ECN Design Tool for Control Development

Revised Points of Departure

T.G. van Engelen, E.J. Wiggelinkhuizen

Abstract

An open source software environment for the design of wind turbine control algorithms has been created. The modularisation enables to modify and to add or remove functionality in a user friendly way. This software environment established the point of departue for the development of specific control algorithms within the project. It consistst of so called functions and scripts in the MATLAB programming language. These MATLAB modules support modelling, control synthesis and data and signal handling. They thus enable a user to implement a modular parametrisation procedure for the control algorithm and a Simulink compatible evaluation of the designed controller. The software has been made typical for the variable speed windturbines with pitch control. However, other wind turbine concepts can also be dealt with.

Acknowlegdement

Eric van der Hooft and Pieter Schaak are cordially acknowledged for their contributions to the development of the open source software which resulted from this part of the project.

CONTENTS

1. INTRODUCTION

This report deals with the first main task of the first work package in the NOVEM-funded project 2020-01-12-10-003, 'Ontwerpgereedschap voor het integraal ontwerpen van windturbine regelingen' (ECN projectnr 7.4153). The purpose of this main task (WP1-HT1) was to provide a thorough base for the foreseen control developments within the project.

Preliminary MATLAB program code for controller synthesis and evaluation has been created in the ARB-funded project 7.4046 ('Ontwerpgereedschappen voor de regeling van windturbines') . The design tool in this stage, typed as the '4046-M-code', has been used as point of departure for this project; it involves implemented basic control functionality. The 4046-M-code had to be revised substantially in order to be able to deal with the planned developments. This revision included the following activities (summary of description of WP1-HT1 in the project's working program):

- definition of turbine concepts and involved control features;
- modularisation of the 4046-M-code concerned with parametrisation of feedback control and of controller evaluation;
- reconfiguration of the 4046-M-code concerned with controller evaluation for inclusion in the graphical MATLAB-Simulink enviroment.

These activities have been performed and are subject of this status report.

Turbine and control concept selection is subject of §2. The revised MATLAB program code is typed as the 'pdp-M-code'; the acronym 'pdp' stands for point of departure. The pdp-M-code enables to efficiently create wind turbine specific software for control development. Such a development environment is restricted to turbines within the selected concepts up to the level of *basic control functionality*[1].

The pdp-M-code consists of MATLAB functions and scripts (code blocks) that:

- **implement a typical modular parametrisation procedure (§4);**
- **implement a typical Simulink compatible controller evaluation (§5);**
- **support modelling, control synthesis and data and signal handling (§6).**

The first two sets of M-code establish typical MATLAB programs for actual wind turbine control development. Together with the supporting M-code, these M-programs set up the pursued pdp-M-code for efficient generation of 'any' control development environment.

An overview of the pdp-M-code is given in §3.

The work package main task WP1-HT1 also includes MATLAB compatible C-code generation, which has not yet been performed. However, this C-code is derived straightforward from the created program code in above listed activities through use of the dedicated MATLAB C-compiler.

[1] See chapter 2

2. TURBINE CONCEPTS AND CONTROL

The two following turbine types are considered for control design:
- variable speed operation with pitch to vane power limitation;
- constant speed operation with pitch to stall power limitation.

These two turbine types are discussed in short in the next two sections; potential control features are emphasized.

2.1 Variable speed with pitch to vane power limitation

Basic control features (preliminary implementation in 4046-M-code):
- rotor speed control by blade angle adjustment based on generator speed feedback and estimated wind speed
- electric torque control by implemented torque/speed-curve in the electric conversion system

Potential control features as foreseen now are:
- fore/aft tower vibration control by blade angle adjustment on tower top acceleration feedback;
- sideward tower vibration control by electric torque control on tower top acceleration feedback;
- drive train distorsion control in connection with collective lead-lag blade bending by electric torque control on generator speed feedback;
- reduction of cyclic blade loads by azimut-dependent individual pitch control;
- 'free' yawing by azimut-dependent individual pitch control

Blade bending vibrations differing from collective lead-lag bending are subject to research in the EU-contracted STABCON-project.

2.2 Constant speed with pitch to stall power limitation

Basic control features (preliminary implementation in 4046-M-code):
- rotor speed control by blade angle adjustment based on rotor speed feedback at decoupled generator from the grid (startup, shut-down);
- electric power control by blade angle adjustment based on

Potential control features as foreseen now are:
- reduction of cyclic blade loads by azimut-dependent individual pitch control;
- fore/aft tower vibration control by active yawing based on tower top acceleration feedback (gyroscopic effect);
- individual lead-lag blade bending vibrations control by mass/spring/damper or functionally equivalent devices in blade tips
- symmetric lead-lag blade bending vibrations control by mass/spring/damper or functionally equivalent devices in tower top

3. OVERVIEW REVISED MATLAB CODE

The objective of the revised MATLAB code, typed as the *point of departure M-code* (pdp-M-code), is as follows:

> *To enable efficient creation of controller parametrisation procedures and of evaluation sessions for a specific wind turbine within the selected concepts.*

The parametrisation procedures establish the 'design stage' of control development; the evaluation sessions the 'simulation stage'. For each specific wind turbine, two unique MATLAB programs are to be created: one for the design stage, another for the simulation stage. Such *application M-code* can only be created with reasonable time limits if *supporting M-code* is available.

The developed application and supporting M-code is summarised in §3.1. Section 3.2 describes how to install this software.

3.1 Application and supporting M-code

The application and supporting M-code has been created as modules in the MATLAB programming language. All M-code modules exist as *M-files* (extension '.m'). An M-file accommodates a function or a script. An M-function maps inputs to outputs through a function header and may have access to *global* variables and files. An M-script is just a block with M-code lines and has access to all variables within the scope of the *calling environment* (M-file or MATLAB shell). An M-script called from the M-shell is (the root of) a MATLAB program being executed.

M-scripts 'parvsp' and 'simvsp' establish the root of a modular parametrisation program and of a Simulink compatible simuluation program. Together with user provided functions and other scripts, the scripts 'parvsp' and 'simvps' set up the application M-code of a *control development environment* for a typical variable speed wind turbine with power limitation by pitch to vane control (§4 and 5). They frequently state calls to supporting M-code modules.

The development time of programs like 'parvsp' and 'simvsp' is minimised by the the support M-code modules (§6). They pertain to data and signal handling, subsystem modelling, and control loop synthesis. They also involve more basic M-code modules for general support and MALTAB startup support.

Figure 3.1 shows the design and simulation stage of the control development process that is composed from application (black) and supporting M-code modules (grey).

DESIGN STAGE
When actived from the MATLAB-shell, M-script 'parvsp' calls the main *user parametrisation functions* (UP-functions) 'parturbinevsp()' and 'parcontrolvsp()':

- UP-function 'parturbinevsp()' imports the required 'root' data for control development from specifiing user parametrisation M-scripts (UP-scripts). The UP-scripts just consist of data assignment directives for the turbine rotor, support structure, drive-train and site and for the operation conditions (see §4.1). The root data are processed into parameters for aerodynamic conversion, generator operation and for dynamic inflow simulation via support functions 'ecnmkcpcqct()', 'ecngenerator4vs()' and 'ecndynflo4p2v()'.

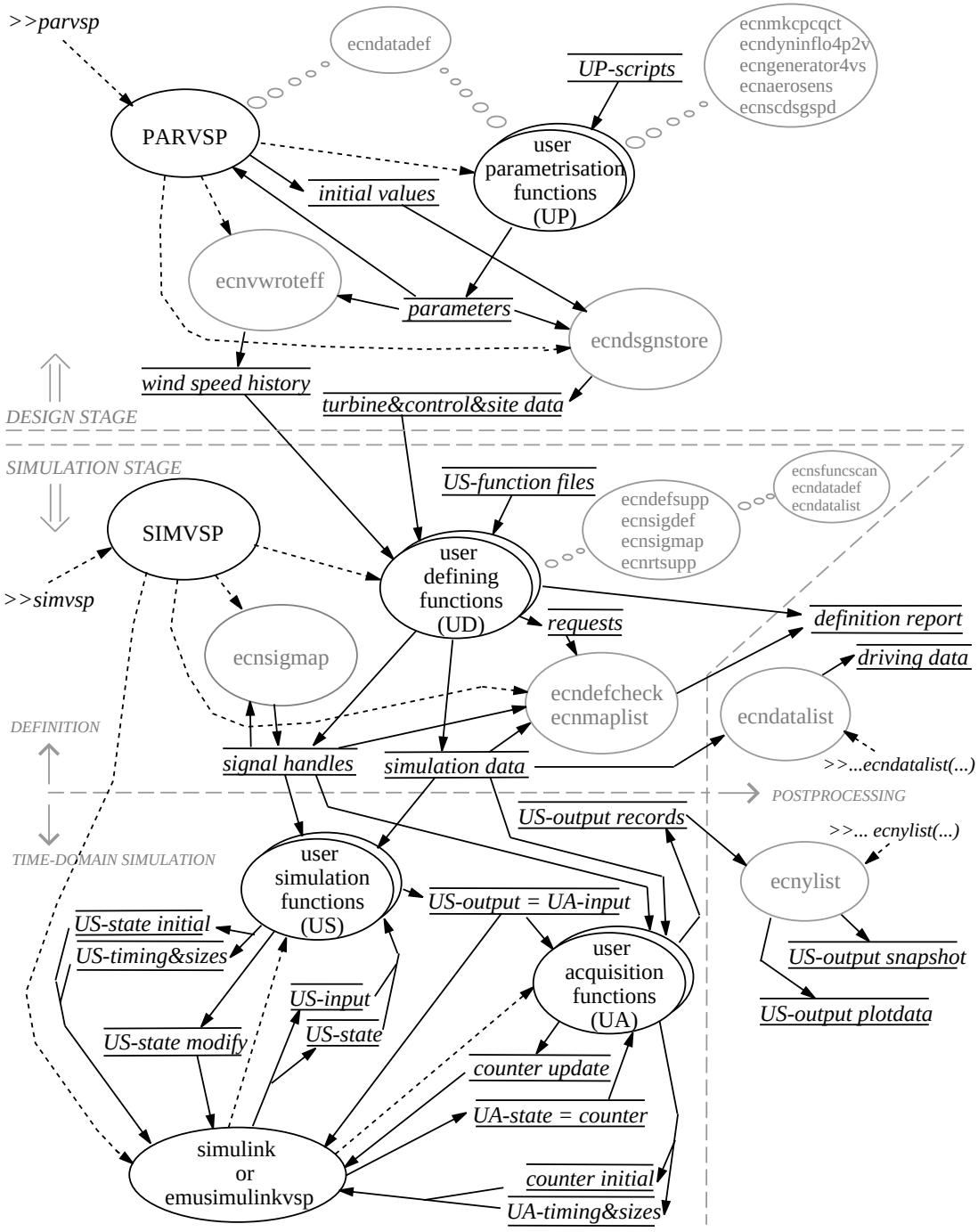

Figure 3.1 *Application and support M-code modules in a control development environment*

- UP-function 'parcontrolvsp()' processes the 'root' data into control and filter parameters for pitch actuation and generator torque setting. Control design driving data are *NOT* imported via UP-scripts but are set by 'parcontrolvsp()' in the calls to control aspect specific UP-functions (see §4.1). Support functions 'ecnaerosens()' and 'ecnscdsgspd()' are used for linearisation of the aerodynamic conversion and for synthesis of rotor speed feedback control.

M-script 'parvsp' by itself calculates the initial values for the simulation state vectors. Support functions 'ecndatadef()' and 'ecndsgnstore()' are used for file storage of the turbine, control and site related data.

Support function 'ecnvwroteff()' creates a data file with a rotor effective wind speed history from the site and geometrical turbine data.

SIMULATION STAGE → DEFINITION
When actived from the MATLAB-shell, M-script 'simvsp' calls the *user defining functions*. The UD-functions are 'dwindvsp()', 'dctrpvsp()', 'dctrgvsp()', 'dturbvsp()' and 'dsimuvsp()', see §5.1). The UD-functions create the simulation data requested for from the results of the design stage and from run-specific settings via support functions 'ecndefsupp()' and 'ecnrtsupp()'. These support functions scan the *user simulation functions* (US-functions) on data and signal requests.

