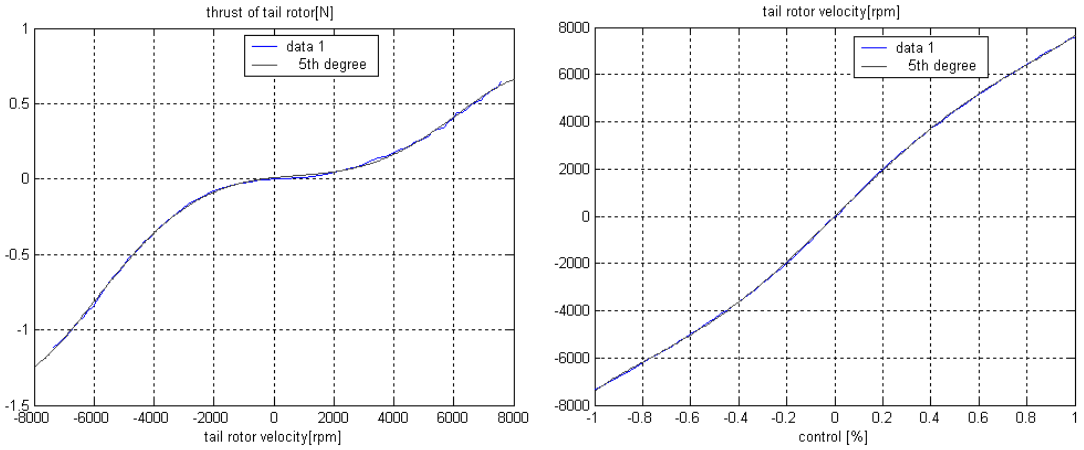


Fig. 4.10 Polynomial approximation of tail rotor characteristics

5. Real-time model

In this section the process of building your own control system is described. The *RT-CON* toolbox is used. An example how to use the TRAS software is shown later in section 5.3. In this section we give indications how to proceed in the real-time 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 (if you are using Visual C++ compiler)

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\R2xxxx\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.

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

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

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

- create a Simulink model of the control system which consists of *TRAS Device Driver* and other blocks chosen from the Simulink library,
- build the executable file under RT-CON (see Fig. 5.1),
- start the real-time code from the *Simulation/Start real-time code* pull-down menus; in this way the system runs in the real-time.

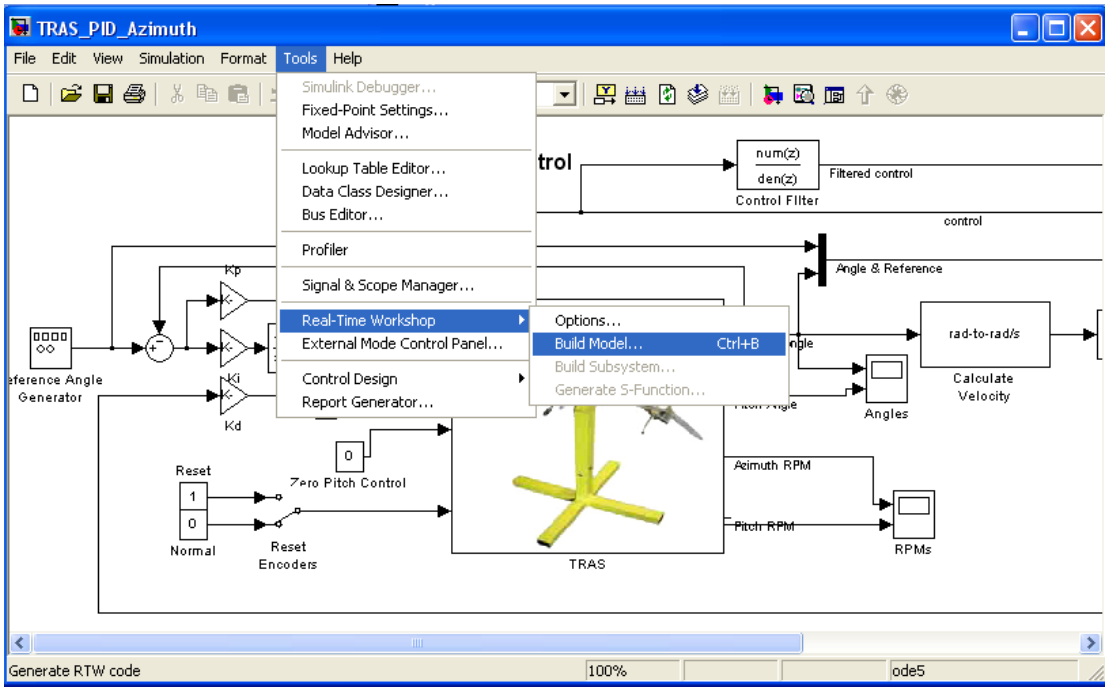


Fig. 5.1. Creating the real-time code using RTW and RT-CON

5.2 Creating a model

The simplest way to create a Simulink model of the control system is to use one of the models included in *Tras Control Window* as a template. For example, click on the *PID Azimuth* button and save it as *MySystem.mdl* name. The *MySystem* Simulink model is shown in Fig. 5.2.

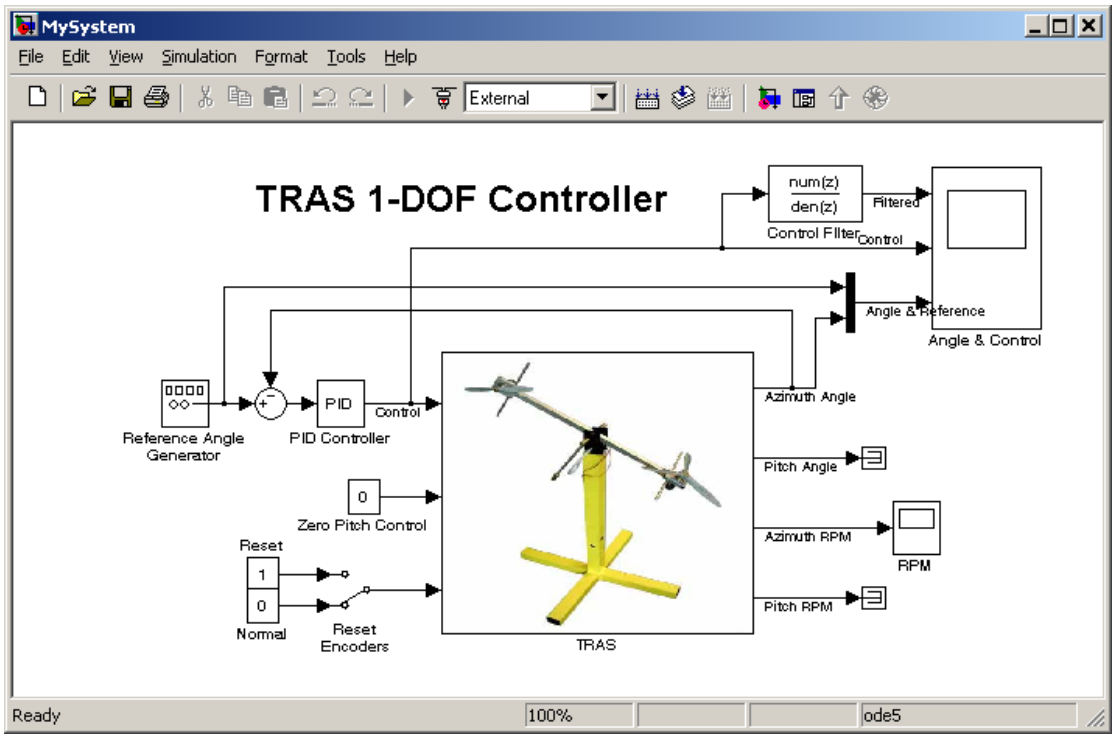


Fig. 5.2 The *MySystem* Simulink model

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

Though it is not obligatory, we recommend you to leave the scope. You need a scope to watch how the system runs. The saturation blocks are built in the *Tras driver* block. They limit the currents to the DC motors for safety reasons. However they are not visible for the user who may complain that the controls saturate despite it would not been predicted. 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 *Tras Device Driver* into the real-time code a special make-file is required. This file is included to the TRAS software.

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

The scope block properties are important for an appropriate data acquisition and watching how the system runs.

The *Scope* block properties are defined in the Scope property window (see Fig. 5.3). 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 select 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. In this case the time space of the plot is equal to 0.01 [s]. Remember mark the *Limit data points to last* checkbox.

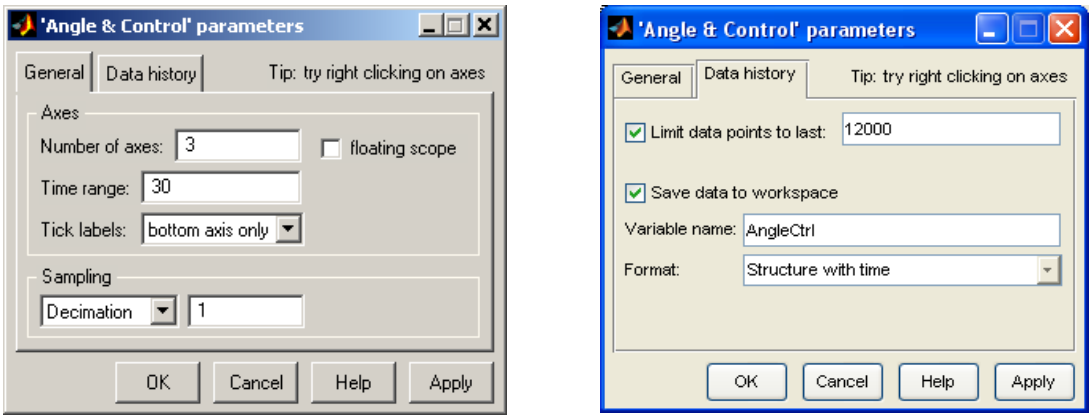


Fig. 5.3 Setting the parameters of the *Scope* block

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. 5.4 opens. Select *Select All* check button, set *Source* as manual, set *Duration* equal to the number of samples you intend to collect, and close the window.

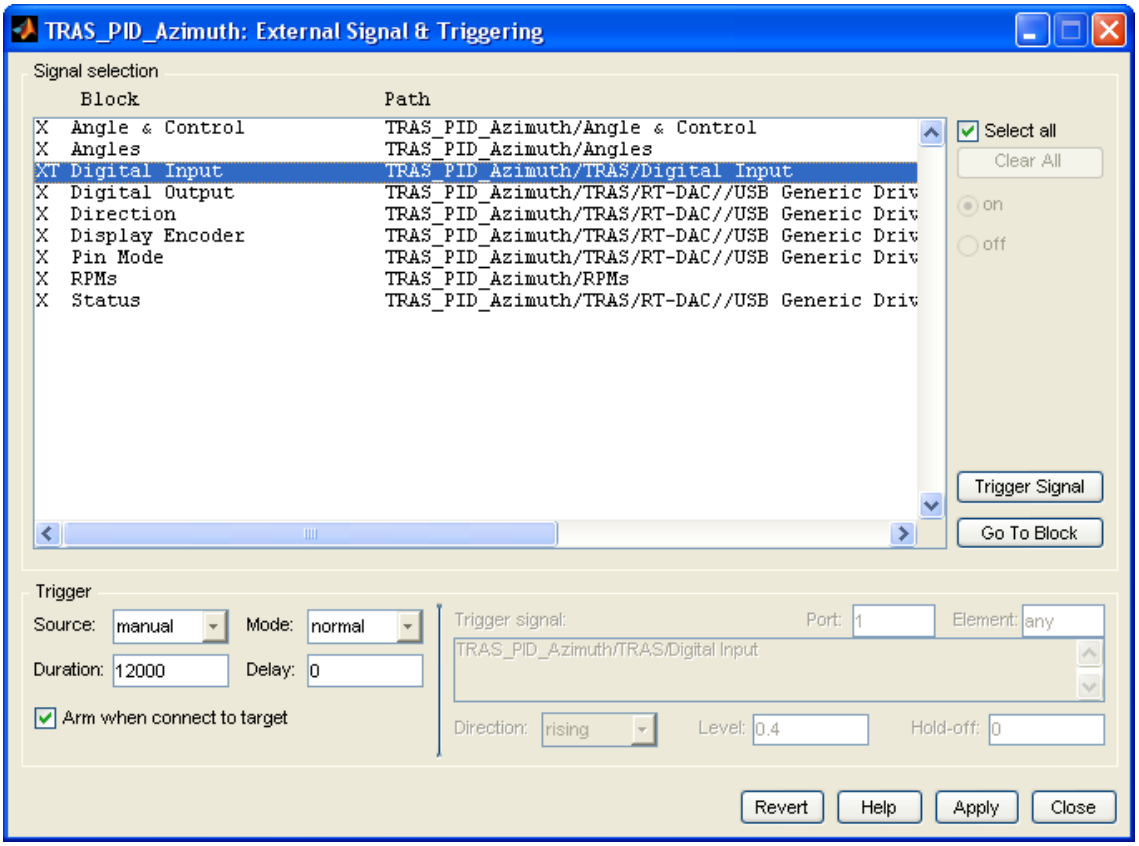


Fig. 5.4 The *External Signal & Triggering* window

5.3 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 Simulink Target Language). The make-file is used to build and download object files to the target hardware automatically.

