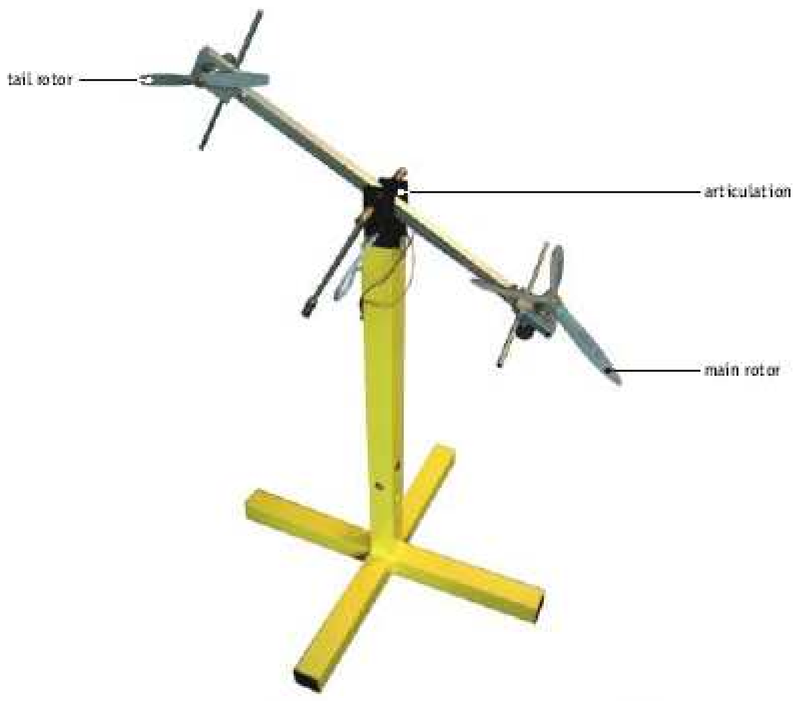
Two Rotor Aero-dynamical System

MATLAB
R2009a/b, R2010a/b, R2011a/b, R2012a
PCI version

User's Manual



www.inteco.com.pl

Table of contents

COPYRIGHT NOTICE

ACKNOWLEDGEMENTS

Inteco acknowledges all trademarks.
MICROSOFT, WINDOWS are registered trademarks of Microsoft Corporation.
MATLAB, Simulink, RTWT and RTW are registered trademarks of Mathworks Inc.

1. Introduction

Two **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 nonlinear MIMO 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 RT-DAC/PCI I/O board configured in the Xilinx® technology. The software operates in real-time under MS Windows® XP/W7 using MATLAB® R2009a/b, R2010a/b, R2011a/b and R201a with RTW and RTWT toolboxes.

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 and RTWT toolboxes. 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.

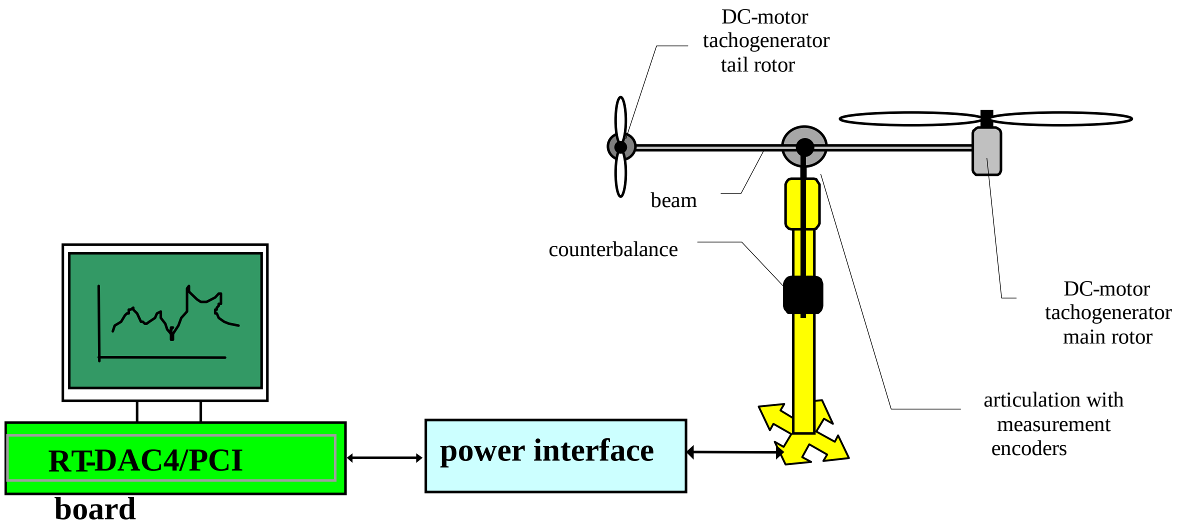


Fig. 1.1 The laboratory set-up: helicopter-like system

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 of the rotors. 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 (RT-DAC/PCI). Fig. 1.2 shows details of the hardware configuration of the control system for TRAS.

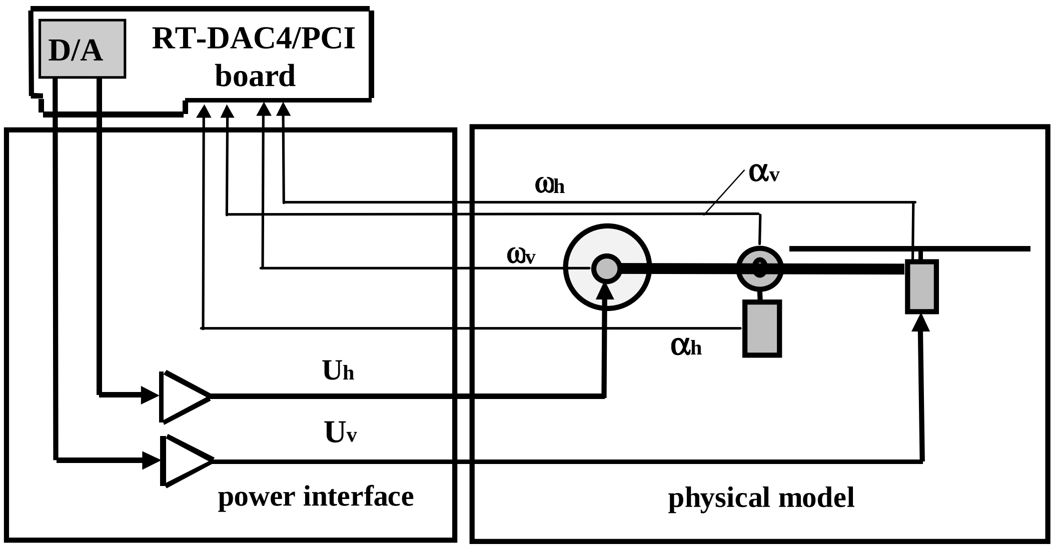


Fig. 1.2 Hardware configuration of TRAS

The control software for TRAS is included in the *TRAS toolbox*. This toolbox uses the RTWT and RTW toolboxes from MATLAB.
TRAS Toolbox is a collection of M-functions, MDL-models and C-code MEX-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-up,
- RT-DAC/PCI I/O board. The board contains FPGA equipped with dedicated logic design,
- Pentium or AMD based personal computer.

Software
For development of the project and automatic building of the real-time program the following software has to be properly installed on the PC:

- Microsoft Windows XP/W7,
- MATLAB version R2009a/b, R2010a/b, R2011ab or R2012a with appropriate versions of Simulink RTW and RTWT toolboxes (not included),
- Control Toolbox from MathWorks Inc. to develop the project,
- The TRAS toolbox which includes specialised drivers for the TRAS System. These drivers are responsible for communication between MATLAB and the RT-DAC/PCI measuring and control board.

⇨ **The built-in Open Watcom compiler is applied.**

Manuals:

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

⇨ **The experiments and corresponding to them measurements have been conducted by the use of the standard INTECO system. Every new system manufactured and developed by INTECO can be slightly different to those standard devices. It explains why a user can obtain results maybe slightly different to these given in the manual.**

1.3 FEATURES of TRAS

- A highly nonlinear MIMO system ideal for illustrating complex control algorithms.
- 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, RTW and RTWT toolboxes from *MathWorks Inc.*

1.4 Software installation

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

2. TRAS Control Window

2.1 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 RTWT toolbox. It is assumed that a user is familiarised with MATLAB tools especially with RTWT toolbox. Therefore we do not include the detailed description of this tool.
The user has a rapid access to all basic functions of the TRAS System from the *TRAS Control Window*. It includes: identification, drivers, simulation model and application examples.
In the Matlab command window type

tras

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

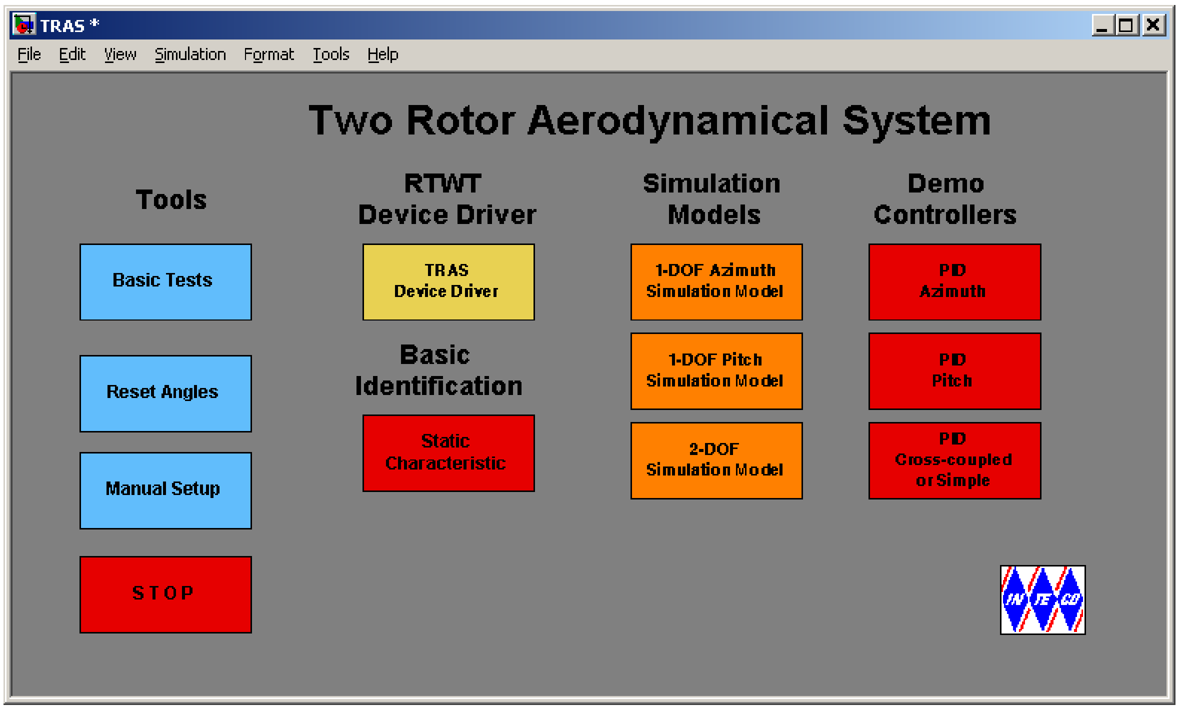


Fig. 2.1 TRAS Control Window

TRAS Control Window contains: testing tools, drivers, models and demo applications. The user has a rapid access to all basic functions of the TRAS control system from *TRAS Control Window*.

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 - RTWT 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.

2.2 Basic test

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)

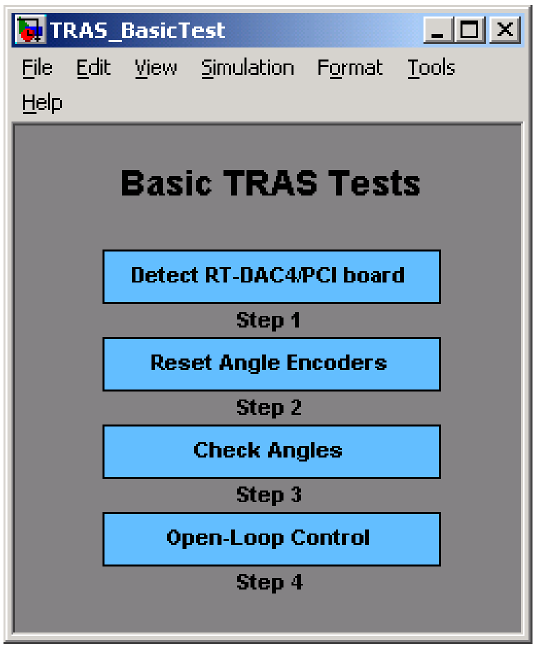
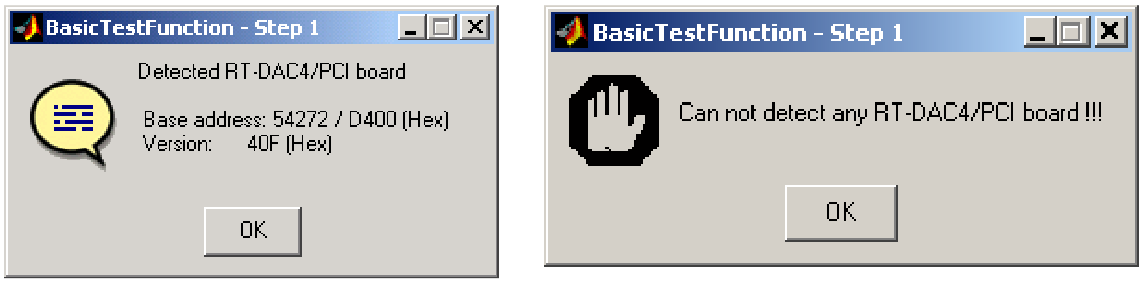


Fig . 2.2 The *Basic Tests* window

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. As well the emergency switch can be used anyway.

The first step in the TRAS testing is to check if the RT-DAC/PCI measuring and control board is installed properly.

- Double click the *Detect RT-DAC/PCI board* button. One of the messages shown in Fig. 2.3 opens. If the board has been correctly installed, the base address, and the number of logic version of the board are displayed.


BasicTestFunction - Step 1

Detected RT-DAC4/PCI board

Base address: 54272 / D400 (Hex)
Version: 40F (Hex)

BasicTestFunction - Step 1

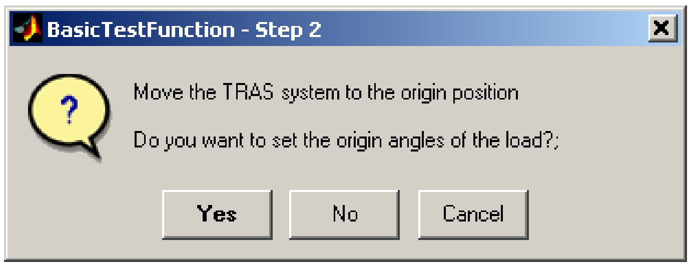
Can not detect any RT-DAC4/PCI board !!!

Fig. 2.3 Result of the step 1

If the board is not detected then check whether the board has been mounted correctly into a slot of the computer. The boards are checked very precisely before sending to a customer. In principle, a wrong assembling is the only reason of failure in detecting the board.

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

- 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.


BasicTestFunction - Step 2

Move the TRAS system to the origin position

Do you want to set the origin angles of the load?;

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 TRAS in all directions and observe measurements on the screen (see Fig. 2.5).

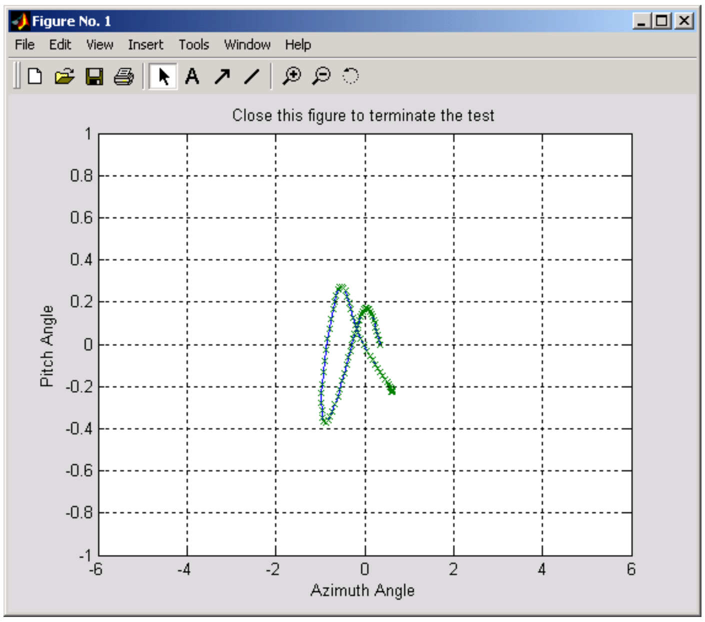


Fig. 2.5 Measurements of the beam motion

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.

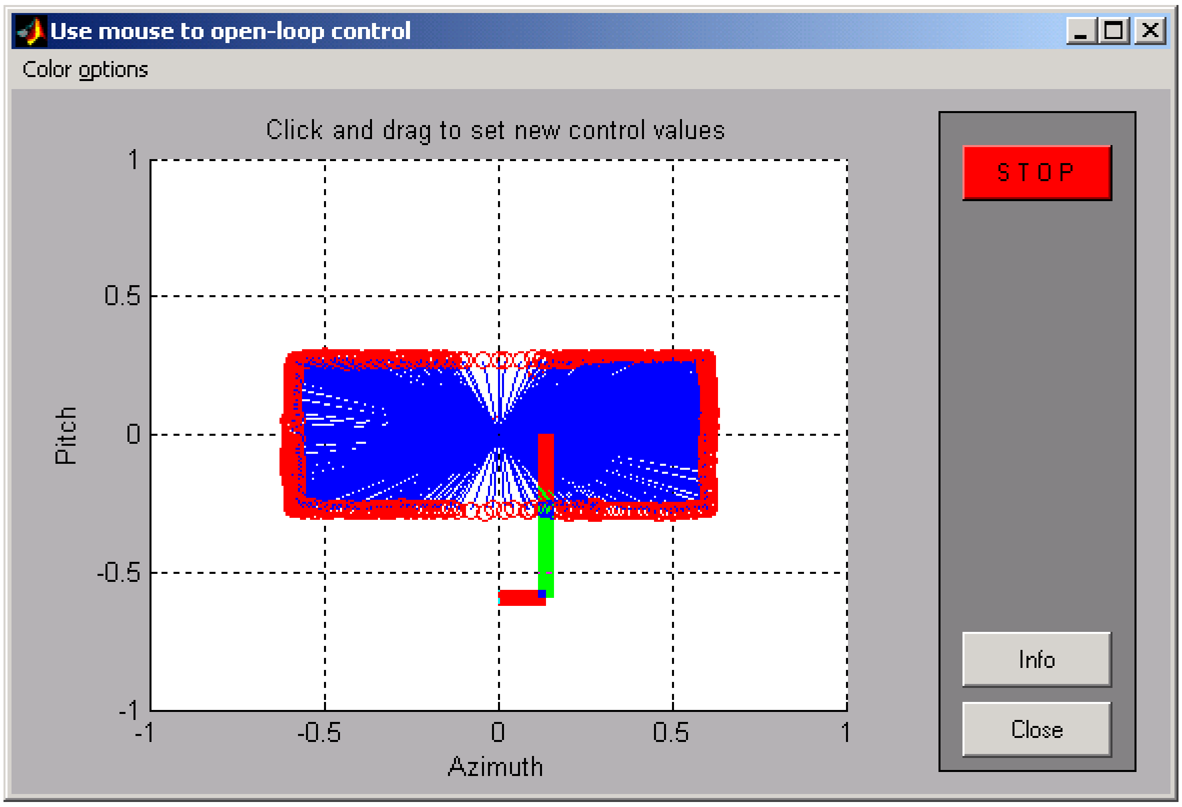

Use mouse to open-loop control
Color options
Click and drag to set new control values
Pitch
Azimuth
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1
STOP
Info
Close

Fig. 2.6 Motors control and checking of tacho-generators

Troubleshooting

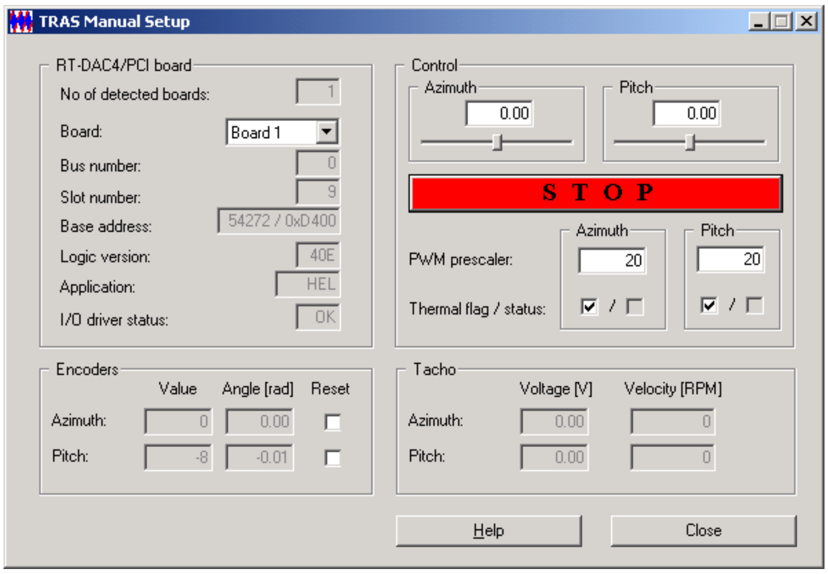
Message or faulty action	Solution
Board not detected	Check mounting of the board. Check if the driver is installed
Angles measurements failed	Check the Enc socket and wiring
Propellers do not rotate	Check the M socket, Mains and ON switch
Velocities are not measured	Check the T socket and wiring

2.3 TRAS Manual Setup

The *TRAS Manual Setup* program gives access to the basic parameters of the laboratory Two Rotor Aerodynamical System setup. The most important data transferred from the RT-DAC/PCI board and the measurements of the TRAS may be shown. Moreover, the control signals may be set.