The UD-functions also create signal handles via support functions 'ecnsigdef()' and 'ecnsigmap()'. The signal handles enable name based indexing of signal vectors in a US-function and array-wise coupling of input and state variables to output variables. M-script 'simvsp' by itself calls to 'ecnsigmap()' for extension of signal handles with output to input signal couplings between US-functions. This allows to connect US-functions in the simulation scheme via the complete output vectors.

Support functions 'ecndefcheck()' and 'ecnmaplist()' provide a definition report that tells if all data and signal requests by the US-functions are satisfied via the UD-functions and which signal couplings have been established.

SIMULATION STAGE → TIME-DOMAIN SIMULATION
M-script 'simvsp' calls to the Simulink kernel or to the user provided function 'emusimulinkvsp() for emulation the Simlink kernel. A Simulink driven simulation is configured by connected so called 'S-functions'. An S-functions is a building block for modular simulation. It consists of different sections, pertaining to initialisation, to state vector update and/or derivative calculation and to output vector calculation. Actually, the core of an S-functions consists of implemented difference and/or differential equations (time-discete, time-continuous).

The involved S-functions are the user provided simulation functions (US-functions) *and* data acquisition functions (UA-functions); the latter only provide periodic storage of US-function output values to an array. The US-functions are (see §5.1, also for UA-functions):
- 'swindvsp()', for wind speed generation (time-discete);
- 'sctrpvsp()', for pitch angle control (time-discrete);
- 'sctrgvsp()', for generator torque control (time-discrete);
- 'sturbvsp()', for wind turbine simulation (time-discete & time-continuous).

The Simulink kernel initially recieves timing and size information from the S-functions as well as the initial state values. Afterwards, the kernel provides numerical integration via the state derivates of the differential equations and state updates for difference equations. The state evoluation is driven by the output to

input vector connection of the S-functions and by the array-wisse available wind speed data in ‘swindvsp()’.

SIMULATION STAGE → POST PROCESSING
The driving data for the simulation can be well formatted printed out with support function ‘ecndatalist()’. Support function ‘ecnylist()’ enables a print out of any sample of stored output data by the UA-functions or to prepare plotting data; print-outs and plots include signal names and dimensions.

3.2 PC set-up for contemporary ‘read and run’

A floppy disk with M-files belongs to this report. The M-files accomadate the supporting M-code and typical application M-code. They can be used for running (parts of) the application programs ‘parvsp’ and ‘simvsp’ while reading this report. Copy the files to your system, by preference in different directories in accordance with the floppy disk:

- MATLAB startup specific M-files in ../mfilesecn/matstart
- general support M-files in ../mfilesecn
- control tool ‘data&signal support’ M-files in ../rgtoolnv/DATSIG/M
- control tool ‘model support’ M-files in ../rgtoolnv/MODEL/M
- control tool ‘control support’ M-files in ../rgtoolnv/CTRL/M
- VSP-parametrisation M-files in ../rgtoolnv/VSPITCH/M

When the below-described MATLAB startup procedure is applied, the directories are to be children of <PROJECTDisk>:/<PROJECTContributor>. In that case, the following assignment directives for M-vars ‘vspitchpathm’, ‘vspitchpathmat’ and ‘vspitchpathps’ in MATLAB-programs ‘parvsp’ and ‘simvsp’ can be maintained:

```
vspitchpathm   = rgtoolnv.vspitchm;
vspitchpathmat = rgtoolnv.vspitchmat;
vspitchpathmat = rgtoolnv.vspitchps;
```

The workspace structure ‘rgtoolnv’ results from startup if the ‘ECN startup M-files’ are installed. The installation consist of

- copy startup M-files to MATLAB file system’s directory toolbox/local:
- check [and modify] next settings in ‘ecnstartup0.m’:

```
PROJECTDisk        = '<mnemonic of destination disk for floppy data>'
PROJECTContributor = '<parent directory name for file systems on floppy>'
ROOTTool           = '<child directory name for control tool M-files>' ['rgtoolnv']
ROOTGenmfiles      = '<child dir name(s) for general M-files>' ['mfilesecn' '<others>']
```

- extend the MATLAB startup file startup.m in ../toolbox/local with following code fragments (see also documentation file ../mfilesecn/startup/ecnstartup.txt):

```
path(pathdef);
ecnstartup0; % user settings for startup driving workspace variables
[PROJECTRoot,PROJECTEntry]=ecnstartupexe(PROJECTDisk,PROJECTContributor,...
                        ROOTTool,SUBIDFTool,SUBDIRApptext,SUBSUBDIRApptext,...
                        SUBDIRAppdevelop,SUBSUBDIRAppdevelop,ROOTGenmfiles);
global xxxROOT;
if ~isempty(xxxROOT),
   eval(sprintf('%s = xxxROOT;',PROJECTRoot));
end
clear global xxxROOT;
```

Make a startup with ‘rgtoolnv’ as directory to be selected.

This startup procedure extends the MATLAB path with M- and MDL-directories.

NOTE: Above listed ‘slashes’ ‘/’ are to be read as ‘backslashes’ ‘\.

4. APPLICATION M-CODE PARAMETRISATION

The MATLAB-program 'parvsp' provides the *design* stage of control development for a variable speed HAWT pitch to vane power limitation (VSP). This consists in making available:
- turbine & site parameters,
- control parameters,
- wind speed history,
- initial values,

for use in the *simulation* stage of VSP control devleopment as set up by Simulink compatible MATLAB-program 'simvsp' (§5).

Actual MATLAB-programs for VSP-parameterisation are likely to deviate from 'parvsp'. However the program structure, the use of supporting MATLAB-functions, and the created MATLAB-variable types will be similar.

§4.1 gives a comprehensive overall program description and a list of the turbine specific files for parameter definition. §4.2 deals in more details with creation of data items and MATLAB binary data files (MAT-files).

4.1 Scope on parametrisation program 'parvsp'

MATLAB-program 'parvsp' generates the data items during control development. Program 'parsvp' by itself determines the initial values for time-domain simulation while it calls:
- parturbinevsp(), for turbine and site parameter generation;
- parcontrolvsp(), for pitch control and EM-torque control parameter generation;
- ecnvwroteff(), for rotor effective wind speed signal generation.

A data item is accommodated by one of the following design specific global structures (DGS):
- DGS 'machine' for 'parturbinevsp()' (turbine and operational data);
- DGS 'site' for 'parturbinevsp()' (climatological data);
- DGS 'linctrl' for 'parcontrolvsp()' (linear controller data);
- DGS 'empctrl' for 'parcontrolvsp()' (settings and non-linear controller data).

In the first design step program 'parvsp' creates the (empty) DGS's by calling 'ecndsgnvarbases()' and makes them accessible from the MATLAB-shell via 'ecnDGSglobal'.

The parametrising M-functions 'parturbinevsp()' and 'parcontrlvsp()' are turbine specific and thus user provided (*UP-functions*). UP-function 'parturbinevsp()' in turn calls data specifying *UP-scripts*:
- 'protorvsp': rotor data
- 'ptowervsp': support structure data
- 'pdrivetrvsp': drive train data
- 'pclimate': climatological data
- 'poperatvsp': operation data

These data satisfy for wind speed generation and turbine parametrisation. UP-function 'parturbinevsp()' completes the latter via calls to:
- 'ecngenerator4vs()': derive generator curve data;

- ‘ecndyninflo4p2v()’: derive rotor effective dynamic inflow modelling data;
- ‘ecndatadef()’: create data items in DGS’s ‘machine’ and ‘site’.

Program ‘parvsp’ continues with the creation of a MAT-file with rotor effective turbulent wind speed data via ‘ecnvwroteff()’.

UP-function ‘parcontrolvsp()’ in turn calls control synthesis UP-functions:
- ‘pbasefb()’: PD-gains and filters for linear rotor speed feedback to pitch speed;
- ‘pgeneratorfb()’: torque/speed-curve and filter for EM-torque setting;
- ‘pwindest()’: wind speed estimation and pitch speed feedforward;
- ‘psetdata()’: empirical control parameters, setpoints and buttons.

In advance of these controller synthesis calls, ‘parvsp’ calls ‘ecnaerosens()’ for calculations of aerodynamic sensitivity functions and control scheduling parameters. Data items are created in DGS’s ‘linctrl’ and ‘empctrl’ for these parameters via ‘ecndatadef()’.

A UP-function uses supporting M-scripts ‘ecnDGSfields2Items’, ‘ecnLSfields2Items’ and ‘ecnDGSfvalues4Check’ for making checks if parameter names are not used twice or more (see help on ‘parturbinevsp()’).

Program ‘parvsp’ finally calculates the initial values for wind speed generation and turbine simulation and calls:
- ‘ecndatadef()’: create initial value data items in DGS’s ‘site’ and ‘machine’;
- ‘ecndsgnstore()’: store all data items in all DGS’s on MAT-file.

The following user developed VSP-specific M-files are in directory ‘VSPITCH/M’ of the floppy disk:
- ‘parvsp.m’; masters the parametrisation;
- ‘parturbinevsp.m’, ‘protorvsp.m’, ‘ptowervsp.m’, ‘pdrivetrvsp.m’, ‘pclimate.m’, ‘poperatvsp.m’; generate turbine and site parameters.
- ‘parcontrolvsp.m’, ’pbasefb.m’, ’pgeneratorfb’, ’pwindest.m’, ’psetdata.m’; generate pitch and EM-torque control parameters;

Program ‘parvsp’ uses the following MATLAB workspace variables (M-vars)
- ‘ turbidf’: *turbine identifier*, default ‘TurVSP’;
- ‘Vwrat’ : *rated windspeed*, calculated during the parametrisation;

for creation of the following MAT-files, which also exist in floppy dir ‘VSPITCH/MAT’:
- ‘paramvsp<turbidf>.mat’, with machine, site and control data (DGS-items);
- ‘vwe<turbidf><Vwrat>.mat’, with rotor effective wind speed data.

4.2 Definition of design specific data

Program ‘parvsp’ creates data items in the design specific global structures (DGS) via calls to ‘ecndatadef()’. A design specific data item belongs to one of the following data groups:
- ‘p’-group: recommended for ‘parameter type’ data items;
- ‘b’-group: recommended for ‘initial value type’ data items.

These data groups exist as substructures ‘p’ and ‘b’ in the design specific global structures (DGS).

A created data item exists of fields in the following substructures of a data group (a ’set’ applies if the data item is a 1D- or 2D-array or a 1-layer structure with up to 2D-arrays as elements):
- ‘v’: value(set) of the data item (elements);

- ‘n’: name of the data item;
- ‘s’: name of M-file with the specification of the data item;
- ‘d’: dimension(set) of the data item (elements);
- ‘c’: conversion factor (set) ‘to SI’ for data item (elements);
- ‘t’ : permission-flag for ‘run-time tuning’ in the simulation program;
- ‘f’: ranking number of the data item in the SGS; agrees with definition order.

Examples of *linear control data items*:
- for ‘alo3p’ (2D-array), generated in feedback synthesis function ‘pbasefb()’ for pitch control , the value set is retrieved by ‘linctrl.p.v.alo3p’;
- for ‘qnlo3p’ (structure), generated in feedback synthesis function ‘pgeneratorfb()’ for generator torque control, the value subset belonging to the field ‘a’ (2D-array) is retreived by ‘linctrl.p.v.qnlo3p.a’.

For further information on the data item related fields see ‘ecndatadef()’ and ’ecndatalist()’ or just type ‘linctrl.p’, ‘linctrl.p.d.alo3p’, ‘linctrl.p.d.qnlo3p’, etc.

Program ‘parvsp’ stores all data items on one MAT-file via ‘ecndsgnstore()’. The value fields are stored as equally named MATLAB variables, which can be:
- scalar, 1D- or 2D-array of type numeric;
- 1-layer structure with up to 2D-arrays of type numeric as fields.