At the beginning 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. 5.5).

The *RTW* page is used to set the real-time build options and then to start the building process of the *RTW.DLL* executable file.

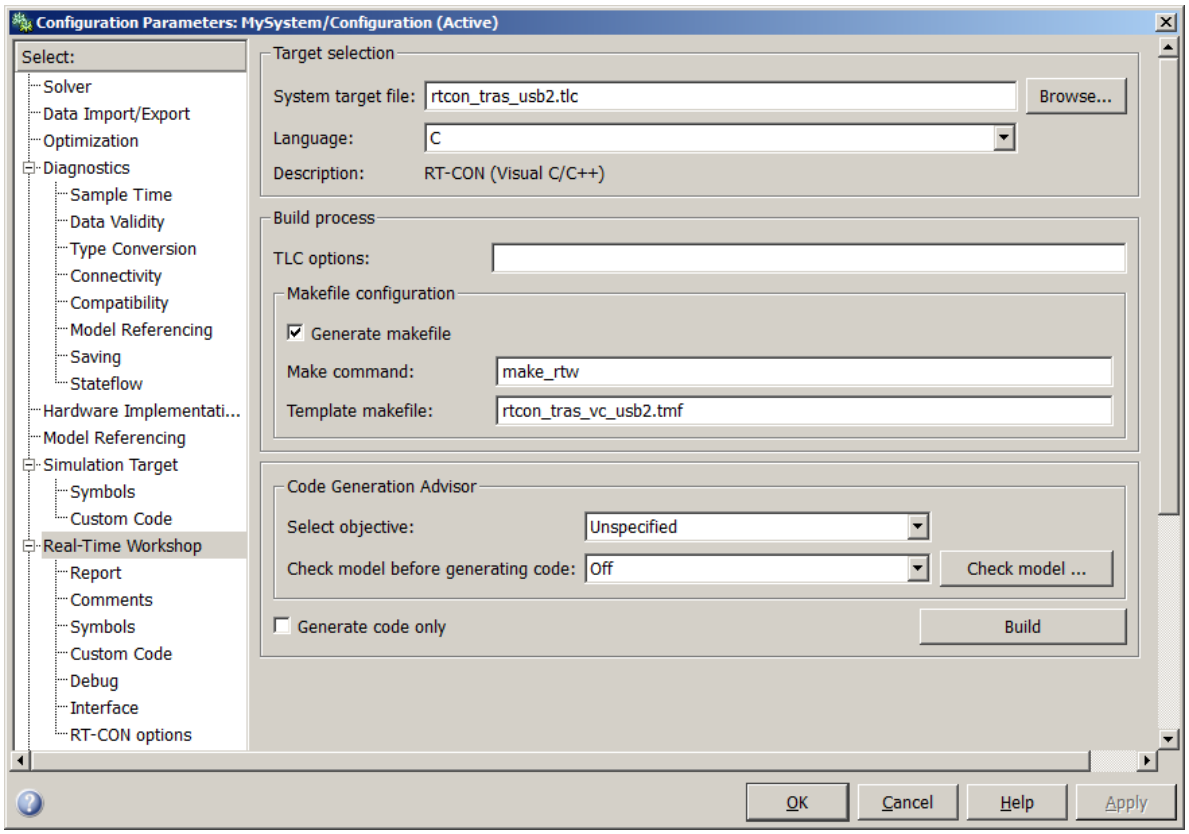


Fig. 5.5. *Real Time Worshop* tab in the *Configuration Parameters* page of the *Simulation parameters* option

The system target file name is *rtcon_tras_USB2.tlc*. It manages the code generation process. The *rtcon_tras_vc_us2b.tmf* template make-file is devoted to C code generation using the Visual C++ compiler.

Click *Interface* tab and next dialog page shown in Fig. 5.6 opens Note that *External mode* is set and *Transport layer* is *RT-CON tcpip*. These options have to be set in the RTW and RT-CON environment.

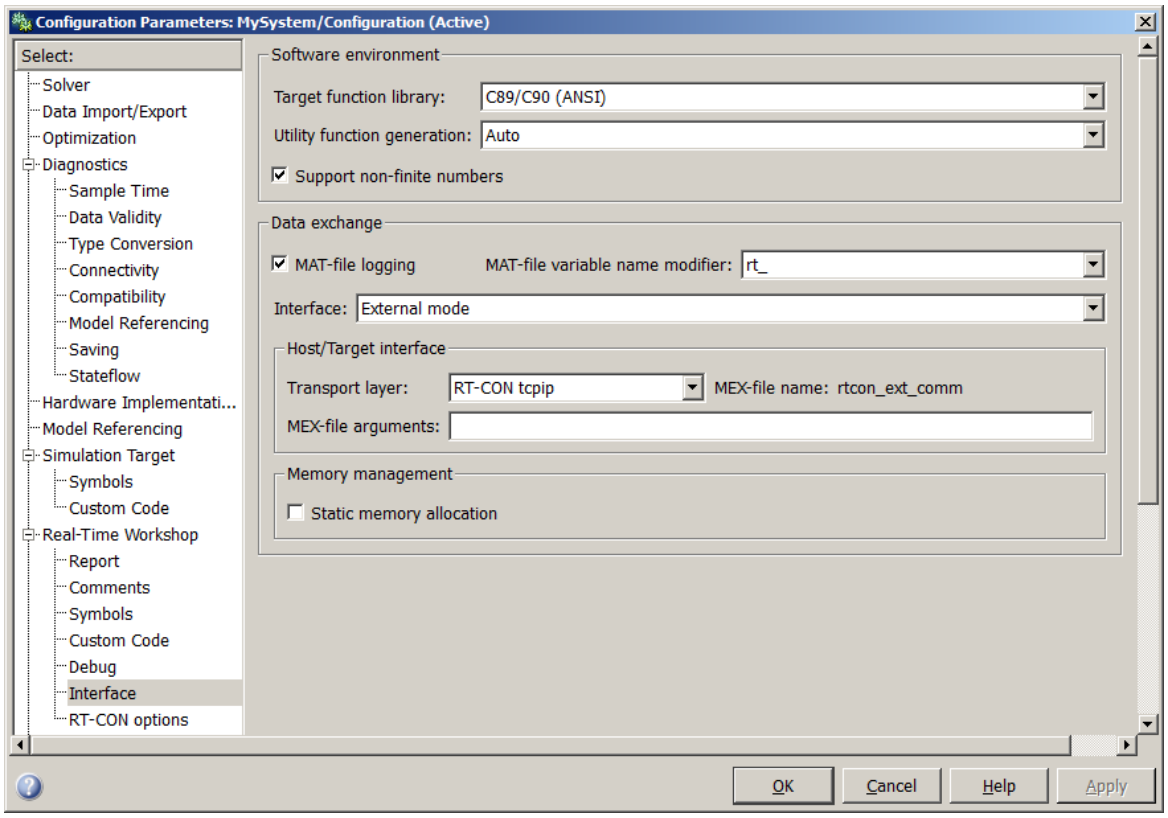


Fig. 5.6 The *Interface* tab in the *Configuration parameters* dialog box

The *Solver* page appears when you select the *Solver* tab (Fig. 5.7). The *Solver* page is used to set the simulation parameters. Several parameters and options are available in the window. The *Fixed-step size* editable text box is set to 0.002 (this is the sampling period given 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.

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.

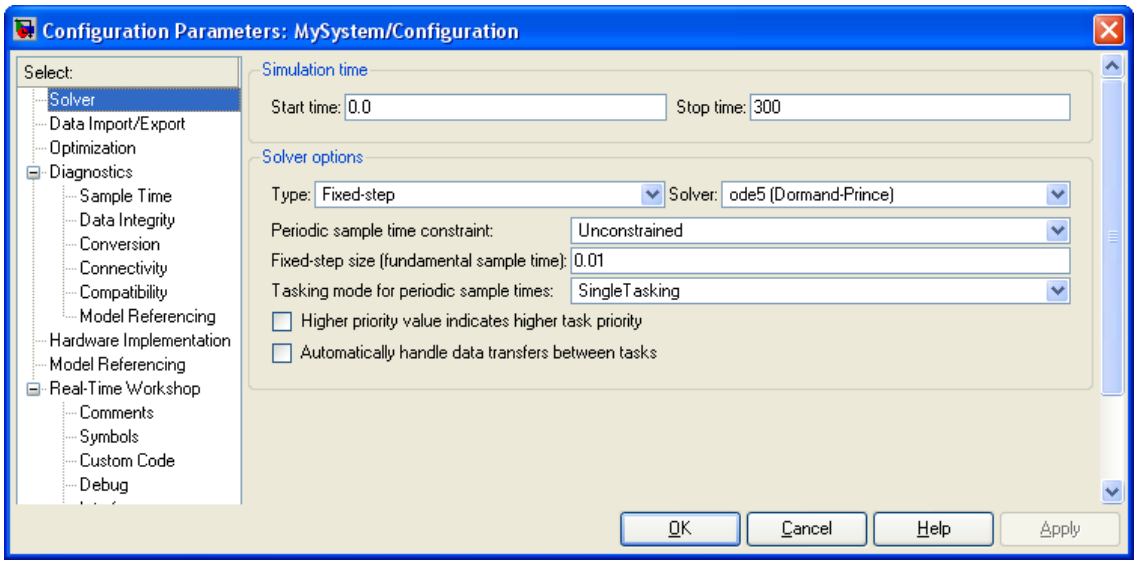


Fig. 5.7 Simulation parameters

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) or

.

Successful compilation and linking processes generate the following message:

Created Real-Time Windows Target module MySystem.rwd.
Successful completion of Real-Time Workshop build procedure for model: MySystem

Otherwise, an error massage is displayed in the MATLAB command window.

6. 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. 5.1. 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.

6.2 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. 5.1.

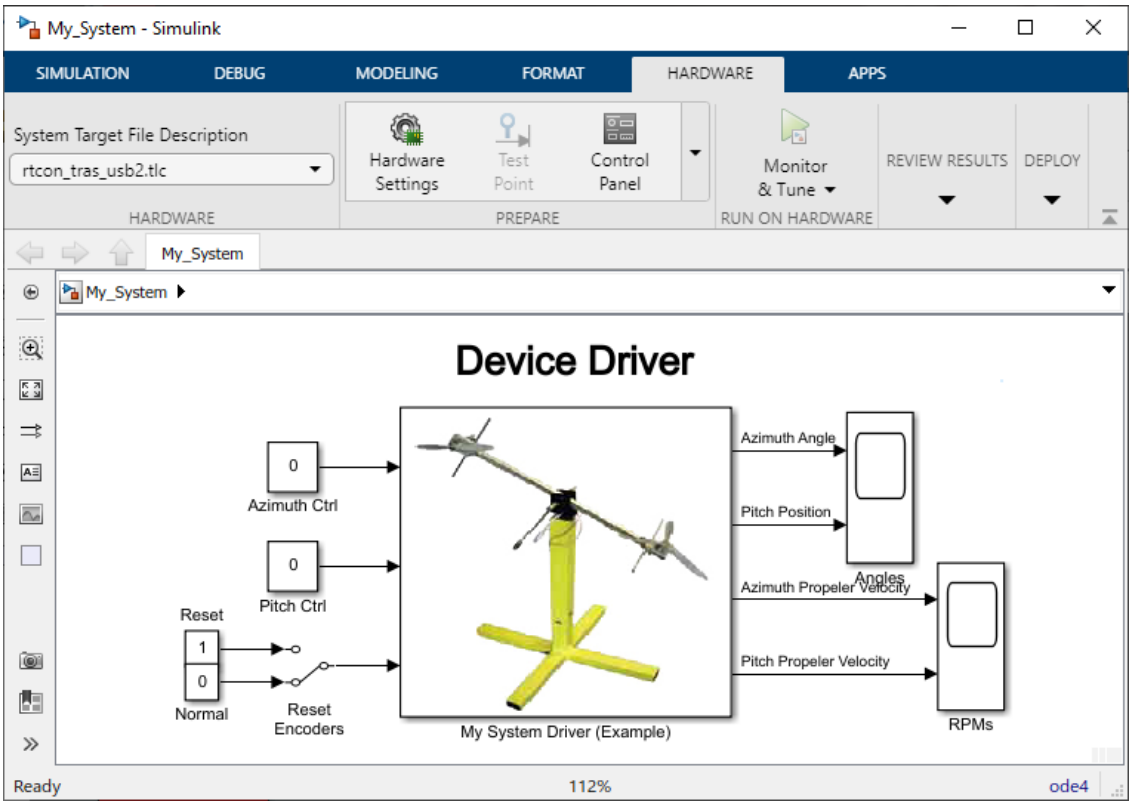


Fig. 6.1. My_System real-time model

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. 5.4 and Fig. 6.3. 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. 5.4 opens. Select *Interface* button to see the external mode options at Fig. 6.3. The system target file name is *rtcon_tras_USB2.tlc*. It manages the code generation process.