The application contains four frames (see Fig. 2.7):

- RT-DAC/PCI board,
- Encoders,
- Control and
- Tacho.



TRAS Manual Setup

RT-DAC4/PCI board
- No of detected boards: 1
- Board: Board 1
- Bus number: 0
- Slot number: 9
- Base address: 54272 / 0xD400
- Logic version: 40E
- Application: HEL
- I/O driver status: OK

Control
- Azimuth: 0.00
- Pitch: 0.00

S T O P

	Azimuth	Pitch
PWM prescaler:	20	20
Thermal flag / status:	✓ / ☐	✓ / ☐

Encoders

	Value	Angle [rad]	Reset
Azimuth:	0	0.00	☐
Pitch:	-8	-0.01	☐

Tacho

	Voltage [V]	Velocity [RPM]
Azimuth:	0.00	0
Pitch:	0.00	0

Fig. 2.7 View of the *TRAS Manual Setup* window

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

***RT-DAC/PCI board* frame**

The *RT-DAC/PCI board* frame presents the main parameters of the PCI board.

No of detected boards

Reads the number of detected RT-DAC/PCI boards. If the number is equal to zero it means that the software has not detected none of the RT-DAC/PCI board. When more then one board is detected the *Board* list must be used to select the board that communicates with the program.

2.3.1.1 Board

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

Bus number

Displays the number of the PCI bus where the current RT-DAC/PCI board is plugged-in. If more then one board is used this parameter may be useful to distinguish the boards.

Slot number

The number of the PCI slot in which the current RT-DAC/PCI board is plugged-in. If more then one board is used this parameter may be useful to distinguish the boards.

Base address

The base address of the current RT-DAC/PCI board. The RT-DAC/PCI board occupies 256 bytes of the I/O address space of the microprocessor. The base address is equal to the beginning of the occupied I/O range. The I/O space is assigned to the board by the computer operating system and may be different for various computers.

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

Logic version
The number of the configuration logic of the on-board FPGA chip. A logic version corresponds to the configuration of the RT-DAC/PCI 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 unchecked 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:
$F_{PWM} = 40000/1023/(1+PWMPrescaler)$ *[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 RT-DAC/PCI 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 tachogenerators. 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.

2.4 RTWT Device Driver

The driver is a software go-between for the real-time MATLAB environment and the RT-DAC/PCI I/O board. The control and measurements are transferred. Click the *TRAS Device Driver* button and the driver window opens (Fig. 2.8).

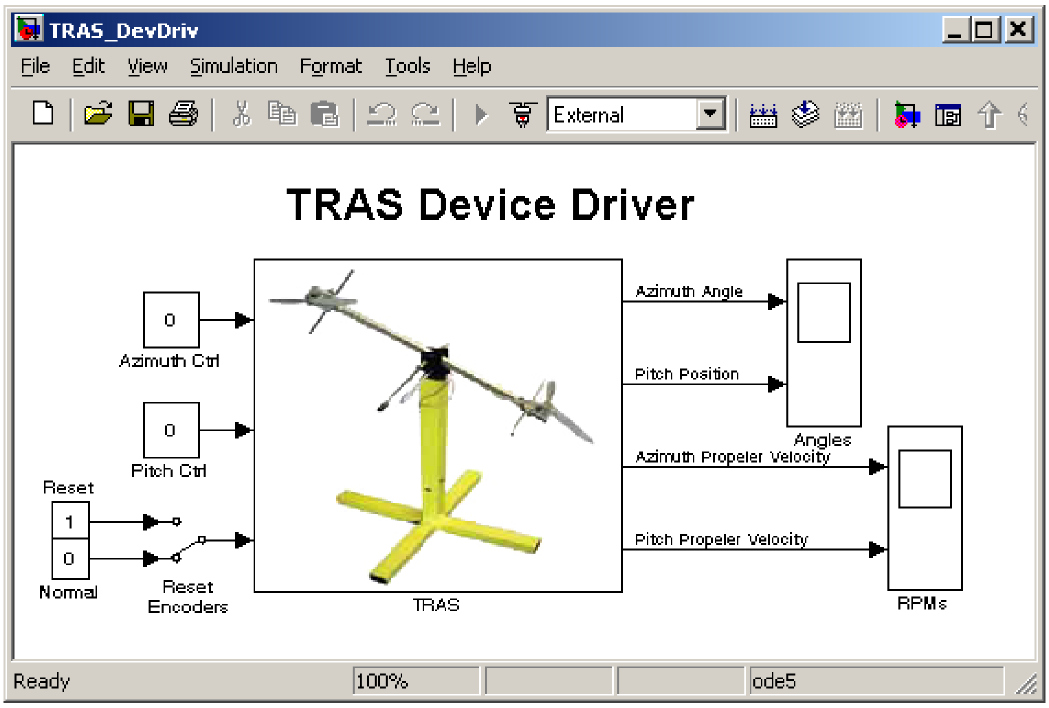


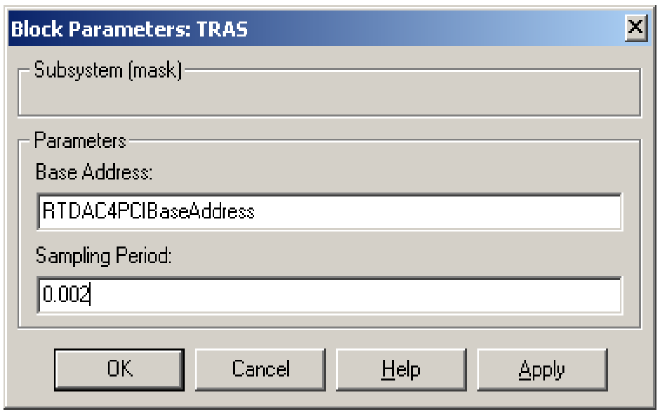
Fig. 2.8 RTWT Device Driver

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

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 the *Reset* signal changes to one the encoders are reset and do not work. If the *Reset* signal is equal to zero encoders work in the standard way. It means when switching occurs, encoders reset and start measure when the switch returns to the zero (normal) position. It is important that the *Reset* switch works only when the real-time code is executed.

The mask of this block (shown Fig. 2.9) contains base address of the RT-DAC/PCI board (automatically detected with the help *RTDACPCIBaseAddress* function) and the sampling period which default value is set to 0.002 sec. If one wants to change the default sampling time he must do it in this mask also.


Block Parameters: TRAS

Subsystem (mask)

Parameters

Base Address:

RTDAC4PCIBaseAddress

Sampling Period:

0.002

Fig. 2.9 Mask of the device driver

The details of the device driver are depicted in Fig. 2.10. The driver uses functions which communicates directly with a logic stored at the RT-DAC/PCI board.

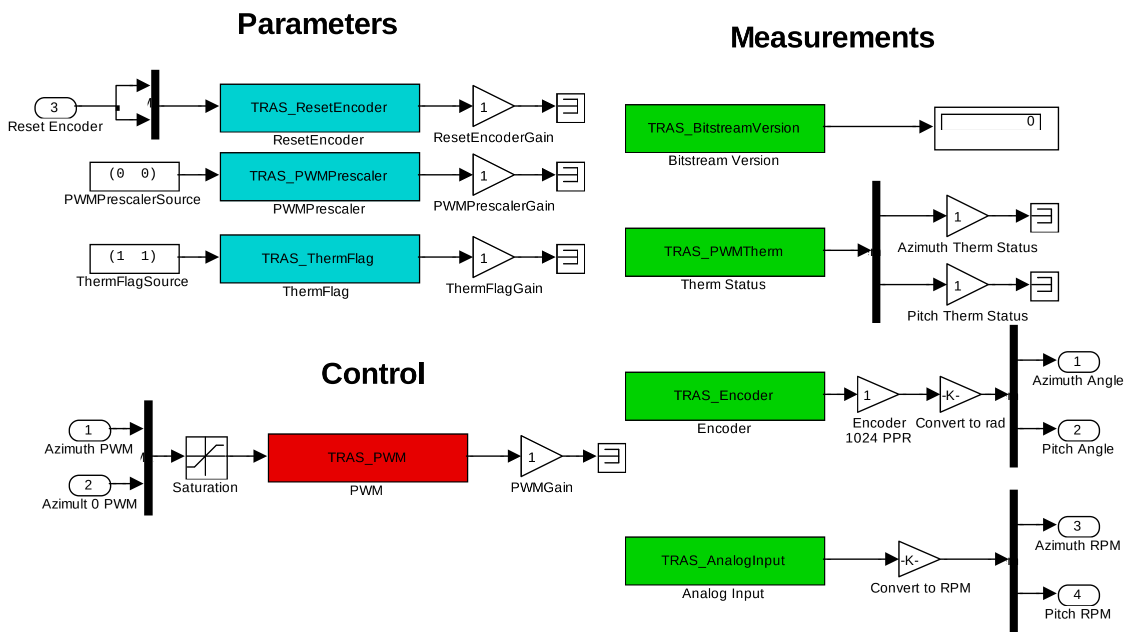


Fig. 2.10 Interior of the RTWT device driver

2.5 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. 2.11. Next, click the subsystem block to see details of the model.

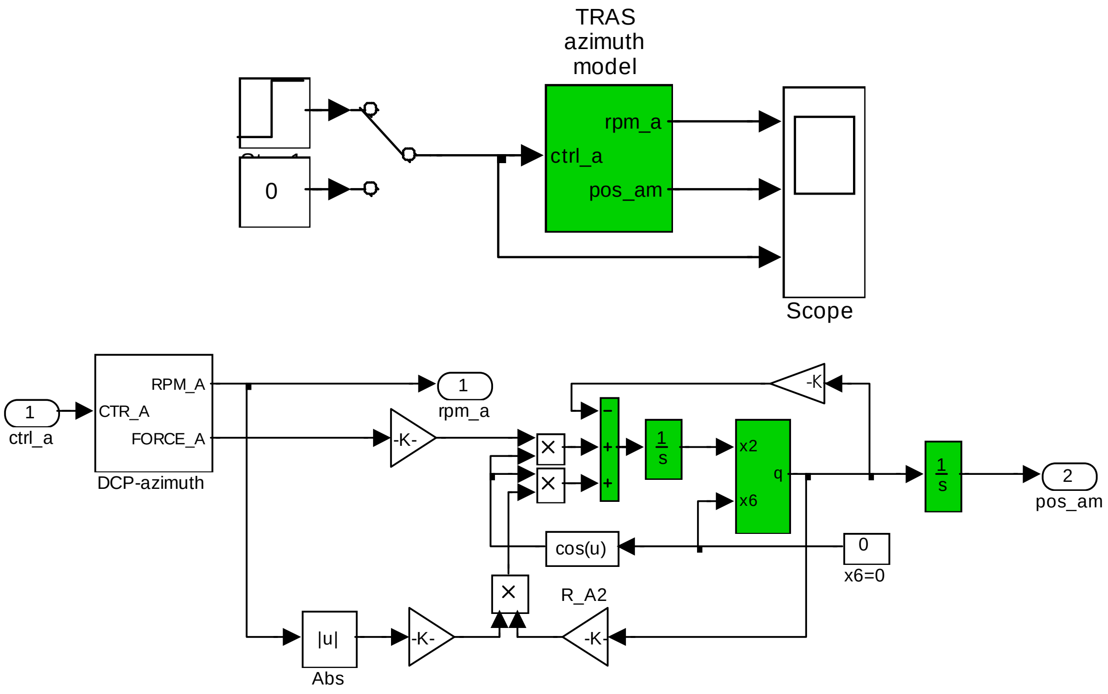
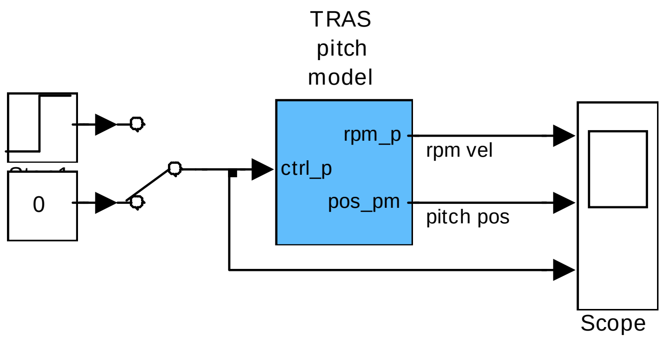


Fig. 2.11 The *Azimuth Simulation model* and its interior

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. 2.12)



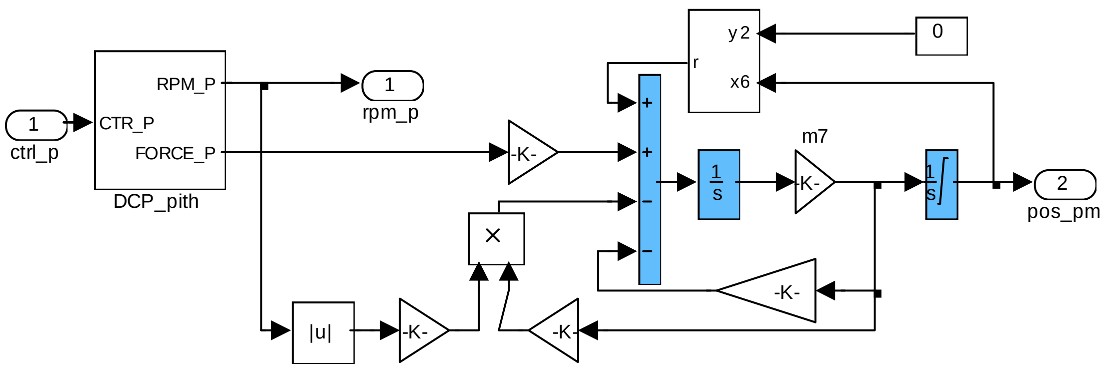
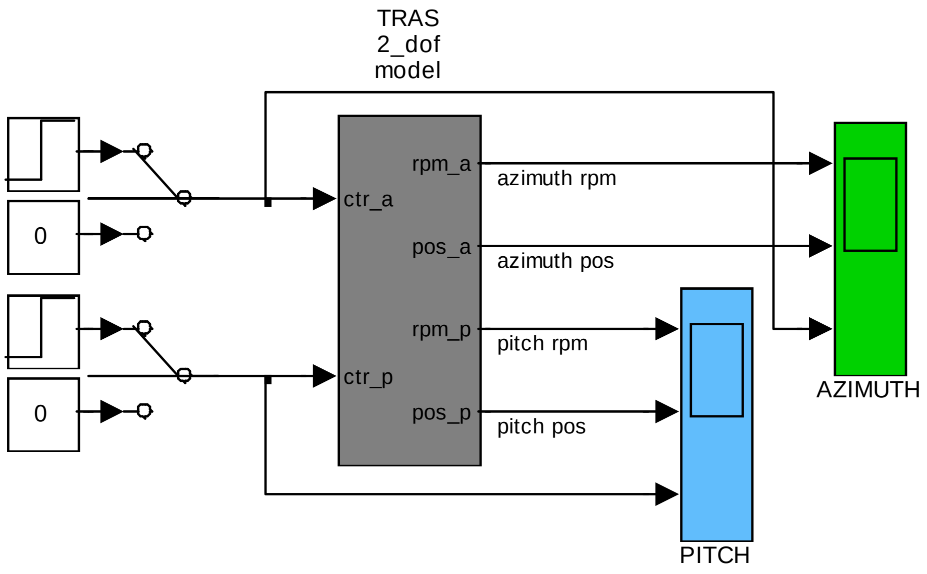


Fig. 2.12 The *Pitch Simulation* model and its interior

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. 2.13)



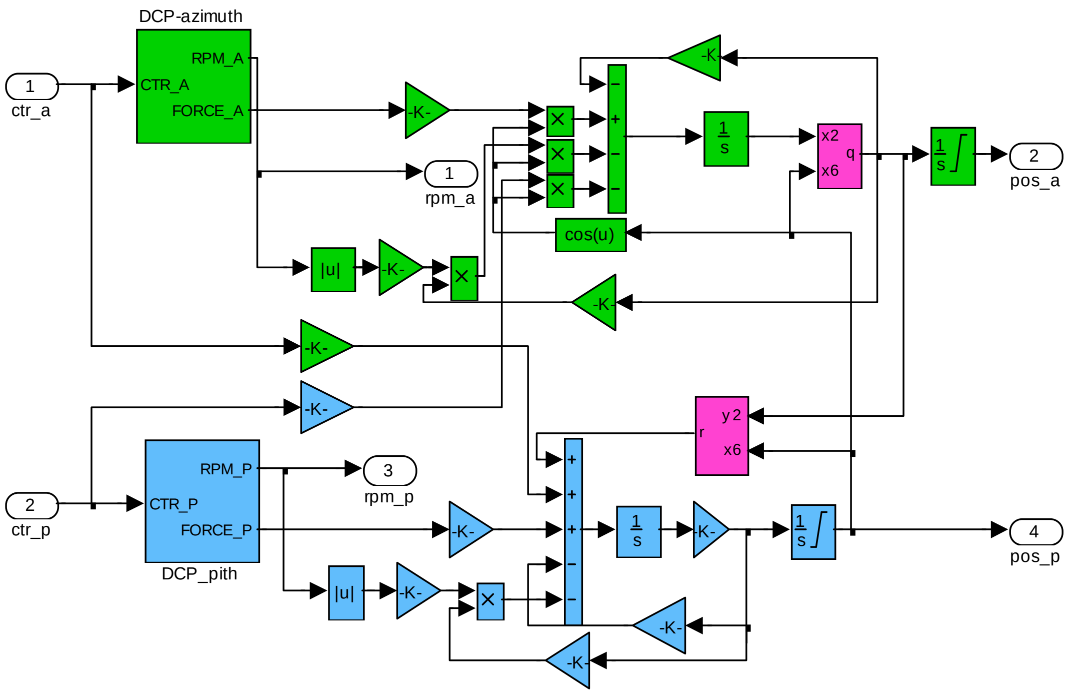


Fig. 2.13 The 2-DOF simulation model and its interior

3. Model and parameters

Modern methods of design and adaptation of real-time controllers require high quality mathematical models of the system. For high order, nonlinear 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. 3.1. 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.

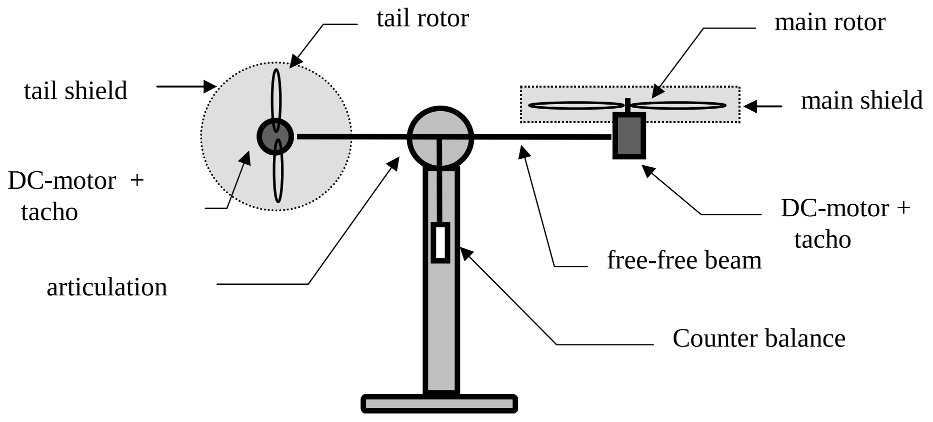


Fig. 3.1. Aero-dynamical model of TRAS

The block diagram of the TRAS model is shown in Fig. 3.2. 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. 3.2). 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.