Further, ‘ecndsgnstore()’ saves structures that contain all dimension fields, names of data value generating M-files and run-time tuning ‘flags’:
- ‘xxxdim4’, with field contents equal to the ‘dimension fields’ ‘d’
- ‘xxxsrc4’, with field contents equal to the ‘M-source fields’ ‘s’
- ‘xxxrtf4’, with field contents equal to the ‘run-time tuning fields’ ‘t’

Program ‘parvsp’ generates wind speed data and stores the belonging MATLAB variables on MAT-file by calling ‘ecnvwroteff()’. The stored M-vars enable to generate a rotor effective wind speed signal that caters for rotational sampling, tower stagnation and windshear. Included normalised azimuth angles and wind speed variations enable to deal with rotor speed variations and different mean wind speeds. In addition an M-var is included with a stair case wind speed profile. See decription of model support M-function ‘ecnvwroteff()’. Further, ‘ecnvwroteff()’ saves structures ‘xxxdim4’ with dimension fields and ‘xxxsrc’ with the data generating M-file.

5. APPLICATION M-CODE CONTROLLER EVALUATION

The MATLAB-program ‘simvsp’ provides the simulation stage of control development for a variable speed HAWT with pitch to vane power limitation (VSP). Program ‘simvsp’ enables to use MATLAB Simulink for simulation within the structure of the ECN control tool.

Simulink allows a modular set-up of the simulation scheme through the use of so called ‘S-functions’. Besids it makes it easy to ‘on-line monitor’ simulation signals and to ‘plug in’ additional signal processing functions for scoping purposes.

The ECN control tool facilitates well organised
- use of control, turbine and site data from the design stage (§4 on ‘parvsp’);
- handling of input, state and output signals within S-functions;
- coupling of input and state to output signals within an S-function (auto);
- coupling of output to input signals between S-functions (cross);
- storage and retrieval of driving data and results of simulations.

Actual MATLAB-programs for VSP-simulation are likely to deviate from ‘simvsp’. However the program structure, the use of supporting M-functions, and the created MATLAB-variable types will be similar.

§5.1 gives a short program description with a list of the user supplied files for simulation. §5.2 and 5.3 deal with the distinghised ‘simulation substages’, i.e. item definition and performing the time-domain simulation.

5.1 Scope on simulation program ‘simvsp’

MATLAB-program ‘simvsp’ defines and performs time-domain simulation. The user provides simulation S-functions (*US-functions*) for initialisation and update of simulation states and calculation of outputs. Each US-function works with a **global** structure (SGS). SGS-fields serve as indices in signal vectors (signal handles) and as initial values and parameters (simulation data). The following US-functions and SGS’s apply in ‘simvsp’:
- ‘swindvsp()’ with SGS ‘wind’: wind speed generation;
- ‘sctrpvsp()’ with SGS ‘ctrp’: pitch angle control;
- ‘sctrgvsp()’ with SGS ‘ctrg’: generator torque control;
- ‘sturbvsp()’ with SGS ‘turb’: turbine simulation.

Further, a so called ‘run-time governing global structure’ (RGS) may be used for last minute (re)definition of data items. RGS ‘simu’ applies in ‘simvsp’.

SGS’s and RGS both pertain to ‘variable data’ and are typed as Variable related Global Structures (VGS).

In the first definition step program ‘simvsp’ creates the (empty) VGS’s by calling ‘ecnsimvarbases()’ and makes them accessible from the MATLAB-shell via ‘ecnVGSglobal’.

Program ‘simvsp’ continues with calls to the user provided M-functions that define signal handles and simulation data in VGS’s (*UD-functions for signal items and data items*):
- ‘dwindvsp()’: data and signal items in SGS ‘wind’;

- ‘dctrpvsp()’: data and signal items in SGS ‘ctrp’;
- ‘dctrgvsp()’: data and signal items in SGS ‘ctrg’;
- ‘dturbvsp()’: data and signal items in SGS ‘turb’;
- ‘dsimuvsp()’: data items in RGS ‘simu’.

Each of the first four UD-functions belongs to a US-function and in turn calls to:
- ‘ecnsigdef()’: create signal item in SGS;
- ‘ecnsigmap()’: extend signal item for signal coupling within S-function;
- ‘ecndefsupp()’[2]: scans *by-S-function-requested* signal and data items and
  - compare requested signal items with signal definitions on calling M-function,
  - create data items via ‘ecndatadef()’ for requested items that exist on MAT-file.

Wind speed data also appear as a data item, viz. in SGS ‘wind’ for S-function ‘swindvsp()’.

The fifth UD-function is used for run-time data definition and in turn calls to:
- ‘ecndatadef()’: create run-time data items (‘r’-items) in RGS and SGS;
- ‘ecnrtsupp()’[3]: compare *by-any-S-function-requested* data items with ‘r’-item definitions on calling M-function and, via ‘ecndatadef()’,
  - create requested SGS-‘r’-items that match to defined RGS-‘r’-items,
  - (re)create requested SGS-‘b,p’-items that match to defined SGS-‘r’-items **iff run-time flag ‘t’$\neq$1** (see §4.2).

After having called the UD-functions program ‘simvsp’ calls to ‘ecndefcheck()’, which checks if all requested data and signal items have been created in the SGS’s.

In the last definition step program ‘simvsp’ calls ‘ecnsigmap()’ for extension of signal items with cross signal couplings.

Text output of supporting M-functions is copied to a definition file:
- ‘ecndatalist()’: run-time modifications, called from ‘ecnrtsupp()’ ;
- ‘ecndefcheck()’: match of requested to created items, called from ‘simvsp’;
- ‘ecnmaplist()’: cross and auto signal couplings, called from ‘simvsp’.

Other supporting M-functions will log messages on the definition file at irregular, but non-fatal, execution (see helps on M-functions).

At time-domain simulation, program ‘simvsp’ calls to the US-functions for initialisation and update of their state vectors and for calculation of output vectors. This can be done with the Simulink kernel via calls to ‘open(<mdlfile4vsp>)’ and ‘sim(<mdlfile4vsp>,<simu.r.v.TdurSim>)’. Argument ‘mdlfilevsp’ is the name of the MDL-file with the Simulink simulation scheme by figure 5.1; ‘simu.r.v.TdurSim’ the simulation time in seconds.

The cross signal couplings enable simple connection of the US-functions ‘swindvsp()’, ‘sturbvsp()’, ‘sctrpvsp()’ and ‘sctrgvsp()’ via their output vectors in accordance with figure 5.1.

The simulation also includes the user provided S-functions ‘awindvsp()’, ‘aturbvsp()’, ‘actrpvsp()’ and ‘actrgvsp()’ for data acquisition. The *UA-functions* have sample times <SGS>.r.v.TdelAcq. These follow from definition of ‘simu.r.v.TdelAcq’ in ‘dsimuvsp()’ via ‘ecndatadef()’, after having been processed with ‘ecnrtsupp()’.

At initialisation the UA-functions create matrices <SGS>.y.a for data storage. The sizes agree with the output vectors of the US-functions and the ratio of duration

[2] ecndefsupp() read S- and M-function fi les via ‘ecnsfuncscan()’ and ‘ecnfi lescan()’
[3] ecnrtsupp() reads defi ning M-function fi le via ‘ecnfi lescan()’

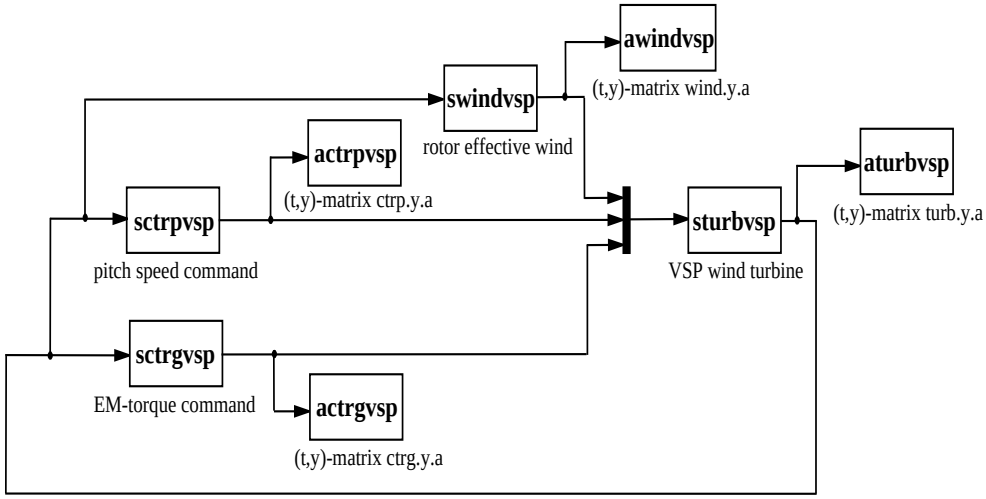

Figure 5.1 *Layout simulink model for typical variable speed pitch to vane wind turbine*

<SGS>.r.v.TdurSim and sample times <SGS>.r.v.TdelAcq; <SGS>.r.v.TdurSim obtained from 'simu.r.v.TdurSim' in 'dsimuvsp()'). Output values are stored as columns in the matrices at $t = 0$ and after every <SGS>.r.v.TdelAcq seconds. The UA-functions use their state variable as array index.

The time-domain simulation can also be performed with user provided M-function 'emusimulinkvsp()'; this M-function implements the required functionality of the Simulink kernel. For numerical integration it applies its built-in function 'intrk4()'.

After simulation, support M-function 'ecnylist()' can be used for plotting and printing of acquired data with signal names and dimensions included.

The user M-files for VSP-specific MATLAB Simulink oriented simulation are (floppy dir. 'VSPITCH/M'):

- 'simvsp.m', with the M-program that masters the wind turbine simulation;
- 'dwindvsp.m', 'dturbvsp.m', 'dctrpvsp.m', 'dctrgvsp.m', 'dsimuvsp.m' with M-functions for data and signal definition for the simulation (*UD-functions*);
- 'swindvsp.m', 'sturbvsp.m', 'sctrpvsp.m', 'sctrgvsp.m', with main S-functions accommodating the equations of motion for the simulation (*US-functions*);
- 'awindvsp.m', 'aturbvsp.m', 'actrpvsp.m', 'actrgvsp.m', with auxiliary S-functions for data acquisition of simulation results (*UA-functions*);
- 'emusimulinkvsp.m' with M-functions for emulation of the Simulink kernel restricted to numerical integration and time-discrete state updates for VSP-simulation.

Program 'simvsp' uses two settings for selection of MAT-files:

- 'turbidf', the turbine idenfitifier (default 'TurVSP');
- 'VwMeanBase' [m/s], the wind speed level for selection of wind data file (default 12.0).

These settings imply the use of parameters and wind data from files

- paramvsp<turbidf>.mat : machine, site and control data,
- vwe<turbidf><creationwindspeed>.mat : rotor effective wind speed data

If more wind speed files exist for <turbidf> then the one is selected with <creationwindspeed> as close as possible to <VwMeanBase>.

It is clear that the default turbine identifier agrees with those in the names of the parameter data file and wind speed file resulting from 'parvsp' in §4. It is also clear that the wind file selection procedure will make a selection list of only one file and will 'select' the supplied wind speed file by 'parvsp'.

5.2 Definition of simulation data and signal handles

The definition part of 'simvp' creates data and signal items in the in the S-function specific global structures (SGS) and the run-time governing global structure (RGS; data only). For involved calls to user provided and supporting M-files see §5.1. Data items make simulation data accessible in the turbine specific S-functions. Signal items enable signal handling in these *US-functions*:
- name-based indexing of input, state and output vectors;
- signal vector coupling via index arrays.

The items exist as fields in substructures of the SGS's [and RGS]:
- data items in substructures 'p', 'b, 'r';
- signal items in substructures 'u', 'x', 'y'.

As concerns the RGS, only the 'r'-field will be used, viz. for last minute data. Further, the RGS has the empty field 'RGS', which makes it recognisable as RGS. All SGS also have the substructure 'm' for miscellaneous items.