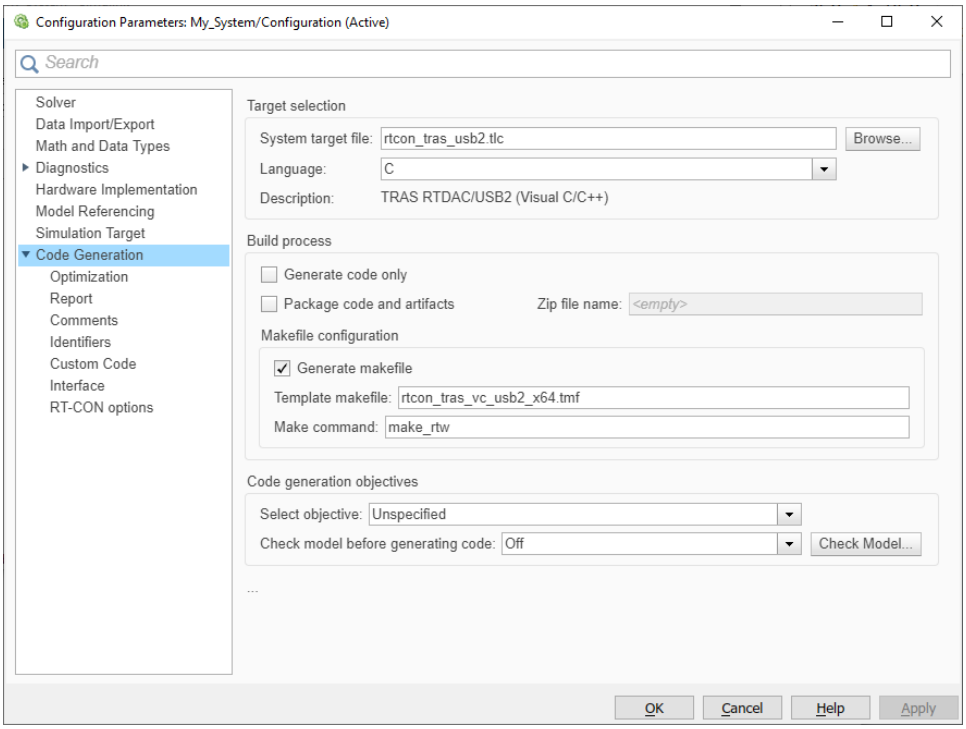


Fig. 6.2 Internal code generation options

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.

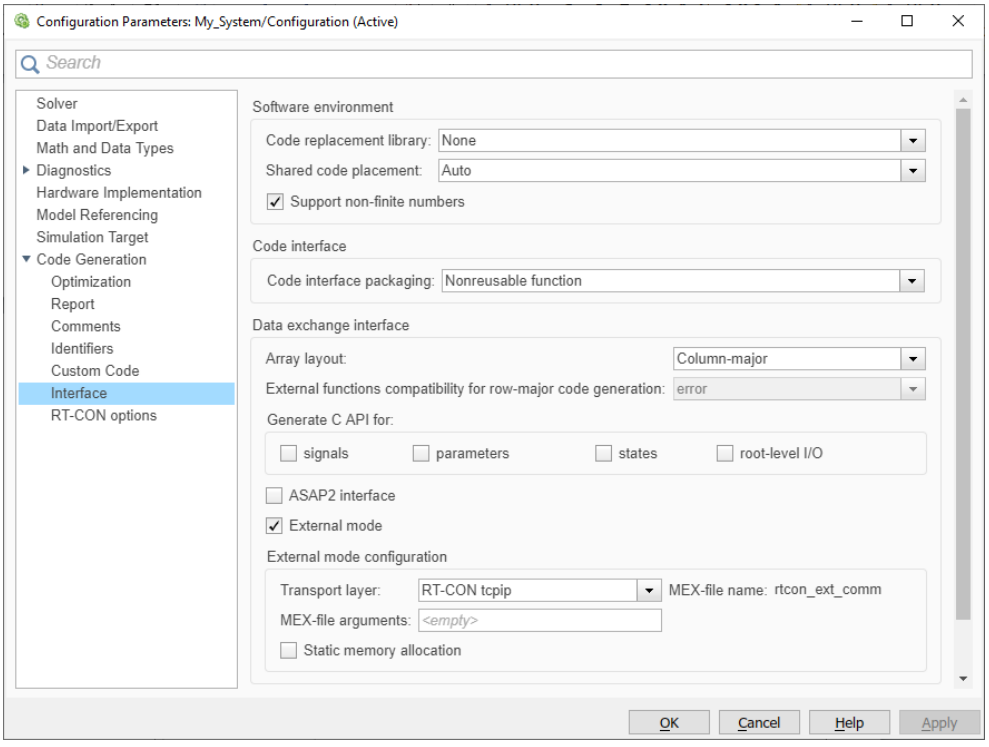


Fig. 6.3. Interface of the external mode options

6.3 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. 5.5). You can also use the keyboard by clicking CTRL+b keys. You will get the same effect in this way.

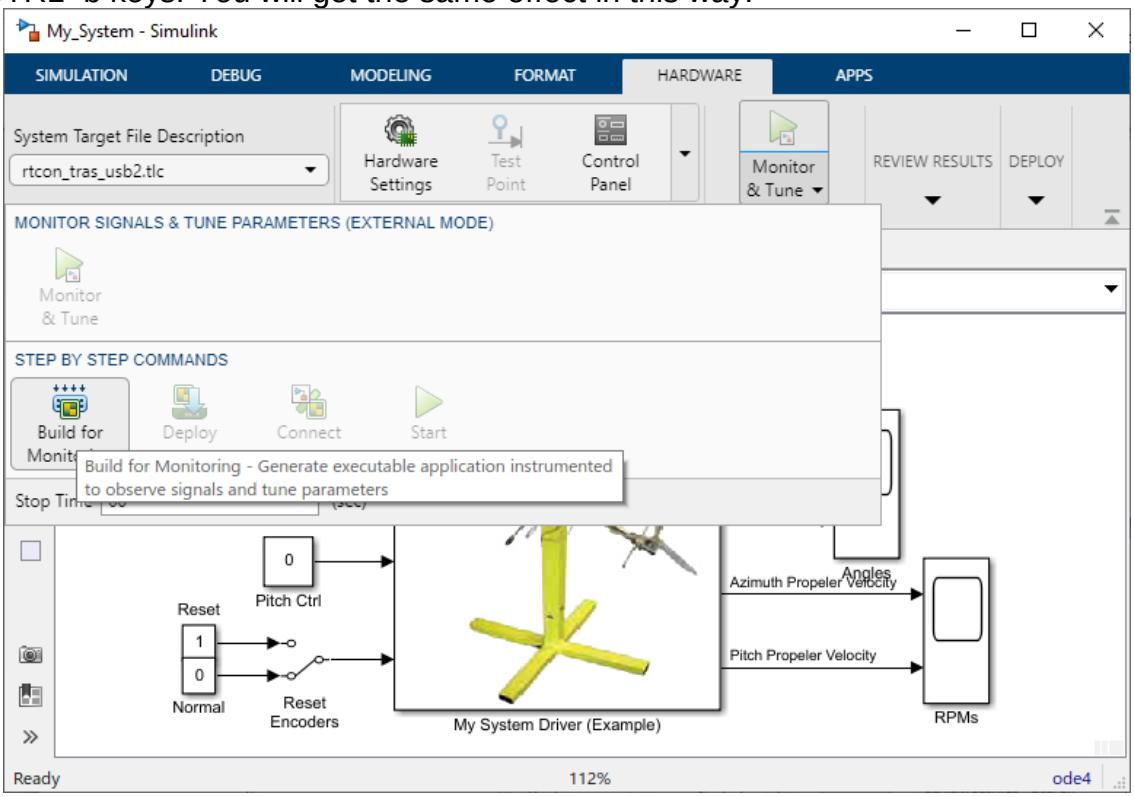


Fig. 6.4. Building model

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. 5.7 will appear. Next click *Connect* button and real-time starts.

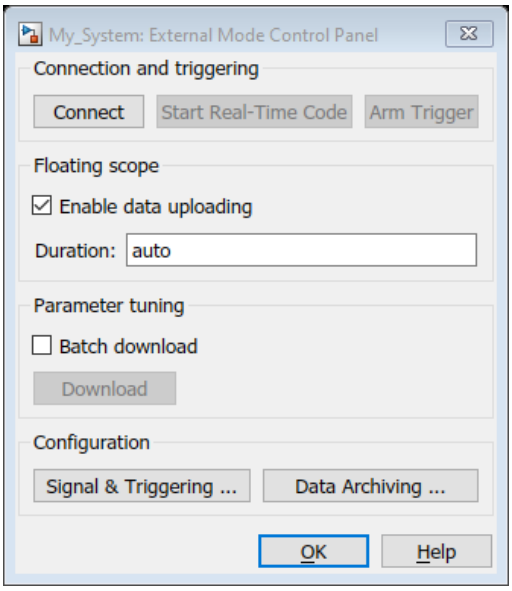


Fig. 6.5 Connect with target

Do not use option *Build Stand-Alone* shown in Fig. 6.6 for compilation .

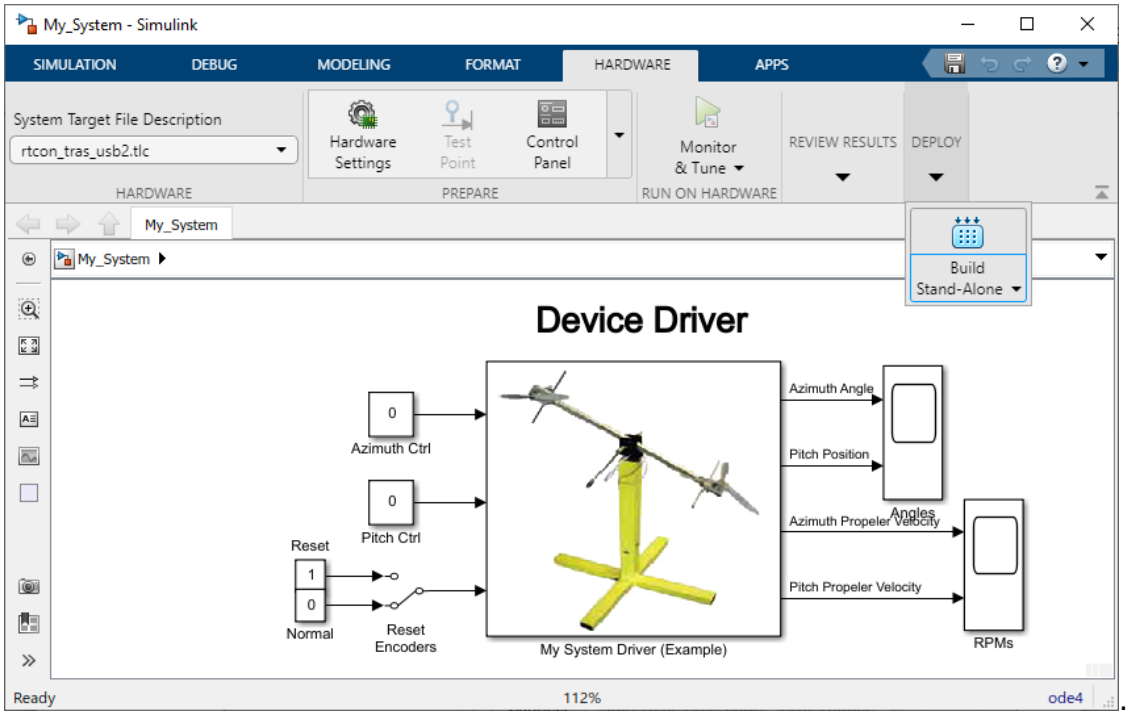


Fig. 6.6 Do not use this option !!