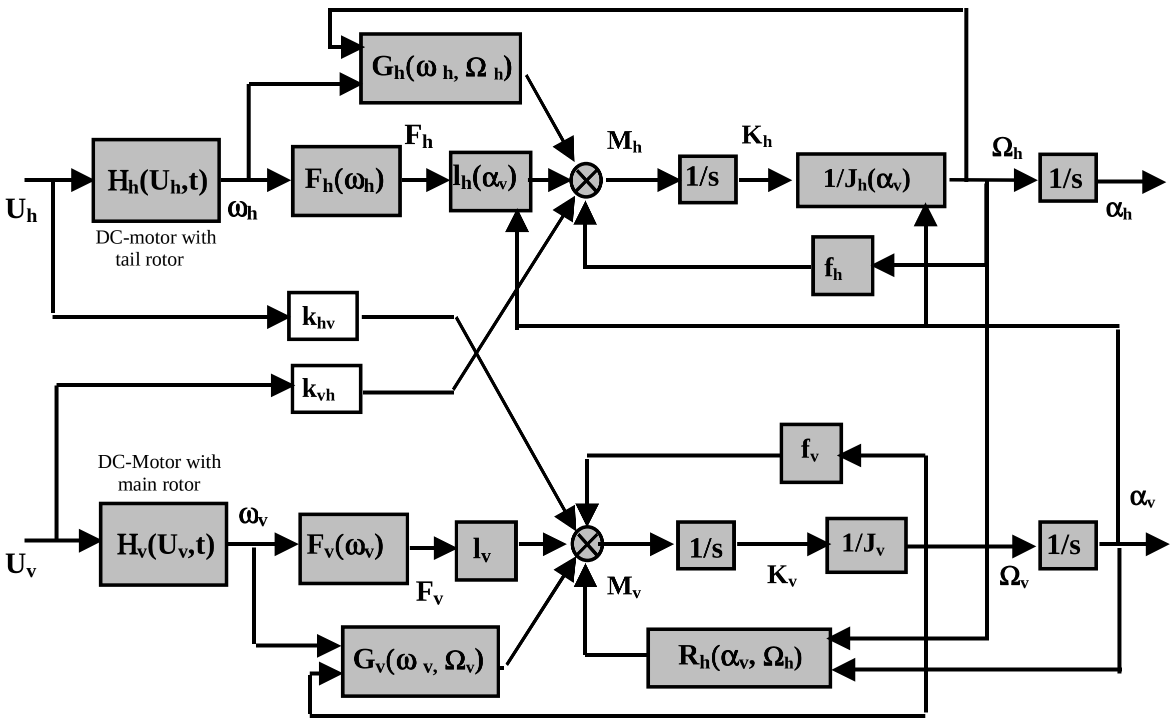


Fig. 3.2 Block diagram of the TRAS model

The following notation is used in Fig. 3.2:

α_h is horizontal position (azimuth position) of TRAS beam [rad],

Ω_h is angular velocity (azimuth velocity) of TRAS beam [rad/s],

U_h is horizontal DC-motor PWM control input,

ω_h is rotational speed of tail rotor [rad/s] - nonlinear function

$\omega_h=H_h(U_h,t)$ [rad/s],

F_h is aerodynamic force from tail rotor - nonlinear function $F_h=F_h(w_h)$ [N],

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

J_h is nonlinear function of moment of inertia with respect to vertical axis, $J_h = J_h(a_v)$ [kg m^2],

M_h is horizontal turning torque [Nm],

K_h is horizontal angular momentum [Nms],

f_h is oment of friction force in vertical axis [Nm],

α_v is vertical position (pitch position) of TRAS beam [rad],

Ω_v is angular velocity (pitch velocity) of TRAS beam [rad/s],

U_v is vertical DC-motor PWM voltage control input,

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

F_v is aerodynamic force from main rotor - nonlinear function $F_v = F_v(\omega_v)$ [N],

l_v is arm of aerodynamic force from main rotor [m],

J_v is moment of inertia with respect to horizontal axis [kg m^2],

M_v is vertical turning moment [Nm],
K_v is vertical angular momentum [Nms],
f_v is moment of friction force in horizontal axis [Nm],
R_v is vertical returning moment $R_h = f_{cf} + f_g = R_h(\alpha_v, \Omega_h)$ [Nm],
J_{hv} is vertical angular momentum from tail rotor [Nms],
J_{vh} is horizontal angular momentum from main rotor [Nms],
H_v is differential equation $\omega_v = H_v(U_v, t)$,
H_h is differential equation $\omega_h = H_h(U_h, t)$,
G_v is aerodynamical damping torque from main rotor $G_v(\omega_v, \Omega_v)$,
G_h is aerodynamical damping torque from tail rotor $G_h(\omega_h, \Omega_h)$.

Controlling the system consists in stabilising the TRAS beam in an arbitrary (within practical limits) desired position (pith and azimuth) or making it to 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).

3.2 Nonlinear model

The mathematical model is developed with some simplifying assumptions. First, it is assumed that the dynamics of the propeller subsystem can be described by the first order differential equations. Further, it is assumed that friction in the system is of the viscous type. It is assumed also that the propeller-air subsystem could be described in accord with postulates of the flow theory.

The above assumptions allow us to define the problem clearly. First, consider the rotation of the beam in the vertical plane i.e. around the **horizontal axis**. Having in mind that the driving torques are produced by the propellers the rotation can be described in principle as the motion of a pendulum. From the second dynamics law of Newton we obtain:

$$M_v = J_v \frac{d^2\alpha_v}{dt^2} \qquad (1)$$

where: M_v - total moment of forces in the vertical plane,
J_v - the sum of moments of inertia relative to the horizontal axis,
α_v - the pitch angle of the beam.

Then:

$$M_v = \sum_{i=1}^{6} M_{vi}, \qquad J_v = \sum_{i=1}^{8} J_{vi}$$

To determine the moments of forces applied to the beam and making it rotate around the horizontal axis consider the situation shown in Fig. 3.3 .

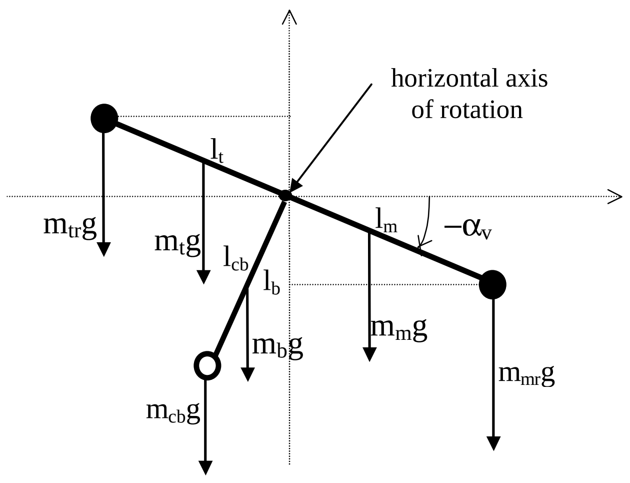


Fig. 3.3 Gravity forces in TRAS corresponding to the return torque which determines the equilibrium position of the system.

$$M_{v1} = g\left\{\left[\left(\frac{m_t}{2}+m_{tr}+m_{ts}\right)l_t-\left(\frac{m_m}{2}+m_{mr}+m_{ms}\right)l_m\right]\cos\alpha_v-\left[\frac{m_b}{2}l_b+m_{cd}l_{cb}\right]\sin\alpha_v\right\}$$

$$M_{v1} = g\left[(A\text{-}B)\ \cos\alpha_v - C\ \sin\alpha_v\right]$$

where:

$$A=\left(\frac{m_t}{2}+m_{tr}+m_{ts}\right)l_t$$

$$B=\left(\frac{m_m}{2}+m_{mr}+m_{ms}\right)l_m$$

$$C=\left[\frac{m_b}{2}l_b+m_{cd}l_{cb}\right]$$

where: M_{v1} is the return torque corresponding to the forces of gravity,

m_{mr} is the mass of the main DC-motor with main rotor,

m_m is the mass of the main part of the beam,

m_{tr} is the mass of the tail motor with tail rotor,

m_t is the mass of the tail part of the beam,

m_{cb} is the mass of the counter-weight,

m_b is the mass of the counter-weight beam,

m_{ms} is the mass of the main shield,

m_{ts} is the mass of the tail shield,

l_m is the length of the main part of the beam,

l_t is the length of the tail part of the beam,

l_b is the length of the counter-weight beam,

l_{cb} is the distance between the counter-weight and the joint.

g is the gravitational acceleration,

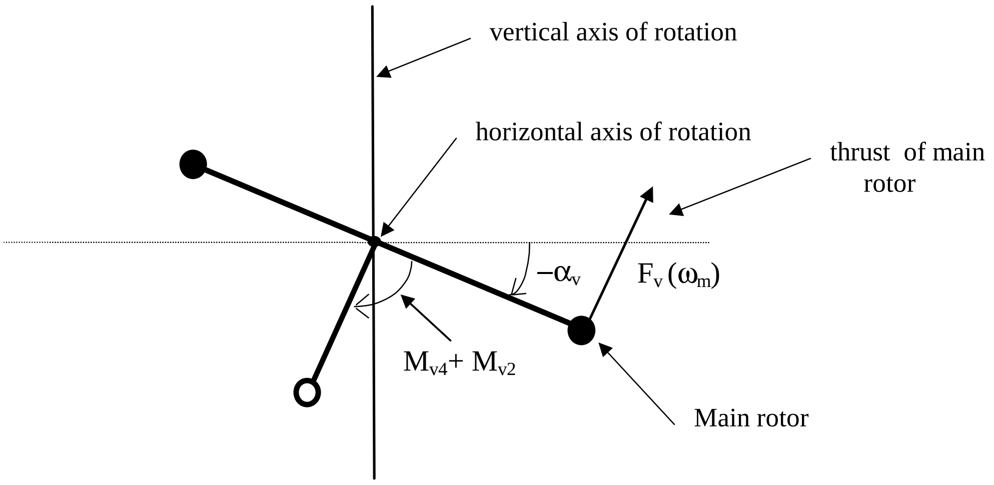


Fig. 3.4 Propulsive force moment and friction moment in TRAS

$$M_{v2} = l_m\, F_v(\omega_m)$$

M_{v2} is the moment of the propulsive force produced by the main rotor,

ω_v is angular velocity of the main rotor,

$F_v(\omega_v)$ denotes the dependence of the propulsive force on the angular velocity of the rotor. It should be measured experimentally (see section 4.5).

$$M_{v3} = -\Omega_h^2\left[\left(\frac{m_m}{2}+m_{mr}+m_{ms}\right)l_m+\left(\frac{m_t}{2}+m_{tr}+m_{ts}\right)l_t+m_{cb}l_{cb}+\frac{m_b}{2}l_b\right]\sin\alpha_v\cos\alpha_v$$

or in the compact form:

$$M_{v3} = -\Omega_h^2\,(A+B+C)\,\sin\alpha_v\cos\alpha_v$$

M_{v3}- is the moment of centrifugal forces corresponding to the motion of the beam around the vertical axis,

and: $\Omega_h = \dfrac{d\alpha_h}{dt}$ (2)

Ω_h- is the angular velocity of the beam around the vertical axis, α_h- is the azimuth angle of the beam.

$$M_{v4} = -\Omega_v\, f_v$$

M_{v4} is the moment of friction depending on the angular velocity of the beam around the horizontal axis.

where: $\Omega_v = \frac{d\alpha_v}{dt}$ (3)

Ω_v is the angular velocity around the horizontal axis,

f_v is a constant,

M_{v5} is the cross moment from U_h, $M_{v5} = U_h k_{hv}$,

k_{hv} is constant,

M_{v6} is the damping torque from rotating propeller $M_{v6} = -a_1 \Omega_v abs(\omega_v)$,

a_1 is constant.

According to Fig. 3.4 we can determine components of the moment of inertia relative to the horizontal axis. Notice that this moment is independent of the position of the beam.

$$J_{v1} = m_{mr}\, l_m^2, \quad J_{v2} = m_m \frac{l_m^2}{3}, \quad J_{v3} = m_{cb}\, l_{cb}^2$$

$$J_{v4} = m_b \frac{l_b^2}{3}, \quad J_{v5} = m_{tr}\, l_t^2, \quad J_{v6} = m_t \frac{l_t^2}{3}$$

$$J_{v7} = \frac{m_{ms}}{2} r_{ms}^2 + m_{ms}\, l_m^2, \qquad J_{v8} = m_{ts}\, r_{ts}^2 + m_{ts}\, l_t^2$$

r_{ms} is the radius of the main shield,

r_{ts} is the radius of the tail shield.

Similarly we can describe the motion of the beam around the **vertical axis**. Having in mind that the driving torques are produced by the rotors and that the moment of inertia depends on the pitch angle of the beam the horizontal motion of the beam (around the vertical axis) can be described in principle as rotative motion of a solid:

$$M_h = J_h \frac{d^2\alpha_h}{dt^2} \tag{4}$$

where: M_h is the sum of moments of forces acting in the horizontal plane,

J_h is the sum of moments of inertia relative to the vertical axis.

Then: $M_h = \sum_{i=1}^{4} M_{hi}$, $J_h = \sum_{i=1}^{8} J_{hi}$

To determine the moments of forces applied to the beam and making it rotate around the vertical axis consider the situation shown in Fig. 3.5.

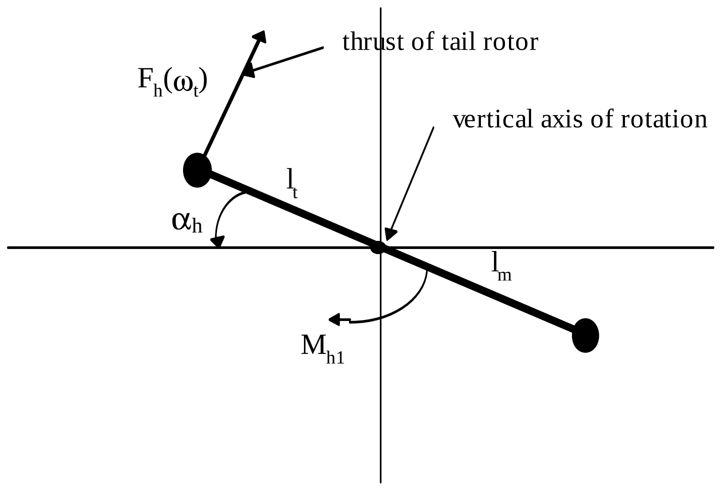


Fig. 3.5 Moments of forces in horizontal plane (as seen from above)

$$M_{h1} = l_t\ F_h(\omega_t)\ \cos\alpha_v$$

ω_t is angular velocity of tail rotor, $F_h(\omega_t)$- denotes the dependence of the propulsive force on the angular velocity of the tail rotor (should be determine experimentally, see section 4.5)

$$M_{h2} = -\Omega_h f_h$$

M_{h2} is the moment of friction depending on the angular velocity of the beam around the vertical axis,
f_h is constant,
M_{h3} is the cross moment from U_v, $M_{h3} = U_v k_{vh}$,
k_{vh} is constant,
M_{h4} is the damping torque from rotating propeller, $M_{h4} = -a_2 \Omega_h abs(\omega_h)$,
a_2 is constant.

According to Fig. 3.5. we can determine components of the moment of inertia relative to vertical axis (it depends on pitch position of the beam).

$$J_{h1} = \frac{m_m}{3}\left(l_m \cos\alpha_v\right)^2, \qquad J_{h2} = \frac{m_t}{3}\left(l_t \cos\alpha_v\right)^2,$$
$$J_{h3} = \frac{m_b}{3}\left(l_b \sin\alpha_v\right)^2, \qquad J_{h4} = m_{tr}\left(l_t \cos\alpha_v\right)^2,$$
$$J_{h5} = m_{mr}\left(l_m \cos\alpha_v\right)^2, \qquad J_{h6} = m_{cb}\left(l_{cb} \sin\alpha_v\right)^2$$
$$J_{h7} = \frac{m_{ts}}{2} r_{ts}^2 + m_{ts}\left(l_t \cos\alpha_v\right)^2, J_{h8} = m_{ms} r_{ms}^2 + m_{ms}\left(l_m \cos\alpha_v\right)^2$$

or in the compact form:

$$J_h = E\ \cos^2\alpha_v + D\ \sin^2\alpha_v + F$$

where: *D*,*E*,*F* are constants:

$$D = \frac{m_b}{3} l_b^2 + m_{cb} l_{cb}^2 ,$$

$$E = \left(\frac{m_m}{3} + m_{mr} + m_{ms} \right) l_m^2 + \left(\frac{m_t}{3} + m_{tr} + m_{ts} \right) l_t^2 ,$$

$$F = m_{ms} r_{ms}^2 + \frac{m_{ts}}{2} r_{ts}^2$$

$$M_{v1} = g \left\{ (A\text{-}B) \cos \alpha_v - C \sin \alpha_v \right\}$$

Using (1)-(4) we can write the equations describing the motion of the system as follows:

$$\frac{d\Omega_v}{dt} = \frac{l_m F_v(\omega_m) - \Omega_v k_v + g\big((A-B) \cos \alpha_v - C \sin \alpha_v\big)}{J_v} \dots \qquad (5)$$

$$\dots \frac{-\frac{1}{2}\Omega_h^2 (A+B+C) \sin 2\alpha_v U_h k_{hv} + U_h k_{hv} - a_1 \Omega_v abs(\omega_v)}{J_v}$$

$$\frac{d\alpha_v}{dt} = \Omega_v \ , \qquad (6)$$

$$\frac{dK_h}{dt} = \frac{M_h}{J_h} = \frac{l_t F_h(\omega_t) \cos \alpha_v - \Omega_h k_h + U_v k_{vh} - a_2 \Omega_h abs(\omega_h)}{D \sin^2 \alpha_v + E \cos^2 \alpha_v + F} \qquad (7)$$

$$\frac{d\alpha_h}{dt} = \Omega_h , \qquad \Omega_h = \frac{K_h}{J_h(\alpha_v)} \qquad (8)$$

and two equations describing the motion of rotors:

$$I_h \frac{d\omega_h}{dt} = U_h - H_h^{-1}(\omega_h) \ \text{ and } \ I_v \frac{d\omega_v}{dt} = U_v - H_v^{-1}(\omega_v)$$

I_h is moment of inertia of the tail rotor,

I_v is moment of inertia of the main rotor.

The above model of the motor-propeller dynamics is obtained by substituting the nonlinear system by a serial connection of a linear dynamic system and static nonlinearity.

3.3 State equations

Finally, the mathematical model of TRAS (compare Fig. 3.2) becomes the set of four nonlinear differential equations with two linear differential equations and four nonlinear functions.

$$U=\begin{bmatrix} u_h \\ u_v \end{bmatrix} \text{ is the input, } X=\begin{bmatrix} K_\mathrm{h} \\ \alpha_\mathrm{h} \\ \omega_\mathrm{h} \\ K_\mathrm{v} \\ \alpha_\mathrm{v} \\ \omega_\mathrm{v} \end{bmatrix} \text{ is the state, and } Y=\begin{bmatrix} \Omega_\mathrm{h} \\ \alpha_\mathrm{h} \\ \omega_\mathrm{t} \\ \Omega_\mathrm{v} \\ \alpha_\mathrm{v} \\ \omega_\mathrm{v} \end{bmatrix} \text{ is the output vector.}$$

In order to apply the model for control of TRAS the parameters and nonlinear functions should be determined first. They can be divided into three groups:

- physical parameters,
- nonlinear static characteristics,
- time constants of the linear part of the model.

They are described in details in the next section.

3.4 Physical parameters

To obtain the values of model coefficients it is necessary to perform some measurements. First, geometrical dimensions and moving masses of TRAS should be measured. There are the following results of measurements for a given laboratory TRAS set-up.