During definition, the *requested* data and signal items for simulation are scanned from the US-function files and stored as 'name lists' in global structure 'mainvarnames'. Such a name list is a field of type 'cell string array' in an item related substructure, like 'mainvarnames.ctrp.p.list' for data items in the 'p'-group of SGS 'ctrp'.

The next two subsections deal with data item and (basic) signal item creation; the last two with exension of signal items for auto and cross couplings.

5.2.1 Data items in global structures for simulation data

A data item in a turbine specific S-function exists as a set of fields in substructure 'p', 'b', or 'r' of an SGS. These substructures are also referred to as data groups:
- 'p'-group: recommended for 'parameter type' data items;
- 'b'-group: recommended for 'initial value type' data items;
- 'r'-group: recommended for 'run-time type' data items.

According to §4.2 a particular data item <d> in data group <g> of global structure <SGS> occupies the field <d> in substructures 'v', 'n', 's', 'd', 'c', 't' and 'f' of <SGS>.<g>. §4.2 also tells that a created item can be modified just before simulation if the 't'-field equals 1 or that a still missing data item can then be created (use of 'r'-group). This *run-time data definition mechanism* enables easy filing of a set of simulation runs for a range of parameter values (parameter sweep):

> A simulation program like 'simvsp' could incorporate a looping mechanism on the 'r'-item defining function like 'dsimusvp()' and the execution of time-domain simulation. Parameter data and initial values are efficiently logged if:
> – storage of the items of each *'p'-, 'b'-, 'r'-group* preceeds simulation 1;
> – storage of the items of only each *'r'-group* preceeds simulation 2,3,. . .

The storage of only ‘r’-items in later runs makes the parameters of these runs completely traceable because all run-time defi ned data *always* appear in the ‘r’-group of SGS’s.

Supporting M-function ‘ecndatalist()’ enables formatted data filing with dimensions and generating M-files included (see help on ‘ecndatalist()’).

Examine M-files ‘sctrpvsp.m’ and ‘sctrgvsp.m’ for use of data items ‘alo3p’ (2D-array) and ‘qnlo3p’ (1-layer structure) in data groups ‘ctrp.p’ and ‘ctrg.p’. These data have already been dealt with as *linear control items* in the design stage (§4.2).

5.2.2 Signal items in global structures for signal handling

A signal item in a turbine specific S-function exists as a set of fields in substructure ‘u’, ’x’, or ’y’ of an SGS. These substructures are also referred to as signal groups:
- ‘u’-group : mandatory for input vector to S-function
- ‘x’-group : mandatory for state vector in S-function
- ‘y’-group : mandatory for output vector of S-function

A created signal item <s> in signal group <g> of global structure <SGS> occupies the field <s> in the following substructures of <SGS>.<g>:
- ‘i’: index-value(set) for the signal item (elements) in the signal vector;
- ‘n’: name of the signal item;
- ‘d’: dimension(set) of the signal item (elements)
- ‘c’: conversion factor (set) ‘to SI’ for signal item (elements)
- ‘f’: ranking number of the signal item in the SGS; agrees with definition order.

A ‘set’ applies if the signal item pertains to a subvector in the signal vector instead of a scalar.

Defining M-function ‘dctrpvsp()’ includes definition directives for state vector element ‘ProductionState’ and state subvector ‘xNrpmLo3p’:

```
ecnsigdef('ctrp','x','xNrpmLo3p',size(ctrp.p.v.alo3p,1),'nodim',fid);
ecnsigdef('ctrp','x','ProductionState', 1,'nodim',fid);
```

Subvector ‘xNrpmLo3p’ belongs to a low pass filter implemented in state space format on S-function ‘sctrpvsp()’ with simulation data ‘alo3p’ as state transition matrix (see §5.2.1). Creation of the signal items yields the integer scalar ‘ctrp.x.i.ProductionState’ and integer 1D-array ‘ctrp.x.i.xNrpmLo3p’. Examine M-file ‘sctrpvsp.m’ for this index-value and index-value-set being used in the state vector ‘x’ as:

```
xNew(ctrp.x.i.xNrpmLo3p) = ctrp.p.v.alo3p*x(ctrp.x.i.xNrpmLo3p) + ctrp.p.v.blo3p*Nrpm;
...
if x(ctrp.x.i.ProductionState) == 1,
```

5.2.3 Auto signal couplings via extended signal items

The defining M-functions include mapping directives for array-wise coupling of input or state vector elements to output vector elements (auto signal coupling). The following directive in defining function ‘dctrpvsp()’ enables state to output mapping in S-function ‘sctrpvsp()’

```
ecnsigmap('ctrp','y','ctrp','x',fid);
```

Such a directive creates two index-arrays in the same S-function specific global structure: source SGS <srcSGS> and destination SGS <destSGS> are the same. Ofcourse, the source signal group <srcSIG> (‘u’ of ‘x’) differs from the destination signal group <destSIG> (‘y’).

The index-arrays are non-empty if source and destination signal vector have equally named elements or subvectors. The index array for the source vector is field id2.<destSGS>.y in source group structure <srcSGS>.u or <srcSGS>.x. Field idf.<srcSGS>.u or idf.<srcSGS>.x in destination group structure <destSGS>.y is used in the destination vector.

The mapping directive also creates substructures ‘ed2’ and ‘mp2’ in the source group structure. The field <destSGS>.y respectively is:
- boolean, telling if all source and destination signals have equal dimension;
- character string, being a formatted page with established signal couplings.

Substructures ‘edf’ and ‘mpf’ are created in the destination group structure with field <srcSGS>.u or <srcSGS>.y; same contents as in corresponding source group fields.

The next conditional array-wise *state to ouput assignment* applies in ‘sctrpvsp()’:

```
if ctrp.y.edf.ctrp.x == 1;
  y(ctrp.y.idf.ctrp.x,1)  = x( ctrp.x.id2.ctrp.y );
else
  y(ctrp.y.idf.ctrp.x,1)  = x( ctrp.x.id2.ctrp.y ) .* ...
         ctrp.x.cnucat( ctrp.x.id2.ctrp.y ) ./ ctrp.y.cnucat( ctrp.y.idf.ctrp.x);
end
```

The second directive applies if equally named state and output variables have different dimensions. The arrays ‘ctrp.x.cnucat’ and ‘ctrp.y.cnucat’ contain conversion factors to SI-units. These are created at signal definition in the defining M-function (see help on ‘ecnsigdef()’).

5.2.4 Cross signal couplings via extended signal items

Program ‘simvsp’ by itself include mapping directives for array-wise coupling of output vector elements of one S-function to input vector elements of another (cross signal coupling). The following directives enable output to input mapping between S-functions ‘sturbvsp()’ and ‘sctrpvsp()’:

```
ecnsigmap('ctrp','u','turb','y');% turbine y-elements to pitch-control u-elements
ecnsigmap('turb','u','ctrp','y');% pitch-control y-elements to turbine u-elements
```

Directives like these each create one index-array in two S-function specific global structures: source SGS <srcSGS> differs from destination SGS <destSGS>. The source and destination signal groups <srcSIG> and <destSIG> always equal ‘y’ and ‘u’.

The index-arrays are non-empty if source and destination signal vector have equally named elements or subvectors. The index array for the source vector is field id2.<destSGS>.u in source group structure <srcSGS>.y. Field idf.<srcSGS>.y in destination group structure <destSGS>.u is used in the destination vector.

The mapping directive also creates substructures ‘ed2’ and ‘mp2’ in the source group. The field <destSGS>.u respectively is:

- boolean, telling if all source and destination signals have equal dimension;
- character string, being a formatted page with established signal couplings.

Substructures ‘edf’ and ‘mpf’ are created in the destination group with field <srcSGS>.y; same contents as in corresponding source group fields.

The M-code that is executed just after an S-function has been entered (entry-M-code) contains the *output to input assignments*. The input vector to S-function ‘sctrpvsp()’ is the output vector of ‘sturbvsp()’, which is typed as ‘y2ctrp’. Actually, ‘sctrpvsp()’ only uses those elements of ‘y2ctrp’ that agree with the input signal definitions in M-function ‘dctrpvsp()’. These elements set up *the* input vector ‘u’ to ‘sctrpvsp()’. The next array-wise *output to input assignment* applies:

```
u = zeros(ctrp.u.sz,1);
if  ctrp.u.edf.turb.y == 1;
  u(ctrp.u.idf.turb.y)  = y2ctrp( turb.y.id2.ctrp.u );
else
  u(ctrp.u.idf.turb.y)  = y2ctrp( turb.y.id2.ctrp.u ) .* ...
         turb.y.cnucat( turb.y.id2.ctrp.u ) ./  ctrp.u.cnucat(ctrp.u.idf.turb.y);
end
```

If equally named outputs of ‘sturbvsp()’ and inputs to ‘sctrpvsp()’ have different dimensions then the conversion arrays ‘turb.y.cnucat’ and ‘ctrp.u.cnucat’ apply (see ‘ecnsigdef()’).

The *output to input assignments* in the entry-M-code of ‘sturbvsp()’ map elements of the stack of output vectors of ‘swindvsp()’, ‘sctrpvsp()’ and ‘sctrgvsp()’ to the actual input vector ‘u’ of ‘sturbvsp()’. The stack of output vectors is typed as ‘y2turb’. Input vector ‘u’ only uses those elements of ‘y2turb’ that agree with the input definitions in ‘dturbvsp()’. Pitch control outputs from ‘sctrpvsp()’ are fed into the input vector of ‘sturbvsp()’ as follows:

```
...
up = zeros(turb.u.sz,1);
if  turb.u.edf.ctrp.y == 1,
  up(turb.u.idf.ctrp.y)  = y2turb( ctrp.y.id2.turb.u + wind.y.sz);
else
  up(turb.u.idf.ctrp.y)  = y2turb( ctrp.y.id2.turb.u + wind.y.sz) .* ...
          ctrp.y.cnucat( ctrp.y.id2.turb.u ) ./  turb.u.cnucat( turb.u.idf.ctrp.y);
end
...
u  = uw+up+ug;
```

The entry-M-code of ‘sturbvsp()’ also contains similar assignments to ‘uw’ and ‘ug’ for wind speed and generator control related inputs. Note that offset ‘wind.y.sz’ in ‘y2turb’ caters for the subdivison of ‘stacked vector y2turb’.

If two or more output vectors in a ‘stacked vector’ have equally named *in ‘u’ used* **elements then a single input becomes the sum of output variables from two or more S-functions.**

5.3 Performing time domain simulation in ‘simvsp’

The simulation run part of ‘simvsp’ calls to the turbine specific S-functions for initialisation and update of state vectors and for output vector calculation. These S-functions are the building blocks for modular simulation of a typical variable speed pitch to vane wind turbine. They effectuate the simulation when called by function ‘emusimulinkvsp()’ or by Simulink via an MDL-file for the simulation layout by fig. 5.1 (5.1).

The following four subsections clear up the mechanism of time domain simulation as well as the implemented turbine and control features and monitoring options by focusing on:
- layout of S-functions in simulation and their interactions;
- protocol for numerical simulation via S-functions;
- functionality of turbine specific S-functions;
- postprocessing of simulation results.

5.3.1 Layout and interaction S-functions

The simulation run part executes the simulation stage dependent code partitions on the S-function files. These code partitions, implemented as (sub) M-functions, pertain to initialisation and update and/or derivative calculation of state vector elements, and to calculation of ouput vector elements. Function emusimulinkvsp() activates these code partitions for for the following ‘main’ and ‘auxiliary’ S-function set:
- swindvsp(), sturbvsp(), sctrpvsp(), sctrgvsp(), ‘main-set’ for HAWT-simulation
- awindvsp(), sturbvsp(), sctrpvsp(), sctrgvsp(), ‘aux-set’ for data acquisition

These functions are used here as regular M-functions; see ‘Application option 2’ (Ao2) in the function file descriptions in the next section. The ’Simulink execution protocol’ is emulated, but only as far as concerns:
- fixed step-size integration for continuous states, and
- discrete state handling with constant sample time,

because the functions do not demand any other Simulink functionality.

