FDC 1.2 – A Simulink Toolbox for Flight Dynamics and Control Analysis

Marc Rauw

2nd edition, May 10, 2001

This edition of the FDC report is distributed exclusively across the Internet. It can be downloaded in PostScript and PDF format via the homepage of the FDC toolbox: *http://www.dutchroll.com*. Of course, the toolbox itself can be downloaded there too. Currently, there are two FDC versions: FDC 1.2 for Matlab 4.2/Simulink 1.3, and FDC 1.3 for Matlab 5.1/Simulink 2.1 or newer. This report describes version 1.2 only; there is no dedicated manual for version 1.3. However, since the differences between versions 1.2 and 1.3 are mainly limited to Matlab-compatibility, on-line help-functions, and the user-interface of some load programs, this report can still be used as reference guide for FDC 1.3. See WHATSNEW.TXT in the DOC directory of FDC 1.3 for more information about the differences between the two versions.

If you wish to make a hard-copy of the report, it is advised to download the PostScript version and send this to a PostScript printer or a software PostScript interpreter such as GHOSTSCRIPT (see *http://www.ghostscript.com* for more information).

This document was originally created with emTEX, a shareware version of LATEX for MS-DOS computers, using the old LATEX 2.09 format. The second edition was prepared with MIKTEX in LATEX 2ε format. The vector figures were created with TEXCAD (part of the emTEX package), VISIO, MATLAB/SIMULINK, and MAYURA DRAW. The bitmap graphics were created with IRFANVIEW, PAINT SHOP PRO, and the wonderful JPEG2PS tool. The Postscript graphics were prepared with the DVI-driver DVIPS and the invaluable GHOSTSCRIPT/GSVIEW.

Instead of a list: All trademarks mentioned in this document are registered to whoever it is that owns them.

Contact information:

e-mail: rauw@worldonline.nl
homepage: http://www.dutchroll.com.

Preface

Welcome to the FDC toolbox!

This report gives an overview of the Flight Dynamics and Control toolbox FDC 1.2, a graphical software environment for the design and analysis of aircraft dynamics and control systems, based upon MATLAB and SIMULINK. Its main goal is to simplify the Flight Control System design process, although it can be applied to a broad range of Stability and Control related problems. An early version of the toolbox was used in practice for an autopilot design process for the De Havilland 'Beaver' aircraft, which was performed by graduate students of the section Stability and Control of the Faculty of Aerospace Engineering (including myself) in 1992/1993. This project provided an impressive demonstration of the flexibility and power of SIMULINK-based tools by enabling us to bring the autopilot project from the early designs in MATLAB to the actual flight test phase in a short period of time with great results (see refs.[22] and [29]). Based upon this 'Beaver' autopilot experience the current version of the FDC toolbox has evolved into an advanced 'proof of concept' package which has matured a lot since the early versions from 1992/1993. FDC 1.2 thus paves the way for future general-purpose toolboxes in the field of Flight Dynamics and Control research.

FDC 1.2 can play an important role in aeronautical education by assisting in Flight Dynamics and Control related training courses. There it can take full advantage of its clear model-structure and its flexibility which enables it to be used for a variety of tasks, e.g. control system design, off-line simulations of the open-loop, uncontrolled aircraft (either non-linear or linear), closed-loop simulations of an automatically controlled aircraft, etc. The non-linear dynamic model of the 'Beaver' aircraft is very suitable for a first introduction to the structure of aircraft dynamics in general due to its relatively simple structure which nevertheless contains many typical aircraft characteristics that make it an ideal basis for a general treatment of aircraft dynamics. For instance, the model clearly exhibits cross-coupling of longitudinal and lateral motions and it describes asymmetrical aircraft behavior due to propeller-slipstream effects. FDC 1.2 can help assisting courses on Automatic Flight Control Systems (AFCS's) if it is used in combination with the various existing control system toolboxes for MATLAB. Its powerful non-linear simulation capabilities make it easy to extend such courses beyond the linear AFCS design and simulations, thus bridging the gap between the control theory and its applications in practice.

About the second edition of this report

Since the first edition in 1997, the Internet has become the standard distribution channel for the FDC toolbox. In addition to the software itself, the PostScript source files for the FDC 1.2 report were also made available over the Internet. This new edition was created primarily to shorten the download-time for the PostScript sources and to make it possible to create a PDF version of the report. Contrary to the previous version, the new sources are resolution-independent, which will enhance the printing quality when applying a high-resolution output device.

The text of the report has not been changed, apart from one section in the install instructions. However, some parts are slightly outdated – in particular the system requirements section. Contrary to the text, many figures have been updated: some bitmap graphics have been replaced by vector graphics, which enhances their quality while reducing the size of the corresponding PostScript files, and other bitmap graphics have been implemented as gray-scale pictures instead of dithered black-and-white pictures. But since the contents are virtually the same, there's no need to re-print this report if you already own a hard-copy of the old version.

One important remark about the software requirements: *FDC 1.2 is not compatible with Matlab 5/Simulink 2 or newer!* To solve this problem, version 1.3 of the FDC toolbox was created for Matlab 5.1/Simulink 2.1 (this version has also been tested successfully with Matlab 5.3/Simulink 3.0). Some other improvements were made as well: the on-line help system was improved and some small changes were made in the user-interface. However, these changes were not big enough to justify a new user-manual. If you use FDC 1.3, the document you're currently reading is still the best reference source available. However, make sure to read the README and WHATSNEW-files for FDC 1.3 too; they contain more information about the differences between versions 1.2 and 1.3.

The structure of this report

Creating a report which treats the complete FDC toolbox from the different points-of-view of all types of users was not easy. The 'proof of concept' character of the FDC toolbox required both a thorough treatment of the mathematical structure of the models and algorithms and a detailed description about the actual implementation of the models and tools in MATLAB/SIMULINK. Moreover, the report should show how to get started as an 'FDC novice', while a detailed description about the 'Beaver' autopilot was deemed necessary to demonstrate the more advanced applications of the FDC toolbox for experienced users. For this reason, it was first planned to divide the report into three separate parts: theoretical backgrounds, reference guide, and tutorial/examples. However, one of the virtues of this toolbox is the fact that the practical implementation of the models and tools makes it easier to comprehend the theory and vice versa. In other words, a formal separation of theory and practice was not that trivial. For this reason it was decided to combine the different elements into one report. So whether you simply want to apply the existing models incorporated in FDC 1.2 for a demonstration of Stability and Control related problems or use the models and tools as a guideline for the implementation of your own models: all necessary information can be found in this report.

This did, however, require some compromises with regard to the structure of the report. It would have been somewhat easier to trace the required information when using separate volumes for the theory, implementation, and examples, but this has been compensated by means of a clear table of contents, chapter headers on top of the pages, and a complete index to simplify searching. The general structure of the report is as follows:

- chapter 1 shows how to install the software package and how to get started with FDC 1.2,
- chapter 2 gives an introduction to the Flight Control System design process, one of the main applications of the FDC package,
- chapter 3 treats the theoretical aspects of the different dynamic models from the FDC toolbox,
- chapter 4 treats the theoretical aspects of some important analytical tools contained in or used by FDC 1.2,
- chapters 5 to 8 provide the link between the theory from chapters 3 and 4 and the FDC

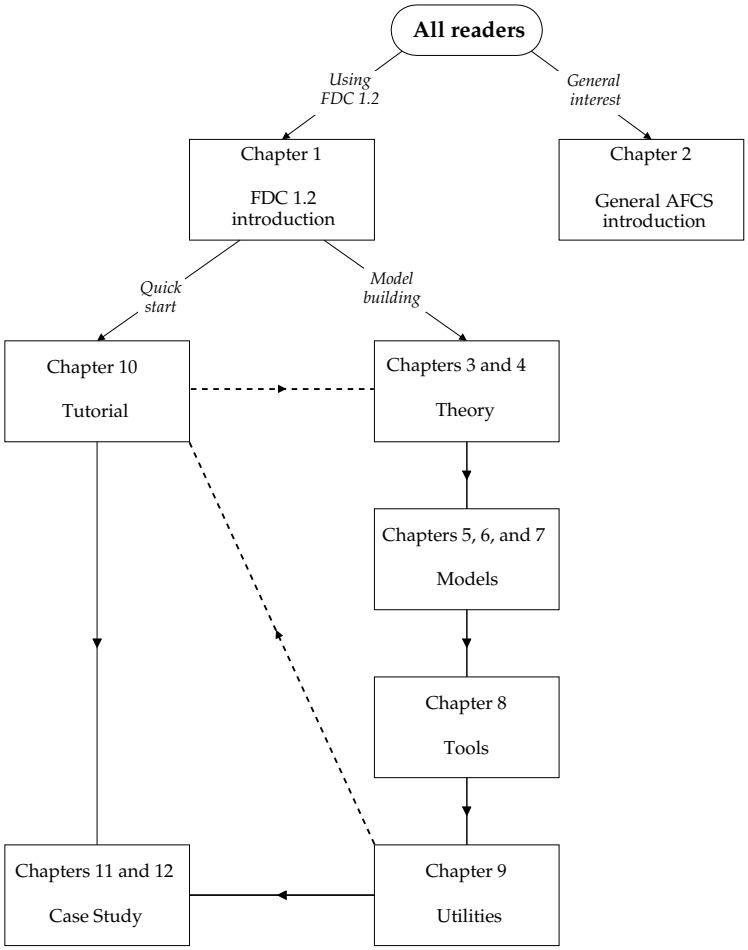


Figure 1: Different ways to approach this report

software by describing the SIMULINK models and MATLAB programs from FDC 1.2 systematically in alphabetical order (these chapters fulfill the function of *reference guide* in this report),

- chapter 9 describes some utilities from the FDC toolbox which are needed for defining or loading model parameters, steady-state initial conditions, system matrices of linearized models, etc. and for post-processing simulation results,
- chapter 10 demonstrates the practical use of the models and tools for the analysis of open-loop aircraft responses,
- chapters 11 and 12 describe a detailed case-study of the 'Beaver' autopilot,
- chapter 13 contains recommendations for further improvements of the FDC toolbox,
- appendix A contain a list of symbols and definitions,
- appendix B gives background information for the derivations of the equations of motion,
- the other appendices contain parameter definitions for the models from chapter 3, the SIMULINK systems from chapters 5 to 7, and the MATLAB tools from chapter 8.

How to read this report

Figure 1 gives a schematic overview of the different ways to use this report. All users should start reading chapter 1 for a short introduction to the package and the install instructions. If you want to get started rightaway to get a general idea about the possibilities of the package you should continue with chapter 10. Users who want to know more about the theoretical backgrounds should read chapters 3 and 4, while the function of a *reference guide* to the MATLAB/SIMULINK implementation is fulfilled by chapters 5 to 6, 8, and 9. Experienced users are encouraged to read chapters 11 and 12 for a detailed case-study of the 'Beaver' autopilot, which clearly demonstrates the enormous flexibility of the SIMULINK environment for solving such control system design problems. However, even though the structure of the autopilot simulation model has been brought to the same standard as the other FDC systems, it is recommended to get familiar with the toolbox first *before* proceeding with this case-study. Users who don't comprehend the open-loop simulations from chapter 10 will most likely have problems with the autopilot simulation models as well. The autopilot simulation model from chapter 12 contains the *complete* set of control laws used for the actual flight tests, so it takes into account many problems encountered in practice. If you plan to design your own autopilot control laws, it is very useful to study chapters 11 and 12, because they contain useful clues for easing out the transition from simulation model to flight test. Chapter 2 provides useful general background information for all readers who are interested in Automatic Flight Control Systems design processes. Finally, chapter 13 highlights a number of further improvements of the FDC toolbox under consideration for future releases.

Typographical conventions

The following typographical conventions are used in this report:

- scalar variables and functions, as well as variable names from software source codes and graphical SIMULINK systems, are typeset in *italics*,
- vectors and vector functions are typeset in **boldface**,
- matrices and frames of reference are denoted with *ITALIC CAPITALS*,
- names of software packages, filenames, and directory names are typeset in SMALL CAPITALS,
- Names of SIMULINK systems are typeset in Sans Serif style, MATLAB functions are either denoted by their filenames (small capitals with extension .M) or in SANS SERIF CAPITALS.
- commands to be typed by the user at the DOS prompt or the MATLAB command-line are typeset in `typewriter` style.

Contact information

The homepage for the FDC toolbox is located at: `http://www.dutchroll.com`. It contains more information about the new version FDC 1.3 and future updates of the FDC toolbox. If you have any questions or comments, please contact the author via e-mail at: `rauw@dutchroll.com`.

Marc Rauw, May 10, 2001

Contents

Chapter 1

Introduction to FDC 1.2

1.1 What is the FDC toolbox?

FDC is an abbreviation of *Flight Dynamics and Control*. The FDC toolbox for MATLAB and SIMULINK makes it possible to analyze aircraft dynamics and flight control systems within one software environment on one PC or workstation. The toolbox has been set up around a general non-linear aircraft model which has been constructed in a modular way in order to provide maximal flexibility to the user. The model can be accessed by means of the graphical user-interface of SIMULINK. Other elements from the toolbox are analytical MATLAB routines for extracting steady-state flight-conditions and determining linearized models around user-specified operating points, SIMULINK models of external atmospheric disturbances that affect the motions of the aircraft, radio-navigation models, models of the 'Beaver' autopilot, and several help-utilities which simplify the handling of the systems.

The package can be applied to a broad range of stability and control related problems by applying MATLAB tools from other toolboxes to the systems from FDC 1.2. The FDC toolbox is particularly useful for the design and analysis of Automatic Flight Control Systems (AFCS's). By giving the designer access to all models and tools required for AFCS design and analysis within one graphical Computer Assisted Control System Design (CACSD) environment the AFCS development cycle can be reduced considerably, as will be shown in chapter 2. The current version 1.2 of the FDC toolbox is an advanced 'proof of concept' package which effectively demonstrates the general ideas behind the application of CACSD tools with a graphical user-interface to the AFCS design process. Currently, the aircraft model has been worked out in detail for the De Havilland DHC-2 'Beaver' aircraft, but due to the modular structure of the models and the flexibility of MATLAB and SIMULINK it will be relatively easy to implement other aircraft models within the same structure and to enhance or refine the models if required.

1.2 Required Hardware and Software

Version 1.2 of the FDC-toolbox was developed for MATLAB 4.0 / SIMULINK 1.2c for MS WINDOWS 3.1.[1] It is necessary to have these or newer versions of these programs installed on your system. MATLAB FOR WINDOWS version 4.0 requires a PC with an 80386 or better processor, equipped with at least 4 Mbytes of RAM. In practice, an i486 computer with 4 Mbytes of RAM, running at a clock-speed of 33 MHz proved to be *just* powerful enough for our AFCS design tasks, but more computing power and more system memory is strongly recommended, especially if you plan to do simulations with stochastic or highly discontinuous input signals.

[1] For Matlab 5.1 / Simulink 2.1 or newer, use FDC 1.3 instead; see the remarks in the Preface.

For real number-crunching tasks, the toolbox should be loaded onto a real powerful personal computer or workstation (the FDC toolboxes for MATLAB 4.0 / SIMULINK 1.2c (or higher) should work well for corresponding workstation versions of MATLAB and SIMULINK, although this has not been tested). The toolbox itself requires approximately 2 Mbytes of free space on your harddisk, but it is recommended to reserve more space for storing simulation results and your own model enhancements on your harddisk.

1.3 License Agreement

Before installing the FDC package to your computer, it is necessary to read this section. *Using FDC 1.2 implies that you agree with the rules from this license.* The basic philosophy behind these rules is that the FDC toolbox should be a system which can be trusted by all types of users to fulfill their needs, while at the same time providing a maximum flexibility for the users to experiment with the FDC files. It is not possible to ensure compatibility between different SIMULINK (sub-) systems and MATLAB programs from the FDC toolbox if different versions of the systems and tools are distributed under the same names. For instance, altering the definitions of the input and output vectors within the existing aircraft model Beaver will render the current systems that call Beaver as a subsystem useless, unless those systems are changed accordingly. This may not be a problem for an individual user who can easily keep track of all changes made to the programs and systems, but if the models are shared with others – which is the ultimate goal of the modular FDC structure – unexpected errors will arise.

For this reason, the license rules only allow distribution of the complete, original distribution files of the FDC toolbox. The user is free to release custom-made versions and add-ons to the FDC toolbox, under the condition that these changes and extensions are clearly marked as being different from the original FDC toolbox. It is *not* allowed to include such extensions to the original FDC distribution diskettes or copies thereof. It is allowed to distribute enhanced FDC tools of models, as long as they are saved under different filenames. These rules are not intended to discourage the user in experimenting with the programs and SIMULINK models from the FDC toolbox. It is allowed to use the existing files or separate elements from these files as 'templates' for own developments, as long as the original source is mentioned in the new version and a new filename is used for distribution.

The following license rules should be taken into account:

- All software written by M.O. Rauw which is part of the FDC toolbox may be used without restrictions.
- The software written by M.O. Rauw which is part of the FDC toolbox may only be distributed in *unchanged* and *complete* form, including all the files listed in the file CONTENTS.TXT, and only if this is done *without* charge. If you want to distribute only parts of the FDC toolbox or charge a fee for handling, etc., you have to contact the author. In particular, commercial distribution on diskette or CD-ROM is prohibited without explicit permission.
- Any custom-made extensions to the FDC package should be distributed separately, using filenames that differ from the original FDC package. Inclusion of extensions to the original FDC distribution diskette is prohibited without explicit permission from the author.
- Elements from the FDC tools or systems may be included in other programs and systems, provided the original source (e.g. 'FDC 1.2 by M.O. Rauw') is referred to in the resulting source codes. Distribution of customized versions of the systems and tools from the FDC toolbox is allowed only if the differences from the original version are clearly marked.

- Distribution of the program documentation other than that included on the FDC distribution diskette is prohibited without explicit permission from the author.
- No guarantee is made as to the proper functioning of the FDC software. No liability will be admitted for damage resulting from using the software.
- Instead of a list: All trademarks used in this document and all the other documents and program files from the FDC toolbox are registered to whoever it is that owns them.

Note: the file LICENSE.TXT from the FDC distribution diskette contains a copy of these license rules. Any last-minute changes to this license agreement will be clearly marked in that file. Please read it before installing FDC 1.2.

1.4 Installing FDC 1.2

Before starting the installation, you should read the README*-file to find out if there are any last-minute changes to the software or installation procedure.* There are two different FDC 1.2 distributions: an Internet distribution and a floppy-disk distribution that uses a self-extracting ZIP file to store the models and tools. Unpacking instructions are given in a README file that is included with each distribution; these instructions will not be repeated here.

After unpacking the FDC models and tools into the right directory, start MATLAB and go to the FDC root-directory. The default FDC root-directory is `c:\fdc12` and you can type: `chdir c:\fdc12` to go there. Next, type `fdc` or `fdcinit` at the MATLAB command-line to start the initialization routine FDCINIT (see figure 1.1). As a first-time user you will be welcomed by some introductionary messages and you will be asked to check the FDC directory structure which is used by FDCINIT to enhance the MATLAB search path (figure 1.2). At this stage, it is not necessary to change anything, unless you have installed FDC to a different root-directory than C:\FDC12; in that case you must specify the new root-directory as demonstrated in figure 1.3 for the directory G:\MYTOOLS\FDC12. FDCINIT will automatically guide you through this process, but remember that the program is not (yet) able to check whether the specified directory names are correct or not, although MATLAB itself issues a warning message if you have specified an incorrect path when finishing the initialization procedure.[1] It is possible to save a changed version of the FDC directory-tree as default setting for future sessions. The initialization routine will automatically enhance the MATLAB search-path with the FDC directories. Do not run or FDCINIT more than once during an FDC-session in order to avoid duplicate entries in the MATLAB search-path.

After finishing this initialization, type `help` at the MATLAB command-line. You will see that the FDC directories have been added to the MATLAB path. Information about each individual FDC directory will be listed if you type `help dirname`, where *dirname* is an FDC subdirectory; e.g. type `help aircraft` for a short explanation about the files in the FDC subdirectory AIRCRAFT. Since most help texts don't fit in the command window, it is necessary to use the slider-button on the right side of the command window to view them completely.

[1] Note: the current user-interface of the FDC toolbox is still largely text-oriented, i.e. the programs display information in the MATLAB command window and they expect all user-inputs to be entered in the command window. However, newer versions of MATLAB automatically convert text-oriented menu's to graphical user-menu's such as the one shown in figure 1.3. This inconsistent behavior may sometimes be confusing, so further improvement of the user-interface will have high priority for future versions of the FDC toolbox. Since the command-window can be hidden behind other windows while FDC programs display information or expect inputs from the user, it is recommended to keep at least some part of the command window clear from other windows. Otherwise you may loose track of the messages which FDC sends to the user. It is therefore recommended to use a large high-resolution display.

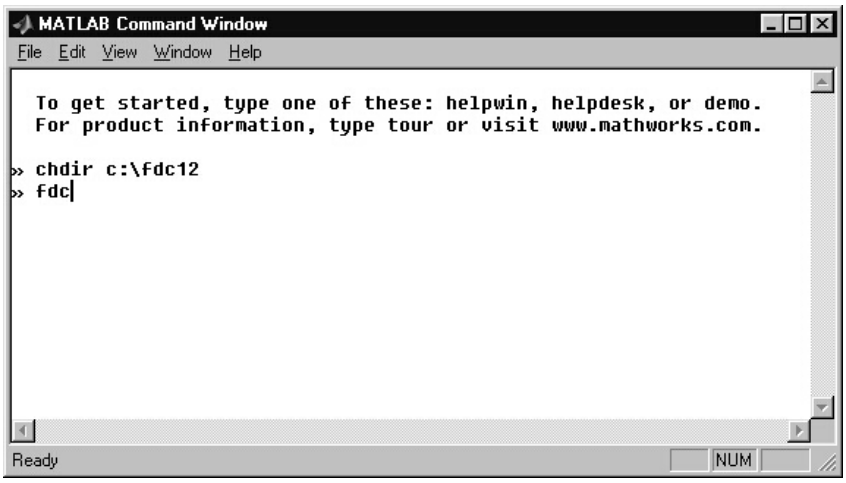


Figure 1.1: Starting the initialization of FDC 1.2

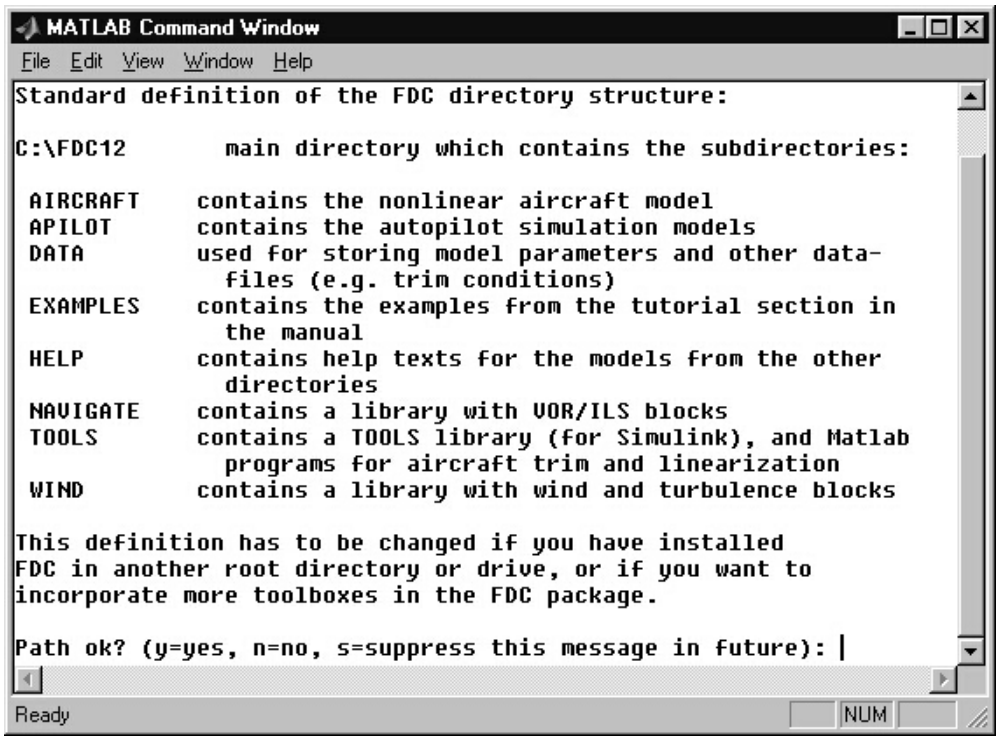


Figure 1.2: Check directory structure during initialization

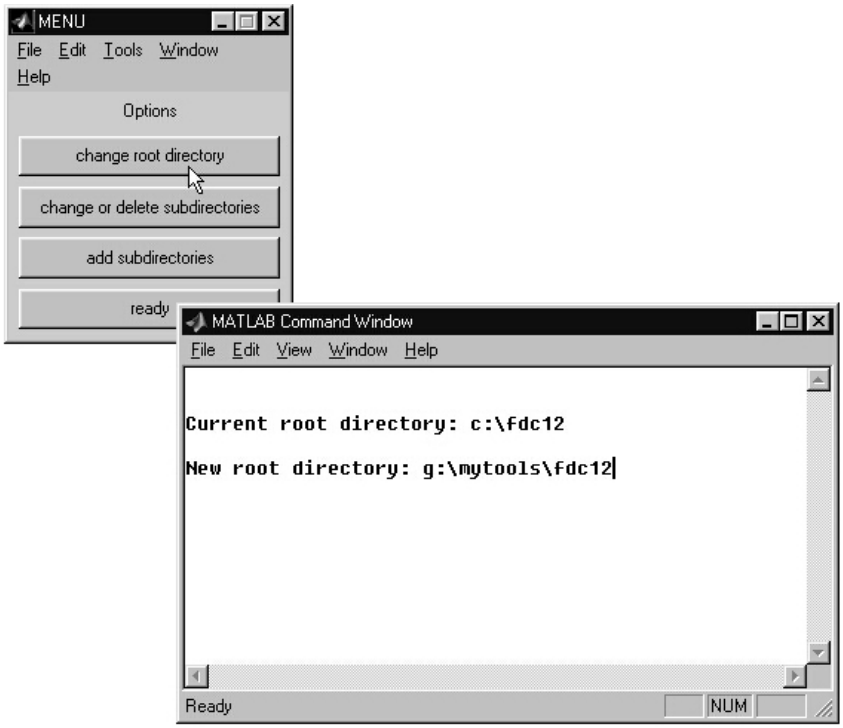


Figure 1.3: Specifying a new root-directory for FDC 1.2 during initialization

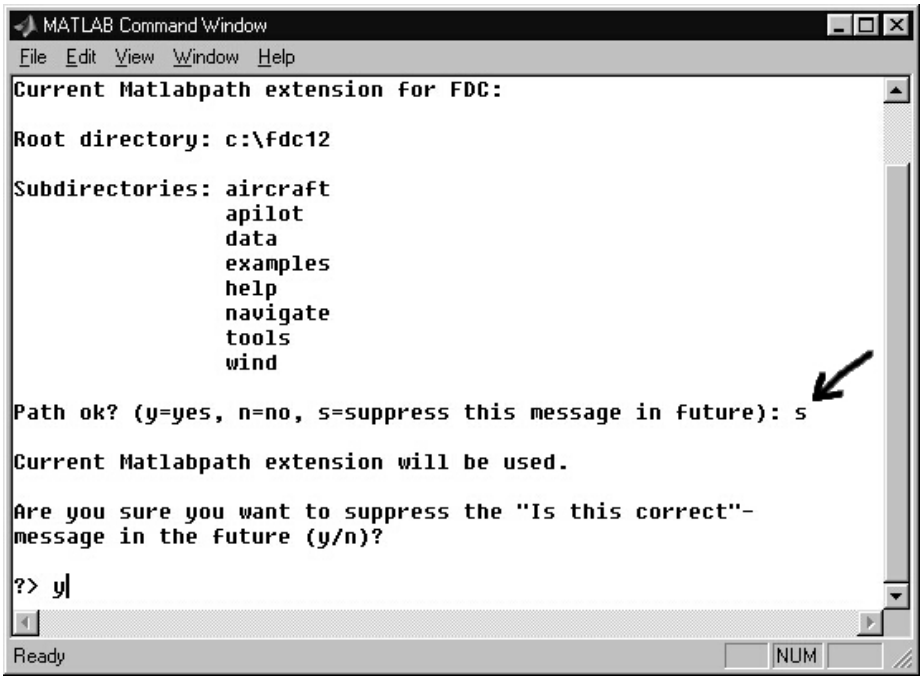


Figure 1.4: Suppressing the directory-check for future FDC-sessions

1.5 Getting started with FDC 1.2

1.5.1 Initializing FDC 1.2

When the installation is finished, the FDC toolbox contains the following subdirectories:

AIRCRAFT	contains the non-linear aircraft model Beaver, the main library FDCLIB and its sublibraries, and a model-parameter definition program MODBUILD,
APILOT	contains simulation models of the 'Beaver' autopilot,
DATA	contains datafiles with model-parameters and is used for storing data such as steady-state trimmed-flight conditions or linearized aircraft models,
DOC	contains text-files with program documentation (README-files, license agreement, list of new features, list of features for future versions of the toolbox, and a complete list of all files from the toolbox)
EXAMPLES	contains examples which demonstrate how to simulate open-loop responses of the aircraft in SIMULINK and how to access the non-linear aircraft model from the MATLAB workspace plus some 'tutorial' systems,
HELP	contains on-line help texts for the graphical subsystems and for the most important analytical tools from the FDC-package,
NAVIGATE	contains the radio-navigation library NAVLIB and its sublibraries,
TOOLS	contains the trim and linearization routines, routines for post-processing simulation results, load routines for model-initialization, and a SIMULINK library FDCTOOLS with useful blocks that can't be found in the standard SIMULINK libraries,
WIND	contains the wind and turbulence library WINDLIB and its sublibraries.

Each time you start an FDC-session you must add the FDC directories to the MATLAB path by running FDCINIT from the FDC root-directory (first change directory, then type `fdc` or `fdcinit`). After the first session the initialization routine will skip the welcome messages, but you still must check the directory path before FDCINIT will add the FDC directories to the MATLAB path. If you want to skip the directory-check in future sessions, you must select the *suppress* option after checking the directories (see figure 1.4). Note: if you plan to enhance the FDC toolbox, it is advisable not to suppress the directory-check, because it can be used to include new subdirectories for your own FDC extensions to the MATLAB path; just click the appropriate button in the menu shown in figure 1.3 and answer the questions in the MATLAB command window. FDCINIT always tries to retrieve its default settings from the file FDCINIT.INI, which is stored in the root-directory of the FDC package. If you change the FDC directory tree and save this definition as the new default setting, the file FDCINIT.INI will be changed accordingly. Once the *suppress* option has been selected during a directory-check, it is not possible to change the tree definition anymore, unless you delete the file FDCINIT.INI from the FDC root-directory before running FDCINIT. If FDCINIT.INI cannot be found in the FDC root-directory, you will be welcomed as if you were a first-time user and the directory-check options will reappear.

1.5.2 Contents of the FDC toolbox

The most important elements from the FDC toolbox are its SIMULINK model libraries: the main library FDCLIB, the wind and turbulence library WINDLIB, and the radio-navigation library NAVLIB. These libraries can be opened by typing `fdclib`, `windlib`, or `navlib` at the MATLAB command-line. FDCLIB provides links to sublibraries containing the different blocks from the non-linear aircraft model, the complete non-linear model of the 'Beaver' aircraft, the wind and navigation libraries, several 'button'-blocks, and some example systems which explain how the

different FDC models can be used in practice. Typing `fdclib` at the MATLAB command-line will open the block-library shown in figure 1.5. Each block from this library can be double-clicked with the mouse to open the corresponding sublibrary or simulation model. The title block in the upper left corner of the library can be double-clicked to provide information about some important block-conventions from the graphical systems in FDC 1.2. The blocks from the left hand side of the main library window will open sublibraries of the aircraft model when double-clicked. The blocks on the right hand side refer to complete simulation models and 'tutorial systems'.

For instance, double-clicking the block Aerodynamics will reveal the sublibrary shown in figure 1.6. This library currently contains the aerodynamic models for the 'Beaver' aircraft, but it can easily be enhanced with aerodynamic models for other aircraft. Opening such sublibraries is useful during model construction work. Since it takes some time to load these sublibraries, it is recommended not to close these libraries until your model construction work is actually finished. If you want to make room on your screen you can better temporarily *minimize* the window instead of closing it. The main library remains accessible in a separate window as long as that window is not closed.[1]

The complete non-linear aircraft model is contained in the SIMULINK system Beaver, which can also be accessed directly by typing `beaver` at the command-line. Examples of open-loop simulation models are contained in the systems OLOOP1, OLOOP2, and OLOOP3, which can either be opened directly by typing `oloop1`, `oloop2`, or `oloop3` at the command-line, or accessed via the main library FDCLIB. 'Tutorials' for these open-loop simulation systems are also available; they are called OLOOP1T, OLOOP2T, and OLOOP3T respectively. Complete autopilot simulation models for the 'Beaver' aircraft are available in the systems APILOT1, APILOT2, and APILOT3. It is recommended not to use these systems until you have gained some experience with the open-loop simulation models, because the autopilot models are relatively complex in comparison to the other systems. This is why the autopilot systems are not accessible from the main model library FDCLIB. See chapters 11 and 12 for a complete description of the autopilot models from FDC 1.2.

1.6 Some warnings

Although it is highly recommended to use the different models and tools from the FDC toolbox for your own experiments, it is important not to neglect the current interactions between the different models and tools. For instance, increasing the number of output signals from the aircraft model by adding new Outport blocks in the first level of the aircraft model will require appropriate changes in the systems that call the aircraft model, e.g. the open-loop simulation models or the autopilot models. Since all parts of the FDC systems are freely accessible by the user, the possibilities to experiment are virtually unlimited. But in order to maintain the integrity of the complete FDC toolbox, it is advised to keep safety copies of the original files somewhere on your system in order to be able to restore possible errors arising during the editing of those files (of course it is always possible to re-install the complete package if things go really wrong), and it is strongly recommended to use different filenames for your own adaptations of the FDC models and tools to clearly distinguish them from the originals. If you want to *distribute* your own developments you are even *obliged* to do this, according to the license agreement from section 1.3.

[1] Tip for MS WINDOWS: under SIMULINK the screen tends to become crowded with all kinds of windows, especially if it is small. To quickly find the appropriate window, try using Alt + Tab or Alt + Shift + Tab to walk through a list of windows currently displayed at the screen, including the minimized windows.

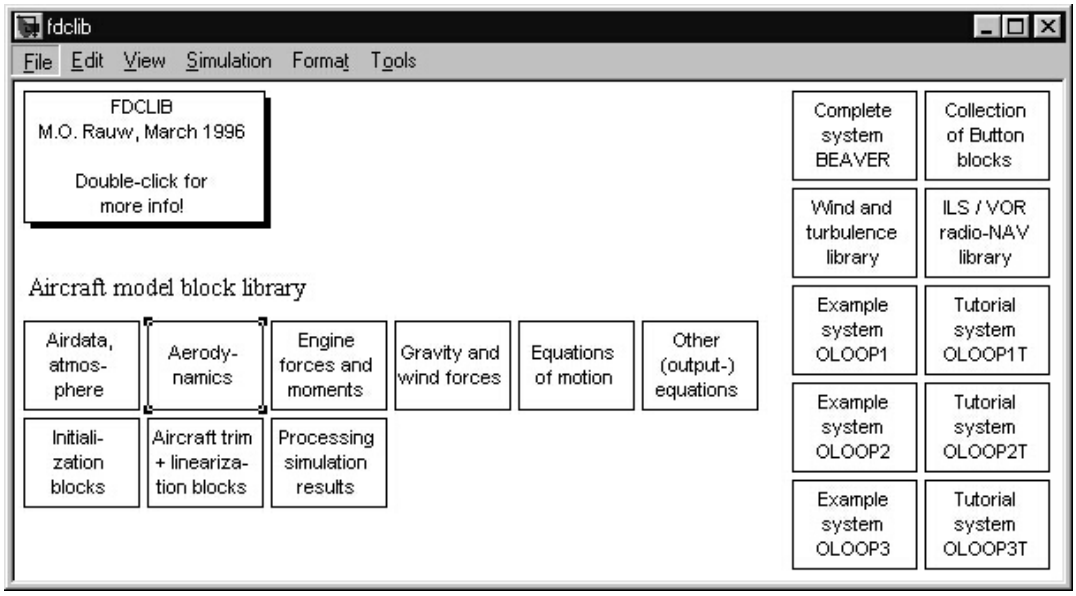


Figure 1.5: Main block-library of the FDC toolbox

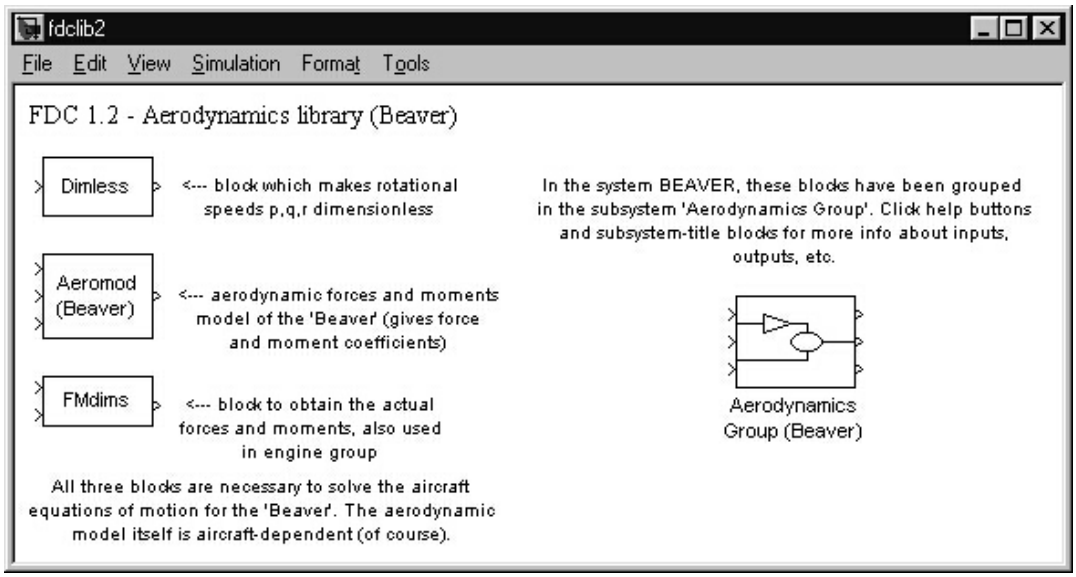


Figure 1.6: Sublibrary with aerodynamic models

If you decide not to read the remainder of this report, in particular chapter 10, you are bound to receive some disturbing error messages when performing experiments with the tools and models from the FDC toolbox. This is quite harmless, but please don't blame the product for it. All systems and tools from FDC 1.2 have performed quite well in practice, so if SIMULINK complains about things like missing parameters you'll probably have done something wrong. The toolbox has become more user-friendly (not to say more fool-proof) since the first versions, but it still requires sufficient basic knowledge to use it. So please keep on reading a bit longer... If you want to get started quickly, the best thing to do now is to continue with chapter 10. See also figure 1 from the preface.

Chapter 2

The Flight Control System design process

2.1 Introduction

Active flight control technology has dramatically changed the way aircraft are designed and flown: the flying qualities of modern aircraft are largely determined by a set of *control laws* in the heart of a computer system! Flight control systems with mechanical linkages have been replaced by full authority, fly-by-wire, digital control systems. Such Automatic Flight Control Systems (AFCS's) can be designed and analyzed effectively by incorporating design-techniques and mathematical dynamic models in a user-friendly Computer Assisted Control System Design (CACSD) package. The FDC-toolbox for MATLAB / SIMULINK is a practical example of such a design-environment. This chapter will outline the general control system *design cycle*, in order to stress the importance of such a CACSD environment for AFCS design and analysis.

2.2 The AFCS design cycle

A practical division of the AFCS design process into a number of different phases is given in ref.[20]. Although this reference is nowadays outdated with respect to the available tools (computer hardware and software), this division is still quite useful:

1. Establish the system purpose and overall system requirements. System purpose can be equated with mission or task definitions. System requirements can be separated in *(i) operational requirements*, derived from the functions needed to accomplish the mission phases and *(ii) implied requirements*, derived from the characteristics of the interconnected components of the control system and the environment in which they operate.
2. Determine the characteristics of *unalterable* elements, command-signals, and external disturbances. The characteristics of some parts of the system cannot easily be changed by the designer. Often the vehicle itself, its control surface actuators and some of its sensors are 'unalterable'.[1] Moreover, the *structure* of the commands and disturbances is a direct consequence of the mission requirements and the environment in which the control system has to operate.

[1] If the AFCS is designed for an all-new aircraft, the selection of the hardware (sensors, actuators, computers, etc.) must be included in the AFCS design and analysis in stead of taking the hardware as being unalterable. In this report, all hardware is considered to be given, so the 'only' problem to be considered is the development of the appropriate control laws to make a given aircraft fly a certain mission.

3. Evolve competing feasible systems (i.e. determine the basic block-diagrams). Usually, there are more ways to achieve the requirements, e.g. with different sensed motion quantities and/or the application of different control theories. Then it is possible to evolve competing candidate systems for selection on the basis of certain desirable properties.
4. Select the 'best' system. The competing designs can be compared on the basis of *(i) design qualities*, which include dynamic performance (speed of response, bandwidth, etc.) and physical characteristics (weight, volume, power consumption, etc.), and *(ii) design quantities*, which include safety, reliability, maintainability, cost, etc. An optimum system is one that has some 'best' combination of these features.
5. Study the selected system in detail. The selected system must be evaluated for all normal and abnormal operating conditions. At each state of the AFCS *validation* the assumptions made earlier in the AFCS design must be checked for validity. If necessary, a new iteration of the design should be started from the point where the wrong assumption was made.

This scheme reflects the AFCS design process within a manufacturing environment. In a research context, the scheme has to be modified somewhat, due to the differences between research and manufacturing tasks. The task of research is to determine what is required and to produce a clear and comprehensive definition of the requirements; the manufacturing task is to make and deliver a reliable and effective product (ref.[27]). Therefore, the first AFCS design phase in particular will be different in a research environment, because the system requirements are often poorly understood or may even be the objective of the research itself. In addition, the design tools may be immature, and their development may again be an objective of the research (the development of the FDC toolbox from this report is an obvious example). The design, simulation, and implementation of control laws within a research context will be similar to the production application, although more flexibility of the tools will sometimes be required. For instance, it should be possible to rapidly alter the control laws within the Flight Control Computers (FCC's) of the aircraft in order to evaluate different solutions to typical control system design problems with a minimum of programming efforts. In the research environment, step 4 does not necessarily need to include the selection of a 'best' system since it may be useful to evaluate competing control solutions all the way up to evaluation in real flight, just to get more knowledge about their advantages and disadvantages. Moreover, the requirements with respect to fail-safety of the AFCS may be less restrictive in a research context than for manufacturing. For instance, during the autopilot design project for the DHC-2 'Beaver' laboratory aircraft, only one FCC (a portable 80286 PC (!), coupled to a 16-bit ROLM computer that handled the I/O functions) was used, whereas AFCS's for production aircraft apply multiple FCC's which cross-check eachother's command signals.

Figure 2.1 shows the first step in the AFCS design process: the definition of the *mission* to be fulfilled by the aircraft, which imposes requirements upon the shape of the *flight-path* and the velocity along this flight-path. The resulting control problem is therefore to generate appropriate deflections of aerodynamic control surfaces or changes in engine power or thrust, necessary to fulfill the mission. This control problem is sketched in figure 2.2. The classical approach to the AFCS design problem is to start with the complete set of non-linear equations of motion, and then make assumptions which enable these equations to be linearized about some local equilibrium point. In the initial phase of the AFCS design project, control system design tools based upon linear system theory can be applied to these linearized models of the aircraft and its subsystems. Programs like MATLAB provide the required computer support for these applications. Although the linear control system design and analysis techniques will provide insight in the essential behavior of the AFCS, only relatively small deviations from the equilibrium state are

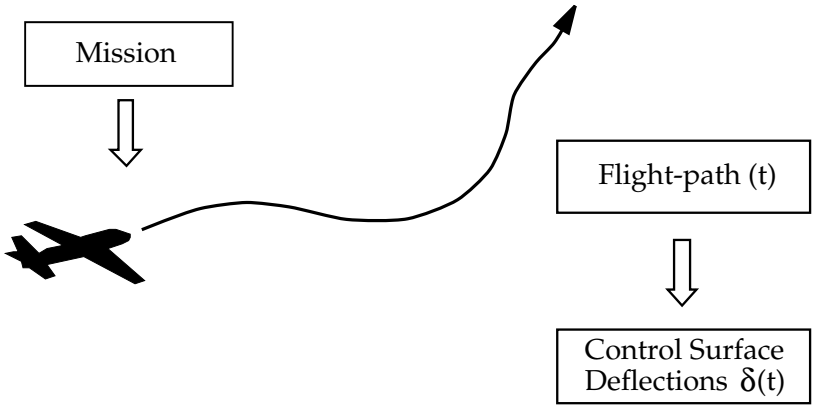


Figure 2.1: Definition of the aircraft's mission

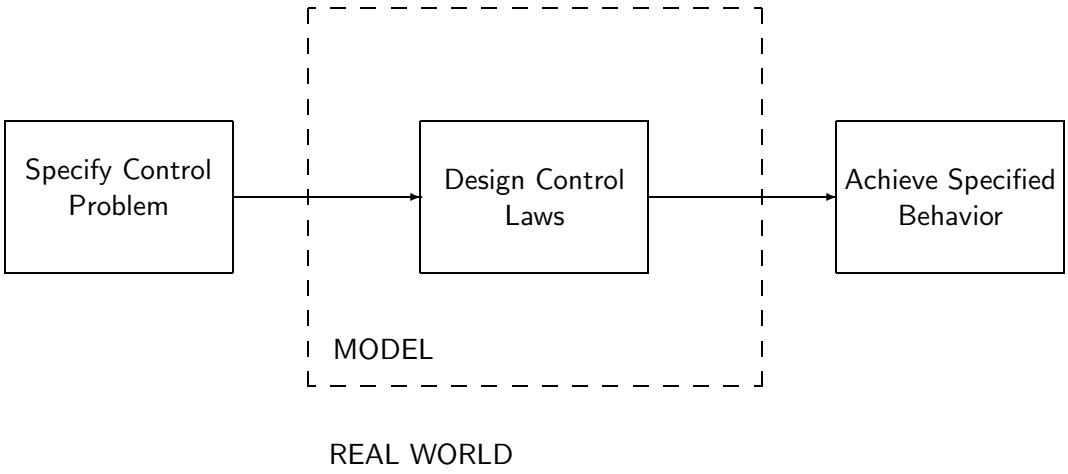


Figure 2.2: The general flight control problem

permitted before the results become invalid in comparison to the real aircraft. Luckily the main purpose of many AFCS control laws themselves is to keep the deviations from the equilibrium state as small as possible, e.g. in order to keep a certain altitude or heading. However, there are other control laws which due to their nature require large deviations from the nominal values, e.g. for selecting a *new* reference heading which differs considerably from the original value. For this reason, detailed *non-linear* simulations must be made in order to validate (and enhance) the results of the linear analysis and design. This will ensure that the AFCS works well over the compete range of the flight-envelope for which it is designed, taking into account a suitable safety margin. This analysis covers a wide range of velocities and altitudes and all possible aircraft configurations. This non-linear *off-line* analysis of the control system can be performed on one PC or workstation, using a software environment such as MATLAB / SIMULINK. 'Off-line' in this respect means that the analysis does not have to be performed in real-time and does not yet include piloted flight-simulation. In a later stage the control system should be evaluated in a real-time flight-simulator, to enable a test-pilot to assess the handling qualities of the automatically controlled aircraft. In particular, the pilot–aircraft interaction should be examined thoroughly, especially if the pilot will be actively involved in the aircraft control loop (which obviously is true for so-called fly-by-wire systems).

Based upon these results it is possible to choose the best solution if there are more feasible solutions to fulfill the mission requirements. If the results of the on-line and off-line analysis are completely satisfactory the next step will be the implementation of the control laws in the Flight Control Computer(s) of the aircraft. The aircraft must be equipped with suitable actuators and sensors, which must be thoroughly tested and calibrated. For some purposes, e.g. certification purposes, it even may be useful to test the complete control system in an *Iron Bird* test-stand arrangement which provides hardware-in-the-loop simulation capabilities. In order to reduce the risks at making conversion errors, it should be possible to couple at least the complete FCC software, but preferably also its hardware, to the real-time flight-simulator and the off-line design environment. This ensures that the control laws evaluated in flight are exactly the same as the last versions analyzed on-ground.[1]

After successfully concluding the simulations and ground tests of the hardware and FCC software, the AFCS can be evaluated in real flight. In an ideal world this phase would only be a straightforward verification of the previous results, but in practice it is often necessary to return to a previous stage in the AFCS development for fixing errors or fine-tuning the control laws. It also may be necessary to update the mathematical models if deficiencies in these models are found during the in-flight evaluations. Quantitative results from the flight tests need to evaluated on-ground to confirm the correct control behavior, for which purpose the off-line environment for AFCS design and analysis is again an ideal platform.

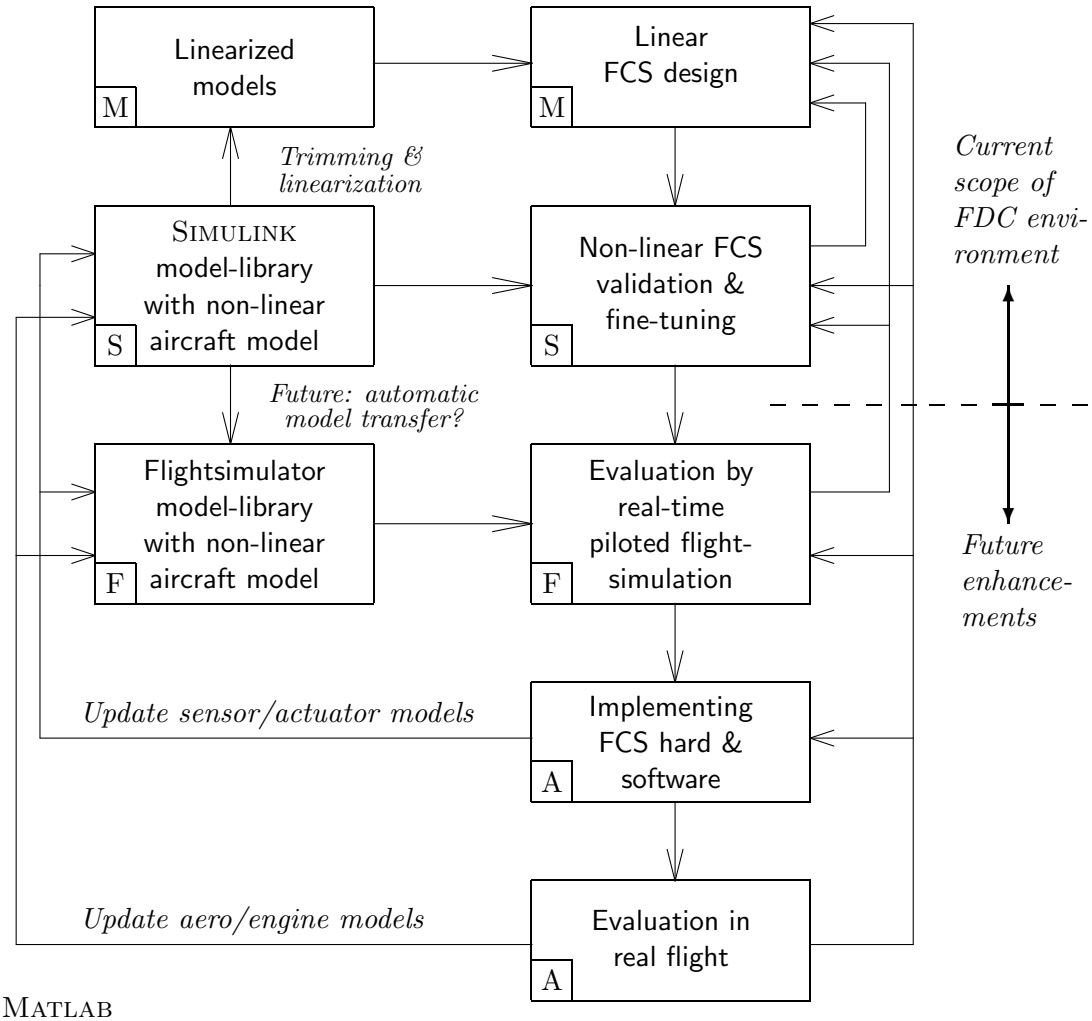
The *iterative* nature of this AFCS development cycle should be acknowledged: at any stage in the process, the discovery of a fault, design error, or previously unrecognized uncertainty requires the return to a previous design stage. It is therefore very important that the transitions between the different development phases are made as straightforward as possible, to reduce the number of transition errors which inevitably will arise if the tools for the different phases are not compatible (Murphy's Law), and to reduce the *time* needed for the AFCS development. For this reason, the AFCS designer needs to have access to the analytical tools for linear and non-linear AFCS design and analysis along with the required mathematical models within an

[1] For the 'Beaver' autopilot project some dramatic examples of conversion errors were encountered, luckily for a large part *before* we actually started the flight-tests. Still, *some* errors were discovered during the actual flight-tests! In this respect, references [22] and [29] provide ample food for thought with regard to possible future AFCS projects... See also the recommendations from chapter 13.

integrated software environment on a single PC or workstation. The software tools for off-line design and analysis should be able to effectively communicate with eachother and with the tools for real-time on-line simulations and the FCC. Moreover, the designer should be able to manipulate all elements of a specific control system, as well as the mathematical models involved in a specific design task, by means of a graphical user-interface.

Examples of integrated AFCS design environments are presented in refs.[8] and [24]. Those papers particularly emphasize the need for *multidisciplinary design* in which aerodynamic, structural, propulsive, and control functions are considered all together. This is important, because modern flight controllers may excite structural modes of the aircraft and interact with the control-actuator dynamics, and because of the increasing need to integrate flight controls with engine controls and load-alleviation functions. Interactions between the aerodynamic, propulsive, and structural models must be taken into account. For future aircraft, the interacting phenomena will become even stronger because they will employ such features as the extensive use of composite materials (resulting in greater aero-elastic coupling) and relaxed static stability (refs.[8] and [24]).

Figure 2.3 summarizes the complete AFCS design cycle. It illustrates the division in different design stages from ref.[20] and the more detailed divisions presented in refs.[8], [24] and [27], and it clearly shows the iterative nature of the whole process. On the left-hand side of the figure the models and tools (software and hardware environments) are shown, while the right-hand side shows the design stages themselves. The figure reflects the current scope of the MATLAB/SIMULINK-based FDC toolbox. As yet it is still necessary to make manual conversions of the control laws to cross the dashed line between off-line and on-line analysis, which still considerably hampers the AFCS development and increases the risk of making errors. Future versions of the toolbox should therefore be equipped with interfacing-tools that simplify this step. It will be a big leap forwards if it becomes possible to automatically transfer complete simulation models from the SIMULINK-environment to the flight-simulator as shown in the figure. A first step towards that goal would be the automatic conversion of the *control laws* from SIMULINK to a high-level programming language. This step will be elaborated further in section 13.2.



M = MATLAB
S = SIMULINK
F = *Real-time flightsimulator*
A = *Actual aircraft*

Figure 2.3: The AFCS design cycle

Chapter 3

Mathematical models – theoretical backgrounds

3.1 Introduction

The FDC toolbox was built around several mathematical models needed for the analysis of the aircraft dynamics and the design and evaluation of Automatic Flight Control Systems. Figure 3.1 shows the closed-loop structure of an automatically controlled aircraft that is affected by external disturbances. In this structure, the models of the aircraft dynamics, sensors, actuators, and computational effects are the basic elements which often can't be altered by the AFCS designer (see section 2.2). The AFCS control laws and the mode-controller which configures the control laws for a specific pilot-selected control task are the subject of our analysis. The design and fine-tuning of these control laws requires application of control system design theory to the dynamic models of the other elements from figure 3.1. Since the motions of the aircraft, and hence the performance of the AFCS, are affected by external disturbances, it is necessary to include these influences to the dynamic model of the aircraft and to derive mathematical models which describe the nature of these disturbances themselves. The external disturbances are mainly due to non-steady atmosphere.