7. Controllers and real-time experiments

In the following section we propose three PID controllers. It is possible to tune the parameters of the controllers without analytical design. Such approach to the control problem seems to be reasonable if a well identified model of the TRAS is not available. The effectiveness of the PID controllers discussed here is illustrated by control experiments.

The experiments and corresponding to them 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 these given in the manual.

7.2 PID controllers

One degree of freedom (1-DOF) control problem is the following. Design a controller that will stabilise the system, or make it follow a desired trajectory in one plane (one degree of freedom) while motion in the other plane is blocked mechanically or being controlled by another controller.

If TRAS is free to move in both axes we refer to the control as two degree of freedom (2-DOF). The four PID controllers for TRAS: PID_{vv}, PID_{vh}, PID_{hv} and PID_{hh} (h-horizontal (azimuth), v-vertical (pitch)) are considered. The subscripts indicates the source-sink relation for the controller. Each control signal (U_v and U_h) is the sum of two controller outputs. For example, vertical control denoted later as U_v is the sum of two output signals: PID_{vv} and PID_{hv}. The internal structure of each PID controller is shown in Fig. 7.13b. There are three parameters to be set for every controller: K_P, K_i and K_d. The TRAS control in the vertical and horizontal planes requires setting altogether 12 (3×4) controller parameters. Saturation blocks introduce four additional I_{sat} parameters: I_{vvsat}, I_{vhsat}, I_{hhsat} and I_{hvsat}, which are the limits of absolute values of the integrals of errors, and two: U_{hmax} and U_{vmax} parameters, which are the limits of absolute value of controls. These 18 (12+4+2) parameters have their default values.

7.3 1-DOF controllers

The task of the one-degree-of-freedom (1-DOF) controllers is to move the TRAS to an arbitrary position in the selected plane and to stabilise it there.

7.3.1 Vertical 1-DOF control

At the beginning we restrict our control objective to stabilising the system in the vertical plane only (using the included clamp). We reduce the original system

to the 1-DOF system by mechanically blocking its freedom to move in the horizontal plane. A corresponding block diagram of the PID control system is presented in Fig. 7.1.

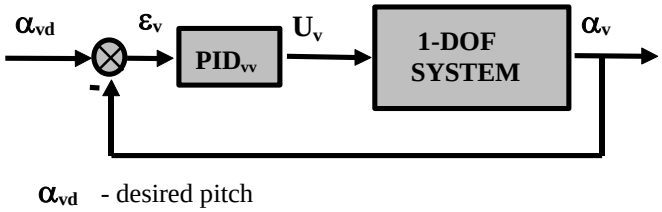


Fig. 7.1 1-DOF pitch control system

The block diagram below shows the system in a more detailed form (Fig. 7.2). Notice, that only the vertical part of the control system is considered.

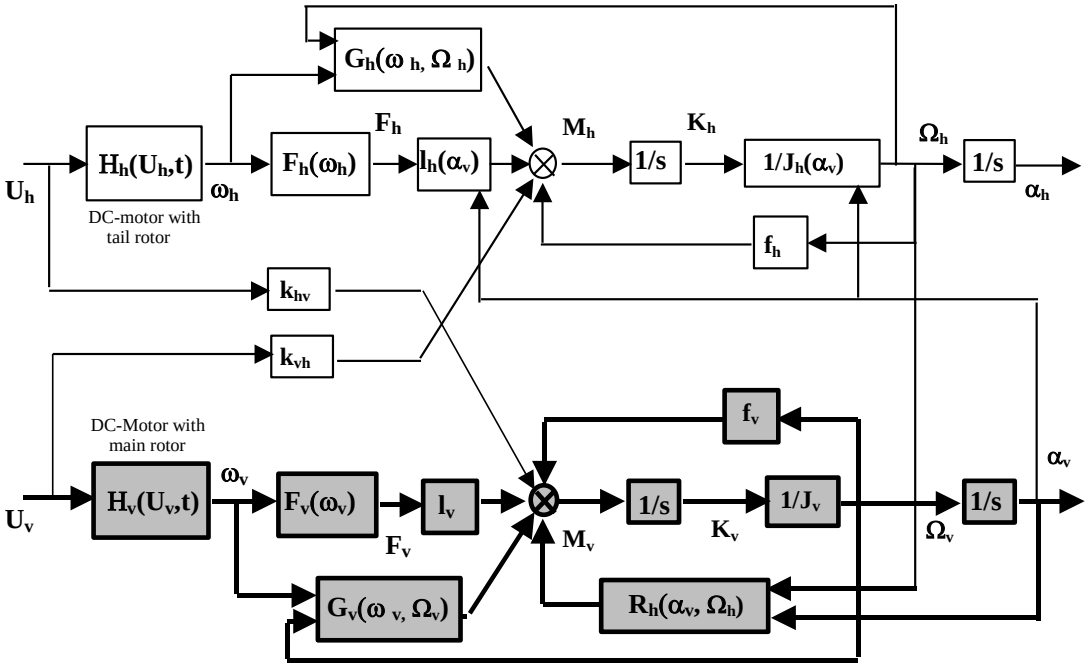


Fig. 7.2 The block diagram 1-DOF system (vertical plane)

7.3.2 Real-time 1-DOF pitch control experiment

Fix the TRAS device in the horizontal plane using the special plastic clamps delivered with TRAS. Set it in the neutral vertical position and wait until the all oscillations are finished. In the *Tras Control Window* double click the *Reset Encoders* block.

Click the *PID Pitch controller* button and the model shown in Fig. 7.3 opens. Set all PID controller coefficients as: $K_p = 0.0339$ $K_i = 0.3532$ and $K_d = 0.7918$. Also set saturation of the integral part of the controller to 1.1. The reference signal choose *square* with amplitude equal to 0.2 [rad] and frequency set 1/60 [Hz]. Build the

model and click on the *Simulation*/*Connect to target* option and *Start real-time code* option.

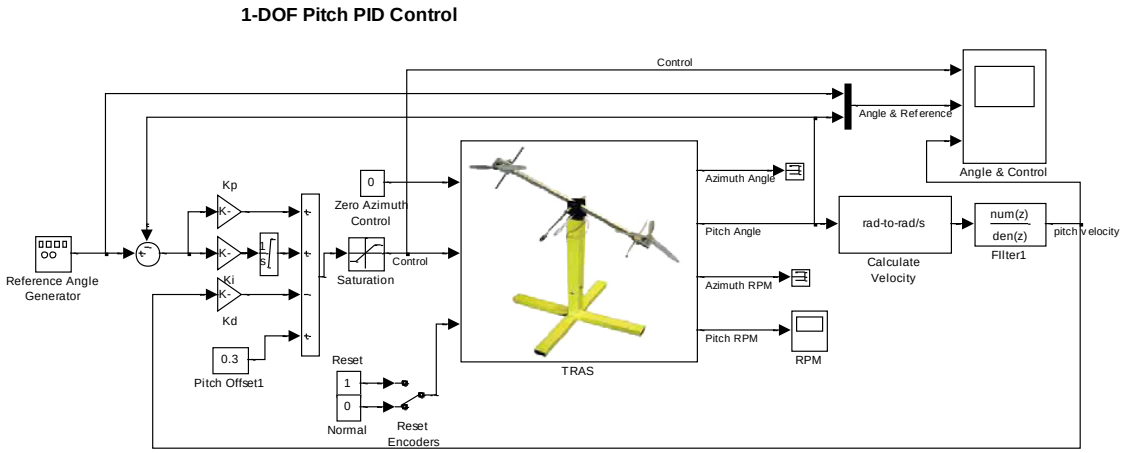


Fig. 7.3 Real-time model for pitch control

The results of the experiment are shown in Fig. 7.4.

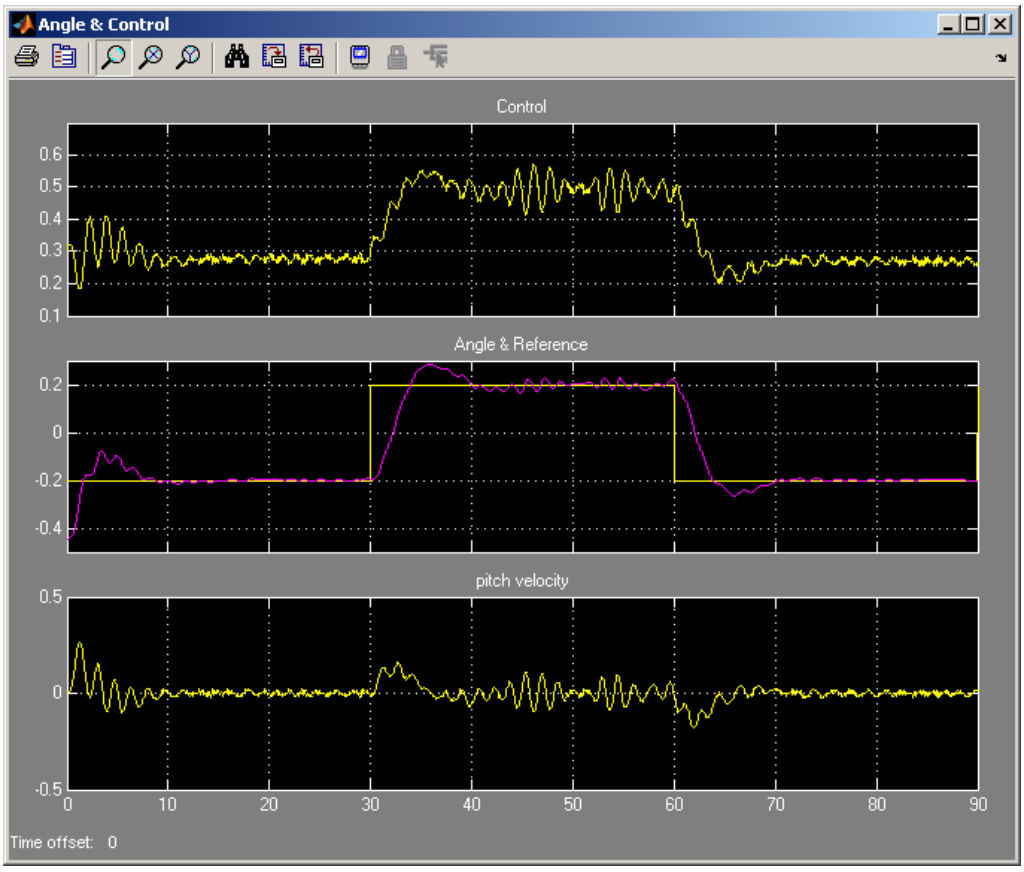


Fig. 7.4 Results of PID pitch control

The details of the above experiment are shown in Fig. 7.5 and Fig. 7.6.