Time discrete functions process input values and provide output values on discrete time points only, say k*Ts for k=0,1,2,... with associated sample number k" = k+1:
- state vector ‘xd’ is updated through equation:
  - xd(k"+1) = f(xd(k"),u(k");
- output vector ‘y’ is provided for the ‘half open’ time interval [k*Ts,(k+1)*Ts) (point (k+1)*Ts NOT included) through equation:
  - y(k") = g(xd(k"),u(k").

Sample times ‘Ts’ are obtained in the initialisation calls to discrete functions.

The pure time discrete main functions swindvsp(), sctrpvsp() and sctrgvsp() provide the wind speed and the command values for pitch speed and EM-torque for function sturbvsp(). They are driven by output elements of sturbvsp() only and have fixed sample times ‘Tdelw’, ‘Tdelp’ and ‘Tdelg’ (locals in emusimulink()). As they do NOT have input feedthrough (y(k")=g(x(k"))), they provide y(k")-elements for other functions in the interval [k*Ts,(k+1)*Ts) that are actually obtained from u(k"-1)-elements (t=(k-1)*Ts). So these functions:

- make available processed time-continuous inputs with a delay that periodically varies from 1*Ts to 2*Ts, say 1.5*Ts on an average.

Main function sturbvsp() is a hybrid function for wind turbine and actuator simulation. A time discrete part, with state subvector 'xtd', delays the pitch speed command value with a (varying) number of the small sample time 'Tdelt' (local). After delay, the pitch command enters a time continuous part, with state subvector 'xtc' and fixed integration step 'H' (=Tdelt; local setting). Function sturbvsp() is driven by output vector elements of all three discrete time main functions. For some of its 'y'-elements, sturbvsp() DOES have input feedthrough (y(t)= ... f(xc(t),u(t),xd(k")) with time t in [k*Ts,(k+1)*Ts)).

The main S-functions are linked via the complete output vectors; they use index-arrays for selection of needed elements (see notes on 'inter S-function signal mapping' in the function file comments).

The auxiliary functions a<SGS>vsp(), with <SGS>=wind, turb, sctrp, sctrg, are pure discrete time with fixed sample times 'Tdelwa', 'Tdelta' and 'Tdelpa' and 'Tdelpa' (locals). They periodically store the output vector from the corresponding main functions s<SGS>vsp(), together with a time stamp, in 2D-arrays that are fields of the S-function specific global structures. This storage, together with 'outputting' the stored data', occurs delayless in the output section of these functions, so they DO have input feedthrough (y(k")=f(u(k"))

5.3.2 Simulation kernel at interconnected S-functions

The numerical simulation uses the following auxiliary variables for timing:

- variable 't' is the actual simulation time point, starting with '0';
- variables 'kw", kp", kg", kt", kwa", kpa", kga" and kta"' are the sample numbers for the discrete time function[part]s, starting with '1' for 't==0'; they pertain respectively to swindvsp(), sctrpvsp(), sctrgvsp(), sturbvsp(), awindvsp(), actrpvsp(), actrgvsp() and aturbvsp().

Discrete time output calculation and state update is to occur as soon as continuous time 't' has become equal to '(k%"-1) * Tdel%', for % = w, p, g, t, wa, pa, ga, ta. In the description of the cyclic part of the numerical simulation such an event is referred to as a 'time hit' for sample no k%". Directives that are dependent on time hits for e.g. kw", kp" and kg" are formulated as:

@hit k%", %=w,p,g: <directives>

Initialisation

- calls with 'Flag == 100' and output processing/interpretation provide:
  - . sample times : Tdelw Tdelp Tdelg Tdelt Tdelwa Tdelpa Tdelga Tdelta
  - . 't0'-state values : xwdUpd xpdUpd xgdUpd xt0 xwadUpd xpadUpd xgadUpd xtadUpd
  - . split-up of xt0 in time-continuous part xtcInt and time-discrete part xtdUpd
  - . sequence of output calculation based on 'direct feedthrough defi nition':
    1. swindvsp(), sctrpvsp(), sctrgvsp() ; mutually interchangeable
    2. sturbvsp(), actrpvsp(), actrgvsp(), awindvsp() ; mutually interchangeable
    3. aturbvsp()

- assign start values for time variable and sample no's discr time function[part]s:
  - . t = 0
  - . kw" = kp" = kg" = kt" = kwa" = kpa" = kga" = kta" = 1

Cyclic part with time increment H

- assign available cont. state vector integration result to actual state vector:
  . xtc(t) = xtcInt
- check on ‘time hits’ for discrete time function[part]s; all hits in 1st cycle
- @hit k%", %=w,p,g,t,wa,pa,ga,ta: assign availabe state updates to actual states:
  . x%d(k%") = x%dUpd
- @hit k%", %=w,p,g: calculate output of swindvsp(), sctrpvsp(), sctrgvsp():
  . y%(k%") = g(x%d(k%"))
- @hit k%a", %=w,p,g: calculate outputs of awindvsp(), actrpvsp(), actrgvsp():
  . y%a(k%a") = g(y%(k%"))
- calculate output of continuous time function:
  . yt(t)= g(xtc(t), xtd(kt"), [yw(kw");yp(kp");yg(kg")])
- @hit kta": calculate output of aturbvsp():
  . yta(kta") = g(yt(t))
- if ‘t >= TdurSim’, break off simulation by jumping out of cyclic part
- RK4-integration of continuous state subvector xtc from t to t+H by calculating:
  . derivative for time t : xtc0’ = fc(xtd(kt"),xtc(t),[yw(kw")...])
  . 1st ‘intermediate’ state: xtc1 = xtc(t) + H/2*xtc0’
  . 1st ‘intermediate’ deriv: xtc1’ = fc(xtd(kt"),xtc1,[yw(kw")...])
  . 2nd ‘intermediate’ state: xtc2 = xtc(t) + H/2*xtc1’
  . 2nd ‘intermediate’ deriv: xtc2’ = fc(xtd(kt"),xtc2,[yw(kw")...])
  . 3rd ‘intermediate’ state: xtc3 = xtc(t) + H*xtc2’
  . 3rd ‘intermediate’ deriv: xtc3’ = fc(xtd(kt"),xtc3,[yw(kw")...])
  . defi nite ‘integrated’ state: xtcInt = xtc(t)+ H*(xtc0’+ 2*(xtc1’+xtc2’)+ xtc3’)/6;
- @hit k%", %=w,p,g,t: calculate discrete state updates for main functions:
  . x%dUpd = f(x%d(kw"), yt(t)), for % = w, p, g
  . xtdUpd = f(xtc(t), xtd(kw"), [yw(kw"); yp(kp"); yg(kg")])
- @hit k%a", %=w,p,g,t: calculate discrete state updates for auxiliary functions:
  . x%adUpd = f(x%ad(k%a"), y%(k%"))
- @hit k%", %=w,p,g,t,wa,pa,ga,ta: sample no’s increment by 1:
  . k%" = k%" + 1
- update time variable with integration time step: t = t + H

Of course, also Simulink can be used for execution of the simulation. Simulation command SIM() (see 4.30 in using simulink ,vs 3) activates the actual simulation. The timing data are made known for all S-functions.

Simultaneous integration of two ‘continuous time functions’

In order to make clear the simulation mechanism by Simulink, the above mentioned integration scheme is extended here for use of simultaneous integration of two time-continuous S-functions.

Assume extra, pure continuous time, function with:
- state vector xEc
- output vector yE (NO input feedthrough) that is fed back as input to sturbvsp()
- input vector equal to output vector of sturbvsp() (=yt)

Cyclic part with time increment H:

- assign available cont. state vector integration results to actual state vectors:

. xEc(t) = xEcInt
. xtc(t) = xtcInt
. ...

- calculate outputs of continuous time function (prescribed order!):
  . yE(t)= g(xEc(t))
  . yt(t)= g(xtc(t), xtd(kt"), [yE(t); yw(kw") ;yp(kp") ;yg(kg")])
- RK4-integration of continuous states xtc and xEc from t to t+H by calculating:
  . derivatives for time t:

    xEc0' = fEc(xEc(t),yt(t))
    xtc0' = ftc(xtd(kt"),xtc(t),[yE(t); yw(kw")...])

  . 1st 'intermediate' states:

    xEc1 = xEc(t) + H/2*xEc0'
    xtc1 = xtc(t) + H/2*xtc0'

  . 1st 'intermediate' outputs (presribed order!):

    yE1 = g(xEc1)
    yt1 = g(xtc1, xtd(kt"), [yE1 ; yw(kw")...])

  . 1st 'intermediate' deriv:

    xEc1' = fc(xEc1, yt1)
    xtc1' = fc(xtd(kt"),xtc1,[yE1 ; yw(kw")...])

  . 2nd 'intermediate' states:

    xEc2 = xEc(t) + H/2*xEc1'
    xtc2 = xtc(t) + H/2*xtc1'

  . 2nd 'intermediate' outputs (presribed order!):

    yE2 = g(xEc2)
    yt2 = g(xtc2, xtd(kt"), [yE2 ; yw(kw")...])

  . 2nd 'intermediate' deriv:

    xEc2' = fc(xEc2, yt2)
    xtc2' = fc(xtd(kt"),xtc2,[yE2 ; yw(kw")...])

  . 3rd 'intermediate' states:

    xEc2 = xEc(t) + H*xEc2'
    xtc2 = xtc(t) + H*xtc2'

  . 3rd 'intermediate' outputs (prescribed order!):

    yE3 = g(xEc3)
    yt3 = g(xtc3, xtd(kt"), [yE3 ; yw(kw")...])

  . 3rd 'intermediate' deriv:

    xEc3' = fc(xEc3, yt3)
    xtc3' = fc(xtd(kt"),xtc3,[yE3 ; yw(kw")...])

  . defi nite 'integrated' states:

    xEcInt = xEc(t)+ H*(xEc0'+ 2*(xEc1'+xEc2')+ xEc3')/6;
    xtcInt = xtc(t)+ H*(xtc0'+ 2*(xtc1'+xtc2')+ xtc3')/6;

  . ...
  . update time variable with integration time step: t = t + H

5.3.3 Turbine specifi c simulation functions

The next four paragraphs list the function headers of the following turbine specific S-functions for variable speed pitch to vane wind turbine simulation:

- swindvsp(); wind speed generation, on S-file swindvsp.m
- sctrpvsp(); pitch angle control, on S-file sctrpvsp.m
- sctrgvsp(); generator torque control, on S-file sctrgvsp.m
- sturbvsp(); turbine simulation, on S-file strubvsp.m