In this chapter the different elements from figure 3.1 will be elaborated, with the exception of the AFCS control laws and the mode-controller. Most attention will be given to the non-linear model of the aircraft dynamics. Due to the clear modular structure of this model it can be applied to a very wide range of aircraft or other vehicles such as spacecraft, ships, or road-vehicles. The few aircraft-dependent parts within this model will be worked out here for the DHC-2 'Beaver' aircraft. That model is quite sophisticated, yet very compact, which makes it especially suited for an introductionary treatment of the structure of the aircraft-dynamics and for the practical demonstration within the 'proof of concept' toolbox from this report. Of the other dynamic models from figure 3.1 some typical examples will be given to provide a basis for a complete model library. In section 3.3 only *atmospheric* disturbances will be elaborated. Section 3.4 describes some important models of radio-navigation signals (VOR and ILS), and section 3.5 gives a brief overview of some other sensor and actuator models that were used for the design and analysis of the 'Beaver' autopilot. The control laws of the 'Beaver' autopilot will be treated in chapter 11 as a practical demonstration of a complete AFCS design project.

Note: a complete list of all symbols, reference frames, abbreviations, and other definitions from this report has been included in appendix A.

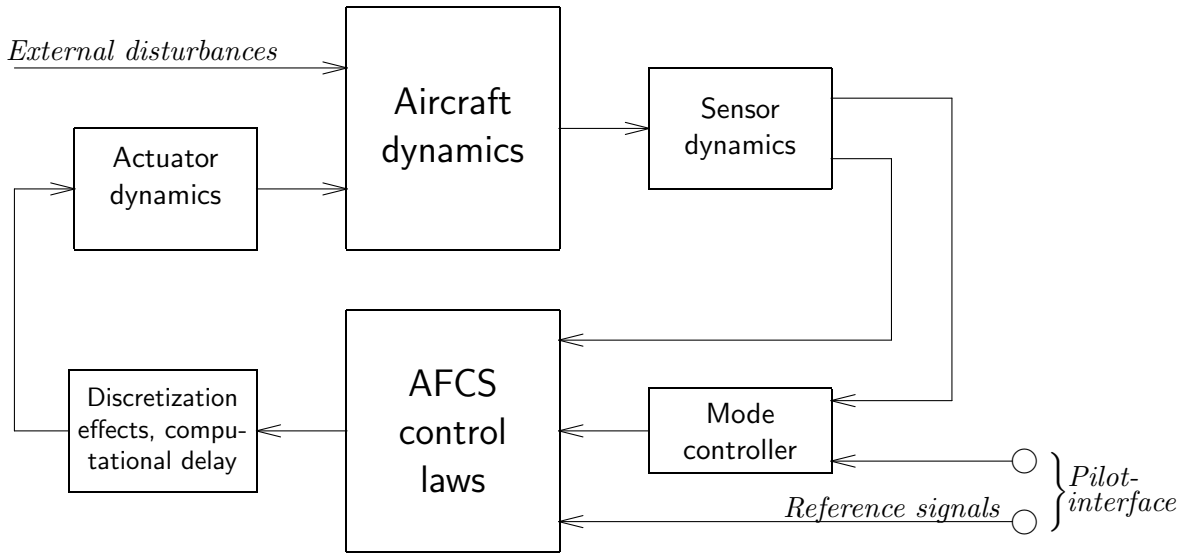


Figure 3.1: Block-diagram of automatically controlled aircraft

3.2 The non-linear aircraft model

The core of the simulation package is a non-linear model of the aircraft dynamics, consisting of twelve Ordinary Differential Equations (ODE's or *state equations*), and a large number of output equations. This model can be broken down into a number of different modules, most of which are independent of the kind of moving vehicle under consideration.

3.2.1 General equations of motion

The aircraft equations of motion are derived from basic Newtonian mechanics. The general force and moment equations for a rigid body are:

$$\mathbf{F} = m\left(\frac{\partial \mathbf{V}}{\partial t} + \mathbf{\Omega} \times \mathbf{V}\right) \tag{3.1}$$

$$\mathbf{M} = \frac{\partial(\mathbf{I} \cdot \mathbf{\Omega})}{\partial t} + \mathbf{\Omega} \times (\mathbf{I} \cdot \mathbf{\Omega}) \tag{3.2}$$

These equations express the motions of a rigid body relatively to an inertial reference frame (see appendix B for the derivation of these rigid body equations). $\mathbf{V} = [\ u\ \ v\ \ w\ ]^T$ is the velocity vector at the center of gravity, $\mathbf{\Omega} = [\ p\ \ q\ \ r\ ]^T$ is the angular velocity vector about the c.g. $\mathbf{F} = [\ F_x\ \ F_y\ \ F_z\ ]^T$ is the *total* external force vector, and $\mathbf{M} = [\ L\ \ M\ \ N\ ]^T$ is the *total* external moment vector. $\mathbf{I}$ is the inertia tensor of the rigid body, which is defined as:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -J_{xy} & -J_{xz} \\ -J_{yx} & I_{yy} & -J_{yz} \\ -J_{zx} & -J_{zy} & I_{zz} \end{bmatrix} \tag{3.3}$$

The coefficients from this tensor are the moments and products of inertia of the rigid body. If the frame of reference is fixed to the vehicle these values are constant, regardless of the attitude of the vehicle. In order to make equations (3.1) and (3.2) usable for control system design and analysis, simulation purposes, system identification, etc., these equations need to be re-written in non-linear *state-space* format. Moving the time-derivatives of the linear and angular velocities to the left hand side of the equations yields:

$$\frac{\partial \mathbf{V}}{\partial t} = \frac{\mathbf{F}}{m} - \mathbf{\Omega} \times \mathbf{V} \tag{3.4}$$

$$\frac{\partial \mathbf{\Omega}}{\partial t} = \mathbf{I}^{-1}\left(\mathbf{M} - \mathbf{\Omega} \times \mathbf{I} \cdot \mathbf{\Omega}\right) \tag{3.5}$$

Together these dynamic equations form a state-space system which is valid for *any* rigid body, e.g. aircraft, spacecraft, road-vehicles, or ships. These equations obviously form the core of the simulation model. The body-axes components of linear and rotational velocities can be regarded as the *state* variables from this model, while the body-axes components of the external forces and moments are the *input* variables of these equations.

This clear picture is complicated by the fact that the external forces and moments themselves depend upon the motion variables of the aircraft. In other words: the state variables themselves must be coupled back to the force and moment equations. Although this makes the equations more complex, it is still possible to combine these equations in a non-linear state space system:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{F}_{tot}(t), \mathbf{M}_{tot}(t)\right) \tag{3.6}$$

with:

$$\begin{aligned} \mathbf{F}_{tot} &= \mathbf{g}_1\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t\right) \\ \mathbf{M}_{tot} &= \mathbf{g}_2\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t\right) \end{aligned}$$

This set of equations is equivalent to the single non-linear state equation:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t\right) \tag{3.7}$$

with state vector $\mathbf{x}$, input vector $\mathbf{u}$, disturbance vector $\mathbf{v}$, and time t. From equations (3.4) and (3.5) it is obvious that the state vector $\mathbf{x}$ at least must contain the linear and angular velocity components from the vectors $\mathbf{V}$ and $\mathbf{\Omega}$. Later it will be shown that six additional state variables, defining the *attitude* and *position* of the aircraft with respect to the Earth, have to be introduced for solving these equations. It is possible that the forces and/or moments do not only depend on the state vector $\mathbf{x}$ but also of its time-derivative $\dot{\mathbf{x}}$. This makes equation (3.7) *implicit*:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{u}(t), \mathbf{v}(t), t) \tag{3.8}$$

Luckily, this implicit relation often can be written like:

$$\dot{\mathbf{x}} = \mathbf{f}_1(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t) + \mathbf{f}_2(\dot{\mathbf{x}}(t), t) \tag{3.9}$$

which makes it somewhat easier to solve numerically, especially if $\mathbf{f}_2$ is a linear function. The practical consequences of this will be outlined for the dynamic model of the 'Beaver' in section 3.2.3.

The state vector $\mathbf{x}$ obviously contains linear and angular velocity components, i.e. the elements from $\mathbf{V}$ and $\mathbf{\Omega}$. In addition to these variables, information about the *spatial orientation* of the aircraft is needed for finding the gravitational force contributions. Furthermore, the *altitude* of the aircraft is needed for the computation of aerodynamic and engine forces which are both affected by changes in air density that depend upon the altitude of the aircraft. The coordinates of the aircraft with respect to the Earth are not needed for solving the equations of motion, but they are useful for other purposes, such as the assessment of the flight-path for certain manoeuvres. Therefore, the complete state vector $\mathbf{x}$ will consist of twelve elements: three linear velocities, three angular velocities, three Euler angles which define the attitude of the aircraft relatively to the Earth, two coordinates and the altitude which define the position of the aircraft, relatively to the Earth. In practice, it turns out to be more convenient to use the true airspeed, angle of attack, and sideslip angle instead of the linear velocity components along the body-axes of the aircraft, yielding the following state vector:

$$\mathbf{x} = [\ V\ \ \alpha\ \ \beta\ \ p\ \ q\ \ r\ \ \psi\ \ \theta\ \ \varphi\ \ x_e\ \ y_e\ \ H\ ]^T \tag{3.10}$$

Appendix B contains a full derivation of the equations of motion, including the kinematic relations which define the attitude and position of the aircraft and the conversion equations from the body-axes velocities to the true airspeed V, angle of attack α, and sideslip angle β. Figure 3.2 gives a graphical overview of the non-linear rigid body dynamics. Together, all elements from this figure represent the non-linear state-space system from equation (3.6). The state variables are obtained by integrating their time-derivatives with respect to time, taking into account the initial value of the state vector, $\mathbf{x}_0$. In order to obtain the time-derivatives of the state variables the state variables are coupled back to the force and moment equations and the equations of motion themselves. All forces and moments must be expressed in components along the body-axes of the vehicle (denoted by the superscript B). Forces and moments which are expressed with respect to other reference frames must be transformed to body-axes components by pre-multiplying the force and moment vectors with the appropriate transformation matrix. In the figure, this is illustrated for the aerodynamic forces and moments, which are transferred from flight-path axes (superscript W) to body-axes, and for the gravitational forces, which are transferred from Earth axes (superscript E) to body-axes. Figure 3.2 forms the basis for the development of the modular structure of the rigid body equations for the FDC 1.2 toolbox.

3.2.2 External forces and moments

The next step in the development of the dynamic model is to identify the different contributions to the external forces and moments acting upon the rigid body. Obviously these contributions are dependent of the kind of vehicle under consideration. Here, forces and moments due to gravitational, propulsive, and aerodynamic effects, plus the influence of non-steady atmosphere will be considered. This comprises an *in-flight* model of a conventional aircraft. Other contributions can easily be included to the model, e.g. ground forces during taxiing, or a ground-effect model for aircraft that fly close to the ground.

Aerodynamic Forces & Moments

The aerodynamic forces and moments depend upon the flight condition, defined by the state vector $\mathbf{x}$, and the external aerodynamic control inputs, defined by the input vector $\mathbf{u}_{aero}$. These inputs are the deflections of the aerodynamic control surfaces (elevator, ailerons, rudder) and the deflection of the flaps. For the DHC-2 'Beaver' aircraft, a sophisticated aerodynamic model has been determined from flight tests in 1988, see ref.[26]. This model expresses the aerodynamic forces and moments along the aircraft's body-axes in terms of *polynomial functions* of the state and input variables and the time-derivative of the state vector:

$$\mathbf{F}_{aero} = \mathbf{d} \cdot \mathbf{p}_1 \left(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}_{aero} \right) \tag{3.11}$$

where $\mathbf{F}_{aero}$ is a vector of aerodynamic forces and moments, and $\mathbf{p}_1$ is a polynomial vector-function that yields *non-dimensional* force and moment coefficients. For the 'Beaver' model, the $\dot{\mathbf{x}}$-term is linear and only takes into account the direct contribution of $\dot{\beta}$ to the aerodynamic side-force Y_a. The pre-multiplication with the vector $\mathbf{d}$ converts these non-dimensional coefficients to dimensional forces and moments; $\mathbf{d}$ equals:

$$\mathbf{d} = q_{dyn} S \left[\begin{array}{cccccc} 1 & 1 & 1 & \frac{b}{2} & \overline{c} & \frac{b}{2} \end{array} \right]^T \tag{3.12}$$

S is the wing-area of the aircraft, b is the wing-span, $\overline{c}$ is the mean aerodynamic chord, and q_{dyn} is the dynamic pressure ($q_{dyn} = \frac{1}{2}\rho V^2$, see section 3.2.4).

The polynomial functions from $\mathbf{p}_1$, describing the aerodynamic force and moment coefficients in

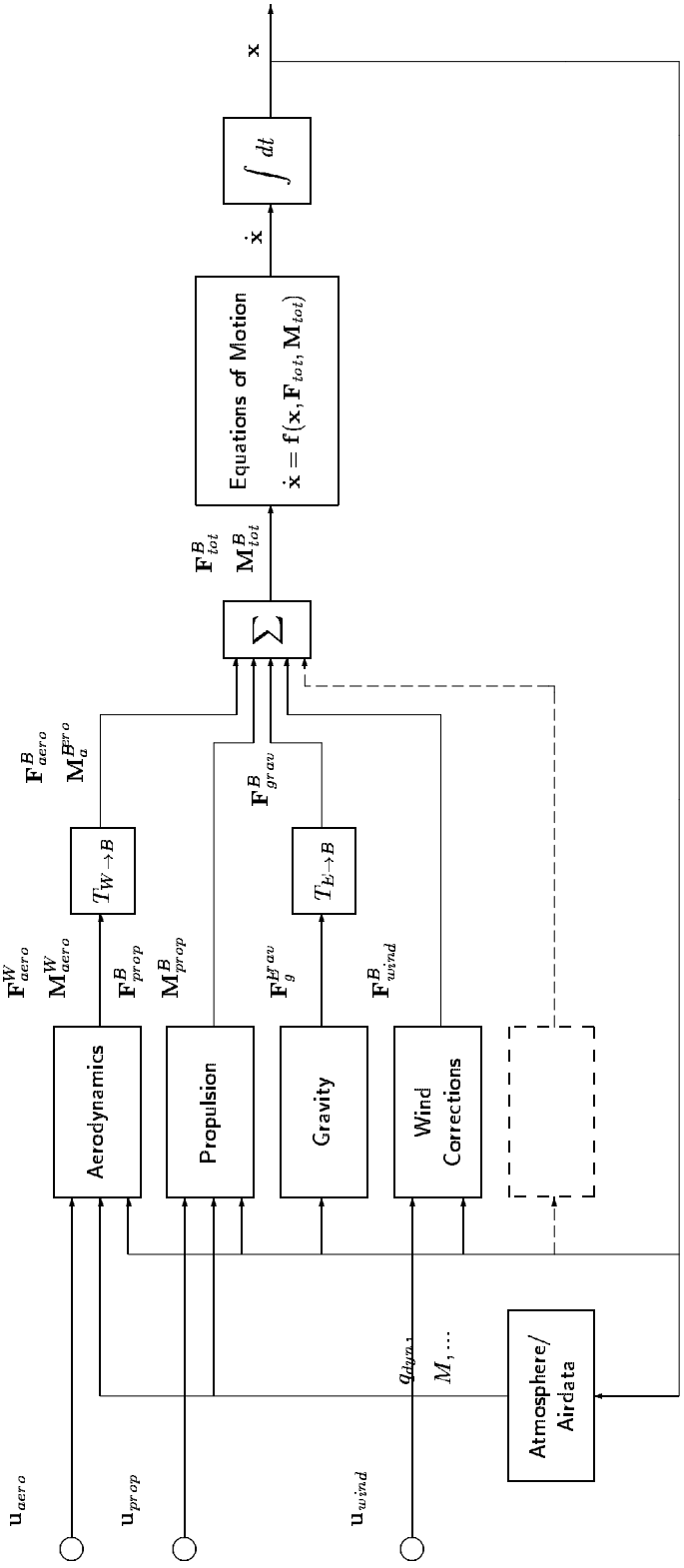


Figure 3.2: Block-diagram of general rigid body dynamics

the body-fixed reference frame are:

$$\begin{aligned}
C_{X_a} &= C_{X_0} + C_{X_\alpha}\alpha + C_{X_{\alpha^2}}\alpha^2 + C_{X_{\alpha^3}}\alpha^3 + C_{X_q}\frac{q\overline{c}}{V} + C_{X_{\delta_r}}\delta_r + C_{X_{\delta_f}}\delta_f + C_{X_{\alpha\delta_f}}\alpha\delta_f \\
C_{Y_a} &= C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{p\,b}{2V} + C_{Y_r}\frac{rb}{2V} + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r + C_{Y_{\delta_r\alpha}}\delta_r\alpha + C_{Y_{\dot\beta}}\frac{\dot\beta b}{2V} \\
C_{Z_a} &= C_{Z_0} + C_{Z_\alpha}\alpha + C_{Z_{\alpha^3}}\alpha^3 + C_{Z_q}\frac{q\overline{c}}{V} + C_{Z_{\delta_e}}\delta_e + C_{Z_{\delta_e\beta^2}}\delta_e\beta^2 + C_{Z_{\delta_f}}\delta_f + C_{Z_{\alpha\delta_f}}\alpha\delta_f \\
C_{l_a} &= C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{p\,b}{2V} + C_{l_r}\frac{rb}{2V} + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta_r + C_{l_{\delta_a\alpha}}\delta_a\alpha \\
C_{m_a} &= C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\alpha^2}}\alpha^2 + C_{m_q}\frac{q\overline{c}}{V} + C_{m_{\delta_e}}\delta_e + C_{m_{\beta^2}}\beta^2 + C_{m_r}\frac{rb}{2V} + C_{m_{\delta_f}}\delta_f \\
C_{n_a} &= C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{p\,b}{2V} + C_{n_r}\frac{rb}{2V} + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r + C_{n_q}\frac{q\overline{c}}{V} + C_{n_{\beta^3}}\beta^3
\end{aligned} \tag{3.13}$$

See table C.3 in appendix C for the values of the stability and control coefficients from these polynomial equations. Notice the cross-coupling between *lateral* motions and *longitudinal* forces and moments. Also notice the contribution of $\dot\beta$ to the aerodynamic side-force Y_a, which explains the occurrence of $\dot{\mathbf{x}}$ in the general polynomial equation (3.11). Due to this phenomenon the state equation (3.6) becomes implicit. In general such relationships are linear, as is illustrated here for the 'Beaver' model. This makes it easy to re-write the state equations as a set of explicit ODE's, as outlined for the DHC-2 'Beaver' aircraft in section 3.2.3. Table C.2 in appendix C gives the flight-condition for which the aerodynamic model has been determined. Corrections to the body-axes moments are necessary if a different position of the center of gravity is used, see ref.[26].

Engine Forces & Moments

The engine forces and moments strongly depend upon the type of aircraft under consideration, in a similar way as the aerodynamic forces and moments. For a piston-engined aircraft like the 'Beaver', the primary engine control inputs are the engine speed n, and the manifold pressure p_z, which directly affect the engine power P. The engine power also varies with altitude due to changes in air-density. In the case of the 'Beaver' aircraft, changes in engine power and airspeed are expressed in terms of variations of the non-dimensional pressure increase in the propeller slipstream *dpt*:

$$dpt = \frac{\Delta p_t}{\frac{1}{2}\rho V^2} = C_1 + C_2\left(\frac{P}{\frac{1}{2}\rho V^3}\right) \tag{3.14}$$

with $\frac{P}{\frac{1}{2}\rho V^3}$ measured in $[kW\,kg^{-1}\,s^3]$ and: $C_1 = 0.08696$, $C_2 = 191.18$, see ref.[26]. The engine power in $[Nm\,s^{-1}]$ can be calculated with the following expression:

$$P = 0.7355\left\{-326.5 + \left(0.00412\,(p_z + 7.4)(n + 2010) + (408.0 - 0.0965\,n)\left(1.0 - \frac{\rho}{\rho_0}\right)\right)\right\} \tag{3.15}$$

where:

p_z = manifold pressure $[''Hg]$,
n = engine speed $[RPM]$,
ρ = air-density $[kg\,m^{-3}]$,
ρ_0 = air-density at sea level = 1.225 $[kg\,m^{-3}]$.

The engine forces and moments, which include propeller slipstream effects, are written as polynomial functions of $\mathbf{x}$ and *dpt* in a similar way as the aerodynamic model, see ref.[26]:

$$\mathbf{F}_{prop} = \mathbf{d}\cdot\mathbf{p}_2\,(\mathbf{x}, dpt) \tag{3.16}$$

where the subscript *prop* denotes *propulsive* effects. The vector function $\mathbf{p}_2$ contains the polynomials for the non-dimensional propulsive force and moment coefficients; the actual forces and moments are again obtained by pre-multiplication with the vector $\mathbf{d}$, see equation (3.12). The polynomial functions gathered in $\mathbf{p}_2$, which describe the propulsive force and moment coefficients in the body-fixed reference frame are:

$$\begin{aligned} C_{X_p} &= C_{X_{dpt}}\, dpt + C_{X_{\alpha\, dpt^2}}\, \alpha\, dpt^2 \\ C_{Y_p} &= 0 \\ C_{Z_p} &= C_{Z_{dpt}}\, dpt \\ C_{l_p} &= C_{l_{\alpha^2 dpt}}\, \alpha^2 dpt \\ C_{m_p} &= C_{m_{dpt}}\, dpt \\ C_{n_p} &= C_{n_{dpt^3}}\, dpt^3 \end{aligned} \tag{3.17}$$

See table C.4 in appendix C for the values of the stability and control coefficients from these polynomial equations.

Gravity forces

The gravity force components along the aircraft's body axes equal:

$$\mathbf{F}_{grav} = \begin{bmatrix} X_{gr} \\ Y_{gr} \\ Z_{gr} \end{bmatrix} = W \cdot \begin{bmatrix} -\sin\theta \\ \cos\theta \sin\varphi \\ \cos\theta \cos\varphi \end{bmatrix} \tag{3.18}$$

where W is the aircraft weight, θ is the pitch angle, and φ is the roll angle of the vehicle. Obviously, the *attitude* of the vehicle with respect to the Earth must be known. The attitude is defined by three rotations from the Earth-fixed reference frame F_E to the body-fixed reference frame F_B.[1] Equation (3.18) reflects the result of the pre-multiplication of the gravity force equation in Earth-axes with the rotation matrix $T_{V\rightarrow B}$.

Forces and moments due to non-steady atmosphere

In section B.3 of appendix B it is shown that it is necessary to make corrections to the external force components along the aircraft's body-axes if the aircraft is flying through non-steady atmosphere. These corrections are equal to:

$$\mathbf{F}_{wind} = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = -m \cdot \begin{bmatrix} \dot{u}_w + q w_w - r v_w \\ \dot{v}_w - p w_w + r u_w \\ \dot{w}_w + p v_w - q u_w \end{bmatrix} \tag{3.19}$$

See appendix B for more details.

3.2.3 Converting implicit state equations to explicit equations

As explained in section 3.2.1, it is possible that the external forces and moments depend upon time-derivatives of state variables, which makes the general state equations implicit, see equation (3.8). An example of this is the dependency of the aerodynamic sideforce upon the time-derivative of the sideslip angle for the 'Beaver' model, shown earlier in the relation:

$$C_{Y_a} = C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{p\,b}{2V} + C_{Y_r}\frac{r b}{2V} + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r + C_{Y_{\delta_r \alpha}}\delta_r \alpha + C_{Y_{\dot{\beta}}}\frac{\dot{\beta} b}{2V} \tag{3.20}$$

[1] Strictly speaking, these rotations were from the vehicle-carried vertical reference frame F_V to the body-fixed reference frame F_B. See section A.7.2 for details.

Notice the last term in this equation! The ODE for the sideslip angle, which is derived in appendix B, is equal to:

$$\dot{\beta} = \frac{1}{Vm}\left(-F_x \cos\alpha \sin\beta + F_y \cos\beta - F_z \sin\alpha \sin\beta\right) + p\sin\alpha - r\cos\alpha \tag{3.21}$$

in which the aerodynamic contribution to the force component F_y is equal to $Y_a = \frac{1}{2}\rho V^2 S\, C_{Y_a}$. Due to this relation, the time-derivative $\dot{\beta}$ appears on both sides of equation (3.21). In theory it is possible to solve this equation numerically but that is not recommended since this would reduce the speed of the computations due to the fact that this yields a so-called *algebraic loop* in the simulation model (see section 4.2.7). In this case, it is easy to convert the $\dot{\beta}$-equation to an explicit equation. First of all, the contribution of $\dot{\beta}$ to the side-force F_y can be written as a separate term:

$$\dot{\beta} \;=\; \frac{1}{Vm}\left(-F_x \cos\alpha \sin\beta + {F_y}^* \cos\beta - F_z \sin\alpha \sin\beta + \tfrac{1}{2}\rho V^2 S\, C_{Y_{\dot{\beta}}} \tfrac{\dot{\beta} b}{2V} \cos\beta\right) + p\sin\alpha - r\cos\alpha \tag{3.22}$$

where ${F_y}^*$ is the side-force *without* the contribution of $\dot{\beta}$. The $\dot{\beta}$-term on the right hand side of this equation can easily be moved to the left-hand side:

$$\dot{\beta}^* \;\equiv\; \dot{\beta}\left(1 - \frac{\rho S b}{4m} C_{Y_{\dot{\beta}}} \cos\beta\right) = \frac{1}{Vm}\left(-F_x \cos\alpha \sin\beta + {F_y}^* \cos\beta - F_z \sin\alpha \sin\beta\right) + p\sin\alpha - r\cos\alpha \tag{3.23}$$

Based upon this equation the following calculation sequence can be used in the simulation model:

1. first compute the external forces and moments as usual, except for the $\dot{\beta}$-contribution to the aerodynamic side-force,
2. substitute the thus obtained forces and moments into the general $\dot{\beta}$ equation, yielding a value $\dot{\beta}^*$ instead of $\dot{\beta}$,
3. compute the true value of $\dot{\beta}$ with the expression $\dot{\beta} = \dot{\beta}^* \left(1 - \frac{\rho S b}{4m} C_{Y_{\dot{\beta}}}\right)^{-1}$.

The last step can be regarded as a correction factor for the originally computed value of $\dot{\beta}$, which was denoted here as $\dot{\beta}^*$. This correction term is aircraft-dependent because it contains the term $C_{Y_{\dot{\beta}}}$, but the equation for $\dot{\beta}^*$ does *not* depend upon the aircraft under consideration. This separation in aircraft-dependent and aircraft-independent terms is desirable to obtain a standardization of the aircraft models.

3.2.4 Atmosphere and airdata variables

In order to compute the aerodynamic and engine forces and moments from their non-dimensional counterparts the dynamic pressure needs to be known. For most aircraft it is necessary to take into account compressibility effects, which requires knowledge of the Mach number, and sometimes it may be necessary to take into account scale-effects which require knowledge of the Reynolds number. Other so-called *airdata* variables and airdata-related variables are needed to compare simulations with measurements in real flight or windtunnel experiments. For this reason, a number of airdata equations have been included in the aircraft model. The airdata variables depend upon atmospheric properties such as the air pressure, density, and temperature. Here we use the ICAO Standard Atmosphere model (see for instance ref.[23]) to determine these properties. According to this model, the air temperature T decreases linearly with increasing altitude in the troposphere (i.e. at altitudes from zero to 11,000 meters above sea level):

$$T = T_0 + \lambda h \tag{3.24}$$

where:

T = air temperature, $[K]$,
h = altitude above sea level, $[m]$,
T_0 = air temperature at sea level, $[K]$,
λ = temperature gradient in troposphere = $-0.0065\ Km^{-1}$.

The air pressure depends upon the altitude according to the basic hydrostatic equation:

$$dp_s = -\rho\, g\, dh \tag{3.25}$$

We assume that the ideal gas law can be applied to the air in the atmosphere:

$$\frac{p_s}{\rho} = \frac{R_a}{M_a}\, T \tag{3.26}$$

Combining these equations and neglecting the altitude-dependency of the gravitational acceleration g yields:

$$\frac{dp_s}{p_s} = -\frac{M_a\, g_0}{R_a T}\, dh \tag{3.27}$$

where:

p_s = air pressure, $[Nm^{-2}]$,
g_0 = gravitational acceleration at sea level = $9.80665\ ms^{-2}$,
M_a = molecular weight of the air, $[kg\ kmol^{-1}]$,
R_a = molar gas constant = $8314.32\ JK^{-1}kmol^{-1}$.

The static air pressure p_s is found after integrating equation (3.27), which yields:

$$\ln\left(\frac{p_s}{p_0}\right) = -\frac{g}{\lambda R}\, \ln\left(\frac{T_0 + \lambda h}{T_0}\right) \tag{3.28}$$

This equation can be written as:

$$\frac{p_s}{p_0} = \left(1 + \frac{\lambda h}{T_0}\right)^{-\frac{g}{\lambda R}} = \left(\frac{T_0}{T}\right)^{\frac{g}{\lambda R}} \tag{3.29}$$

where:

p_0 = air pressure at sea level = $101325\ Nm^{-2}$,
R = specific gas constant = $R_a/M_0 = 287.05\ JK^{-1}kg^{-1}$, with:
$M_0 = 28.9644\ kg\ kmol^{-1}$ = molecular weight of the air at sea level.

The gravitational acceleration g was held constant during this integration. This actually means that the geometrical altitude h in this equation must be replaced by the geopotential altitude H.[1] In this report the slight distinction between h and H will be neglected, in view of the relatively low altitudes considered. But from now on the symbol H will be used to denote the altitude (just a little reminder of this small inaccuracy). Contrary to the pressure equation (3.29), where the acceleration g was assumed to be equal to g_0 for all altitudes, the model does take into account changes in g with altitude for the computation of the aircraft's weight. The actual gravitational acceleration is then computed with the following equation:

$$g = g_0 \left(\frac{R_{Earth}}{R_{Earth} + h}\right)^2 \tag{3.30}$$

where:

R_{Earth} = radius of the Earth = $6371020\ m$

[1] The geopotential altitude H is defined as: $H \equiv \int_0^h \frac{g}{g_0} dh$

The air-density ρ (in $[kgm^{-3}]$) is calculated from p_s and T by means of the ideal gas law (3.26), which yields:

$$\rho = \frac{p_s M_a}{R_a T} = \frac{p_s}{RT} \tag{3.31}$$

The aerodynamic and propulsive forces and moments that act upon the aircraft are functions of the dynamic pressure q_{dyn}, which takes into account changes in airspeed and changes in air-density:

$$q_{dyn} = \tfrac{1}{2}\rho V^2 \tag{3.32}$$

For aircraft which fly at higher airspeeds than the 'Beaver' it is necessary to take into account the compressibility of the air, for which the Mach number M is needed. M is defined as:

$$M = \frac{V}{a} \tag{3.33}$$

where a is the speed of sound (in $[ms^{-1}]$):

$$a = \sqrt{\gamma RT} \tag{3.34}$$

For the 'Beaver' aircraft it is not necessary to take into account these compressibility effects, though it is still quite useful to know the value of M. For instance, if one wants to compare flight-test results with the simulations it is useful to compute the Mach-dependent impact pressure q_c and total temperature T_t since those quantities can be measured in flight. The impact pressure is equal to:

$$q_c = p_s \left\{ \left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\frac{\gamma}{\gamma - 1}} - 1 \right\} \tag{3.35}$$

where $\gamma = 1.4$ = ratio of the specific heats of air with constant pressure and air with constant volume, respectively. The total temperature T_t is equal to:

$$T_t = T \left(1 + \frac{\gamma - 1}{2} M^2 \right) \tag{3.36}$$

Other important variables are the *calibrated* and *equivalent* airspeeds V_c and V_e, which are computed with the equations:

$$V_c = \sqrt{\frac{2\gamma}{\gamma - 1} \frac{p_0}{\rho_0} \left\{ \left(1 + \frac{q_c}{p_0} \right)^{\frac{\gamma}{\gamma - 1}} - 1 \right\}} \tag{3.37}$$

$$V_e = V\sqrt{\frac{\rho}{\rho_0}} = \sqrt{\frac{2q_{dyn}}{\rho_0}} \tag{3.38}$$

Sometimes it is necessary to take into account scale effects, e.g. if the aerodynamic model is determined by windtunnel measurements on a scale model. In that case, the Reynolds number needs to be known. Often this is computed with respect to the mean aerodynamic chord $\overline{c}$, which yields the non-dimensional value:

$$R_c = \frac{\rho V \overline{c}}{\mu} \tag{3.39}$$

The Reynolds number per unit length (in $[m^{-1}]$) is equal to:

$$R_e = \frac{\rho V}{\mu} \tag{3.40}$$

In equations (3.39) and (3.40), μ is the coefficient of the dynamic viscosity, which can be calculated with the equation of Sutherland:

$$\mu = \frac{1.458 \cdot 10^{-6} T^{\frac{3}{2}}}{T + 110.4} \tag{3.41}$$

Summarizing: If we want to solve the equations of motion for the 'Beaver' aircraft, the variables p, T, ρ, and q_{dyn} *must* be calculated. All other airdata (-related) variables are useful for many analytical purposes and some of them are needed for solving the equations of motion if, contrary to the 'Beaver' model, compressibility or scale effects are taken into account.

3.2.5 Additional output variables

In the previous paragraph we have obtained a list of state variables, time-derivatives of the state variables, forces and moments, atmospheric variables, and airdata variables. It is possible to include a large number of additional output variables to this list. Here we will include additional normalized kinematic accelerations, specific forces, body-axes velocity rates and some flight-path (-related) variables. It is easy to enhance this list if required.

Kinematic accelerations and specific forces

It is possible to calculate a number of interesting accelerations and outputs from accelerometers, which are often important in the aircraft control analysis and design (e.g. for achieving appropriate turn-coordination by means of a control-loop that uses the acceleration along the Y_B axis as feedback signal, or for applications in the field of manoeuvre load limiting). The aircraft model from the FDC-toolbox considers accelerations in the vehicle's center of gravity only, but equations for positions outside the center of gravity can easily be included if necessary. See for instance ref.[9].

The body-axis acceleration vector $\mathbf{a}$ can be expressed as:

$$\mathbf{a} = \dot{\mathbf{V}} = \frac{\partial \mathbf{V}}{\partial t} + \boldsymbol{\Omega} \times \mathbf{V} \tag{3.42}$$

where $\boldsymbol{\Omega}$ is the rotational velocity vector of the aircraft. Expanding equation (3.42) into its components along the body-axes and substituting for $\dot{u}$, $\dot{v}$, and $\dot{w}$ (see equation (3.45)) yields:

$$\begin{aligned} a_{x,k} &= \frac{1}{g_0}(\dot{u} + qw - rv) = \frac{F_x}{W} \\ a_{y,k} &= \frac{1}{g_0}(\dot{v} + ru - pw) = \frac{F_y}{W} \\ a_{z,k} &= \frac{1}{g_0}(\dot{w} + pv - qu) = \frac{F_z}{W} \end{aligned} \tag{3.43}$$

The accelerations are measured in units of g which explains the division by g_0. $W = mg$ is the total weight of the aircraft, measured in $[N]$. The index k is used to denote that these variables represent *kinematic* accelerations in the body-fixed reference frame.

The outputs of accelerometers along the body-axes at the vehicle's center of gravity – usually called *specific forces* – are equal to the kinematic body accelerations minus the gravity terms:

$$\begin{aligned} A_x &= a_{x,k} + \sin\theta &&= (F_x - X_{gr}) / W \\ A_y &= a_{y,k} - \cos\theta \sin\varphi &&= (F_y - Y_{gr}) / W \\ A_z &= a_{z,k} + \cos\theta \cos\varphi &&= (F_z - Z_{gr}) / W \end{aligned} \tag{3.44}$$

A_x, A_y, and A_z are measured in units of g. These accelerations represent what is actually felt in the aircraft's center of gravity.

Body-axes velocity rates

For educational purposes, it may be useful to take a closer look at the body-axes components of the accelerations of the aircraft. The body-axes velocity rates $\dot{u}$, $\dot{v}$, and $\dot{w}$ are equal to:

$$\begin{aligned} \dot{u} &= \frac{F_x}{m} - qw + rv \\ \dot{v} &= \frac{F_y}{m} + pw - ru \\ \dot{w} &= \frac{F_z}{m} - pv + qu \end{aligned} \tag{3.45}$$

Note that $\dot{u}$, $\dot{v}$, and $\dot{w}$ differ from the kinematic accelerations from equation (3.42) in that they do not contain the angular and translational velocity cross-product terms. (Also notice that $\dot{u}$, $\dot{v}$, and $\dot{w}$ have not been expressed in $[g]$.) The body-axes velocity components u, v, and w are no state variables due to the fact that it was more convenient to use the true airspeed V, angle of attack α, and sideslip angle β instead; see section B.2 of appendix B. For this reason, equations (3.45) have been implemented as additional *output* equations in the FDC models.

Flight-path variables

Some additional flight-path (-related) variables have been included to the aircraft model from the FDC toolbox. First of all, the flight-path angle γ is computed, using the following expression:

$$\gamma = \arcsin\left(\frac{\dot{H}}{V}\right) \tag{3.46}$$

This angle is for instance useful during approach simulations where it determines how much the aircraft deviates from the standard glide-path. The acceleration in the direction of the velocity vector $\mathbf{V}$, measured in units of g, is called the flight-path acceleration *fpa*. It is equal to:

$$fpa = \frac{\sqrt{\dot{u}^2 + \dot{v}^2 + \dot{w}^2}}{g_0} = \frac{\dot{V}}{g_0} \tag{3.47}$$

Other flight-path related variables are the azimuth angle χ and the bank angle Φ, which are obtained with the following equations:

$$\chi = \beta + \psi \tag{3.48}$$

$$\Phi = \varphi \cdot \cos(\theta - \alpha_0) \tag{3.49}$$

with α_0 the value of the angle of attack in a steady-state flight condition (ref.[23]). See also the description of the *flight-path* or *wind* reference frame F_W in appendix A, section A.7.2.

3.3 External atmospheric disturbances

3.3.1 Deterministic disturbances

The velocity and direction of the mean wind with respect to the ground usually is not constant along the flight-path. This variation of the *mean* wind along the flight-path is called wind shear.[1] The influence of wind shear upon the motions of the aircraft is of particular importance during the final approach and landing, and take-off and initial climb. An idealized profile of the mean wind as a function of altitude is shown in figure 3.3. More extreme wind profiles in lower atmosphere have been measured (see for instance ref.[1]) and have sometimes resulted in serious

[1] Some textbooks denote the local variations of the wind with respect to the ground *including atmospheric turbulence* as wind shear. In this report the turbulence is considered separately.

accidents. A very serious type of wind shear is encountered in so-called *microbursts*, where large nose winds are followed by large tail winds in just a couple of seconds.

In this report we only use the ICAO standard atmosphere (see for instance ref.[23]) which has a standard temperature lapse-rate $\lambda = dT/dH = -0.0065\ Km^{-1}$. For this lapse-rate a typical idealized wind profile can be represented by the following expression (see ref.[1]):

$$\begin{aligned} V_w &= V_{w\,9.15}\frac{H^{0.2545} - 0.4097}{1.3470} \qquad (0 < h < 300m) \\ V_w &= 2.86585\, V_{w_{9.15}} \qquad (h \geq 300m) \end{aligned} \tag{3.50}$$

$V_{w\,9.15}$ is the wind speed at 9.15 m altitude. The wind profile in figure 3.3 is based upon a value $V_{w\,9.15} = 1\ ms^{-1}$. Wind profiles which are typical for other values of the temperature lapse rate are presented in ref.[1]. This model of the wind speeds in the boundary layer of the Earth is not adequate for the more extreme wind-profiles that can occur in practice in the lower atmosphere. Hence, actual measurements of extreme wind profiles may have to be used for the assessment by simulations of Automatic Flight Control Systems.

The aircraft equations of motion – derived in appendix B – use wind velocities along the aircraft's body-axes to determine the influence upon the motions of the aircraft. If we assume that the vertical wind velocity component w_w is zero, we have the situation shown in figure 3.4. In this figure, the wind direction with respect to the Earth-fixed reference frame is denoted by ψ_w and the total wind velocity by V_w. Note that the wind direction is the direction from where the wind is blowing, so ψ_w is zero if the wind is blowing *from* the North. The components along the X_B and Y_B-axes are now equal to:

$$\begin{aligned} u_w &= \quad V_w \cos(\psi_w - \pi)\cos\psi + V_w \sin(\psi_w - \pi)\sin\psi \\ v_w &= -V_w \cos(\psi_w - \pi)\sin\psi + V_w \sin(\psi_w - \pi)\cos\psi \end{aligned} \tag{3.51}$$

where ψ is the heading of the aircraft and all angles have been measured in $[rad]$. If we consider atmospheric turbulence too it is necessary to add the turbulence velocity components to these wind components.

3.3.2 Stochastic disturbances

The theory of stochastic processes provides a convenient means for describing atmospheric turbulence accurately enough for most simulations, e.g. for the assessment of AFCS control laws. Auto power density spectra form the basic elements of the turbulence model. In the literature, several sets of these spectra can be found. They all require the selection of intensity levels and scale lengths before they can be applied in simulations. The following six assumptions concerning stochastic processes are usually made when they are applied to atmospheric turbulence (ref.[1]):

1. Ergodicity, which means that time averages in the process are equal to corresponding ensemble averages. This assumption makes it possible to determine all required statistical properties related to a given set of atmospheric conditions from a single time history of sufficient length.
2. Stationarity, which deals with temporal properties of turbulence. If the statistical properties of a process are not affected by a shift in the time origin, this process is called stationary.
3. Homogeneity, which deals with spatial properties of turbulence. Turbulence may be called homogeneous if its statistical properties are not affected by a spatial translation of the reference frame.

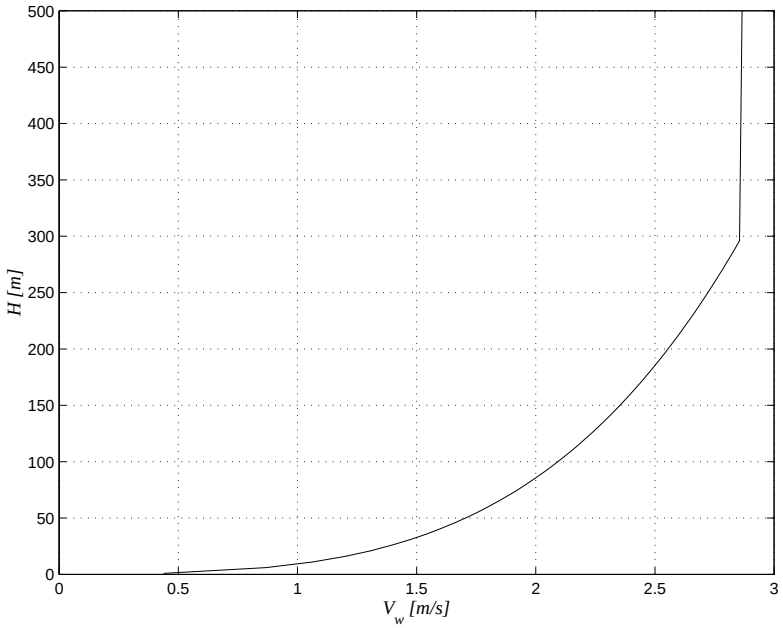


Figure 3.3: Wind profile for $\lambda = -0.0065\ [Km^{-1}]$ and $V_{w\,9.15} = 1\ [ms^{-1}]$

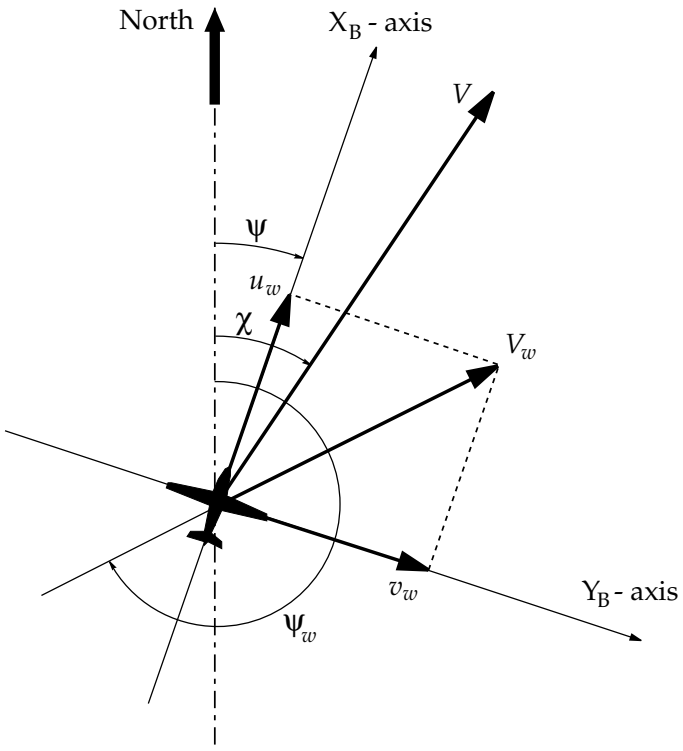


Figure 3.4: Wind velocity components along the aircraft's body-axes

4. Isotropy, which means that the statistical properties are not changed by a rotation or a deflection of the frame of reference. Complete isotropy implies homogeneity. Because of isotropy, the three mean-square velocity components and their scale lengths are equal:

$$\begin{aligned} {\sigma_u}^2 &= {\sigma_v}^2 = {\sigma_w}^2 \equiv \sigma^2 \\ L_u &= L_v = L_w \equiv L_g \end{aligned} \tag{3.52}$$

5. Taylor's hypothesis of 'frozen atmosphere', which implies that gust velocities are functions of the position in the atmosphere only. During the short time-interval in which the aircraft is under the influence of the velocities at a certain point in the atmosphere, these velocities are assumed not to change with time. This hypothesis allows spatial correlation functions and frequencies to be related to correlation functions and frequencies in the time-domain. The following relations are used:

$$\Delta x = V\tau \tag{3.53}$$

$$\omega = \Omega V \tag{3.54}$$

Δx = distance between to points in space; $[m]$
V = true airspeed of the aircraft; $[ms^{-1}]$
τ = time needed by the aircraft to cover the distance Δx; $[s]$
Ω = spatial frequency; $[rad\ m^{-1}]$
ω = temporal frequency; $[rad\ s^{-1}]$

6. Normality, which means that the probability density function of each turbulence velocity component is Gaussian. With this assumption, the information of the covariance matrix *only* is sufficient for a total statistical description of atmospheric turbulence (ref.[21]).

Experimental data on atmospheric turbulence at low altitudes, i.e. in the boundary layer of the Earth, do not satisfy all these assumptions (ref.[1]). At low altitudes the assumptions of homogeneity and isotropy are not very valid due to the proximity of the ground. Both assumptions are affected by terrain roughness and the height above the ground. The assumption of stationarity is satisfied only over the short periods of time during which the meteorological conditions remain reasonably constant. Stationarity is also affected by the shape and roughness of the ground surface below the aircraft. Taylor's hypothesis seems to be valid as long as the aircraft's velocity is large relative to the encountered turbulence velocity. For this reason it is somewhat doubtful that the hypothesis is fully valid when simulating the final approach and landing of S/VTOL aircraft.[1] Finally, measurements have provided evidence that atmospheric turbulence is not perfectly Gaussian. The measured departures from a normal amplitude distribution are small, but pilots seem to be quite sensitive to these effects. Actual atmospheric turbulence possesses what is sometimes called a 'patchy' structure (ref.[1]).

In the models used for the FDC toolbox, assumptions 1 to 6 have all been maintained. These turbulence may have to be enhanced in the future to assure a more accurate description of actual atmospheric turbulence, particularly for the simulation of final approach and landing. Of course it is also possible to insert measurements of actual turbulence as input signals to the FDC models if a very high accuracy is required.

Power spectra of atmospheric turbulence

Several analytical power spectral density functions have been obtained from measured data. The von Kármán spectral density functions seem to best fit the available theoretical and experimental

[1]STOL = short take-off and landing, VTOL = vertical take-off and landing. Aircraft from these categories are able to fly with very low airspeeds.

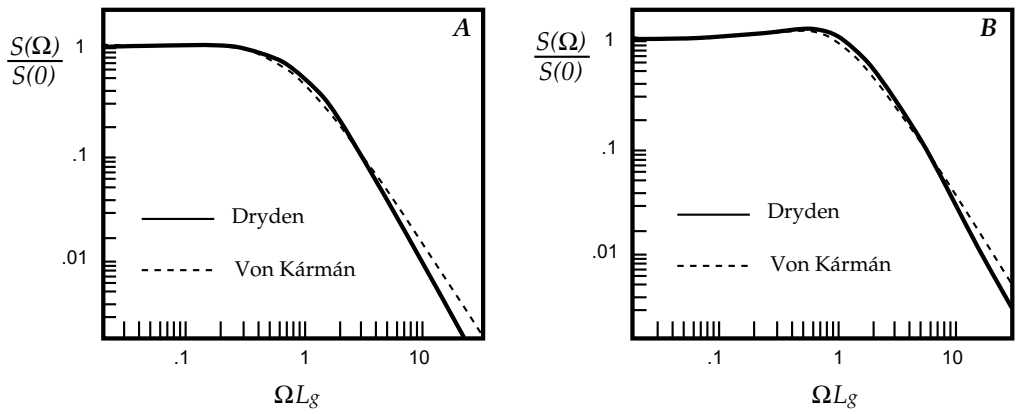


Figure 3.5: Von Kármán and Dryden spectra (A: longitudinal, B: lateral/vertical)

data on atmospheric turbulence, particularly at higher spatial frequencies (ref.[21]). The von Kármán spectra for the three components of the turbulence velocity are:

$$S_{u_g u_g}(\Omega) = 2{\sigma_u}^2 L_u \frac{1}{(1+(1.339\,L_u\Omega)^2)^{\frac{5}{6}}} \tag{3.55}$$

$$S_{v_g v_g}(\Omega) = {\sigma_v}^2 L_v \frac{1+\frac{8}{3}(1.339 L_v\Omega)^2}{(1+(1.339\,L_v\Omega)^2)^{\frac{11}{6}}} \tag{3.56}$$

$$S_{w_g w_g}(\Omega) = {\sigma_w}^2 L_w \frac{1+\frac{8}{3}(1.339 L_w\Omega)^2}{(1+(1.339\,L_w\Omega)^2)^{\frac{11}{6}}} \tag{3.57}$$

The cross spectral density functions are zero in isotropic turbulence at any point in space. Although this approximation is not very valid at low altitudes, the cross covariances – and hence, the cross power spectral densities – are usually neglected (ref.[14]). The von Kármán spectra yield an asymptotic behavior of $S(\Omega) \sim \Omega^{-5/3}$ as Ω approaches infinity. See figure 3.5.

A major drawback of the von Kármán spectral densities is that they are not rational functions of Ω. For this reason the following power spectral density model is often used for flight simulation purposes:

$$S_{u_g u_g}(\Omega) = 2{\sigma_u}^2 L_u \frac{1}{1+(\Omega L_u)^2} \tag{3.58}$$

$$S_{v_g v_g}(\Omega) = {\sigma_v}^2 L_v \frac{1+3(\Omega L_v)^2}{(1+(\Omega L_v)^2)^2} \tag{3.59}$$

$$S_{w_g w_g}(\Omega) = {\sigma_w}^2 L_w \frac{1+3(\Omega L_w)^2}{(1+(\Omega L_w)^2)^2} \tag{3.60}$$

These functions are the Dryden spectra. In figure 3.5 the Dryden spectra have been compared with the von Kármán spectra. The most obvious difference is the asymptotic behavior at large values of the spatial frequency, the former having a slope of $-\frac{5}{3}$ and the latter a slope of -2.

Filter design for atmospheric turbulence

For simulation purposes it would be practical to model atmospheric turbulence as white noise passing through a linear, rational 'forming filter', as shown in figure 3.6. The relationship

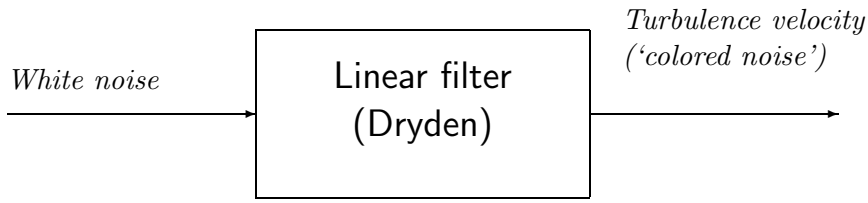


Figure 3.6: Modeling atmospheric turbulence as filtered white noise

between the auto-spectral density of the output signal y and the auto-spectral density of the input signal u of a *linear* filter can be written as:

$$S_{yy}(\omega) = |H_{yu}(\omega)|^2 S_{uu}(\omega) \tag{3.61}$$

where $|H_{yu}(\omega)|$ denotes the amplitude response of the filter. If the input signal u is white noise, its spectral density satisfies:

$$S_{uu}(\omega) = 1 \tag{3.62}$$

so for white noise, relation (3.61) simplifies to:

$$S_{yy}(\omega) = |H_{yu}(\omega)|^2 \tag{3.63}$$

To apply these relations, the spatial spectral density functions of the turbulence velocities must be transformed to functions of ω, which is possible because we assume Taylor's hypothesis to be valid. This transformation is given by:

$$S(\omega) = \frac{1}{V} S\left(\Omega = \frac{\omega}{V}\right) \tag{3.64}$$

Notice the term $1/V$ that arises in the spectral density function. See ref.[21] for more details.

The Dryden spectra were developed to approximate the von Kármán turbulence spectra by means of rational functions. This makes it possible to apply relation (3.64) for the generation of turbulence velocity components from white noise inputs. From the definitions of the Dryden spectra in equations (3.58) to (3.60) and relation (3.64) the following expressions are found:

$$|H_{u_g w_1}(\omega)|^2 = {\sigma_u}^2 \frac{L_u}{V} \frac{1}{1 + \left(L_u \frac{\omega}{V}\right)^2} \tag{3.65}$$

$$|H_{v_g w_2}(\omega)|^2 = {\sigma_v}^2 \frac{L_v}{V} \frac{1 + 3\left(L_v \frac{\omega}{V}\right)^2}{1 + \left(L_v \frac{\omega}{V}\right)^2} \tag{3.66}$$

$$|H_{w_g w_3}(\omega)|^2 = {\sigma_w}^2 \frac{L_w}{V} \frac{1 + 3\left(L_w \frac{\omega}{V}\right)^2}{1 + \left(L_w \frac{\omega}{V}\right)^2} \tag{3.67}$$

Solving equations (3.65) to (3.67) yields the following candidate functions for the frequency responses of the forming filters:

$$H_{u_g w_1}(\omega) = \sigma_u \sqrt{\frac{2L_u}{V}} \frac{1}{1 \pm \frac{L_u}{V} j\omega} \tag{3.68}$$

$$H_{v_g w_2}(\omega) = \sigma_v \sqrt{\frac{L_v}{V}} \frac{1 \pm \sqrt{3}\frac{L_v}{V} j\omega}{\left(1 \pm \frac{L_v}{V} j\omega\right)^2} \tag{3.69}$$

$$H_{w_g w_3}(\omega) \quad = \quad \sigma_w \sqrt{\frac{L_w}{V}} \frac{1 \pm \sqrt{3}\,\frac{L_w}{V} j\omega}{\left(1 \pm \frac{L_w}{V} j\omega\right)^2} \tag{3.70}$$

In these equations w_1, w_2, and w_3 are independent white noise signals. Choosing the minus sign in the denominators would lead to unstable filters and hence should be rejected for physical reasons. Choosing the minus sign in the numerators leads to non-minimum phase systems (ref.[21]). Therefore we shall use positive signs in both the numerator and denominator.

It is easy to implement these filters in a simulation package like SIMULINK. If white noise is approximated by a sequence of Gaussian distributed random numbers, it is then very easy to obtain the required time-trajectories of the turbulence velocity components. These random sequences should be completely independent, which may not be obvious if the simulation software uses some initial starting value or 'seed' for its random-generator. It is possible to implement better approximations of the von Kármán turbulence spectra by using a series of additional lead-lag networks for adding differentiating and integrating terms to the basic first-order Dryden-filters. A full derivation of these approximations can be found in ref.[1]. In the FDC toolbox only linear filters for the Dryden spectra have been used.

3.4 Radio-navigation models

3.4.1 The Instrument Landing System

Nominal ILS signals

The Instrument Landing System (ILS) is the standard aid for non-visual approaches to landing, in use throughout the world today. Under certain circumstances it can provide guidance data of such integrity that fully coupled approaches and landings may be achieved. The system comprises three distinct parts of on-ground equipment of which only the localizer and glideslope signals will be considered in the FDC models:

1. the *localizer* transmitter, which gives guidance in the horizontal plane,
2. the *glideslope* (or glide-path) transmitter, which provides vertical guidance,
3. two or three *marker beacons*, situated on the approach line, which give an indication of the distance from the approaching aircraft to the runway.