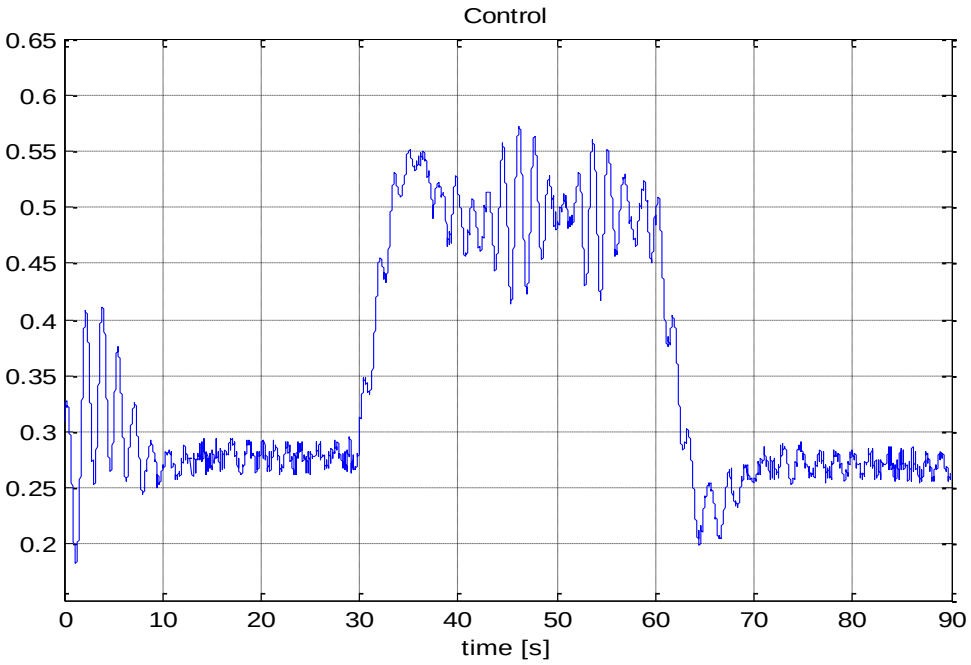


Fig. 7.5 The control

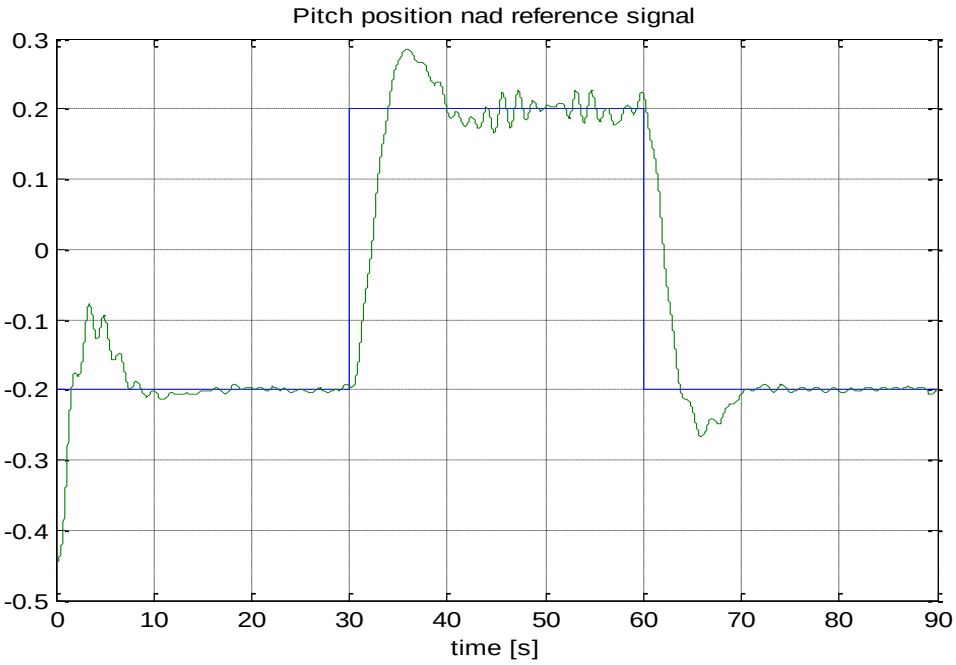


Fig. 7.6 The pitch position and reference signal

7.3.3 Horizontal 1-DOF control

In the next experiment we will apply stabilising PID controller in the horizontal plane. We block the system in one axis so that it cannot move in the vertical plane (using the included fixing rectangle).. A corresponding block diagram of the control system is shown in Fig. 7.7, and in a more detailed form in Fig. 7.8.

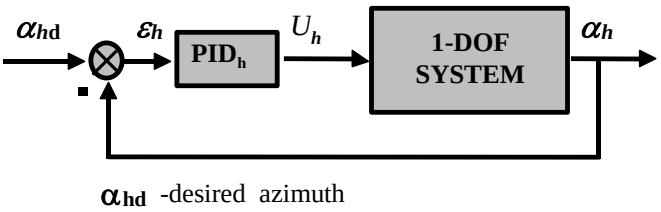


Fig. 7.7 1-DOF control closed-loop system (azimuth stabilisation)

Notice that only the 'horizontal' part of the control system is considered.

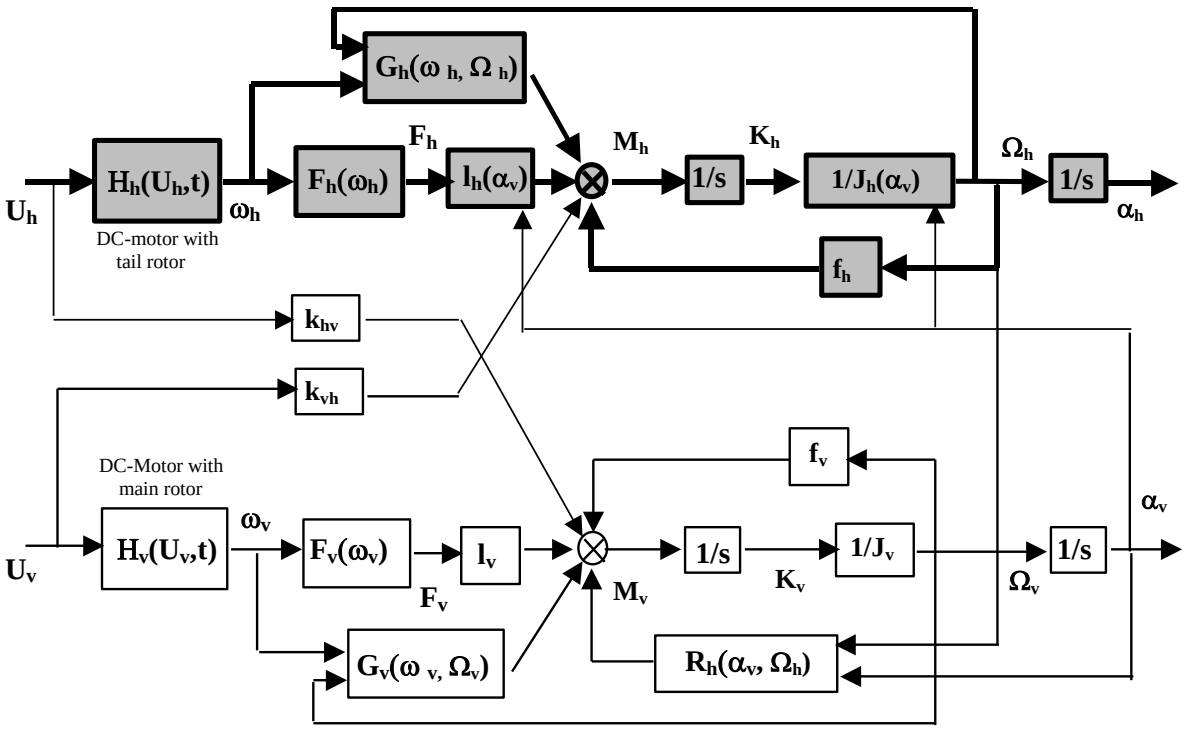


Fig. 7.8 The block diagram of 1-DOF system (horizontal plane)

7.3.4 Real-time 1-DOF azimuth control experiment

Fix the TRAS device in the vertical plane using the special fixing rectangle delivered with TRAS. Set it in the zero position and click on the *Reset Encoders* block in *Tras Control Window*.

Click *PID Azimuth controller* and the model shown in Fig. 7.9 opens. Set all PID controller coefficients as $K_p = 1.9758$ $K_i = 0.0115$ and $K_d = 1.4008$. Build the

model and click on the *Simulation*/*Connect to target* and *Start real-time code* options.

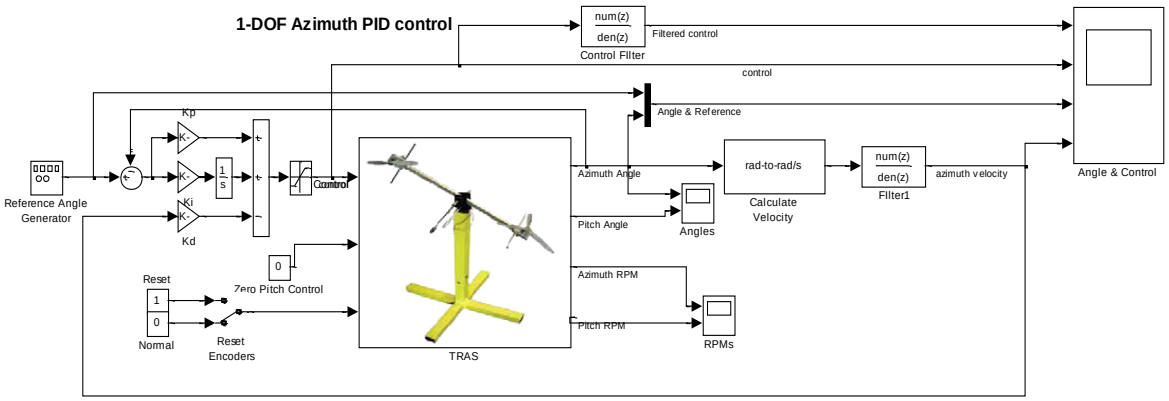


Fig. 7.9 Real-time model for PID azimuth control

The results of the experiment are shown in Fig. 7.10. Notice, that control similar to the pitch control changes with a high frequency..

Fig. 7.10 Results of PID azimuth control

The details of the above experiment are shown in Fig. 7.11 and Fig. 7.12.

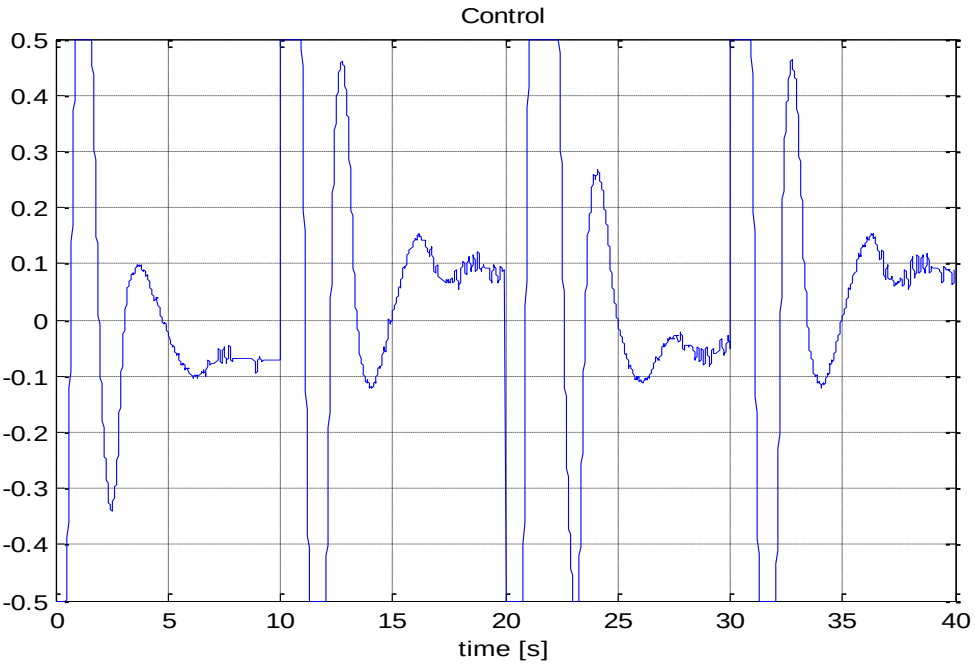


Fig. 7.11 Control

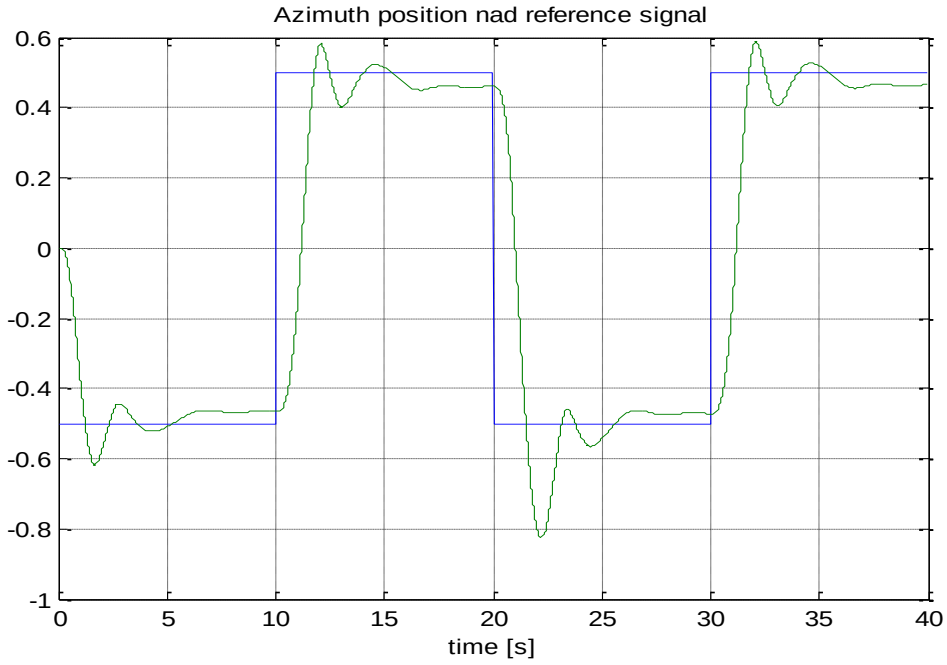


Fig. 7.12 The azimuth position and reference signal

7.4 2-DOF PID controller

The structure of the cross-coupled multivariable PID controller is shown in Fig. 7.13.