MATLAB file swindvsp.m

```
SWINDVSP(): governing equations wind speed generation at var-spd HAWT simulation

CALL: [sys,x0,str,Ts]=swindvsp(t,x,y2wind,SimStatusFlag); % 'str' always empty

Associated global structure 'wind' and defining function 'dwindvsp()'.

Functionality:
 * samples the rotor effective wind speed variation which is azimuth angle dependent
   (including rotational sampling caused by wind shear and tower shadow);

 * scales the mean wind speed from the design value by the simulation data towards
   the target simulation mean wind speed 'VwMean' (multiplier 'muHubv') and,
   optionally, towards the gust wind speed 'VwMean+VwAmplGust' (additional
   multiplier 'muGust').

 Actually the wind speed has a 'feedthrough dependence' on the azimuth angle.
 In order  to exclude an algebraic loop between the turbine S-function and
 wind S-function, the wind speed output signals of swindvsp() are delayed for
 one period (i.e. sample time TdelSimv of the input wind series from file; see
 dwindvsp).

Run Time items (define in dsimuvsp()):

  VwMean           : target average wind speed in simulation run
  VwGrow4Adap2Mean : grow rate of average wind spd from 'start value Uhubv' to VwMean
  VwGrow4Gust      : grow rate of gust (m/s^2)
  VwAmplGust       : amplitude of gust
  VwTbegGust       : time instance for starting the gust
  VwTdurGust       : duration of gust (at gust level)
  TdurSim          : duration of simulation
  TdelAcq          : sample time for data acquistion of outputs Vw0p and VwEff

Signal vectors:
 * y = ... (output vector)
    muHubv      [-]: multiplier on 'design mean wind speed'; grows to target/design
    muGust      [-]: multipl. on mean wind speed; grows to gust/mean and falls again
    VwEff     [m/s]: rotor effective wind speed incl. turbulence & tower shadow
    Vw0p      [m/s]: rotor uniform wind speed ('0p-mode')

 * x = ... (state vector)
    i0p         [-]: index in wind speed data array; azimuth driven
    muHubv      [-]: multiplier on 'design mean wind speed'; grows to target/design
    muGust      [-]: multipl. on mean wind speed; grows to gust/mean and falls again
    VwEff     [m/s]: rotor effective wind speed incl. turbulence & tower shadow
    Vw0p      [m/s]: rotor uniform wind speed ('0p-mode')

 * y2wind = ... ('global' input vector)
```

```
   <output vector of sturbvsp(); see sturbvsp.m>

Function swindvsp() uses fields of global structure 'wind' as indices in signal
vectors y, x and
 * u = ... ('local' input vector)
    Azim [rad]: rotor azimuth angle

Field creation in function dwindvsp() (see dwindvsp.m).

Application options (Ao):
1. S-function swindvsp() in equally named model block in Simulink .mdl-file, or
2. M-function swindvsp() called by M-function emusimulinkvsp() (Simulink emulation)

Function file hierarchy:
  sctrpvsp()
    mdlInitializeSizes() : called at 'SimStatusFlag'  = 0 (@AO1) or = 100 (@AO2)
    mdlOutputs()         : called at 'SimStatusFlag'  = 3 (@AO1 and @AO2)
    mdlUpdate()          : called at 'SimStatusFlag'  = 2 (@AO1 and @AO2)
```

MATLAB file sctrpvsp.m

```
SCTRPVSP(): governing equations pitch speed commanding for var-spd pitch to vane HAWT

CALL: [sys,x0,str,Ts]=sctrpvsp(t,x,y2ctrp, SimStatusFlag); % 'str' always empty

Associated global structure 'ctrp' and defining function 'dctrpvsp()'.

Functionality:
 * discrete time, state controlled pitch speed command value calculation for power
   production; effective delay ~ 1.5*sample time (see NOTE 2); 1st order low pass
   rotor speed and actual rotor speed used for full->partial and partial->full
   transient in production state; production state  values:
    0 <=> partial load
    1 <=> full load
 * data conditioning for feedback and feedforward control:
     - low-pass filter with cut-off frequency < 3p ('lo3p'); order N inverse
       chebyshev low-pass filter (order N state space impl.)
     - band stop filter in band around 2nd asymm. lead-lag blade bending mode
       ('bandstop'); order N inv. cheby. stop filter (order 2N state space impl.)
 * partial load:
    - pitch angle target value from low-pass 3p filtered rotor speed (table);
    - pitch speed command value from P-feedback of pitch angle error.
 * full load:
    - rotor speed setpoint adaptation to 1st order filtered pitch angle
    - pitch speed command value  for regular rotor speed control by parallel
      (band stop, WSE and EWFF optional through 'buttons').
       . PD-feedback on lo3p rotor speed and lo3p & band stop rotor acceleration
       . feedforward of the estimated wind speed (EWFF); wind speed estimation (WSE)
         on lo3p rotor speed, lo3p electric power and lo3p & band stop rotor acc;
         feedforward of 1st order low-pass d/dt(WSE) towards pitch speed command;
         for lo3p rotor speed > upper threshold: EWFF only towards feather. position
         for lo3p rotor speed < lower threshold: EWFF only towards working position
    - forced rotor speed limiter; pitch to vane over fixed definable interval with
      fixed definable pitch speed; overruls PD and EWFF if necessary;
    - dynamic inflow compensation through lead-lag filter in of case regular rotor
      speed; optional through 'button'.
```

```
 * pitch speed limitation
    - pitching beyond cams avoided
    - pitch speed command value within limits
    - inactivity zone with hysteresis avoids unneccesary small pitching actions

Signal vectors:
 * y = ... (output vector)
    NrpmRef       [rpm]: internally used rotor speed setpoint
    dThdtset     [dg/s]: command value for pitch speed servo in VSP wind turbine
    dThdtsetPD   [dg/s]: (unbound) contribution of PD rotor speed feedback to dThdtset
    dThdtsetEWFF [dg/s]: (unbound) contrib. of fed forward wind speed est. to dThdtset
    dThdtsetFuz  [dg/s]: contribution of forced rotor speed limiter to dThdtset
    VwEst         [m/s]: estimated wind speed
    NrpmLo3p      [rpm]: low-pass 3p filtered rotor speed
    xNrpmLo3p       [-]: state vector of low-pass 3p filter for rotor speed
    ThLo3p         [dg]: low-pass 3p filtered pitch angle
    PkWLo3p        [kW]: low-pass 3p filtered electric pwoer
    ProductionState [-]: production state (values 0 / 1)
    BusyFuzCtrl     [-]: state of forced rotor speed limiter (0/1)
    VwEstState      [-]: state of wind speed estimater (0/1)
    VwEstValid      [-]: validity boolean for wind speed estimation (0/1)
    dThdtsetState   [-]: state of inactivity zone on pitch speed command value (0/1)

 * x = ... (state vector)
    NrpmSmpOld    [rpm]: 1 sample time delayed rotor speed input value
    NrpmRef       [rpm]: internally used rotor speed setpoint
    dThdtset     [dg/s]: command value for pitch speed servo in VSP wind turbine
    dThdtsetPD   [dg/s]: (unbound) contribution of PD rotor speed feedback to dThdtset
    dThdtsetEWFF [dg/s]: (unbound) contrib. of fed forward wind speed est. to dThdtset
    dThdtsetFuz  [dg/s]: contribution of forced rotor speed limiter to dThdtset
    VwEst         [m/s]: resuting wind speed from wind speed estimation procedue (WSE)
    TaeLimFilt     [Nm]: 1st order low-pass value for aero-torque upper limit for WSE
    TaeRconFil      Nm]: 1st order low-pass value of reconstructed aero-torque
    NrpmLo3p      [rpm]: low-pass 3p filtered rotor speed
    NrpmDdtLo3p     [-]: low-pass 3p filtered rotor acceleration
    NrpmDdtBnt      [-]: band stop & low-pass 3p filtered rotor acceleration
    xNrpmLo3p       [-]: state vector of low-pass 3p filter for rotor speed
    xNrpmDdtLo3p    [-]: state vector of low-pass 3p filter for rotor acceleration
    xNrpmDdtBnt     [-]: state vector of band stop filter for rotor acceleration
    ThLo3p         [dg]: low-pass 3p filtered pitch angle
    xThLo3p         [-]: state vector of low-pass 3p filtered pitch angle
    PkWLo3p        [kW]: low-pass 3p filtered electric pwoer
    xPkWLo3p        [-]: state vector of low-pass 3p filtered electric pwoer
    DIcomp          [-]: state vector of dynamic inflow compensating lead-lag filter
    ProductionState [-]: production state (values 0 / 1)
    BusyFuzCtrl     [-]: state of forced rotor speed limiter (0/1)
    VwEstState      [-]: state of wind speed estimater (0/1)
    VwEstValid      [-]: validity boolean for wind speed estimation (0/1)
    dThdtsetState   [-]: state of inactivity zone on pitch speed command value (0/1)
    NrpmSwitch    [rpm]: 1st order low-pass rotor speed for full->part transient

 * y2ctrp = ... ('global' input vector)
    <output vector of sturbvsp(); see sturbvsp.m>

Function sctrpvsp() uses fields of global structure 'ctrp' as indices in signal
vectors y, x and
 * u = ... ('local' input vector)
    OmegaG  [rpm]: generator rotor speed (slow shaft equivalent)
```

```
    Th       [dg]: pitch angle
    Pe       [kW]: generated electric power

Field creation in function dctrpvsp() (see dctrpvsp.m).

Application options (Ao):
1. S-function sctrpvsp() in equally named model block in Simulink .mdl-file, or
2. M-function sctrpvsp() called by M-function emusimulinkvsp() (Simulink emulation)

Function execution depends on value 'SimStatusFlag':
 case   0 : def. dimensions & sample time ('sys'&'Ts'); init. states ('x0') (@Ao1)
 case 100 : def. dimensions & sample time ('sys'&'Ts'); init. states ('x0') (@Ao2)
 case   3 : calc. output vector ('sys'=  'y'(k) = g('x'(k))')          (@Ao1&@Ao2)
 case   2 : update state vector ('sys'='x'(k+1) = f('x'(k),'u'(k))     (@Ao1&@Ao2)
```

MATLAB file sctrgvsp.m

```
SCTRGVSP(): governing equations EM-torque commanding for var-spd pitch to vane HAWT

CALL: [sys,x0,str,Ts]=sctrgvsp(t,x,y2ctrg, SimStatusFlag); % 'str' always empty

Associated global structure 'ctrg' and defining function 'dctrgvsp()'.

Functionality:
 * discrete time, state controlled EM-torque/generator speed curve (q/n-curve);
   effective delay ~ 1.5*sample time (see NOTE 2);  generator state values:
    0 <=> zero load                    (q==0)
    1 <=> partial load 'begin'         (steep linear q/n-curve)
    4 <=> partial load 'optimum lambda' (smooth quadratic q/n-curve)
    5 <=> partial load 'end'           (steep linear q/n-curve)
    6 <=> full load                    (smooth 'q*n=constant' curve)

Signal vectors:
 * y = ... (output vector)
    Teset    [Nm]: command value for EM-torque servo in VSP wind turbine
    GenState  [-]: generator state

 * x = ... (state vector)
    Teset    [Nm]
    GenState  [-]

 * y2ctrg = ... ('global' input vector)
   <output vector of sturbvsp(); see sturbvsp.m>

Function sctrgvsp() uses fields of global structure 'ctrg' as indices in signal
vectors y, x and
 * u = ... ('local' input vector)
    OmegaG  [rpm]: generator rotor speed (slow shaft equivalent)

Field creation in function dctrgvsp() (see dctrgvsp.m).

Application options (Ao):
1. S-function sctrgvsp() in equally named model block in Simulink .mdl-file, or
2. M-function sctrgvsp() called by M-function emusimulinkvsp() (Simulink emulation)

Function execution depends on value 'SimStatusFlag':
```

```
 case   0 : def. dimensions & sample time ('sys'&'Ts'); init. states ('x0') (@Ao1)
 case 100 : def. dimensions & sample time ('sys'&'Ts'); init. states ('x0') (@Ao2)
 case   3 : calc. output vector ('sys'=  'y'(k) = g('x'(k))')          (@Ao1&@Ao2)
 case   2 : update state vector ('sys'='x'(k+1) = f('x'(k),'u'(k))     (@Ao1&@Ao2)
```