		$m_{tr}=0.225$ [kg]
$l_t=0.216$	[m]	$m_{mr}=0.252$ [kg]
$l_m=0.202$	[m]	$m_{cb}=0.0256$ [kg]
$l_b=0.145$	[m]	$m_t=0.032$ [kg]
$l_{cb}=0.15$	[m]	$m_m=0.03$ [kg]
$r_{ms}=0.145$	[m]	$m_b=0.01$ [kg]
$r_{ts}=0.10$	[m]	$m_{ts}=0.061$ [kg]
		$m_{ms}=0.083$ [kg]

Using the above measurements the moment of inertia about the horizontal axis can be calculated as:

$$J_v=\sum_i^8 J_{iv} \;=\; 0.0307\ [\mathrm{kg\,m^2}]\,.$$

The terms of the sum are calculated from elementary physics laws:

$$J_{v1} = m_{mr} l_m^2 \quad = 0.0103 \ [\mathrm{kg\,m^2}]$$
$$J_{v2} = m_m l_m^2/3 \quad = 0.00040 \quad [\mathrm{kg\,m^2}]$$
$$J_{v3} = m_{cb} l_{cb}^2 \quad = 0.00057 [\mathrm{kg\,m^2}]$$
$$J_{v4} = m_b l_b^2/3 \quad = \ 0.0007 \ [\mathrm{kg\,m^2}]$$
$$J_{v5} = m_{tr} l_t^2 \quad = 0.0105 \ [\mathrm{kg\,m^2}]$$
$$J_{v6} = m_t l_t^2/3 \quad = 0.00049 \ [\mathrm{kg\,m^2}]$$
$$J_{v7} = m_{ms}(r_{ms}^2/2 + l_m^2) \quad = 0.0043 \quad [\mathrm{kg\,m^2}]$$
$$J_{v8} = m_{ts}(r_{ts}^2/2 + l_t^2) \quad = \ 0.0035 \quad [\mathrm{kg\,m^2}]$$

The calculated moment of inertia about the vertical axis is:

$$J_h = \sum_i^8 J_{hi} \ ,$$

where the terms of the sum are:

$$J_{h1} = m_t (l_t \cos\alpha_v)^2/3 \quad = 0.00049 \ \cos^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h2} = m_m (l_m \cos\alpha_v)^2/3 \quad = 0.0004 \ \cos^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h3} = m_b (l_b \sin a_v)^2/3 \quad = 0.0007 \ \sin^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h4} = m_{mr} (l_m \cos\alpha_v)^2 \quad = 0.0103 \ \cos^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h5} = m_{tr} (l_t \cos\alpha_v)^2 \quad = 0.0105 \ \cos^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h6} = m_{cb} (l_{cb} \sin \ \alpha_v)^2 \quad = 0.00057 \quad \sin^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h7} = m_{ts} \left(r_{ts}^2/2 + l_t^2 \cos^2\alpha_v\right) \quad = 0.0003 \ + \ 0.0028 \ \cos^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$
$$J_{h8} = m_{ms} \left(r_{ms}^2/2 + l_m^2 \cos^2\alpha_v\right) = 0.00087 \ + \ 0.0034 \ \cos^2\alpha_v \qquad [\mathrm{kg\,m^2}]$$

giving finally (Fig. 3.6):

$$J_h = \sum_i^9 J_{hi} \ = E\cos^2\alpha_v + D\sin^2\alpha_v + \ F = 0.0279 \ \cos^2\alpha_v + 0.0013 \ \sin^2\alpha_v + \ 0.0021 .$$

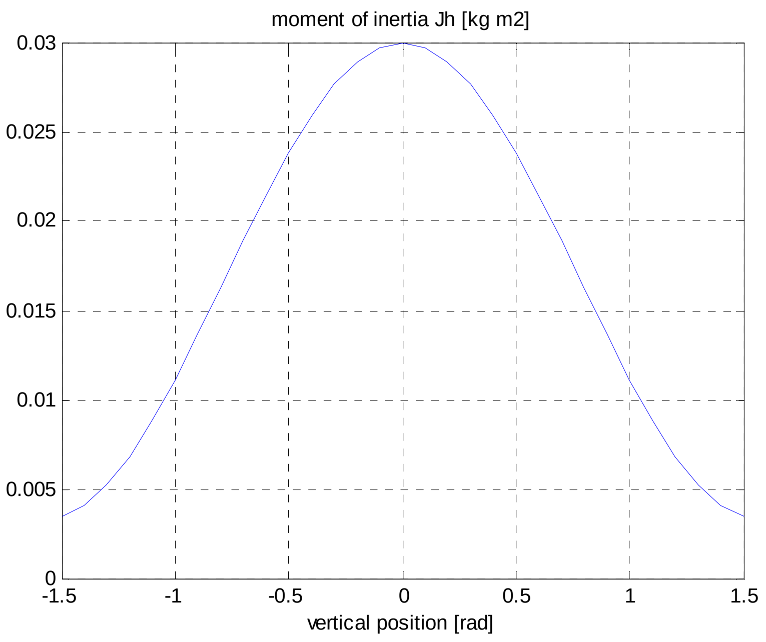


Fig. 3.6 Moment of inertia J_h

The returning torque from gravity forces is expressed by

$$M_r = \sum_i^8 M_{ri} ,$$

and its components are given by:

$$\begin{aligned}
M_{r1} &= -9.81\, m_m l_m \cos\alpha_v / 2 &&= -0.0297 \cos\alpha_v \text{ [N m]}\\
M_{r2} &= -9.81\, m_{mr} l_m \cos\alpha_v &&= -0.4994 \cos\alpha_v \text{ [N m]}\\
M_{r3} &= -9.81\, m_{ms} l_m \cos\alpha_v &&= -0.1645 \cos\alpha_v \text{ [N m]}\\
M_{r4} &= +9.81\, m_t l_t \cos\alpha_v / 2 &&= 0.0339 \cos\alpha_v \text{ [N m]}\\
M_{r5} &= +9.81\, m_{tr} l_t \cos\alpha_v &&= 0.4768 \cos\alpha_v \text{ [N m]}\\
M_{r6} &= +9.81\, m_{ts} l_t \cos\alpha_v &&= 0.1293 \cos\alpha_v \text{ [N m]}\\
M_{r7} &= -9.81\, m_b l_b \sin\alpha_v / 2 &&= 0.0711 \sin\alpha_v \text{ [N m]}\\
M_{r8} &= -9.81\, m_{cb} l_{cb} \sin\alpha_v &&= 0.0377 \sin\alpha_v \text{ [N m]}
\end{aligned}$$

giving finally (Fig. 3.7)

$$M_r = \sum_i^8 M_{ri} = (-0.0536 \cos\alpha_v + 0.1088 \sin\alpha_v) \text{ [N m]}.$$

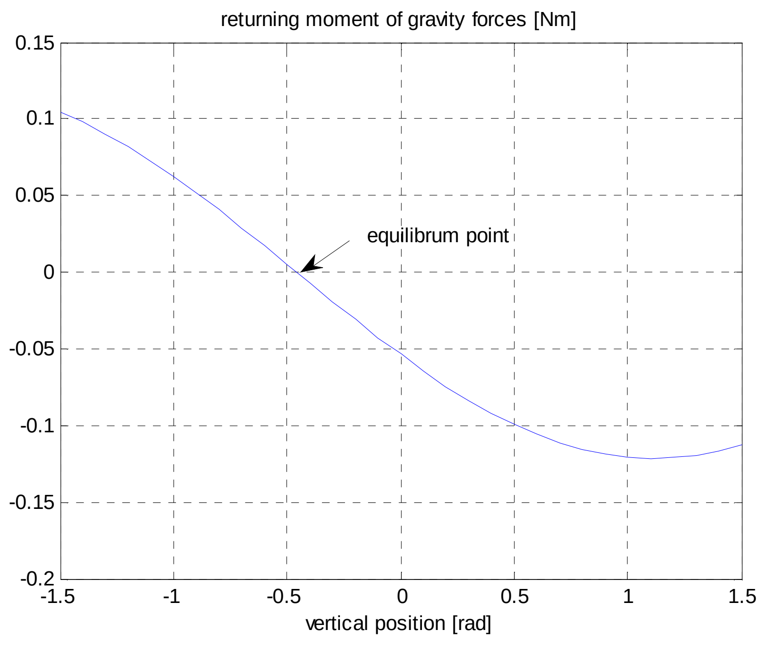


Fig. 3.7 Returning moment of the gravity forces M_r

The moment of the centrifugal forces is:

$$M_{cf} = \sum_{i}^{6} M_{cfi} \ ,$$

where

$$M_{cf1} = (m_{tr}+m_{ts})l_t^2\ \Omega_h^2\ \cos\alpha_v \sin\alpha_v \quad =0.0133\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v \quad [\mathrm{N\,m}]$$
$$M_{cf2} = m_t l_t^2\ \Omega_h^2\ \cos\alpha_v \sin\alpha_v/4 \quad =\ 0.00037\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v \quad [\mathrm{N\,m}]$$
$$M_{cf3} = m_b l_b^2\ \Omega_h^2\ \cos\alpha_v \sin\alpha_v/4 \quad =0.00052\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v \quad [\mathrm{N\,m}]$$
$$M_{cf4} = m_{cb} l_{cb}^2\ \Omega_h^2\ \cos\alpha_v \sin\alpha_v \quad =0.00057\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v \quad [\mathrm{N\,m}]$$
$$M_{cf5} = m_m l_m^2\ \Omega_h^2\ \cos\alpha_v \sin\alpha_v/4 \quad =0.0003\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v \quad [\mathrm{N\,m}]$$
$$M_{cf6} = (m_{mr}+m_{ms})l_m^2\ \Omega_h^2\ \cos\alpha_v \sin\alpha_v \quad =0.0137\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v \quad [\mathrm{N\,m}]$$

giving finally (Fig. 3.8)

$$M_{cf} = \sum_{i}^{6} M_{cfi}\ =\ 0.02876\ \ \Omega_h^2 \cos\alpha_v\ \sin\alpha_v\ \ [\mathrm{N\,m}]\,.$$

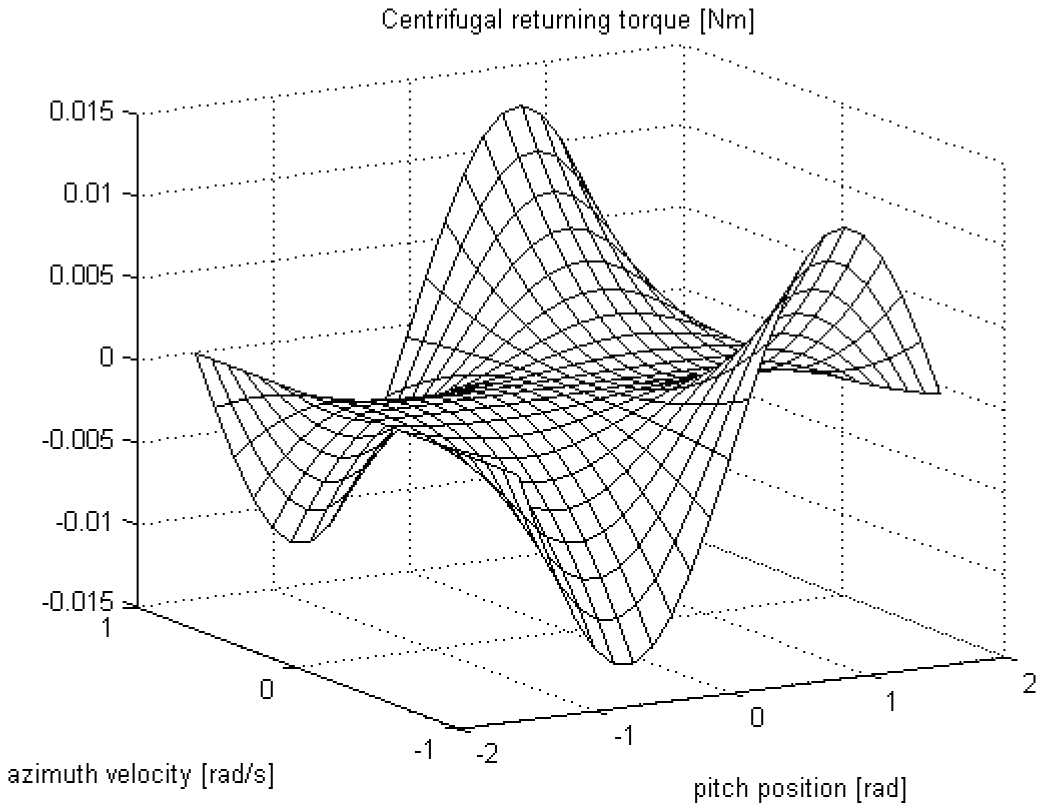


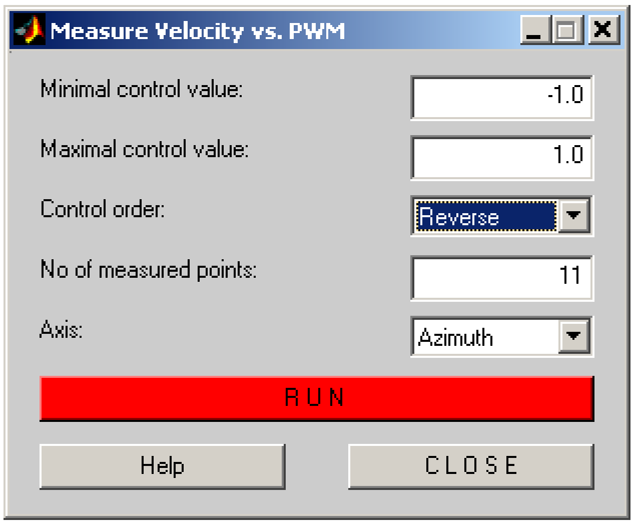
Fig. 3.8 Centrifugal returning torque

3.5 Static characteristics

It is necessary to identify the following functions:

- Two nonlinear input characteristics determining dependence of the DC-motor rotational speed versus the input voltage (RPM characteristics): $\omega_v = H_v(U_v)$ and $\omega_h = H_h(U_h)$

To measure the characteristics double click the *Static characteristics* button in *TRAS Control Window*. The window given in Fig. 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 corresponds to the input voltage range [-24V, +24V].



Minimal control value: -1.0

Maximal control value: 1.0

Control order: Reverse

No of measured points: 11

Axis: Azimuth

Fig. 3.How to collect the points of the static characteristic

Block the beam before the experiment is started. Choose *Azimuth* axis (tail rotor) and click the *Run* button. The constant value of the control activates the DC motor so long as a steady state of the shaft angular velocity is achieved. 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. Simultaneously, the measurements are displayed in the screen. 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. 3.9.

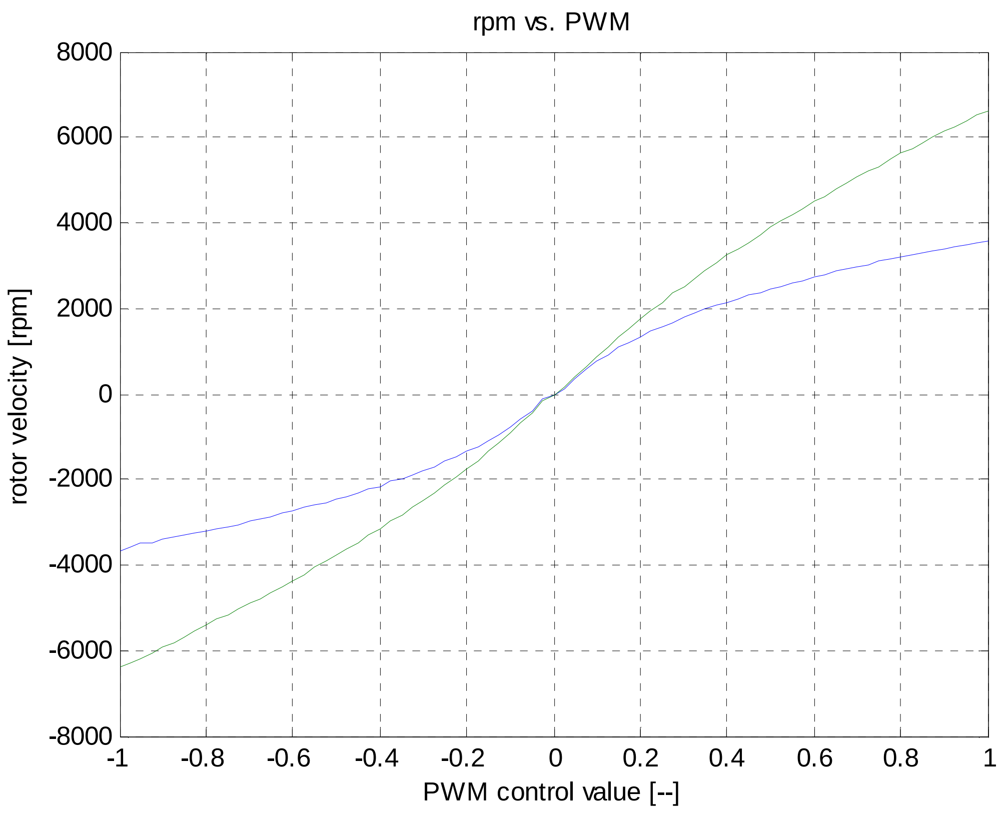


Fig. 3.9 Main and tail rotor static characteristics

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

- Two nonlinear 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 (not delivered with the system) is necessary to measure the force created by rotational movements of the propellers. The characteristics included in *TRAS Toolbox* and shown in this section has been obtained by the manufacturer of TRAS.

3.5.1 Main rotor thrust characteristics

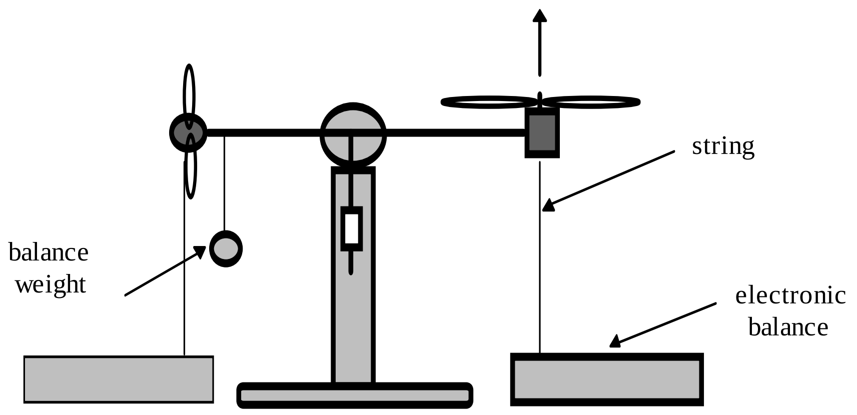


Fig. 3.10 Measuring of the main rotor thrust characteristics

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. 3.10) .

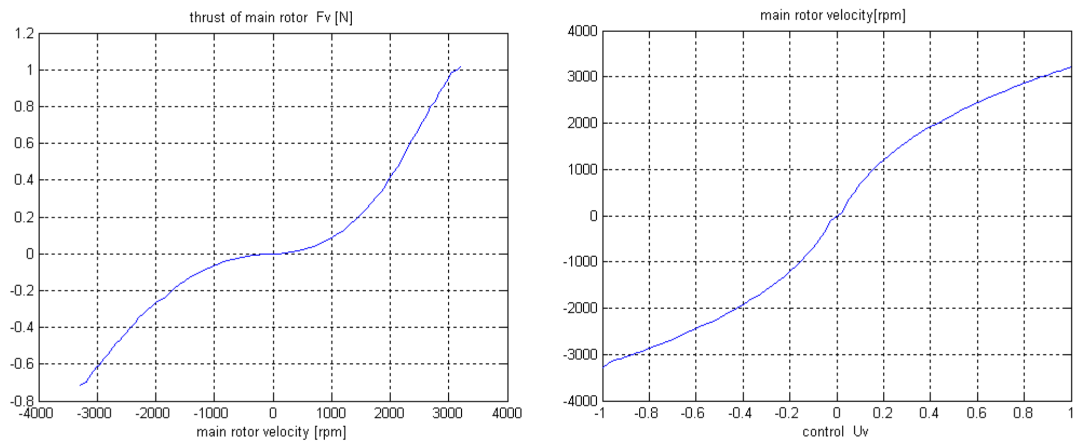


Fig. 3.11 Measured static thrust characteristics of the main rotor

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:

$$\tilde{F}_v = -1.8\cdot 10^{-18}\,\omega_v^5 - 7.8\cdot 10^{-16}\,\omega_v^4 + 4.1\cdot 10^{-11}\,\omega_v^3 + 2.7\cdot 10^{-8}\,\omega_v^2 + 3.5\cdot 10^{-5}\,\omega_v - 0.014$$
$$\tilde{\omega}_v = -5.2\cdot 10^3\,U_v^7 - 1.1\cdot 10^2\,U_v^6 + 1.1\cdot 10^4\,U_v^5 + 1.3\cdot 10^2\,U_v^4 - 9.2\cdot 10^3\,U_v^3 - 31U_v^2 + 6.1\cdot 10^3\,U_v - 4.5$$

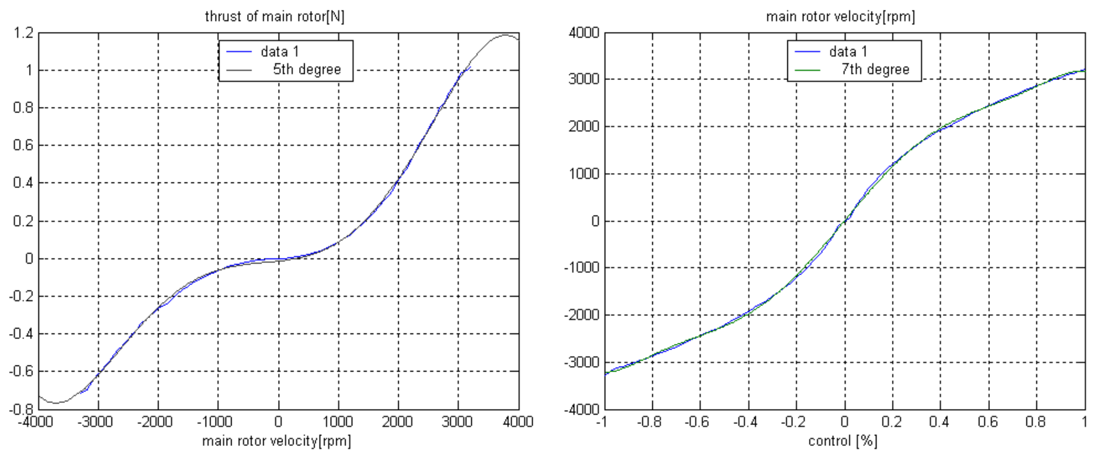


Fig. 3.12 Polynomial approximation of the main rotor characteristics

3.5.2 Tail rotor thrust characteristics

Fig. 3.13 shows laboratory set-up for measuring thrust of the tail rotor.