Figures 3.7 and 3.8 show the lay-out of the ILS ground equipment and the approach path. The localizer signal is emitted by an antenna, situated beyond the up-wind end of the runway. Operating at a frequency within the 108.0 to 112 *MHz* frequency band it radiates a signal modulated by 90 and 150 *Hz* tones, in which the 90 *Hz* predominates to the left hand side of the approach path and 150 *Hz* predominates to the right, as seen from an aircraft on final approach. Figure 3.9 shows the required minimum coverage of the localizer signals according to ICAO (ref.[2]).

The glideslope antenna is located some 300 meters beyond the runway threshold (approximately adjacent to the touch-down point) and at about 120 to 150 m from the runway centerline. The frequency of the glideslope signal lies within the 328.6 to 335.0 *MHz* band. The signal is modulated by 90 and 150 *MHz* tones, in which the 90 *Hz* is predominant *above* the desired glide-path and 150 *Hz beneath* the glide-path. Due to the position of the glideslope antenna, the intersection of the localizer reference plane and the glideslope reference cone is actually a *hyperbola* which is located a small distance above the idealized straight glide-path. This is shown in figure 3.10. The minimum coverage of the glideslope signals, required by ICAO (ref.[2]), is shown in figure 3.11.

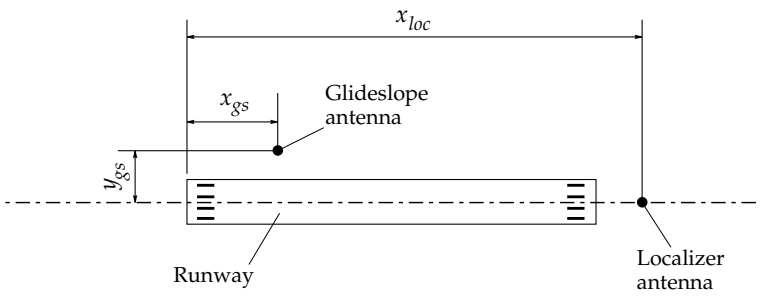


Figure 3.7: Positioning of ILS ground equipment

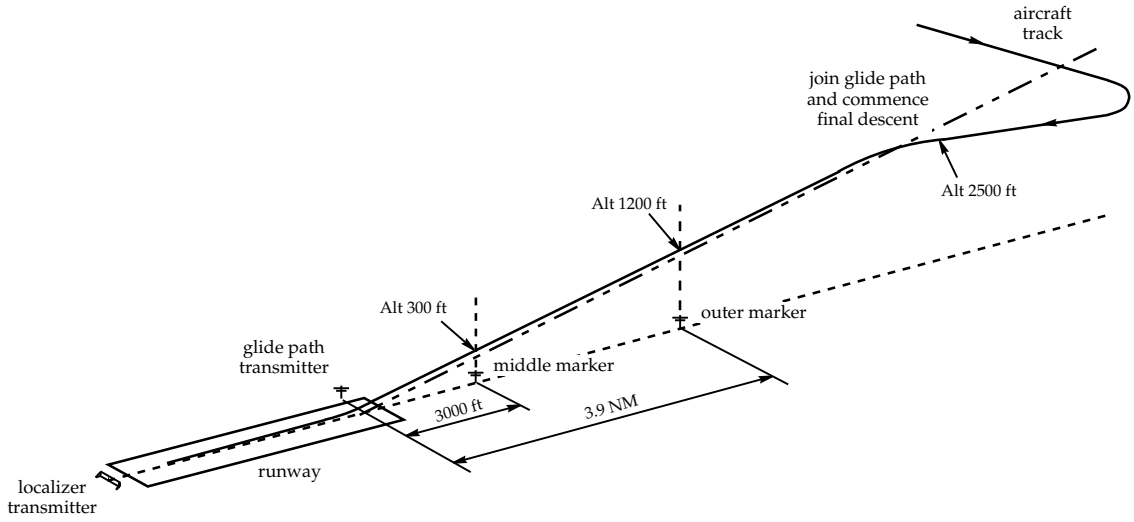


Figure 3.8: Lay-out of the approach path

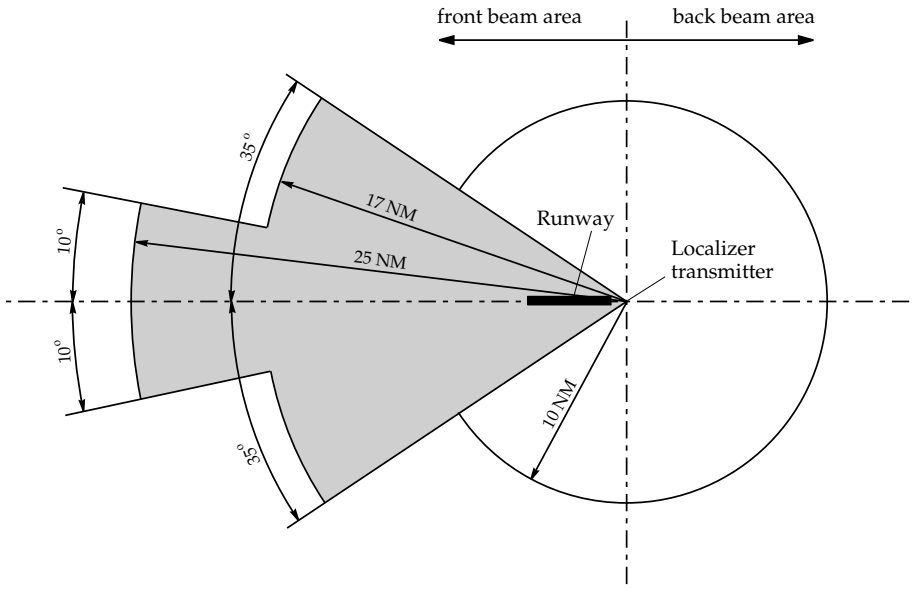


Figure 3.9: Required coverage of localizer signal

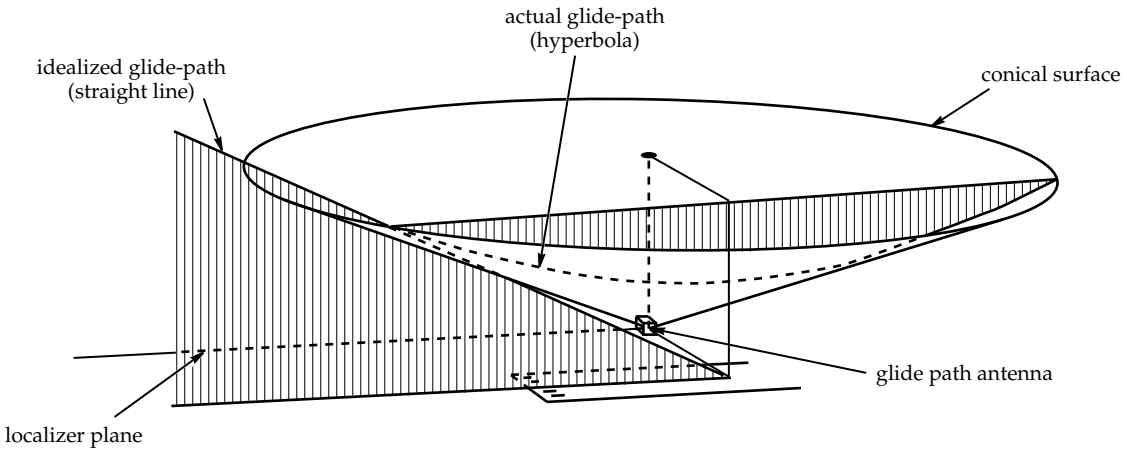


Figure 3.10: Hyperbolic intersection of localizer and glideslope reference planes

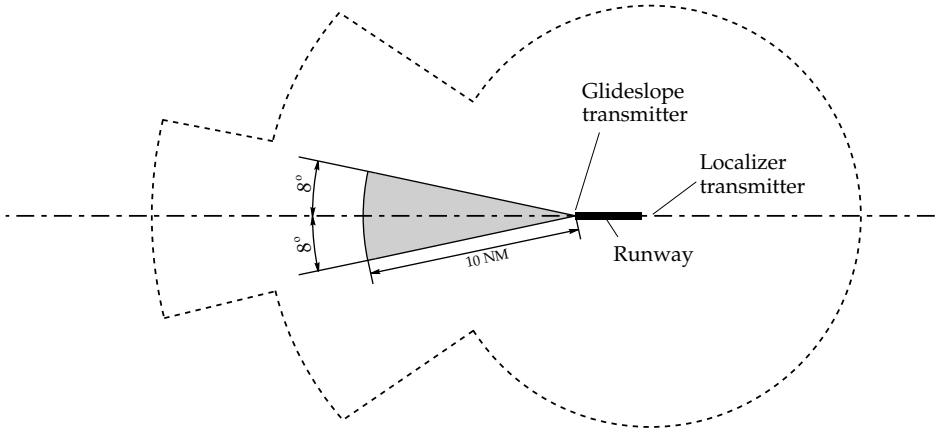
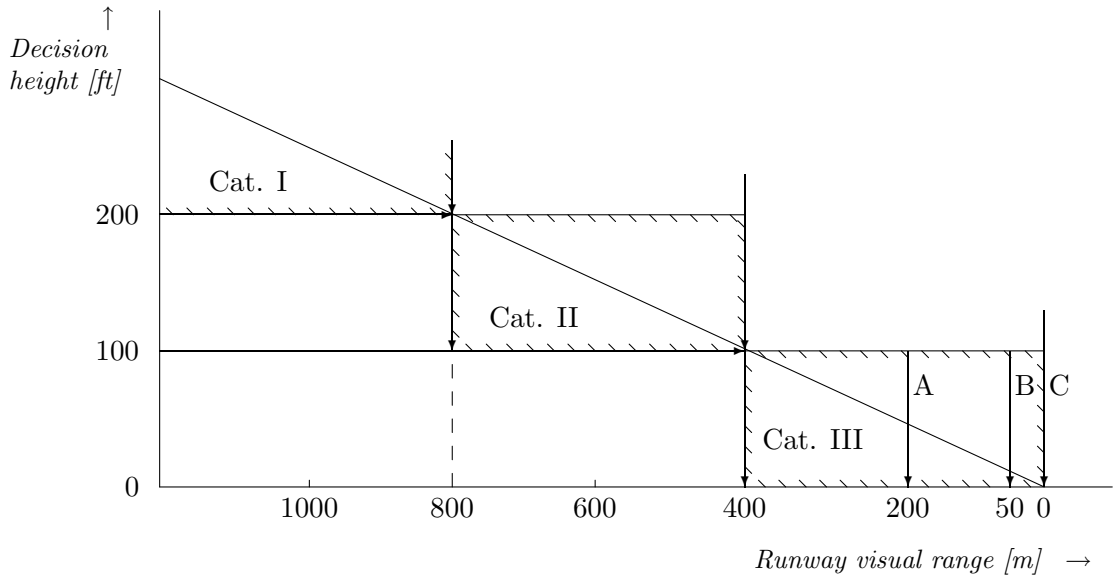


Figure 3.11: Required coverage of glideslope signal compared to localizer coverage



Cat. I : Operation down to minima of 200 *ft* decision height and runway visual range of 800 *m* with a high probability of approach success

Cat. II : Operation down to minima below 200 *ft* decision height and runway visual range of 800 *m*, and as low as 100 *ft* decision height and runway visual range of 400 *m* with a high probability of approach success

Cat. III A: Operation down to and along the surface of the runway, with external visual reference during the final phase of the landing down to runway visual range minima of 200 *m*

Cat. III B: Operation to and along the surface of the runway and taxiways with visibility sufficient only for visual taxiing comparable to runway visual range in the order of 50 *m*

Cat. III C: Operation to and along the surface of the runway and taxiways without external visual reference

Figure 3.12: ILS performance categories

An ILS installation is said to belong to a certain performance category, representing the meteorological conditions under which it is to be used. These conditions are summarized in figure 3.12. An ILS installation of category I is intended to provide guidance down to an altitude of 200 ft, a category II installation provides guidance down to 100 ft, and an installation of category III must provide guidance down to the runway surface. Only cat. III signals can be used for fully automatic landings. If the aircraft is making an approach under cat. I conditions, the pilot should either see the runway lights at an altitude of 200 ft or cancel the final approach and go-around.[1]

The localizer and glideslope signals are received on board the aircraft. They are displayed in an appropriate form to the pilot and may be fed directly to an automatic pilot as well. The nominal ILS signals on board the aircraft are expressed in terms of the currents, supplied to the pilot's cockpit instrument. The magnitude of the localizer current i_{loc} depends upon the angle Γ_{loc} (measured in $[rad]$) between the localizer reference plane and a vertical plane that passes through the localizer antenna, as depicted in figure 3.14.

[1]The altitude at which the runway lights should be visible is called the *decision height*. Some airlines use larger decision height values than figure 3.12.

The localizer current is:

$$i_{loc} = S_{loc}\,\Gamma_{loc} \qquad [\mu A] \tag{3.71}$$

where S_{loc} is the sensitivity of the localizer system. According to ref.[2], S_{loc} has to satisfy the following equation:

$$S_{loc} = 1.40\,x_{loc} \qquad [\mu A\ rad^{-1}] \tag{3.72}$$

where x_{loc} is the distance from the localizer antenna to the runway threshold (measured in $[m]$), see figure 3.8.

For the simulation of ILS-approaches, a *runway-fixed* reference frame $F_F = O_F X_F Y_F Z_F$ will be introduced. The X_F-axis is directed along the runway centerline in the direction of take-off and landing. Z_F points downwards and Y_F points rightwards as seen from an aircraft on final approach. At time $t = 0$ the position of the aircraft's c.g. coincides with the origin of the Earth-fixed reference frame, hence: $x_e = 0$, $y_e = 0$, and $H = H_0$. The position of the origin O_F of the runway reference frame at $t = 0$ is given by the coordinates x_{RW} and y_{RW}, measured relatively to the Earth-fixed reference frame, and the altitude of the runway above sea level, H_{RW}. From figures 3.13 and 3.14, the following transformations from the coordinates x_e and y_e referenced to the Earth-fixed reference frame F_E to the coordinates x_f and y_f referenced to F_F can be deduced:

$$x_f = (x_e - x_{RW})\cos\psi_{RW} + (y_e - y_{RW})\sin\psi_{RW} \tag{3.73}$$

$$y_f = -(x_e - x_{RW})\sin\psi_{RW} + (y_e - y_{RW})\cos\psi_{RW} \tag{3.74}$$

where ψ_{RW} is the heading of the runway, measured relatively to the North. The height of the aircraft above aerodrome level, H_f, is equal to:

$$H_f = H - H_{RW} \tag{3.75}$$

H is the altitude of the aircraft above sea level. (Note: the reference frame $F'_E = O'_E X'_E Y'_E Z'_E$ in figure 3.13 is an intermediate frame of reference, which has the same orientation as F_E, but an origin that has been moved to the projection point of O_F on the horizontal plane at sea level. Hence: $x'_e = x_e - x_{RW}$ and $y'_e = y_e - y_{RW}$; see also figure 3.14.)

As can be seen from figure 3.14, Γ_{loc} can be computed from the coordinates x_f and y_f as follows:

$$\left.\begin{aligned} R_{loc} &= \sqrt{{y_f}^2 + (x_{loc} - x_f)^2} \\ d_{loc} &= y_f \end{aligned}\right\} \quad \Gamma_{loc} = \arcsin\left(\frac{d_{loc}}{R_{loc}}\right) \tag{3.76}$$

Γ_{loc} and d_{loc} are positive if the aircraft flies at the right hand side of the localizer reference plane heading towards the runway. The locations which provide a constant *glideslope* current lie on a cone, as shown in figures 3.10 and 3.15. The nominal glide path has an elevation angle γ_{gs} which normally has a value between -2° and -4°. Obviously γ_{gs} is negative since the aircraft *descents* along the glide path. The magnitude of the glideslope current is proportional to the glideslope error angle ε_{gs} $[rad]$ (see figure 3.15):

$$i_{gs} = S_{gs}\,\varepsilon_{gs} \qquad [\mu A] \tag{3.77}$$

where S_{gs} is the sensitivity of the glideslope system, which according to ref.[2] equals:

$$S_{gs} = \frac{625}{|\gamma_{gs}|} \qquad [\mu A\,rad^{-1}] \tag{3.78}$$

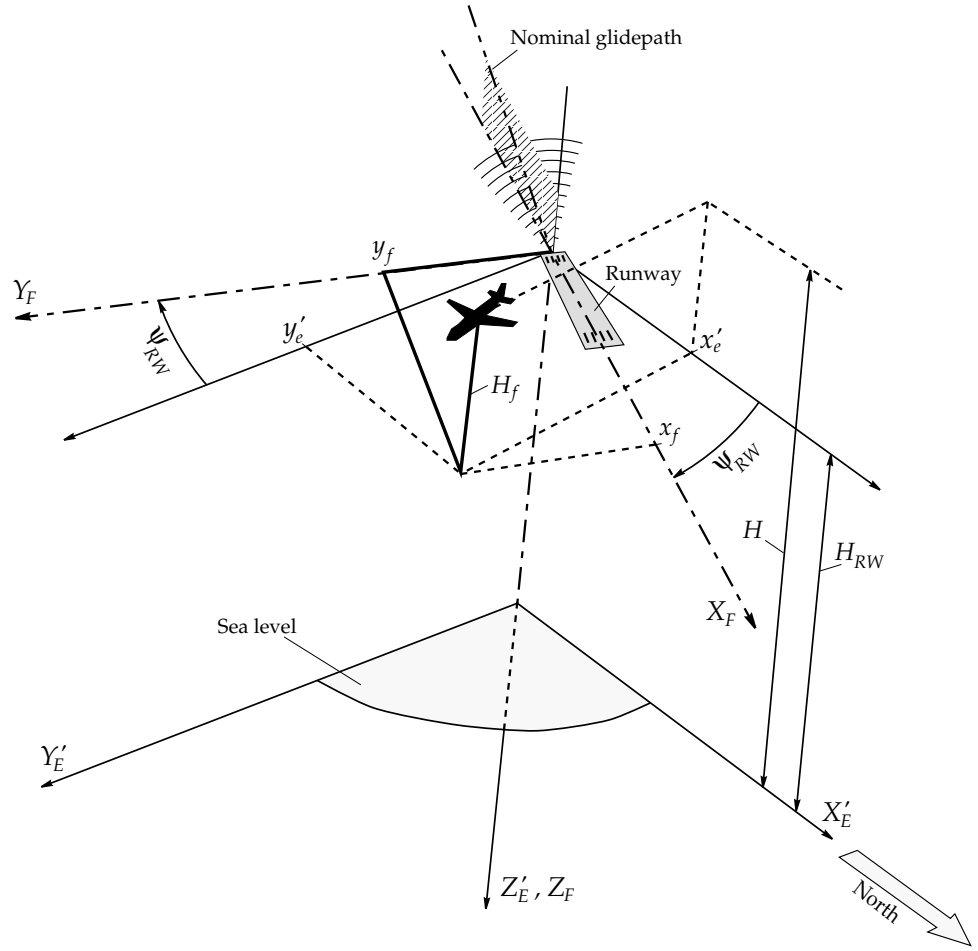


Figure 3.13: Definition of Earth-fixed axes $X'_E Y'_E Z'_E$ and runway axes $X_F Y_F Z_F$ (the aircraft is turning right after a missed approach)

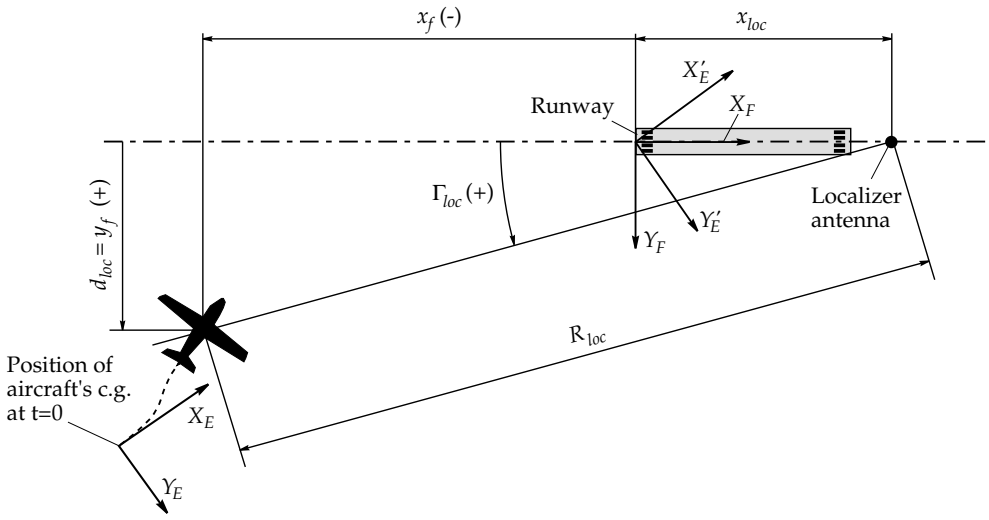


Figure 3.14: Localizer geometry and definition of X'_E, Y'_E, X_F, and Y_F-axes

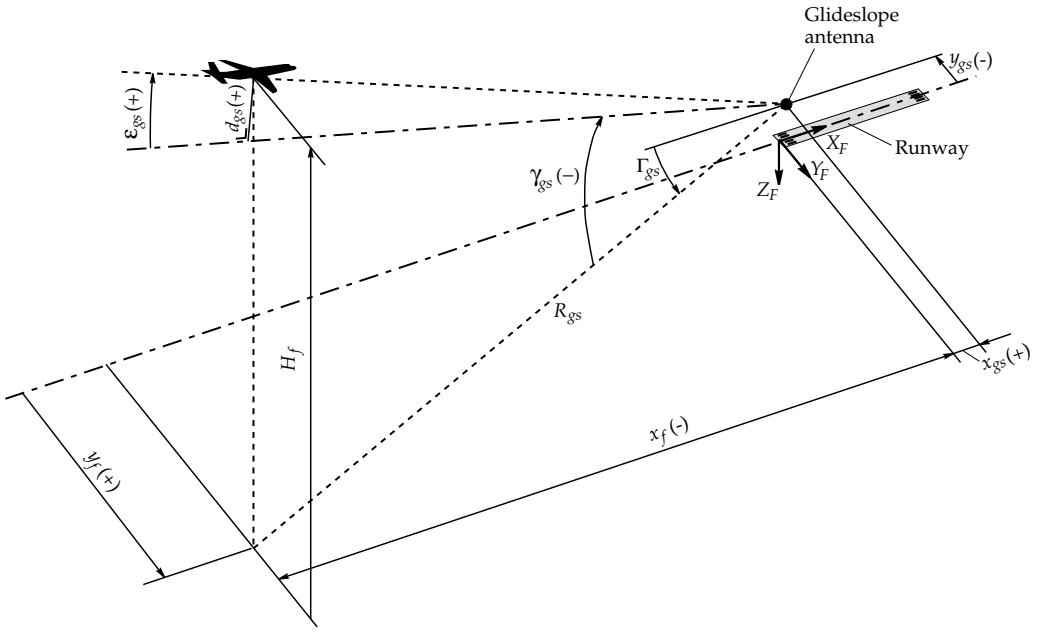


Figure 3.15: Glideslope geometry

From figure 3.15 it can be seen that the error angle ε_{gs} can be computed from the coordinates x_f and y_f and the height above the runway, H_f, with the following expressions:

$$R_{gs} = \sqrt{(x_{gs} - x_f)^2 + (y_f - y_{gs})^2} \tag{3.79}$$

$$\varepsilon_{gs} = \gamma_{gs} + \arctan\left(\frac{H_f}{R_{gs}}\right) \tag{3.80}$$

The distance from the aircraft to the nominal glideslope is:

$$d_{gs} = (R_{gs} \tan\gamma_{gs} + H_f)\cos\gamma_{gs} \tag{3.81}$$

In these expressions ε_{gs} and d_{gs} are positive if the aircraft flies *above* the glideslope reference line. Notice that γ_{gs} is always negative! In order to check if the aircraft flies in the area where the glideslope signals can be received (figure 3.11), the angle Γ_{gs} is calculated too. Due to the position of the glideslope antenna this angle is not exactly equal to Γ_{loc}, although the differences are small. Computing Γ_{gs} is straightforward:

$$\Gamma_{gs} = \arcsin\left(\frac{y_f - y_{gs}}{R_{gs}}\right) \tag{3.82}$$

Steady-state ILS offset errors

ICAO has established limits for ILS steady-state offset errors introduced by the ground equipment, see ref.[2]. These limits are of course most stringent for cat. III approaches. Tables 3.1 and 3.2 give these limits for localizer and glideslope transmitters, respectively. The nominal glide path must pass over the runway threshold at an altitude of $15 \pm 3\ m$. The maximum values of the localizer current i_{loc} and the glideslope current i_{gs} are limited to $\pm 150\ \mu A$, hence $\pm 150\ \mu A$ represents a full-scale deflection on the cockpit instrument (ref.[2]).

Performance category of ILS system	Maximum deviation from nominal localizer sensitivity [%]	Maximum deviation of localizer runway reference plane from centerline at runway threshold [m]
I	$\pm$ 17	$\pm$ 10.5
II	$\pm$ 17 ($\pm$ 10 where practicable)	$\pm$ 7.5 ($\pm$ 4.5 for new installations)
III	$\pm$ 10	$\pm$ 3

Table 3.1: Maximum permissible localizer steady-state errors

Performance category of ILS system	Maximum deviation from nominal glideslope sensitivity [%]	Maximum deviation of nominal glideslope elevation angle
I	$\pm$ 25	$\pm$ 0.075 γ_{gs}
II	$\pm$ 20	$\pm$ 0.075 γ_{gs}
III	$\pm$ 10	$\pm$ 0.04 γ_{gs}

Table 3.2: Maximum permissible glideslope steady-state errors

ILS noise characteristics

Due to interference effects caused by buildings, high voltage cables, etc., the actual ILS signals become distorted in the spatial and time domains. To an approaching aircraft, these distortions appear as noise in the time-domain, superimposed on the nominal ILS signals. Based on available experimental data, localizer and glideslope noise may be approximated by stochastic signals which have rather simple power spectral density functions.

Refs.[1] and [14] present power spectra for ILS noise, which are expressed in the same general form as the Dryden model for longitudinal atmospheric turbulence, see equation (3.58). The power spectral density function for localizer noise can be approximated by:[1]

$$S_{loc}(\Omega) = 2{\sigma_{loc}}^2 L_{loc} \frac{1}{1 + (\Omega L_{loc})^2} \qquad \left[\mu A^2 rad^{-1} m\right] \tag{3.83}$$

where:

σ_{loc} = standard deviation of the localizer noise,
L_{loc} = 'scale' of the localizer noise, approximately 130 m
Ω = spatial frequency $[rad\, m^{-1}]$

[1]Note: the expressions for ILS noise and atmospheric turbulence given in refs.[1] and [14] have an additional term π in the denominator. The Dryden spectra from ref.[21] do not contain this term, due to a slightly different definition of the Fourier transform. In this report, the definition of the Dryden filters from ref.[21] has been used, and due to the similarity of the expressions for ILS noise the term π will be omitted here too.

The definitions of the Fourier transform and the inverse Fourier transform used in ref.[21] are:

$$X(\omega) = F\{x(t)\} = \int_{-\infty}^{\infty} x(t) e^{-j\omega t}\, dt; \qquad x(t) = F^{-1}\{X(\omega)\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{j\omega t}\, d\omega$$

The power spectral density of the glide path noise appears to be similar to the localizer noise and may be approximated by:

$$S_{gs}(\Omega) = 2{\sigma_{gs}}^2 L_{gs} \frac{1}{1+(\Omega L_{gs})^2} \qquad \left[\mu A^2 rad^{-1} m\right] \tag{3.84}$$

where:

σ_{gs} = standard deviation of the glideslope noise,
L_{gs} = 'scale' of the glideslope noise, approximately 85 m
Ω = spatial frequency $[rad\, m^{-1}]$

For atmospheric turbulence it is often assumed that the turbulence velocities are functions only of the position in the atmosphere (the frozen field concept or Taylor's hypothesis). This assumption can be made because aircraft usually fly at large speeds compared to turbulence velocities. Using Taylor's hypothesis for the ILS noise will probably induce errors, especially for aircraft with very low final approach speeds such as the 'Beaver'. Still, this expression makes it possible to convert the spatial power spectral density functions to temporal expressions in ω, which can be used for practical simulations. One should remember that the power spectral density functions are in any case approximations of the actual ILS noise, so if a really accurate representation of ILS noise is required for simulations (e.g. for assessing automatic cat. III landing systems) an actual calibration of the localizer and glideslope signals for the site in question should be used.

With Taylor's hypothesis it is possible to substitute $\omega = \Omega V$. Then the ILS noise can be modeled as a white-noise signal which is sent through a linear forming filter in the same way as the derivations for atmospheric turbulence shown in figure 3.6. The resulting filters are:

$$H_{loc}(\omega) = \sigma_{loc}\sqrt{\frac{2L_{loc}}{V}}\frac{1}{1+\frac{L_{loc}}{V}j\omega} \tag{3.85}$$

$$H_{gs}(\omega) = \sigma_{gs}\sqrt{\frac{2L_{gs}}{V}}\frac{1}{1+\frac{L_{gs}}{V}j\omega} \tag{3.86}$$

Alternative shapes of the power spectral density functions for localizer and glideslope noise are given in ref.[17]. These expressions were based upon average power spectral density plots of beam noise at several airports:

$$S_{loc} = |H_{loc}(\omega)|^2 = \frac{25\,(1.5+j\omega)^2}{(0.35+j\omega)^2(10+j\omega)^2} \qquad \left[\mu A^2 rad^{-1} s\right] \tag{3.87}$$

$$S_{gs} = |H_{gs}(\omega)|^2 = \frac{15.9}{(0.25+j\omega)^2} \qquad \left[\mu A^2 rad^{-1} s\right] \tag{3.88}$$

The filters for these spectral density functions are:

$$H_{loc}(\omega) = \pm\frac{5\,(1.5+j\omega)}{(0.35+j\omega)(10+j\omega)} \tag{3.89}$$

$$H_{gs}(\omega) = \pm\frac{3.9875}{0.25+j\omega} \tag{3.90}$$

Both ILS noise models have been implemented in the FDC toolbox. The maximum allowable values of the standard deviations of localizer and glideslope noise, according to ICAO standards (ref.[2]) are given in figure 3.16.

In addition to the ILS noise and steady-state errors, specific deterministic interference patterns may occur due to signal reflections from aircraft in the vicinity of the glideslope and/or localizer transmitters. These disturbances may be quite severe and should be taken into account for the evaluation of automatic landing systems. It is possible to construct relatively simple

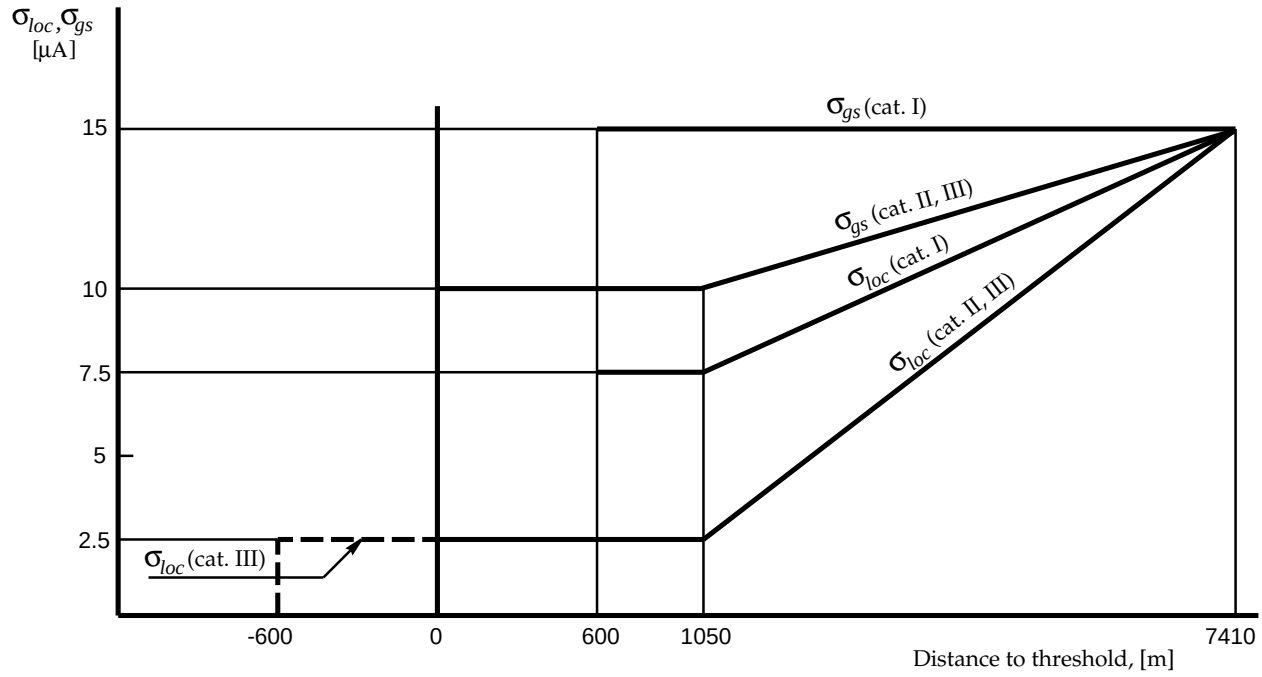


Figure 3.16: Maximum allowable ILS localizer and glideslope noise

models of these interference effects, but the FDC toolbox does not contain these models yet. See ref.[1] for more details.

3.4.2 The VOR navigation system

Nominal VOR signals

The Very high frequency Omnidirectional Radio range (VOR) system is a standard short-range radio-navigation aid. The system uses the 108-118 MHz frequency range. The VOR ground station radiates a cardioid pattern that rotates 30 times per second, thus generating a 30 Hz sine wave at the output of the airborne VOR receiver. The ground station also radiates an omnidirectional signal which is modulated with a 30 Hz reference tone. The phase difference between the two 30 Hz tones is a function of the *bearing* of the aircraft, relatively to the VOR ground station. A position fix can be obtained by using two or more VOR's or a combination of VOR and DME information.[1] See refs.[3], [4], or [18].

Figure 3.17 shows the geometry of the VOR system. Simple general-aviation VOR systems make it possible for the pilot to fly along a VOR radial, which must be selected by means of the 'Omni Bearing Selector' (OBS). This reference bearing is called 'Course Datum' (CD). The bearing where the aircraft is actually flying is denoted by QDR (a term used in radio telephony). The course deviation angle Γ_{VOR}, which is equal to the angle between the reference bearing and the actual bearing, is shown on the cockpit instrument. This information may also be used by an automatic control system to automatically follow a VOR radial. Modern airliners and business aircraft have more advanced *Area Navigation* systems which use information of multiple VOR stations and of other navigation equipment to follow arbitrary routes between so-called *waypoints*. In this report we will limit ourselves to the use of VOR systems for tracking VOR radials.

[1] DME = Distance Measuring Equipment.

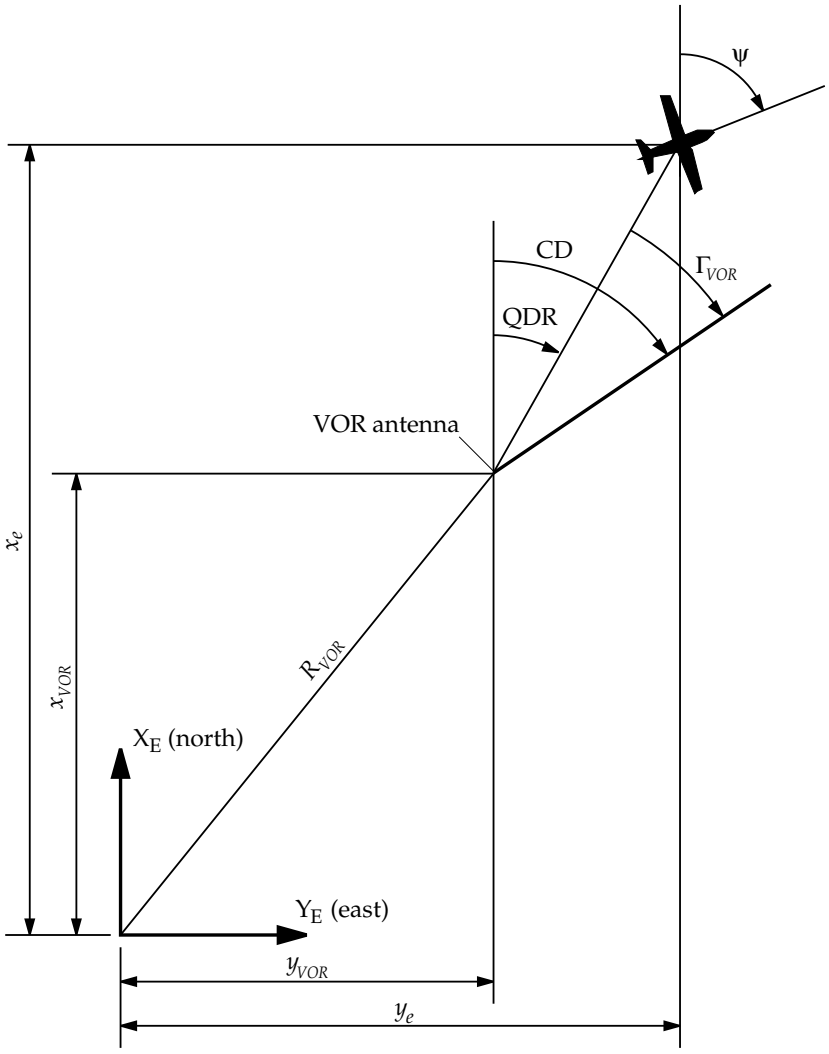


Figure 3.17: Geometry of VOR navigation

In order to compute VOR signals for simulation purposes it is necessary to know the exact positions of the VOR station and the aircraft with respect to the Earth-fixed reference frame. If the position of the VOR ground station is given by (x_{VOR}, y_{VOR}, H_{VOR}) and the position of the aircraft by (x_e, y_e, H), the following equations can be used to compute Γ_{VOR}:

$$QDR = \arctan\left(\frac{y_e - y_{VOR}}{x_e - x_{VOR}}\right) \tag{3.91}$$

$$\Gamma_{VOR} = CD - QDR \tag{3.92}$$

A typical value for a full-scale deflection of the cockpit instrument is $\Gamma_{VOR} = 10°$.

It is necessary to know whether the aircraft flies to or from the VOR transmitter. This information is visualized in the cockpit by means of a To-From indicator. If $|\psi - QDR| > 90°$ the *To* indicator should be visible; if $|\psi - QDR| < 90°$ the *From* indicator must be shown. In figure 3.17 the aircraft flies *from* the VOR transmitter, which is in accordance with these relations since $\psi - QDR \approx 40° < 90°$.

VOR coverage and cone of silence

The ground distance R_{VOR} can be used to determine whether the aircraft flies in an area where the VOR signals can be received with appropriate reliability. This distance is equal to:

$$R_{VOR} = \sqrt{(x_e - x_{VOR})^2 + (y_e - y_{VOR})^2} \tag{3.93}$$

If the aircraft flies in a certain area in the direct neighborhood of the VOR transmitter, the signals are not accurate. This area is formed by a cone with a top-angle of approximately 80 to 120 degrees, the so-called *Cone of Silence*. This has been shown in figure 3.18. See also ref. [3]. The aircraft flies outside the cone of silence if:

$$\xi \equiv \arctan\left(\frac{H - H_{VOR}}{R_{VOR}}\right) \leq 90° - (40° \text{ to } 60°) \tag{3.94}$$

Table 3.3 gives the maximum coverage of the VOR signals as a function of the height above ground level, according to ref.[4]. Based upon this table, the following approximative function for the VOR coverage as a function of the altitude (in [m]!) was found with the MATLAB function POLYFIT:

$$Range = 1000\left(-2.3570 \cdot 10^{-6}(H - H_{VOR})^2 + 5.7087 \cdot 10^{-2}(H - H_{VOR}) + 80.8612\right) \tag{3.95}$$

Another often used rule-of-thumb for determining the VOR coverage (see ref.[3]) is:

$$Range = 1.2\left(\sqrt{h} + \sqrt{h_{VOR}}\right) \tag{3.96}$$

Range is measured in nautical miles, h is the height above the ground measured in [*ft*] and h_{VOR} is the elevation of the VOR antenna above the ground measured in [*ft*]. Usually the latter term is neglected. For implementation in the FDC toolbox it will be necessary to convert this expression to S.I. units and substitute $h = H - H_{VOR}$. However, since the values from equations (3.95) and (3.96) don't differ much, only equation (3.95) has been implemented in the toolbox.

VOR steady-state errors

The nominal VOR signals become distorted by VOR noise and steady-state errors. There are two types of systematic errors: ground station errors and airborne equipment errors. Each of these errors comprises both the equipment and antenna errors and site or location errors.

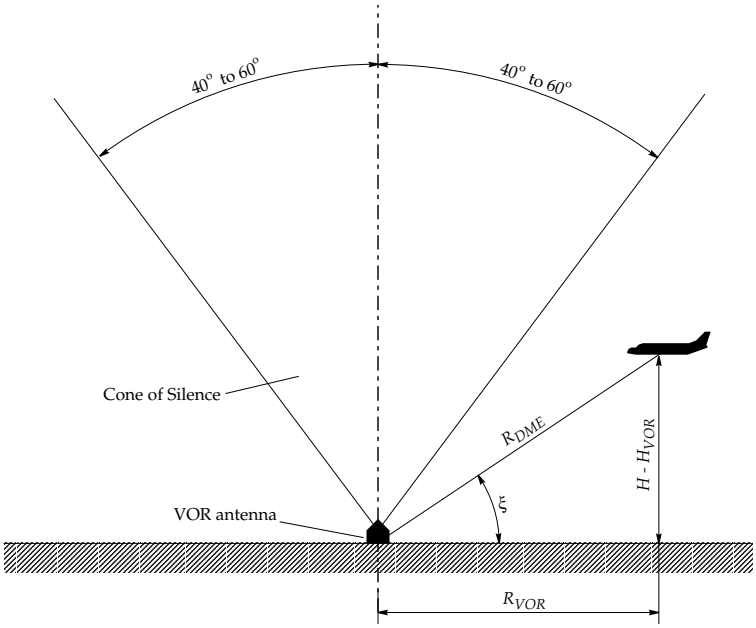


Figure 3.18: The 'Cone of Silence'

Height [*ft*]	VOR range [*NM*]
1000	50
5000	92
20000	182
30000	220
3000	75
5000	95

Table 3.3: VOR coverage based on two different flight tests

ICAO has established the following rules (refs.[2] and [4]):

1. the error of the airborne equipment must be smaller than $\pm 2°$ at a distance from the antenna of four times the wavelength and at an elevation-angle of $0°$ to $40°$,
2. the maximum error for the ground station is $\pm 3.5°$.

Ref.[4] presents some results of measurements of ground equipment errors. Typical measured values are errors of $\pm 1.4°$ to $\pm 2.5°$. Besides the constant steady-state errors of the ground and airborne equipment there are also random errors such as variations of supply voltage of the ground and/or airborne equipment, temperature changes, inaccurate instrument reading, etc. According to ref.[4] the following values for the overall VOR system error were obtained from flight tests using commercial aircraft:

$\varepsilon < \pm 1.7°$ (68% of the tests)
$\varepsilon < \pm 3.4°$ (95% of the tests)
$\varepsilon < \pm 5.1°$ (99.7% of the tests)

Since ref.[4] is already somewhat outdated, it can safely be assumed that modern VOR stations and receivers are more accurate.

3.5 Sensors, Actuators, Flight Control Computer

Depending upon the tasks of the control system and its hardware characteristics, other dynamic submodels may be necessary to complete the block-diagram from figure 3.1. The control laws use measured signals, which do not fully correspond with the 'ideal' outputs from the aircraft model, due to the characteristics of the sensors that measure them. It may be necessary to add noise components and/or steady-state errors, to take into account filter characteristics of the sensors, to include time-delays, etc. Moreover, it may be necessary to take into account the influence of computer processing upon the signals in case a digital Flight Control Computer is used to determine the control surface deflections. In the example of the 'Beaver' autopilot, a quantization effect in the altitude measurements was encountered due to a Least Significant Bit that corresponded with 4 feet. And although a steady-state error of 4 feet would have been quite acceptable for the Altitude Hold mode of our autopilot, the resulting discontinuities in the altitude signal proved to yield unacceptable control characteristics (see ref.[22]) that required additional filtering of the altitude signal. This example shows how important it can be to take into account the differences between the 'ideal' outputs from the aircraft model and the measured signals which enter the control laws in practice.

On the output side of the control laws, the 'ideal' control surface deflections are distorted due to the characteristics of the cables and actuators and because of additional time-delays and discretization effects due to the characteristics of the computer systems. Moreover, the range of deflections of the actuators and the control surfaces is limited and the rate of change of these deflections may be limited for reasons of constructional strength and stiffness. Since these effects may have an adverse effect upon the behavior of the AFCS, it is necessary to analyze their influence in non-linear simulations.

Apart from the radio-navigation models from the previous section, which in fact are just sophisticated sensor models, the FDC 1.2 toolbox does not yet contain a complete library of sensor models, actuator models, and computational effects. It does contain some simple ad-hoc solutions for the assessment of the 'Beaver' autopilot, including simple linear second-order models of the actuator dynamics and the elastic behavior of the cables between the actuators and the control surfaces (for the 'Beaver' aircraft these cables were unusually long). The altitude signal was artificially quantized to take into account the Least Significant Bit of 4 feet. Computational delays can be taken into account by means of standard Time Delay blocks from SIMULINK. Future versions of the toolbox should be equipped with standard libraries to provide 'of the shelf' sensor and actuator models for general control system design purposes.

Chapter 4

Analytical tools – theoretical backgrounds

4.1 Introduction

This section describes the theoretical backgrounds of some important analytical tools from SIMULINK and FDC 1.2. In section 4.2, the numerical integration methods from SIMULINK are described. This section provides some general theoretical backgrounds on numerical integration techniques, which may be useful for determining the most suitable integration method and the kinds of errors to expect. This section can be regarded as a theoretical addition to the somewhat limited treatment of the simulation tools in the SIMULINK user manual. In section 4.3 the trim-routine from FDC 1.2 is treated. Contrary to the simulation and linearization tools, the trim-routine does not use the built-in SIMULINK trim-functions, but a custom-made algorithm which is especially suited for the determination of steady-state flight conditions. Section 4.3 provides the theoretical basis for the treatment of the trim routines from section 8.2. Section 4.4 gives a short theoretical description about the linearization tool from SIMULINK which is used by some routines from FDC 1.2 to linearize the non-linear aircraft model in some user-specified operating point.

4.2 Simulation tools

4.2.1 Introduction

FDC 1.2 makes use of the built-in simulation functions of SIMULINK for the determination of aircraft responses to control inputs or external disturbances. These simulation functions are fulfilled by six different numerical integration routines:

1 – EULER. This is a one-step integration method which simply multiplies the time-derivatives of the state variables with the step-size. Since it requires much smaller step-sizes than the other methods in order to achieve the same accuracy, this method is not recommended for the majority of simulation problems. It will not be used for FDC simulations.

2 – LINSIM. This method divides the system in linear and non-linear subsystems. The linear subsystems are discretized and then solved in a straightforward way, leaving only the non-linear system dynamics to be solved numerically. This works best for systems which are 'relatively linear'. For the highly non-linear SIMULINK models from the FDC toolbox this method is not suited.

3 – RK23. This is a third-order Runge-Kutta method which uses a second-order method for step-size control. It is a general purpose method which works well for a large range of simulation problems and for the simulation of systems with discontinuities. In theory this makes it a suitable candidate for FDC simulations, although RK45 and ADAMS/GEAR turned out to be faster in practice.

4 – RK45. This is a fifth-order Runge-Kutta method which uses a fourth-order method for step-size control. The method is usually faster and more accurate than RK23 and it produces fewer output points. For this reason it has been applied in practice for FDC simulations. If the input signals are smooth, the ADAMS/GEAR methods may be more suitable than the Runge-Kutta methods.

5 – ADAMS. This is a so-called predictor-corrector method which uses a variable number of points for the generation of one output point. It works well for systems with smooth state-trajectories and it can be used in combination with GEAR. This combination has been applied for FDC simulations in cases where the state-trajectories were smooth, i.e. if the systems were not affected by noisy disturbances or many discontinuous input signals.

6 – GEAR. This method is primarily designed for the simulation of systems with a mixture of very fast and slow dynamics (so-called 'stiff' systems, see section 4.2.5). It has been applied in combination with ADAMS for FDC simulations, where the autopilot systems in particular exhibited such a mixture of fast and slow dynamics (namely the fast dynamics of the control actuators and the slow dynamics of the phugoid and spiral modes of the aircraft). The Runge-Kutta method RK45 was used for FDC simulations in cases where the input signals to the aircraft model contained noisy disturbances or many discontinuities.

In the next sections, some theoretical aspects of these methods will be outlined. These sections are largely based upon refs.[12], [13], and [25]. The actually used integration equations in SIMULINK are neither accessible to the users, nor described in detail in the SIMULINK user's manual, so it is not sure that the equations from the next sections are exactly equal to the relations used by SIMULINK, but they do give a good general overview of the theoretical backgrounds of numerical integration.

4.2.2 The type of problems considered

The numerical integration methods used in SIMULINK are designed to determine the time-trajectories of continuous state variables of dynamical systems described by *ordinary differential equations* (ODE's):

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t); \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{4.1}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input vector, $\mathbf{f}$ is some non-linear function, and $\mathbf{x}_0$ is the initial value of the state vector at time t_0. Since few differential equations can be solved exactly, the solutions of these ODE's must be approximated numerically. The numerical integration methods have been developed for solving so-called *initial value problems*:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t); \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{4.2}$$

This equation does not take into account the input vector $\mathbf{u}$ anymore, but the methods for solving equation (4.2) are similar to the methods for dealing with equation (4.1). If the state vector $\mathbf{x}$ has N elements, N constants of integration appear in the solution of equation (4.2). A unique solution to this system can be obtained only if the initial values of the states are specified. The techniques for solving the vector equation (4.2) are essentially the same as the techniques used for solving scalar initial value problems given by the equation:

$$\dot{x}(t) = f\left(x(t), t\right) \tag{4.3}$$

The numerical integrators from SIMULINK generate a sequence of discrete points t_0, t_1, t_2, ... in time, possibly with variable spacing $h_n = t_{n+1} - t_n$ (the step-size). At each point t_n the solution $x(t_n)$ is approximated by x_n, which is computed from earlier values of x. If k earlier values x_n, x_{n-1}, ..., x_{n-k+1} are used for the computation of x_{n+1}, the method is called a 'k-step method'. For instance, the Euler method (the integrator EULER in SIMULINK) is a single-step method:

$$x_{n+1} = x_n + h_n \cdot f(x_n, t_n) \tag{4.4}$$

4.2.3 Stability, errors, and order of a numerical integration method

Figure 4.1 shows a typical family of solutions of a first-order differential equation for different initial values x_0. Here, a wrong value of the initial condition yields an error which increases in time, i.e. the differential equation is *unstable*. The figure demonstrates how the Euler approximation generally crosses from one solution to another between to time steps. For this unstable differential equation, the resulting error increases in time. In figure 4.2 the solutions of the ODE converge as time proceeds which means that this ODE is stable and numerical integration errors do not increase with time. Non-linear differential equations may be unstable in some regions and stable in others. For systems of ODE's, the situation is even more complex. One should always be aware of possible instabilities of dynamic systems when assessing numerical results.

There are two types of errors in numerical integration processes:

1. discretization errors
2. round-off errors

Discretization errors are a property of the numerical integration method, while round-off errors occur due to the finite number of digits used in the calculations (hence they are a property of the computer and the program that is used). In general, the total error decreases as the step-size h_n decreases, until a point where the round-off error becomes dominant. This is illustrated in figure 4.3. Due to these errors it is possible that the numerical solution becomes unstable, even if the ODE itself is stable. See for instance figure 4.4, which shows a system that is numerically integrated with the Euler method with a too large step-size.

The *order* of a numerical integration method is defined in terms of the local discretization error δ_n, obtained when the method is applied to problems with smooth solutions. A method is of order p if a number C exists so that:

$$|\delta_n| \leq C {h_n}^{p+1} \tag{4.5}$$

C may depend on the derivatives of the function which defines the differential equation and on the length of the interval over which the solution is sought, but it should be independent of the step number n and the step-size h_n.

4.2.4 Different categories of numerical integration methods

According to ref.[12], it is possible to distinguish between *four* general categories of step-by-step methods which will briefly be discussed here. All results can easily be converted to vector notations for sets of Ordinary Differential Equations.

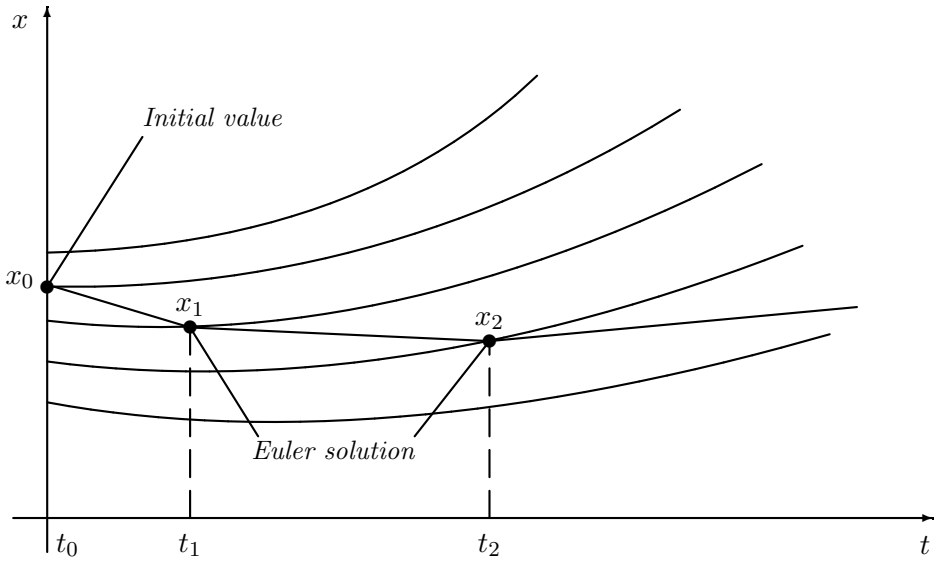


Figure 4.1: Family of solutions of unstable ODE (Euler method displayed graphically)

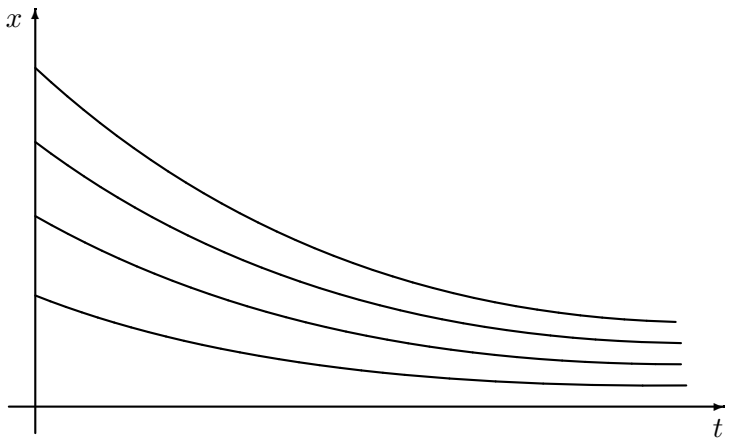


Figure 4.2: Family of solutions of stable ODE

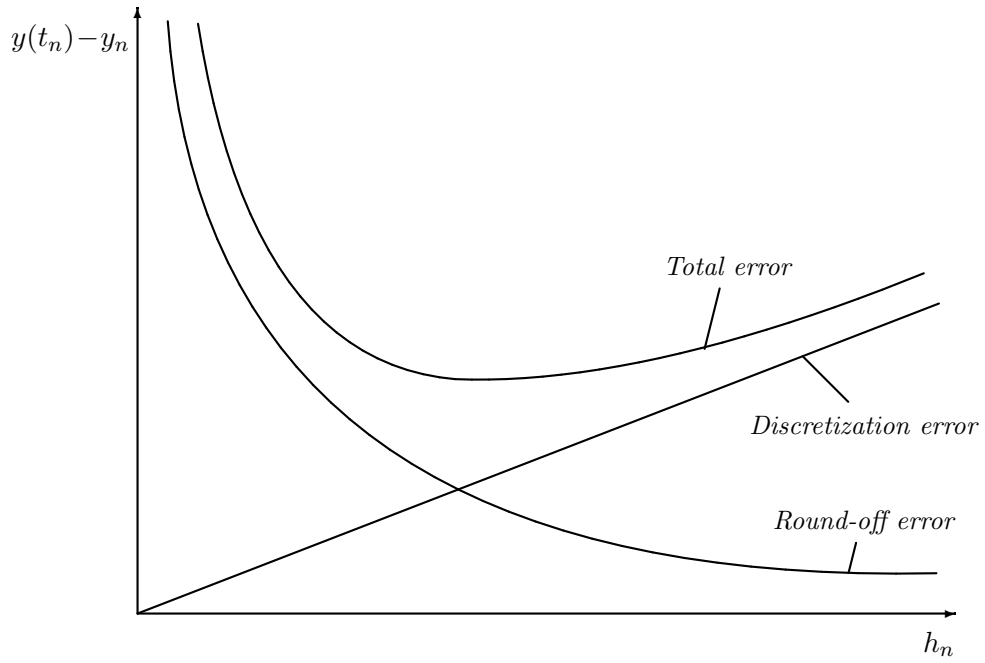


Figure 4.3: Discretization error, round-off error, and total error as a function of the step-size

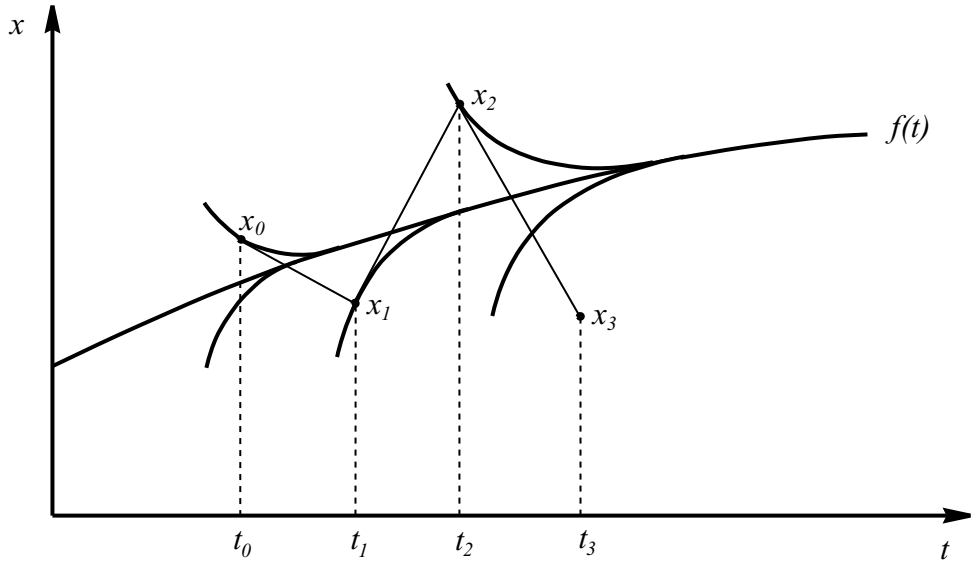


Figure 4.4: Unstable Euler solution for stable ODE

1. Taylor series methods

A smooth solution $x(t)$ of equation (4.3) can be approximated by a Taylor series expansion:

$$x(t+h_n) = x(t) + h_n\dot{x}(t) + \frac{{h_n}^2}{2!}\,\ddot{x}(t) + \dots \tag{4.6}$$

Provided it is possible to determine higher-order time-derivatives of x, a numerical method of order p can be obtained by using:

$$x_{n+1} = x_n + h_n\dot{x}_n + \frac{{h_n}^2}{2!}\,\ddot{x}_n + \ \dots\ + \frac{{h_n}^p}{p!}\left(\frac{d^p x}{dt^p}\right) \tag{4.7}$$

The first neglected term provides an estimate of the local discretization error. An example of a Taylor series method is the Euler method (equation (4.4)), which neglects all time-derivatives of order two and higher. Hence, the local discretization error of the Euler method is of order h^2.