MATLAB file sturbvsp.m

```
STURBVSP(): governing equations for simulation var-spd pitch to vane HAWT

CALL: [sys,x0,str,Ts]=sturbvsp(t,x,y2turb,SimStatusFlag); % 'str' always empty

Associated global structure 'turb' and defining function 'dturbvsp()'.

This hybrid S-function sturbvsp() has a time-discrete part for pitch actuator delay
and  a time-continuous part for all remaining turbine dynamics including acutator
equipment; respective state vectors are xd and xc. Seperate S-function code
partitions exist for update of xd and calculation of d xc / dt (=xc').

The time-discrete part delays the pitch speed command value with a multiple of the
non-varying discrete sample time H. This command value enters the time-continuous
part through an element of xd; WITHOUT ANY USER-CODE BUFFER, the Simulink protocol
for xd-update and xc'-calculation causes 1*H delay in the command value for the
time-continuous pitch actuator dynamics (see NOTE 2).

Time-discrete functionality
---------------------------
State controlled delay of the pitch speed command value over a multiple of the sample
time H. A basic, non-varying, delay is included for modeling very fast dynamics.
Input to the 'basic delay code partition' is the pitch speed command value from
sctrpvsp(). This command value will be extra delayed if a desired change in the
pitch speed necessates to overcoming a difference in Coulomb pitch bearing friction.
For this, two state variables define the  'Coulomb friction status', which tells if,
and in that case how, the pitch actuator:
 - has entered the dead zone caused by Coulomb Friction
 - will leave the dead zone caused by Coulomb Friction

After the 'friction zone' has been entered a 'change requirement' on the actuator's
pitching torque applies. The pitching torque reacts by time-linear growth. The
additional delay is realised by feeding zero-input into the basic delay code
until the pitching torque satisfies the 'change requirement'.

For further details see subfunction mdlUpdate().

Time-continous functionality
----------------------------
Derivative calculation for the equations of motion (EOM) of the wind turbine;
includes all actuator equipment behaviour except the pitch actuator delay.
The EOM are formulated as a set of ordinary 1st order non-linear differential
equations. The EOM agree with the following model properties:
 * rigid rotor with power and thrust coeff. data (Cq,Ct) for aerodynamic conversion
 * dynamic inflow through Cq&Ct-specific 1st order lead-lag networks on pitch-angle
 * drive-train  with 1st torsional mode; scaled operation on slow shaft speed level
 * 1st fore/aft and sideward bending mode in tubular tower
 * normalised 2nd order system for EM-torque servo behaviour
 * normalised 2nd order system for pitch speed servo behaviour (delay excluded)
```

```
For further details see subfunction mdlDerivsOrOutputs().

Signal vectors
--------------
 * y = ... (output vector)
    VwEff              [m/s]: rotor effective wind spd incl. turbul. & tower shadow
    Vw0p               [m/s]: rotor uniform wind speed ('0p-mode')
    dThdtset          [dg/s]: command value for pitch speed servo in VSP wind turbine
    Teset               [Nm]: command value for EM-torque servo in VSP wind turbine
    dThdtsetDelay     [dg/s]: pitch comm. after friction dead zone & basic delay
    OmegaR           [rad/s]: turbine rotor speed
    Azim               [rad]: rotor azimuth angle
    Gamma              [rad]: drive-train distortion (slow shaft equivalent)
    xnod                 [m]: tower nodding displacement
    xnay                 [m]: tower naying displacement
    dThdt             [dg/s]: pitch angle adjustment speed
    Th                  [dg]: pitch angle  (xc)
    OmegaG           [rad/s]: generator speed (slow shaft equivalent)
    Pa                  [MW]: aerodynamic power captured by rotor
    Pe                  [MW]: electric power delivered by generator to grid
    Fa                  [kN]: aerodynamic thrust force on torot

 * x = ... (state vector; stacks cont. time (xc) and discr. time (xd): x=[xc;xd])
    OmegaR           [rad/s]: turbine rotor speed (xc)
    Azim               [rad]: rotor azimuth angle (xc)
    dGammadt         [rad/s]: drive-train distortion speed (slow shaft equiv) (xc)
    Gamma              [rad]: drive-train distortion (slow shaft equiv) (xc)
    dxnoddt            [m/s]: tower nodding displacement (fore/aft) speed (xc)
    xnod                 [m]: tower nodding displacement (xc)
    dxnaydt            [m/s]: tower naying displacement (sideward) speed (xc)
    xnay                 [m]: tower naying displacement (xc)
    dsqThdtsq      [dg/s**2]: pitch angle adjustment acceleration  (xc)
    dThdt             [dg/s]: pitch angle adjustment speed  (xc)
    Th                  [dg]: pitch angle  (xc)
    dTedt             [Nm/s]: EM-torque adjustement speed (xc)
    Te                  [Nm]: EM-torque  (xc)
    xThCqDI             [dg]: state in dynamic inflow simulation for aero torque (xc)
    xThCtDI             [dg]: state in dynamic inflow simulation for axial force (xc)
    dThdtsetOld       [dg/s]: pitch spd comm. value in prev. discr. time cycle (xd)
    CoulombZoneExit      [-]: exit status of pitch actuator's friction dead zone (xd)
    CoulombZoneEntry     [-]: entry status of pitch act.'s friction dead zone (xd)
    TqDelAct            [Nm]: change in pitching torque; pursues dead zone exit (xd)
    dThdtsetDelayLoad [dg/s]: delayed pitch comm. by act.'s friction dead zone (xd)
    xBasicDelayMinus1 [dg/s]: shift register for simulation of basic pitch delay (xd)
    dThdtsetDelay     [dg/s]: pitch comm. after friction dead zone & basic delay(xd)

 * y2turb = ... ('global' input vector)
    <stacked output vectors of swindvsp(), sctrpvsp(), sctrgvsp(); see s...vsp.m>

Function sturbvsp() uses fields of global structure 'turb' as indices in signal
vectors y, x and
 * u = ... ('local' input vector)
    VwEff     [m/s]: rotor effective wind speed incl. turbulence & tower shadow
    Vw0p      [m/s]: rotor uniform wind speed ('0p-mode')
    dThdtset [dg/s]: command value for pitch speed servo in VSP wind turbine
    Teset      [Nm]: command value for EM-torque servo in VSP wind turbine

Field creation in function dturbvsp() (see dturbvsp.m).
```

```
Application options (Ao):
1. S-function sturbvsp() in equally named model block in Simulink .mdl-file, or
2. M-function sturbvsp() called by M-function emusimulinkvsp() (Simulink emulation)

Function execution depends on value 'SimStatusFlag' ('y' output vector, 'u' input
vector, 'xc' continuous state vector; 'xd' = discrete state vector):
 case   0:def. dimensions & sample time ('sys'&'H'); init. ('xd0','xc0')      (@Ao1)
 case 100:def. dimensions & sample time ('sys'&'H'); init. ('xd0','xc0')      (@Ao2)
 case   3:calculate  'y'     ('sys'=y(k+e)     = g(xc(k+e),u(k+e),xd(k)) (@Ao1&@Ao2)
 case   1:differentiate 'xc' ('sys'=dxc/dt(k+e)=fc(xc(k+e),u(k+e),xd(k)) (@Ao1&@Ao2)
 case   2:update 'xd'        ('sys'=xd(k+1)    =fd(xd(k),u(k))           (@Ao1&@Ao2)
```

5.3.4 Postprocessing simulation results

Use support M-function 'ecnylist()' for listing output vector data that have been acquired in functions a<SGS>...(). Function 'ecnylists()' also prepares plotting because the output structure has fields that are arrays with time- and 'y'-data, supplemented with 'y'-element names and dimensions.

Use function 'ecndatalist()' for listing of data items as used in the simulation.

The listings obtained from 'ecnylist()' and 'ecndatalist()' can be included straightforward in a report (use 'verbatim' option).

The next two paragraphs list the function headers of these supporting M-functions for postprocessing.

MATLAB file ecnylist.m

```
ECNYLIST(): prepare plotting & make list of acquired 'y-data' of main S-function

CALL: sout=ecnylist(SGSname,PtsId,SigId,linelength,fid,leadsp, OutOption);

LOG
 June 25, 2002: creation date

TEST & EXAMPLE
 >> load([ecnmfiledir('ecnsimvarbases'),'\TEST\strucvsp020607'])
 >> yout = ecnylist('turb',7,[],80,1,2,1);
 >> yout = ecnylist('turb',100,[2,4,5,7,8,10,12],[],[],[],[]);
 >> yout = ecnylist('turb',100,'VwEff OmegaR Gamma xnod xnay Th Pae');

INPUT
 SGSname : name of S-function specific global structure in which 'y-data' are stored
 PtsId  [<all>]: No. of time points that is listed / prepared for plotting
 SigId  [<all>]: Signal selector, being:
                 * character string with output element / subvector names, or
                 * integer array with index numbers in y-vector
                 * integer scalar < 0, impying selection of y(1) ... y(-<SigId>)
 linelength [80]: length of lines [no chars] in output value list
 fid        [1]: file identifier for messages; 1->screen; otherwise from fopen()
 leadsp     [0]: no. of leading spaces before each line
 OutOption  [0]: option for formatted output page to 'sout' (0/1: NO/YES)

    LEADING AND TRAILING SPACES IN STRING-INPUTS ARE REMOVED BEFORE PROCESSING
```

```
OUTPUT
 sout : output structure with following fields:
         tv : 1D-array (column) with time points
         tn : cell string array with 1-element: 'time'
         td : cell string array with 1-element: '[s]'
         yv : 2D-array with y-data as columns
         yn : cell string array with y-element names
         yd : cell string array with y-element dimension
         [page: formatted output list; only if OutOption == 1; much slower!!]

GLOBAL
 global mainvarnames physdims varqualities <SGSname>

ECN M-FUNCTIONS
 ecnintincellel()
 ecncvstr2ca()
 ecnrmspaces()
 ecnfunctrace()

DESCRIPTION
 Processes acquired y-vector data in 2D-array '<SGSname>.y.a'. It makes
 output structure 'sout' with time and y-element value-arrays, names and
 dimensions. Optionally, a formatted output page with time- and y-values
 is added as field to 'sout'. The 'listing option' dramatically raises the
 data processing time for this function 'ecnylist'.
```

MATLAB file ecndatalist.m

```
ECNDATALIST(): list data items in design or simulation specific global structure (GS)

CALL sout=ecndatalist(GSname,dataidf,dataname,linelength,fid,idx1,idx2,leadsp);

LOG
 JUNE 14, 2002: creation

TEST & EXAMPLES
 M-program testecndatafun.m with ASCII-results in testecndatafun.txt

INPUT
 GSname    : name of global structure for main variable that is used in the
      S-function to which the concerned data item is related (string var)
 dataidf        : identifier of data substructure to be showed (string var 'p', 'r','b')
 dataname  ['*']: name of data-item to be showed (string var),
  '<name>'      : list data item with name <name>; <name> = 'num', 'struc' or 'struc.num'
  '<n1>,<n2>'   : list data items <n1> and <n2>, etc; separate by comma or space
  '*' -> list all data items in data group
  '* \ <n1>,<n2>' -> list all data items in data group except those specified
                     by <n1>, <n2> etc.: e.g.
                          '* \ a, c, b.T' excludes  items a, c and b.T from list
                  note: empty input ([]) for dataname equals '*'!
 linelength [80]: length of lines [no chars] in output page sout
 fid         [1]: file identifier for messages; 1->screen; otherwise from fopen()
 idx1   [<all>]: list with row-indices (in case of matrix)
 idx2   [<all>]: list with column-indices (in case of matrix)
      . in case vector: list idx1 is used for selection of elements unless
```

```
      . in case of row vector and both idx2 and idx1 are specified.
      . only effect if 'dataname' is ONE SINGLE 'num' or 'struc.num'

 leadsp     [0]:  no. of leading spaces before each line
     AT WILDCARD '*' FOR DATANAME: ALL ARRAY-ELEMENTS ARE PRINTED
     LEADING AND TRAILING SPACES IN STRING-INPUTS ARE REMOVED BEFORE PROCESSING

OUTPUT
 sout  : string with formatted output list == '<fail ecndatalist()>' at invalid input

GLOBAL
 mainvarnames, varqualities, dsgnvarnames, dsgnvarqualites, <SGSname>

USES MATLAB-FUNCTIONS (user supplied)
 ecnrmspaces()
 ecncvstr2ca()
 ecnstrcmp()
 ecncomplist()
 ecnisfieldgen()
 ecncsa2str()
 ecnreportnum() [in this function file included]

DESCRIPTION
 Prints to output page sout the value(s) of data item(s) <dataname> in data
 group <dataidf> of  a design specific global structure (DGS) or S-function
 specific or run governing global structure (VGS). The global structure
 name 'GSname' is matched to the fields in globals 'dsgnvarnames' and
 'mainvarnames'. It is now known if data are listed in the control design stage
 or in the simulation stage. Allowed data groups in the design stage are retrieved
 from global 'dsgnvarqualities'; in the simulation stage from 'varqualities.

 Continues with a check on data group validity for 'dataidf'.
 Then a list is composed  for the data items to be printed out in report format.
 These data items are printed by use of the 'in function file function'
 ecnreportnum(). Deviation from the real array sizes  in the print-out is only
 possible if a single, non-structure, data item is printed out. See INPUT for
 detailed input item descriptions.
```

6. SUPPORTING M-CODE

MATLAB-code modules have been created for minimising the development time of programs like ‘parvsp’ and ‘simvsp’ that set up an actual control development environment (§4 and §5). These supporting M-code modules are *functions* and *scripts* (M-files).