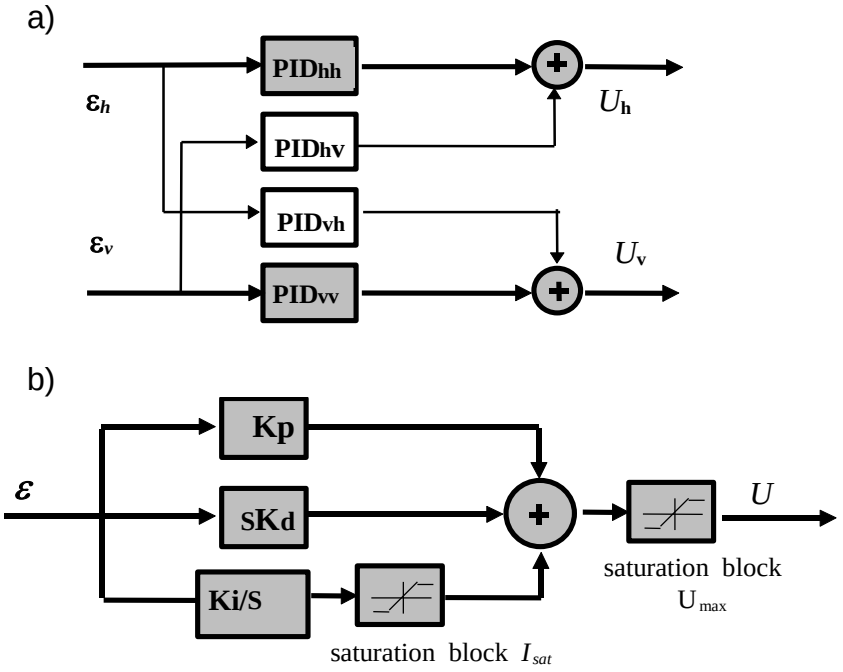


Fig. 7.13 Structure of the cross-coupled PID controller
a) general b) single PID block

The controller is described by the equations given bellow.

$\varepsilon_v = \alpha_{vd} - \alpha_v$,

$\varepsilon_h = \alpha_{hd} - \alpha_h$,

where: $\varepsilon_v, \varepsilon_h$ are errors of vertical (pitch) and horizontal angle (azimuth), α_{vd}, α_{hd} are reference values of vertical and horizontal angles, α_v, α_h are vertical and horizontal angles.

The integrators are described by the following equations:

$$I_{vv}(t) = K_{ivv}\int_0^t \varepsilon_v dt, \qquad for - I_{vvsat} \le I_{vv} \le I_{vvsat}$$

$$if\,(I_{vv} > I_{vvsat})\,then\,I_{vv} = I_{vvsat},\ if\,(I_{vv} < -I_{vvsat})\,then\,I_{vv} = -I_{vvsat},$$

$$I_{vh}(t) = K_{ivh}\int_0^t \varepsilon_v dt, \qquad for - I_{vhsat} \le I_{vh} \le I_{vhsat}$$

$$if\,(I_{vh} > I_{vhsat})\,then\,I_{vh} = I_{vhsat},\ if\,(I_{vh} < -I_{vhsat})\,then\,I_{vh} = -I_{vhsat},$$

$$I_{hv}(t) = K_{ihv}\int_0^t \varepsilon_h dt, \qquad for - I_{hvsat} \le I_{hv} \le I_{hvsat}$$

$$if\,(I_{hv} > I_{hvsat})\,then\,I_{hv} = I_{hvsat},\ if\,(I_{hv} < -I_{hvsat})\,then\,I_{hv} = -I_{hvsat},$$

$$I_{hh}(t) = K_{ihh}\int_0^t \varepsilon_h dt, \qquad for\ -I_{hhsat} \le I_{hh} \le I_{hhsat}$$

$$if\,(I_{hh} > I_{hhsat})\,then\,I_{hh} = I_{hhsat},\ if\,(I_{hh} < -I_{hhsat})\,then\,I_{hh} = -I_{hhsat},$$

where: $K_{ivv}, K_{ivh}, K_{ihv}, K_{ihh}$ are gains of the I parts,
$I_{vvsat}, I_{vhsat}, I_{hvsat}, I_{hhsat}$ are saturation's of the integrators.

Finally, vertical and horizontal controls are:

$$U_v = K_{pvv}\varepsilon_v + I_{vv}(t) + K_{dvv}\frac{d\varepsilon_v}{dt} + K_{pvh}\varepsilon_h + I_{vh}(t) + K_{dvh}\frac{d\varepsilon_h}{dt}, \quad \text{for } -U_{v\max} \le U_v \le U_{v\max}$$

$$if\,(U_v > U_{v\max})\,then\,U_v = U_{v\max},\ if\,(U_v < -U_{v\max})\,then\,U_v = -U_{v\max},$$

$$U_h = K_{phv}\varepsilon_h + I_{hv}(t) + K_{dhv}\frac{d\varepsilon_v}{dt} + K_{phh}\varepsilon_h + I_{hh}(t) + K_{dhh}\frac{d\varepsilon_h}{dt}, \quad \text{for } -U_{h\max} \le U_h \le U_{h\max}$$

$$if\,(U_h > U_{h\max})\,then\,U_h = U_{h\max},\ if\,(U_h < -U_{h\max})\,then\,U_h = -U_{h\max},$$

Where:
$K_{pvv}, K_{pvh}, K_{phv}, K_{phh}, K_{dvv}, K_{dvh}, K_{phv}, K_{phh}$ are parameters of the controllers,
$U_{v\max}, U_{h\max}$ are the saturation limits of the vertical and horizontal controls.

7.4.1 Simple PID controller

The simple PID controller controls the vertical and horizontal movements separately. In this control system influence of one rotor on the motion in the other plane is not compensated by the controller structure. The system is not de-coupled. The control system of this kind is presented in Fig. 7.14. The controller structure is presented in Fig. 7.15.

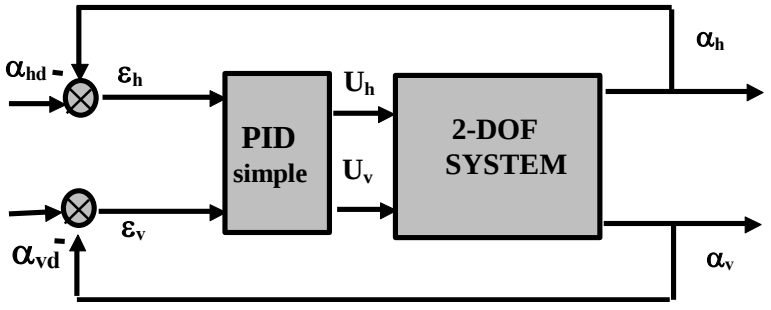


Fig. 7.14 The block diagram of 2-DOF control system with a simple PID-controller

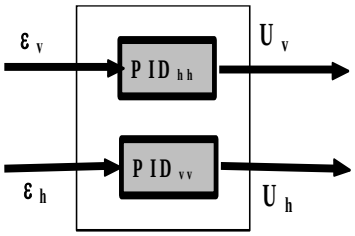


Fig. 7.15 The block diagram of the simple PID-controller

7.4.2 Cross-coupled PID controller

The cross-coupled PID controller controls the system in the pitch and azimuth planes. In this control system influence of one rotor on the motion in the other plane can be compensated by the cross-coupled structure of the controller. The control system is shown in Fig. 7.16. The cross-coupled PID controller structure is shown in Fig. 7.17.

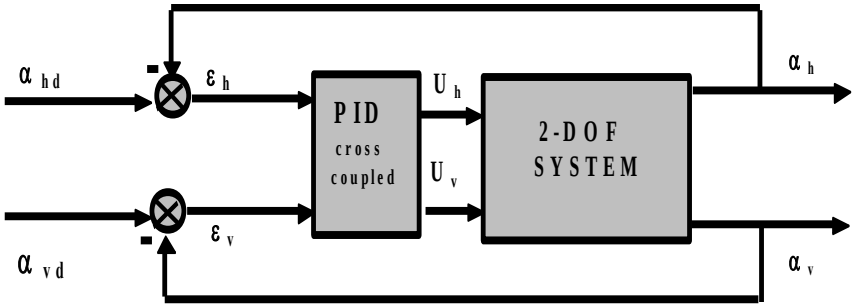


Fig. 7.16 The block diagram of the 2-DOF control system with the cross-coupled PID-controller

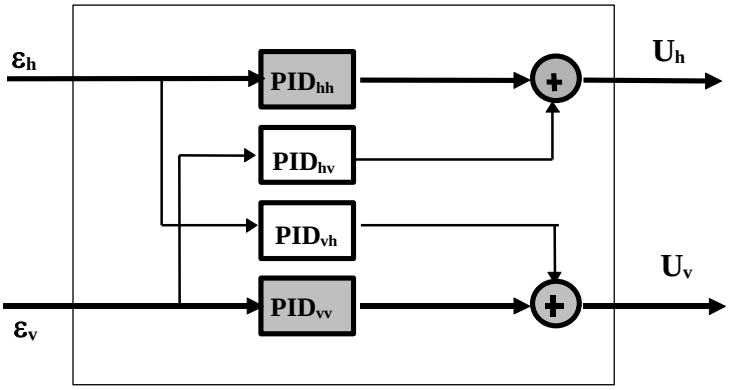


Fig. 7.17 The block diagram of the cross-coupled PID controller

7.4.3 Real-time 2-DOF control with the cross-coupled PID controller

The task in this case is the same as in the previous sections but TRAS is not mechanically blocked, and therefore it is free to move in both planes.

Click the *2-DOF controller* button and the model shown in Fig. 7.18 opens. Set the coefficients of the crossed PID controllers as follows:

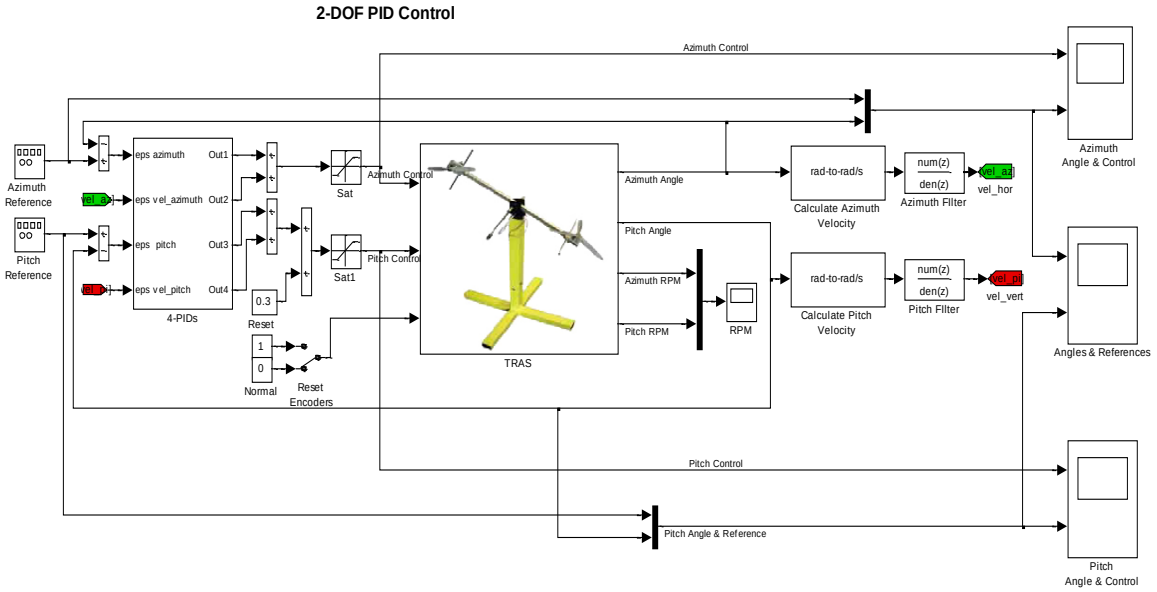


Fig. 7.18 Real-time model of the 2-DOF control task

Click "*4 PIDs*" block and set the controller parameters as in Fig. 7.19.