2. Runge-Kutta methods

Runge-Kutta methods approximate Taylor series methods without evaluating time-derivatives beyond the first. The higher-order derivatives are replaced by a number of evaluations of the function f. Modern Runge-Kutta algorithms typically include techniques for estimating the discretization error in order to control the step-size. Runge-Kutta methods require only one value x_n in order to compute x_{n+1}, which makes them *self-starting*. The reader is referred to ref.[13] for a derivation of a second-order Runge-Kutta method which clearly demonstrates the relations between the Taylor series and the Runge-Kutta approximation.

The fifth-order Runge-Kutta method, used by the integrator RK45 can be described by the following formulas:

$$k_i \;=\; h_n f\left(x_n + \sum_{j=1}^{i-1}\beta_{ij}\,k_i,\ t_n + \alpha_i\,h_n\right); \qquad i = 1,\dots,6 \tag{4.8}$$

$$x_{n+1} \;=\; x_n + \sum_{i=1}^{6}\gamma_i\,k_i \tag{4.9}$$

These equations contain 27 coefficients: 6 α's, 15 β's, and 6 γ's. The β's form a lower triangular array, so that each k_i is obtained from the previous k's. By expanding all k's in Taylor series, substituting the expansions into the formula for x_{n+1}, and comparing the result with the Taylor series for the true *local* solution of the differential equation the coefficients can be determined. This is elaborated in detail in ref.[13].

The combination of coefficients used in the fifth-order Runge-Kutta Fehlberg routine are given in table 4.1, see refs.[12] and [13]. The table also contains a second set of coefficients ${\gamma_i}^*$ which together yield another method which is accurate to fourth order. This fourth-order solution is therefore:

$${x_{n+1}}^* = x_n + \sum_{i=1}^{6}{\gamma_i}^* k_i \tag{4.10}$$

The difference δ_n between the fifth and fourth-order solutions, used for step-size control, is equal to:

$$\delta_n \equiv x_{n+1} - {x_{n+1}}^* = \sum_{i=1}^{6}(\gamma_i - {\gamma_i}^*) \tag{4.11}$$

α_i	β_{ij}					γ_i	$\gamma_i{}^*$
0						$\frac{16}{135}$	$\frac{25}{216}$
$\frac{1}{4}$	$\frac{1}{4}$					0	0
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				$\frac{6656}{12825}$	$\frac{1408}{2565}$
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			$\frac{28561}{56430}$	$\frac{2197}{4104}$
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		$-\frac{9}{50}$	$-\frac{1}{5}$
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	$\frac{2}{55}$	0

Table 4.1: Fehlberg coefficients for integrator RK45

A vector equivalent of this scheme is used by the MATLAB routine ODE45 (and probably also by the SIMULINK integrator RK45). In this MATLAB routine the acceptable error is calculated with:

$$\tau_n = tol \cdot \max\{|x_n|, 1\} \tag{4.12}$$

or a vector equivalent of this equation, where *tol* is the desired accuracy. Knowing the acceptable error τ_n and the error estimate δ_n, ODE45 updates the step-size as follows:

$$h_{n+1} = \min\left\{ h_{max}\,,\ 0.8\,h_n \left(\frac{\tau_n}{\delta_n}\right)^{\frac{1}{5}} \right\} \qquad \text{with: } h_{n+1} \geq h_{min} \tag{4.13}$$

Obviously the step-size is not updated if $\delta_n = 0$. Similar expressions are applied for vector equations. The integrator RK23 uses a similar scheme. This algorithm is probably equal to the code from the MATLAB routine ODE23 which updates the state variable with the following expressions:

$$\begin{aligned} k_1 &= f\,(x,\,t) \\ k_2 &= f\,(x + h_n k_1,\,t + h_n) \\ k_3 &= f\left(x + \frac{h_n}{4}(k_1 + k_2),\,t + \frac{h_n}{2}\right) \\ x_{n+1} &= x_n + \frac{h_n}{6}(k_1 + 4k_3 + k_2) \end{aligned} \tag{4.14}$$

where h_n is updated in a similar manner as for the RK45 method.

3. Multistep methods

Contrary to Runge-Kutta and Taylor series methods, multistep methods use information at previous points to obtain a better accuracy. Multistep methods can be very effective. They usually require less function evaluations than one-step methods of equal accuracy. Furthermore, an estimate of the discretization error can often be trivially obtained (ref.[12]). All linear multistep methods can be considered as special cases of the formula:

$$x_{n+1} = \sum_{i=1}^{k} \alpha_i\, x_{n+1-i} + h_n \sum_{i=0}^{k} \beta_i\, f_{n+1-i} \tag{4.15}$$

where $f_i = f\,(x_i, t_i)$, k is an integer, and either α_k or β_k is not zero. This formula defines the general k-step method. It is linear because every f_i appears linearly in the equation; f itself does not necessarily have to be a linear function in its arguments. After the method is 'started', each step requires the calculation of x_{n+1} from the known values x_n, x_{n-1}, ..., x_{n-k+1}, f_n,

$i:$	1	2	3	4	5	6
β_{1i}	1					
$2\beta_{2i}$	3	-1				
$12\beta_{3i}$	23	-16	5			
$24\beta_{4i}$	55	-59	37	-9		
$720\beta_{5i}$	1901	-2774	2616	-1274	251	
$1440\beta_{6i}$	4277	-7923	9982	-7298	2877	-475

Table 4.2: Coefficients of the Adams-Bashforth integration method

$i:$	1	2	3	4	5	6
${\beta_{1i}}^*$	1					
$2{\beta_{2i}}^*$	1	1				
$12{\beta_{3i}}^*$	5	8	-1			
$24{\beta_{4i}}^*$	9	19	-5	1		
$720{\beta_{5i}}^*$	251	646	-264	106	-19	
$1440{\beta_{6i}}^*$	475	1427	-798	482	-173	27

Table 4.3: Coefficients of the Adams-Moulton integration method

$f_{n-1}, \ldots, f_{n-k+1}$. If $\beta_0 = 0$ this method is *explicit* and the calculation is straightforward. If $\beta_0 \neq 0$ the method is *implicit* because $f_{n+1} = f(x_{n+1}, t_{n+1})$ is then needed to solve for x_{n+1}.

Usually a combination of two multistep methods is used for computing each step of the solution: an explicit method, called *predictor*, followed by one or more applications of an implicit method, which is called a *corrector*. The ADAMS method is a predictor-corrector multistep method. It is not known exactly what integration scheme is used by SIMULINK, but the most important Adams methods found in literature are the explicit Adams-Bashforth integration method and the implicit Adams-Moulton method. Probably these algorithms are used by SIMULINK too. The k-step Adams-Bashforth formula can be written as:

$$x_{n+1} = x_n + h_n \sum_{i=1}^{k} \beta_{ki} f_{n+1-i} \tag{4.16}$$

Table 4.2 lists some values β_{ki} for this method. The k-step Adams-Moulton formula is equal to:

$$x_{n+1} = x_{n-1} + h_n \sum_{i=0}^{k-1} {\beta_{ki}}^* f_{n+1-i} \tag{4.17}$$

Table 4.3 lists some values ${\beta_{ki}}^*$ for this method. Often the Adams-Bashforth method is used as predictor and Adams-Moulton as corrector.

4. Extrapolation methods

The predictor methods actually extrapolate the value x_{n+1} from known previous values of x and the function evaluations f. There exist other types of extrapolation methods as well. These methods will not be discussed here, since the SIMULINK integrators do not use them. See ref.[13] for more information.

k :	2	3	4	5	6
β_0	$\frac{2}{3}$	$\frac{6}{11}$	$\frac{12}{25}$	$\frac{60}{137}$	$\frac{60}{147}$
α_1	$\frac{4}{3}$	$\frac{18}{11}$	$\frac{48}{25}$	$\frac{300}{137}$	$\frac{360}{147}$
α_2	$-\frac{1}{3}$	$-\frac{9}{11}$	$-\frac{36}{25}$	$-\frac{300}{137}$	$-\frac{450}{147}$
α_3		$\frac{2}{11}$	$\frac{16}{25}$	$\frac{200}{137}$	$\frac{400}{147}$
α_4			$-\frac{3}{25}$	$-\frac{75}{137}$	$-\frac{225}{147}$
α_5				$\frac{12}{137}$	$\frac{72}{147}$
α_6					$-\frac{10}{147}$

Table 4.4: Coefficients for stiffly stable integrator (GEAR)

4.2.5 Stiff differential equations

'Stiffness' of the differential equations can roughly be defined as the presence of one or more fast decay processes in time, with a time constant that is short compared to the time-span of interest. The *time constant* is defined as the time in which a solution to a differential equation decays by a factor $1/e$. In a physical system, different elements often have different time constants, which means that some solutions to differential equations decay much faster than others. In such cases the signals with fast dynamics will determine the stability of the integration method, even though these components may have decayed to insignificant levels. Figure 4.4 shows a family of solutions of a stiff system, which is integrated by an Euler method with a too large step-size. Although this system is stable, the numerical solution diverges rapidly. The only way to solve this problem is by reducing the step-size, but eventually round-off and discretization errors will accumulate enough to result in another instability. The transient part of the solution, which decays very fast, prevents an increase in step size, although the solution is very smooth after only a few seconds.

SIMULINK contains two integrators which are particularly suited for solving stiff ODE's: GEAR and LINSIM. Of these integrators, LINSIM is only suited for 'almost linear' systems. GEAR works well for systems with smooth, non-linear, stiff solutions. It uses a predictor-corrector method which takes a variable number of steps between two output points. This method is probably based upon the stiffly stable method from ref.[13], which can be written as:

$$x_n = \sum_{i=1}^{k} \alpha_i \, x_{n-i} + h_n \, \beta_0 \, f_n \tag{4.18}$$

The coefficients α_i and β_0 are listed in table 4.4. The method differs from the Adams method in the way in which the implicit expression is solved. In SIMULINK it is possible to use a combination of ADAMS and GEAR, where SIMULINK itself will decide which of the two methods to use, depending upon the stiffness of the equation.

4.2.6 Obtaining state-models from transfer functions

In SIMULINK it is possible to define system dynamics by means of *transfer functions*, which are converted to state equations by SIMULINK itself to make it possible to apply the numerical integration routines. It is useful to know how this conversion may be achieved in order to know more about the computation methods of SIMULINK, and also because it enables us to implement transfer functions with non-constant coefficients within graphical SIMULINK systems. In the case

of the FDC toolbox, this conversion scheme has been applied in practice for the implementation of the Dryden turbulence filters which are defined by transfer functions with airspeed-dependent coefficients.

Several possible transformations from transfer functions to linear state models can be found in the literature, see for instance refs.[5], [7], or [25]. Only one method will be shown here. Consider the following transfer function:

$$H(s) \equiv \frac{Y(s)}{U(s)} = \frac{b_{n-1}s^{n-1} + \ \ldots\ F + b_2 s^2 + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \ \ldots\ + a_2 s^2 + a_1 s + a_0} \tag{4.19}$$

To transform this equation into a state model, it will first be rewritten and a help function $V(s)$ will be introduced:

$$V(s) \equiv \frac{Y(s)}{b_{n-1}s^{n-1} + \ \ldots\ + b_1 s + b_0} = \frac{U(s)}{s^n + a_{n-1}s^{n-1} + \ \ldots\ + a_1 s + a_0} \tag{4.20}$$

This yields the following equations:

$$Y(s) \ = \ \left(b_{n-1}s^{n-1} + \ \ldots\ + b_1 s + b_0\right) V(s) \tag{4.21}$$

$$s^n V(s) \ = \ U(s) - a_{n-1}s^{n-1}V(s) - \ \ldots\ - a_1 s V(s) - a_0 V(s) \tag{4.22}$$

Equations (4.21) and (4.22) can be used to obtain a linear state model. Figure 4.5 shows the expanded block-diagram equivalent of transfer function (4.19). The diagram is constructed with first-order integrals (the $1/s$ blocks) and simple gain blocks. If we define the state vector for this transfer function block as:

$$\mathbf{x} = V(s) \begin{bmatrix} 1 \\ s \\ s^2 \\ s^3 \\ \vdots \\ s^{n-1} \end{bmatrix} \tag{4.23}$$

the following linear state equations are found:

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + \mathbf{b}u \\ y &= \mathbf{c} \cdot \mathbf{x} \end{aligned} \tag{4.24}$$

with state matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -a_0 & -a_1 & -a_2 & \ldots & -a_{n-1} \end{bmatrix} \ ; \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \ ; \quad \mathbf{c} = [\, b_0 \ b_1 \ b_2 \ \ldots \ b_{n-1} \,] \tag{4.25}$$

For the implementation of the Dryden turbulence filters and for one filter in the Altitude Select mode of the 'Beaver' autopilot simulation model, block-diagrams similar to figure 4.5 have been used in practice. Note that transfer functions with constant coefficients can be implemented in SIMULINK directly by means of built-in transfer-function blocks.

4.2.7 Algebraic loops

One difficulty that can arise during simulation of a continuous-time system on a digital computer is the occurrence of *algebraic loops*. Consider a system in block-diagram representation. The actual model is in fact a parallel system: all variables change simultaneously. But the calculation

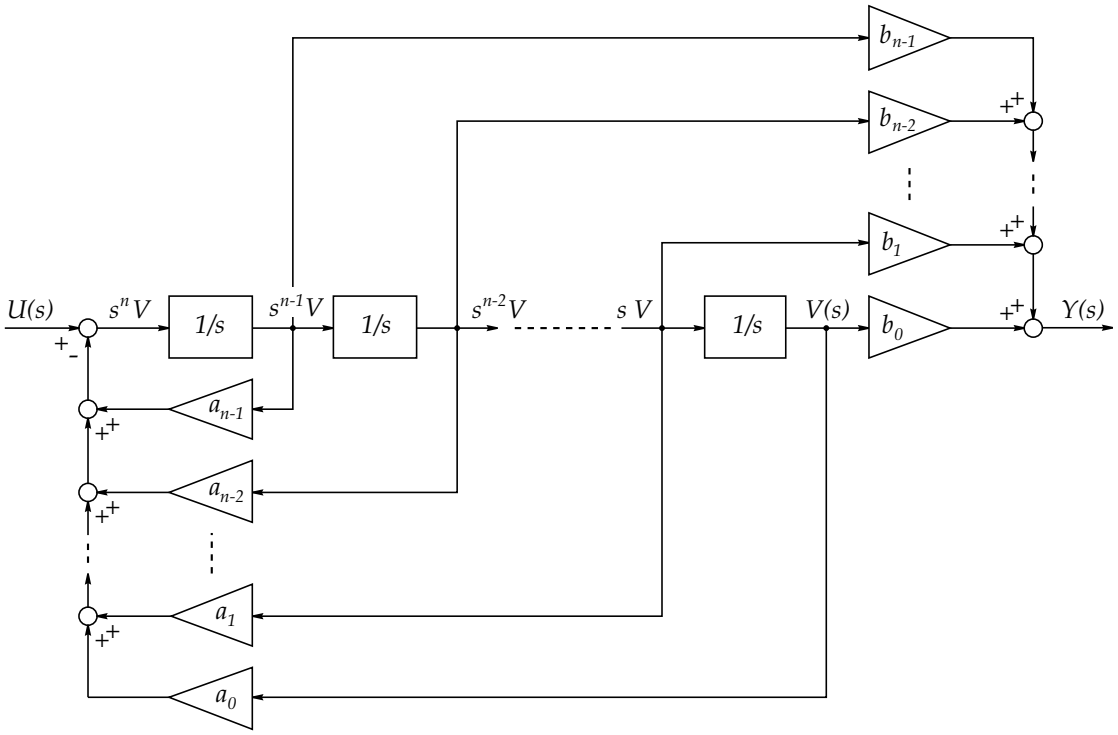


Figure 4.5: Block-diagram equivalent of transfer function

of responses of a parallel system on a *digital* computer system (with one microprocessor) can only be done sequentially. This means that the simulation program must choose an order of calculations. However, when feedback is applied, it is possible that no suitable sequence can be found. This situation occurs when two or more blocks with direct feed-through of their inputs form a feedback-loop. This is called an algebraic loop. For instance, consider the system in figure 4.6, consisting of a gain A with negative unity feedback. If this system is analyzed with an integrator with step width h_n, it is possible to write:

$$\begin{aligned} e_n &= u_n - y_{n-1} \\ y_n &= Ae_n \end{aligned}$$

where $y_n \equiv y(n\,h_n)$ and $y_{n-1} \equiv y((n-1)\,h_{n-1})$ (n is an integer). Taking the Z-transform of these equations yields:

$$\frac{Y(z)}{U(z)} = \frac{Az}{z+A}$$

which indicates that additional dynamics due to the sequential calculation of the parallel feedback system have been introduced (see figure 4.7).

When SIMULINK detects an algebraic loop it will use an iterative Newton-Rhapson method to solve the resulting implicit problem (see the SIMULINK user's manual and ref.[5]). First, a new block ALB is added to the system which contains the implicit algebraic equations that result due to the algebraic loop, see figure 4.8. This block has an input $g(x)$ and an output x. The block tries to find a value of x such that $g(x) = x$. This means that the following non-linear equation has to be solved:

$$G(x) = g(x) - x = 0 \tag{4.26}$$

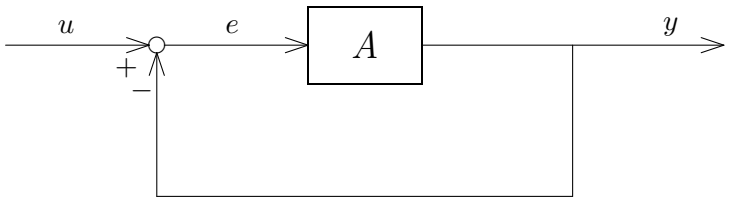


Figure 4.6: Gain with unity feedback: an algebraic loop

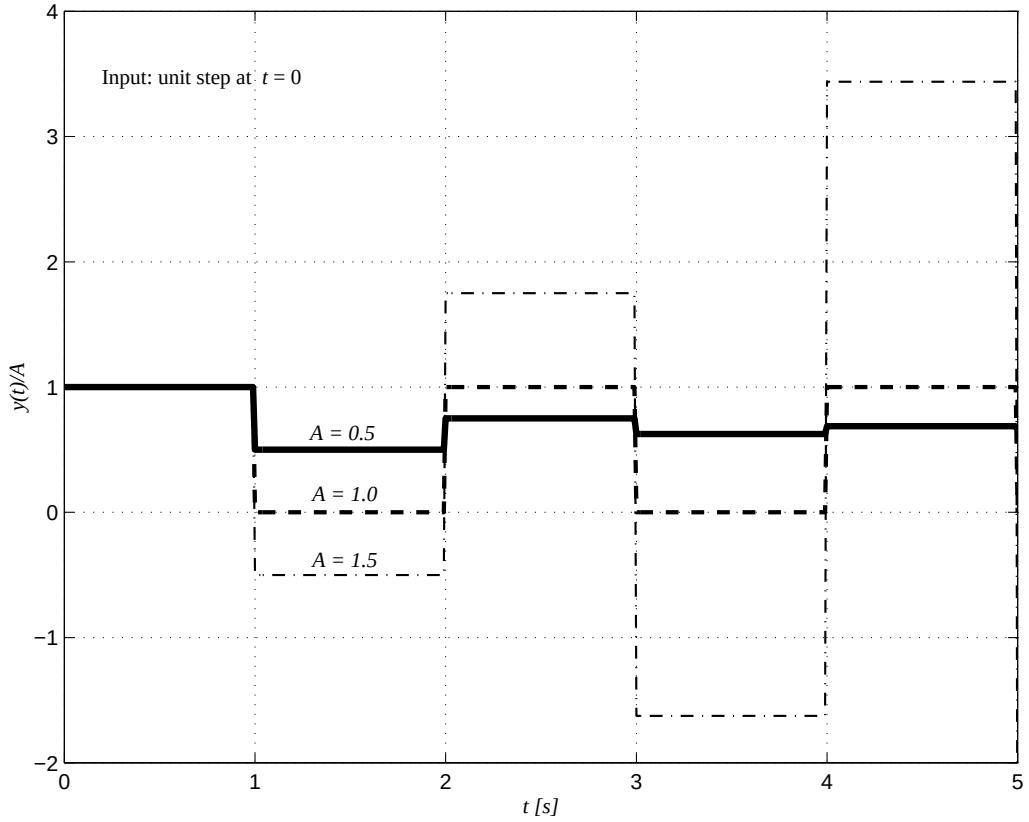


Figure 4.7: Dynamics, introduced by an algebraic loop (gain with unity feedback)

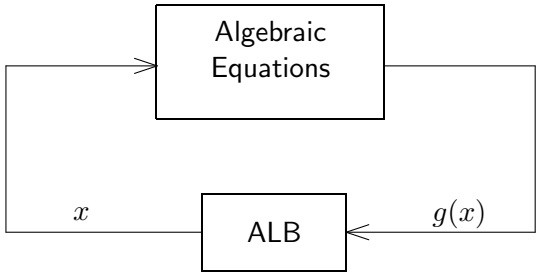


Figure 4.8: Iterative solution of an algebraic loop

This is done by applying the Newton-Rhapson method:

$$x_{i+1} = x_i - \left(G'(x_i)\right)^{-1} G(x_i) \tag{4.27}$$

where:

$$G'(x_i) = \frac{G(x_i + \Delta x)}{\Delta x}; \qquad (\Delta x \text{ small}) \tag{4.28}$$

The subscript i denotes the iteration number, not the time. SIMULINK returns an error if it can't solve the algebraic loop within 200 iterations. Note that the Newton-Rhapson iterations have to be carried out for every time-step, i.e. the integration will inevitably slow down. Moreover, algebraic loops may be too complex for SIMULINK to find a solution. Therefore one should try to avoid algebraic loops whenever possible. It is possible to 'break' an algebraic loop by including a dynamical element in the feedback loop, e.g. a filter. See ref.[5] for more details.

4.3 The trimming facility from FDC 1.2

The FDC toolbox contains a custom trimming routine which can be applied to find steady-state flight conditions. Such flight conditions can be applied as 'operating points' for the linearization process, which is necessary for the design of AFCS's based upon linear system theory, and as initial conditions for simulations. Although SIMULINK contains a general trim routine (the MATLAB function TRIM), the special problem of trimming a non-linear aircraft model can better be solved by means of a specialized aircraft trim routine. The theoretical backgrounds for such a routine will be shown in this section. This section has largely been based upon ref.[25].

4.3.1 Definition of steady-state flight

Recall the non-linear state equation (3.7) from section 3.2 for describing the general rigid-body dynamics:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t) \tag{4.29}$$

with state vector $\mathbf{x}$, input vector $\mathbf{u}$, and external disturbance vector $\mathbf{v}$. This equation is actually a special case of the more general *implicit* differential equation:

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t) = 0 \tag{4.30}$$

In ref.[25], a *singular point* or *equilibrium point* of a time-invariant system with no external control inputs is defined as:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{u}) = 0\,, \text{ with: } \dot{\mathbf{x}} = 0 \text{ and: } \mathbf{u} = 0 \text{ or constant} \tag{4.31}$$

Here the disturbance vector $\mathbf{v}$ has been omitted. The system is 'at rest' when all of the time-derivatives are identically zero.

Steady-state flight can be defined as a condition in which all of the motion variables are constant or zero and all acceleration components are zero. This definition is very restrictive, unless some simplifying assumptions are made. The definition allows steady wings-level flight and steady turning flight if the flat-Earth equations of motion are used, assuming that the aircraft's mass remains constant during the motions of consideration. If the change in atmospheric density with altitude is neglected during the trim process, a wings-level climb and a climbing turn are permitted as steady-state flight conditions too. The equations for the coordinates x_e and y_e, and the altitude H then don't couple back into the equations of motion and don't need to be used in finding a steady-state equation. So steady-state flight can be defined in terms of the remaining *nine* state variables of the flat-Earth equations:

$$\dot{p},\ \dot{q},\ \dot{r},\ \dot{V},\ \dot{\alpha},\ \dot{\beta} = 0\,,\ \ \mathbf{u} = \text{constant} \tag{4.32}$$

Additional constraints have to be made to define the exact flight condition. Here we consider steady wings-level flight, steady turning flight, steady pull-up or push-over, and steady rolls, which are defined by the following constraints:

steady wings-level flight:	$\varphi,\ \dot{\varphi},\ \dot{\theta},\ \dot{\psi} = 0$	(i.e. $p,\ q,\ r = 0$)
steady turning flight:	$\dot{\varphi},\ \dot{\theta} = 0,$	$\dot{\psi} =$ turn rate
steady pull-up:	$\varphi,\ \dot{\varphi},\ \dot{\psi} = 0,$	$\dot{\theta} =$ pull-up rate
steady roll:	$\dot{\theta},\ \dot{\psi} = 0,$	$\dot{\varphi} =$ roll rate

The conditions $\dot{p},\ \dot{q},\ \dot{r} = 0$ require the angular rates – and therefore also the aerodynamic and thrust moments – to be zero or constant. The conditions $\dot{V},\ \dot{\alpha},\ \dot{\beta} = 0$ require the aerodynamic forces to be zero or constant. For this reason, the steady-state pull-up/push-over and steady roll conditions can only exist instantaneously. Still, it can be useful to trim the aircraft dynamics in such flight conditions (and use the resulting trim values of $\mathbf{x}$ and $\mathbf{u}$ to linearize the aircraft model for these flight conditions) because control systems must operate there too.

To find a steady-state flight condition, a set of non-linear simultaneous equations, derived from the state model, must be solved. Due to the very complex functional dependence of the aerodynamic data, it is in general not possible to solve these equations analytically. Instead, a numerical algorithm must be used to iteratively adjust the independent variables until some solution criterion is met. The solution will be approximate, but can be made arbitrarily close to the exact solution by tightening up the criterion. Also, the solution may not be unique; for example, steady-state level flight at a given engine power level can correspond to two different airspeeds and angles of attack. Our knowledge of aircraft behavior makes it possible to specify the required steady-state condition so that the trim algorithm will converge on an appropriate solution. The trim algorithm presented here will deal with the aircraft model only through its input and output signals. It does not have to work *within* the model to balance forces and moments separately, which makes the trim-routine generally applicable. Hence, any aircraft model using the same input and state vectors as the model from this report can be trimmed with the same program, the internal structure of the aircraft model does not matter.

In the next three sections, first it will be shown how the steady-state condition can be specified, how many of the control variables may be chosen independently, and what constraints exist on the remaining variables. Then, an algorithm which solves the non-linear equations numerically will be developed.

4.3.2 Specification of the flight condition

The (initial) values of the altitude, airspeed, and climb angle for steady-state flight must be specified by the user within the limits imposed by the engine power. Assuming that the configuration of the aircraft (flap setting, landing gear up or down, etc.) is pre-specified, it can be expected that a unique combination of control inputs and remaining state variables will exist. For the 'Beaver' model, the flap setting δ_f and the engine speed n will be pre-specified. In general, it is not possible to determine any analytical constraints on the remaining control variables δ_e, δ_a, δ_r, and p_z, so these control inputs must be adjusted by the numerical trim algorithm. This is not the case for all state variables.

The three states which define the position of the aircraft (x_e, y_e, and H) can temporarily be eliminated from consideration, because the only relevant component of the position vector is the (initial) altitude H, which can be pre-specified. For steady translational flight, the state variables φ, p, q, and r are identically zero and ψ can be selected freely by the user. This leaves V, α, β, and θ to be considered. The sideslip angle β must be adjusted by the trim algorithm to zero out the sideforce F_y, which leaves V, α, and θ. It is customary to impose a flight-path angle constraint on the steady-state condition, so finally the variables V and γ remain to be specified by the user. In the next section, a general rate-of-climb constraint which allows non-zero values of the roll angle will be given.

In steady-state turning flight, φ, p, q, and r will differ from zero. The turn can be specified by the yaw rate $\dot{\psi}$ or by the turn radius R ($\dot{\psi} = V/R$); the initial heading can still be specified freely. Then p, q, and r can be determined from the kinematic relations given in appendix B (equation (B.60)), given the attitude angles θ and φ. If the roll angle φ is known, the required pitch angle θ can be obtained from the rate-of-climb constraint, which will be treated in the next section. It is possible to specify the roll angle freely, but then in general a significant sideslip angle β will occur which will yield a *skidding turn*. Therefore, a constraint for *coordinated turns* will be included, which will compute the roll and sideslip such that the aircraft is banked at an angle with no component of the aerodynamic side-force Y_a.

4.3.3 The rate-of-climb and turn-coordination constraints

The rate-of-climb and turn-coordination constraints must be solved simultaneously, because the first constraint involves θ while the latter one involves both θ and φ. According to ref.[25], the flight condition must satisfy the following equation:

$$\sin\gamma = a\sin\theta - b\cos\theta \tag{4.33}$$

where:

$$\begin{aligned} a &= \cos\alpha\cos\beta \\ b &= \sin\varphi\sin\beta + \cos\varphi\sin\alpha\cos\beta \end{aligned} \tag{4.34}$$

Solving for θ, the resulting rate-of-climb constraint is found to be:

$$\tan\theta = \frac{a\,b + \sin\gamma\sqrt{a^2 - \sin^2\gamma + b^2}}{a^2 - \sin^2\gamma}\,, \quad \theta \neq \pm\tfrac{\pi}{2} \tag{4.35}$$

The coordinated turn constraint can be written as:

$$\sin\varphi = G\cos\theta\,(\sin\alpha\tan\theta + \cos\alpha\cos\varphi) \tag{4.36}$$

with $G = \dot{\psi}V/g_0$ (see ref.[25]). This equation must be used in conjunction with (4.35) to trim the aircraft for turning flight with a specified rate-of-climb. If the equations (4.35) and (4.36) are solved simultaneously, the only remaining variables to be adjusted by the numerical trim

algorithm are the angle of attack and sideslip angle and the control inputs. According to ref.[25] the simultaneous solution equals:

$$\tan\varphi = G\,\frac{\cos\beta}{\cos\alpha}\,\frac{(\tilde{a}-\tilde{b}^2)+\tilde{b}\tan\alpha\sqrt{\tilde{c}\,(1-\tilde{b}^2)+G^2\sin^2\beta}}{\tilde{a}^2-\tilde{b}^2\,(1+\tilde{c}\tan^2\alpha)} \tag{4.37}$$

where:

$$\begin{aligned}\tilde{a} &= 1-G\tan\alpha\sin\beta\\ \tilde{b} &= \frac{\sin\gamma}{\cos\beta}\\ \tilde{c} &= 1+G^2\cos^2\beta\end{aligned} \tag{4.38}$$

The value of φ given by equation (4.37) can be substituted in (4.35) to solve for θ. For skidding turns φ can be selected freely by the user, so then only the rate-of-climb constraint remains to be solved. When the flight-path angle is zero, equation (4.37) reduces to:

$$\tan\varphi = \frac{G\cos\beta}{\cos\alpha - G\sin\alpha\sin\beta} \tag{4.39}$$

With these flight-path constraints we can develop a general aircraft-trim algorithm which will be described next.

4.3.4 The resulting steady-state trimmed-flight algorithm

The trim algorithm determines steady-state flight conditions by searching for the state and control vectors for which the state derivatives $\dot{V}$, $\dot{\alpha}$, $\dot{\beta}$, $\dot{p}$, $\dot{q}$, and $\dot{r}$ are identically zero. This is realized in practice by applying a numerical minimization routine to a scalar *cost function* J, which equals:

$$J = c_1\dot{V}^2 + c_2\dot{\alpha}^2 + c_3\dot{\beta}^2 + c_4\dot{p}^2 + c_5\dot{q}^2 + c_6\dot{r}^2 \tag{4.40}$$

where c_i, $i \in \{1,2,\ldots,6\}$, are weighting constants. According to ref.[25], the *Simplex* algorithm usually performs well as minimization algorithm for the aircraft-trim problem. Figure 4.9 shows the resulting trim algorithm. First the flight condition must be specified. The trim program must make an initial guess for the independent state variables and the control variables which will be adjusted during the trim process. Next, the minimization routine which searches for the values of $\mathbf{x}$ and $\mathbf{u}$ for which the cost function J is minimal will be started. The elements of these vectors which are adjusted by the minimization routine or the constraints are updated for each iteration step. The state equations are then evaluated for the new values of $\mathbf{u}$ and $\mathbf{x}$ to find the time-derivative of the state-vector, $\dot{\mathbf{x}}$. Substituting the results in equation (4.40) yields the new value of J, which is returned to the minimization routine. A stop criterion which depends upon the change of J between two iterations is used to decide when to finish the minimization procedure. Also, the maximum number of iterations is limited so the process will stop if no minimum can be found.

4.4 The linearization facility

FDC 1.2 contains the linearization utility ACLIN, which can be used to extract a linearized aircraft model from the graphical, non-linear SIMULINK implementation of the aircraft dynamics. This utility calls the SIMULINK program LINMOD for the actual linearization process. This section will briefly describe the theoretical backgrounds of this linearization process.

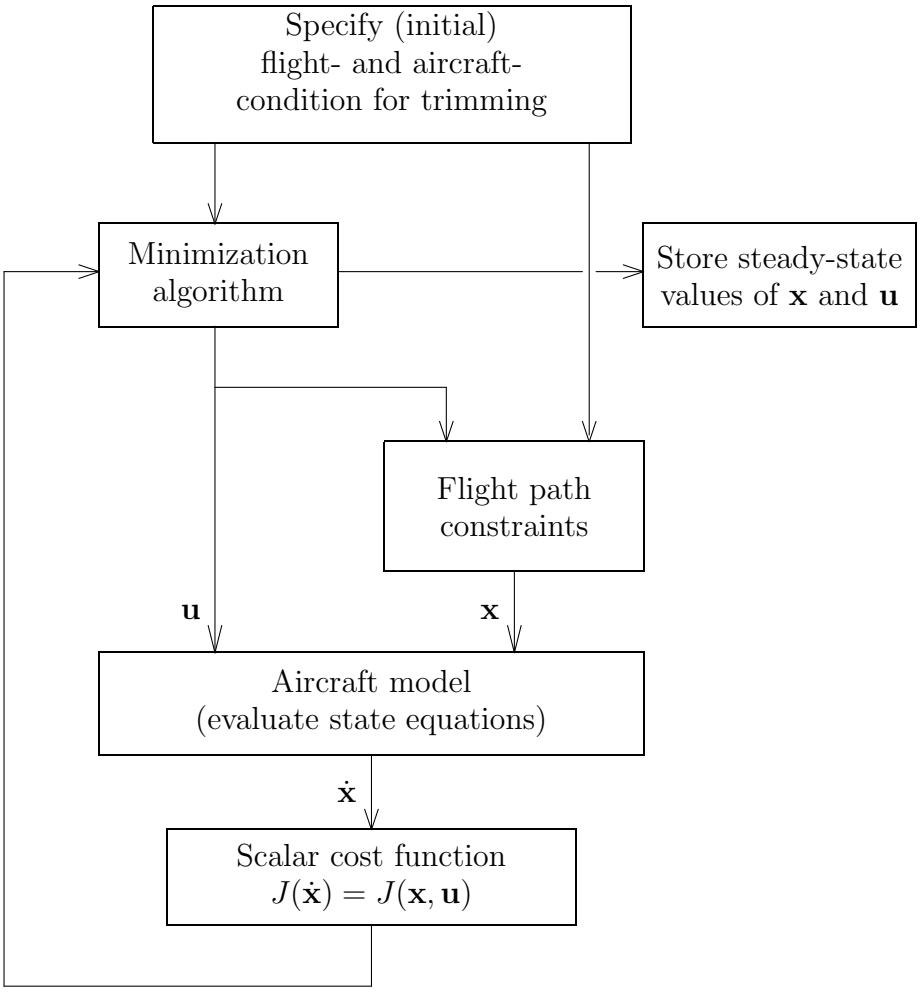


Figure 4.9: Trim algorithm from FDC 1.2

First of all, consider the non-linear state equation:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\,\mathbf{x}(t), \mathbf{u}(t)\,) \tag{4.41}$$

This expression can be linearized around a certain *operating point* $(\mathbf{x}_0, \mathbf{u}_0)$:

$$\dot{\mathbf{x}}(t) \approx \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x} - \mathbf{x}_0) + \dot{\mathbf{x}}_0 \tag{4.42}$$

where $\dot{\mathbf{x}}_0 = \mathbf{f}(\,\mathbf{x}_0, \mathbf{u}_0\,)$. Moving $\dot{\mathbf{x}}_0$ to the left-hand side of the equation yields:

$$\dot{\mathbf{x}} - \dot{\mathbf{x}}_0 = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x} - \mathbf{x}_0) \tag{4.43}$$

Now define:

$$\begin{aligned}
\mathbf{x}' &= \mathbf{x} - \mathbf{x}_0, \quad \text{a vector of length } n \\
\mathbf{u}' &= \mathbf{u} - \mathbf{u}_0, \quad \text{a vector of length } m \\
A &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \\
B &= \frac{\partial \mathbf{f}}{\partial \mathbf{u}}
\end{aligned}$$

Substituting these expressions in equation (4.43) yields the small-perturbations equation:

$$\dot{\mathbf{x}}' = A\mathbf{x}' + B\mathbf{u}' \tag{4.44}$$

which is the desired linear state equation. The task of a numerical linearization algorithm is to determine the elements of the matrices A and B. The matrix A can be written out as follows:

$$\begin{aligned}
A &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \approx \begin{bmatrix} \frac{\Delta f_1}{\Delta x_1} & \cdots & \frac{\Delta f_1}{\Delta x_n} \\ \vdots & & \vdots \\ \frac{\Delta f_n}{\Delta x_1} & \cdots & \frac{\Delta f_n}{\Delta x_n} \end{bmatrix} = \\
&= \begin{bmatrix} \frac{f_1(\mathbf{x}_0+\Delta\mathbf{x}_1,\mathbf{u}_0)-f_1(\mathbf{x}_0,\mathbf{u}_0)}{\Delta x_1} & \cdots & \frac{f_1(\mathbf{x}_0+\Delta\mathbf{x}_n,\mathbf{u}_0)-f_1(\mathbf{x}_0,\mathbf{u}_0)}{\Delta x_n} \\ \vdots & & \\ \frac{f_n(\mathbf{x}_0+\Delta\mathbf{x}_1,\mathbf{u}_0)-f_n(\mathbf{x}_0,\mathbf{u}_0)}{\Delta x_1} & \cdots & \frac{f_n(\mathbf{x}_0+\Delta\mathbf{x}_n,\mathbf{u}_0)-f_n(\mathbf{x}_0,\mathbf{u}_0)}{\Delta x_n} \end{bmatrix}
\end{aligned} \tag{4.45}$$

with:

$$\Delta\mathbf{x}_i = \Delta x_i \cdot \begin{bmatrix} \delta_{i,1} \\ \delta_{i,2} \\ \vdots \\ \delta_{i,n} \end{bmatrix} \qquad \text{with:} \quad \delta_{i,j} = \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases} \tag{4.46}$$

The *columns* of A can be gathered in vectors, yielding:

$$\begin{aligned}
A &= \begin{bmatrix} \frac{\mathbf{f}(\mathbf{x}_0+\Delta\mathbf{x}_1,\mathbf{u}_0)-\mathbf{f}(\mathbf{x}_0,\mathbf{u}_0)}{\Delta x_1} & \cdots & \frac{\mathbf{f}(\mathbf{x}_0+\Delta\mathbf{x}_n,\mathbf{u}_0)-\mathbf{f}(\mathbf{x}_0,\mathbf{u}_0)}{\Delta x_n} \end{bmatrix} = \\
&= \begin{bmatrix} \frac{\dot{\mathbf{x}}_{x_1}-\dot{\mathbf{x}}_0}{\Delta x_1} & \cdots & \frac{\dot{\mathbf{x}}_{x_n}-\dot{\mathbf{x}}_0}{\Delta x_n} \end{bmatrix}
\end{aligned} \tag{4.47}$$

Obviously, in this equation we define:

$$\dot{\mathbf{x}}_{x_i} \equiv \mathbf{f}(\mathbf{x}_0 + \Delta\mathbf{x}_i, \mathbf{u}_0) \tag{4.48}$$

which represents the output from the state equation (4.41) in the operating point with the i^{th} element of the *state* vector being perturbed by the amount Δx_i. If this perturbation is chosen properly, it is now easy to determine an approximation of the matrix A by subsequently computing its columns ($i = 1, \ldots, n$).

In a similar way, the matrix B can be derived. This time, the elements of the input vector $\mathbf{u}$ need to be perturbed. The resulting approximation of B then equals:

$$\begin{aligned} B &= \left[\begin{array}{ccc} \frac{\mathbf{f}(\mathbf{x}_0,\mathbf{u}_0+\Delta\mathbf{u}_1)-\mathbf{f}(\mathbf{x}_0,\mathbf{u}_0)}{\Delta u_1} & \cdots & \frac{\mathbf{f}(\mathbf{x}_0,\mathbf{u}_0+\Delta\mathbf{u}_m)-\mathbf{f}(\mathbf{x}_0,\mathbf{u}_0)}{\Delta u_m} \end{array} \right] = \\ &= \left[\begin{array}{ccc} \frac{\dot{\mathbf{x}}_{u_1}-\dot{\mathbf{x}}_0}{\Delta u_1} & \cdots & \frac{\dot{\mathbf{x}}_{u_m}-\dot{\mathbf{x}}_0}{\Delta u_m} \end{array} \right] \end{aligned} \tag{4.49}$$

In this equation the columns of B were gathered in vectors. The following definitions were applied:

$$\Delta\mathbf{u}_i = \Delta u_i \cdot \left[\begin{array}{c} \delta_{i,1} \\ \delta_{i,2} \\ \vdots \\ \delta_{i,m} \end{array} \right] \qquad \text{with:} \quad \delta_{i,j} = \left\{ \begin{array}{l} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{array} \right. \tag{4.50}$$

and:

$$\dot{\mathbf{x}}_{u_i} \equiv \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0 + \Delta\mathbf{u}_i) \tag{4.51}$$

Equation (4.51) represents the output from the state equation (4.41) in the operating point with the i^{th} element of the *input* vector perturbed by the amount Δu_i.

The non-linear *output* equation equals:

$$\dot{\mathbf{y}}(t) = \mathbf{g}(\,\mathbf{x}(t), \mathbf{u}(t)\,) \tag{4.52}$$

This equation can also be linearized around the operating point $(\mathbf{x}_0, \mathbf{u}_0)$:

$$\dot{\mathbf{y}}(t) \approx \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}_0) + \frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{x} - \mathbf{x}_0) + \mathbf{y}_0 \tag{4.53}$$

where $\mathbf{y}_0 = \mathbf{g}(\,\mathbf{x}_0, \mathbf{u}_0\,)$. This equation can be developed into the following small-perturbation equation for the output vector $\mathbf{y}$:

$$\mathbf{y}' \equiv \mathbf{y} - \mathbf{y}_0 = C\mathbf{x}' + D\mathbf{u}' \tag{4.54}$$

with:

$$\begin{aligned} C &= \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \\ D &= \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \end{aligned}$$

C and D can be approximated using the same procedure as for the matrices A and B from the state equation. The sizes of these matrices are:

$$\begin{aligned} A&: \quad (n \times n) \\ B&: \quad (n \times m) \\ C&: \quad (m \times n) \\ D&: \quad (m \times m) \end{aligned}$$

This method has been implemented in the SIMULINK routine LINMOD. There is also a routine LINMOD2, which uses a more advanced version of this method, but the basic principles still apply.

Chapter 5

FDC implementation of the aircraft model

5.1 General structure of the aircraft model

Recall figure 3.2 from chapter 3. This figure provides the general framework for the SIMULINK implementation of the non-linear aircraft model. The modular structure from figure 3.2 is clearly reflected in the SIMULINK block-diagram from figure 5.1 which shows the simulation model of the 'Beaver' aircraft. From left to right we see the atmosphere and airdata equations in the block Airdata Group, the different contributions to the external forces and moments in the blocks Aerodynamics Group, Engine Group, Gravity, and Fwind, a summation of the external forces and moments in the block FMsort, and the equations of motion themselves in the block Aircraft Equations of Motion. This latter block contains the twelve state equations, some correction blocks, and an integrator which determines the time-trajectories of the state variables. Some output variables which are not necessary for solving the equations of motion have been gathered in the subsystem Additional Outputs. The block Hlpfcn in the feedback-loop computes some often used sines and cosines of angular values from the state vector. On the left-hand side of the block-diagram, we see the external inputvectors $\mathbf{u}_{aero}$ and $\mathbf{u}_{prop}$, which contain inputs to the aerodynamic and engine models, and $\mathbf{u}_{wind}$, which contains wind velocity components along the aircraft's body-axes, including contributions from atmospheric turbulence. On the right-hand side of the diagram, all results are gathered in a number of output vectors.

The block-diagram from figure 5.1 thus provides the general picture of the aircraft dynamics, but it can't be used as first level of the SIMULINK model because it does not contain all necessary interfacing functions. Figure 5.2 shows the top-level of the SIMULINK implementation of the aircraft model, which fulfills the input/output functions of the system. In this level, all twelve scalar input variables are gathered in the three input vectors from figure 5.1 by means of Mux blocks. Since SIMULINK does not allow the use of input *vectors* in the top-level of a system, twelve scalar Inport blocks were needed. Figure 5.1 shows the contents of the central subsystem Beaver dynamics and output equations. On the left-hand side of this subsystem the outputs are combined in one output vector which is sent to the MATLAB workspace. The most important results are extracted from this vector by means of Demux blocks and connected to (scalar) Outport blocks, thus enabling us to connect the aircraft model to other SIMULINK systems, such as an autopilot model. Again, due to limitations of SIMULINK itself, it is not possible to connect

output *vectors* to these Outport blocks.[1]

There are three To Workspace blocks, one of which is connected to a Clock in order to create a time-axis for the analysis of simulation results. They blocks send all input signals to the matrix *In*, all output signals to the matrix *Out*, and a time-axis to the vector *time* within the MATLAB workspace. After running a simulation, these matrices can be examined, and used for plotting results. Enter `type inputs.hlp` or `type outputs.hlp` for on-line help, or double-click the blue-colored Mux blocks with shadow borders in the top-level of the aircraft model[2]. See also the descriptions of RESULTS and RESPLOT in chapter 9. The definitions of the matrices *In* and *Out* are given in section E.2 of appendix E. The To Workspace block that is connected to the Clock creates a time-vector *time*, which is needed for plotting purposes because SIMULINK integrators use variable step-widths. Storage of the input signals is useful in cases where they are generated by other SIMULINK systems, e.g. control laws or external disturbance blocks.

The input signals which enter the system through the Inport blocks in the top-level of a SIMULINK system will be called *S-function inputs* in the remainder of this report. Similarly, the outputs which leave the system through the Outport blocks in the top-level will be called *S-function outputs* from now on. For practical reasons, only a small subset of the output signals have been connected to the Outport blocks in the top-level of the system, but it is relatively easy to change this set by adding new Outport blocks and making the right connections. For many purposes the current set of outputs will be sufficient. Contrary to the S-function outputs, the To Workspace blocks take into account *all* output signals. These results will be called *simulation results* from now on. So S-function outputs are intended to be used by other systems while simulation results are sent to the MATLAB workspace for further analysis *after* finishing the simulations. In this chapter, the top-level from the aircraft model Beaver shown in figure 5.2 will be denoted as Level 1, while the second level, shown in figure 5.1, will be called Level 2. Level 1 can be opened by typing `beaver` at the MATLAB command-line. Zooming in to deeper levels is possible by double-clicking the subsystems with the mouse.

The masked subsystem blocks from the system Beaver have also been sorted in the following sublibraries of the main model library FDCLIB:

1. airdata, atmosphere,
2. aerodynamics,
3. engine forces and moments,
4. gravity and wind forces,
5. equations of motion,
6. other (output-) equations.

These sublibraries may be useful if you plan to use the system Beaver as a template for the implementation of other aircraft models. In order to illustrate the connections between the different masked subsystem blocks, FDCLIB also contains the unmasked subsystems from Beaver. The model library can be opened by typing `fdclib` at the command-line.

[1] Luckily, SIMULINK does not have any restrictions with regard to the use of vector inputs and outputs for *subsystems* such as the diagram from figure 5.1, so this rather confusing *Muxing* and *Demuxing* is required in the top-levels of graphical systems only.

[2] All blocks with a blue foreground and white background having shadow borders are linked to an on-line help text that will be displayed in the MATLAB command window when such a block is double-clicked. For more information about the color conventions from FDC 1.2, enter `type colors.hlp` at the command-line.

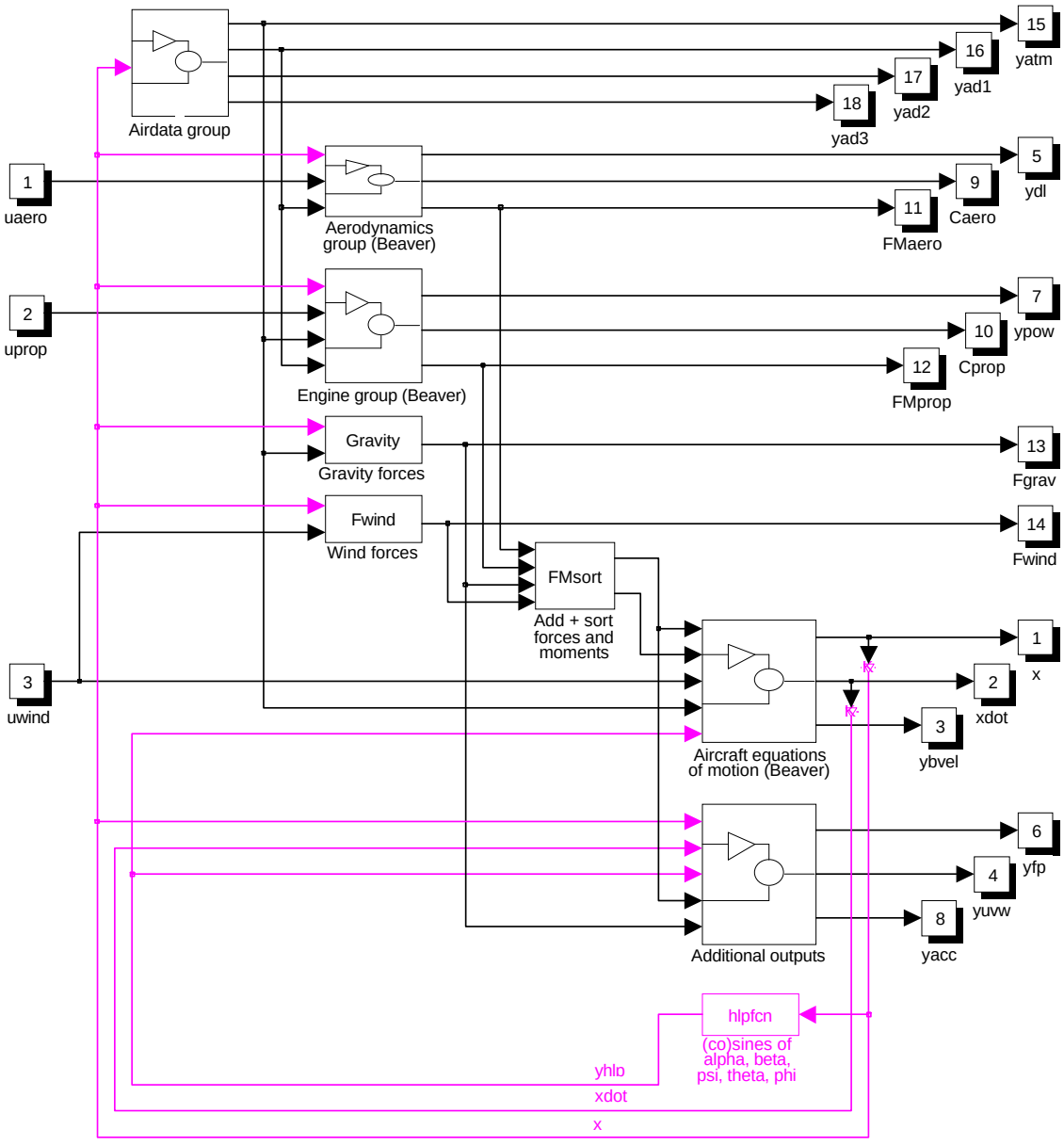


Figure 5.1: Block-diagram of level 2 of the aircraft model Beaver

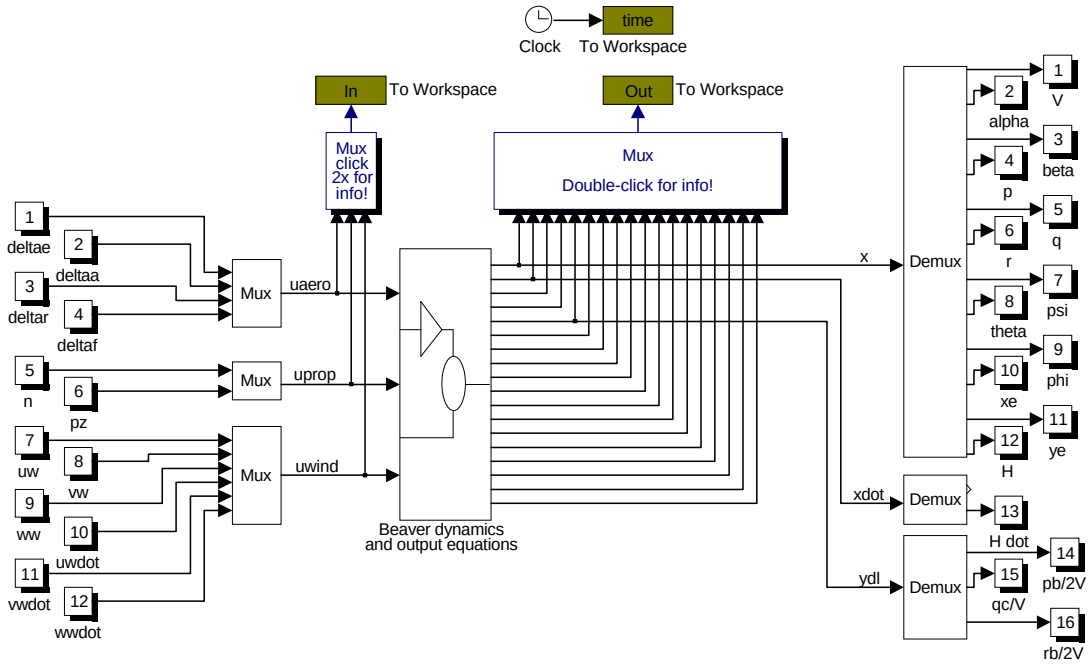


Figure 5.2: Block-diagram of top-level of the aircraft model Beaver

5.2 Conventions used in the Reference Guide chapters

The remainder of this chapter contains a systematical description of the SIMULINK implementation of the aircraft model from chapter 3. Chapter 6 describes the implementation of the wind and turbulence models, chapter 7 describes the implementation of the radio-navigation models, and chapter 8 gives an overview of the SIMULINK implementation of the analytical tools for the FDC toolbox. With regard to the description of the graphical SIMULINK systems these chapters all use the same conventions. For each block in these systems the basic equations, inputvariables, outputvariables, parameters, and connections to other blocks will be listed. If a block contains other subsystems, a list of those subsystems will be given instead of the equations. For each separate chapter, the blocks will be treated in alphabetical order. The name of the subsystems is shown in the upper left corner of the pages; the upper right corner shows the position of the subsystem within the aircraft model (this chapter) or the model library (chapters 6 and 7). For instance, the description of the block 12 ODEs from the system Beaver has the following header:

12 ODEs *Level 1 / Level 2 / Aircraft Equations of Motion / 12 ODEs*

which indicates that the subsystem 12 ODEs is contained in the third level of the SIMULINK system Beaver. This subsystem can be accessed by opening the system Beaver (which will reveal Level 1) and then double-clicking the blocks Beaver dynamics and output equations (to reveal Level 2), Aircraft equations of Motion, and 12 ODEs, respectively. If you try to zoom in further, for instance by double-clicking the block Eulerdot, you will see that the contents of this block have been hidden from the user by means of the *Masking* function of SIMULINK. The individual equations can be accessed only if the block is *Unmasked* first. This may seem somewhat inconvenient, but it facilitates the re-usability of blocks from the aircraft model and hence the overall modularity of the system, and it protects the system from being inadvertently disrupted at the level of the basic equations, which is the most difficult level to debug. Also, this black-box approach makes it theoretically possible to implement separate blocks by means of MATLAB, C

or FORTRAN subroutines without changing their appearance within the overall system. Since it is assumed here that the user has direct access to the SIMULINK models, this chapter does not contain any pictures of masked subsystems. If you want to analyze the contents of masked subsystems you must therefore open the respective system on your own computer.