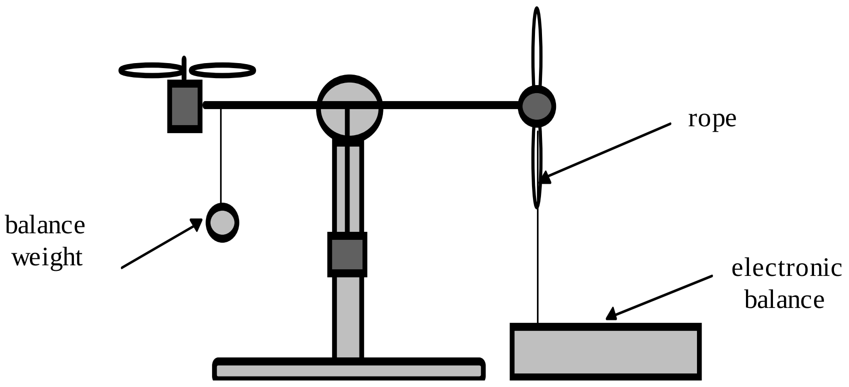


Fig. 3.13 Laboratory set-up for the tail rotor thrust characteristics

To measure the static thrust characteristics one should to rearrange the laboratory set-up as shown in Fig. 3.13 and the electronic balance should be used.

The measured by the manufacturer of TRAS thrust static characteristics of the tail motor are given in Fig. 3.14.

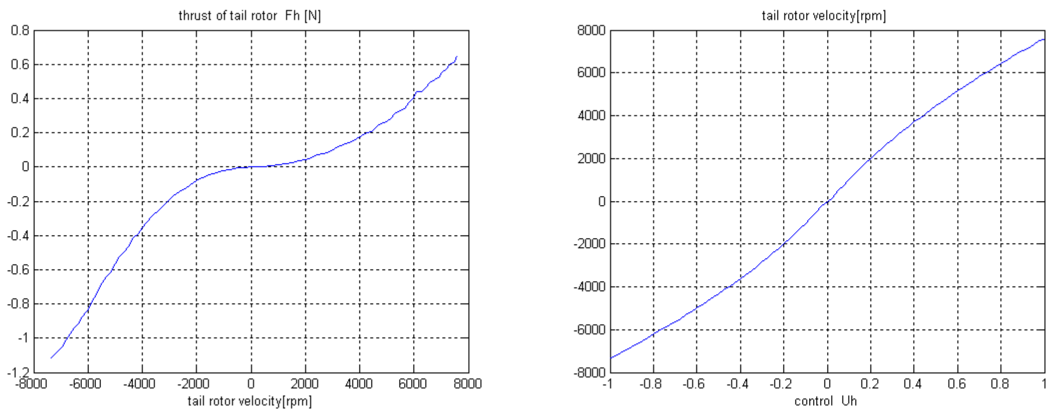


Fig. 3.14 Thrust characteristics measured for the tail rotor.

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:

$$\tilde{F}_h = -2.6\cdot10^{-20}\omega_h^5 + 4.1\cdot10^{-17}\omega_h^4 + 3.2\cdot10^{-12}\omega_h^3 - 7.3\cdot10^{-9}\omega_h^2 + 2.1\cdot10^{-5}\omega_v + 0.0091$$
$$\tilde{\omega}_h = 2.2\cdot10^3 U_h^5 - 1.7\cdot10^2 U_v^4 - 4.5\cdot10^3 U_v^3 + 3\cdot10^2 U_v^2 + 9.8\cdot10^3 U_v - 9.2$$

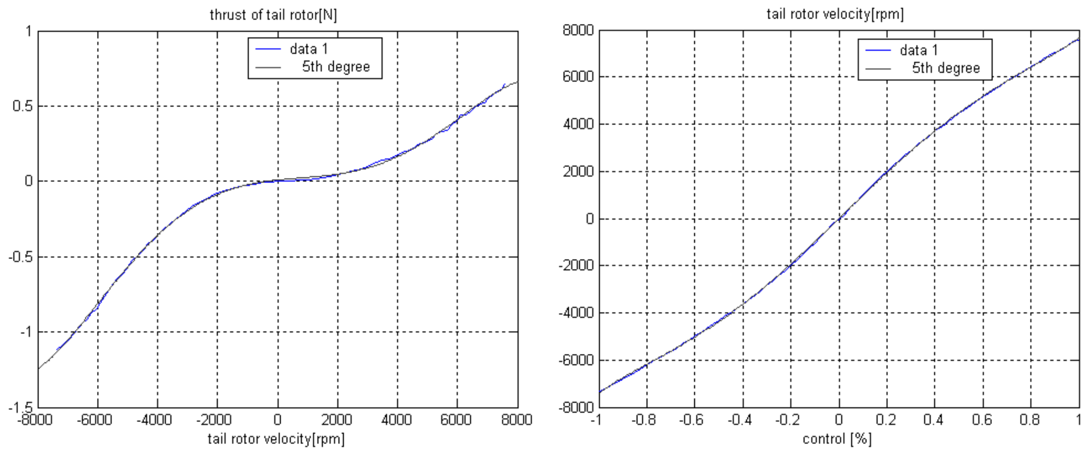


Fig. 3.15 Polynomial approximation of tail rotor characteristics

4. RTWT model

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

⇨ **Before start, test your MATLAB configuration by building and running an example of a real-time application. Real-time Windows Target Toolbox includes the rtvdp.mdl model. Running this model will test the installation of the Real-Time Workshop, Real-Time Windows Target toolboxes and the Real-Time Windows Target kernel.**
In the MATLAB window, type
rtvdp
Next, build and run the real-time model.

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 RTWT,
- start the real-time code from the *Simulation/Start real-time code* pull-down menu. In this way the system runs in real-time.

4.2 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 *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. 4.1.

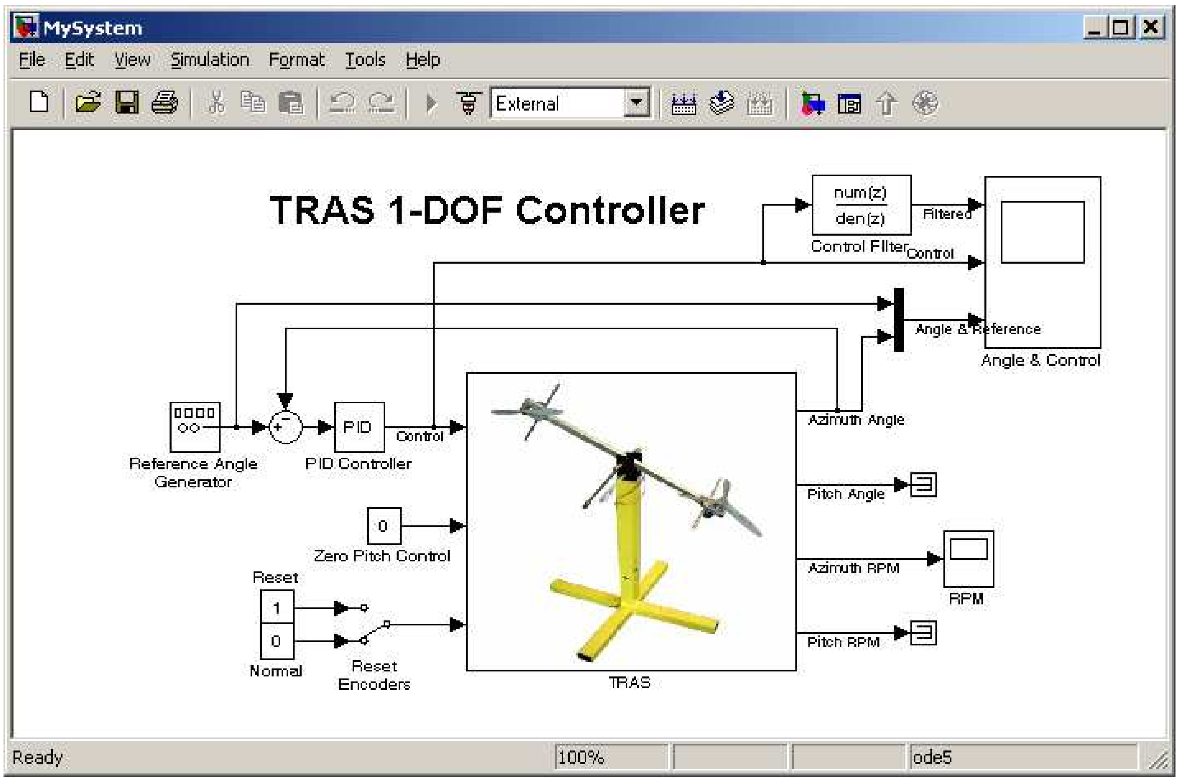
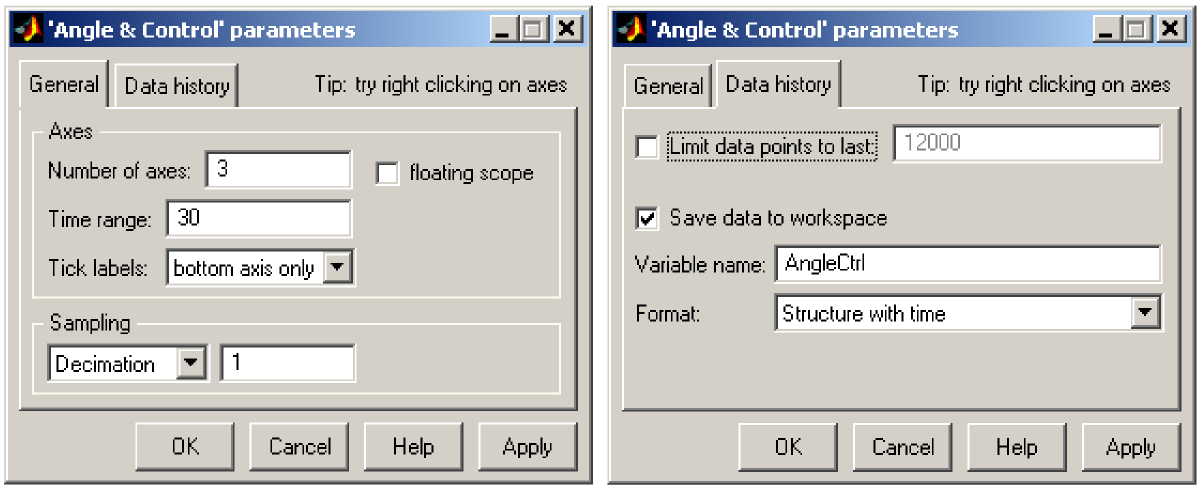


Fig. 4.1 The *MySystem* Simulink model

Now, you can modify the model. You have absolute freedom to develop your own controller. Remember to leave the *TRAS* device driver block in the window. This is necessary to work in RTWT 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 amaze at the saturation of controls. Other blocks remaining in the window are not necessary for our new project.
Creating your own model on the basis of an old example ensures that all internal options of the model are set properly. These options are required to proceed with compiling and linking in a proper way. To put the *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 or RTWT references manual).
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. 4.2). 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.


'Angle & Control' parameters

General | Data history | Tip: try right clicking on axes

Axes
Number of axes: 3 | ☐ floating scope
Time range: 30
Tick labels: bottom axis only

Sampling
Decimation | 1

OK | Cancel | Help | Apply

'Angle & Control' parameters

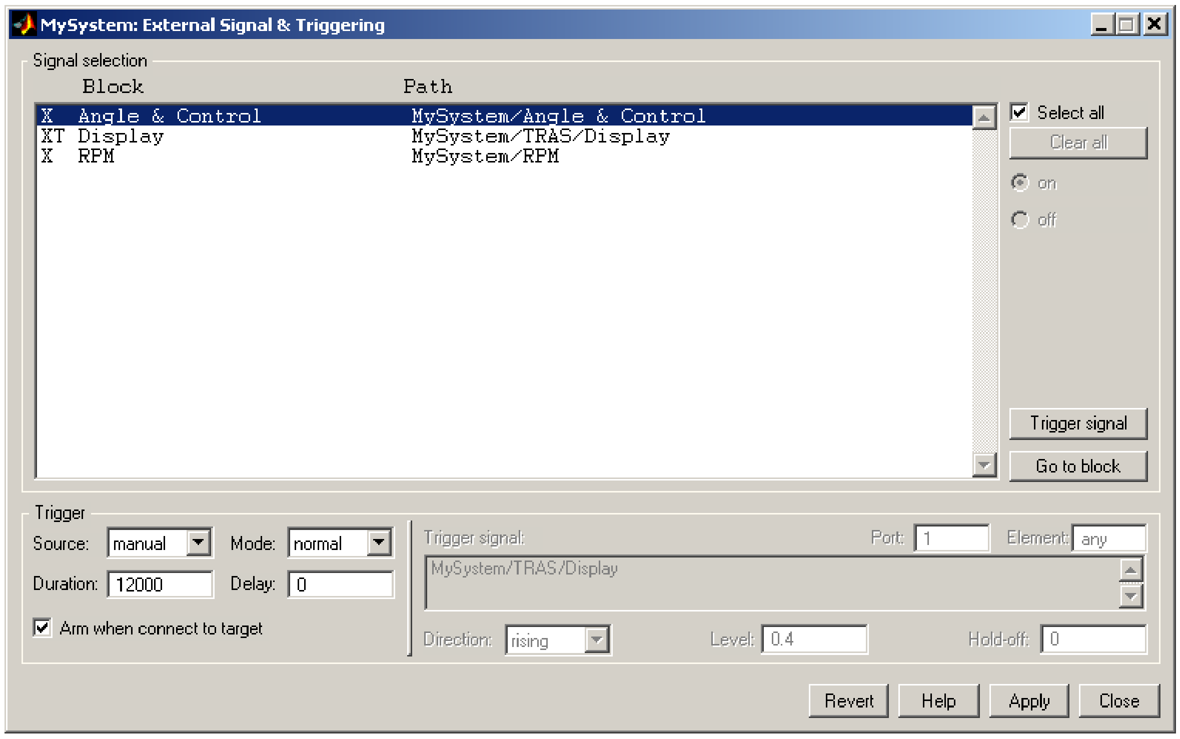
General | Data history | Tip: try right clicking on axes

☐ Limit data points to last: 12000
☑ Save data to workspace
Variable name: AngleCtrl
Format: Structure with time

OK | Cancel | Help | Apply

Fig. 4.2 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. 4.3 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.


MySystem: External Signal & Triggering

Signal selection

```
    Block                         Path
X   Angle & Control               MySystem/Angle & Control
XT  Display                       MySystem/TRAS/Display
X   RPM                           MySystem/RPM
```

☑ Select all
Clear all
◉ on
○ off
Trigger signal
Go to block

Trigger
Source: manual | Mode: normal
Duration: 12000 | Delay: 0
☑ Arm when connect to target

Trigger signal: | Port: 1 | Element: any
MySystem/TRAS/Display
Direction: rising | Level: 0.4 | Hold-off: 0

Revert | Help | Apply | Close

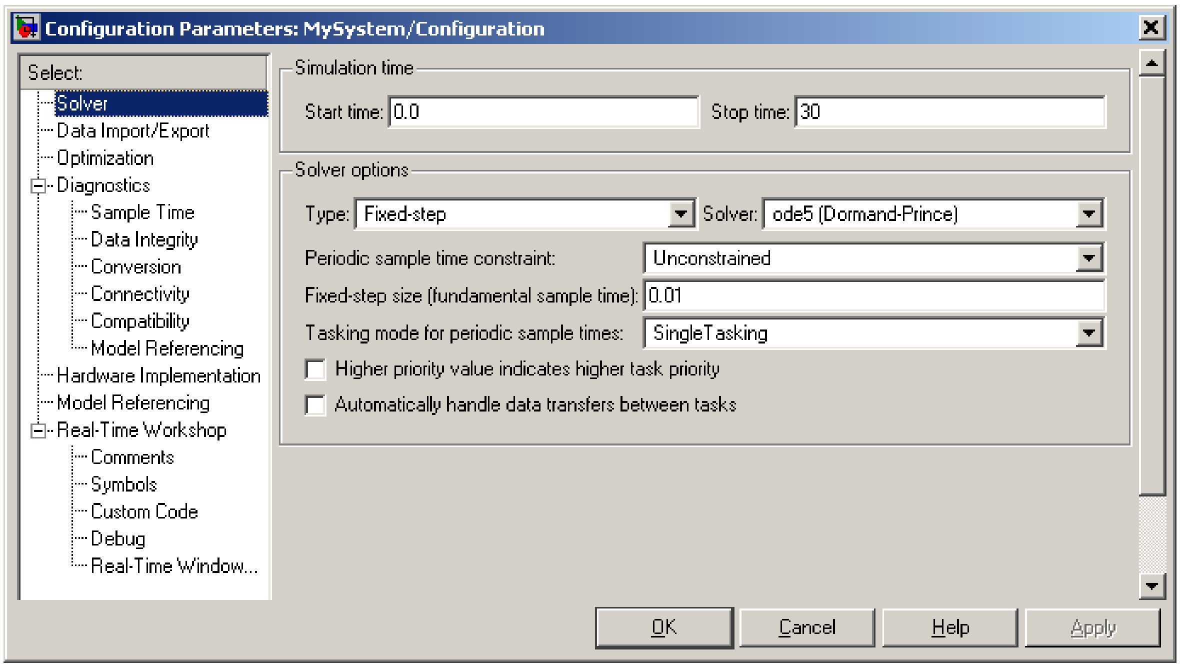
Fig. 4.3 The *External Signal & Triggering* window

4.3 Code generation and the build process

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

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

First, you have to specify the simulation parameters of your Simulink model in the *Simulation parameters* dialog box (Fig. 4.4). The *Real-Time Workshop* and *Solver* tabs contain critical parameters.


Configuration Parameters: MySystem/Configuration

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Integrity
  - Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
- Hardware Implementation
- Model Referencing
- Real-Time Workshop
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Real-Time Window...

Simulation time

Start time: 0.0 Stop time: 30

Solver options

Type: Fixed-step Solver: ode5 (Dormand-Prince)

Periodic sample time constraint: Unconstrained

Fixed-step size (fundamental sample time): 0.01

Tasking mode for periodic sample times: SingleTasking

☐ Higher priority value indicates higher task priority

☐ Automatically handle data transfers between tasks

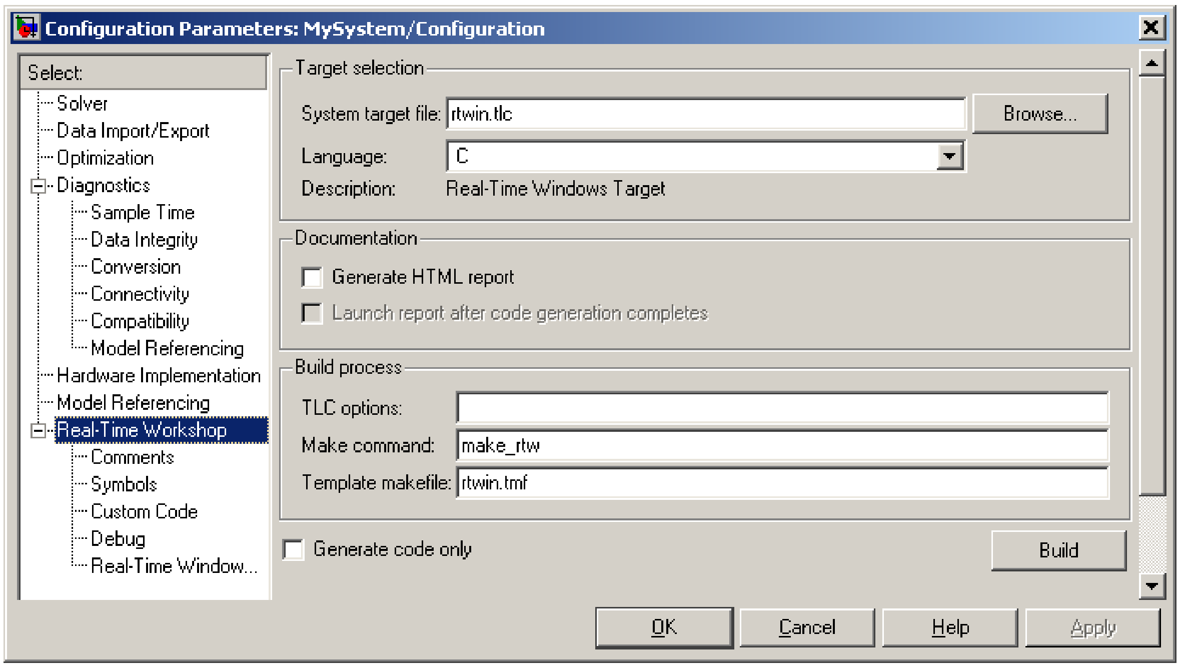
Fig. 4.4 *Solver* tab

The *Solver* tab allows you to set the simulation parameters. Several parameters and options are available in the window. The *Fixed-step size* editable text box is set to 0.01 (this is the sampling period in seconds).
The *Start time* has to be set to 0 (Fig. 4.4). The solver method has to be selected. In our example the fifth-order integration method – *ode5* is chosen. The *Stop time* field defines the length of the experiment. This value may be set to a large number. Each experiment can be terminated by pressing the *Stop real-time code* button.