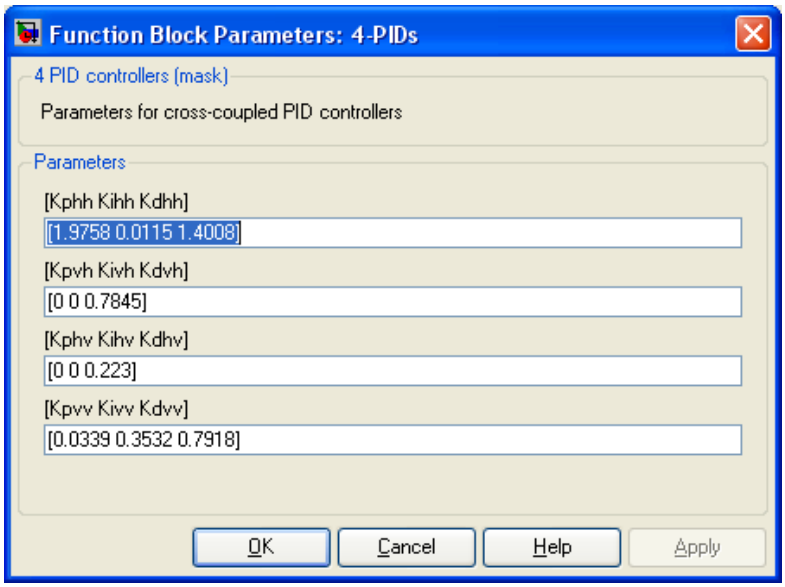


Fig. 7.19. Coeffitients of the cross-coupled PID controller

Also set the reference signals as follows: the reference azimuth signal as square wave with 0.4 [rad] amplitude and 1/50 [Hz] frequency, and the reference signal for azimuth as sin wave with the amplitude 0.1 and frequency 1/60 [Hz].

Build the model and click on the *Simulation*/*Connect to target* option and *Start real-time code* option.

The results of the experiment are shown in Fig. 7.20.

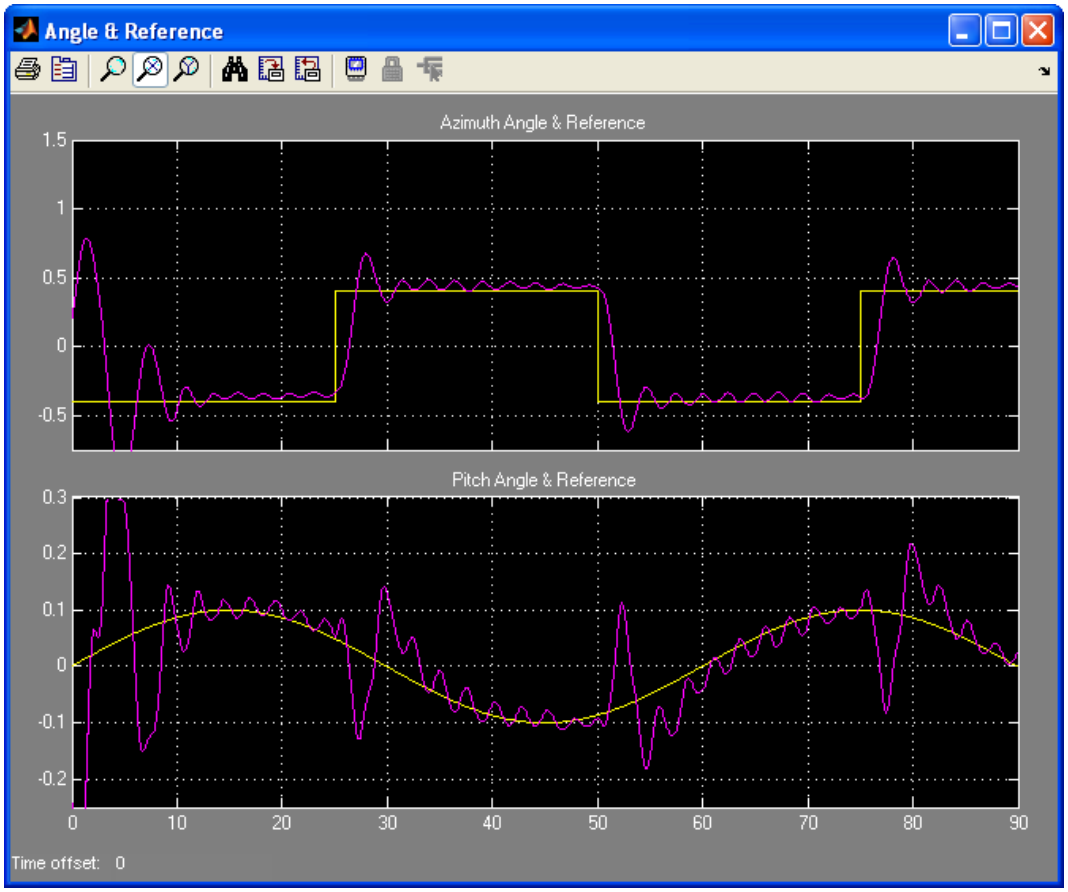


Fig. 7.20 Results of the 2-DOF control with the cross-coupled PID controller (azimuth and pitch positions)

8. PID controller parameters tuning

There are several methods of designing closed-loop control systems. In order to obtain optimal (or sub-optimal) settings of parameters for the PID controllers the so-called tuning methods may be used. The following tuning methods can be distinguished:

- Tuning based on the time or frequency responses. An experiment is performed with the process and with the model of the process. Tuning rules are based on time or frequency responses of the system. This method is not used for *TRAS.*
- More general method is the minimisation of a objective function. The idea of this method for *TRAS* with a PID controller is presented in Fig. 8.1.

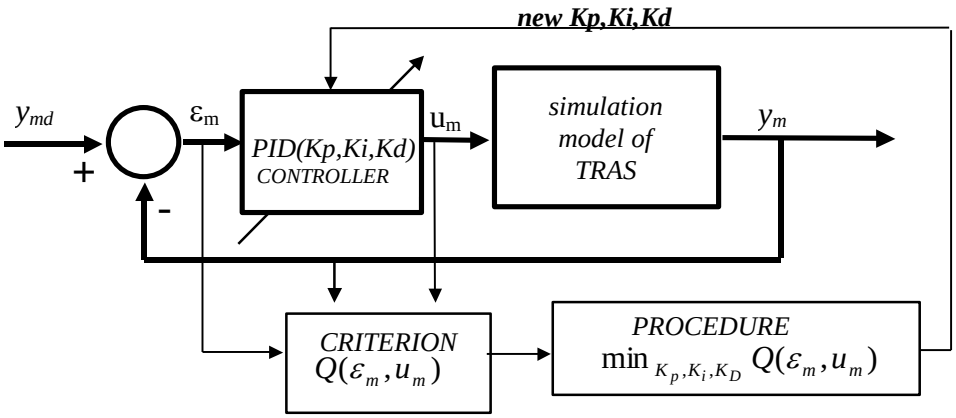


Fig. 8.1.Schematic diagram of the PID parameters tuning

In the case of *TRAS* the following criterion is used to tune the PID parameters for the all experiments described in the previous section

$$Q = \int_o^{T_k} \left(4\varepsilon_h^2 + \varepsilon_v^2 + (u_v + 0.1)^2\right) dt$$

where: $T_k = 80$ [s] – simulation time , ε_h - azimuth position error, ε_v - pitch position error, u_v - pitch control. The 0.1 coefficient is the value of the pitch control which keeps the beam in horizontal position.

Note, that selection of the criterion is a rather complicated and difficult task. It is closely related to the project assumptions. The project assumptions consist a basis for its construction.

The *TRAS Toolbox* includes the m-files to perform optimisation procedures of PID controller parameters. These m-files are as follows:

- *pid_azimuth.m*
- *pid_pitch.m*
- *pid_cross.m*
- *pid_simple.m*.

Each of these files uses his own simulation model and the criterion m-file in optimisation process. See the body of these files to learn how the optimisation procedure is performed.

9. Description of the CTRAS class properties

The *CTRAS* is a MATLAB class, which gives the access to all the features of the RTDAC/USB2 board equipped with the logic for the Twin Rotor Aerodynamical System model. The RTDAC/USB2 board is an interface between the control software executed by a PC computer and the power-interface electronic of the TRAS model. The logic on the board contains the following blocks:

- incremental encoder registers – two 32-bit registers to measure the positions of the incremental encoders. There are two identical encoder inputs, that are applied to measure the azimuth and pitch angles;
- incremental encoder resets logic. The incremental encoders generate different output waves when the encoder rotates clockwise and counter-clockwise. The encoders are not able to detect the reference ("zero") position. To determine the "zero" position the incremental encoder registers have to be set to zero by the computer program;
- PWM generation blocks – generates the Pulse-Width Modulation output signals applied to control the azimuth and pitch DC drives. Simultaneously the direction signals and the brake signals are generated to control the power interface module. The PWM prescalers determines the frequencies of the PWM wave;
- power interface thermal flags –the thermal flags can be used to disable the operation of the overheated DC motors;
- interface to the on-board analog-to-digital converter. The A/D converter is applied to measure the output voltages from the tachogenerator.

All the parameters and measured variables from the RTDAC/USB2 board are accessible by appropriate properties of the *CTRAS* class.

In the MATLAB environment the object of the *CTRAS* class is created by the command:

object_name = CTRAS;

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 a 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 *CTRAS* 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 subjected to the property
Arguments	Describes arguments of the set method

See	Refers to other related properties
Examples	Provides examples how the property can be used

9.2 BitstreamVersion

Purpose: Read the version of the logic stored in the RTDAC/USB2 board.

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

Description: The property determines the version of the logic design for the RTDAC/USB2 board. The TRAS 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.

Example: Create the CTRAS object:
tr = CTRAS;
Display their properties by typing the command:
tr

```
Type:                CTRAS Object
Bitstream ver.:      x40F
Encoder:             [ 2  65517 ][bit]
Reset Encoder:       [ 0  0 ]
Input voltage:       [ -0.01  -0.02 ][V]
PWM:                 [ 0  0 ]
PWM Prescaler:       [ 1  1 ]
PWM Thermal Status:  [ 0  0 ]
PWM Thermal Flag:    [ 1  1 ]
Angle:               [ 0.003068  -0.029146 ][rad]
RPM:                 [ -19  -9 ][RPM]
Time:                753.7 [sec]
```

9.3 Encoder

Purpose: Read the incremental encoder registers.

Synopsis: *enc = get(tr, 'Encoder');*

Description: The property returns two digits. They are equal to the number of impulses generated by the corresponding encoders. The encoder counters are 16-bit numbers so the values of this property is from –32768 to 32767. When an encoder counter is reset the value is set to zero. The first encoder register corresponds to the azimuth position and the second register corresponds to the pitch position.
The incremental encoders generate 4096 pulses per rotation. The values of the *Encoder* property should be converted into physical units.

See: ResetEncoder, Angle, AngleScaleCoeff

9.4 Angle

Purpose: Read the angle of the encoders.

Synopsis*:* *angle_rad = get(tr, 'Angle');*

Description: The property returns two angles of the corresponding encoders. The first value corresponds to the azimuth and the second to the pitch position. To calculate the angle the encoder counters are multiplied by the values defined as the *AngleScaleCoeff* property. The angles are expressed in radians.

See: Encoder, AngleScaleCoeff

9.5 AngleScaleCoeff

Purpose: Read the coefficients applied to convert the encoder counter values into physical units.