The **connections** paragraphs from the alphabetical block-descriptions list the connections between different subsystems, ordered by their input and output signals. For the description of the blocks from the non-linear aircraft model, the exact level of the block within the system Beaver is shown between brackets. For instance, the signal $\mathbf{y}_{atm}$ is sent from the block Atmosph to the block Airdata 1, which both are part of the subsystem Airdata Group. In the **connections** section from Airdata 1 this is written as: "$\mathbf{y}_{atm}$ comes from the block Atmosph (Airdata Group)". The first two path-items, being Level 1 and Level 2, have not been mentioned here to shorten the notations, although they *are* mentioned in the header lines of the block-descriptions.

5.2.1 On-line help for FCD 1.2

Each graphical subsystem contains a blue-colored title-block which lists the block-name, the author, and the month in which it was made. Shadow borders around these blocks mean that they will reveal more information about the subsystem when being double-clicked. This is done by listing the contents of the appropriate help-file in the MATLAB command-window. The help-files are contained in the FDC subdirectory HELP. They can be read also with an ASCII viewer or editor, or by typing `type blkname.hlp` at the MATLAB command-line, where *blkname* is the name of the help-file, which is usually directly related to the name of the subsystem block. All reference chapters from this report will refer to the appropriate help-text for each subsystem block, whenever on-line help is available. Moreover, if you double-click a masked subsystem blocks and then click on the *help* button, the name of the appropriate help-file will be displayed. On-line help for the MATLAB subroutines from FDC 1.2 can be viewed by typing `help progname` at the command-line, where *progname* is the name of the MATLAB routine of interest; e.g. type `help actrim` for more information about the aircraft trim routine. Of course you can also view the contents of the M-files with an ASCII viewer or editor.

Some particularly useful help-texts are:

- INPUTS.HLP, contains information about the inputs to the system Beaver,
- OUTPUTS.HLP, contains information about the outputs from the system Beaver,
- COLORS.HLP, gives an overview about the color conventions used in graphical systems from FDC 1.2,
- EQMOTION.HLP, describes the subsystem Aircraft Equations of Motion from the non-linear aircraft model,
- LEVEL1.HLP, contains information about the first level of the system Beaver,
- LEVEL2.HLP, contains information about the second level of the system Beaver,
- FDCLIB.HLP, gives a short description about the main FDC library FDCLIB.

The next pages (pp. 74 to 108) describe the blocks and subsystems from the aircraft model Beaver in alphabetical order.

Note: appendix A contains a list of symbols, reference frames, abbreviations, and other definitions; appendix E lists all acronyms and variable names used by the FDC models and tools.

12 ODEs

Type

Non-masked subsystem, aircraft-independent, essential for solving state equations.

Description

The subsystem 12 ODEs contains the twelve non-linear Ordinary Differential Equations describing the aircraft dynamics. These equations of motion are valid for all rigid bodies, assuming a flat, non-rotating Earth (see appendix B). The time-derivatives of the twelve state variables are non-linear functions of the state variables themselves and the external forces and moments, which, in turn, depend upon the state variables and external inputs.

Subsystems/masked blocks

There are four masked subsystem blocks contained in the subsystem 12 ODEs:

Vabdot:	computes time-derivatives of the true airspeed, angle of attack, and sideslip angle
pqrdot:	computes time-derivatives of the angular velocities along the body-axes of the aircraft
Eulerdot:	computes time-derivatives of the Euler angles
xyHdot:	computes time-derivatives of the coordinates and the altitude above sea-level

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{M}_{tot} = [\ L\ M\ N\ ]^T$ — *total external moments, Mtot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

$\mathbf{y}^*_{bvel} = [\ u + u_w\ v + v_w\ w + w_w\ ]^T$ — *body-axes velocity components plus wind, ybvel**

Outputs

$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ — *time-derivative of state vector, xdot*

Note: the vector $\dot{\mathbf{x}}$ that leaves the subsystem 12 ODEs must be corrected for the influence of $\dot{\beta}$ upon the aerodynamic side force Y_a. This correction is made in the block xdotcorr; see the accompanying description for more details.

Parameters

The masked subsystem Vabdot needs the parameter matrix *GM1*; the masked subsystem pqrdot needs the matrix *GM2*. Use the routine MODBUILD (section 9.2) for defining these parameters, or use LOADER (section 9.3.1) for loading them into the MATLAB workspace. See appendix D for the definitions of *GM1* and *GM2*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ come from FMsort; $\mathbf{y}_{hlp}$ comes from Hlpfcn; $\mathbf{y}^*_{bvel}$ is the sum of the output from uvw (Equations of Motion) and the wind velocity components from the external input vector $\mathbf{u}_{wind}$

out: $\dot{\mathbf{x}}$ (not corrected for implicit nature of the $\dot{\beta}$-equation) is connected to the block xdotcorr (Equations of Motion)

Enter `type 12odes.hlp` at the command-line for on-line help. □

Accel

Type

Masked subsystem, aircraft-independent, not essential for solving state equations.

Description

The masked subsystem block Accel computes some accelerations and specific forces (outputs of accelerometers) in the aircraft's center of gravity. These output variables are useful for many purposes in AFCS design, e.g. turn-coordination by means of feedback of the specific force along the Y_B axis, or manoeuvre load limiting. However, these variables are *not* used for actually solving the equations of motion themselves! This block does not compute accelerations for points outside the center of gravity, but similar blocks can be implemented easily.

Equations

As outlined in section 3.2.5 the kinematic accelerations in the aircraft's body-axes are equal to:

- Kinematic accelerations $a_{x,k}$, $a_{y,k}$, and $a_{z,k}$ along the body-axes, measured in the vehicle's c.g.:

$$\begin{aligned} a_{x,k} &= \frac{1}{g_0}(\dot{u} + qw - rv) = \frac{F_x}{W} \\ a_{y,k} &= \frac{1}{g_0}(\dot{v} + ru - pw) = \frac{F_y}{W} \\ a_{z,k} &= \frac{1}{g_0}(\dot{w} + pv - qu) = \frac{F_z}{W} \end{aligned}$$

These accelerations are measured in units of g which explains the division by g_0. $W = mg$ is the total weight of the aircraft, measured in $[N]$. Note: the kinematic accelerations differ from the body-axes velocity rates $\dot{u}$, $\dot{v}$, and $\dot{w}$ because the latter do not contain the angular and translational velocity cross-product terms; see section 3.2.5.

- Outputs of body-axis accelerometers (specific forces) A_x, A_y, and A_z at the c.g., $[g]$:

$$\begin{aligned} A_x &= a_{x,k} + \sin\theta &&= (F_x - X_{gr})/W \\ A_y &= a_{y,k} - \cos\theta\sin\varphi &&= (F_y - Y_{gr})/W \\ A_z &= a_{z,k} + \cos\theta\cos\varphi &&= (F_z - Z_{gr})/W \end{aligned}$$

Inputs

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{F}_{grav} = [\ X_{gr}\ Y_{gr}\ Z_{gr}\ ]^T$ — *gravity forces, Fgrav*

Outputs

$\mathbf{y}_{acc} = [\ A_x\ A_y\ A_z\ a_{x,k}\ a_{y,k}\ a_{z,k}\ ]^T$ — *specific forces and accelerations, yacc*

Parameters

Accel needs the parameter vector *GM1* to extract the mass m of the aircraft (the mass has been implemented as a *parameter*, i.e. it is assumed that it is constant during the relative short time intervals considered). Use MODBUILD (section 9.2) to define *GM1*, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{F}_{tot}$ comes from the block FMsort; $\mathbf{F}_{grav}$ comes from Gravity

out: $\mathbf{y}_{acc}$ is not used by any other block in the system Beaver

Enter `type accel.hlp` at the command-line for on-line help. □

Additional Outputs

Type

Non-masked subsystem, aircraft-independent, not essential for solving state equations.

Description

The non-masked subsystem Additional Outputs contains output equations which are not needed for the solution of the state equations and which can not logically be included into another non-masked subsystem from the non-linear aircraft model. The user is free to add or delete 'additional output blocks' to/from this subsystem, but it is not allowed to use output signals from this subsystem as inputs to the forces and moments blocks or the equations of motion themselves. Any function of the states, their time-derivatives, external inputs and/or disturbances, or outputs from other subsystems within the SIMULINK-implementation of the aircraft model may be included in the subsystem Additional Outputs.[1]

Subsystems/masked blocks

Currently, there are three masked subsystem blocks contained in the subsystem Additional Outputs:

Accel:	computes specific forces (outputs of accelerometers) and kinematic accelerations
Flpath:	computes some flight-path variables
uvwdot:	computes the time-derivatives of the body-axes velocity components

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ — *time-derivative of state vector, xdot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{F}_{grav} = [\ X_{gr}\ Y_{gr}\ Z_{gr}\ ]^T$ — *gravity forces, Fgrav*

Outputs

$\mathbf{y}_{fp} = [\ \gamma\ fpa\ \chi\ \Phi\ ]^T$ — *flight-path variables, yfp*

$\mathbf{y}_{uvw} = [\ \dot{u}\ \dot{v}\ \dot{w}\ ]^T$ — *time-derivatives of body-axes velocities, yuvw*

$\mathbf{y}_{acc} = [\ A_x\ A_y\ A_z\ a_{x,k}\ a_{y,k}\ a_{z,k}\ ]^T$ — *specific forces and accelerations, yacc*

Parameters

The masked subsystem Accel needs the parameter vector *GM1*; the masked subsystem Flpath needs the initial value of the state vector, *xinco*. Use the routine MODBUILD (section 9.2) for defining *GM1*, or use LOADER (section 9.3.1) for loading this parameter vector into the MATLAB workspace. Use ACTRIM (section 8.2) for defining a trimmed initial flight condition, or use INCOLOAD (section 9.3.2) to load one to the workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\dot{\mathbf{x}}$ comes from xfix (Equations of Motion); $\mathbf{y}_{hlp}$ comes from Hlpfcn; $\mathbf{F}_{tot}$ comes from FMsort; $\mathbf{F}_{grav}$ comes from Gravity

out: $\mathbf{y}_{fp}$ which leaves the block Flightpath (Additional Outputs), $\mathbf{y}_{uvw}$ from uvwdot (Additional Outputs), and $\mathbf{y}_{acc}$ from Accel (Additional Outputs) are not connected to any other block in the system Beaver

Enter `type moreouts.hlp` at the command-line for on-line help. □

[1] Since the outputs from the subsystem Additional Outputs are not involved in the solution of the state equations themselves, the use of time-derivatives of state variables within this subsystem does not yield *algebraic loops*. However, care must be taken when feeding back outputs from this subsystem to the input-side of the aircraft model, e.g. in autopilot control loops, because that may yield an algebraic loop which may be too complicated for SIMULINK to solve. See also section 4.2.7.

Aerodynamics Group (Beaver)

Type

Non-masked subsystem, aircraft-dependent, essential for solving state equations.

Description

The subsystem Aerodynamics Group contains blocks for computing aerodynamic forces and moments which act upon the aircraft under consideration. In this case the aerodynamic model of the DHC-2 'Beaver', described in ref.[26], has been implemented. The subsystem Aerodynamics Group does not compute the contributions to the forces and moments due to the slipstream-effects of the propeller; this is done in the subsystem Engine Group.

Subsystems/masked blocks

There are three masked subsystem blocks contained in the subsystem Aerodynamics Group (Beaver):

- Aeromod: computes the non-dimensional aerodynamic force and moment coefficients for the aircraft under consideration (currently Aeromod contains the aerodynamic model of the 'Beaver' from ref.[26])
- Dimless: computes non-dimensional angular velocities
- FMdims: converts the non-dimensional force and moment coefficients to dimensional forces and moments

Since the block Aeromod (Beaver) is aircraft-dependent, it needs to be replaced if a model of another aircraft is implemented. However, if the definitions of the non-dimensional angular velocities are different in another aircraft model, it may be more convenient to replace the whole subsystem Aerodynamic Group instead. Building a large library of aerodynamic models for different aircraft in SIMULINK makes it easy to simulate many types of aircraft with a minimum of programming efforts.

Inputs

$\mathbf{x}$	$= [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$	*state vector, x*
$\mathbf{u}_{aero}$	$= [\ \delta_e\ \delta_a\ \delta_r\ \delta_f\ ]^T$	*aerodynamic control inputs, uaero*
$\mathbf{y}_{ad1}$	$= [\ a\ M\ q_{dyn}\ ]^T$	*basic airdata variables, yad1*

Outputs

$\mathbf{y}_{dl}$	$= [\ \frac{pb}{2V}\ \frac{q\overline{c}}{V}\ \frac{rb}{2V}\ ]^T$	*non-dimensional angular velocities, ydl*
$\mathbf{C}_{aero}$	$= [\ C_{X_a}\ C_{Y_a}\ C_{Z_a}\ C_{l_a}\ C_{m_a}\ C_{n_a}\ ]^T$	*aerodynamic force and moment coefficients, Caero*
$\mathbf{FM}_{aero}$	$= [\ X_a\ Y_a\ Z_a\ L_a\ M_a\ N_a\ ]^T$	*dimensional aerodynamic forces and moments, FMaero*

Parameters

The masked subsystem block Dimless and the block FMdims need the parameter-vector *GM1*; the block Aeromod (Beaver) needs the parameter-matrix *AM*. Use MODBUILD (section 9.2) for defining these parameters, or use LOADER (section 9.3.1) for loading them into the MATLAB workspace. See appendix D for the definitions of *AM* and *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{u}_{aero}$ is an external input vector with aerodynamic control inputs; $\mathbf{y}_{ad1}$ comes from Airdata1 (Airdata Group);

out: $\mathbf{y}_{dl}$ from Dimless (Aerodynamics Group) is used by the block Aeromod (Aerodynamics Group); $\mathbf{C}_{aero}$ from Aeromod (Aerodynamics Group) is used by FMdims (Aerodynamics Group); $\mathbf{FM}_{aero}$ from FMdims (Aerodynamics Group) is used by the block FMsort

Enter `type aerogrp.hlp` at the command-line for on-line help. □

Aeromod (Beaver)

Type

Masked subsystem, aircraft-dependent, essential for solving state equations.

Description

The masked subsystem Aeromod (Beaver) contains the aerodynamic model for the DHC-2 'Beaver' aircraft, described in ref.[26]. For this aircraft, the aerodynamic forces and moments are expressed in terms of non-linear polynomial functions of state variables and external aerodynamic control inputs. If a model of another aircraft is implemented within the SIMULINK framework of the 'Beaver' model, this masked subsystem block must be updated, which shouldn't be too difficult due to its *black-box* structure. In many aircraft models, aerodynamic forces and moments are computed by reading out data from multi-dimensional tables, which can be implemented by means of Lookup Table blocks from the standard SIMULINK-libraries. Note: the aerodynamic model of the 'Beaver' as implemented within Aeromod does not take into account contributions to the forces and moments due to the propeller slipstream. This is done in the masked subsystem block Engmod within the subsystem Engine Group.

Equations

The aerodynamic model of the 'Beaver' expresses the aerodynamic force and moment coefficients in terms of non-linear polynomial functions of the state variables and aerodynamic control inputs. The model includes longitudinal-lateral cross-coupling effects, as well as unsteady aerodynamics, but it does not take into account the influence of compressibility, as airspeed is assumed to be low. See ref.[26] for more details.

- Aerodynamic force and moment coefficients measured in the body-fixed reference frame:

$$
\begin{aligned}
C_{X_a} &= C_{X_0} + C_{X_\alpha}\alpha + C_{X_{\alpha^2}}\alpha^2 + C_{X_{\alpha^3}}\alpha^3 + C_{X_q}\frac{q\overline{c}}{V} + C_{X_{\delta_r}}\delta_r + C_{X_{\delta_f}}\delta_f + C_{X_{\alpha\delta_f}}\alpha\delta_f \\
C^*_{Y_a} &= C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{p\,b}{2V} + C_{Y_r}\frac{rb}{2V} + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r + C_{Y_{\delta_r\alpha}}\delta_r\alpha \\
C_{Z_a} &= C_{Z_0} + C_{Z_\alpha}\alpha + C_{Z_{\alpha^3}}\alpha^3 + C_{Z_q}\frac{q\overline{c}}{V} + C_{Z_{\delta_e}}\delta_e + C_{Z_{\delta_e\beta^2}}\delta_e\beta^2 + C_{Z_{\delta_f}}\delta_f + C_{Z_{\alpha\delta_f}}\alpha\delta_f \\
C_{l_a} &= C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{p\,b}{2V} + C_{l_r}\frac{rb}{2V} + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta_r + C_{l_{\delta_a\alpha}}\delta_a\alpha \\
C_{m_a} &= C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\alpha^2}}\alpha^2 + C_{m_q}\frac{q\overline{c}}{V} + C_{m_{\delta_e}}\delta_e + C_{m_{\beta^2}}\beta^2 + C_{m_r}\frac{rb}{2V} + C_{m_{\delta_f}}\delta_f \\
C_{n_a} &= C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{p\,b}{2V} + C_{n_r}\frac{rb}{2V} + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r + C_{n_q}\frac{q\overline{c}}{V} + C_{n_{\beta^3}}\beta^3
\end{aligned}
$$

The values of the stability and control derivatives, i.e. the coefficients of the polynomials, are listed in table C.3. See also the description of the subroutine MODBUILD in chapter 8.

If the block Aeromod (Beaver) is unmasked and double-clicked one can see how the polynomial evaluation has been implemented by means of a multiplication of the vector:

$$
\left[\begin{array}{cccccccccccccccc} 1 & \alpha & \alpha^2 & \alpha^3 & \beta & \beta^2 & \beta^3 & \frac{pb}{2V} & \frac{q\overline{c}}{V} & \frac{rb}{2V} & \delta_e & \delta_a & \delta_r & \alpha\delta_f & \alpha\delta_r & \alpha\delta_a & \delta_e\beta^2 & 0 \end{array}\right]^T
$$

with the constant parameter matrix AM in which the stability and control derivatives of the 'Beaver' are contained. Note that the influence of $\dot{\beta}$ upon C_{Y_a} has been omitted here (the last element of the multiplication vector is zero instead of $\dot{\beta}b/2V$). This was necessary in order to prevent the $\dot{\beta}$ equation from becoming implicit which would yield an algebraic loop in the simulation model (see section 4.2.7). The error which results from neglecting this term is corrected in the block xdotcorr. With this omission of the $\dot{\beta}$-contribution, C_{Y_a} has been denoted as $C^*_{Y_a}$.

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$		*state vector, x*
$\mathbf{u}_{aero} = [\ \delta_e\ \delta_a\ \delta_r\ \delta_f\ ]^T$		*aerodynamic control inputs, uaero*
$\mathbf{y}_{dl} = [\ \frac{pb}{2V}\ \frac{q\overline{c}}{V}\ \frac{rb}{2V}\ ]^T$		*non-dimensional angular velocities, ydl*

Outputs

$\mathbf{C}_{aero} = [\ C_{X_a}\ C^*_{Y_a}\ C_{Z_a}\ C_{l_a}\ C_{m_a}\ C_{n_a}\ ]^T$ *aerodynamic force and moment coefficients, Caero*

Parameters

Aeromod needs the parameter matrix AM for reading out the stability and control coefficients. Use the routine MODBUILD (section 9.2) to define this matrix, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of the matrix AM.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{u}_{aero}$ is an external input vector with aerodynamic control inputs; $\mathbf{y}_{dl}$ comes from Dimless (Aerodynamics Group)

out: $\mathbf{C}_{aero}$ is connected to FMdims (Aerodynamics Group)

Enter `type aeromod.hlp` at the command-line for on-line help. □

Aircraft Equations of Motion (Beaver)

Type

Non-masked subsystem, largely aircraft-independent except for the block xdotcorr, essential for solving the equations of motion.

Description

The subsystem Aircraft Equations of Motion contains the actual non-linear equations of motion in which the time-derivatives of the state variables are determined and an Integrator-block for finding the time-trajectories of the state variables themselves. Currently, this subsystem still contains one aircraft-dependent element, namely xdotcorr which is needed to correct the time-derivative of the sideslip angle for implicitness of the aerodynamic model.

Subsystems/masked blocks

There are two masked subsystems contained in the subsystem Aircraft Equations of Motion (Beaver):

uvw: computes the body-axes velocity components as a function of the true airspeed, angle of attack, and sideslip angle

xdotcorr: makes corrections to the time-derivatives of the state variables in order to take into account the implicit influence of these time-derivatives upon the external (aerodynamic) forces and moments (here xdotcorr corrects the time-derivative of the sideslip angle to account for the contribution of $\dot{\beta}$ itself to the aerodynamic sideforce Y_a)

In addition, Aircraft Equations of Motion contains one non-masked subsystem:

12 ODEs: contains the state equations which express the time-derivatives of the state variables in terms of these states themselves and the external forces and moments

and a standard Integrator-block is included to determine the time-trajectories of the state variables as a function of the time-derivatives and the initial condition.

Inputs

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{M}_{tot} = [\ L\ M\ N\ ]^T$ — *total external moments, Mtot*

$\mathbf{u}_{wind} = [\ u_w\ v_w\ w_w\ \dot{u}_w\ \dot{v}_w\ \dot{w}_w\ ]^T$ — *wind velocity components along body-axes and their time-derivatives, uwind*

$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$ — *basic atmospheric properties, yatm*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ — *time-derivative of state vector, xdot*

$\mathbf{y}_{bvel} = [\ u\ v\ w\ ]^T$ — *body-axes velocity components, ybvel*

Parameters

The masked subsystem block xdotcorr needs the parameter matrix *AM* and the vector *GM1*. The subsystem 12 ODEs needs the parameter vector *GM1* and the matrix *GM2*. Use the routine MODBUILD (section 9.2) for defining these parameters, or use LOADER (section 9.3.1) for loading them into the MATLAB workspace. See appendix D for the definitions of *AM*, *GM1*, and *GM2*. The block Integrator requires the initial value of the state vector, *xinco*, which can be computed with the trim routine ACTRIM or loaded from file with INCOLOAD.

Connections

in: $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ come from FMsort; $\mathbf{u}_{wind}$ is an external input with wind and turbulence velocities and their time-derivatives; $\mathbf{y}_{atm}$ comes from Atmosph (Airdata Group); $\mathbf{y}_{hlp}$ comes from Hlpfcn

out: $\mathbf{x}$ which leaves the block Integrator (Equations of Motion) is connected to many other blocks in the system Beaver; $\dot{\mathbf{x}}$ from xfix is connected to Accel, Flpath, and uvwdot (all from the subsystem Additional Outputs); $\mathbf{y}_{bvel}$ from uvw is connected to xyHdot (Equations of Motion / 12 ODEs)

Enter `type eqmotion.hlp` at the command-line for on-line help. □

Airdata1

Type

Masked subsystem, aircraft-independent, necessary for solving state equations.

Description

The masked subsystem block Airdata1 is used to compute the most important airdata variables which are often necessary for solving the equations of motion. In the 'Beaver' model only the dynamic pressure q_{dyn} is actually needed for solving the state equations, but for faster flying aircraft the Mach number M is needed too. See Atmosph, Airdata2, and Airdata3 for other airdata (-related) equations.

Equations

- Dynamic pressure q_{dyn}, $[kg\,m^{-2}]$:

  $$q_{dyn} = \tfrac{1}{2}\rho V^2$$

- Speed of sound a, $[ms^{-1}]$:

  $$a = \sqrt{\gamma R T}$$

  where $\gamma = 1.4$ = ratio of specific heats of air with constant pressure and constant volume, respectively, and $R = R_a/M_0 = 287.05\ JK^{-1}kg^{-1}$ = specific gas constant of the air ($M_0 = 28.9644\ kg\ kmol^{-1}$ = molecular weight of the air at sea level).

- Mach number M, $[-]$:

  $$M = \frac{V}{a}$$

Inputs

$\mathbf{x}$	$= [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$	*state vector, x*
$\mathbf{y}_{atm}$	$= [\ \rho\ p_s\ T\ \mu\ g\ ]^T$	*basic atmospheric properties, yatm*

Outputs

$\mathbf{y}_{ad1}$	$= [\ a\ M\ q_{dyn}\ ]^T$	*basic airdata variables, yad1*

Parameters

All parameters for Airdata1 are defined within the block itself; Airdata 1 does *not* use parameters from the MATLAB workspace.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{y}_{atm}$ comes from Atmosph (Airdata Group)

out: $\mathbf{y}_{ad1}$ is connected to the blocks Airdata2 and Airdata3 (all from the subsystem Airdata Group)

Enter `type airdata1.hlp` at the command-line for on-line help. □

Airdata2

Level 1 / Level 2 / Airdata Group / Airdata 2

Type

Masked subsystem, aircraft-independent, in general not necessary for solving the equations of motion.

Description

The masked subsystem Airdata2 is used to compute the impact pressure q_c, the calibrated airspeed V_c, and the equivalent airspeed V_e. For most aircraft, these airdata (-related) variables are not necessary for actually solving the equations of motion, but they may be useful for other purposes. For this reason, the block Airdata2 can be deleted from the SIMULINK model without affecting the solutions of the ODEs. The equations from this block are independent of the aircraft under consideration.

Equations

- Impact pressure q_c, $[Nm^{-2}]$:

$$q_c = p_s \left\{ \left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\frac{\gamma}{\gamma - 1}} - 1 \right\}$$

where γ is the ratio of specific heats of air with constant pressure and constant volume respectively ($\gamma = 1.4$).

- Calibrated airspeed V_c, $[ms^{-1}]$:

$$V_c = \sqrt{\frac{2\gamma}{\gamma - 1} \frac{p_0}{\rho_0} \left\{ \left(1 + \frac{q_c}{p_0} \right)^{\frac{\gamma - 1}{\gamma}} - 1 \right\}}$$

where $p_0 = 101325\ Nm^{-2}$ = air pressure at sea level and $\rho_0 = 1.225\ kg\,m^{-3}$ = air density at sea level.

- Equivalent airspeed V_e, $[ms^{-1}]$:

$$V_e = V \sqrt{\frac{\rho}{\rho_0}} = \sqrt{\frac{2 q_{dyn}}{\rho_0}}$$

Inputs

$\mathbf{y}_{atm}$	$= [\ \rho\ p_s\ T\ \mu\ g\ ]^T$	*basic atmospheric properties, yatm*
$\mathbf{y}_{ad1}$	$= [\ a\ M\ q_{dyn}\ ]^T$	*basic airdata variables, yad1*

Outputs

$\mathbf{y}_{ad2}$	$= [\ q_c\ V_e\ V_c\ ]^T$	*additional airdata (-related) variables, yad2*

Parameters

All parameters for Airdata2 are defined within the block itself; Airdata 2 does *not* use parameters from the MATLAB workspace.

Connections

in: $\mathbf{y}_{atm}$ comes from the block Atmosph (Airdata Group); $\mathbf{y}_{ad1}$ comes from Airdata1 (Airdata Group)
out: $\mathbf{y}_{ad2}$ is not connected to any other block in the system Beaver

Enter `type airdata2.hlp` at the command-line for on-line help. □

Airdata3

Type

Masked subsystem, aircraft-independent, in general not necessary for solving the equations of motion.

Description

The masked subsystem Airdata3 is used to compute the Reynolds number R_c, which refers to the mean aerodynamic chord of the aircraft, the Reynolds number per unit length R_e, and the total temperature T_t. Usually, these airdata (-related) variables are not necessary for actually solving the equations of motion, but they may be useful for other purposes, such as comparing simulations with windtunnel measurements or tests in real flight. The block Airdata3 can be deleted from the SIMULINK model of the aircraft without affecting the solutions of the ODEs. The equations from this block are independent of the aircraft under consideration.

Equations

- Total temperature T_t, $[K]$:

$$T_t = T\left(1 + \frac{\gamma - 1}{2} M^2\right)$$

- Reynolds number R_c, $[-]$:

$$R_c = \frac{\rho V \overline{c}}{\mu}$$

- Reynolds number per unit length R_e, $[m^{-1}]$:

$$R_e = \frac{\rho V}{\mu}$$

Inputs

$\mathbf{x}$	$= [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$	*state vector, x*
$\mathbf{y}_{atm}$	$= [\ \rho\ p_s\ T\ \mu\ g\ ]^T$	*basic atmospheric properties, yatm*
$\mathbf{y}_{ad1}$	$= [\ a\ M\ q_{dyn}\ ]^T$	*basic airdata variables, yad1*

Outputs

$\mathbf{y}_{ad3}$	$= [\ T_t\ R_e\ R_c\ ]^T$	*additional airdata (-related) variables, yad3*

Parameters

Airdata3 needs the parameter vector *GM1* in order to read out the mean aerodynamic chord $\overline{c}$. Use the routine MODBUILD (section 9.2) to define this vector, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{y}_{atm}$ comes from Atmosph (Airdata Group); $\mathbf{y}_{ad1}$ comes from Airdata1 (Airdata Group)

out: $\mathbf{y}_{ad3}$ is not connected to any other block in the system Beaver

Enter `type airdata3.hlp` at the command-line for on-line help. □

Airdata Group

Type

Non-masked subsystem, aircraft-independent, partly necessary for solving the equations of motion, see the description of the masked subsystems within Airdata Group.

Description

The subsystem Airdata Group is used to compute airdata (-related) variables, of which some are needed for solving the equations of motion, while others are included as 'additional outputs'. Obviously, the latter variables may be deleted from the SIMULINK model without affecting the solutions of the equations of motion.

Subsystems/masked blocks

There are four masked subsystem blocks contained in the subsystem Airdata Group:

- Atmosph: computes basic atmospheric properties (air-temperature, pressure, density), using the ICAO Standard Atmosphere model, as well as the dynamic viscosity of the air and the gravitational acceleration
- Airdata1: computes the most important airdata variables which usually are needed for solving the equations of motion of aircraft
- Airdata2: computes more airdata (-related) variables which may be useful for such purposes as comparing simulations with real flight experiments, windtunnel measurements, etc.
- Airdata3: computes still more airdata (-related) variables

The blocks Airdata2 and Airdata3 can be deleted from the SIMULINK model of the aircraft without affecting the solution of the equations of motion; the other two blocks cannot.

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

Outputs

$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$ — *basic atmospheric properties, yatm*

$\mathbf{y}_{ad1} = [\ a\ M\ q_{dyn}\ ]^T$ — *basic airdata variables, yad1*

$\mathbf{y}_{ad2} = [\ q_c\ V_e\ V_c\ ]^T$ — *additional airdata (-related) variables, yad2*

$\mathbf{y}_{ad3} = [\ T_t\ R_e\ R_c\ ]^T$ — *additional airdata (-related) variables, yad3*

Parameters

The blocks Airdata1, Airdata2, and Atmosph all define their own parameters without reading them from the MATLAB workspace. Airdata3 needs the parameter *GM1*. Use the routine MODBUILD (section 9.2) to define this parameter matrix, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion)

out: $\mathbf{y}_{atm}$ which leaves the block Atmosph is connected to the blocks Airdata1, Airdata2, and Airdata3 (all from the subsystem Airdata Group), Power (Engine Group), Gravity, and xdotcorr (Equations of Motion); $\mathbf{y}_{ad1}$ from Airdata1 is connected to Airdata2 (Airdata Group), Airdata3 (Airdata Group), and FMdims (Aerodynamics Group or Engine Group); $\mathbf{y}_{ad2}$ from Airdata2 and $\mathbf{y}_{ad3}$ from Airdata3 are not connected to any other block in the system Beaver

Enter `type adgrp.hlp` at the command-line for on-line help. □

Atmosph

Type

Masked subsystem, aircraft-independent, necessary for solving state equations.

Description

The block Atmosph is used to compute some basic atmospheric properties, using the ICAO Standard Atmosphere model (see for instance ref.[23] for a description of that model). Atmosph also computes the gravitational acceleration g and the dynamic viscosity μ. The outputs from Atmosph are used by the block Airdata1 for the calculation of airdata variables that need to be known for solving the equations of motion. Therefore, Atmosph should *not* be deleted from the SIMULINK model of the aircraft!

Equations

- Air temperature T in the troposphere, according to the ICAO Standard Atmosphere model, $[K]$:

$$T = T_0 + \lambda H$$

where $T_0 = 288.15\ K$ = air temperature at sea level and $\lambda = -0.0065\ Km^{-1}$ = temperature gradient in troposphere. In this equation the small difference between the geometrical altitude h and the geopotential altitude H has been neglected in view of the altitudes considered; see section 3.2.4.

- Static air pressure in Standard Atmosphere p_s, $[Nm^{-2}]$:

$$p_s = p_0 \left(\frac{T_0}{T} \right)^{\frac{g}{\lambda R}}$$

where $p_0 = 101325\ Nm^{-2}$ = air pressure at sea level and $R = R_a/M_0 = 287.05\ JK^{-1}kg^{-1}$ = specific gas constant of the air ($M_0 = 28.9644\ kg\ kmol^{-1}$ = molecular weight of the air at sea level).

- Air density ρ, $[kgm^{-3}]$, according to the gas law for ideal gasses:

$$\rho = \frac{p_s}{RT}$$

- Coefficient of the dynamic viscosity μ, $[kgm^{-1}s^{-1}]$, according to Sutherland's equation (see ref.[23]):

$$\mu = \frac{1.458 \cdot 10^{-6}\, T^{\frac{3}{2}}}{T + 110.4}$$

- Gravitational acceleration g, $[ms^{-2}]$:

$$g = g_0 \left(\frac{R_{Earth}}{R_{Earth} + h} \right)^2$$

where $g_0 = 9.80665\ ms^{-2}$ = gravitational acceleration at sea level and $R_{Earth} = 6371020\ m$ = radius of the Earth. Note: although this equation uses the radius of the Earth, the *state* equations in the block 12 ODEs are still based upon a flat-Earth model! *This* equation only takes into account the altitude-dependency of the gravitational acceleration.

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

Outputs

$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$ — *basic atmospheric properties, yatm*

Parameters

All parameters for Atmosph are defined within the block itself; Atmosph does *not* use parameters from the MATLAB workspace.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion)

out: $\mathbf{y}_{atm}$ is connected to the blocks Airdata1, Airdata2, Airdata3 (all from the subsystem Airdata Group), Power (Engine Group), Gravity, and xdotcorr (Equations of Motion)

Enter `type atmosph.hlp` at the command-line for on-line help. □

Dimless

Level 1 / Level 2 / Aerodynamics Group / Dimless

Type

Masked subsystem, aircraft-independent, necessary for solving the equations of motion.

Description

The masked subsystem block Dimless is used to obtain non-dimensional roll, pitch, and yaw rates, needed by the aerodynamic model of the 'Beaver'. If you plan to implement a model of another aircraft within the structure of the 'Beaver' model, be sure to compare the definitions of non-dimensional variables with the equations given below. Many textbooks use different expressions for making the angular velocities non-dimensional; the definitions used here are typical for most models developed within the section Stability and Control of the Faculty of Aerospace Engineering. Apart from this, the equations are independent of the aircraft under consideration.

Equations

- Non-dimensional roll rate:

$$p \rightarrow \frac{p\,b}{2V}$$

- Non-dimensional pitch rate:

$$q \rightarrow \frac{q\overline{c}}{V}$$

- Non-dimensional yaw rate:

$$r \rightarrow \frac{r\,b}{2V}$$

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ *state vector, x*

Outputs

$\mathbf{y}_{dl} = [\ \frac{pb}{2V}\ \frac{q\overline{c}}{V}\ \frac{rb}{2V}\ ]^T$ *non-dimensional angular velocities, ydl*

Parameters

Dimless needs the parameter vector *GM1* for reading out the wing span b and mean aerodynamic chord $\overline{c}$. Use the routine MODBUILD (section 9.2) to define this vector, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion)
out: $\mathbf{y}_{dl}$ is connected to Aeromod (Aerodynamics Group)

Enter `type dimless.hlp` at the command-line for on-line help. □

Engine Group (Beaver)

Level 1 / Level 2 / Engine Group

Type

Non-masked subsystem, aircraft-dependent, necessary for solving state equations.

Description

The subsystem Engine Group contains blocks which determine the engine power of the 'Beaver' and compute the resulting forces and moments due to operation of the powerplant, including contributions of the propeller slipstream. Naturally, this subsystem is dependent of the aircraft under consideration. Users who want to adapt the system Beaver for the implementation of a model of another aircraft need to replace the engine model from this subsystem.

Subsystems/masked blocks

There are three masked subsystems contained within the subsystem Engine Group:

Power:	computes the engine power and the non-dimensional pressure increase across the propeller for the 'Beaver' aircraft
Engmod:	computes non-dimensional force and moment coefficients which are caused by the operation of the powerplant, including slipstream effects
FMdims:	makes the non-dimensional force and moment coefficients dimensional

Inputs

$\mathbf{x}$	$=$	$[\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$	*state vector, x*
$\mathbf{u}_{prop}$	$=$	$[\ n\ p_z\ ]^T$	*external propulsion inputs, uprop*
$\mathbf{y}_{atm}$	$=$	$[\ \rho\ p_s\ T\ \mu\ g\ ]^T$	*basic atmospheric properties, yatm*
$\mathbf{y}_{ad1}$	$=$	$[\ a\ M\ q_{dyn}\ ]^T$	*basic airdata variables, yad1*

Outputs

$\mathbf{y}_{pow}$	$=$	$[\ dpt\ P\ ]^T$	*engine power related variables, ypow*
$\mathbf{C}_{prop}$	$=$	$[\ C_{X_p}\ C_{Y_p}\ C_{z_p}\ C_{l_p}\ C_{m_p}\ C_{n_p}\ ]^T$	*propulsive force and moment coefficients, Cprop*
$\mathbf{FM}_{prop}$	$=$	$[\ X_p\ Y_p\ Z_p\ L_p\ M_p\ N_p\ ]^T$	*dimensional propulsive forces and moments, FMprop*

Parameters

The block Engmod needs the engine model parameter matrix *EM*; FMdims needs the parameter vector *GM1*. Use the routine MODBUILD (section 9.2) to define these parameters, or use LOADER (section 9.3.1) to load them into the MATLAB workspace. See appendix D for the definitions of *EM* and *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{u}_{prop}$ is an external input vector with engine inputs; $\mathbf{y}_{atm}$ comes from Atmosph (Airdata Group); $\mathbf{y}_{ad1}$ comes from Airdata1 (Airdata Group)

out: $\mathbf{y}_{pow}$ which leaves Power (Engine Group) is connected to the block Engmod (Engine Group); $\mathbf{C}_{prop}$ from Engmod (Engine Group) is connected to FMdims (Engine Group); $\mathbf{FM}_{prop}$ from FMdims (Engine Group) is connected to FMsort

Enter `type enggrp.hlp` at the command-line for on-line help. □

Engmod (Beaver)

Level 1 / Level 2 / Engine Group / Engmod

Type

Masked subsystem, aircraft-dependent, necessary for solving the equations of motion.

Description

The masked subsystem block Engmod (Beaver) contains the engine forces and moments model for the 'Beaver' aircraft, according to ref.[26]. It computes the force and moment coefficients which are caused by the operation of the powerplant, including the contribution of the propeller slipstream. This engine forces and moments model can be used as a typical example for other propeller-driven aircraft. Obviously, this block is aircraft-dependent, which implies that it must be replaced if a model of another aircraft is to be implemented within the FDC structure. Due to the black-box character of this block, this is not too difficult, even if the structure of the engine model itself differs considerably from the polynomial structure of the 'Beaver' model (e.g. a model which is based upon multi-dimensional look-up tables), or if the aircraft has another type of engine (e.g. a jet engine instead of a piston-driven propeller) or more than one engine.

Equations

• Non-dimensional force and moment coefficients along the body axes due to operation of the power-plant, including the influence of the propeller slipstream (see ref.[26]):

$$
\begin{aligned}
C_{X_p} &= C_{X_{dpt}}\, dpt + C_{X_{\alpha\, dpt^2}}\, \alpha\, dpt^2 \\
C_{Y_p} &= 0 \\
C_{Z_p} &= C_{Z_{dpt}}\, dpt \\
C_{l_p} &= C_{l_{\alpha^2 dpt}}\, \alpha^2\, dpt \\
C_{m_p} &= C_{m_{dpt}}\, dpt \\
C_{n_p} &= C_{n_{dpt^3}}\, dpt^3
\end{aligned}
$$

See table C.4 in appendix C for the values of the stability derivatives from the polynomial equations for the non-dimensional propulsive forces and moments. See also the description of MODBUILD in chapter 4.

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\mathbf{y}_{pow} = [\ dpt\ P\ ]^T$ — *engine power related variables, ypow*

Outputs

$\mathbf{C}_{prop} = [\ C_{X_p}\ C_{Y_p}\ C_{z_p}\ C_{l_p}\ C_{m_p}\ C_{n_p}\ ]^T$ — *propulsive force and moment coefficients, Cprop*

Parameters

Engmod needs the parameter matrix *EM* for reading out the stability coefficients of the engine forces and moments model. Use the routine MODBUILD (section 9.2) for defining this matrix, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *EM*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{y}_{pow}$ comes from Power (Engine Group)
out: $\mathbf{C}_{prop}$ is connected to FMdims (Engine Group)

Enter `type engmod.hlp` at the command-line for on-line help. □

Eulerdot

Type

Masked subsystem, aircraft-independent, contains three state equations.

Description

The masked subsystem block Eulerdot is used to compute the time-derivatives of the Euler angles, i.e. the yaw angle ψ, pitch angle θ, and roll angle φ, which belong to the twelve time-derivatives of the state variables of the non-linear aircraft model. Eulerdot is contained within the subsystem 12 ODEs, which itself is contained within the subsystem Aircraft Equations of Motion. In 12 ODEs the three outputs from Eulerdot are *Muxed* together with the time-derivatives of the other nine state variables.

Equations

• Time-derivatives of the Euler angles ψ, θ, and φ, $[rad\ s^{-1}]$:

$$\begin{aligned} \dot{\psi} &= \frac{q\sin\varphi + r\cos\varphi}{\cos\theta} \\ \dot{\theta} &= q\cos\varphi - r\sin\varphi \\ \dot{\varphi} &= p + \dot{\psi}\sin\theta \end{aligned}$$

Inputs

$\mathbf{u}_{eul} = [\ \mathbf{x}^T\ \mathbf{F}_{tot}{}^T\ \mathbf{M}_{tot}{}^T\ \mathbf{y}_{hlp}{}^T\ ]^T$ — *input vector to* Eulerdot, *ueul*

where:

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{M}_{tot} = [\ L\ M\ N\ ]^T$ — *total external moments, Mtot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{y}_{eul} = [\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ ]^T$ *(part of* $\dot{\mathbf{x}}$*)* — *time-derivatives of the Euler angles, yeul*

Parameters

none

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ come from FMsort; $\mathbf{y}_{hlp}$ comes from Hlpfcn

out: $\mathbf{y}_{eul}$ is *Muxed* together with the time-derivatives of the other state variables into the vector $\dot{\mathbf{x}}$ (not corrected for the implicit nature of the $\dot{\beta}$-equation), which is connected to xdotcorr (Equations of Motion)

Enter `type eulerdot.hlp` at the command-line for on-line help. □

Flpath

Type

Masked subsystem, aircraft-independent, not necessary for solving the equations of motion.

Description

The masked subsystem Flpath is used to compute some flight-path (-related) variables. Its outputs are not used by any other block from the non-linear aircraft model, hence they are not essential for solving the equations of motion. For this reason, the block Flpath has been put in the subsystem Additional Outputs, which may be deleted from the non-linear aircraft model at any time. You are free to add or delete equations to/from this subsystem according to your own needs.

Equations

- Flight-path angle γ, $[rad]$:

$$\gamma = \arcsin\left(\frac{\dot{H}}{V}\right)$$

- Flight-path acceleration fpa = acceleration in the direction of the true airspeed vector $\mathbf{V}$, $[g]$:

$$fpa = \frac{\dot{V}}{g_0}$$

with $g_0 = 9.80665\ ms^{-1}$ = gravitational acceleration at sea level.

- Azimuth angle χ, $[rad]$:

$$\chi = \beta + \psi$$

- Bank angle Φ, $[rad]$:

$$\Phi = \varphi \cos(\theta - \alpha_0)$$

Inputs

$\mathbf{x}$	$=$	$[\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$	*state vector, x*
$\dot{\mathbf{x}}$	$=$	$[\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$	*time-derivative of state vector, xdot*

Outputs

$\mathbf{y}_{fp}$	$=$	$[\ \gamma\ fpa\ \chi\ \Phi\ ]^T$	*flight-path variables, yfp*

Parameters

Flpath needs the steady-state value of the angle of attack α_0 for computing the bank angle. This initial value is extracted from the vector *xinco*, which can be computed with the aircraft trim routine ACTRIM (section 8.2) or loaded into the MATLAB workspace from a file, using INCOLOAD (section 9.3.2). The gravity constant g_0, which is used to normalize the flight-path acceleration *fpa* is defined within the block Flpath itself.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\dot{\mathbf{x}}$ comes from xfix (Equations of Motion)
out: $\mathbf{y}_{fp}$ is not connected to any other block in the system Beaver

Enter `type flpath.hlp` at the command-line for on-line help. □

FMdims

Level 1 / Level 2 / Aerodynamics Group / FMdims
Level 1 / Level 2 / Engine Group / FMdims

Type

Masked subsystem, aircraft-independent, necessary for solving state equations.

Description

For the 'Beaver' aircraft, both the aerodynamic model and the engine forces and moments model express the forces and moments in terms of non-dimensional coefficients. The masked subsystem block FMdims is used to obtain dimensional forces and moments from these non-dimensional coefficients. The equations used here are typical for most aircraft models used within the section Stability and Control of the Faculty of Aerospace Engineering, but other textbooks may use different equations.

Equations

- Dimensional forces, $[N]$:

$$\begin{aligned} X_a &= C_{X_a}\ q_{dyn}\, S \\ Y_a &= C_{Y_a}\ q_{dyn}\, S \\ Z_a &= C_{Z_a}\ q_{dyn}\, S \end{aligned}$$

- Dimensional moments, $[Nm]$:

$$\begin{aligned} L_a &= C_{l_a}\ \ q_{dyn}\, S\, b \\ M_a &= C_{m_a}\ q_{dyn}\, S\, \overline{c} \\ N_a &= C_{n_a}\ \ q_{dyn}\, S\, b \end{aligned}$$

(replace the index a by p for FMdims in the subsystem Engine Group).

Inputs

$\mathbf{y}_{ad1} = [\ a\ M\ q_{dyn}\ ]^T$ — *basic airdata variables, yad1*

$\mathbf{C}_{aero} = [\ C_{X_a}\ C_{Y_a}\ C_{z_a}\ C_{l_a}\ C_{m_a}\ C_{n_a}\ ]^T$ — *aerodynamic force and moment coefficients, Caero*

or:

$\mathbf{C}_{prop} = [\ C_{X_p}\ C_{Y_p}\ C_{z_p}\ C_{l_p}\ C_{m_p}\ C_{n_p}\ ]^T$ — *propulsive force and moment coefficients, Cprop*

Outputs

$\mathbf{FM}_{aero} = [\ X_a\ Y_a\ Z_a\ L_a\ M_a\ N_a\ ]^T$ — *dimensional aerodynamic forces and moments, FMaero*

or:

$\mathbf{FM}_{prop} = [\ X_p\ Y_p\ Z_p\ L_p\ M_p\ N_p\ ]^T$ — *dimensional propulsive forces and moments, FMprop*

Parameters

FMdims needs the parameter vector *GM1* for reading out the mean aerodynamic chord $\overline{c}$, the wing span b, and the wing surface S. Use the routine MODBUILD (section 9.2) to define this vector, or use LOADER (section:9.3.1) for loading it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{y}_{ad1}$ comes from Airdata1 (Airdata Group); $\mathbf{C}_{aero}$ comes from Aeromod (Aerodynamics Group); $\mathbf{C}_{prop}$ comes from Engmod (Engine Group)

out: $\mathbf{FM}_{aero}$ or $\mathbf{FM}_{prop}$ are connected to FMsort

Enter `type fmdims.hlp` at the command-line for on-line help. □

FMsort

Type

Masked subsystem, aircraft-independent, necessary for solving state equations.

Description

The masked subsystem block FMsort is used to compute the resulting forces and moments and to sort them in two separate vectors. Its inputs are vectors from the blocks Aerodynamics Group, Engine Group, Gravity, and Fwind, in which the different contributions to the external forces and moments acting upon the aircraft are computed. By using a separate block FMsort for sorting these forces and moments and adding the different contributions to each other, it becomes easy to implement other contributions to these forces and moments within the model, e.g. forces from the landing gear for taxiing aircraft. For such enhancements the user must include the appropriate subsystem(s) for computing the additional contributions to the body-axes forces and moments and make the appropriate changes to the block FMsort (after unmasking this block).

Equations

- Resulting forces along the body-axes, $[N]$:

$$\begin{aligned} F_x &= X_a + X_p + X_{gr} + X_w \\ F_y &= Y_a + Y_p + Y_{gr} + Y_w \\ F_z &= Z_a + Z_p + Z_{gr} + Z_w \end{aligned}$$

- Resulting moments about the body-axes, $[Nm]$:

$$\begin{aligned} L &= L_a + L_p \\ M &= M_a + M_p \\ N &= N_a + N_p \end{aligned}$$

Inputs

$\mathbf{FM}_{aero}$	$= [\ X_a\ Y_a\ Z_a\ L_a\ M_a\ N_a\ ]^T$	*dimensional aerodynamic forces and moments, FMaero*
$\mathbf{FM}_{prop}$	$= [\ X_p\ Y_p\ Z_p\ L_p\ M_p\ N_p\ ]^T$	*dimensional propulsive forces and moments, FMprop*
$\mathbf{F}_{grav}$	$= [\ X_{gr}\ Y_{gr}\ Z_{gr}\ ]^T$	*gravity force components along body-axes, Fgrav*
$\mathbf{F}_{wind}$	$= [\ X_w\ Y_w\ Z_w\ ]^T$	*corrections to body-axes forces in non-steady atmosphere, Fwind*

Outputs

$\mathbf{F}_{tot}$	$= [\ F_x\ F_y\ F_z\ ]^T$	*total external forces, Ftot*
$\mathbf{M}_{tot}$	$= [\ L\ M\ N\ ]^T$	*total external moments, Mtot*

Parameters

none

Connections

in: $\mathbf{FM}_{aero}$ comes from Aeromod (Aerodynamics Group); $\mathbf{FM}_{prop}$ comes from Engmod (Engine Group); $\mathbf{F}_{grav}$ comes from Gravity; $\mathbf{F}_{wind}$ comes from Fwind

out: $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ are both connected to Vabdot, pqrdot, Eulerdot, and xyHdot (all from the subsystem Equations of Motion / 12 ODEs); $\mathbf{F}_{tot}$ is also connected to Accel (Additional Outputs)

Enter `type fmsort.hlp` at the command-line for on-line help. □

Fwind

Type

Masked subsystem, aircraft-independent, necessary for solving the equations of motion in a non-steady atmosphere.

Description

The masked subsystem block Fwind is used to compute correction terms which are to be added to the forces along the aircraft's body-axes if flight in non-steady atmosphere is considered. These correction terms depend upon the components of the wind velocity vector V_w along the aircraft's body-axes, the time-derivatives of these wind components, and the roll, pitch, and yaw rates of the aircraft.

Equations

• Correction terms to the body-axes forces in non-steady atmosphere, $[N]$:

$$\begin{aligned} X_w &= -m\,(\dot{u}_w + qw_w - rv_w) \\ Y_w &= -m\,(\dot{v}_w - pw_w + ru_w) \\ Z_w &= -m\,(\dot{w}_w + pv_w - qu_w) \end{aligned}$$

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\mathbf{u}_{wind} = [\ u_w\ v_w\ w_w\ \dot{u}_w\ \dot{v}_w\ \dot{w}_w\ ]^T$ — *wind velocity components along body-axes and their time-derivatives, uwind*

Outputs

$\mathbf{F}_{wind} = [\ X_w\ Y_w\ Z_w\ ]^T$ — *corrections to body-axes forces in non-steady atmosphere, Fwind*

Parameters

Fwind needs the parameter vector *GM1* in order to extract the mass m of the aircraft (the mass has been implemented as a *parameter*, i.e. it is assumed that it is constant during the relative short time intervals considered). Use the routine MODBUILD (section 9.2) to define *GM1*, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{u}_{wind}$ is an external input vector with wind and turbulence velocities and the time-derivatives of these velocity components

out: $\mathbf{F}_{wind}$ is connected to FMsort

Enter `type fwind.hlp` at the command-line for on-line help. □

Gravity

Level 1 / Level 2 / Gravity

Type

Masked subsystem, aircraft-independent, necessary for solving state equations.

Description

The masked subsystem block Gravity computes the contribution of the aircraft's weight W along the body-axes. In order to do so, the Euler angles ψ, θ, and φ need to be known. In the current model, the gravitational acceleration varies with height. This variable is obtained from the block Atmosph.

Equations

• Contribution of the aircraft weight to the forces along the body-axes, $[N]$:

$$\begin{aligned} X_{gr} &= -W \sin\theta \\ Y_{gr} &= W \cos\theta \sin\varphi \\ Z_{gr} &= W \cos\theta \cos\varphi \end{aligned}$$

with: $W = mg$ = weight of the aircraft, $[N]$.