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

Third party compiler is not requested. The built-in Open Watcom compiler is used to create real-time executable code for RTWT.
The RTW tag is shown in Fig. 4.5. The system target file name is *rtwin.tlc*. It manages the code generation process. Notice, that *rtwin.tmf* template makefile is used. This file is default one for RTWT building process.

-



Configuration Parameters: MySystem/Configuration

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Integrity
  - Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
- Hardware Implementation
- Model Referencing
- Real-Time Workshop
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Real-Time Window...

Target selection

System target file: rtwin.tlc

Language: C

Description: Real-Time Windows Target

Documentation

[ ] Generate HTML report

[ ] Launch report after code generation completes

Build process

TLC options:

Make command: make_rtw

Template makefile: rtwin.tmf

[ ] Generate code only

Fig. 4.5 *Configuration parameters/Real-Time Workshop* tab

If all the parameters are set properly you can start the real-time executable building process. For this purpose press the *Build* push button at the Real Time Workshop tag (Fig. 4.5), or simply “CRTL + B”. Successful compilation and linking processes generate the following message:

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

Otherwise, an error message is displayed in the MATLAB Command Window.
In this case check again your MATLAB configuration and simulation parameters.

5. 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 TRAS is not available. The effectiveness of the PID controllers discussed here is illustrated by control experiments.

PID controllers

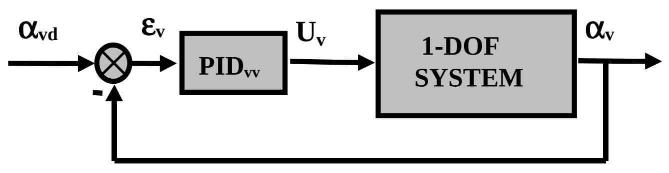
One degree of freedom (1-DOF) control problem is the following. Design a controller that will stabilise the system, or make it follow a desired trajectory in one plane (one degree of freedom) while motion in the other plane is blocked mechanically or being controlled by another controller.
If TRAS is 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. 5.15b. 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.

5.2 1-DOF controllers

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

5.2.1 Vertical 1-DOF control

At the beginning we restrict our control objective to stabilising the system in the vertical plane only. We reduce the original system to the 1-DOF system by mechanically fixing (using the included clamp) its freedom to move in the horizontal plane. A corresponding block diagram of the PID control system is shown in Fig. 5.1.



α_{vd} - desired pitch

Fig. 5.1 1-DOF pitch control system

The block diagram below shows the system in a more detailed form (Fig. 5.2). Notice, that only the vertical part of the control system is considered.

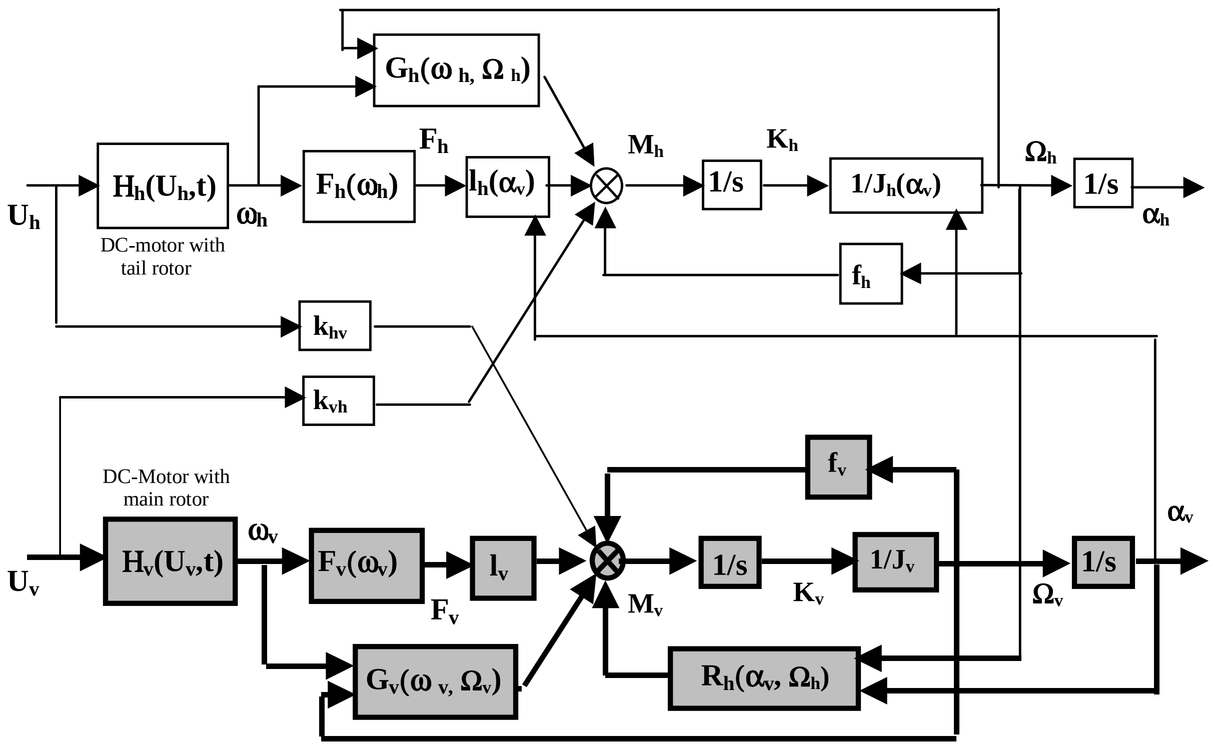


Fig. 5.2 The block diagram 1-DOF system (vertical plane)

5.2.2 Real-time 1-DOF pitch control experiment

Fix TRAS 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 damped. In *Tras Control Window* double click the *Reset Encoders* block.

Click the *PID Pitch controller* button and the model shown in Fig. 5.3 opens. Set all PID controller coefficients as: $K_p = 0.6784$ $K_i = 0.4415$ and $K_d = 1.31196$. Also set saturation of the integral part of the controller to 1.43. Build the model and click on the *Simulation*/*Connect to target* option and *Start real-time code* option.

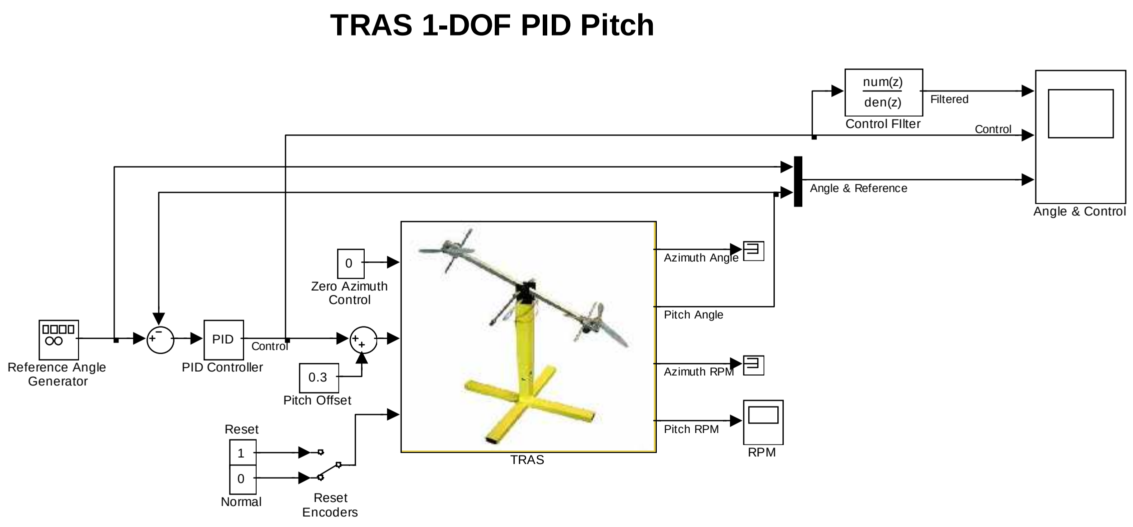


Fig. 5.3 Real-time model for the pitch control

The results of the experiment are shown in Fig. 5.4. Notice, that control changes with high frequency. This phenomena appears due to the quantization effects of the signal caused by the differential part of the controller. For this reason the control signal is filtered to obtaining an average value of the control signal. It is shown in the upper part of Fig. 5.4.

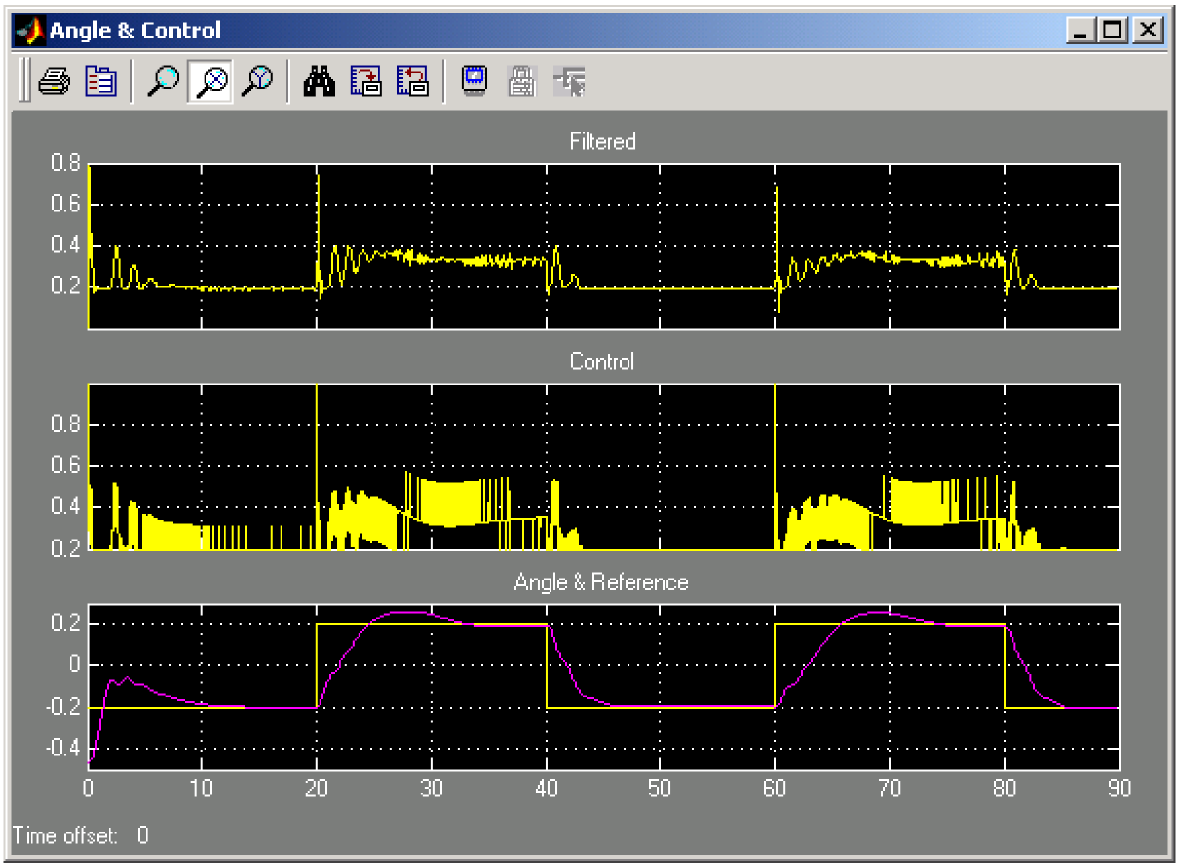


Fig. 5.4 Results of the PID pitch control

The details of the above experiment are shown in Fig. 5.5, Fig. 5.6 and Fig. 5.7.

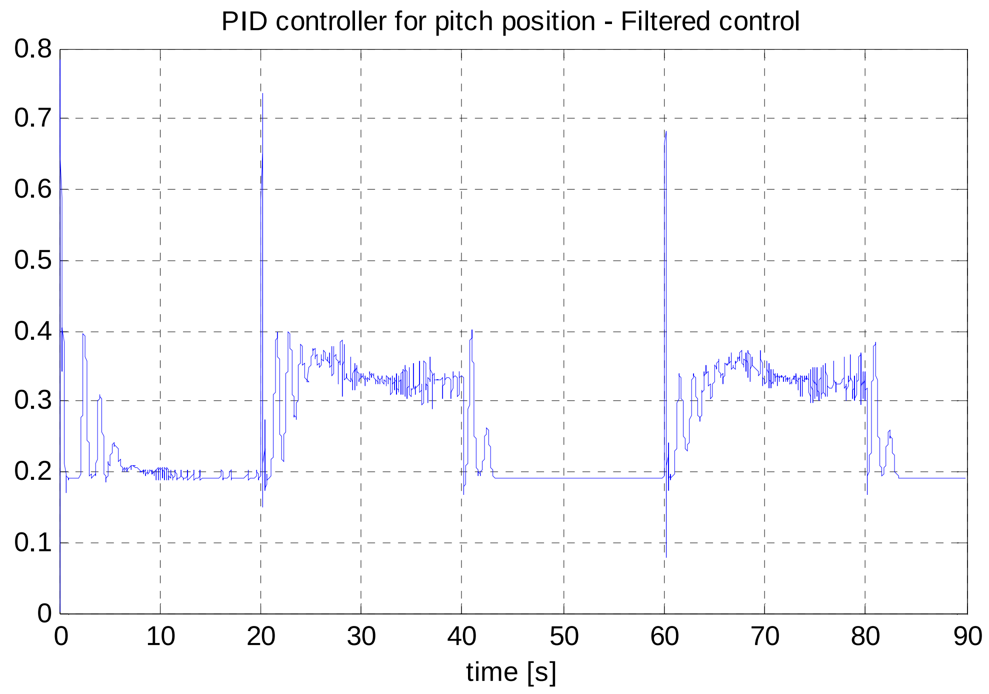


Fig. 5.5 The filtered control

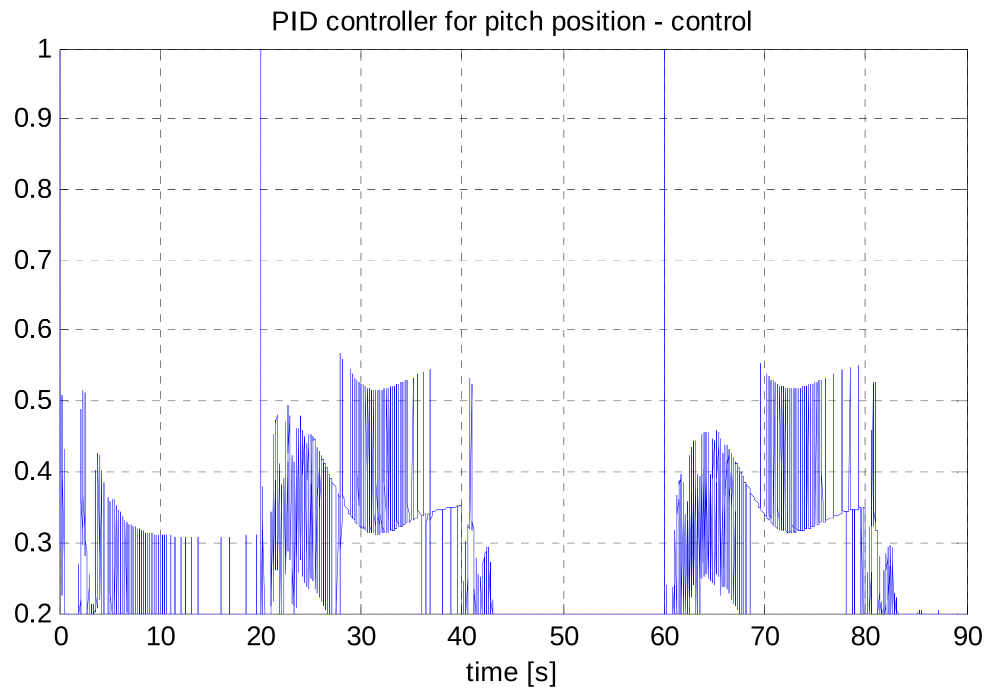


Fig. 5.6 The non filtered control

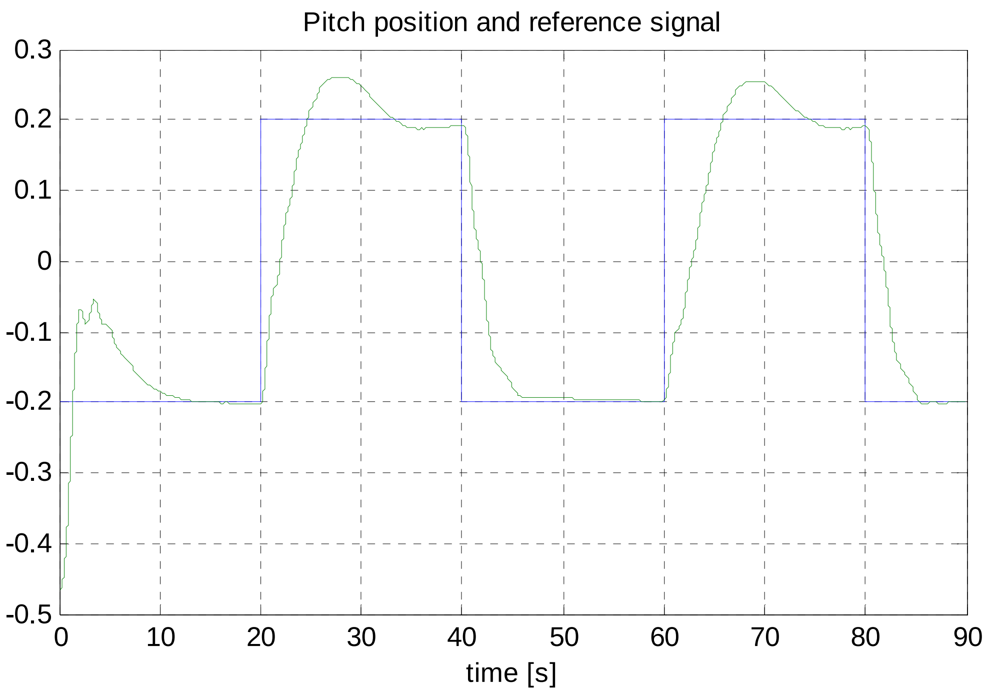
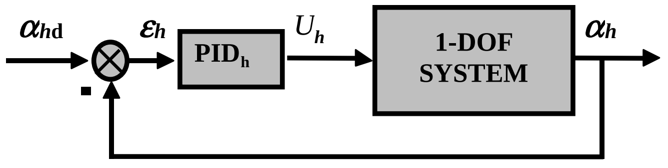


Fig. 5.7 Pitch position and the reference signal

5.2.3 Horizontal 1-DOF control

In the next experiment we 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. 5.8, and in a more detailed form in Fig. 5.9.