Synopsis*:* *scale_coeff = get(tr, 'AngleScaleCoeff');*

Description: The property returns two digits. They are equal to the coefficients applied to convert encoder impulses into radians. The incremental encoders generate 4096 pulses per rotation so the coefficients are equal to 2*pi/4096.

See: Encoder, Angle

9.6 PWM

Purpose: Set the direction and duty cycle of the PWM control waves.

Synopsis*:* *PWM = get(tr, 'PWM');*
set(tr, 'PWM', [NewAzimuthPWM NewPitchPWM]);

Description: The property determines the duty cycle and direction of the PWM control waves for the azimuth and pitch DC drives. The PWM waves and the direction signals are used to control the DC drives so in fact this property is responsible for the DC motor control signals. The *NewAzimuthPWM* and *NewPitchPWM* variables are scalars in the range from –1 to 1. 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.

The PWM wave is not generated if the corresponding thermal flag is set and the power amplifier is overheated.

See*: PWMPrescaler, Therm, ThermFlag*
Example: set(tr, 'PWM', [-0.3 0.0]);

9.7 PWMPrescaler

Purpose: Determine the frequency of the PWM waves.

Synopsis*:* *Prescaler = get(tr, 'PWMPrescaler');*
set(tr, 'PWMPrescaler', [NewAzimuthPrescaler NewPitchPrescaler]);

Description: The prescaler values can vary from 0 to 63. The 0 value generates the maximal PWM frequency. The value 63 generates the minimal frequency. The first prescaler value is responsible for the azimuth PWM frequency and the second for the pitch PWM frequency. The frequency of the generated PWM wave is given by the formula:
$\mathbf{P}WM_{frequency} = 40MHz / 1023 / (Prescaler+1)$

See*:* *PWM*

9.8 Stop

Purpose: Sets the control signal to zero.

Synopsis*:* *set(tr, 'Stop');*

Description: This property can be called only by the set method. It sets the zero control of the DC motors and is equivalent to the *set(tr, 'PWM', [0 0])* call.

See*:* *PWM*

9.9 ResetEncoder

Purpose: Reset the encoder counters.

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

Description: The property is used to reset the encoder registers. The *ResetFlags* is a 1x2 vector. Each element of this vector is responsible for one encoder register (the first value controls the reset signal of the azimuth encoder and the second controls the reset of the pitch encoder). If the reset flag is equal to 1 the appropriate register is set to zero. If the flag is equal to 0 the appropriate register counts encoder impulses.

See*:* *Encoder*

Example*:* To reset only the first encoder register execute the command:
set(tr, 'ResetEncoder', [1 0]);

9.10 Voltage

Purpose: Read two voltage values.

Synopsis*:* *Volt = get(tr, 'Voltage');*

Description: Returns the voltage of two analog inputs. The analog inputs are applied to measure the output of the tachogenerators.

See*:* *RPM*

9.11 RPM

Purpose: Read velocity of the propelers.

Synopsis*:* *RPM = get(tr, 'RPM');*

Description: Returns the velocities of the propellers. The property contains two values. The first one is equal to the azimuth propeller velocity. The second one is equal to the pitch propeller velocity.

See: Voltage, RPMScaleCoeff

9.12 RPMScaleCoeff

Purpose: Read the coefficients applied to convert the tachgenerator voltage values into physical units.

Synopsis*:* *scale_coeff = get(tr, 'RPMScaleCoeff');*

Description: The property returns two digits. They are equal to the coefficients applied to convert tachogenerator voltages into RPMs.

See: Voltage, RPM

9.13 Therm

Purpose: Read thermal status flags of the power amplifiers.

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

Description: Returns the thermal flag of the power amplifier. When the temperature of a power amplifier is too high the corresponding flag is set to 1. The property contains two flags. The first one corresponds to the thermal status of the power interface for the azimuth DC drive. The second one corresponds to the thermal status of the pitch power amplifier.

See: *ThermFlag*

9.14 ThermFlag

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

Synopsis: *ThermFlag = get(tr, 'ThermFlag');*
set(tr, 'ThermFlag', [NewAzimuthThermFlag NewPitchThermFlag]);

Description: If the *NewAzimuthThermFlag* or/and *NewPitchThermFlag* are equal to 1 the azimuth or/and DC motors are not excited by from the PWM waves when the corresponding power interfaces is overheated.

See: *Therm*

9.15 Time

Purpose: Return time information.

Synopsis: *T = get(tr, 'Time');*

Description: The *CTRAS* object contains the time counter. When a *CTRAS* 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 which elapsed since the object was created.

9.16 Quick reference table

Property name	Operation*	Description
BaseAddress	R	Read the base address of the RTDAC/USB2 board
BitstreamVersion	R	Read the version of the logic design for the RTDAC/USB2 board
Encoder	R	Read the incremental encoder registers
Angle	R	Read the angles of the encoders
AngleScaleCoeff	R	Read the coefficients applied to convert encoder positions into radians
PWM	R+S	Read/set the parameters of the PWM waves
PWMPrescaler	R+S	Read/set the frequency of the PWM waves
Stop	S	Set the control signal to zero
ResetEncoder	R+S	Reset the encoder counters or read the reset flags
Voltage	R	Read the input voltages
RPM	R	Read velocities of the propelers
RPMScaleCoeff	R	Read the coefficients applied to convert tachogenerator voltages into RPMs
Therm	R	Read the thermal flags of the power amplifiers
ThermFlag	R+S	Read/set the automatic power down flags of the power amplifiers
Time	R	Read time information

- R – read-only property, S – allowed only set operation, R+S –property may be read and set

9.17 CTRAS Example

To familiarise a reader with the CTRAS class this section presents an M-file example that uses the properties of the CTRAS class to measure the static characteristics of the DC motor. The static characteristics is a diagram showing the relation between DC motor control signal and the velocity of the propellers. The M-file changes the control signal and waits until the system

reaches a steady-state. The velocity of the propeller is proportional to the voltage generated by the tacho-generator.

The M-file is written in the M-function form. The name of the M-function is *TRAS_PWM2RPM*. The body of this function is given below. The comments within the function describe the main measurement stages.

The function requires five parameters:

- `SelectRotor` – selects the propeller used during the measurements. Available values are: 'A' for azimuth propeller, 'P' for pitch propeller and 'AP' for both propellers.
- `CtrlDirection` - a string that selects how to change the control value. The 'A' string causes the control is changed in ascending manner (from minimal to maximal control value), the 'D' string causes the control is changed in descending order (from maximal to minimal value) and the 'R' string causes reverse double changes (from minimal to maximal and after that from maximal to minimal control values),
- `MinControl, MaxControl`- minimal and maximal control values. The control values must be set within the –1.0 to +1.0 range,
- `NoOfPoints` - number of characteristic points within the range where changes the control signal. The exact number of points of the characteristics declared by this parameter is enlarged to two points namely at the ends of the control range.

```
function ChStat = ...
             TRAS_PWM2RPM( SelectRotor, CtrlDirection, ...
             MinControl, MaxControl, NoOfPoints )

    SelectRotor   = lower( SelectRotor );
    CtrlDirection = lower( CtrlDirection );
    NoOfPoints    = max( 1, NoOfPoints-1 );

    % Control step
    Step = (MaxControl-MinControl) / NoOfPoints;

    switch CtrlDirection
      case 'a'
        Ctrl = MinControl:Step:MaxControl;
      case 'd'
        Ctrl = MaxControl:-Step:MinControl;
      case 'r'
        Ctrl = [ MinControl:Step:MaxControl MaxControl:-Step:MinControl];
      otherwise  % This should not happen
        error('The CtrlDirection must be ''A'',''D'' or ''R''.')
    end

    switch SelectRotor
      case 'a'
        ACtrl = Ctrl; PCtrl = 0*Ctrl;
      case 'p'
        ACtrl = 0*Ctrl; PCtrl = Ctrl;
      case { 'ap', 'pa' }
        ACtrl = Ctrl; PCtrl = Ctrl;
      otherwise  % This should not happen
        error('The SelectRotor must be ''A'', ''P'' or ''AP''.')
```

```
      end

      FigNum = figure( 'Visible', 'on', ...
                       'NumberTitle', 'off', ...
                         'Name', 'Rotor velocity vs. PWM characteristic', ...
                       'Menubar', 'none' );
      tr = ctras;
      ret = [];
      for i=1:length(Ctrl)
        set( tr, 'PWM', [ACtrl(i) PCtrl(i)] );
        pause( 10 )
        ret(i,1)   = Ctrl(i);
        AuxVolt = [0 0];
        for j=1:10
          AuxVolt = AuxVolt + get( tr, 'RPM' );
        end
        ret(i,2:3) = AuxVolt/10;
        AuxVolt = 0;
        for j=1:10
          AuxVolt = AuxVolt + get( tr, 'AD', 10 );
        end
        ret(i,4) = AuxVolt/10;
        plot( ret(:,1), ret(:,2:3), 'x' );
        hold on; plot( ret(:,1), ret(:,2:3) ); hold off; grid
        title( 'RPM vs. PWM' ); xlabel('PWM control value'); ylabel( 'Rotor
velocity [RPM]' );
      end

      ChStat.Control = ret(:,1);
      ChStat.RPM     = ret(:,2:3);
      ChStat.Force   = ret(:,4);

      % Switch off the control
      set( tr, 'PWM', [0 0] );
      % Set return variable
      ChStat.Control = ret(:,1);
      ChStat.RPM     = ret(:,2:3);
      ChStat.Force   = ret(:,4);

      % Switch off the control signals
      set( tr, 'PWM', [0 0] );
```

The diagram generated by the call:

tras_pwm2rpm('ap', 'r', -0.5, 0.5, 11)

is shown below. Two curves represent static characteristics of the azimuth and pitch propellers.

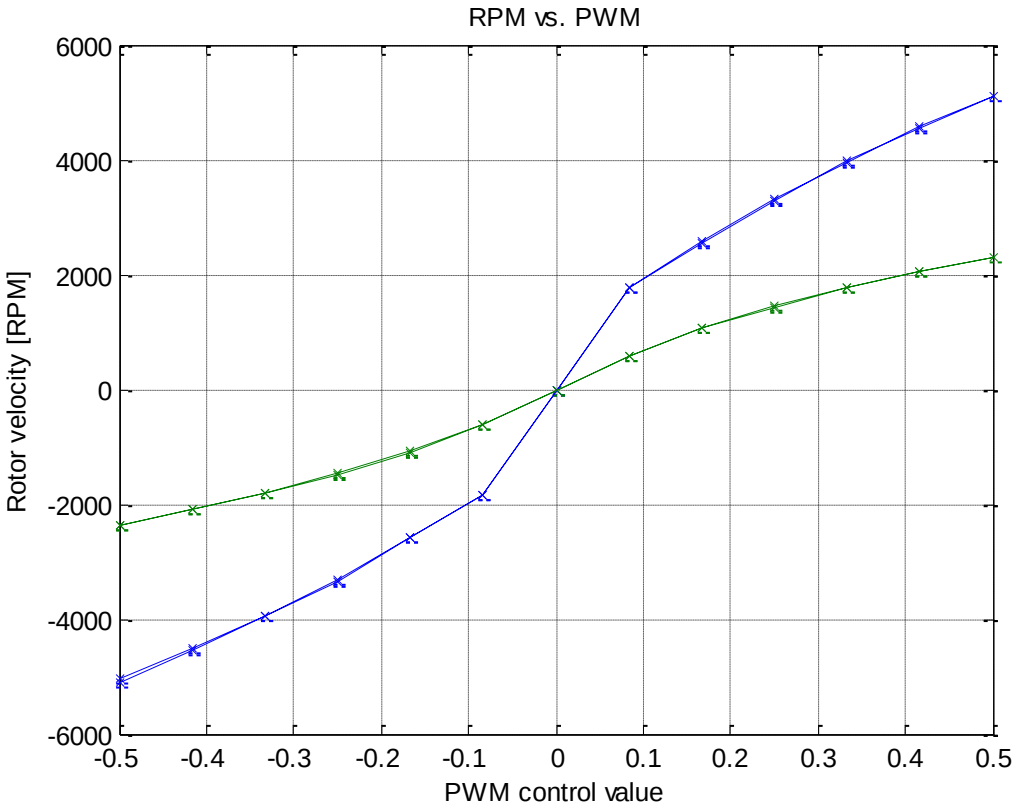


Fig. 9.1 Static characteristics