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ *state vector, x*

$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$ *basic atmospheric properties, yatm*

Outputs

$\mathbf{F}_{grav} = [\ X_{gr}\ Y_{gr}\ Z_{gr}\ ]^T$ *gravity force components along body-axes, Fgrav*

Parameters

Gravity needs the parameter vector *GM1* in order to read out the mass m (which has been implemented as a parameter, i.e. it is assumed to be constant during the relatively short time interval considered). Use the routine MODBUILD (section 9.2) to define *GM1*, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{y}_{atm}$ comes from Atmosph (Airdata Group)

out: $\mathbf{F}_{grav}$ is connected to FMsort

Enter `type gravity.hlp` at the command-line for on-line help. □

Hlpfcn

Type

Masked subsystem, aircraft-independent help-function block, necessary for solving the equations of motion due to the current model structure.

Description

The masked subsystem block Hlpfcn is used to compute some frequently used sines and cosines of the angle of attack, sideslip angle, and Euler angles. These computations have been combined in one block for reasons of efficiency; in this way many double computations have been eliminated in the simulation model. The outputs from this blocks are used by several other subsystems. For this reason, the block Hlpfcn has been included in a *feedback* loop within the system Beaver.

Equations

Hlpfcn simply computes the required sines and cosines, and puts the results into one vector.

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ *state vector, x*

Outputs

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Parameters

none

Connections

in: **x** comes from the block Integrator (Equations of Motion)

out: $\mathbf{y}_{hlp}$ is connected to uvw, xdotcorr (Equations of Motion), Eulerdot, pqrdot, Vabdot, xyHdot (all from the subsystem Equations of Motion / 12 ODEs, and uvwdot (Additional Outputs)

Enter `type hlpfcn.hlp` at the command-line for on-line help. □

Integrator

Type

Standard SIMULINK block, necessary for solving the equations of motion.

Description

The block Integrator is used to obtain the time-trajectories of the twelve state variables from their time-derivatives. It expects the initial values of the state variables to be defined within the MATLAB workspace before starting a new simulation.

Equations

• Update of the state vector for the current time-step:

$$\mathbf{x}(t_{n+1}) \;=\; \mathbf{x}(t_n) + \int_{t_n}^{t_{n+1}} \dot{\mathbf{x}}(t)dt; \quad n = 0,\ 1,\ \ldots$$

where the integral is approximated by a numerical integration method as explained in section 4.2.

Inputs

$\dot{\mathbf{x}} \;=\; [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ — *time-derivative of state vector, xdot*

Outputs

$\mathbf{x} \;=\; [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

Parameters

The initial value of the state vector must be defined in the vector *xinco*. A steady-state initial value can be computed with the aircraft trim routine ACTRIM (section 8.2), or it can be loaded into the MATLAB workspace from a file, using INCOLOAD (section 9.3.2). Of course, it is also possible to manually define *xinco*.

Connections

in: $\dot{\mathbf{x}}$ comes from the block xdotcorr (Equations of Motion)
out: $\mathbf{x}$ is connected to most other blocks in the system Beaver □

Level 1

Type

First level of aircraft model (graphical SIMULINK system which contains the input/output blocks for the aircraft model.

Description

The first level of the system Beaver fulfills the input/output functions of this system. All inputs and outputs are sent to the workspace by means of To Workspace blocks. A time-signal is also sent to the workspace in order to make it possible to plot the simulation results. The most important outputs are connected to Outport blocks. These outputs can be used by other SIMULINK systems. Inputs from other SIMULINK systems are sent through by means of twelve Inport blocks (six control inputs, and six Inports for atmospheric disturbances). Unfortunately, SIMULINK does not allow passing *vector* signals through the Inports and Outports in the first level of a graphical system. This is why only a subset of the outputs from Beaver leave the system through these Outport blocks. All results are available in the workspace, however. The signals which are passed through the Inports and Outports are called *S-function* inputs and outputs, respectively in all help-files for the FDC toolbox.

Subsystems/masked blocks

The first level of the system Beaver contains Inport, Outport, Mux, Demux, and To Workspace blocks to manipulate the inputs and outputs from this system. A clock block has been included to provide a time-basis for plotting purposes in the MATLAB workspace. The actual aircraft model is contained in the subsystem Beaver dynamics and output equations, which has shortly been denoted by Level 2 in this report.

Inputs

The inputs to Level 1 are the twelve *S-function inputs* to the system Beaver: δ_e, δ_a, δ_r, δ_f, n, p_z, u_w, v_w, w_w, $\dot{u}_w$, $\dot{v}_w$, and $\dot{w}_w$. These inputs are connected to scalar Inport blocks and *Muxed* in order to obtain the three external input vectors $\mathbf{u}_{aero}$, $\mathbf{u}_{prop}$, and $\mathbf{u}_{wind}$. See the description of Level 2 for the definitions of these vectors. See also figure 5.2 from section 5.1.

Outputs

Level 1 has sixteen *S-function outputs*, which are connected to scalar Outport blocks: V, α, β, p, q, r, ψ, θ, φ, x_e, y_e, H, $\dot{H}$, $\frac{pb}{2V}$, $\frac{q\overline{c}}{V}$, and $\frac{rb}{2V}$. These outputs are obtained from the output vectors $\mathbf{x}$, $\dot{\mathbf{x}}$, and $\mathbf{y}_{dl}$, as shown in figure 5.2 from section 5.1. In addition, all output signals from the subsystem Level 2 are sent to the MATLAB workspace by means of To Workspace blocks. See the description of Level 2 for the definitions of the output vectors.

Parameters

The subsystem Beaver dynamics and output relations (Level 2) needs the variables *AM*, *EM*, *GM1*, *GM2*, *xfix*, and *xinco*, which contain the model parameters for the 'Beaver', to be defined in the MATLAB workspace. See the description of Level 2 for more details.

Connections

Level 1 is the I/O level of the system Beaver, where all input and output vectors are *Muxed* together and sent to the MATLAB workspace by means of To Workspace blocks (see the on-line help files INPUTS.HLP and OUTPUTS.HLP for more details). All inputs are sent to and all outputs are extracted from the subsystem Level 2.

Enter `type level1.hlp` at the command-line for on-line help. □

Level 2

Level 1 / Level 2

Type

Subsystem of Level 1 which contains the actual modular non-linear aircraft model.

Description

The second level of the system Beaver contains the actual implementation of the non-linear aircraft model. Its block-diagram is shown in figure 5.1 in section 5.1 (compare this diagram with the scheme from figure 3.2 in chapter 3).

Subsystems/masked blocks

There are four masked subsystem blocks contained in Level 2:

- FMsort: adds the different contributions to the external forces and moments and sorts out separate vectors with the force and moment components along the aircraft's body-axes
- Fwind: computes the contributions to the external forces and moments due to non-steady atmosphere (wind and turbulence)
- Gravity: computes the gravitational forces along the aircraft's body-axes
- Hlpfcn: computes sines and cosines of α, β, ψ θ, and φ, needed by other blocks from Beaver

In addition, Level 2 contains five non-masked subsystems:

- Additional outputs**:** computes some 'additional' output variables which are not necessary for the solution of the equations of motion themselves (in this case, additional accelerations and flight-path variables are determined)
- Aerodynamics Group (Beaver)**:** computes the aerodynamic forces and moments for the 'Beaver' aircraft
- Aircraft equations of motion (Beaver)**:** contains the general differential equations for the rigid-body dynamics plus an aircraft-dependent correction for the time-derivatives of the state variables
- Airdata Group**:** computes airdata (-related) variables
- Engine Group (Beaver)**:** computes the propulsive forces and moments for the 'Beaver' aircraft

See figure 5.1 in section 5.1 for more details.

Inputs

$\mathbf{u}_{aero} = [\ \delta_e\ \delta_a\ \delta_r\ \delta_f\ ]^T$		*aerodynamic control inputs, uaero*
$\mathbf{u}_{prop} = [\ n\ p_z\ ]^T$		*external propulsion inputs, uprop*
$\mathbf{u}_{wind} = [\ u_w\ v_w\ w_w\ \dot{u}_w\ \dot{v}_w\ \dot{w}_w\ ]^T$		*wind velocity components along body-axes and their time-derivatives, uwind*

Outputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$		*state vector, x*
$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$		*time-derivative of state vector, xdot*
$\mathbf{y}_{bvel} = [\ u\ v\ w\ ]^T$		*body-axes velocity components, ybvel*
$\mathbf{y}_{uvw} = [\ \dot{u}\ \dot{v}\ \dot{w}\ ]^T$		*time-derivatives of body-axes velocities, yuvw*
$\mathbf{y}_{dl} = [\ \frac{pb}{2V}\ \frac{q\bar{c}}{V}\ \frac{rb}{2V}\ ]^T$		*non-dimensional angular velocities, ydl*
$\mathbf{y}_{fp} = [\ \gamma\ fpa\ \chi\ \Phi\ ]^T$		*flight-path variables, yfp*
$\mathbf{y}_{pow} = [\ dpt\ P\ ]^T$		*engine power related variables, ypow*
$\mathbf{y}_{acc} = [\ A_x\ A_y\ A_z\ a_{x,k}\ a_{y,k}\ a_{z,k}\ ]^T$		*specific forces and accelerations, yacc*
$\mathbf{C}_{aero} = [\ C_{X_a}\ C_{Y_a}\ C_{Z_a}\ C_{l_a}\ C_{m_a}\ C_{n_a}\ ]^T$		*aerodynamic force and moment coefficients, Caero*
$\mathbf{C}_{prop} = [\ C_{X_p}\ C_{Y_p}\ C_{z_p}\ C_{l_p}\ C_{m_p}\ C_{n_p}\ ]^T$		*propulsive force and moment coefficients, Cprop*
$\mathbf{FM}_{aero} = [\ X_a\ Y_a\ Z_a\ L_a\ M_a\ N_a\ ]^T$		*dimensional aerodynamic forces and moments, FMaero*
$\mathbf{FM}_{prop} = [\ X_p\ Y_p\ Z_p\ L_p\ M_p\ N_p\ ]^T$		*dimensional propulsive forces and moments, FMprop*
$\mathbf{F}_{grav} = [\ X_{gr}\ Y_{gr}\ Z_{gr}\ ]^T$		*gravity forces, Fgrav*

$\mathbf{F}_{wind} = [\ X_w\ Y_w\ Z_w\ ]^T$ *corrections to body-axes forces in non-steady atmosphere, Fwind*

$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$ *basic atmospheric properties, yatm*

$\mathbf{y}_{ad1} = [\ a\ M\ q_{dyn}\ ]^T$ *basic airdata variables, yad1*

$\mathbf{y}_{ad2} = [\ q_c\ V_e\ V_c\ ]^T$ *additional airdata (-related) variables, yad2*

$\mathbf{y}_{ad3} = [\ T_t\ R_e\ R_c\ ]^T$ *additional airdata (-related) variables, yad3*

Parameters

The subsystems from Level 2 require the variables AM, EM, $GM1$, $GM2$, $xfix$, and $xinco$ to be defined in the MATLAB workspace. The following list shows which variables are needed by which (masked or non-masked) subsystem from Level 2:

AM: Aerodynamics Group (Beaver),Aircraft Equations of Motion (Beaver)

EM: Engine Group (Beaver)

GM1: Additional Outputs, Aerodynamics Group (Beaver), Aircraft Equations of Motion (Beaver), Airdata Group, Engine Group (Beaver), Fwind, Gravity

GM2: Aircraft Equations of Motion (Beaver)

xfix: Aircraft Equations of Motion (Beaver)

xinco: Additional Outputs, Aircraft Equations of Motion (Beaver)

The variables AM, EM, $GM1$, and $GM2$ can be defined by the routine MODBUILD (section 9.2) or retrieved from file by LOADER (section 9.3.1). The variable $xfix$ can be used to fix state variables artificially to their initial values. It is defined by LOADER and can be altered by FIXSTATE (section 9.5) if required. The initial value of the state vector is obtained from $xinco$, which can be computed with the trim routine ACTRIM (section 8.2), or loaded from file with INCOLOAD (section 9.3.2).

Connections

in: all input vectors are obtained from Level 1 by means of Inport blocks in Level 2

out: all output vectors from the aircraft model (except for some very trivial results such as the help vector $\mathbf{y}_{hlp}$ are sent to Level 1 by means of standard Outport blocks

Enter `type level2.hlp` at the command-line for on-line help. □

Power (Beaver)

Type

Masked subsystem block, aircraft-dependent, necessary for solving state equations.

Description

The masked subsystem block Power (Beaver) is used to compute the engine power P and the non-dimensional increase in total pressure across the propeller *dpt*. For the 'Beaver', there is a simple relation between P, *dpt*, and the airspeed V. Moreover, the engine power itself is expressed by only one simple equation, which made it very easy to implement the engine model of the 'Beaver' in the masked SIMULINK subsystem Power. See also ref.[26]. The non-dimensional pressure-increase is used to compute the contribution of the engine to the external forces and moments and the influence of changes in airspeed. Obviously, the block Power (Beaver) will need to be replaced by another engine model if the FDC model structure is used to implement a model of another aircraft. In that case, it may be necessary to use more complex solutions, such as a *table lookup* routine instead of the simple equations used here. Due to the black-box structure of the model, this shouldn't be a problem.

Equations

• Non-dimensional pressure increase across the propeller *dpt*, $[-]$:

$$dpt = \frac{\Delta p_t}{\frac{1}{2}\rho V^2} = C_1 + C_2 \left(\frac{P}{\frac{1}{2}\rho V^3} \right)$$

with $\frac{P}{\frac{1}{2}\rho V^3}$ measured in $[kW\, kg^{-1}\, s^3]$ and: $C_1 = 0.08696$, $C_2 = 191.18$, see ref.[26].

• Engine power P, $[Nm\, s^{-1}]$:

$$P = 0.7355 \left\{ -326.5 + \left(0.00412(p_z + 7.4)(n + 2010) + (408.0 - 0.0965n) \left(1.0 - \frac{\rho}{\rho_0} \right) \right) \right\}$$

Inputs

$\mathbf{x}$	$=$	$[\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$	*state vector, x*
$\mathbf{u}_{prop}$	$=$	$[\ n\ p_z\ ]^T$	*external propulsion inputs, uprop*
$\mathbf{y}_{atm}$	$=$	$[\ \rho\ p_s\ T\ \mu\ g\ ]^T$	*basic atmospheric properties, yatm*

Outputs

$\mathbf{y}_{pow}$	$=$	$[\ dpt\ P\ ]^T$	*engine power related variables, ypow*

Parameters

Power does *not* use parameters from the MATLAB workspace; it defines all required parameters within the block itself.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{u}_{prop}$ is an external input vector with engine inputs; $\mathbf{y}_{atm}$ comes from Atmosph (Airdata Group)

out: $\mathbf{y}_{pow}$ is connected to Engmod (Engine Group)

Enter `type power.hlp` at the command-line for on-line help. □

Type

Masked subsystem block, aircraft-independent, contains three state equations.

Description

The masked subsystem block pqrdot is used to compute the time-derivatives of the roll rate p, pitch rate q, and the yaw rate r. These time-derivatives are functions of the angular velocities themselves and the external moments about the body-axes of the aircraft, as shown in the equations below. The coefficients P_{pp}, P_{pq}, P_{pr}, ... , R_m, and R_n are inertia parameters, see table B.2 in appendix B. The block pqrdot is contained in the subsystem 12 ODEs, which itself is contained in the subsystem Aircraft Equations of Motion. In 12 ODEs the three outputs from pqrdot are *Muxed* together with the time-derivatives of the other nine state variables.

Equations

• Time-derivatives of the angular velocities along the body-axes, $[rad\ s^{-2}]$:

$$\begin{aligned}
\dot{p} &= P_{pp}p^2 + P_{pq}pq + P_{pr}pr + P_{qq}q^2 + P_{qr}qr + P_{rr}r^2 + P_lL + P_mM + P_nN \\
\dot{q} &= Q_{pp}p^2 + Q_{pq}pq + Q_{pr}pr + Q_{qq}q^2 + Q_{qr}qr + Q_{rr}r^2 + Q_lL + Q_mM + Q_nN \\
\dot{r} &= R_{pp}p^2 + R_{pq}pq + R_{pr}pr + R_{qq}q^2 + R_{qr}qr + R_{rr}r^2 + R_lL + R_mM + R_nN
\end{aligned}$$

Inputs

$\mathbf{u}_{pqr} = [\ \mathbf{x}^T\ \mathbf{F}_{tot}{}^T\ \mathbf{M}_{tot}{}^T\ \mathbf{y}_{hlp}{}^T\ ]^T$ *input vector to* pqrdot, *upqr*

where:

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ *state vector, x*

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ *total external forces, Ftot*

$\mathbf{M}_{tot} = [\ L\ M\ N\ ]^T$ *total external moments, Mtot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{y}_{pqr} = [\ \dot{p}\ \dot{q}\ \dot{r}\ ]^T$ *(part of* $\dot{\mathbf{x}}$*)* *time-derivatives of the angular velocities, ypqr*

Parameters

The block pqrdot needs the parameter matrix *GM2* for reading out the inertia parameters (note: the moments and products of inertia are considered to be constant during the motions of interest). Use the routine MODBUILD (section 9.2) to define this matrix, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM2*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ come from the block FMsort; $\mathbf{y}_{hlp}$ comes from the block Hlpfcn

out: $\mathbf{y}_{pqr}$ is *Muxed* together with the time-derivatives of the other state variables into the vector $\dot{\mathbf{x}}$ (not corrected for implicit nature of the $\dot{\beta}$-equation), which is connected to xdotcorr (Equations of Motion)

Enter `type pqrdot.hlp` at the command-line for on-line help. □

Type

Masked subsystem block, aircraft-independent, necessary for solving state equations.

Description

The block uvw is used to compute the body-axes velocity components from the angle of attack α, sideslip angle β, and true airspeed V. This is necessary for determining the coordinates x_e and y_e and the altitude H of the aircraft.

Equations

- Velocity component u along the X_B-axis, $[ms^{-1}]$:

  $u = V \cos\alpha \cos\beta$

- Velocity component v along the Y_B-axis, $[ms^{-1}]$:

  $v = V \sin\beta$

- Velocity component w along the Z_B-axis, $[ms^{-1}]$:

  $w = V \sin\alpha \cos\beta$

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ *state vector, x*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{y}_{bvel} = [\ u\ v\ w\ ]^T$ *body-axes velocity components, ybvel*

Parameters

none

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{y}_{hlp}$ comes from Hlpfcn

out: $\mathbf{y}_{bvel}$ is connected to xyHdot (Equations of Motion / 12 ODEs)

Enter `type uvw.hlp` at the command-line for on-line help. □

uvwdot

Type

Masked subsystem block, aircraft-independent, *not* necessary for solving the equations of motion (does not contain any state equations itself, because it is an *additional-output* block).

Description

For some purposes it may be useful to know the time-derivatives of the body-axes velocity components u, v, and w in addition to the body-axes velocity components themselves. Since u, v, and w are not used as state variables of the aircraft model – they were replaced by V, α, and β – their time-derivatives are not needed for solving the equations of motion themselves. (The velocity components u, v, and w are computed in the block uvw as a function of V, α, and β; uvw is contained in the subsystem Aircraft Equations of Motion.) Therefore a separate block uvwdot is used to compute these time-derivatives. This block has been included in the subsystem Additional Outputs (!) in order to make clear that this block can be deleted from the system without affecting the solutions of the equations of motion.

Equations

- Time-derivative of the velocity component along the X_B-axis, $[ms^{-2}]$:

  $$\dot{u} = \dot{V}\cos\alpha\cos\beta - V(\dot{\alpha}\sin\alpha\cos\beta + \dot{\beta}\cos\alpha\sin\beta)$$

- Time-derivative of the velocity component along the Y_B-axis, $[ms^{-2}]$:

  $$\dot{v} = \dot{V}\sin\beta + V\dot{\beta}\cos\beta$$

- Time-derivative of the velocity component along the Z_B-axis, $[ms^{-2}]$:

  $$\dot{w} = \dot{V}\sin\alpha\cos\beta + V(\dot{\alpha}\cos\alpha\cos\beta - \dot{\beta}\sin\alpha\sin\beta)$$

Inputs

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ *state vector, x*

$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ *time-derivative of state vector, xdot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{y}_{uvw} = [\ \dot{u}\ \dot{v}\ \dot{w}\ ]^T$ *time-derivatives of the body-axes velocity components, yuvw*

Parameters

none

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\dot{\mathbf{x}}$ comes from the block xfix (Equations of Motion); $\mathbf{y}_{hlp}$ comes from the block Hlpfcn;

out: $\mathbf{y}_{uvw}$ is not connected to any other block from the system Beaver (!)

Enter `type uvwdot.hlp` at the command-line for on-line help. □

Vabdot

Type

Masked subsystem block, aircraft-independent, contains three state equations.

Description

The masked subsystem block Vabdot computes the time-derivatives of the true airspeed V, angle of attack α, and sideslip angle β. Since V, α, and β are state variables of the dynamic model, their time-derivatives are essential for solving the equations of motion. The block Vabdot is contained within the subsystem 12 ODEs, which itself is a subsystem of Aircraft Equations of Motion. In 12 ODEs the three outputs from Vabdot are *Muxed* together with the time-derivatives of the other nine state variables.

Equations

- Time-derivative of the true airspeed V, $[ms^{-2}]$:

$$\dot{V} = \frac{1}{m}\left(F_x \cos\alpha \cos\beta + F_y \sin\beta + F_z \sin\alpha \cos\beta\right)$$

- Time-derivative of the angle of attack α, $[rad\, s^{-1}]$:

$$\dot{\alpha} = \frac{1}{V\cos\beta}\left\{\frac{1}{m}\left(-F_x \sin\alpha + F_z \cos\alpha\right)\right\} + q - \left(p\cos\alpha + r\sin\alpha\right)\tan\beta$$

- Time-derivative of the sideslip angle β, $[rad\, s^{-1}]$:

$$\dot{\beta} = \frac{1}{V}\left\{\frac{1}{m}\left(-F_x \cos\alpha \sin\beta + F_y \cos\beta - F_z \sin\alpha \sin\beta\right)\right\} + p\sin\alpha - r\cos\alpha$$

Inputs

$\mathbf{u}_{Vab} = [\ \mathbf{x}^T\ \mathbf{F}_{tot}{}^T\ \mathbf{M}_{tot}{}^T\ \mathbf{y}_{hlp}{}^T\ ]^T$ — *input vector to* Vabdot, *uVab*

where:

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{M}_{tot} = [\ L\ M\ N\ ]^T$ — *total external moments, Mtot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{y}_{Vab} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ ]^T$ *(part of* $\dot{\mathbf{x}}$*)* — *time-derivatives of* V, α, *and* β, *yVab*

Parameters

Vabdot needs the parameter vector *GM1* in order to retrieve the mass m (the mass is used as a parameter, i.e. it is assumed to be constant during the relatively short time-interval considered). Use the routine MODBUILD (section 9.2) to define this vector, or use LOADER (section 9.3.1) to load it into the MATLAB workspace. See appendix D for the definition of *GM1*.

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ come from FMsort; $\mathbf{y}_{hlp}$ comes from Hlpfcn

out: $\mathbf{y}_{Vab}$ is *Muxed* together with the time-derivatives of the other state variables into the vector $\dot{\mathbf{x}}$ (not corrected for the implicit nature of the $\dot{\beta}$-equation), which is connected to xdotcorr (Equations of Motion)

Enter `type vabdot.hlp` at the command-line for on-line help. □

xdotcorr (Beaver)

Type

Masked subsystem block, aircraft-dependent, necessary for solving state equations.

Description

The differential equation for the sideslip angle from the 'Beaver' model is *implicit*, because $\dot{\beta}$ appears on both sides of the equation: $\dot{\beta}$ is a function of the sideforce F_y, while the aerodynamic component of this sideforce itself depends upon $\dot{\beta}$. Since such implicit equations are difficult for SIMULINK to solve (see section 4.2.7), this equation must be expanded to an explicit ODE. For the 'Beaver' model this is relatively easy since both dependencies are linear, but the resulting $\dot{\beta}$-equation then contains an aircraft-dependent term which ruins the anticipated aircraft-independent model structure. This problem has been solved by neglecting the $\dot{\beta}$-term during the computation of the aerodynamic side-force (see the description of Aeromod) and applying the appropriate corrections to the thus computed value of $\dot{\beta}$ in a separate correction block xdotcorr. Here, the correction block has been configured for the 'Beaver' model. The block xdotcorr is the only block in the subsystem Aircraft Equations of Motion that needs to be replaced if a model of another aircraft is to be implemented. Still, this one aircraft-dependent block is one too many, so in future versions of the FDC toolbox this problem should be solved differently.

Equations

The $\dot{\beta}$-equation for the 'Beaver' can be written as:

$$\dot{\beta} = \frac{1}{Vm}\left(-F_x \cos\alpha \sin\beta + {F_y}^* \cos\beta - F_z \sin\alpha \sin\beta + \frac{1}{2}\rho V^2 S C_{Y_{\dot{\beta}}} \frac{\dot{\beta} b}{2V} \cos\beta\right) + p \sin\alpha - r \cos\alpha$$

where ${F_y}^*$ is the side-force *without* the contribution of $\dot{\beta}$. The $\dot{\beta}$-term on the right hand side of this equation is moved to the left-hand side:

$$\dot{\beta}^* \equiv \dot{\beta}\left(1 - \frac{\rho S b}{4m} C_{Y_{\dot{\beta}}} \cos\beta\right) = \frac{1}{Vm}\left(-F_x \cos\alpha \sin\beta + {F_y}^* \cos\beta - F_z \sin\alpha \sin\beta\right) + p \sin\alpha - r \cos\alpha$$

Based upon this equation the following calculation sequence has been used in the system Beaver:

1. the external forces and moments are computed as usual, except for the $\dot{\beta}$-contribution to the aerodynamic side-force,
2. the thus obtained forces and moments are substituted into the general $\dot{\beta}$ equation, yielding $\dot{\beta}^*$ instead of $\dot{\beta}$,
3. the true value of $\dot{\beta}$ is computed with the expression $\dot{\beta} = \dot{\beta}^* \left(1 - \frac{\rho S b}{4m} C_{Y_{\dot{\beta}}}\right)^{-1}$.

The last step represents a correction to the originally computed value of $\dot{\beta}$, which was denoted as $\dot{\beta}^*$; this correction is contained in xdotcorr. See also sectionsec:implicit.

Inputs

$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ *time-derivative of state vector, xdot (uncorrected for $\dot{\beta}$)*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$ often used sines and cosines, *yhlp*

$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$ *basic atmospheric properties, yatm*

Outputs

$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ *time-derivative of state vector, xdot (corrected for $\dot{\beta}$)*

Parameters

The block xdotcorr needs the parameter vector *GM1* in order to extract the wing span, wing surface, and mass of the aircraft (the mass has been implemented as a parameter, i.e. it assumed to be constant during the relatively short time-intervals considered). It also needs the parameter matrix *AM* for reading out the stability derivative $C_{Y_{\dot{\beta}}}$. Use MODBUILD (section 9.2) for defining these parameters, or use LOADER (section 9.3.1) to load them into the MATLAB workspace. See appendix D for the definitions of *GM1* and *AM*.

Connections

in: $\dot{\mathbf{x}}$ (not corrected for the implicit nature of the $\dot{\beta}$-equation) comes from the subsystem 12 ODEs; $\mathbf{y}_{hlp}$ comes from the block Hlpfcn; $\mathbf{y}_{atm}$ comes from the block Atmosph (Airdata Group)

out: $\dot{\mathbf{x}}$ (with $\dot{\beta}$-correction) is connected to the block xfix (Equations of Motion)

Enter `type xdotcorr.hlp` at the command-line for on-line help. □

Type

Masked Gain block, aircraft-independent, not necessary for solving the equations of motion themselves but quite useful for purposes such as autopilot design and analysis.

Description

Sometimes it is useful to artificially fix state variables to their initial values by simply setting their time-derivatives to zero, thus totally disregarding the values of the time-derivatives resulting from the model equations. For instance, it may be useful to analyze longitudinal-lateral cross-coupling effects by comparing results from the full model with results obtained by artificially neglecting longitudinal or lateral motions of the aircraft. Another application is for the design of an 'autothrottle' which serves to maintain a constant airspeed by means of power-inputs to the engine. Comparing results with a system where the airspeed V is artificially fixed to its initial value can help in assessing the performance of the controller.

The block xfix is a masked Gain block from the SIMULINK library Linear, which multiplies the time-derivatives of all state variables with a value of either one (use the computed time-derivative) or zero (artificially fix the state variable).

Equations

• Modified time-derivative of the state vector, obtained by multiplying the computed value element-by-element with the vector *xfix*:

$$\dot{\mathbf{x}}_{new} = \dot{\mathbf{x}}_{old} * xfix = \begin{bmatrix} \dot{V} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\varphi} \\ \dot{x}_e \\ \dot{y}_e \\ \dot{H} \end{bmatrix} * \begin{bmatrix} xfix(1) \\ xfix(2) \\ xfix(3) \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ xfix(12) \end{bmatrix} \qquad \text{where:} \quad xfix(i) = \begin{cases} 0 \\ 1 \end{cases} \quad i \in \{1, 2, \ldots, 12\}$$

Inputs

$\dot{\mathbf{x}}_{old} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ *time-derivative of state vector, xdot (true value)*

Outputs

$\dot{\mathbf{x}}_{new} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ *time-derivative of state vector, xdot (partially fixed value)*

Parameters

The block xfix requires the multiplication factor of the gain block to be defined in the variable *xfix* in the MATLAB workspace. This variable is automatically set to 1 by the routine LOADER (section 9.3.1) if the other system parameters are loaded from file. If it is required to fix individual states, *xfix* can either be defined by hand or by calling the routine FIXSTATE (section 9.5).

Connections

in: $\dot{\mathbf{x}}$ comes from the block xdotcorr (Equations of Motion)

out: $\dot{\mathbf{x}}$ (with elements that may have been set to zero artificially) is connected to the block Integrator (Equations of Motion)

Enter `type xfix.hlp` at the command-line for on-line help. See also the description of the MATLAB subroutine FIXSTATE in section 9.5 or type `help fixstate` at the command-line. □

xyHdot

Type

Masked subsystem block, aircraft-independent, contains three of the state equations.

Description

The masked subsystem block xyHdot computes the time-derivatives of the aircraft's X and Y-coordinates x_e and y_e, measured with respect to the Earth-fixed reference frame, and the rate of climb or descent $\dot{H}$. The variables x_e, y_e and H are all state variables of the non-linear aircraft model. For most purposes it is possible to omit x_e and y_e in the simulation model, because the motions of the aircraft are not affected by its coordinates with respect to the Earth. However, for the sake of completeness these two variables have been included to the model; the coordinates can for instance be used in the simulation of ILS approaches to a runway. Notice that the altitude H should *not* be omitted from the simulation model because of the altitude-dependency of the air temperature, pressure, and density, which affect the outputs of both the aerodynamic and engine models. The block xyHdot is contained in the subsystem 12 ODEs, which itself is contained in the subsystem Aircraft Equations of Motion. In 12 ODEs the three outputs from xyHdot are *Muxed* together with the time-derivatives of the other nine state variables.

Equations

- Time-derivative of the X-coordinate x_e, $[ms^{-1}]$:

$$\dot{x}_e = \{u_e \cos\theta + (v_e \sin\varphi + w_e \cos\varphi)\sin\theta\}\cos\psi - (v_e \cos\varphi - w_e \sin\varphi)\sin\psi$$

- Time-derivative of the Y-coordinate y_e, $[ms^{-1}]$:

$$\dot{y}_e = \{u_e \cos\theta + (v_e \sin\varphi + w_e \cos\varphi)\sin\theta\}\sin\psi + (v_e \cos\varphi - w_e \sin\varphi)\cos\psi$$

- Time-derivative of the altitude H, $[ms^{-1}]$:

$$\dot{H} = -\dot{z}_e = u_e \sin\theta - (v_e \sin\varphi + w_e \cos\varphi)\cos\theta$$

Inputs

$\mathbf{y}^*_{bvel} = [\ u + u_w\ v + v_w\ w + w_w\ ]^T$ — *body-axes velocity components plus wind, ybvel**

$\mathbf{u}_{xyH} = [\ \mathbf{x}^T\ {\mathbf{F}_{tot}}^T\ {\mathbf{M}_{tot}}^T\ {\mathbf{y}_{hlp}}^T\ ]^T$ — *input vector to* xyHdot*, uxyH*

where:

$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$ — *state vector, x*

$\mathbf{F}_{tot} = [\ F_x\ F_y\ F_z\ ]^T$ — *total external forces, Ftot*

$\mathbf{M}_{tot} = [\ L\ M\ N\ ]^T$ — *total external moments, Mtot*

$\mathbf{y}_{hlp} = [\ \cos\alpha\ \sin\alpha\ \cos\beta\ \sin\beta\ \tan\beta\ \sin\psi\ \cos\psi\ \sin\theta\ \cos\theta\ \sin\varphi\ \cos\varphi\ ]^T$

often used sines and cosines, *yhlp*

Outputs

$\mathbf{y}_{xyH} = [\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$ *(part of* $\dot{\mathbf{x}}$*)* — *time-derivatives of* x_e*,* y_e*, and* H*, yxyH*

Parameters

none

Connections

in: $\mathbf{x}$ comes from the block Integrator (Equations of Motion); $\mathbf{F}_{tot}$ and $\mathbf{M}_{tot}$ come from FMsort; $\mathbf{y}_{hlp}$ comes from Hlpfcn; $\mathbf{y}^*_{bvel}$ is the sum of the output from uvw (Equations of Motion) and the wind velocity components from the external input vector $\mathbf{u}_{wind}$

out: $\mathbf{y}_{xyH}$ is *Muxed* together with the time-derivatives of the other state variables into the vector $\dot{\mathbf{x}}$ (not corrected for the implicit nature of the $\dot{\beta}$-equation), which is connected to xdotcorr (Equations of Motion)

Enter `type xyhdot.hlp` at the command-line for on-line help. □

Chapter 6

FDC implementation of the atmospheric disturbance models

The FDC library WINDLIB contains the SIMULINK implementation of the wind and turbulence models from section 3.3. Figure 6.1 shows the main window of this library, while figures 6.2 and 6.3 show the wind and turbulence sublibraries. The main library can be opened by typing `windlib` at the MATLAB command-line, or via the main FDC library FDCLIB, which can be opened by typing `fdclib`. This chapter describes the different blocks from the library WINDLIB in alphabetical order.

See section 5.2 for the typographical conventions used in this chapter.

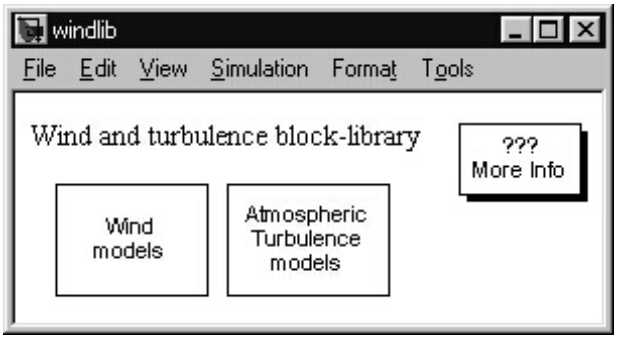


Figure 6.1: Wind and turbulence library WINDLIB

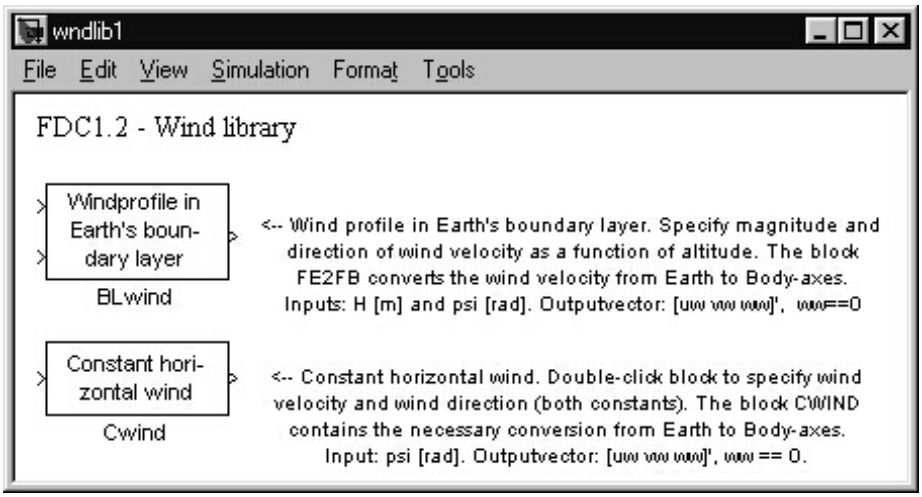


Figure 6.2: Sublibrary with wind-models

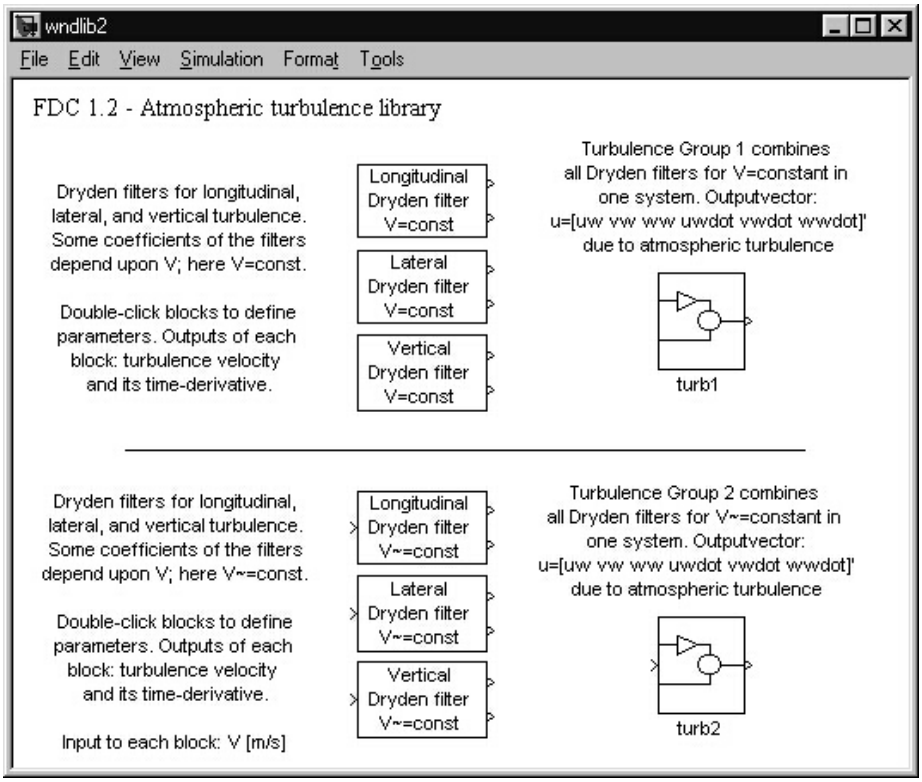


Figure 6.3: Sublibrary with turbulence models

BLWIND

Library WINDLIB / wind / BLWIND

Type

Masked subsystem block.

Description

The block BLWIND calculates components of the wind velocity along the aircraft's body-axes in the boundary layer of the Earth ('BL' = Boundary Layer), which is about 300 m high. The wind velocity has been defined as a function of the altitude, while the wind direction can also be defined as a function of the altitude by the user. By default, a wind profile according to ref.[1] has been implemented.

Equations

• Total wind velocity in the boundary layer of the Earth, $[ms^{-1}]$:

$$\begin{aligned} V_w &= V_{w\,9.15}\frac{H^{0.2545}-0.4097}{1.3470} && (0 < h < 300m) \\ V_w &= 2.86585\,V_{w_{9.15}} && (h \geq 300m) \end{aligned}$$

$V_{w\,9.15}$ is the wind speed at 9.15 m altitude.

• Wind velocity components along the aircraft's body-axes, $[ms^{-1}]$:

$$\begin{aligned} u_w &= V_w\cos(\psi_w-\pi)\cos\psi + V_w\sin(\psi_w-\pi)\sin\psi \\ v_w &= -V_w\cos(\psi_w-\pi)\sin\psi + V_w\sin(\psi_w-\pi)\cos\psi \end{aligned}$$

where ψ_w is the wind direction with respect to the Earth ($\psi_w = 0$ *rad* if the wind is blowing to the *South*).[1]

Inputs

H	altitude, *H*
ψ	heading, *psi*

Outputs

$[\ u_w\ \ v_w\ \ w_w\ ]^T$	wind velocities along body-axes, *[uw,vw,ww]'*

Parameters

BLWIND does not require parameters to be specified, but the user can change the equations for determining the wind velocity and direction if required by double-clicking the appropriate blocks.

Connections

in: H and ψ are usually extracted from the non-linear aircraft model (enter `type outputs.hlp` at the command-line for more information about the outputs from the aircraft model)

out: the wind velocity components can be used as input signals to the aircraft model (note: due to the non-constant wind it is necessary to enter the time-derivatives of these signals too; send the outputs from BLWIND through Derivative blocks from the SIMULINK library Linear and *Mux* these results together with the velocity components themselves)

Enter `type blwind.hlp` at the command-line for on-line help. □

[1] This axis transformation is performed in the subsystem Wind to body-axes within the block BLWIND.

CWIND

Library WINDLIB / wind / CWIND

Type

Masked subsystem block.

Description

The block CWIND computes components of the wind velocity along the aircraft's body-axes for a constant horizontal wind ('C' = Constant). The user must specify the wind direction (being the direction where the wind is blowing *from*!) and the wind speed after double-clicking the block.

Equations

• Wind velocity components along the aircraft's body-axes, $[ms^{-1}]$:

$$\begin{aligned} u_w &= V_w \cos(\psi_w - \pi)\cos\psi + V_w \sin(\psi_w - \pi)\sin\psi \\ v_w &= -V_w \cos(\psi_w - \pi)\sin\psi + V_w \sin(\psi_w - \pi)\cos\psi \end{aligned}$$

where ψ_w is the wind direction with respect to the Earth ($\psi_w = 0$ *rad* if the wind is blowing to the *South*).[1]

Inputs

ψ	heading, *psi*

Outputs

$[\ u_w\ \ v_w\ \ w_w\ ]^T$	wind velocities along body-axes, *[uw,vw,ww]'*

Parameters

The user must specify the wind velocity and wind direction by double-clicking the block CWIND.

Connections

in: ψ is usually extracted from the non-linear aircraft model (enter `type outputs.hlp` at the command-line for more information about the outputs from the aircraft model)

out: the wind velocity components can be used as input signals to the aircraft model

Enter `type cwind.hlp` at the command-line for on-line help. □

[1] Contrary to the block BLWIND, CWIND does not use an additional internal masked subsystem block for the implementation of this transformation from Earth to body-axes.

UDRYD1

Library WINDLIB / turbulence / UDRYD1

Type

Masked subsystem block.

Description

The block UDRYD1 generates a turbulence velocity component and its time-derivative along the X_B axis of the aircraft, using a longitudinal Dryden filter with constant coefficients. The user must specify the longitudinal scale-length of the turbulence L_{u_g}, the standard deviation σ_{u_g}, and the expected mean airspeed of the aircraft. Variations of the filter coefficients with the airspeed are not taken into account by UDRYD1. These variations usually are very small.

Equations

• Longitudinal turbulence velocity, $[ms^{-1}]$:

$$u_g(s) = H_{u_g w_1}\, w_1(s)$$

where w_1 is a white noise signal, generated internally within the block UDRYD1 and $H_{u_g w_1}$ is the transfer function of the longitudinal turbulence velocity filter.

• Transfer function of longitudinal turbulence filter:

$$H_{u_g w_1}(s) = \sigma_u \sqrt{\frac{2L_u}{V}}\, \frac{1}{1 + \frac{L_u}{V} s}$$

Note: the value of V used by UDRYD1 is kept constant during the simulations; it must be specified by the user.

Inputs

none

Outputs

u_g	longitudinal turbulence velocity, *ug*
$\dot{u}_g$	time-derivative of longitudinal turbulence velocity, *ug dot*

Parameters

The user must specify the scale length L_{ug}, the standard deviation σ_{ug}, and the estimated mean value of the true airspeed for which the motions are evaluated by double-clicking the block UDRYD1.

Connections

in: no connections

out: u_g and $\dot{u}_g$ are usually *Muxed* together with other turbulence velocities and their time-derivatives; they can be used as inputs to the aircraft model

Enter `type udryd1.hlp` at the command-line for on-line help. □

UDRYD2

Library WINDLIB / turbulence / UDRYD2

Type

Masked subsystem block.

Description

The block UDRYD2 generates a turbulence velocity component and its time-derivative along the X_B axis of the aircraft, using a longitudinal Dryden filter with airspeed-dependent coefficients. The user must specify the longitudinal scale-length of the turbulence L_{u_g} and the standard deviation σ_{u_g}, but no mean airspeed, since the filter coefficients are updated during the simulations as a function of the *actual* airspeed, which is used as input signal.

Equations

• Longitudinal turbulence velocity, $[ms^{-1}]$:

$$u_g(s) = H_{u_g w_1}\, w_1(s)$$

where w_1 is a white noise signal, generated internally within the block UDRYD2 and $H_{u_g w_1}$ is the transfer function of the longitudinal turbulence velocity filter.

• Transfer function of longitudinal turbulence filter:

$$H_{u_g w_1}(s) = \sigma_u \sqrt{\frac{2L_u}{V}}\, \frac{1}{1 + \frac{L_u}{V} s}$$

Note: since the value of V used by UDRYD2 is *not* kept constant during the simulations, a block-diagram equivalent of this transfer function has been created using the theory from section 4.2.6 in chapter 4. The gains from this block-diagram equivalent are 'scheduled' as a function of the current value of the airspeed V.

Inputs

V	true airspeed, V

Outputs

u_g	longitudinal turbulence velocity, *ug*
$\dot{u}_g$	time-derivative of longitudinal turbulence velocity, *ug dot*

Parameters

The user must specify the scale length L_{ug} and the standard deviation σ_{ug} by double-clicking the block UDRYD2.

Connections

in: V is usually extracted from the non-linear aircraft model

out: u_g and $\dot{u}_g$ are usually *Muxed* together with other turbulence velocities and their time-derivatives; they can be used as inputs to the aircraft model

Enter `type udryd2.hlp` at the command-line for on-line help. □

VDRYD1

Library WINDLIB / turbulence / VDRYD1

Type

Masked subsystem block.

Description

The block VDRYD1 generates a turbulence velocity component and its time-derivative along the Y_B axis of the aircraft, using a lateral Dryden filter with constant coefficients. The user must specify the lateral scale-length of the turbulence L_{v_g}, the standard deviation σ_{v_g}, and the expected mean airspeed of the aircraft. Variations of the filter coefficients with the airspeed are not taken into account by VDRYD1. These variations usually are very small.

Equations

- Lateral turbulence velocity, $[ms^{-1}]$:

$$v_g(s) = H_{v_g w_2}\, w_2(s)$$

where w_2 is a white noise signal, generated internally within the block VDRYD1 and $H_{v_g w_2}$ is the transfer function of the lateral turbulence velocity filter.

- Transfer function of lateral turbulence filter:

$$H_{v_g w_2}(s) = \sigma_v \sqrt{\frac{2L_v}{V}}\, \frac{1 + \sqrt{3}\frac{L_v}{V}s}{\left(1 + \frac{L_v}{V}s\right)^2}$$

Note: the value of V used by VDRYD1 is kept constant during the simulations; it must be specified by the user.

Inputs

none

Outputs

v_g	lateral turbulence velocity, *vg*
$\dot{v}_g$	time-derivative of lateral turbulence velocity, *vg dot*

Parameters

The user must specify the scale length L_{vg}, the standard deviation σ_{vg}, and the estimated mean value of the true airspeed for which the motions are evaluated by double-clicking the block VDRYD1.

Connections

in: no connections

out: v_g and $\dot{v}_g$ are usually *Muxed* together with other turbulence velocities and their time-derivatives; they can be used as inputs to the aircraft model

Enter `type vdryd1.hlp` at the command-line for on-line help. □

VDRYD2

Library WINDLIB / turbulence / VDRYD2

Type

Masked subsystem block.

Description

The block VDRYD2 generates a turbulence velocity component and its time-derivative along the Y_B axis of the aircraft, using a lateral Dryden filter with airspeed-dependent coefficients. The user must specify the lateral scale-length of the turbulence L_{v_g} and the standard deviation σ_{v_g}, but no mean airspeed, since the filter coefficients are updated during the simulations as a function of the *actual* airspeed, which is used as input signal.

Equations

- Lateral turbulence velocity, $[ms^{-1}]$:

$$v_g(s) = H_{v_g w_2}\, w_2(s)$$

where w_2 is a white noise signal, generated internally within the block VDRYD2 and $H_{v_g w_2}$ is the transfer function of the lateral turbulence velocity filter.

- Transfer function of lateral turbulence filter:

$$H_{v_g w_2}(s) = \sigma_v \sqrt{\frac{2L_v}{V}}\, \frac{1 + \sqrt{3}\frac{L_v}{V}s}{\left(1 + \frac{L_v}{V}s\right)^2}$$

Note: since the value of V used by VDRYD2 is *not* kept constant during the simulations, a block-diagram equivalent of this transfer function has been created using the theory from section 4.2.6 in chapter 4. The gains from this block-diagram equivalent are 'scheduled' as a function of the current value of the airspeed V.

Inputs

V	true airspeed, V

Outputs

v_g	lateral turbulence velocity, *vg*
$\dot{v}_g$	time-derivative of lateral turbulence velocity, *vg dot*

Parameters

The user must specify the scale length L_{v_g} and the standard deviation σ_{v_g} by double-clicking the block VDRYD2.

Connections

in: V is usually extracted from the non-linear aircraft model

out: v_g and $\dot{v}_g$ are usually *Muxed* together with other turbulence velocities and their time-derivatives; they can be used as inputs to the aircraft model

Enter `type vdryd2.hlp` at the command-line for on-line help. □

WDRYD1

Library WINDLIB / turbulence / WDRYD1

Type

Masked subsystem block.

Description

The block WDRYD1 generates a turbulence velocity component and its time-derivative along the Z_B axis of the aircraft, using a vertical Dryden filter with constant coefficients. The user must specify the vertical scale-length of the turbulence L_{w_g}, the standard deviation σ_{w_g}, and the expected mean airspeed of the aircraft. Variations of the filter coefficients with the airspeed are not taken into account by WDRYD1. These variations usually are very small.

Equations

- Vertical turbulence velocity, $[ms^{-1}]$:

$$w_g(s) = H_{w_g w_3}\, w_3(s)$$

where w_3 is a white noise signal, generated internally within the block WDRYD1 and $H_{w_g w_3}$ is the transfer function of the vertical turbulence velocity filter.

- Transfer function of vertical turbulence filter:

$$H_{w_g w_3}(s) = \sigma_w \sqrt{\frac{2L_w}{V}}\, \frac{1 + \sqrt{3}\frac{L_w}{V}s}{\left(1 + \frac{L_w}{V}s\right)^2}$$

Note: the value of V used by WDRYD1 is kept constant during the simulations; it must be specified by the user.

Inputs

none

Outputs

w_g	vertical turbulence velocity, *wg*
$\dot{w}_g$	time-derivative of vertical turbulence velocity, *wg dot*

Parameters

The user must specify the scale length L_{wg}, the standard deviation σ_{wg}, and the estimated mean value of the true airspeed for which the motions are evaluated by double-clicking the block WDRYD1.

Connections

in: no connections

out: w_g and $\dot{w}_g$ are usually *Muxed* together with other turbulence velocities and their time-derivatives; they can be used as inputs to the aircraft model

Enter `type wdryd1.hlp` at the command-line for on-line help. □

WDRYD2

Library WINDLIB / turbulence / WDRYD2

Type

Masked subsystem block.

Description

The block WDRYD2 generates a turbulence velocity component and its time-derivative along the Z_B axis of the aircraft, using a vertical Dryden filter with airspeed-dependent coefficients. The user must specify the vertical scale-length of the turbulence L_{w_g} and the standard deviation σ_{w_g}, but no mean airspeed, since the filter coefficients are updated during the simulations as a function of the *actual* airspeed, which is used as input signal.

Equations

- Vertical turbulence velocity, $[ms^{-1}]$:

$$w_g(s) = H_{w_g w_3}\, w_3(s)$$

where w_3 is a white noise signal, generated internally within the block WDRYD1 and $H_{w_g w_3}$ is the transfer function of the vertical turbulence velocity filter.

- Transfer function of vertical turbulence filter:

$$H_{w_g w_3}(s) = \sigma_w \sqrt{\frac{2L_w}{V}}\, \frac{1 + \sqrt{3}\frac{L_w}{V}s}{\left(1 + \frac{L_w}{V}s\right)^2}$$

Note: since the value of V used by WDRYD2 is *not* kept constant during the simulations, a block-diagram equivalent of this transfer function has been created using the theory from section 4.2.6 in chapter 4. The gains from this block-diagram equivalent are 'scheduled' as a function of the current value of the airspeed V.

Inputs

V	true airspeed, V

Outputs

w_g	vertical turbulence velocity, *wg*
$\dot{w}_g$	time-derivative of vertical turbulence velocity, *wg dot*

Parameters

The user must specify the scale length L_{wg} and the standard deviation σ_{wg} by double-clicking the block WDRYD2.

Connections

in: V is usually extracted from the non-linear aircraft model

out: w_g and $\dot{w}_g$ are usually *Muxed* together with other turbulence velocities and their time-derivatives; they can be used as inputs to the aircraft model

Enter `type wdryd2.hlp` at the command-line for on-line help. □

Chapter 7

FDC implementation of the radio-navigation models

The FDC library NAVLIB contains simulation models of radio-navigation tools, described in section 3.4. Figure 7.1 shows the main window of this library, while figures 7.2 and 7.3 show the sublibraries with the ILS and VOR models, respectively. The main library can be opened by typing `navlib` at the MATLAB command-line, or via the main FDC library FDCLIB, which can be opened by typing `fdclib`. This chapter describes the different blocks from the library NAVLIB in alphabetical order.

See section 5.2 for the typographical conventions used in this chapter.

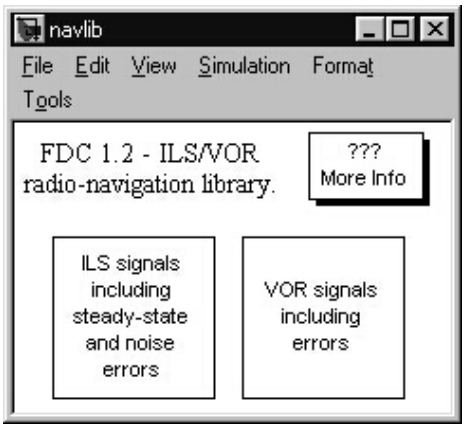


Figure 7.1: Radio-navigation library NAVLIB

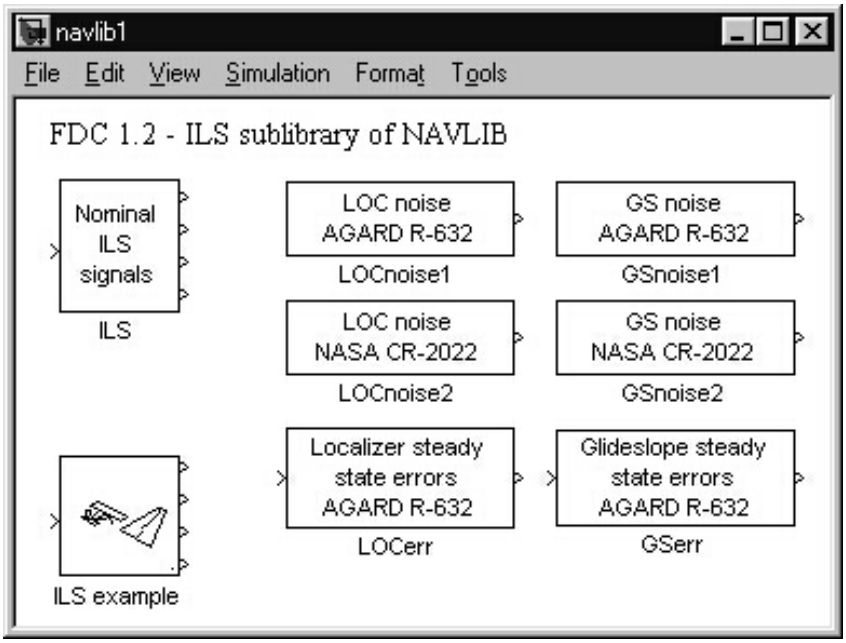


Figure 7.2: Sublibrary with ILS models

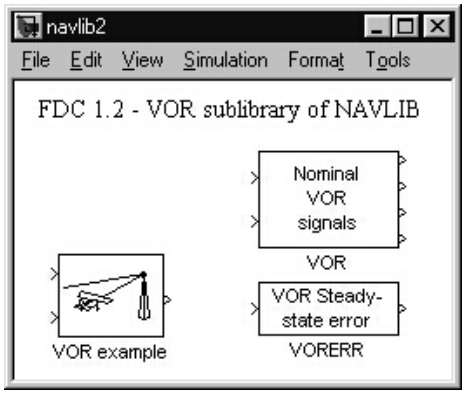


Figure 7.3: Sublibrary with VOR models

GSERR

Library NAVLIB / ILS / GSERR

Type

Masked subsystem block.