α_{hd} -desired azimuth

Fig. 5.8 1-DOF control closed-loop system (azimuth stabilisation)

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

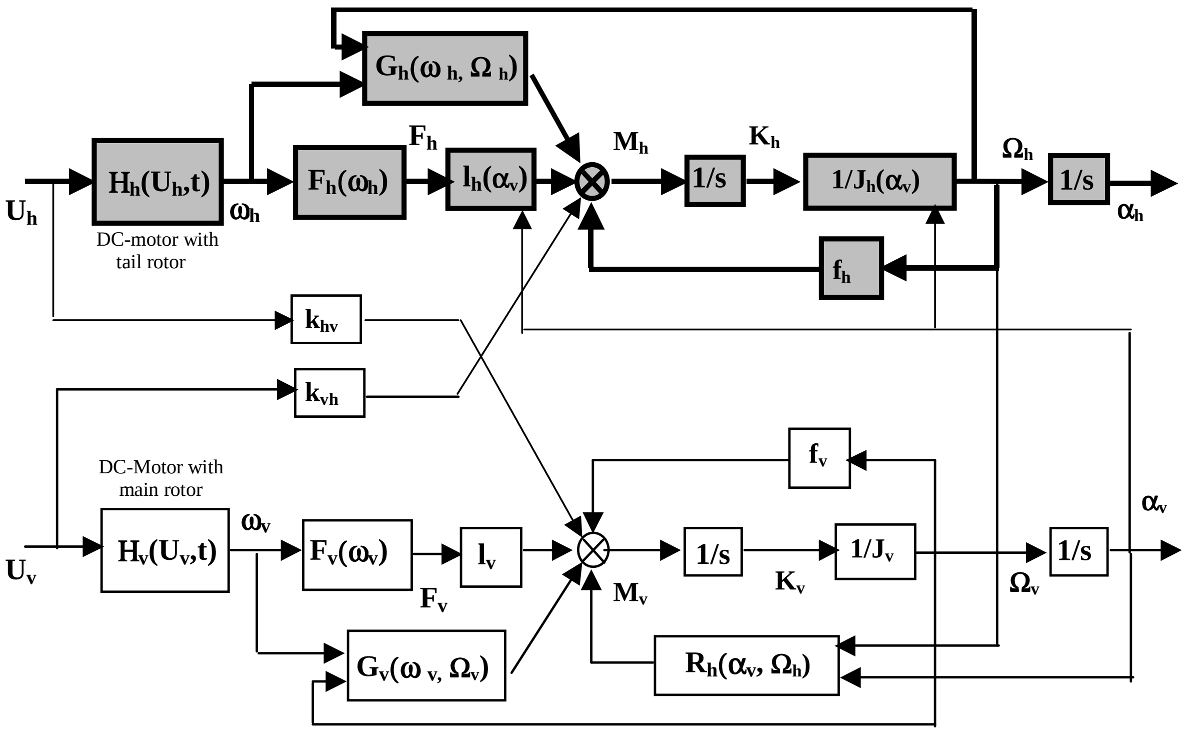


Fig. 5.9 The block diagram of 1-DOF system (horizontal plane)

5.2.4 Real-time 1-DOF azimuth control experiment

Fix TRAS 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. 5.10 opens. Set all PID controller coefficients as $K_p = 4.9395$ $K_i = 0.0022$ and $K_d = 5.1898$. Also set saturation of the integral part of the controller to 1.0. Build the model and click on the *Simulation*/*Connect to target* and *Start real-time code* options.

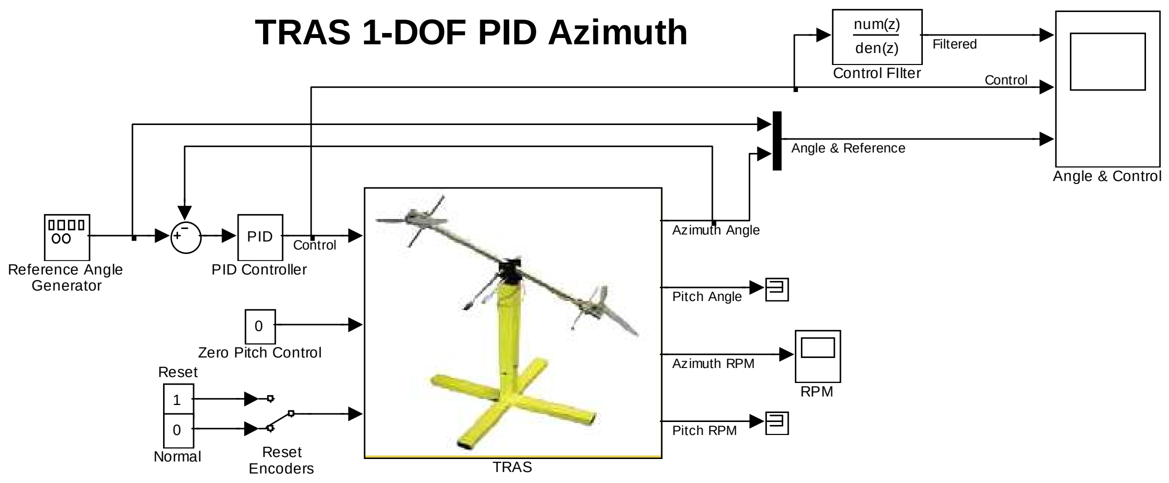


Fig. 5.10 Real-time model for the PID azimuth control

The results of the experiment are shown in Fig. 5.11. Notice a high frequency of the control similar to that in the case of the pitch control. This phenomena appears due to the quantization effects of the signal caused by the differential part of the controller. For this reason the control signal is filtered. It is shown in the upper part of Fig. 5.11.

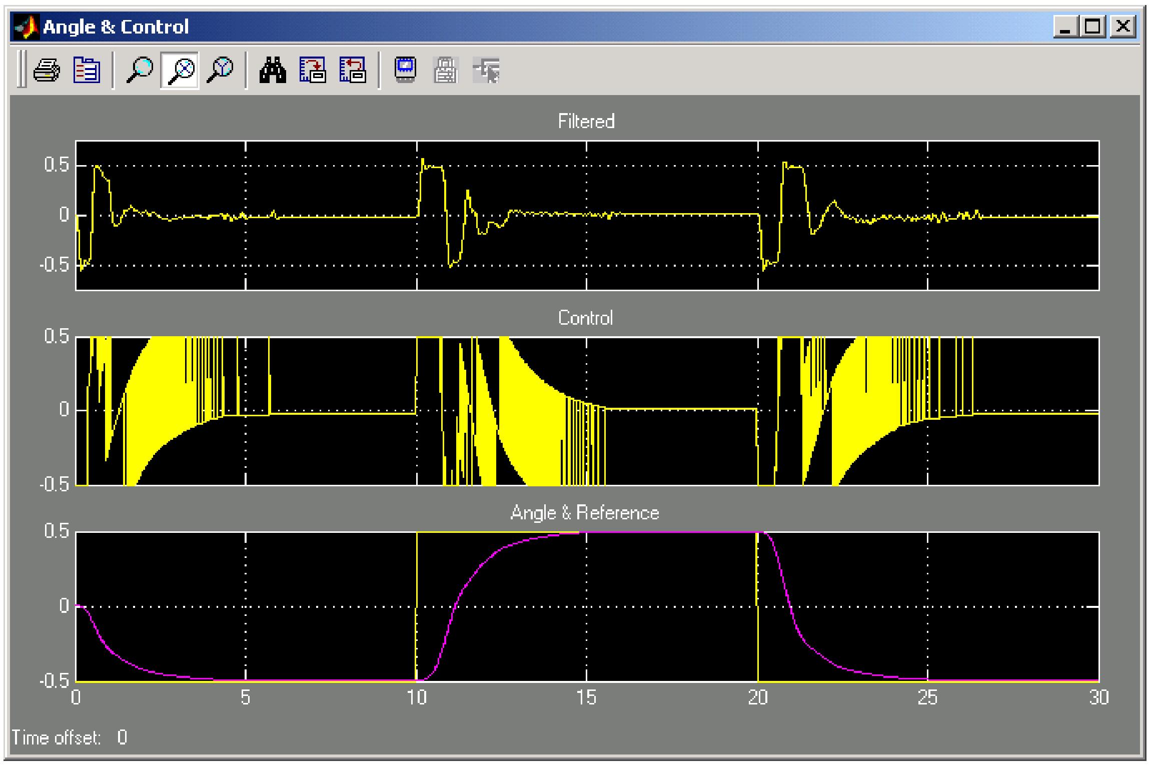


Fig. 5.11 Results of the PID azimuth control

The details of the above experiment are shown in Fig. 5.12, Fig. 5.13and Fig. 5.14.

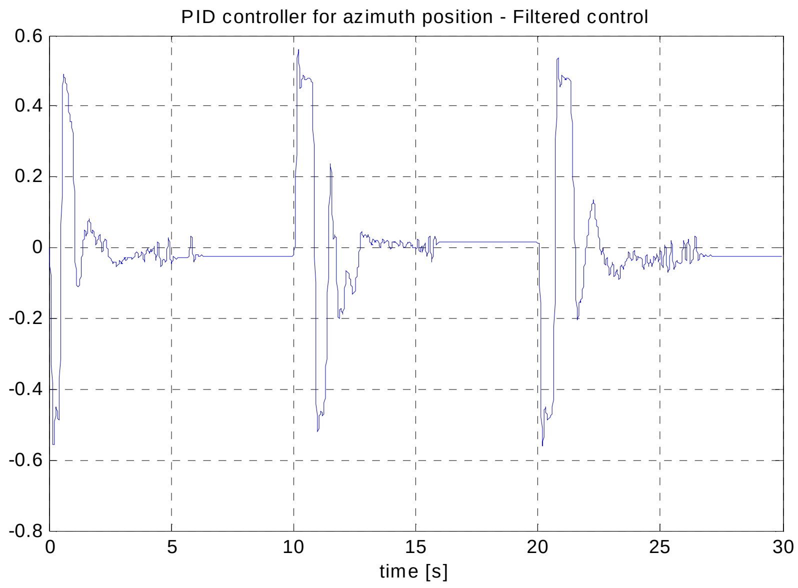


Fig. 5.12 The filtered control

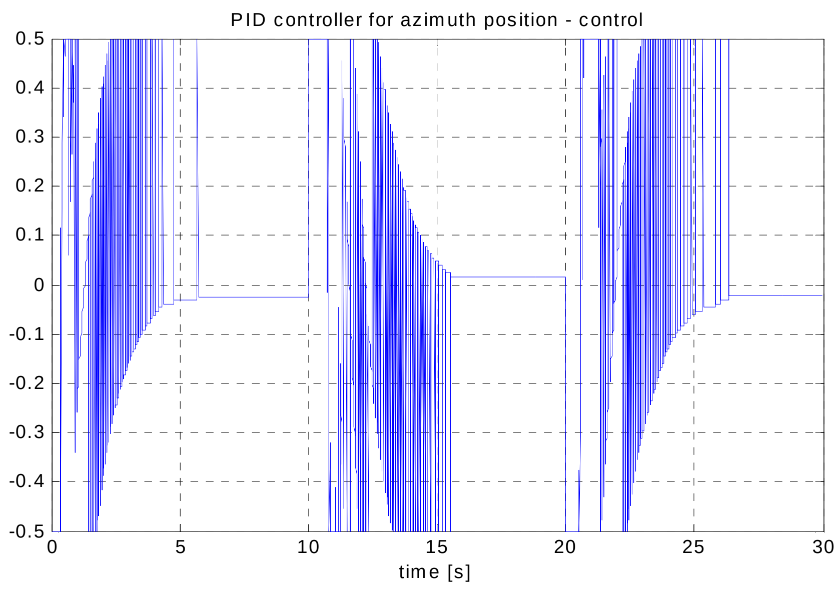


Fig. 5.13 The non filtered control

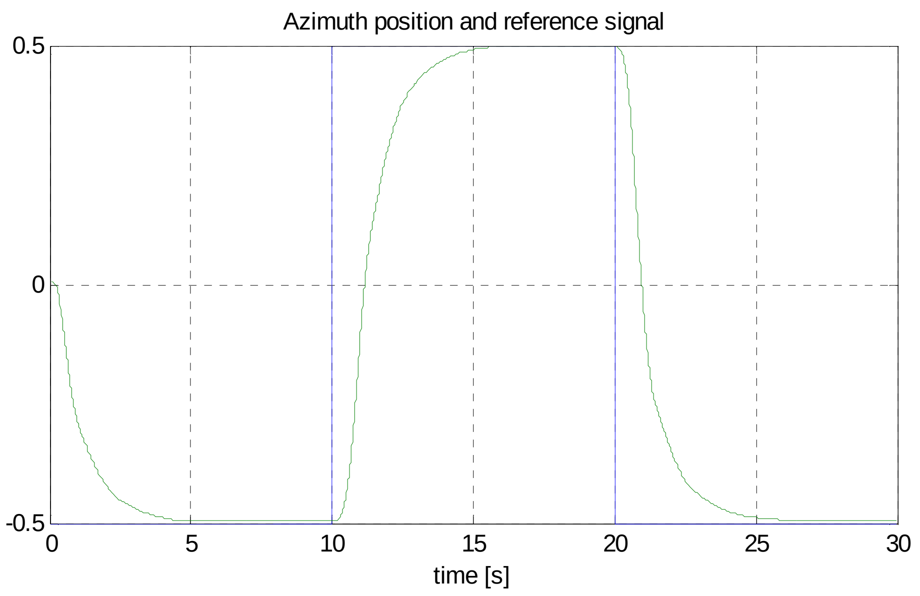
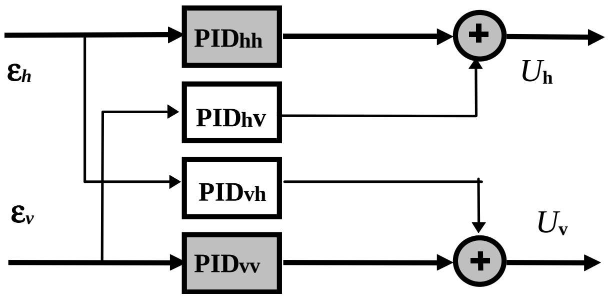


Fig. 5.14 The azimuth position and reference signal

5.3 2-DOF PID controller

The structure of the cross-coupled multivariable PID controller is shown in Fig. 5.15.

a)



b)

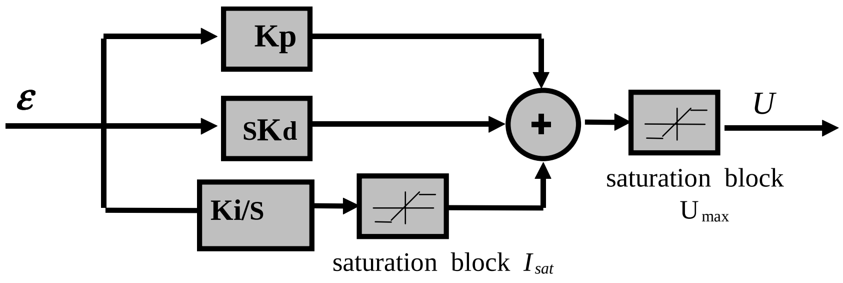


Fig. 5.15 Structure of the cross-coupled PID controller

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 the vertical (pitch) and horizontal angle (azimuth), α_{vd}, α_{hd} are the reference values of the vertical and horizontal angles, α_v, α_h are the 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.

5.3.1 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 shown in Fig. 5.16. The controller structure is shown in Fig. 5.17.

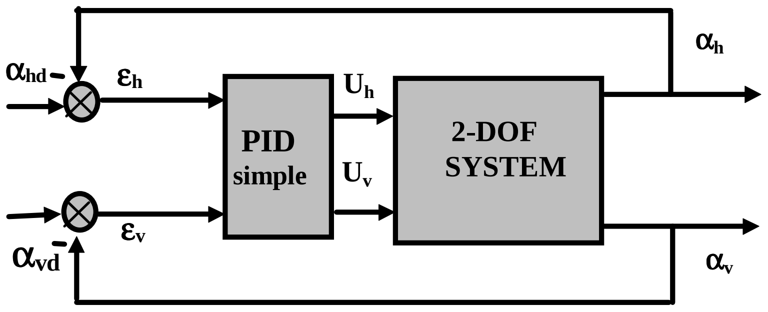


Fig. 5.16 The block diagram of 2-DOF control system with a simple PID-controller

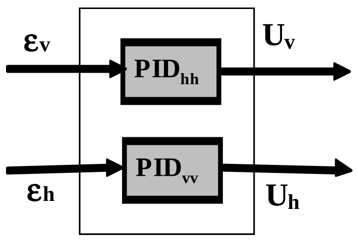


Fig. 5.17 The block diagram of the simple PID-controller

5.3.2 Real-time 2-DOF control with the simple PID controller

The control 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. 5.18 opens. Set all coefficients of the crossed PID controllers to zero. In this way the simple PID controller is obtained. Set coefficients of the azimuth controller as follows: $K_{phh} = 3.1352$ $K_{ihh} = 0.0$ and $K_{Dhh} = 2.2094$. The integral saturation set to 1.0. Set the coefficients of the pitch PID controllers as: $K_{pvv} = 1.2627$ $K_{ivv} = 1.4014$ and $K_{dvv} = 1.2074$. Set saturation of the integral part of the controller to 1.43. Set the reference azimuth signal as square wave with 0.2 [rad] amplitude and 1/40 [Hz] frequency. Set the reference signal for pitch as sinusoidal wave with the amplitude and frequency as 1/30 [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. 5.19 and Fig. 5.20. The azimuth position does not reach the desired position and the pitch position is weakly damped when disturbances from the rapid motions of the azimuth axis occur.

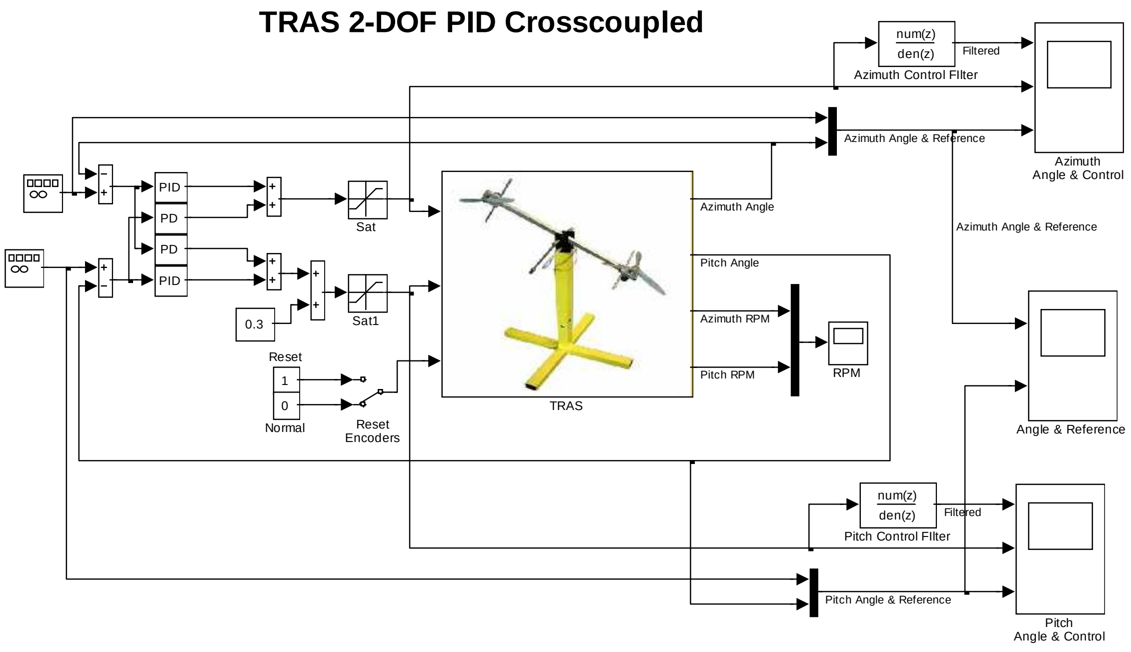


Fig. 5.18 Real-time model of the 2-DOF control task

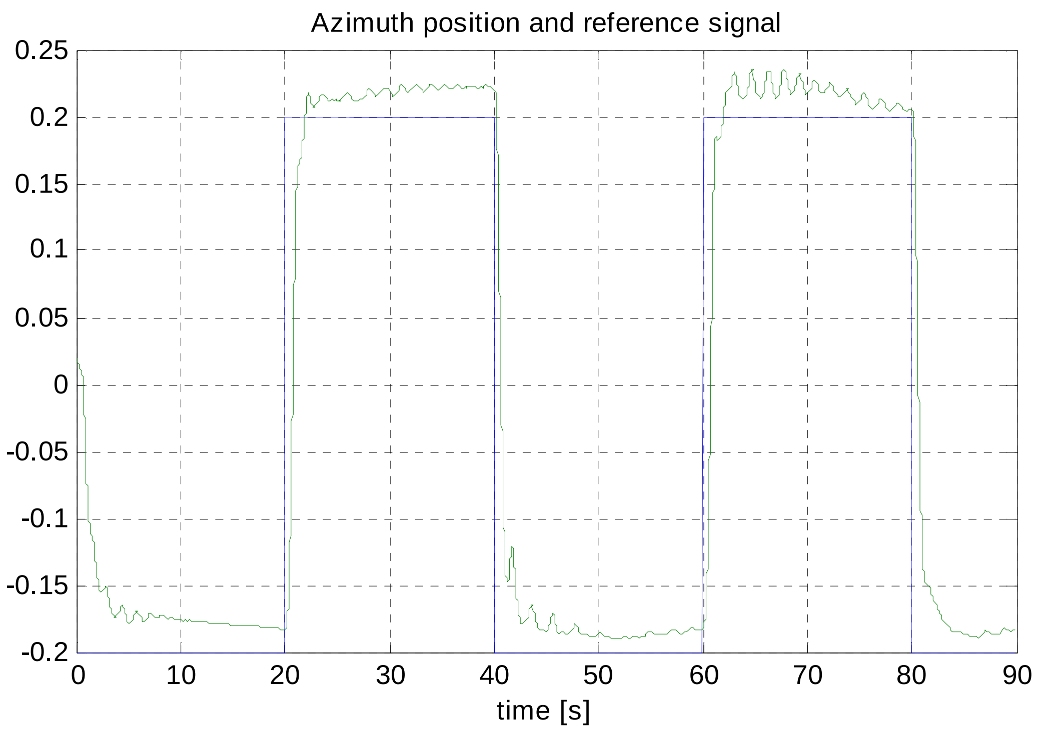


Fig. 5.19 Results of the 2-DOF control with the simple PID controller (azimuth position)