§6.1 lists the kinds of developed functions and scripts and describes the included documentation in the M-files. §6.2 lists the first comment line of all developed support functions and scripts.

6.1 Classes of support M-code modules and documentation

The developed M-functions and M-scripts are created as so called M-files (extension ‘.m’). The developed M-files are subdivided into sets for:
- data and signal handling;
- subsystem modelling;
- control loop synthesis;
- general support;
- MATLAB startup support.

An M-function basicly transforms function input into function output as identified by the function header; input and output usually are MATLAB workspace variables. In addition it may handle **global** variables *while these do not appear in the function header*, and handle **files** with data or plots. Function M-files always contain so called ‘help information’ that is ordered in accordance with the next labels:
- CALL : function header for how to call the M-function;
- LOG : creation date and updates;
- TEST & EXAMPLE : applied test and example for getting acquainted;
- INPUT : description of input (I) variables passed via function header;
- OUTPUT : description of output (O) variables passed via function header;
- GLOBALS : list of I/O variables that are global in the MATLAB workspace;
- ECN M-FILES : list of used M-files that are not ‘MATLAB supplied’;
- DESCRIPTION : description of implemented functionality.

An M-script is just a block with M-code lines. It can be included in M-functions for generating the same code lines in a number of functions; this is done if it is problematic to implement the functionality via a (sub) M-function. An M-script can also set up ‘by its own’ an executable MATLAB-program or can be included in an other script; the latter for the same reason as in M-functions. Script M-files contain similar ‘help information’ as function M-files.

6.2 Developed support M-functions and M-scripts

The first comment line in an M-file always gives a short functional description, starting with the M-file name (without ‘.m’). If it concerns a function, the M-file name is extended with braces ‘()’. The next subsections list the first line of the developed M-files in the distinghuised sets.

6.2.1 M-files for data and signal handling

```
ECNDATADEF(): create data item in design or simulation specific global structure (GS)

ECNDATADEL(): delete data item from design or simulation specific global structure (GS)

ECNDATALIST(): list data items in design or simulation specific global structure (GS)

ECNDEFCHECK(): check if data and signal items on S-function files exist in SGS's

ECNDEFSUPP(): check created signal items & creates data items  from 'design' MAT-file

ECNDGSFIELDS2ITEMS: put fields of design spec. global structures (DGS's) in workspace

ECNDGSFVALUES4CHECK: check item-redefine in design specific global structures (DGS)

ECNDGSGLOBAL: make design specific glob. structs (DGS's) global to M-function or M-shell

ECNDIMCON4DATA(): define physical dimensions & conversion factors via global 'physdims'

ECNDSGNSTORE(): store data items of design specific global struct. (DGS) on MAT-file

ECNDSGNVARBASES(): prepare definition of control design specific global structures (DGS)

ECNFILESCAN(): scan 'substr up to separator' before/after 'tokenset' occurs on M-file

ECNLSFIELDS2ITEMS: put fields of new structures relative to 'localsList' in workspace

ECNMAPCLEAR(): delete all index-pairs that enable signal mappings from S-function structures

ECNMAPLIST(): list mappings 'to' and 'from' signal vector elements of S-functions

ECNMNEMODIM(): return mnemonic belonging to dimension in acc with global 'physdims.ex'

ECNPHYSDIMS(): define mnemonics, physical dimension strings and conversion to SI factors

ECNRTSUPP(): write RGS/SGS r-items in all equal-name SGS r-its/SGS-spec eq-nm p,b-its

ECNFUNCSCAN(): scan S-function file on used data and signal items of global structure

ECNSFUNCSCANLEV1(): scan all S-func files on all 1st level fields of SGS and SGS.m

ECNSIGATTR(): return string for signal vector element with name, dim [, subvec el.nr]

ECNSIGDEF(): create signal item in S-function specific global structure (SGS)

ECNSIGLIST(): list signal items in S-function specific global structure (SGS)

ECNSIGMAP(): create index-array pair for 'equal-name' element(set)s in 2 signal vectors

ECNSIMCLEAR(): clean up data/signal groups in simulation specific global structs (VGS)

ECNSIMVARBASES(): prepare definition of simulation specific global structures (SGS,RGS)

ECNVAR2STR : makes list '<xxxitemHead>report' with data items by 'xxxitemNames'

ECNVGSGLOBAL: give M-func or M-shell access to simulation spec. global struct (VGS's)
```

ECNVWFILESELECT(): search best matching wind speed MAT-file to specified mean wind speed

ECNYLIST(): prepare plotting & make list of acquired 'y-data' of main S-function

6.2.2 M-files for modelling

ECNAEROSENS(): calculate aerodyn. sensitiv. functions and schedule-parameters for control

ECNDYNINFLO4P2V: derive dyn. inflow pars for 'pitch2vane-HAWT-sim' with rotor coeff. data

ECNFUISENLN(): calculate sensitivities for d/dt(axial induction speed) in rotor annulus

ECNGENERATOR4VS: derive generator curve data for a variable speed wind turbine

ECNINDLNAER(): solve axial and tangential induction equ. for linear profile aerodynamics

ECNMKCPCQCT: convert rotor coefficient input decks into DAT-& MAT-files and workspace variables

ECNSTEPTRAN(): calculate stepresponse for transfer funtion data or tf-expression

ECNTIMEGEN(): create time series from 1-side auto spectrum by inverse Fourier transform

ECNVWEFFSHR(): create normalised longitudinal wind speed variations due to wind shear

ECNVWEFFTOW(): creates normalised long/vert/lat wind speed variations due to tower shadow

ECNVWROTEFF(): create rotor effective wind speed MAT-file, intermediate data and plots

6.2.3 M-files for controller synthesis

ECNSCDSGSPD(): calculate pairs of PD-gains for control of double integr. with time delay

6.2.4 General support M-files

ECNADD2VEC(): extends vector 'in1' with elements of vector 'in2'; 'in1' keeps orientation

ECNADDFILES(): compose axis-label from 'true label', plotfile name, date and making function

ECNBOUND(): bound a numeric 1D-array between a maximum and minimum value

ECNCELL2ITEMS: create data items for input cell arrays or get defaults

ECNCOMPITEM(): compare item-values for strings&numerics; item-types for structs&cells

ECNCOMPLIST(): compare item-values for 1D-numeric arrays or 1D-cell arrays with strings

ECNCSA2STR(): convert cell array with string variables to output character string

ECNCSA2STR1: convert cell string array to '1-line-str' with '1-space separated' items

ECNCVSTR2CA(): convert 'multi-item' string to character array and to 1D-cell array

ECNDELFIGS: delete figure boxes

```
ECNENVLCRV2(): calc (x,y)-pairs of enveloping curve for encl. surf. by input (x,y)-pairs

ECNFELDMAP(): search for occurrence orders of equal-name fields in 2 input structures

ECNFILE2STR(): extend char string with TXT-file contents OR prepares this extension

ECNFUNC2STR(): envelope a character string with 'function name header' and '... tail'

ECNFUNCTRACE() : return character string with scope on nested MATLAB calls

ECNFZERO(): search for 'x' in 'f(x) == 0' old function; equals using 'inline()' in fzero()

ECNINTINCELLEL(): return index numbers of element-values in cell array

ECNISFIELDGEN(): return ranking no[-pair]'s of field[subfield pair]'s in a structure

ECNISTYPE(): return '<empty>','numeric', 'char', 'cell', 'struct' or 'unknown' as item type

ECNLASTFELD(): return contents 'con' and name 'str' of last field in structure

ECNMYLINE(): return linetype for plotline dependent on integer input

ECNLISTUM(): copy unique elements from 2 cell-arrays with strings into output cell-array

ECNMFILEDIR(): return directory & name of calling M-function OR dir of specified M-function

ECNMHEAD2FILE(): write headers of M-files to TEX-format output character string

ECNMIRROR():  reverse sign and index series of y: x(1:N)=-y(N:-1:1)

ECNMODULO(): return modulo-function of 'in' for given 'range'

ECNMTIMES(): exist as 'MTIMES()' in '@tf control-subdir': ~matlabr11\toolbox\control\@tf

ECNPAUSE: M-code edition of M-function ecnpausefun(); do help on ecnpausefun()

ECNPAUSEFUN(): make 'pause' if input 'pauseon' equals 1 or is omitted, otherwise no 'pause'

ECNPHSDG(): retrieve phase shift of transfer function data in degrees

ECNPRINTDATA(): return print-out of numeric, string or up-to-2-layer cell or struct

ECNRANGEBND(): make vector from lower limit to upper limit with given stepsize

ECNRMSPACES(): remove spaces, newlines, carriage returns and ascii-0's from string

ECNSTR2COLCA : get column cell array  with input string 'str' in all 'n' elements

ECNSTRCENTER(): extend char string with blanks or enveloping tokens of longer string

ECNSTRCMP(): compare two (cell arrays with) character strings; returns 0 at empty-CELL

ECNSTRPARSE(): parse & process 'include' and 'exclude' items from input character string

ECNSTRSUBRP(): replace all equal substrings in character string by another substring

ECNSTRUCSKEL(): create 1-layer structure with empty fields (skeleton structure)
```

ECNTIMESTAMP(): makes character string with date&time identification

ECNUNIQUE() : select unique elements from numeric or cell string array; no sort

ECNVC2COL(): transpose row vector to column (leave matrix and column vector unchanged)

ECNVC2ROW(): transpose row vector to rowumn (leave matrix and rowumn vector unchanged)

ECNWHOMAT(): list names of 'MAT-file variables' includ. struct fields OR expand structure

6.2.5 Start-up support M-fi les

ECNCELL2STR(): convert cell string array to '1-line-str' with '1-space separated' items

ECNPARSEPROJDIR(): parse directories in file system [and extend MATLAB search path]

ECNSTARTUP0: user definable character strings for startup

ECNSTARTUPEXE(): startup settings for application development on in-house MATLAB tools

ECNSTR2CELL(): parse items from input character string and put in cell string array

7. CONCLUSION

The described turbine specific MATLAB programs ‘parvsp’ and ‘simvsp’, for controller parametrisation and evaluation, serve as ‘a frame of reference’ for the creation of a control development environment for any wind turbine within the selected concepts. The creation of such an environment is strongly accelerated by use of the developed supporting MATLAB code.

The selected concepts are HAWTs that are characterised by variable speed operation with pitch to vane control or by constant speed operation with pitch to stall control.

The supporting MATLAB code consist of functions and scripts for data and signal handling, modelling and control synthesis. It also includes dedicated startup code for file system creation and version management at control design support in industrial applications.

Summarized
The turbine specific programs ‘parvsp’ and ‘simvsp’ and the support functions and scripts establish a good point of departure for the further execution of this project as well as for industrial control applications.

	Date: December 2003	**Report No.:** ECN-C-03-141
Title	ECN Design Tool for Control Development; Revised Points of Departure	
Author	T.G. van Engelen, E.J. Wiggelinkhuizen	
Principal(s)	NOVEM, Dutch Ministry of Economic Affairs	
ECN project number **Principal's order number**	7.5153 2020-01-12-10-003	
Programmes	Energieprogramma DEN, ARB	

Abstract
An open source software environment for the design of wind turbine control algorithms has been created. The modularisation enables to modify and to add or remove functionality in a user friendly way. This software environment established the point of departue for the development of specific control algorithms within the project. It consistst of so called functions and scripts in the MATLAB programming language. These MATLAB modules support modelling, control synthesis and data and signal handling. They thus enable a user to implement a modular parametrisation procedure for the control algorithm and a Simulink compatible evaluation of the designed controller. The software has been made typical for the variable speed windturbines with pitch control. However, other wind turbine concepts can also be dealt with.

Keywords
windturbine, control, software, open source, offshore, stability

Authorization	**Name**	**Signature**	**Date**
Checked			
Approved			
Authorised			