Description

The block GSERR contains the formula's for steady-state errors in the glideslope signal. Notice that the equations are expressed in terms of glideslope *currents* through the cockpit indicator in the aircraft!

Equations

• glideslope signal with steady-state errors, $[\mu A]$:

$$i_{gs,actual} = K_{S_{gs}} \left(i_{gs,nominal} + \Delta i_{gs} \right)$$

where the multiplication factor $K_{S_{gs}}$ takes into account the offset in the glideslope sensitivity S_{gs} and Δi_{gs} is an error signal that is caused by an offset in the nominal glideslope elevation angle γ_{gs} (the user must specify the offset-values as percentages of the maximum allowable values, according to table 3.2 from chapter 3; GSERR converts all signals to *currents* through the glideslope indicator in the aircraft).

Inputs

$i_{gs,nominal}$	nomimal glideslope current, *igs (nominal)*

Outputs

$i_{gs,actual}$	glideslope current with steady-state errors, *igs (actual)*

Parameters

The user must specify the ILS performance category (1, 2, or 3), the offset in S_{gs} in terms of percents of its maximum allowable error according to table 3.2, and the glideslope misalignment in terms of percents of its maximum allowable value.

Connections

in: $i_{gs,nominal}$ comes from the block ILS, which determines the nominal ILS signals

out: $i_{gs,actual}$ can be connected to a sum block for adding the glideslope noise signal (the resulting signal can be used as an input to the control laws of an automatic approach system)

Enter `type gserr.hlp` at the command-line for on-line help. □

GSNOISE

Library NAVLIB / ILS / GSNOISE

Type

Masked subsystem blocks.

Description

The GSNOISE blocks contain glideslope noise models. There are two versions: GSNOISE1, based upon the noise models from ref.[1] (AGARD R-632), and GSNOISE2, based upon ref.[17] (NASA CR-2022).

Equations

- Glideslope noise, $[\mu A]$:

$$\Delta i^*_{gs}(s) = H_{gs}\, w'_1(s)$$

where Δi^*_{gs} is the glideslope noise, w'_1 is a white noise signal, generated internally within the block GSNOISE1 or GSNOISE2, and H_{gs} is the transfer function of the glideslope noise filter.

- Transfer function of the glideslope noise filter according to AGARD R-632:

$$H_{gs}(s) = \sigma_{gs}\sqrt{\frac{2L_{gs}}{V}}\,\frac{1}{1+\frac{L_{gs}}{V}s}$$

Note: the value of V used by GSNOISE1 is kept constant during the simulations. It is equaled to the approach speed which must be specified by the user.

- Transfer function of the glideslope noise filter according to NASA CR-2022:

$$H_{gs}(s) = \frac{3.9875}{0.25+s}$$

Inputs

none

Outputs

Δi^*_{gs}	Glideslope noise, *D_igs**

Parameters

For the AGARD R-632 version, the user must specify the scale length L_{gs}, the standard deviation σ_{gs}, and the approach speed. For the NASA CR-2022 version no parameters have to entered.

Connections

in: no connections

out: Δi^*_{gs} must be connected to a sum block for adding it to the original glideslope current signal i_{gs} (note: this must be done *after* taking into account possible steady-state errors by the block GSERR!)

Enter `type gsnoise.hlp` at the command-line for on-line help. □

ILS

Library NAVLIB / ILS / ILS

Type

Masked subsystem block.

Description

The block ILS is used to determine the nominal ILS signals for a given position of the aircraft. The block also computes some closely associated properties, which provide more information about the current position of the aircraft with respect to the runway and the glideslope and localizer reference planes. The validity of the signals is checked by an included masked subsystem block, called ILSTEST.

Equations

- Currents through the ILS indicators in the cockpit, $[\mu A]$:

$$\begin{aligned} i_{gs} &= S_{gs}\,\varepsilon_{gs} \\ i_{loc} &= S_{loc}\,\Gamma_{loc} \end{aligned}$$

- Angles between nominal and current ILS planes, $[rad]$:

$$\begin{aligned} \varepsilon_{gs} &= \gamma_{gs} + \arctan\left(\frac{H_f}{R_{gs}}\right) \\ \Gamma_{loc} &= \arcsin\left(\frac{d_{loc}}{R_{loc}}\right) \end{aligned}$$

- X and Y-coordinates of the aircraft with respect to the runway fixed reference frame, $[m]$:

$$\begin{aligned} x_f &= (x_e - x_{RW})\cos\psi_{RW} + (y_e - y_{RW})\sin\psi_{RW} \\ y_f &= -(x_e - x_{RW})\sin\psi_{RW} + (y_e - y_{RW})\cos\psi_{RW} \end{aligned}$$

- Height of the aircraft above aerodrome level, $[m]$:

$$H_f = H - H_{RW}$$

- Distance from the aircraft to the glide-path ($\perp$ nominal glide-path), $[m]$:

$$d_{gs} = (R_{gs}\tan\gamma_{gs} + H_f)\cos\gamma_{gs}$$

- Ground-distances from the aircraft to the glideslope and localizer transmitters, $[m]$:

$$\begin{aligned} R_{gs} &= \sqrt{(x_{gs} - x_f)^2 + (y_f - y_{gs})^2} \\ R_{loc} &= \sqrt{{y_f}^2 + (x_{loc} - x_f)^2} \end{aligned}$$

- Two flags which determine whether the glideslope and localizer signals can be received with sufficient accuracy are computed by evaluating logical Boolean expressions, derived from the glideslope and localizer coverage from figures 3.11 and 3.9 (chapter 3). The glideslope signal is valid if the flag *GS_flag* is equal to 1; the localizer signal is valid if *LOC_flag* is equal to 1. If not, the aircraft flies outside their respective ranges. Note: these flags are computed in the internal masked subsystem block ILSTEST.

Inputs

$\mathbf{u}_{ils} = [\ x_e\ y_e\ H\ ]^T$ — *aircraft coordinates and altitude, uils*

Outputs

$\mathbf{y}_{ils\ 1} = [\ i_{gs}\ i_{loc}\ ]^T$ — *nominal currents through ILS indicators, yils1*

$\mathbf{y}_{ils\ 2} = [\ \varepsilon_{gs}\ \Gamma_{loc}\ ]^T$ — *angles between nominal and current ILS planes, yils2*

$\mathbf{y}_{ils\ 3} = [\ x_f\ y_f\ H_f\ d_{gs}\ R_{gs}\ R_{loc}\ ]^T$ — *distances defining the aircraft's approach position, yils3*

$\mathbf{y}_{ils4} = [\ GS\_flag\ LOC\_flag\ ]^T$ — *flags defining the validity of the ILS signals, yils4*

Note: section 3.4.1 gives the exact definitions of all variables from these equations!

Parameters

The user must define the following parameters by double-clicking the block ILS:

- runway heading ψ_{RW}
- the coordinates x_{RW}, y_{RW}, and H_{RW} of the origin of the runway-fixed reference frame, measured with respect to the Earth-axes,
- the distance x_{loc} from the runway threshold to the localizer antenna, measured along the runway centerline,
- the distance x_{gs} from the threshold to the projection of the glideslope antenna upon the centerline,
- the perpendicular distance y_{gs} from the centerline to the glideslope antenna,
- the nominal glideslope angle γ_{gs}.

See also section 3.4.1.

Connections

in: $\mathbf{u}_{ils}$ is usually obtained from the non-linear aircraft model

out: $\mathbf{y}_{ils\ 1}$ is connected to the steady-state ILS error blocks GSERR and LOCERR, because those blocks express errors in terms of ILS currents; the other output vectors are primarily used for evaluations of simulation results and are therefore connected to a To Workspace

Enter `type ils.hlp` at the command-line for on-line help about the block ILS. Enter `type ilstest.hlp` for more information about the validity-check of the ILS signals in the subsystem ILSTEST. □

ILS example

Library NAVLIB / ILS / ILS example

Type

Masked subsystem block (contents accessible without *unmasking*).

Description

The subsystem ILS example demonstrates how to combine the different ILS-related blocks to obtain a realistic ILS simulation model. This block is used within the 'Beaver' autopilot models APILOT2 and APILOT3 with a slightly different input definition. The nominal ILS signals are sent to the MATLAB workspace by means of a To Workspace block; it is easy to add similar blocks for noise and offset signals if required.

Subsystems/masked blocks

The subsystem ILS example contains five masked subsystem blocks:

ILS:	computes the nominal ILS signals
GSERR:	takes into account steady-state errors in the glideslope signal
GSNOISE:	computes glideslope noise
LOCERR:	takes into account steady-state errors in the localizer signal
LOCNOISE:	computes localizer noise

Notice that the disturbance models used in ILS example are all based upon ref.[1].

The values of the glideslope and localizer *currents* that leave the error blocks are converted back to the error angles ε_{gs} and Γ_{loc} by means of the multiplication factors $1/K_{gs}$ and $1/K_{loc}$, respectively. ILS example also uses approximating differentiating filters $\frac{1}{s+1}$ in order to determine the time-derivatives of the ILS signals (these approximated time-derivatives were needed for the 'Beaver' autopilot models APILOT2 and APILOT3, see sections 11.3.4 and 11.4.3 for more details).

Inputs

$\mathbf{u}_{ils} = [\ x_e\ y_e\ H\ ]^T$ *aircraft coordinates and altitude, uils*

Outputs

ε_{gs}	glideslope error angle with respect to nominal glide-path, *epsilon_gs*
$\dot{\varepsilon}_{gs}$	approximated time-derivative of ε_{gs}, *d(epsilon_gs)/dt*
Γ_{loc}	localizer error angle with respect to runway centerline, *Gamma_loc*
$\dot{\Gamma}_{loc}$	approximated time-derivative of Γ_{loc}, *d(Gamma_loc)/dt*

Note: *all* outputs from the block ILS are sent to the variable *yils* in the MATLAB workspace by means of a To Workspace block. For storing other signals from ILS example you must include more To Workspace blocks. You may also want to create a time-basis by means of a Clock that is connected to a To Workspace block, but if you connect ILS example to the non-linear aircraft model this is already being taken care of within the aircraft model itself (see section 5.1 for more details).

Parameters

The user must specify the properties of the ILS errors by double-clicking the blocks GSERR, GSNOISE, LOCERR, and LOCNOISE. The geometrical properties of the ILS system normally would have to be entered by the user after double-clicking the block ILS, but for this example they are extracted from the MATLAB workspace, using the following variables:

gamgs	nominal glideslope angle, [*deg*] (!)
HRW	elevation of the runway, [*m*]
psiRW	heading of the runway, [*deg*] (!)
xgs	X-coordinate of the glideslope transmitter in runway-axes, [m]
xloc	X-coordinate of the localizer transmitter in runway-axes, [m]
xRW	X-coordinate of origin of runway-axes with respect to Earth-axes, [m]
ygs	X-coordinate of the glideslope transmitter in runway-axes, [m]
yRW	Y-coordinate of origin of runway-axes with respect to Earth-axes, [m]

Furthermore, the sensitivity of the glideslope and localizer systems must be defined in the variables *Sgs* and *Sloc* within the MATLAB workspace.[1]

[1] Although these sensitivity values were already computed in the block ILS, we also need them for recovering the angles ε_{gs} and Γ_{loc} from the ILS *currents* which are obtained by the ILS error blocks. This is due to the fact that the approach control laws of the 'Beaver' were based upon these angles, rather than the ILS currents.

In order to facilitate this definition, a MATLAB macro ILSINIT has been created. Run this program by typing `ilsinit` at the command-line and you will be asked automatically to specify all parameters. You can also run APINIT and then select the option *Initialize VOR and/or ILS systems*. Type `help ilsinit` or `help apinit` for on-line help. See also section 12.4.1.

Connections

in: $\mathbf{u}_{ils}$ is usually extracted from the non-linear aircraft model

out: the outputs from ILS example can be connected to control system blocks, as demonstrated in the systems APILOT2 and APILOT3, or sent to the workspace by means of a To Workspace block

Enter `type ilsxmpl.hlp` at the command-line for on-line help. □

LOCERR

Library NAVLIB / ILS / LOCERR

Type

Masked subsystem block.

Description

The block LOCERR contains the formula's for steady-state errors in the localizer signal. Notice that the equations are expressed in terms of localizer *currents* through the cockpit indicator in the aircraft!

Equations

• localizer signal with steady-state errors, $[\mu A]$:

$$i_{loc,actual} = K_{S_{loc}} \left(i_{loc,nominal} + \Delta i_{loc} \right)$$

where the multiplication factor $K_{S_{loc}}$ takes into account the offset in the localizer sensitivity S_{loc} and Δi_{loc} is an offset in the localizer reference plane, i.e. a deviation from runway centerline (the user must specify the offset-values as percentages of the maximum allowable values, according to table 3.1 from chapter 3).

Inputs

$i_{loc,nominal}$	nomimal localizer current, *iloc (nominal)*

Outputs

$i_{loc,actual}$	localizer current with steady-state errors, *iloc (actual)*

Parameters

The user must specify the performance category (1, 2, or 3), the percentage of maximum allowable offset in localizer sensitivity, the percentage of maximum allowable localizer misalignment and the distance from the runway threshold to the localizer antenna by double-clicking the block LOCERR.

Connections

in: $i_{loc,nominal}$ comes from the block ILS, which determines the nominal ILS signals

out: $i_{loc,actual}$ can be connected to a sum block for adding the localizer noise signal (the resulting signal can be used as an input to the control laws of an automatic approach system)

Enter `type locerr.hlp` at the command-line for on-line help. □

LOCNOISE

Library NAVLIB / ILS / LOCNOISE

Type

Masked subsystem blocks.

Description

The LOCNOISE blocks contain glideslope noise models. There are two versions: LOCNOISE1, based upon the noise models from ref.[1] (AGARD R-632), and LOCNOISE2, based upon ref.[17] (NASA CR-2022).

Equations

- Localizer noise, $[\mu A]$:

$$\Delta i^*_{loc}(s) = H_{loc}\, w'_2(s)$$

where Δi^*_{loc} is the localizer noise, w'_2 is a white noise signal, generated internally within the block LOCNOISE1 or LOCNOISE2, and H_{loc} is the transfer function of the localizer noise filter.

- Transfer function of the localizer noise filter according to AGARD R-632:

$$H_{loc}(s) = \sigma_{loc}\sqrt{\frac{2L_{loc}}{V}}\,\frac{1}{1+\frac{L_{loc}}{V}s}$$

Note: the value of V used by LOCNOISE1 is kept constant during the simulations. It is equaled to the approach speed which must be specified by the user.

- Transfer function of the localizer noise filter according to NASA CR-2022:

$$H_{loc}(s) = \frac{5\,(1.5+s)}{(0.35+s)(10+s)}$$

Inputs

none

Outputs

Δi^*_{loc}	Localizer noise, *D_iloc**

Parameters

For the AGARD R-632 version, the user must specify the scale length L_{loc}, the standard deviation σ_{loc}, and the approach speed. For the NASA CR-2022 version, no parameters have to entered.

Connections

in: no connections

out: Δi^*_{loc} must be connected to a sum block for adding it to the original localizer current signal i_{loc} (note: this must be done *after* taking into account possible steady-state errors by the block LOCERR!)

Enter `type locnoise.hlp` at the command-line for on-line help. □

VOR

Library NAVLIB / VOR / VOR

Type

Masked subsystem block.

Description

The block VOR computes the nominal VOR signals which an aircraft will receive if it flies at a certain position with respect to the VOR station. In addition, the ground-distance to the VOR station, two flags which define if the VOR signal is valid, and a *To/From* flag are computed.

Equations

- current VOR bearing on which the aircraft is flying, [*rad*]:

$$QDR = \arctan\left(\frac{y_e - y_{VOR}}{x_e - x_{VOR}}\right)$$

- deviation from desired VOR bearing, [*rad*]:

$$\Gamma_{VOR} = CD - QDR$$

- ground-distance from the aircraft to the VOR station, [m]:

$$R_{VOR} = \sqrt{(x_e - x_{VOR})^2 + (y_e - y_{VOR})^2}$$

- the cone of silence flag is set to 1 if the aircraft enters the cone of silence, i.e. if:

$$\arctan\left(\frac{H - H_{VOR}}{R_{VOR}}\right) > 90^\circ - (40^\circ \text{ to } 60^\circ)$$

- the range flag is set to 1 if the aircraft flies outside the area where the VOR signals can be received with appropriate accuracy, i.e. if:

$$R_{VOR} > Range$$

where:

$$Range = 1000\ \left(-2.3570 \cdot 10^{-6}(H - H_{VOR})^2 + 5.7087 \cdot 10^{-2}(H - H_{VOR}) + 80.8612\right)$$

- the To/From flag is set to 1 if the aircraft flies *to* the VOR station, i.e. if:

$$|\psi - QDR| > 90^\circ$$

Inputs

$\mathbf{u}_{VOR} = [\ x_e\ y_e\ H\ ]^T$	*coordinates of the aircraft with respect to the Earth, uVOR*
ψ	heading of the aircraft, *psi*

Outputs

$\mathbf{y}_{VOR1} = \Gamma_{VOR}$	nominal VOR angle, *Gamma_VOR*
$\mathbf{y}_{VOR2} = R_{VOR}$	ground distance from aircraft to VOR station, *R_VOR*
$\mathbf{y}_{VOR3} = [$ *Cone-of-silence flag, Range flag* $]^T$	*flags specifying VOR validity, yVOR3*
$\mathbf{y}_{VOR4} =$ *ToFrom*	To/From flag, *ToFrom*

Parameters

The user must specify the X and Y coordinates of the VOR station, measured relative to the initial position of the aircraft, the elevation of the VOR station, and the Course Datum (reference bearing on which the aircraft should fly).

Connections

in: $\mathbf{u}_{VOR}$ and ψ are usually extracted from the non-linear aircraft model

out: $\mathbf{y}_{VOR1}$ is connected to the block VORERR to take into account steady-state errors in the VOR signal; the other outputs can be used by control law blocks and/or sent to the workspace by means of To Workspace blocks

Enter `type vor.hlp` at the command-line for on-line help. □

VORERR

Library NAVLIB / VOR / VORERR

Type

Masked subsystem block.

Description

The block VORERR implements a steady-state error in the VOR signal by multiplying the VOR signal with a gain value that slightly differs from 1. Note: FDC 1.2 currently does not contain more accurate steady-state error models for the VOR system, nor does it contain VOR noise models.

Equations

- VOR signal with steady-state error, $[rad]$:

$$\Gamma_{VOR,actual} = K_{VOR_{err}} \cdot \Gamma_{VOR,nominal}$$

where $K_{VOR_{err}}$ is equal to 1 plus the overall percentile VOR system error.

Inputs

$\Gamma_{VOR,nominal}$	nominal VOR signal, *Gamma_VOR (nominal)*

Outputs

$\Gamma_{VOR,actual}$	VOR signal with steady-state errors, *Gamma_VOR (actual)*

Parameters

The user must specify the overall percentile VOR system error by double-clicking the block VORERR.

Connections

in: $\Gamma_{VOR,nominal}$ is retrieved from the block VOR, which computes the nominal VOR signals
out: $\Gamma_{VOR,actual}$ will usually be connected to some control law, or sent to the MATLAB workspace

Enter `type vorerr.hlp` at the command-line for on-line help. □

VOR example

Library NAVLIB / VOR / VOR example

Type

Masked subsystem block (contents accessible without *unmasking*).

Description

The subsystem VOR example shows how to combine the different VOR-related blocks into one complete VOR simulation model.

Equations

The subsystem VOR example contains two masked subsystem blocks:

VOR: computes the nominal VOR signal
VORERR: takes into account steady-state errors in the VOR signal

Inputs

$\mathbf{u}_{VOR} = [\ x_e\ y_e\ H\ ]^T$ — *aircraft coordinates and altitude, uVOR*

ψ — heading of the aircraft, *psi*

Outputs

Γ_{VOR} — nominal VOR angle, *Gamma_VOR*

Note: *all* outputs from the block VOR are sent to the variable *yvor* in the MATLAB workspace by means of a To Workspace block. For storing other signals you must include more To Workspace blocks. You may also want to create a time-basis by means of a Clock that is connected to a To Workspace block, but if you connect VOR example to the non-linear aircraft model this is already being taken care of within the aircraft model (see section 5.1 for more details).

Parameters

The user must specify the properties of the steady-state errors in the VOR signal by double-clicking the block VORERR. The geometrical data which determine the nominal VOR signal would normally have to be entered by the user after double-clicking the block VOR, but for this example they are extracted from the MATLAB workspace, using the following variables:

CD	Course Datum (reference value for the VOR radial), [*deg*] (!)
HVOR	altitude of the VOR transmitter above sea level, [m]
xVOR	X-coordinate of VOR transmitter relative to the aircraft at $t = 0$ *sec*, [m]
yVOR	Y-coordinate of VOR transmitter relative to the aircraft at $t = 0$ *sec*, [m]

Connections

in: $\mathbf{u}_{VOR}$ and ψ are usually extracted from the non-linear aircraft model

out: the output signal from VOR example can be connected to control system blocks, as demonstrated in the systems APILOT2 and APILOT3, or sent to the workspace by means of a To Workspace block

Enter `type vorxmpl.hlp` at the command-line for on-line help. □

Chapter 8

Implementation of the analytical tools in FDC 1.2

8.1 Introduction

This chapter describes the trimming program ACTRIM and the linearization tool ACLIN. The first program is a real trimming tool which actually contains the aircraft trim algorithm from figure 4.9, while the latter program uses the SIMULINK routine LINMOD to do the actual linearization. For the simulations, FDC uses the standard built-in Runge-Kutta and Adams/Gear methods of SIMULINK, so no additional programs were developed for that purpose. See chapter 4 for the theoretical backgrounds.

8.2 The trimming facility

8.2.1 Program structure of ACTRIM

The program ACTRIM contains the trimming algorithm from section 4.3, which was specially tailored for the search of steady-state flight conditions. The source-code of this program has been stored in the file ACTRIM.M in the subdirectory TOOLS. ACTRIM uses two subroutines: *(i)* ACCONSTR, which contains the flight-path constraints and kinematic relationships, and *(ii)* ACCOST, which evaluates the cost-function for the minimization algorithm. The source-codes of these subroutines are stored in the files ACCONSTR.M and ACCOST.M in the subdirectory TOOLS. ACTRIM uses the MATLAB minimization routine FMINS for the numerical determination of the trimmed flight condition. This routine is based upon the Simplex search method; type `help fmins` for more details about this function. The basic program-structure of ACTRIM and its subroutines ACCONSTR and ACCOST is shown in figures 8.1 to 8.3. This structure closely resembles the diagram from figure 4.9.

The user must first choose a flight condition from the main menu and specify those motion variables that cannot be derived directly from the specified flight condition and which are not adjusted by the trim algorithm itself. ACTRIM then creates two vectors:

1. *ctrim*, which contains the user-specified values of the states, inputs, and some time-derivatives of state variables,
2. *vtrim*, which contains the independent input and state variables that will be adjusted numerically by the trim algorithm

Obviously, the c in *ctrim* denotes *constant* values, while the v in *vtrim* denotes *variables*. The current version of ACTRIM, designed for the 'Beaver' aircraft, defines these vectors as follows:

$$\begin{aligned} ctrim &= \left[V \;\; H \;\; \psi \;\; (\gamma) \;\; \frac{\dot{\psi} V}{g_0} \;\; \dot{\psi} \;\; \dot{\theta} \;\; \dot{\varphi} \;\; \delta_f \;\; n \;\; \varphi \right]^T \\ vtrim &= [\; \alpha \;\; \beta \;\; \delta_e \;\; \delta_a \;\; \delta_r \;\; p_z \,(\text{or } \gamma) \;]^T \end{aligned}$$

The minimization routine FMINS is used to search the values of the independent motion variables gathered in *vtrim*, which minimize the cost function from the subroutine ACCOST. ACCOST itself calls the subroutine ACCONSTR in order to determine the value of the state and input vectors in accordance with the flight-path constraints from section 4.3.3. These values are substituted in the non-linear state equation of the aircraft model:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{8.1}$$

by calling the S-function Beaver in order to find the time-derivatives of the state variables. These are substituted in the cost function J:

$$J = c_1 \dot{V}^2 + c_2 \dot{\alpha}^2 + c_3 \dot{\beta}^2 + c_4 \dot{p}^2 + c_5 \dot{q}^2 + c_6 \dot{r}^2 \tag{8.2}$$

The current implementation of ACCOST uses the following values of the weighting constants c_1 to c_6:

$$\begin{aligned} c_1 &= 1 \\ c_2 = c_3 &= 2 \\ c_4 = c_5 = c_6 &= 5 \end{aligned}$$

See the source-codes ACTRIM.M, ACCONSTR.M, and ACCOST.M in the FDC subdirectory TOOLS for more details. The source-codes contain complete lists of variables and many comments to help you comprehending the structure of these programs.

The numerical iterations are finished if the minimum value of the cost function has been achieved within a certain termination-tolerance of the minimization function FMINS, or if the maximum number of iterations is reached and the user decides to stop the minimization process. Then the subroutine ACCONSTR is called once more in order to extract the input vector $\mathbf{u}$ and state vector $\mathbf{x}$ from *ctrim* and *vtrim*, which at this point contain the values for the trimmed flight condition. Also the S-function Beaver is called once more to find the time-derivative of the state vector for the trimmed condition. The results can be saved to a data file if desired.

8.2.2 Using ACTRIM in practice

ACTRIM can be started by typing `actrim` at the MATLAB command-line. It can also be started by clicking the ACTRIM button-blocks, which have been included in several graphical SIMULINK systems from FDC 1.2. ACTRIM first loads the aircraft model parameters from the file AIRCRAFT.DAT by calling the function LOADER (see section 9.3.1). The user must enter the name of the aircraft model (by default set to Beaver) which will then be initialized. Then a graphical menu with several possible steady-state flight conditions will appear, see figure 8.4. Depending upon the selection you make you will be asked to specify certain initial and/or fixed values of the variables involved in the trimming process.[1] Most often, a steady wings-level flight condition will be evaluated. In that case you must specify the desired airspeed, altitude, heading, flap

[1] Notice that these variables are only valid for the 'Beaver' aircraft; other motion variables will be needed if you want to apply ACTRIM to other types of aircraft. Unfortunately, this still requires re-programming of the source codes, which is not very convenient. Future versions of the trim program should therefore be equipped with more general ways of defining the motion variables.

DEFINE FLIGHT-CONDITION -Steady wings-level flight -Steady turning flight (coordinated or uncoordinated) -Steady pull-up or push-over -Steady roll (in stability-axes or body-axes)
Combine all fixed states, inputs, and time-derivatives of states in the vector *ctrim*
Combine all independent states and inputvariables in the vector *vtrim*, which will be adjusted by the numerical optimization algorithm
Iterate until solution is found or optimization is cancelled by the user
Call *FMINS* for cost function *ACCOST* (see subroutine). *ACCOST* itself calls *ACCONSTR*
Call *ACCONSTR* once more to find resulting trimmed values of **x** and **u**
Call SIMULINK system *Beaver* once more to find time-derivative of **x** for trimmed condition

Figure 8.1: Program structure of ACTRIM (main aircraft trim program)

Calculate q and j from flight-path constraints
Compute *p*, *q*, and *r*, using the kinematic relations from section B.4
Combine these results with the fixed states, which specify the flight-condition, and the independent states, adjusted by the minimization algorithm, into the current state vector **x**
Combine the fixed inputs, specified by the user to define the aircraft-condition, with the independent inputs, adjusted by the minimization algorithm, into the current input vector **u**
Return **x** and **u**

Figure 8.2: Program structure of ACCONSTR (flight-path constraints & kinematic relations)

Call *ACCONSTR* to find constrained values of **x** and **u**
Call SIMULINK model *Beaver* to find current value of the time-derivative of the state vector
Compute cost function *J*
Return *J* to the minimization routine

Figure 8.3: Program structure of ACCOST (contains the cost function)

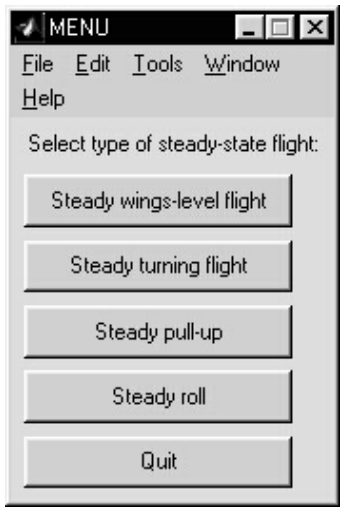


Figure 8.4: Main user-menu of ACTRIM

angle, and engine RPM of the aircraft. You may choose to define the flight-path angle or manifold pressure, which leaves one of these two variables to be numerically solved by the trimming algorithm. If you specify a value of the flight-path angle, ACTRIM will ask you to define an initial estimate for the manifold pressure as well (usually the default values will do just fine). After defining these variables, the numerical iteration process will be started. If the solution has converged enough according to the trimming options specified within the source code, the results will be displayed on the screen. If the maximum number of iterations is exceeded without a solution having been found, a warning message will appear on the screen and you must choose whether to perform more iterations or accept the best solution found thus far. If the solution hasn't converged after a few more attempts it is probably not possible to find a steady solution within the tolerance specified in the source code of the trim program.

If you choose to save the trimmed flight condition to a file, you will be asked to specify the destination directory (by default the FDC subdirectory DATA) and the filename. The file will get the extension .TRI to specify that it represents a trimmed flight condition. This will end the program. An example of this trim process will be given in section 10.5.

ACTRIM stores the results in the following variables:[1]

- *xinco* = $\mathbf{x}(0)$ is the steady-state value of the state vector from the non-linear aircraft model; $\mathbf{x} \equiv [\, V \;\; \alpha \;\; \beta \;\; p \;\; q \;\; r \;\; \psi \;\; \theta \;\; \varphi \;\; x_e \;\; y_e \;\; H \,]^T$,
- *xdot0* = $\dot{\mathbf{x}}(0)$ is the time-derivative of the state vector, valid for the current flight condition (if a trimmed flight condition was found most elements of *xdot0* will be almost equal to zero, as explained in section 4.3),
- *uaero0* = $\mathbf{u}_{aero}(0)$ is the steady-state value of the input vector to the aerodynamic forces & moment functions of the aircraft model; $\mathbf{u}_{aero} \equiv [\, \delta_e \;\; \delta_a \;\; \delta_r \;\; \delta_f \,]^T$,
- *uprop0* = $\mathbf{u}_{prop}(0)$ is the steady-state value of the input vector to the propulsion functions of the aircraft model; $\mathbf{u}_{prop} \equiv [\, n \;\; p_z \,]^T$,
- *trimdef* is a text-matrix which contains the user-specified variables that define the flight condition and aircraft configuration, the name of the SIMULINK system of the aircraft model, the definitions of the state and input vectors used by the aircraft model, a short explanation about the other variables, and the date and time when ACTRIM was used to find this steady-state condition.

Although ACTRIM works perfectly well for finding initial flight conditions for non-linear simulations, it is not really suited for searching large numbers of steady-state flight conditions, e.g. in order to find the steady-state elevator-deflection as a function of airspeed. In such cases it may be easier to write a customized trim routine, which may contain pieces of software code from the source code ACTRIM.M. For example, the program TRIMDEMO contains only the trimming commands of ACTRIM, while using its own routines for defining the flight condition and displaying results on the screen. The source code TRIMDEMO.M has been stored in the FDC subdirectory EXAMPLES. See section 10.5 for more details about that example program. In order to enhance the flexibility of the trimming program, it is planned to divide the separate functions from ACTRIM into separate generalized subroutines for future versions of the FDC toolbox. However, since the source-codes themselves contain many explanatory comment-lines and a full list of variables, it is not very complicated to adapt the programs if required. Remember not to violate the license agreement if you wish to distribute such adapted FDC programs...

8.3 The linearization facility

8.3.1 Program structure of ACLIN

The program ACLIN has been designed to extract linearized aircraft models from the non-linear SIMULINK system Beaver (or similar models) in a user-specified operating point.[2] Its main goal is to simplify the model definitions for the user; the actual linearization process is left to the SIMULINK function LINMOD. Figure 8.5 shows the general structure of this program. ACLIN first asks which aircraft model to be used (by default Beaver). Next, the operating point needs to be defined – either by loading it from file, manually defining it in the workspace, using an existing operating point from the workspace, or running ACTRIM to obtain a new steady-state trimmed-flight condition as operating point for the linearization. If the aircraft model parameters are not yet present in the workspace, they are loaded by means of the routine LOADER (see

[1] Due to the fact that these vectors are often used as initial values for non-linear simulations, they use the extensions *0* or *inco* in their variable names.

[2] ACLIN currently works only for aircraft models which use the same definitions of input and output vectors as the system Beaver. For other types of aircraft models, the source-codes must be adapted. This again is not quite satisfactory for future enhancements of the FDC toolbox, so a more modular approach will be needed in future releases.

Define operating point:	- Load operating point from file, or - Manually define operating point, or - Use operating point defined in workspace, or - Run *ACTRIM* to find operating point
Load aircraft model parameters	
Call *LINMOD* to obtain linearized aircraft model	
Select state variables for linear aircraft model (use all twelve states or select a subset)	
Select control inputs for linear aircraft model (use all six control inputs or select a subset)	
Either add or don't add wind & turbulence inputs	
Present results and save them to a file if required	

Figure 8.5: Program-structure of ACLIN

section 9.3.1). The linearization routine LINMOD then determines the full 12^{th}-order system matrices of the linearized aircraft model, which subsequently can be simpified by neglecting the influence of certain state and/or input variables. If desired, the resulting matrices can be saved to a datafile, which will get the extension .LIN.

8.3.2 Using ACLIN in practice

ACLIN can be started by typing `aclin` at the command-line. It will also be started if you double-click the 'button' blocks of ACLIN, which are contained in several graphical SIMULINK systems from FDC 1.2. ACLIN first asks the user to enter the name of the aircraft model (by default Beaver). Next, the method of defining an operating point must be selected by clicking one of the buttons from the menu shown in figure 8.6. The operating point can be defined in the following ways:

- It is possible to load an operating point from a file. Usually this will be a trimmed flight condition, obtained with ACTRIM. If you choose this option, the program INCOLOAD will be started (see section 9.3.2).
- An operating point can be defined manually. After selecting this option the user will be asked to enter values for all state variables of the aircraft model and all control inputs, i.e. V, α, β, p, q, r, ψ, θ, φ, x_e, y_e, H, δ_e, δ_a, δ_r, δ_f, n, and p_z.
- If an operating point already exists in the MATLAB workspace, that is: if the variables *xinco*, *uaero0*, and *uprop0* are present in the workspace, it is possible to that operating point for the linearization process.
- ACTRIM can be called in order to find an appropriate steady-state operating point for the linearization. See section 8.2.2.

After defining the operating point, ACLIN will load the model parameters from file, using the routine LOADER (see section 9.3.1), unless these parameters are already present in the workspace. Next, the linearization routine LINMOD will be called to find the linear aircraft model matrices *Aac*, *Bac*, *Cac*, and *Dac*. Since the 'Beaver' model uses 12 state variables and 12 inputs (including wind & turbulence), these matrices all have the dimensions 12×12. It is possible to extract simplified submatrices from *Aac* and *Bac* by specifying a vector with the element numbers of the

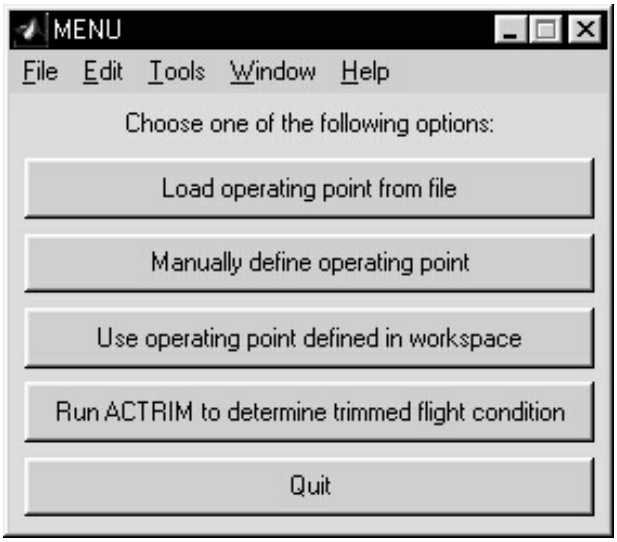


Figure 8.6: User-menu of ACLIN, used to determine operating points

required state variables and control inputs. This option can also be used to change the *order* of the state and/or input variables: just type all twelve element numbers in the required changed order. Moreover, it is possible to eliminate the wind & turbulence inputs from the model. If the user thus specifies a simplified model, two additional matrices will be constructed in the MATLAB workspace: *Aac_s* and *Bac_s*. The extension *s* in these variable names denotes a simplified model. Of course, the exact definitions of these matrices depend upon the user-specified element numbers from the state and input vectors. Remember that any simplification of the model matrices yields less accurate results than the complete 12^{th}-order model. One possible application of this simplification is to de-couple the symmetrical and asymmetrical equations of motion in the linearized model. In order to help you remind the definitions of all results, including the simplified model matrices *Aac_s* and *Bac_s*, a text-matrix *lindef* is created. This matrix also contains the definition of the operating point and the date and time when the results were determined by ACLIN.

Finally ACLIN will ask you whether you want to save the results to a file. If you answer *Y*, you will be asked to enter a directory name (by default, the FDC subdirectory DATA will be used) and a filename. The file will get the extension .LIN, specifying a linearized model. In addition to the model matrices and the text-matrix *lindef*, it is possible to include the operating point to this file. This will be stored in the variables *xinco*, *uaero0*, *uprop0*, and, if present in the workspace, *trimdef*. See the definitions in section 8.2.2, page 137. An example of this linearization process is given in section 10.4.

Chapter 9

Other utilities for the FDC toolbox

9.1 The FDC initialization routine FDCINIT

The routine FDCINIT initializes the FDC package by extending the MATLAB search path with the FDC directories. The first time you run FDCINIT by typing `fdcinit` or `fdc` at the command-line, you will get some welcome messages before being asked to specify the search path extension (see also section 1.5). The default FDC directory structure is:

```
C:\FDC12  \AIRCRAFT
          \APILOT
          \DATA
          \DOC
          \EXAMPLES
          \HELP
          \NAVIGATE
          \TOOLS
          \WIND
```

FDCINIT makes it easy to change the path-names, extend the search path with new directories, or delete directories from the FDC search path if they are not needed anymore. Just answer *N* to the question whether the FDC path is correct, and select the appropriate menu-items to change the FDC path according to your wishes. The next time you start FDCINIT the new FDC path will be displayed as default. It is possible to suppress the question to check the FDC path for future sessions. If you still want to change the FDC path after suppressing this check once, you must delete the file FDCINIT.INI from the FDC root-directory before running FDCINIT again. You will then be welcomed again like a first-time user. See section 1.5 for more details.

9.2 The aircraft model parameter definition macro MODBUILD

Although the aerodynamic, propulsive, and geometrical properties of different types of aircraft can differ strongly, the structure of the aircraft model is quite generic. In theory it is, to some extent, also possible to specify standard structures for the aircraft-dependent submodels, but this requires clear, well defined modeling standards. In practice this is often not the case, especially when using model data from different sources. However, for the practical use of the non-linear aircraft model this does not matter too much since all aircraft-dependent elements from this model can be combined in separate subsystems, which can be treated as *black-boxes*. In this way it does not matter whether an aerodynamic model is being described by means of non-linear polynomial equations with constant coefficients, or by equations with non-constant coefficients

which are extracted from tables. It is not necessary to use standardized data formats for the model *parameters* either. In the MATLAB/SIMULINK environment it is easy to load system parameters for the SIMULINK systems into the MATLAB workspace, regardless of their exact definitions.

Thus, in general, there are no restrictions to the definitions of model parameters. This provides maximum flexibility for the implementation of other aircraft models within the framework of the system Beaver. However, there is one exception to this rule: the main geometric properties of the aircraft and its mass-distribution data must be defined in two *standardized* data-matrices called *GM1* and *GM2*, respectively. Currently it is not possible to alter the definitions of these matrices without changing the contents of the subsystem Aircraft Equations of Motion which forms the core part of the non-linear aircraft model. This is mainly due to the fact that the current aircraft model considers the aircraft geometry and mass-distribution to be constant during the motions of interest. Future versions of the toolbox should feature on-line computation of these properties, which will allow us to simulate the motions of vehicles with non-constant geometry, e.g. variable wing-sweep, or significant sudden changes in mass and/or mass-distribution, e.g. dropping loads from flying aircraft.

For the current implementation of the 'Beaver' model, the model parameters are defined in four data-matrices, including the earlier mentioned matrices for storing geometrical data and mass distribution (*GM1* and *GM2*). The matrices *AM* and *EM* contain the coefficients for the aerodynamic model and engine forces and moments model of the 'Beaver', using data from ref.[26]. Appendix D contains the exact definitions of these parameter matrices. Before starting a simulation involving the system Beaver the parameter matrices need to be present in the MATLAB workspace. For this reason, they have been gathered in the datafile AIRCRAFT.DAT (stored in the subdirectory DATA) which can be loaded into the workspace by means of the MATLAB macro LOADER (see section 9.3.1). The datafile itself was generated by the MATLAB macro MODBUILD. This macro defines the values of all model parameters, stores the results in the data matrices *AM*, *EM*, *GM1*, and *GM2*, and saves these matrices to the datafile AIRCRAFT.DAT. In order to obtain the inertial parameters, MODBUILD contains the equations from tables B.1 and B.2 from appendix B.

For the FDC users it is normally not necessary to run MODBUILD, since all results are already available in the file AIRCRAFT.DAT within the FDC subdirectory DATA. If the user wants to change one or more model parameters, for instance because an improved version of the aerodynamic or engine models has been obtained, the quickest way to update the datamatrices is to edit the source-code MODBUILD.M accordingly.[1] Since MODBUILD also computes the inertial parameters, changes in geometrical properties or mass-distribution also have to be taken into account by editing MODBUILD.M. This may seem rather complicated, but due to the clear structure of MODBUILD.M it is not *that* difficult. If you want to implement a model of another aircraft within the framework of the system Beaver it is recommended to use at least the part of MODBUILD where the matrices *GM1* and *GM2* are determined, in order to get the appropriate matrix definitions. The definitions of the aerodynamic and engine model parameter matrices *AM* and *EM* may be useful as a guideline for implementations of other aircraft models, but, as said before, you are not restricted to the use of those parameter matrices and you are free to apply your own data structure instead. Future versions of the FDC toolbox will probably be equipped with more flexible parameter-definition tools. If you accidentally destroy the file AIRCRAFT.DAT, it can be retrieved by running MODBUILD again (type `modbuild` at the command-line).

[1] The file MODBUILD.M has been stored in the subdirectory AIRCRAFT, because it directly relates to the non-linear aircraft model itself.

```
Loading model parameters from AIRCRAFT.dat
==========================================

Specify directory (default: c:\fdc12\data):  g:\mytools\fdc\data

Datamatrices AM, EM, GM1, and GM2 loaded.

Ready.

>>
```

Figure 9.1: The MATLAB command-window when running LOADER

9.3 Routines to load data from files

9.3.1 The model-parameter load routine LOADER

The routine LOADER is used for loading the parameter matrices for the aircraft model from the datafile AIRCRAFT.DAT. In FDC 1.2, this datafile contains the model parameters for the system Beaver, but it is planned to use a similar data structure for other aircraft models in future versions of the toolbox. In addition to loading the model parameters, LOADER also defines the vector *xfix*, which is used in the aircraft model to artificially fix elements of the state vector to their initial values. By default, no state variables are fixed, unless you have already changed the vector *xfix* yourself; see section 9.5 and the description of the block xfix in chapter 5 for more information.

If you type `loader` at the MATLAB command-line, or double-click a button block LOADER in a graphical SIMULINK system from FDC 1.2, the program will ask you to specify the directory in which it will search for the file AIRCRAFT.DAT. Normally, the default directory (being the FDC subdirectory DATA) will be correct, so you only have to press Enter. If you have saved a customized version of AIRCRAFT.DAT in another directory or if the path-definition of FDCINIT is not correct, you must enter the correct directory name. If LOADER cannot find AIRCRAFT.DAT in the specified directory, it will asks whether to run MODBUILD to create this file. Else, the file will be loaded into the MATLAB workspace. Figure 9.1 shows what the command-window will look like. In this example, the directory G:\MYTOOLS\FDC\DATA is specified in stead of the suggested default directory C:\FDC12\DATA. If you activate LOADER by means of a button-block in a graphical SIMULINK system, you must activate the MATLAB command-window by yourself, which may not be obvious if the command-window is hidden behind other windows. Therefore it is recommended to keep at least a part of the command-window in sight. This somewhat inconvenient user-interface is an inheritage of the original version of the FDC toolbox which ran under MATLAB 3.5 for WINDOWS where this problem did not arise. Expect a new version of LOADER, featuring a graphical user-menu, to be released soon.

9.3.2 The load routine INCOLOAD

The routine INCOLOAD is used for loading trimmed flight conditions, system matrices of linearized aircraft models, or other datafiles into the MATLAB workspace. It can be started by typing `incoload` at the command-line, or double-clicking an INCOLOAD button within a graphical SIMULINK system from FDC 1.2, after which a user menu will be displayed, see figure 9.2.

Figure 9.2: The main menu from INCOLOAD

```
FDC 1.2 - INCOLOAD
------------------
Load data for simulations of FDC systems.


Specify directory (default:  c:\fdc12\data):  g:\mytools\fdc\data

Enter filename without extension (8 characters max.):

> cr4520

Enter extension (3 characters), default = tri:

>

Loading data from file

g:\mytools\fdc\data\cr4520.tri


Your variables are:

uprop0              xdot0
trimdef             xinco
uaero0

Ready.

>>
```

Figure 9.3: Specification of a filename for trimmed-flight condition

Here the type of data to be loaded into the workspace must be specified. After clicking the appropriate button you will be asked to specify the name of the directory (in the MATLAB command-window!). By default this is set to the FDC sub-directory DATA. Next you must enter the filename without extension (INCOLOAD does not support filenames longer than 8 characters) and the file extension (not longer than 3 characters). By default the file extension is set to .TRI for trimmed flight conditions, .LIN for linearized models, and .MAT for other datafiles, depending upon the choice made in the options menu. INCOLOAD will then try to load the datafile into the workspace. If the file cannot be found, a warning message will be displayed; otherwise the file will be loaded and all variables present in the workspace after loading the file will be displayed in the command-window. Figure 9.3 shows what the command-window will look like if you choose to load a trimmed flight condition from file. In this example, the datafile CR4520.TRI is retrieved from the directory G:\MYTOOLS\FDC\DATA, which differs from the default directory C:\FDC12\DATA.

Note: expect an improved version of INCOLOAD which will feature a 'real' graphical user-interface that bypasses the command-window to be released soon.

9.4 Programs for post-processing simulation results

9.4.1 The routine RESULTS

During simulations, all results are sent to the MATLAB workspace by means of To Workspace blocks. For the non-linear aircraft model currently time-trajectories of 89 output variables, 12 input variables, and the time-points themselves are stored in the MATLAB workspace in the output variables *Out*, *In*, and *time*, respectively.[1] In order to facilitate the processing of these results, a MATLAB macro RESULTS has been created. This macro extracts time-trajectories of individual variables from these matrices in easily recognizable variables such as *alpha*, *deltae*, *qdyn*, *Tt*, etc. After running RESULTS it becomes quite easy to plot the time-trajectories of output signals by simply typing:

```
plot(time,V);
plot(time,alpha);
plot(time,deltae);
```

and so forth. See appendix E for a complete list of software acronyms for the different symbols used in this report. Future versions of the FDC package should include more advanced options for selecting which output signals are to be send to the workspace and for automating the generation of graphical presentations of the simulation results.

9.4.2 The routine RESPLOT

In order to get a quick overview of the last simulation results, the MATLAB macro RESPLOT has been created. This macro should be applied *after* running RESULTS. It plots the most important output variables in a graphics window. First, the true airspeed V, angle of attack α, sideslip angle β, and altitude H will be displayed. Pressing a key then reveals the angular velocities p, q, and r. Next, the Euler angles ψ, θ, and φ are displayed. The final screen contains the aerodynamic input signals δ_e, δ_a, and δ_r, as well as the wind velocity components u_w, v_w, and w_w. If you want to plot other output variables or change the order of appearance, you will

[1] Note: under some circumstances SIMULINK does not properly send the simulation results to the workspace. To solve this problem, a routine RECOVER has been created; see section 9.4.3.

Figure 9.4: Main menu of FIXSTATE

have to edit the file RESPLOT.M in the subdirectory TOOLS. Of course it is also possible to write your own plotting macros or enhance the capabilities of the RESULTS utility.
It is obvious that the current version of RESPLOT lacks the desired flexibility and interactiveness of a sophisticated plotting routine. Its sole purpose at this moment is to facilitate the first rough analysis of simulation results, and to help novice users in visualizing simulation results from the workspace. Future versions of the toolbox should contain more sophisticated plotting utilities which allow users to customize the graphs, define the variables to plot, and save or print the results in an interactive way.

9.4.3 The routine RECOVER

During simulations of the non-linear aircraft model the results are sent to the MATLAB workspace through To Workspace blocks. However, this does not always function properly due to a bug in SIMULINK, which has been noticed for SIMULINK for MS WINDOWS up to version 1.2c. Luckily it is often still possible to retrieve the results if they are not present in the workspace after running a simulation by simply calling the appropriate SIMULINK model once more. The routine RECOVER simplifies this process somewhat. If you can't find the results after a simulation, type `recover('sysname')`, where *sysname* is the name of the SIMULINK model, e.g. type `recover('beaver')` for the system Beaver. The routine RESULTS automatically calls RECOVER if it cannot find the matrices *In* and *Out* or the time-vector *time*.

9.5 The routine FIXSTATE to artificially fix state variables

The non-linear aircraft model contains a gain-block xfix, which is used to artificially fix state variables to their initial values. This may for instance be useful if you want to neglect longitudinal-lateral cross-coupling effects, or if you want to fix the airspeed to its initial value to simulate an 'ideal' autothrottle system. In order to fix certain states, the block xfix multiplies the time-derivative of the state vector with the vector *xfix*. This multiplication vector has twelve elements that correspond with the twelve state variables. These elements are either equal to one, in which case the actual time-derivative of the corresponding state variable is taken into account, or zero, in which case the time-derivative of the corresponding state variable is artificially set to zero. In

the latter case, the state variable will remain equal to its initial value. The block xfix has been described in detail in chapter 5.

In order to facilitate the definition of the multiplication vector *xfix*, the routine FIXSTATE has been designed. Figure 9.4 shows the main menu of this routine. It has the following options:

- Fix asymmetrical state variable. This yields a simplified model which takes into account the symmetrical equations of motion only. If you select this option, FIXSTATE will ask whether the variable y_e needs to be fixed as well. It will then try to re-initialize the aircraft model (this is only possible if the initial condition has already been defined in the workspace by means of the trim routine ACTRIM, the load routine INCOLOAD, or manual definition of the variable *xinco*) and display the fixed state variables, being: β, p, r, ψ, φ, and if desired y_e.
- Fix symmetrical state variables. This simplifies the aircraft model to the asymmetrical equations of motion only. If you select this option, FIXSTATE will ask whether the variables x_e and H need to be fixed as well and it will try to re-initialize the aircraft model. It will then display the fixed state variables, being: V, α, q, θ, and if desired also x_e and H.
- Fix arbitrary state variables. If you select this option, FIXSTATE will ask you to specify a vector with the element numbers of the state variables you want to fix. With the state vector being $\mathbf{x} = [\ V\ \ \alpha\ \ \beta\ \ p\ \ q\ \ r\ \ \psi\ \ \theta\ \ \varphi\ \ x_e\ \ y_e\ \ H\ ]^T$, you can for instance fix θ and x_e by specifying the vector `[8 10]`.
- Don not fix any state variables. Use this option to *reset* the original configuration in which all states can vary freely.

Although it is necessary to have the variable *xfix* defined in the MATLAB workspace, it is not necessary to run FIXSTATE if you don't want to fix any states. The default value of *xfix* can be defined manually by simply typing: `xfix = 1` (which is equivalent to `xfix = ones(1,12)`), but if you use LOADER to retrieve the model parameters from file, the default value of *xfix* will be set automatically if the variable is not yet present in the workspace. Use FIXSTATE only if you actually want to fix state variables or reset the model back to its original configuration.

9.6 The routine SYSTPROP to compute linear system-properties

In order to facilitate the analysis of a linear system, the utility SYSTPROP (which stands for 'system properties') has been included to the FDC 1.2 package. This utility can be applied to *any* linear system in state space or transfer function format, although its main goal in the FDC toolbox is to facilitate the analysis of linearized aircraft models obtained by the linearization routine ACLIN (see section 8.3). SYSTPROP computes the following properties of a linear system:

- time constant τ,
- natural frequency of the undamped system ω_0,
- eigenfrequency of the system ω_n,
- period P,
- damping factor ζ,
- percentage overshoot $P.O.$,
- peak-time T_{peak},
- settling time T_{set},
- halve-time $T_{1/2}$.

The results are displayed on the screen, using the number-formatting routine NUM2STR2, which is a customized version of the MATLAB routine NUM2STR. They are also stored in the file SYSTPROP.DAT. Type `systprop` or `help systprop` at the command-line to see how it works (type `help num2str2` to find out more about the number-formatting routine NUM2STR2). Refer to ref.[28] for the theoretical backgrounds.

Note: SYSTPROP *requires the Control System Toolbox to function properly!*

9.7 The SIMULINK library FDCTOOLS

FDC 1.2 contains a library FDCTOOLS with some useful new blocks that can be applied in graphical SIMULINK systems. Some of these blocks have been applied in other graphical systems from FDC 1.2, others have been implemented for general use. The library itself is contained in the file FDCTOOLS.M in the subdirectory TOOLS. It can be opened by typing `fdctools` at the command-line. If you want to copy the tools library separately, be sure to include the files NSWITCH.M and SOFTLIM.M too (these files belong to the blocks n-switch and Soft-limiter from the tools library).

9.7.1 Input blocks from FDCTOOLS

The library FDCTOOLS contains two new input blocks:

- Block fcn generates a block-shaped input signal. The user can specify the initial value of the signal, the amplitude of the block, and the duration of the block-input.
- Doublet generates a doublet signal. The user can specify the initial value of the signal, the length of the two time-intervals for the upper and lower block-shaped part of the signal, the values of the signal during those two time-intervals, and the starting time of the doublet.

These blocks were created with the *Mask* utility of SIMULINK. Unmask these blocks to see how they work.

9.7.2 Gain scheduling blocks from FDCTOOLS

The block Scheduled Gain makes it possible to implement a gain-scheduling system. The second input of this block is multiplied by a signal that depends upon the first input. The user must specify the gain-scheduling function in the internal Fcn block of Scheduled Gain. If a *vector* input is used as gain-scheduling signal, it is possible to change the gain value as a function of all vector elements. For instance, it would be possible to change gains in an autopilot system as a function of both the airspeed and the altitude. In the systems APILOT1 to APILOT3, the use of the gain-scheduling blocks has been demonstrated for the 'Beaver' autopilot. If you apply many Scheduled Gain blocks, it is recommended to delete the internal title and More Info blocks, leaving only the actual gain-scheduling function. This will considerably reduce the size of the M-file that contains your graphical system. Note: if you *flip* or *rotate* Scheduled Gain blocks, the icon of this block will not automatically change its direction. Although this may look somewhat odd, it does not affect the results.

9.7.3 Switches from FDCTOOLS

SIMULINK 1.2C contains a switch block that selects which one of two input signals is passed through, depending upon a switch-control signal. The library FDCTOOLS contains some new switch blocks with enhanced functionality:

- The block On/off switch, which is based upon the standard switch block from SIMULINK. It has two inputs: a switch-control signal, and the main signal which is either passed through or blocked. If the first input is equal to 1, the second input is passed through; otherwise the output of the On/off switch is set to zero.
- The block 6-switch, which is a masked subsystem that makes it possible to select one out of six main input signals by setting an additional seventh switch-control signal. The first input of the block 6-switch is the switch-control signal, which must be equal to a number i, with $i \in \{1, 2, 3, 4, 5, 6\}$. The output signal from the 6-switch is equal to the $i + 1^{th}$ input. If $i \notin \{1, 2, 3, 4, 5, 6\}$, the output signal will be equal to zero. The block 6-switch has been created with the MATLAB macro NSWITCH. If you want to make an n-switch (where n is an arbitrary integer value which defines the number of input signals to the desired n-switch block) you can run NSWITCH from the command-line by typing `nswitch(`n`)`. If you type `nswitch` without an input argument, NSWITCH itself will prompt you to enter the value of the integer n. Although there is no upper-limit for the number of input signals, your screen size will impose a practical limit. The time needed for building the graphical n-switch block increases exponentially with n. Try running NSWITCH for some different values of n to see for yourself! Type `help nswitch` at the command-line for on-line help.