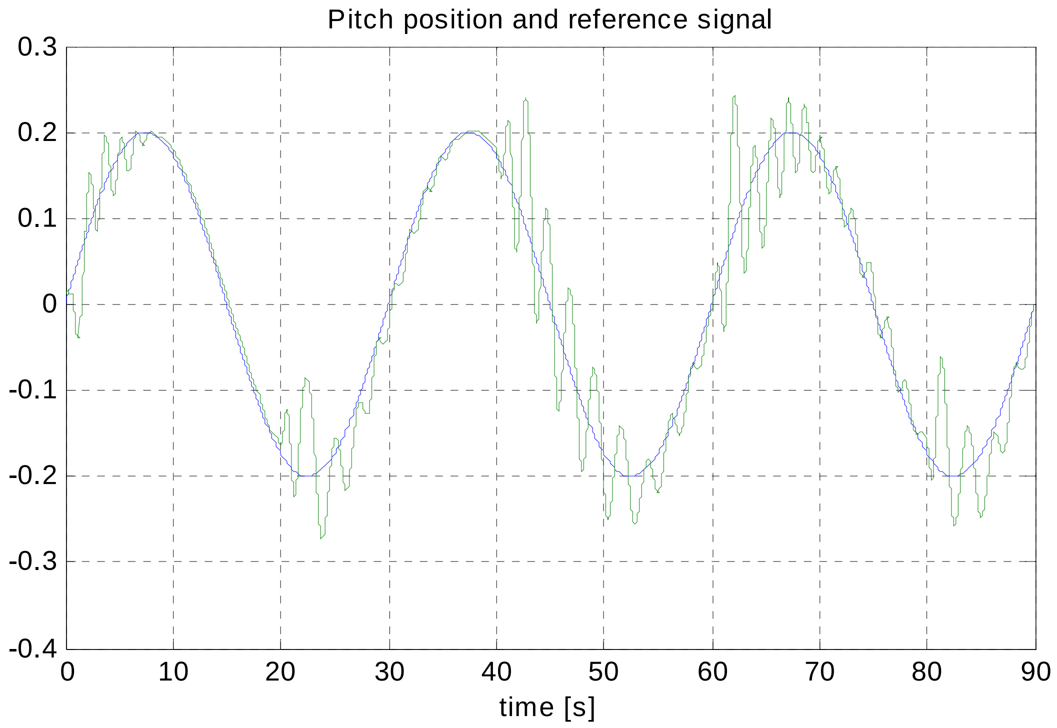


Fig. 5.20 Results of the 2-DOF control with the simple PID controller (pitch position)

5.3.3 Cross-coupled PID controller

The cross-coupled PID controller steers the system in the pitch and azimuth planes. In this control system the 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. 5.21. The cross-coupled PID controller structure is shown in Fig. 5.22.

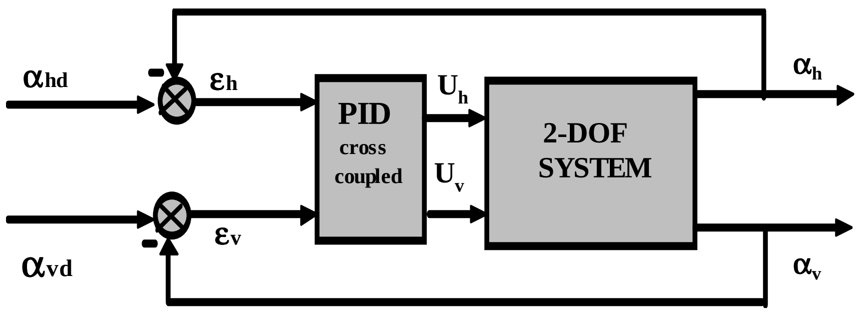


Fig. 5.21 The block diagram of the 2-DOF control system with the cross-coupled PID-controller

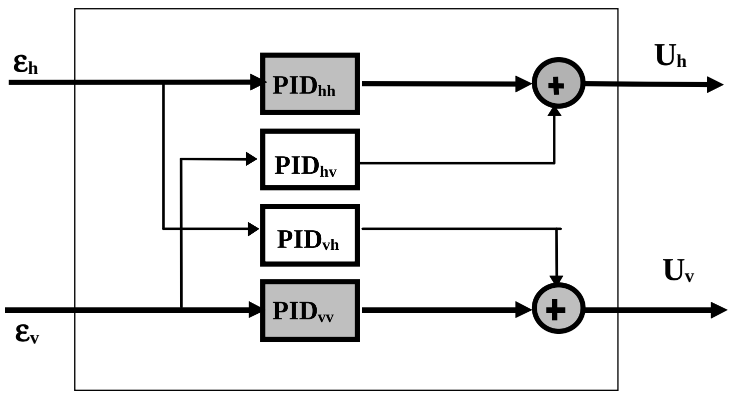


Fig. 5.22 The block diagram of the cross-coupled PID controller

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

Click the *2-DOF controller* button and the model shown in Fig. 5.18 opens. Set the coefficients of the crossed PID controllers as follows:

PID_{hh} (the azimuth controller)

$K_{phh} = 3.2465$ $K_{ihh} = 0.0367$ and $K_{dhh} = 2.152$. Set the integral saturation to 1.0.

PID_{hv} (the cross azimuth-pitch controller)

$K_{phv} = -0.9334$ $K_{ihv} = 0.0$ and $K_{Dvv} = -0.7845$,

PID_{vh} (the cross pitch-azimuth controller)

$K_{pvh} = -0.0363$ $K_{ivh} = 0.0$ and $K_{Dvh} = -0.0223$,

PID_{vv} (the pitch controller)

$K_{pvv} = 0.4978$ $K_{ivv} = 0.4392$ and $K_{Dvv} = 0.4464$. Set the saturation of the integral part of the controller to 1.43.

Also set the reference signals as in the previous experiment: the reference azimuth signal as square wave with 0.2 [rad] amplitude and 1/40 [Hz] frequency and the reference signal for azimuth as sinusoidal wave with the amplitude and frequency as before.
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. 5.23 and Fig. 5.24.

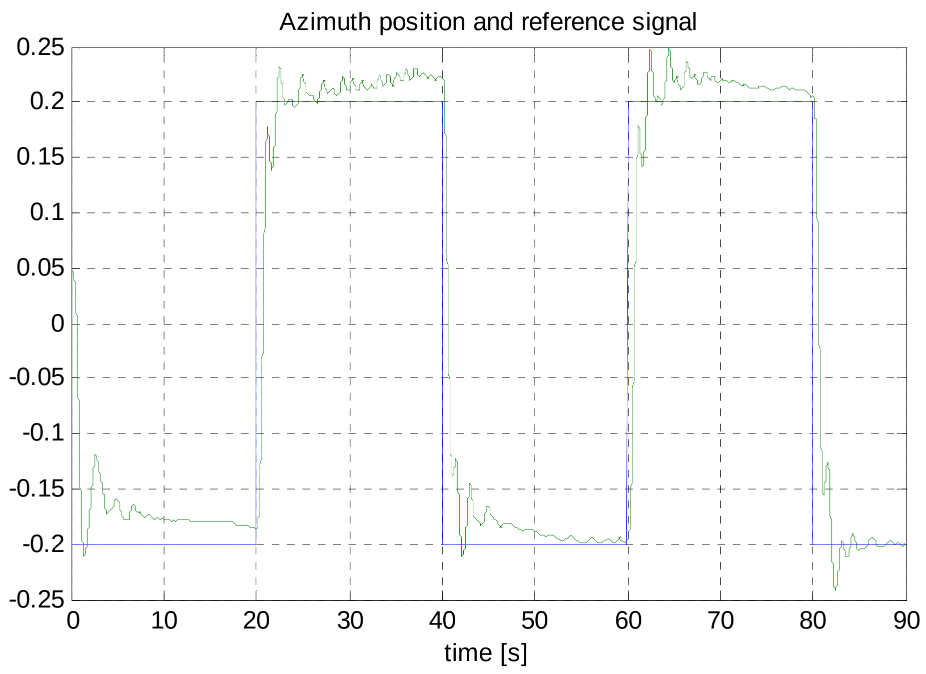


Fig. 5.23 Results of the 2-DOF control with the cross-coupled PID controller (azimuth position)

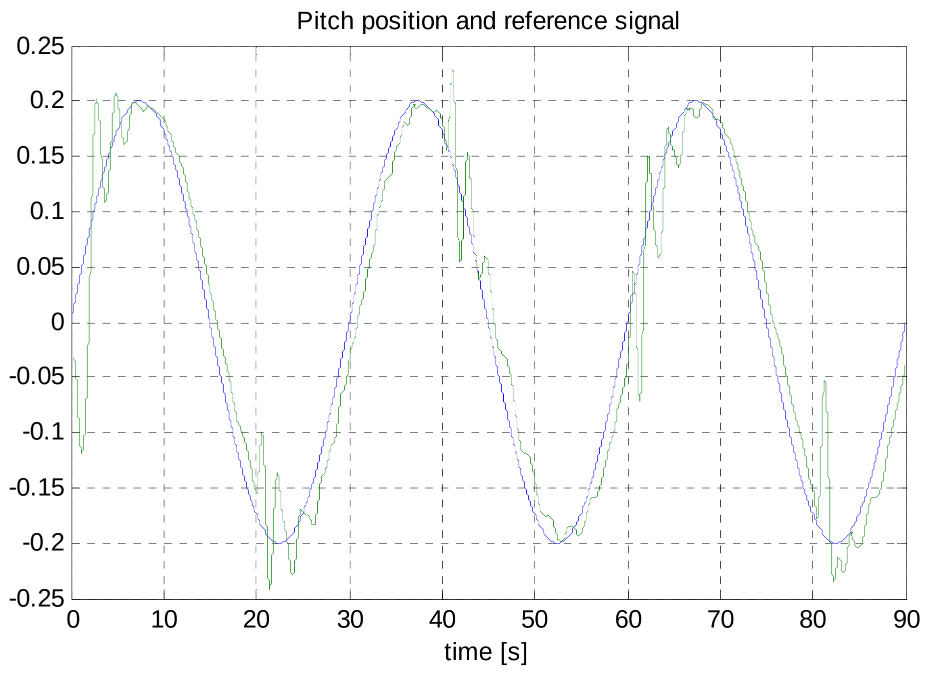


Fig. 5.24 Results of the 2-DOF control with the cross-coupled PID controller (pitch position)

5.3.5 Comparison the simple and cross-coupled PID controller

Results of experiments for the simple and cross-coupled controllers are compared in Fig. 5.25 and Fig. 5.26. It can be seen that compensating action of the coupling controller improves the control quality especially for the pitch angle.

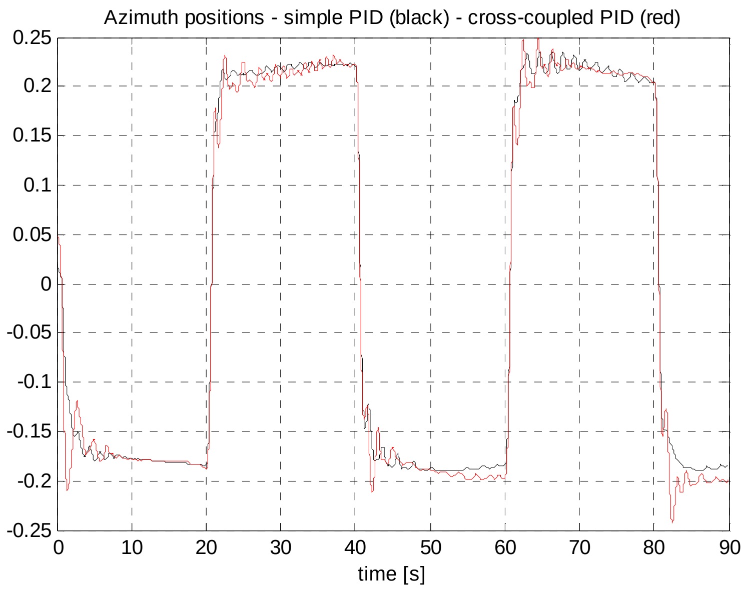


Fig. 5.25 Comparison the simple to the cross-coupled PID control (azimuth)

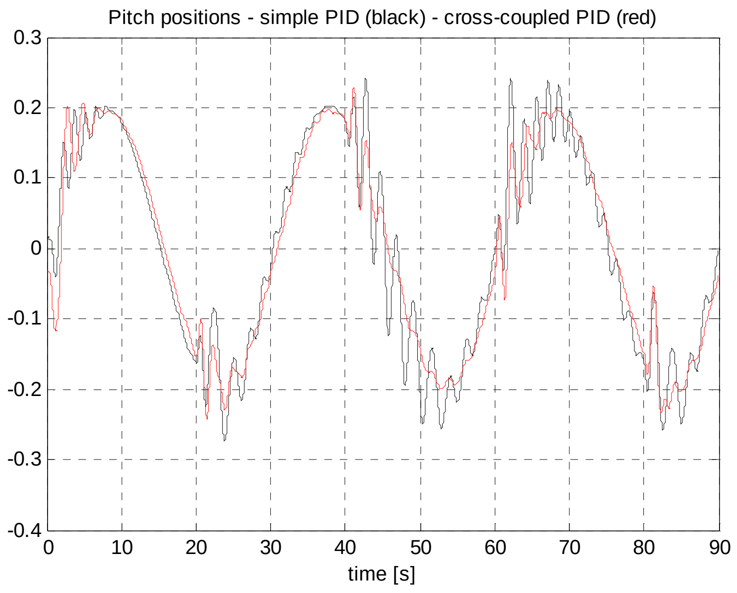


Fig. 5.26 Comparison the simple to the cross-coupled PID control (pitch)

6. PID controller parameters tuning

There are several methods to design 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. 6.1.

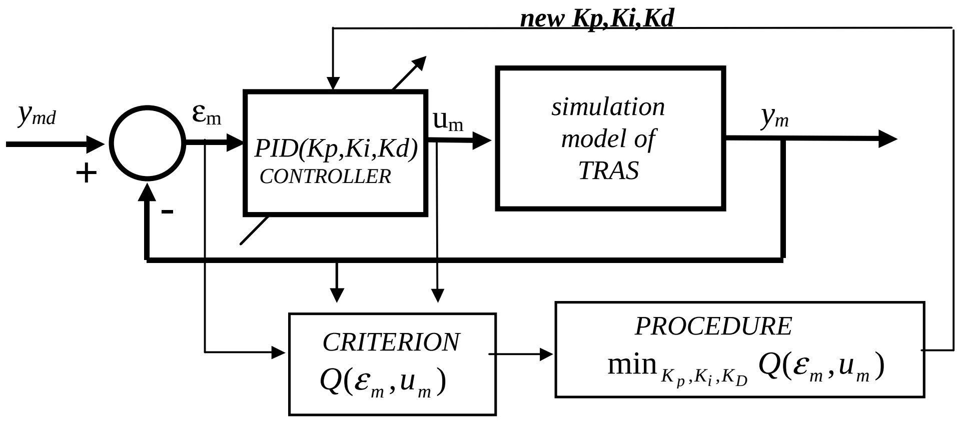


Fig. 6.1.Schematic diagram of the PID parameters tuning

In the case of *TRAS* the following criterion is used to tune the PID parameters for 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] is the simulation time , ε_h is the azimuth position error, ε_v is the pitch position error, u_v is the pitch control. The 0.1 coefficient is the value of the pitch control which keeps the beam in the horizontal position.

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

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

These files use their own simulation models and the criterions m-file in optimisation process. See the body of these files to learn how the optimisation procedure is performed.

7. Description of the CTRAS class properties

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

- incremental encoder registers – two 16-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 a 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 RT-DAC/PCI 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

7.2 BaseAddress

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

Synopsis: *BaseAddress = get(tr, ‘BaseAddress’);*

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

Example: Create the CTRAS object:

tr = CTRAS;

Display their properties by typing the command:

tr

```
Type:                CTRAS Object
BaseAddress:         54272 / D400 Hex
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]
```

Read the base address:

BA = get(tr, ‘BaseAddress’)

7.3 BitstreamVersion

Purpose: Read the version of the logic stored in the RT-DAC/PCI board.

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

Description: The property determines the version of the logic design for the RT-DAC/PCI board. TRAS 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.

7.4 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*

7.5 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*

7.6 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*

7.7 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]);*

7.8 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:
$\mathbf{P}\text{WM}_{\text{frequency}} = 40\text{MHz} / 1023 / (\text{Prescaler}+1)$

See*:* *PWM*

7.9 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*

7.10 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]);

7.11 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*

7.12 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*

7.13 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*

7.14 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*

7.15 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 the PWM waves when the corresponding power interfaces is overheated.

See: *Therm*

7.16 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.

7.17 Quick reference table

Property name	**Operation**[*]	**Description**
BaseAddress	R	Read the base address of the RT-DAC/PCI board
BitstreamVersion	R	Read the version of the logic design for the RT-DAC/PCI 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

7.18 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 tachogenerator.

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 );

% Calculate control signal 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
```

```
% Select the rotor(s) used during the experiment
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

% Create figure that presents the current measurements
FigNum = figure( 'Visible', 'on', ...
                 'NumberTitle', 'off', ...
                 'Name', 'Rotor velocity vs. PWM characteristic', ...
                 'Menubar', 'none' );
tr = ctras;
ret = [];
for i=1:length(Ctrl)
  % Set control signal(s)
  set( tr, 'PWM', [ACtrl(i) PCtrl(i)] );
  % Wait for steady-state
  pause( 10 )
  ret(i,1)   = Ctrl(i);
  % Read a number of tacho voltages to calculate
  % the average tacho output
  AuxVolt = [0 0];
  for j=1:10000
    AuxVolt = AuxVolt + get( tr, 'RPM' );
  end
  ret(i,2:3) = AuxVolt/10000;
  % Plot results
  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

% 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.

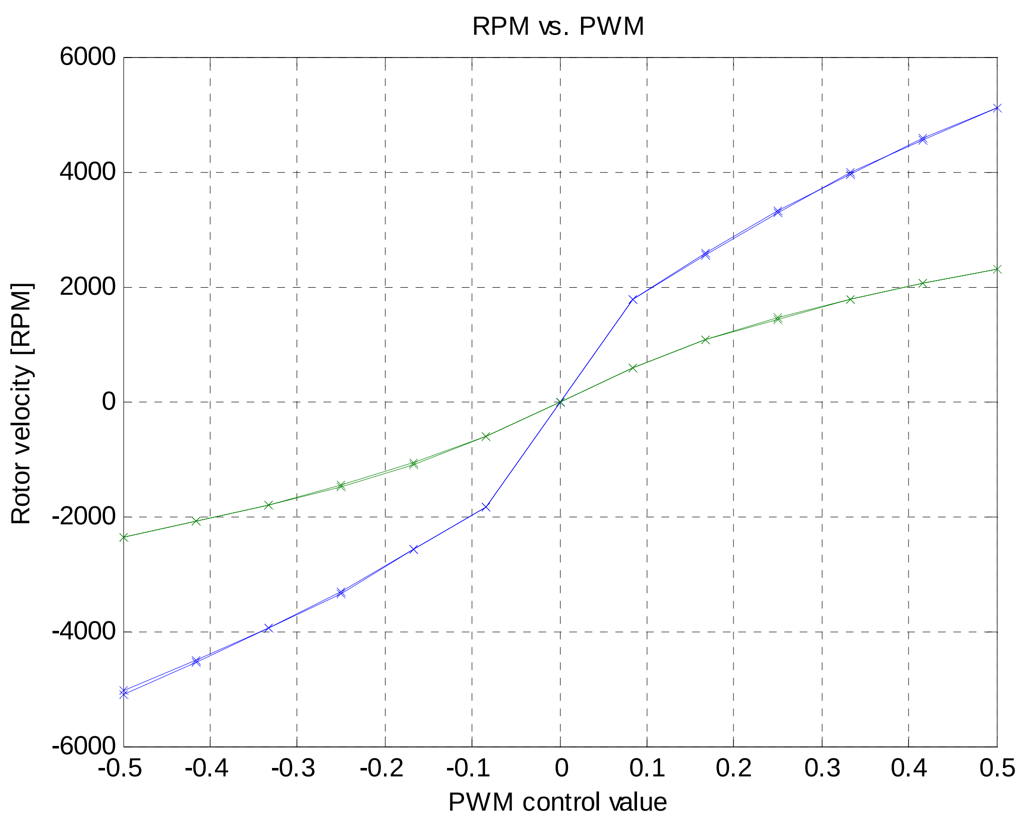


Fig. 7.1 Static characteristics