9.7.4 Discrete signal blocks from FDCTOOLS

The block MA-filter contains a Moving Average filter. The output from this block is equal to the average of a number of samples taken from the input signal. This number, along with the sample time, can be specified freely by the user: double-click the MA-filter block to enter these values.. A second variant of the MA-filter block has been included for users who have the CONTROL SYSTEM TOOLBOX from MATLAB. The only difference between the two versions is that the latter creates a nice graphical icon, using the MATLAB function DSTEP from the CONTROL SYSTEM TOOLBOX. If you double-click the button-block under which this second version of the MA-filter has been hidden, an error message will appear if the function DSTEP can't be found.

9.7.5 Non-linear function blocks from FDCTOOLS

The block Soft-limiter contains a limiter with a smooth transition to the limit values. The user can specify the *range* of the limiter and the part of this range where the input signal is passed through directly (linear throughput of the input signal). Values of the input signal that exceed this linear range will be reduced to a value within the limiter-range. The signal reduction is larger for input signals which further exceed the linear range, i.e. the output signal *asymptotically* reaches the maximum or minimum allowed value, as defined by the limiter range setting. The block Soft-limiter has been implemented as a graphical S-function block which calls the MATLAB subroutine SOFTLIM. It was not possible to use a normal MATLAB subroutine-block, because such blocks do not allow parameters to be sent from the graphical SIMULINK block to the subroutine. For this reason the subroutine has been implemented as an *S-function* of which only the *output* relation is used. See the source-code of SOFTLIM.M in the subdirectory TOOLS for more details.

Chapter 10

Performing open-loop analysis with FDC 1.2

10.1 Introduction

This chapter explains how to generate open-loop responses with the 'Beaver' simulation model from chapter 5. It is useful to read this section to get acquainted with the practical use of the FDC models and tools. Once you master this part of the report it will be much easier to understand the description of the autopilot case-study from chapters 11 and 12. FDC 1.2 contains three open-loop simulation models:

1. OLOOP1 is used to obtain non-linear aircraft responses to control inputs,
2. OLOOP2 is used to obtain non-linear aircraft responses to atmospheric turbulence,
3. OLOOP3 is used to obtain linear aircraft responses to control inputs.

In addition, there are three corresponding 'tutorial' systems which explain the functions of all elements within these open-loop systems. These tutorial system are called OLOOP1T, OLOOP2T, and OLOOP3T, respectively. These open-loop systems will be treated in sections 10.2 to 10.4. Section 10.5 describes a MATLAB program which uses the aircraft trim algorithm to determine the trimmed-flight elevator deflection curve of the aircraft.

10.2 Non-linear responses to deterministic inputs – OLOOP1

10.2.1 Structure of the system OLOOP1

The SIMULINK system OLOOP1 can be used to obtain open-loop simulations of the 'Beaver'. After initialization of the toolbox (section 1.5), it can be opened by typing `oloop1` at the MATLAB command-line or by double-clicking the OLOOP1 button in the library FDCLIB. There is also a 'tutorial' system OLOOP1T which explains the meaning of all blocks from OLOOP1. This system can be opened by typing `oloop1t` at the command-line or double-clicking the OLOOP1T button in FDCLIB. A picture of OLOOP1 is shown in figure 10.1.

The core of this simulation model is an S-function block which calls the system Beaver (see chapter 5 for a description of this simulation model). As explained in section 5.1, the connections between Beaver and other subsystems are made by means of Inport and Outport blocks within the first level of Beaver. When calling Beaver from within another simulation model such as OLOOP1, it is necessary to apply an input vector with as many elements as there are Inport blocks in the first level of Beaver. Obviously, the output vector will have has as many elements

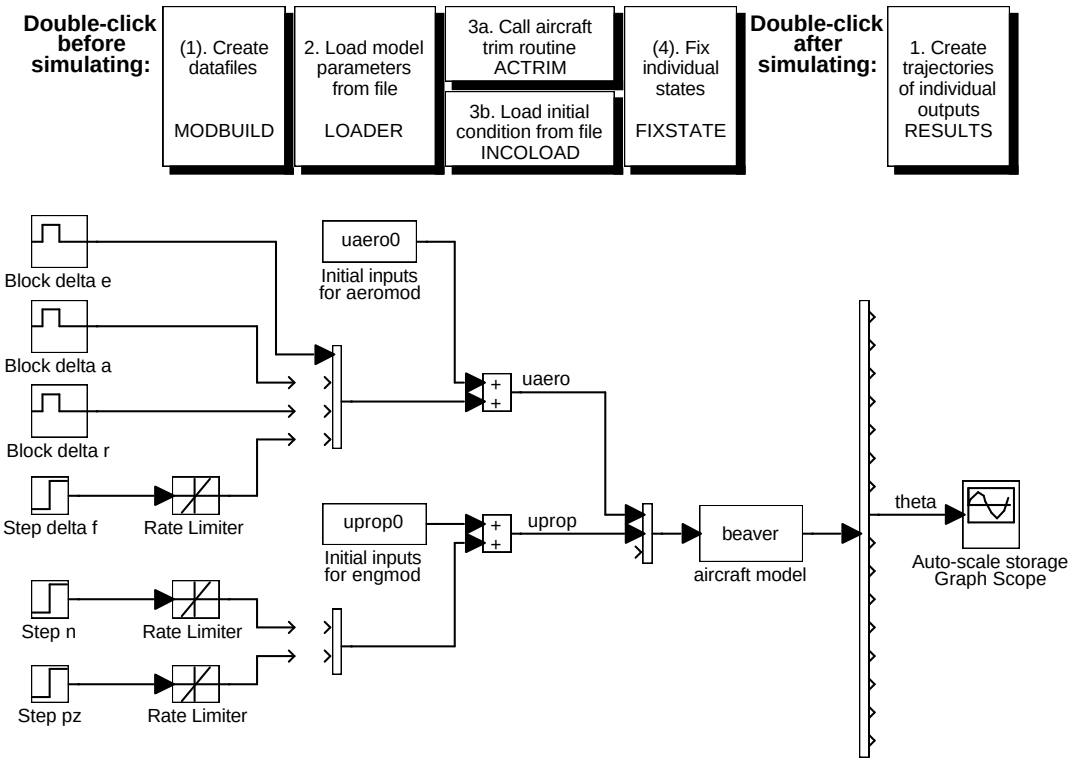


Figure 10.1: Block-diagram of the open-loop system OLOOP1

as there are Outports in the first level of Beaver. As explained in section 5.1, these so-called *S-function outputs* cover only a subset of the total number of outputs from the aircraft model, due to the fact that SIMULINK does not allow the use of vector signals by the Inport and Outport blocks in the first level of a graphical model. However, *all* outputs are sent to the workspace during simulations, from where they can be accessed for further analysis. See also the definitions of the output matrices in section E.2 of appendix E. For the system 'Beaver' there are sixteen S-function outputs, which are gathered in one output vector that leaves the S-function block within OLOOP1:

$$\mathbf{y} = [\underbrace{V \;\; \alpha \;\; \beta \;\; p \;\; q \;\; r \;\; \psi \;\; \theta \;\; \varphi \;\; x_e \;\; y_e \;\; H}_{\mathbf{x}} \;\; \dot{H} \;\; \underbrace{\tfrac{pb}{2V} \;\; \tfrac{q\bar{c}}{V} \;\; \tfrac{rb}{2V}}_{\mathbf{y}_{dl}}]^T \tag{10.1}$$

The non-dimensional rotational velocities $\frac{pb}{2V}$, $\frac{q\bar{c}}{V}$, and $\frac{rb}{2V}$ were needed for the autopilot simulation models, see section 12.3. For most purposes, this selection of output signals is quite sufficient – the number of outputs can be increased by appropriate editing of the first level of Beaver.[1] There are twelve input signals, which enter the S-function block in OLOOP1 by means of one input vector:

$$\mathbf{u} = [\underbrace{\delta_e \;\; \delta_a \;\; \delta_r \;\; \delta_f}_{\mathbf{u}_{aero}} \;\; \underbrace{n \;\; p_z}_{\mathbf{u}_{prop}} \;\; \underbrace{u_w \;\; v_w \;\; w_w \;\; \dot{u}_w \;\; \dot{v}_w \;\; \dot{w}_w}_{\mathbf{u}_{wind}}]^T \tag{10.2}$$

The first six elements of this vector are the control inputs, while the latter six elements represent atmospheric disturbances. The number of S-function inputs is equal to the total number of input

[1] Remember however, that any change in the Inport and Outport definitions from the system Beaver must be taken into account in all systems and MATLAB programs which contain calls to Beaver. If you plan to do this it is therefore recommended to save the modified system under a new filename.

signals to the system Beaver. The definitions of the input and output vectors are also given in section E.2 of appendix E, and it can be retrieved from the command-line by typing `type inputs.hlp` or `type outputs.hlp`.

On the output side of the S-function block, a Demux block with 16 outputs has been connected. This enables us to connect Scope blocks for monitoring individual output trajectories during the simulations. Detailed analysis of the results is also possible after finishing a simulation, because all results are sent to the workspace by means of To Workspace blocks within the system Beaver. On the input side of the S-function block, a Mux block is used to combine the three input vectors $\mathbf{u}_{aero}$, $\mathbf{u}_{prop}$, and $\mathbf{u}_{wind}$ into the vector $\mathbf{u}$ from equation (10.2). OLOOP1 limits itself to the control inputs, i.e. $\mathbf{u}_{aero}$ and $\mathbf{u}_{prop}$, while OLOOP2 has been designed for open-loop simulations of aircraft responses in atmospheric turbulence. For the system OLOOP1 we can therefore leave the $\mathbf{u}_{wind}$ line to the Mux block unconnected. Two additional Mux blocks are included to construct aerodynamic and propulsive input vectors from their scalar elements.

It is important to point out the fundamental difference between linear and non-linear aircraft models. Linear models use small-perturbation signals which describe the responses of the aircraft in terms of *deviations* from the nominal values of its motion variables. The input signals to these models represent deviations from the nominal values of the control inputs. On the other hand, non-linear models use the true values of all signals. For instance, if we want to analyze the response of the aircraft to a block-shaped elevator input we would supply only the *change* in elevator deflection in case of a linear model, while the *total* elevator deflection, i.e. the initial value plus the block-shaped change in elevator deflection, must be supplied in case of a non-linear model. Therefore it is necessary to add the initial values of the input vectors ($\mathbf{u}_{aero}(0)$ and $\mathbf{u}_{prop}(0)$) to the test-signals before they are entered into the non-linear aircraft model. This explains the function of the two Sum blocks in OLOOP1. These Sum blocks add the initial values of the aerodynamic and propulsive input vectors to the block-shaped test-inputs. The initial values of the input vectors are obtained from the MATLAB workspace via the variables *uaero0* and *uprop0*, which enter the graphical system through two Constant blocks. These initial values can be specified manually, or obtained by means of the aircraft trim program ACTRIM, as will be shown in the next section.

In the example system OLOOP1, the following test-signals are supplied by default:

- a block-shaped change in elevator deflection: $\Delta\delta_e = 3^\circ$ during 2 seconds,
- a block-shaped change in aileron deflection: $\Delta\delta_a = 3^\circ$ during 2 seconds,
- a block-shaped change in rudder deflection: $\Delta\delta_r = 3^\circ$ during 2 seconds,
- a ramp-shaped change in flap setting: $\Delta\delta_f = 3^\circ$ within 3 seconds,
- a ramp-shaped change in engine RPM: $\Delta n = 200$ $[RPM]$ within 4 seconds,
- a ramp-shaped change in manifold pressure: $\Delta p_z = 2$ $[''Hg]$ within 2 seconds.

(These signals originally were chosen to validate the simulation results by comparing them with existing data; they trigger all important characteristic motions of the aircraft.) The block-inputs are extracted from the FDC library FDCTOOLS, which can be opened by typing `fdctools` at the command-line. See section 9.7 for a description of this block-library. The ramp-shaped inputs are created by passing a step-input through a Rate Limiter (both are standard SIMULINK blocks). A different shape of the test signals can be obtained by changing the block-parameters after double-clicking the input blocks. Of course, it is also possible to replace these blocks by other types of input shapes from the standard SIMULINK libraries or from FDCTOOLS. In figure 10.1 only the elevator test signal has actually been connected to the aircraft model. The other test

inputs can easily be connected by drawing the corresponding signal lines.

Before starting a simulation of OLOOP1 it may be useful to take a closer look at the 'tutorial' system OLOOP1T. This system provides a step-by-step explanation of the construction of OLOOP1 itself, which will probably help you to comprehend this simulation model.

10.2.2 Performing simulations with OLOOP1

First, open the system OLOOP1. Suppose we want to evaluate the responses of the aircraft to the (default) block-shaped elevator input. In that case we don't have to edit the system on the input side. Before we can start a simulation, it is necessary to define the system parameters in the MATLAB workspace. First of all, the system Beaver requires the parameter vector *GM1* and the parameter matrices *AM*, *EM*, and *GM2* to be present in the MATLAB workspace. These matrices are defined in appendix D. They can be retrieved from the file AIRCRAFT.DAT, which is contained in the FDC subdirectory DATA by means of the utility LOADER (see section 9.3.1). So after opening OLOOP1 first double-click the button LOADER. You must activate the MATLAB command-window yourself after double-clicking this button to specify the directory where LOADER will search for AIRCRAFT.DAT – usually the default directory will do fine. Next, the initial flight-condition must be defined or computed. You can use ACTRIM to determine a steady-state flight-condition (see the example in section 10.5), or use INCOLOAD to load a flight-condition from file. The FDC subdirectory DATA contains at least the following two files with steady-state initial flight conditions: CR4520.TRI and CR4560.TRI. The file-extension .TRI reveals that these files were created by the routine ACTRIM. Their filenames have the following meaning: CR means 'Cruise' condition, 45 denotes the initial value of the true airspeed in $[ms^{-1}]$, 20 denotes the initial flight-level, i.e. the altitude in $[ft]$ divided by 100. In this case, start INCOLOAD by double-clicking its button in OLOOP1 or by typing `incoload` at the command-line. Click the first button from the user-menu to retrieve a trimmed flight condition, and specify the directory, filename, and extension in the command-window. If you want to keep state variables from the aircraft model fixed to their initial values, run FIXSTATE by double-clicking its button in the system OLOOP1 or by typing `fixstate` at the command-line (see section 9.5). Here we will evaluate the complete aircraft model, so it is not necessary to run FIXSTATE.

Now start the simulation. For the systems from the FDC toolbox the simulation parameters already have been defined appropriately, so you can simply select *Start Simulation* in the simulation menu. By default, the system OLOOP1 sends the 8^{th} output, being θ,[1] to a Graph Scope which will display its time-trajectory in a graphical window and automatically sets the figure axes. This is useful for monitoring the simulation, but for real analysis of the results it is more convenient to make plots after finishing the simulation, using the simulation results from the workspace. If everything goes well you will see a good example of a short-period oscillation followed by the phugoid mode of the 'Beaver' aircraft. If you want to monitor another variable during simulations you must connect the Graph Scope to another output from the Demux block.

Try performing some more simulations – it is really quite simple! Notice that by default only one of the input signals, namely a block-shaped elevator deflection, is connected to the aircraft model. This may lead to some MATLAB warnings about unconnected output lines from the other input signal blocks, which in this case obviously can be disregarded. If you want to analyze responses to the other input signals, simply connect their output line to the appropriate Mux block.

[1] See the definition of the S-function output vector from equation 10.1 or enter `type outputs.hlp` at the command-line for on-line help.

10.2.3 Analyzing simulation results

After a complete simulation of OLOOP1, the workspace should contain the variables *time*, *In*, and *Out*, which represent the time-axis, input trajectories, and output trajectories, respectively. See section E.2 of appendix E for the definitions of *In* and *Out*, or type `type inputs.hlp` or `type outputs.hlp` for on-line help at the command-line. It is useful to check which variables are present in the workspace after finishing the simulation by typing `who`, because under certain circumstances SIMULINK does not properly send the simulation results to the workspace. This bug has been noticed for all SIMULINK versions for WINDOWS up to version 1.2c. If *time*, *In*, and *Out* are not present in the workspace after a simulation something has gone wrong. In that case try to recover the simulation results by running the routine RECOVER for the system Beaver (type `recover('beaver')` at the command-line). If this doesn't solve the problem you have to repeat the simulation. See also section 9.4.3.

The variables *time*, *In*, and *Out* may be used rightaway to plot the results, but it is also possible to run the MATLAB macro RESULTS first in order to simplify this process. Note: RESULTS can only be applied if you haven't changed the definitions of the input and/or output vectors which are sent to the workspace from the system Beaver. RESULTS yields a large number of new variables such as *alpha*, *deltae*, *Hdot*, etc. which can be plotted against time by typing `plot(time,alpha)`, `plot(time,deltae)`, etc. The variable names are mostly literal representations of the corresponding symbols; see appendix E or the source-code of RESULTS for more details. RESULTS can be started by typing `results` at the command-line, or by double-clicking its button block in OLOOP1. After running RESULTS you can plot the most important time-trajectories with the MATLAB macro RESPLOT. See also sections 9.4.1 and 9.4.2.

10.3 Non-linear responses to stochastic inputs – OLOOP2

10.3.1 Structure of the system OLOOP2

The system OLOOP2 (figure 10.2) contains an open-loop simulation model of the 'Beaver' aircraft flying through atmospheric turbulence. The corresponding 'tutorial' system OLOOP2T explains the function of each block in this system. Like OLOOP1, the core of this system is an S-function block which calls the system Beaver. A Demux block on the right-hand side of the system extracts the sixteen individual output signals from the S-function output vector. Each output from Demux can be connected to a Scope block for monitoring the simulations; in the example system a Graph Scope has been connected to the third output line which corresponds with the sideslip angle β (the complete definition of the S-function output vector from Beaver was given in equation (10.1) and can be displayed in the MATLAB command window by typing `type outputs.hlp`). On the input side a Mux block combines the two control input vectors and the vector with wind and turbulence velocities and their time-derivatives. Due to the fact that the aircraft model is not linear, it is necessary to enter the initial values of the control input vectors $\mathbf{u}_{aero}$ and $\mathbf{u}_{prop}$ into the system, even though we only want to analyze the responses to atmospheric turbulence. The atmospheric turbulence model was copied from the library WINDLIB, which was described in chapter 6. Here, Dryden filters with constant coefficients are used. The parameters from these filters can be changed by double-clicking the subsystem Atmospheric Turbulence Group and then double-clicking each Dryden Filter block. Of course, it is possible to connect and disconnect the Dryden Filter blocks according to your own wishes. By default the scale lengths are set to 150 m, the standard deviations are 1 ms^{-1}, and the velocity for which the filters are determined is 45 ms^{-1}. This velocity lies exactly between the upper and lower limits of the flight envelope of the 'Beaver'. On-line help for the subsystem Atmospheric Turbulence Group can be displayed by double-clicking its title-block, or by typing `type turb1.hlp` at the command-line.

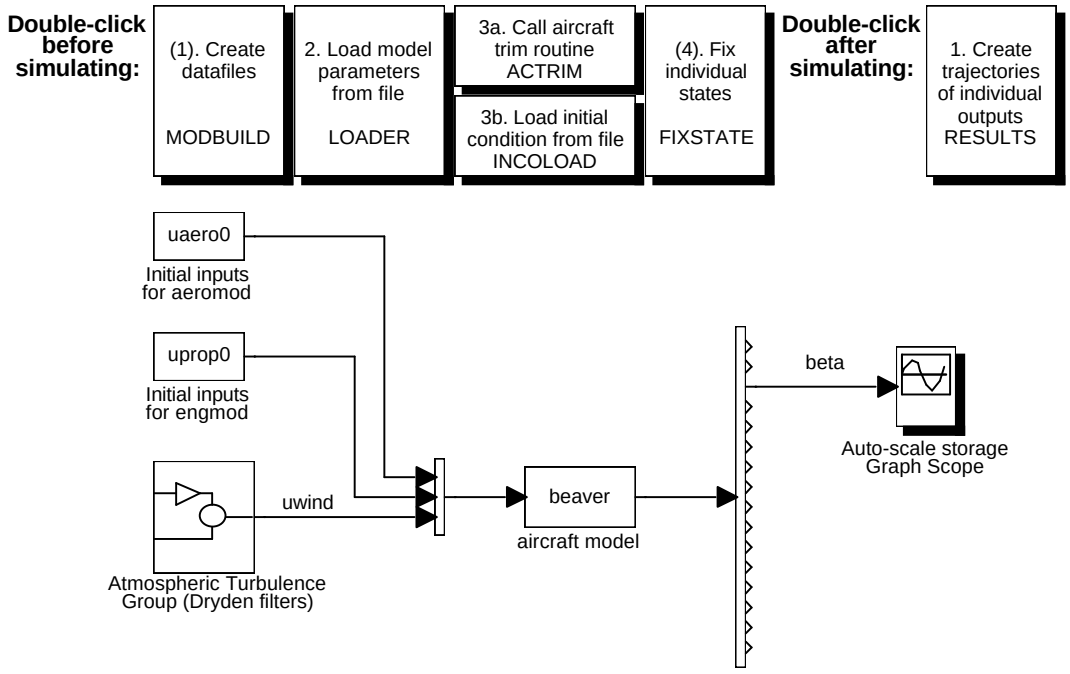


Figure 10.2: Block-diagram of the open-loop system OLOOP2

10.3.2 Performing simulations with OLOOP2 and analyzing the results

First, run LOADER in order to retrieve the model parameters for the system Beaver from the file AIRCRAFT.DAT. Use INCOLOAD to retrieve an initial condition from file, e.g. load CR4520.TRI or CR4560.TRI from the subdirectory DATA, or determine a steady-state initial condition with ACTRIM. Run FIXSTATE if you want to artificially fix state variables from the aircraft model. Next, start the simulation by selecting *Start* in the simulation menu. The results can be analyzed by means of RESULTS and RESPLOT. It is useful to make separate simulations for longitudinal, lateral, and vertical turbulence, by connecting the respective blocks within the subsystem Atmospheric Turbulence Group.

10.4 Linear responses to deterministic inputs – OLOOP3

10.4.1 Structure of the system OLOOP3

Figure 10.3 shows the system OLOOP3, which contains a *linear* open-loop simulation model of the 'Beaver'. In stead of the non-linear S-function block from OLOOP1 and OLOOP2, it defines the aircraft dynamics by means of a linear State-Space block, which extracts its model matrices *Aac*, *Bac*, *Cac*, and *Dac* from the MATLAB workspace. These matrices can be extracted from the non-linear system Beaver by means of the linearization program ACLIN; see section 8.3 for more details. The purpose of OLOOP3 is to determine linear aircraft responses to the same control inputs as used in the non-linear system OLOOP1. Contrary to OLOOP1, OLOOP3 is a small-perturbations model. Therefore, it was not necessary to add the initial conditions $\mathbf{u}_{aero}(0)$ and $\mathbf{u}_{prop}(0)$ to the test-signals, hence the test-signals could be connected directly to a Mux block to get the small-deviations version of the input vector from equation (10.2). For more information about OLOOP3 you should consult the 'tutorial' system OLOOP3T, which can be opened by typing `oloop3t` at the command-line.

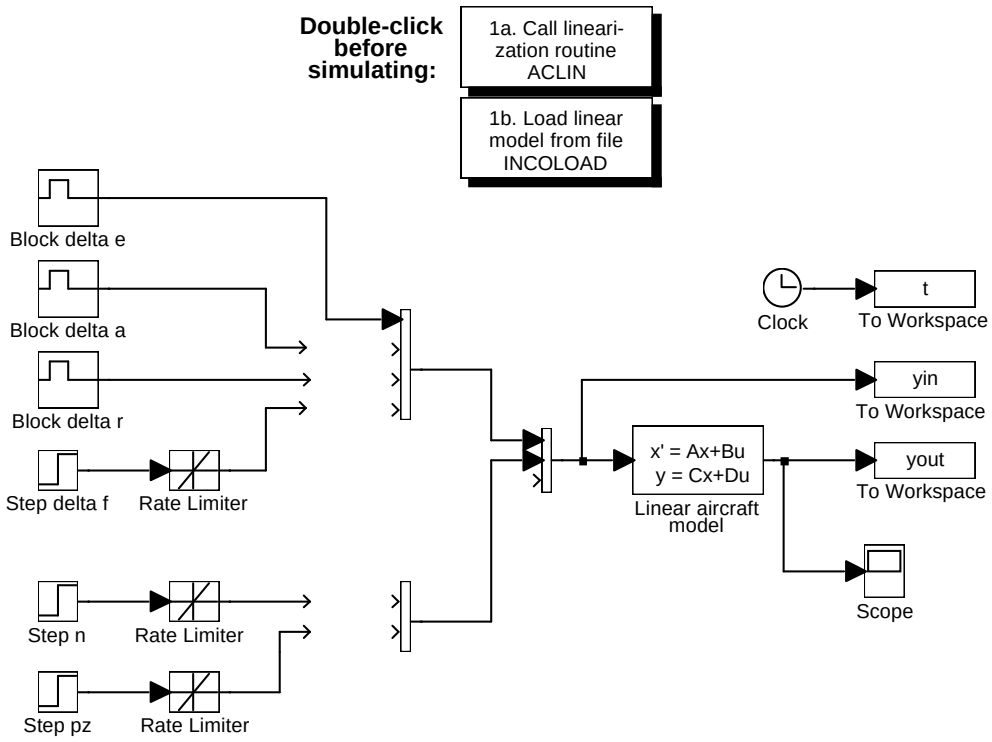


Figure 10.3: Block-diagram of the open-loop system OLOOP3

10.4.2 Performing simulations with OLOOP3 and analyzing the results

The definition of model parameters is different from the definitions for OLOOP1 and OLOOP2. Here, it is only necessary to define the matrices of the linear aircraft model (*Aac*, *Bac*, *Cac*, and *Dac*) in the MATLAB workspace. It is not necessary to define an initial condition, because the linearized model expresses all signals in deviations from their initial values. The aircraft model matrices can be loaded from file by running INCOLOAD or determined by the linearization program ACLIN. By default, the subdirectory DATA contains two files with linearized aircraft models: CR4520.LIN and CR4560.LIN.

For this example we will use ACLIN to obtain a linear aircraft model. Double-click the ACLIN button in OLOOP3, or type `aclin` at the command-line to start this program. You will be welcomed by the message from figure 10.4 and you must enter the name of the aircraft model (by default Beaver). Next, choose a method for defining the operating point in the workspace. In this case, click the first button of the user-menu to retrieve a trimmed-flight condition from file. Here we will load the file CR4560.TRI from the subdirectory DATA, see figure 10.6. ACLIN will then start LOADER to retrieve the model parameters for the non-linear aircraft model and start the linearization process itself. It is then possible to select a subset of the state equations and/or input equations, but since we need the full 12^{th}-order model, the questions from figures 10.9 and 10.10 are answered by pressing Enter. Save the results to the file TESTFILE.LIN in the subdirectory DATA.

All model matrices are now specified in the workspace and the simulation can be started by selecting *Start* from the simulation menu. If you want to view the process of the simulation itself, double-click the Scope block first before starting the simulation. Due to the fact that the model is now linear, the simulation will be very fast.[1] The results from the simulations will

[1]Due to the fact that it is a linear model, the integrator LINSIM was applied for simulations of the system

```
>> aclin

FDC 1.2 - ACLIN

Linearize nonlinear aircraft model in SIMULINK.
================================================

Enter name of the aircraft model in Simulink (default: BEAVER)

> beaver
```

Figure 10.4: When you start ACLIN you must first specify the model name

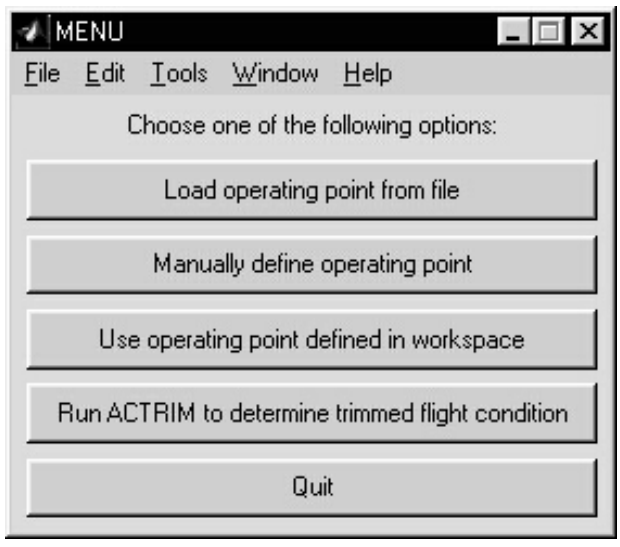


Figure 10.5: Click the first button fro the user-menu to load an operating point from file

```
Specify directory (default:  c:\fdc12\data):

Enter filename without extension (8 characters max.):

> cr4560

Enter extension (3 characters), default = tri:

>

Loading operating point from file

     c:\fdc12\data\cr4560.tri

<<< Press a key to proceed with model definition >>>
```

Figure 10.6: Specify the directory, filename, and extension (here: C:\FDC12\DATA\CR4560.TRI)

```
Loading model parameters from AIRCRAFT.dat
------------------------------------------

Specify directory (default:  c:\fdc12\data):

Datamatrices AM, EM, GM1, and GM2 loaded.

Ready.

<<< Press a key to proceed with linearization >>>
```

Figure 10.7: ACLIN now calls LOADER to retrieve the model parameters

```
Now linearizing S-function beaver

Wait a moment, please...

Linearization succeeded.

<<< Press a key to continue >>>
```

Figure 10.8: Next, the linearization function LINMOD is called

```
Select states
-------------

The current state vector is:

x = [ V  alpha  beta  p   q   r  psi  theta  phi  xe  ye  H ]'

      1    2      3   4   5   6   7     8     9   10  11  12

Enter vector with element numbers of states you want to use
(enter = use all states):

>
```

Figure 10.9: It is possible to select a subset of the state vector, but we need all state variables

```
Select inputs
-------------

The current control-input vector is:

u = [ deltae deltaa deltar deltaf   n   pz ]'

          1      2      3     4     5    6

Enter vector with element numbers of control inputs that you want
to use (enter = use all control inputs):

>

Include wind & turbulence inputs (y/n)? y
```

Figure 10.10: We also need all input variables, including the wind and turbulence inputs

```
State-space matrices of complete 12th-order system:
Aac, Bac, Cac, and Dac.

Save linear state-space model to file (y/n)? y


Enter data for storing the linear model.

Specify directory (default:  c:\fdc12\data):

Enter filename without extension (8 characters max.):

> testfile


The linear state-space model will be saved to the file:

     c:\fdc12\data\testfile.lin

Is this correct (y/n)? y

Include operating point {xinco,uaero0,uprop0} to file (y/n)? y


Saving linear state-space model to the file

     c:\fdc12\data\testfile.lin

<<< Press a key >>>
```

Figure 10.11: The results can be saved to a file (here: TESTFILE.LIN)

```
State-space matrices of linearized aircraft model: Aac, Bac, Cac,
and Dac (ac = aircraft). Operating point is defined by the vectors
xinco, uaero0, and uprop0.

Examine the nonlinear aircraft model in Simulink for the current
definition of the outputvector. The S-function BEAVER uses:

 y = [x' dH/dt pb/2V qc/V rb/2V]'

which contains all relevant information for the autopilot simu-
lation model APILOT.

See matrix lindef for more details!

Ready.
```

Figure 10.12: The final screen message at the end of the linearization process

be stored in the vector t, which contains the time-axis, and the matrices *yin* and *yout*, which contain the time trajectories of the input and output variables, respectively. Notice that these variables are not equal to the standard results *time*, *In*, and *Out*, due to the fact that OLOOP3 does not call the non-linear model Beaver which constructs the latter variables. For this reason it is not possible to apply RESULTS and RESPLOT here. In stead you must plot individual input and/or output trajectories by typing:

```
plot(t,yin(:,1));
plot(t,yin(:,2));
plot(t,yout(:,6));
```

and so forth. The definitions of *yin* and *yout* correspond with the definitions of the S-function input and output vectors given in equations (10.2) and (10.1), although here only *deviations* from nominal input and output values are represented.

10.5 Trim-demo: trimmed-flight elevator deflection curve

From the SIMULINK models we can extract quite useful information about the characteristics of the aircraft. For example, it is easy to determine the trimmed-flight elevator deflection curve, which provides information about the stability and control characteristics of the aircraft (see for instance ref.[15]). Figure 10.13 shows two trimmed-flight elevator curves for the 'Beaver' aircraft which were extracted from the SIMULINK system Beaver, using the trim algorithm from section 4.3. In the figure, these curves have been compared to flight-test results, in order to get a general idea about the validity of the model.[1] The solid line and cross-points denote computed and measured values for *low* engine power; the dotted line and circle-points denote computed and measured values for *high* engine power.

OLOOP3. For non-linear simulations this routine was not suitable, so RK45 or ADAMS/GEAR were applied for all other simulation models from FDC 1.2. See section 4.2 for more details about numerical integration methods.

[1]These flight-test results were obtained as a part of the practical training for students in Aeronautical Engineering at Delft University of Technology. Due to the somewhat limited accuracy of these measurements, the

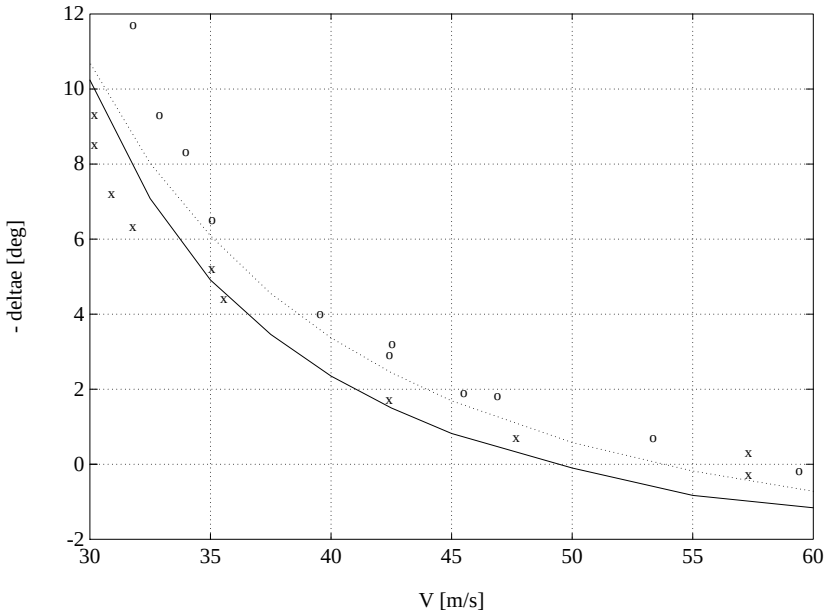


Figure 10.13: Trimmed-flight elevator deflection curves for the 'Beaver'

The MATLAB program TRIMDEMO demonstrates how these elevator deflection curves can be obtained. This program first asks the user to specify the altitude, flap-angle, and engine RPM for which the elevator curve should be computed. It automatically defines the other motion variables for wings-level flight conditions. The trim procedure from figure 4.9 is then performed for 10 different values of the airspeed, which all lay within the flight-envelope of the 'Beaver'. The part of TRIMDEMO.M which actually determines the trimmed flight conditions has largely been copied from the general trim program ACTRIM.M (see section 8.2).[1] After determination of the trimmed flight conditions, TRIMDEMO plots the relation between the steady-state elevator deflection δ_e and the true airspeed. Of course it is easy to modify TRIMDEMO.M if you want to view steady-state curves of different state and/or input variables. The source code of TRIMDEMO is contained in the file TRIMDEMO.M within the FDC subdirectory EXAMPLES. It is useful to compare this with the source-code from ACTRIM, which is stored in the file ACTRIM.M within the subdirectory TOOLS.

In stead of using TRIMDEMO it is also possible to create the elevator curve with the general aircraft trim program ACTRIM. In this case you will have to define the flight conditions manually for each velocity that you want to include in the δ_e-plot. Figures 10.14 to 10.21 demonstrate the complete trim-procedure for a velocity of 35 ms^{-1}. Of course, it is quite cumbersome to repeat this procedure for ten different velocities, so it is much more practical to use TRIMDEMO for this particular task. Still, this clearly demonstrates the practical use of ACTRIM.

flight test results should be used for *qualitative* comparison only.

[1] For future versions of the toolbox it is planned to divide the ACTRIM into more separate MATLAB functions, which will make it easier to write utilities like TRIMDEMO.

```
>> actrim

FDC 1.2 - ACTRIM
================

Find steady-state trimmed-flight condition for
nonlinear aircraft model in Simulink.

<<< Press a key >>>
```

Figure 10.14: When you start ACTRIM you get this welcome message

```
Loading model parameters from AIRCRAFT.dat
------------------------------------------

Specify directory (default:  c:\fdc12\data):

Datamatrices AM, EM, GM1, and GM2 loaded.

Ready.
```

Figure 10.15: ACTRIM first calls LOADER to retrieve the model parameters

```
Give name of system with aircraft model (8 characters max.)
default = beaver

> beaver

Simulink will first make an internal representation of
the system beaver. Press a key...

<<< Ready, press a key >>>
```

Figure 10.16: Next, the name of the aircraft model must be entered

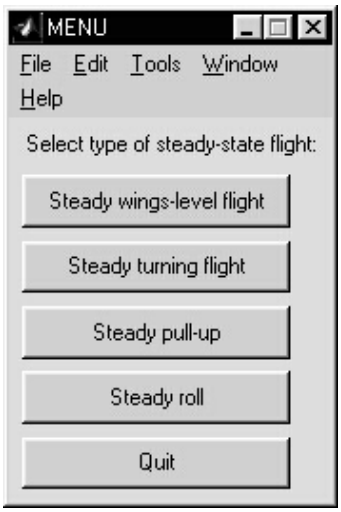


Figure 10.17: Next, select a trim-condition (here: steady wings-level flight, see next figure

```
Steady wings-level flight.
==========================

Give desired airspeed [m/s], default = 45: 35

Give (initial) altitude [m], default = 0: 2000*0.3048

Give heading [deg], default = 0:

Use specified flight-path angle or manifold pressure (f/m)? m

Give flap angle [deg], default = 0:

Give engine speed [RPM], default = 1800:

Give manifold pressure pz ["Hg], default = 20:


Searching for stable solution. Wait a moment...


Iteration stopped.

<<< Press a key to get results >>>
```

Figure 10.18: Some starting values for the trim process must be specified

```
State vector (trimmed):

                  x =  3.5000e+001
                       2.1131e-001
                      -2.0667e-002
                                 0
                                 0
                                 0
                                 0
                       1.9190e-001
                                 0
                                 0
                                 0
                                 0

Input vector (trimmed):

                  u =  -9.3083e-002
                       9.6242e-003
                      -4.9506e-002
                                 0
                       1.8000e+003
                       2.0000e+001
                                 0
                                 0
                                 0
                                 0
                                 0
                                 0

Time derivative of state vector (trimmed):

                xdot = -1.8871e-004
                      -1.2348e-005
                       4.6356e-004
                      -2.5027e-005
                      -2.0660e-005
                      -5.2604e-005
                                 0
                                 0
                                 0
                       3.4986e+001
                      -7.2330e-001
                      -6.7909e-001
```

Figure 10.19: The resulting states, inputs, and time-derivatives are shown

```
Save trimmed condition to file (y/n)? y
Enter data for storing the trimmed-flight condition.


Specify directory (default:  c:\fdc12\data):

Enter filename without extension (8 characters max.):

> testfile

The trimmed-flight condition will be saved to the file:

c:\fdc12\data\testfile.tri


Is this correct (y/n)? y

Saving trimmed-flight condition to the file

c:\fdc12\data\testfile.tri

<<< Press a key >>>
```

Figure 10.20: The results can be saved to a file (here: TESTFILE.TRI)

```
The results have been stored in the following variables:

xinco = [V alpha beta p q r psi theta phi xe ye H]' = state vector
xdot0 = dx/dt(0)
uaero0= [deltae deltaa deltar deltaf]' = initial input vector for
                                          aerodynamic model
uprop0= [n pz]' = initial input vector for engine model

The text-matrix 'trimdef' contains more info about the trimmed
flightcondition.

Ready.

>>
```

Figure 10.21: The final screen messages at the end of the trim process

Chapter 11

'Beaver' autopilot – theoretical backgrounds

11.1 Introduction

The SIMULINK models from chapters 5 to 7 played an important role for the design and evaluation of the control laws for the 'Beaver' autopilot at the Faculty of Aerospace Engineering, Section Stability and Control. This project served to gain practical experience in designing an autopilot from scratch to actual flight tests. The control laws were based upon *classical* control theory, which resulted in a *baseline autopilot* that served as an example for similar developments for the new Cessna Citation II 'National Fly-by-wire Testbed', and provided a basic reference against which modern control design methods could be measured. In this report, the resulting control laws will be treated as a case study which will demonstrate the power and flexibility of the FDC toolbox. This chapter describes the required theoretical backgrounds. The implementation of the control structure within SIMULINK will be treated in the next chapter. These two chapters are intended primarily as a demonstration for the more experienced FDC users, but they also provide a good basis for similar Automatic Flight Control System design projects in the future. However, for a detailed description about the 'Beaver' autopilot project the reader is referred to the MSc-theses of M.O. Rauw and P.N.H. Wever (refs. [22] and [29]), which, though not published for public use, may still be available at the Section Stability and Control of the Faculty of Aerospace Engineering.

11.2 Basic autopilot functions

The functions of an autopilot can be divided in the areas of *guidance* and *control*. These functions are defined as follows (ref.[20]):

Guidance: the action of determining the course and speed to be followed by the vehicle, relative to some reference system.

Control: the development and application of appropriate forces and moments to the vehicle, which: *(i)* establish some equilibrium state of vehicle motion, and *(ii)* restore a disturbed vehicle to its equilibrium state (operating point) and/or regulate, within desired limits, its departure from operating point conditions.

The boundary between these two areas is seldom inherently sharp, because of functional, operational, and equipment interactions that they may share. The *control* loops ensure a fast and stable response of the aircraft to the commands created by the *guidance* loops. They must also

eliminate the influence of external disturbances such as atmospheric turbulence. As a result of the separation between guidance and control task, the autopilot structure can be divided in *inner* and *outer* loops. The *control* function is fulfilled by the inner loops (figure 11.1), which control the pitch and roll angles of the aircraft, i.e. the aircraft's attitude relatively to the Earth. The actual pitch and roll commands are created by the outer loops which *guide* the aircraft, equipped with the inner-loop control structure, along the desired flight-path (figure 11.2).

A combination of control loops needed to fulfill a certain guidance or control function is called an *autopilot mode*. It is possible to make a distinction between lateral and longitudinal modes, even though the lateral and longitudinal motions of the aircraft are not totally independent. To prevent the lateral movements affecting the performance of the longitudinal guidance and control loops, it is necessary to include some lateral/longitudinal interconnections, e.g. a turn-compensation which compensates for lost of lift due to rolling (a lateral motion) by means of elevator deflection (a longitudinal control input).

11.3 The longitudinal autopilot modes

11.3.1 Pitch Attitude Hold mode

The Pitch Attitude Hold mode (PAH) is the basic longitudinal autopilot mode; it controls the pitch angle by applying appropriate deflections of the elevator if the actual pitch angle differs from the desired reference value. Normally, the PAH mode serves as inner loop for the Altitude Hold, Altitude Select, and Glideslope modes (after adding a filter in the θ-loop). It is also possible to select the PAH mode separately, for instance in order to control the pitch-attitude of the aircraft by means of longitudinal *side-stick* inputs (fly-by-wire control). In addition, the longitudinal part of the Go-Around mode is based upon the PAH control laws, see section 11.3.5. The pitch angle θ is fed back to damp the phugoid mode of the 'Beaver' and to ensure that the desired pitch angle is maintained. A proportional and integrating controller is applied in order to make sure that no steady-state errors in the pitch angle will remain. As long as the error signal $\theta - \theta_{ref}$ is not equal to zero, the signal from the integrator will increase, which leads to an increasing elevator deflection which eliminates the error. A feedback-loop of the pitch rate to the elevator has been included to compensate for the small decrease in damping of the short-period mode due to the θ-feedback. See refs.[6], [19], or [20].

The block-diagram of the PAH mode is shown in figure 11.3. A loop for turn-compensation has been added later, see section 11.5. Also, some signals from this block-diagram will be limited. The feedback-signals are obtained by means of on-board sensors which have not been drawn in this block-diagram. All gain factors are functions of the airspeed V, see table 11.3 at the end of this chapter.

11.3.2 Altitude Hold mode

The Altitude Hold mode (ALH) is used to maintain a reference altitude which is specified by the pilot. This mode uses the PAH mode with an additional *washout filter* in the θ-loop as inner-loop. In other words: the ALH mode fulfills a basic *guidance* function of the 'Beaver' autopilot. If the reference altitude differs too much from the actual altitude, the *mode controller* automatically switches to the Altitude Select mode, see section 11.3.3.

The difference between the reference altitude and the actual altitude, $\Delta H = H_{ref} - H$, is fed back via an amplifier to the inner loops, hence, the outer loop generates a pitch command signal θ_{ref} for the inner loops. The washout filter in the θ-loop is necessary, because as soon as

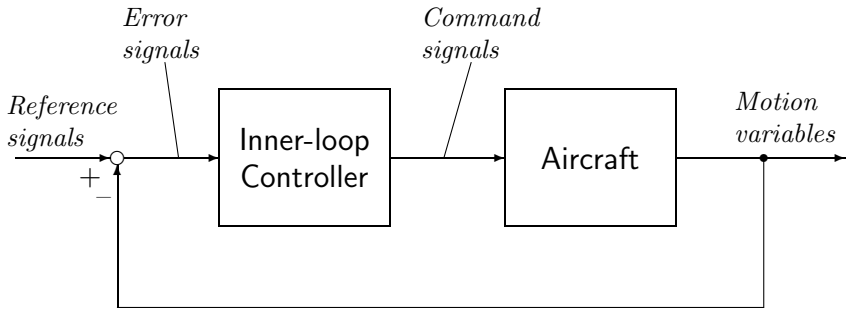


Figure 11.1: Control function fulfilled by inner-loops of the autopilot.

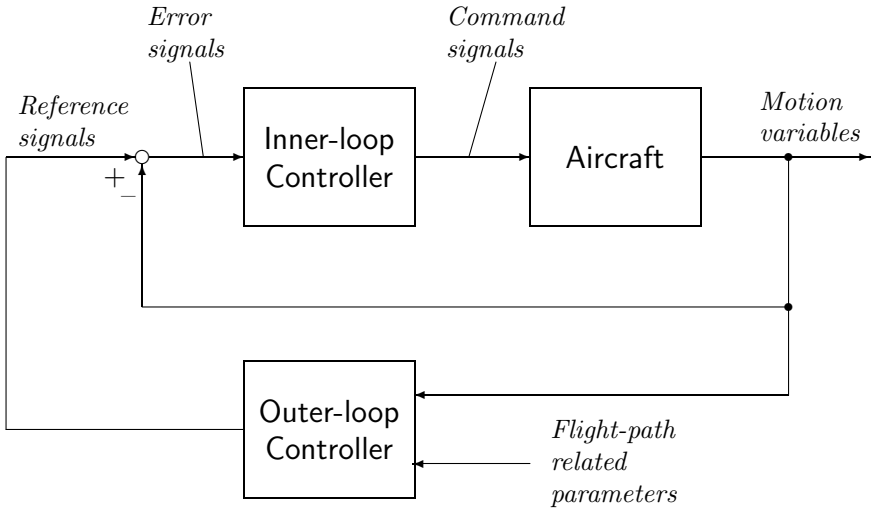


Figure 11.2: Guidance function fulfilled by outer-loops of the autopilot.

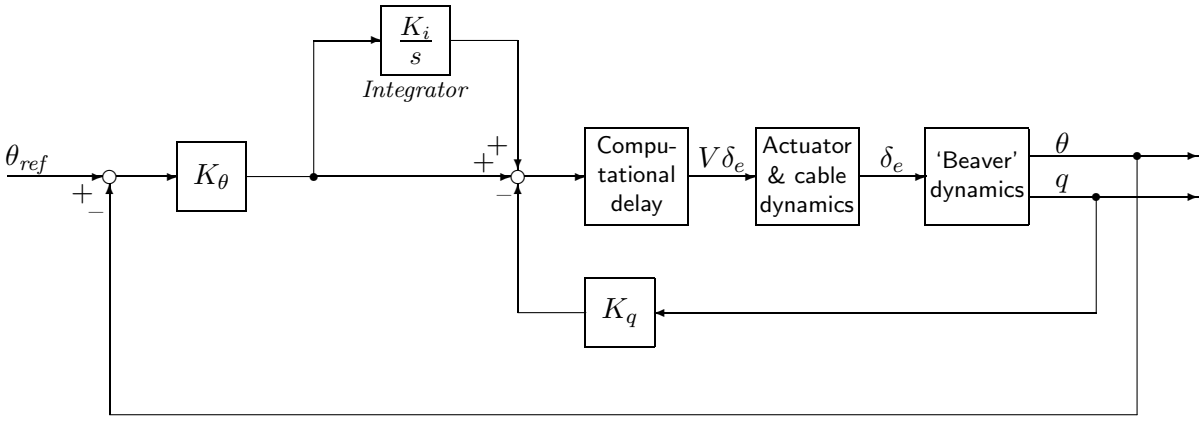


Figure 11.3: Block-diagram of the Pitch Attitude Hold mode (without turn-compensation)

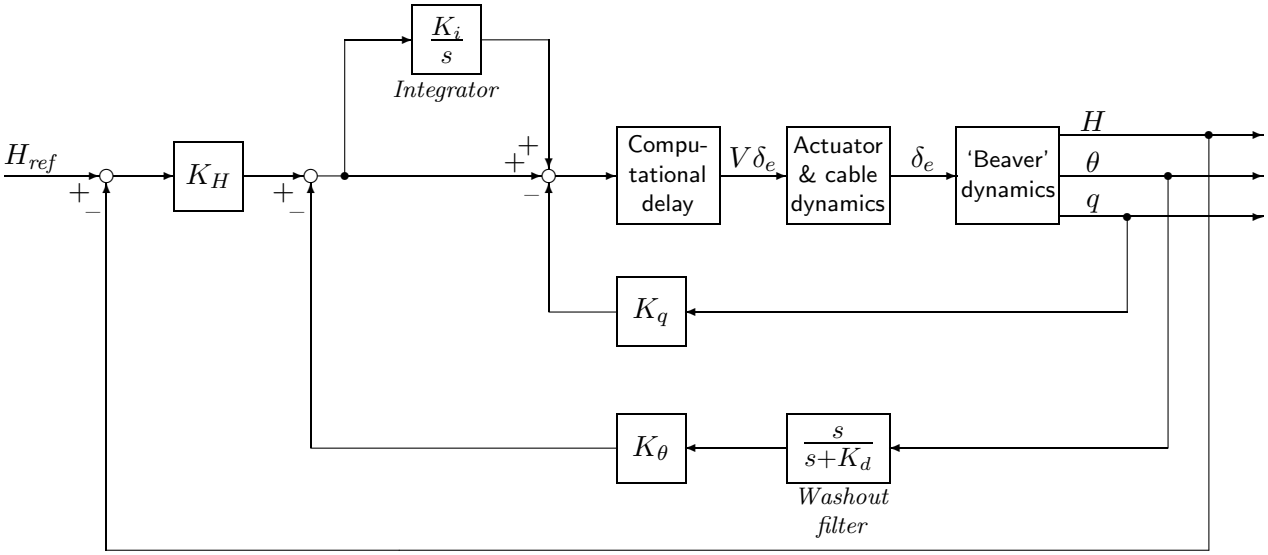


Figure 11.4: Block-diagram of the Altitude Hold mode (without turn-compensation)

H_{ref} is reached the command signal θ_{ref} will become zero, whereas the actual value of the pitch angle in level-flight usually differs from zero. But since the signal that leaves the washout filter will be very small if the changes in pitch angle are equal or close to zero, the inner-loop does not try to maintain a pitch angle $\theta = 0$ anymore.

The block-diagram of the ALH mode is shown in figure 11.4. An additional loop for turn-compensation has been added later, see section 11.5. The feedback-signals are obtained by means of on-board sensors which have not been drawn in this block-diagram. The gains depend upon the airspeed V, see table 11.3 at the end of this chapter.

11.3.3 Altitude Select mode

The Altitude Select mode actually controls the rate of climb of the aircraft. The climb rate $\dot{H}$ is fed back via a filter to the pitch channel. The PAH mode with an additional washout filter in the θ-loop serves as inner-loop for this mode, so the ALS mode can be regarded as a *guidance* mode. The mode controller of the 'Beaver' autopilot automatically decides which rate of climb is to be maintained if the pilot enters a certain desired reference altitude. If the difference between the reference altitude and the actual altitude exceeds a certain value, the ALS mode will automatically be engaged. The mode controller switches from ALS to ALH mode if the aircraft enters a certain 'altitude window' around the desired altitude. It is essential that the pilot maintains a sufficient airspeed by increasing the engine power during climbs, because otherwise the reference value of the rate of climb cannot be reached. This is due to the fact that the system does not include an *autothrottle* which would take care of selecting the right engine power itself (this is not uncommon in simple general-aviation autopilots).

The block-diagram of the ALS mode is given in figure 11.5. Notice that the feedback-signals are obtained by means of on-board sensors which have not been drawn in this block-diagram. The gains depend upon the airspeed V, see table 11.3 at the end of this chapter.

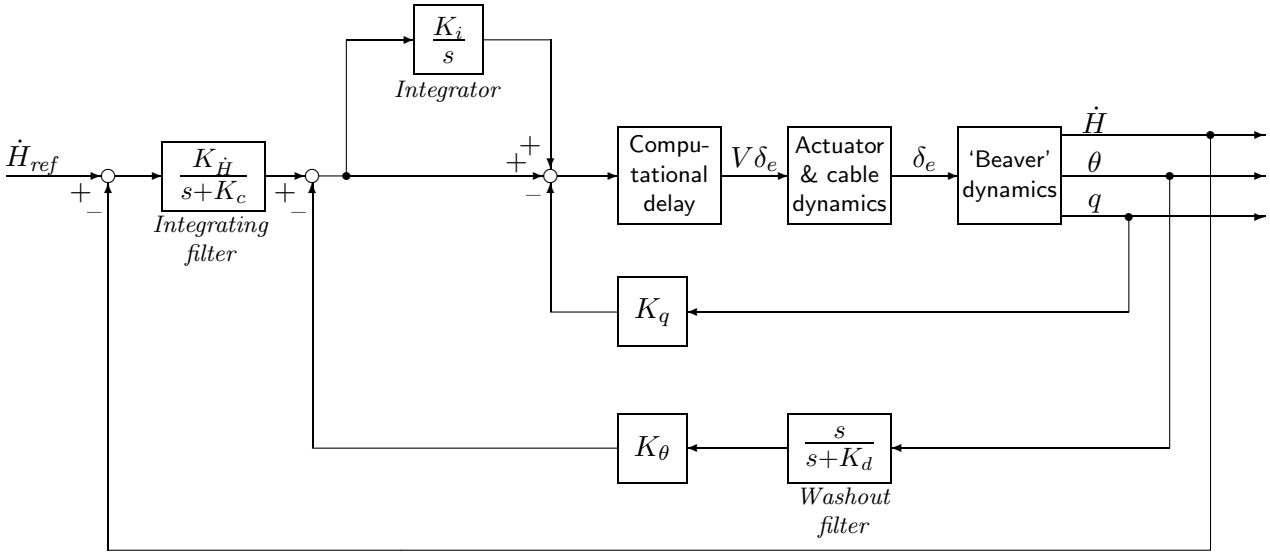


Figure 11.5: Block-diagram of the Altitude Select mode

11.3.4 Longitudinal part of the Approach mode: Glideslope

In the Approach mode, the 'Beaver' is guided along the reference planes of the glideslope and localizer. These reference planes are provided by radio signals of the Instrument Landing System (ILS), which can be detected in the aircraft (see for instance ref.[3]). The Glideslope mode (GS) is the longitudinal part of the Approach mode which brings the aircraft from level-flight into a descent, following the glideslope reference plane. The glideslope signal is emitted by an antenna which is located at some 300 meters beyond the runway threshold. The angle between the glideslope reference plane and the horizontal equals some value between 2 and 4 degrees (refs.[1], [3], and [14]). See section 3.4.1 for a description of the ILS system.

The Glideslope mode uses a feedback signal of the error angle ε_{gs} between the line through the aircraft and glideslope transmitter and the glideslope reference plane. An estimate of the rate of change of this error angle is also obtained by applying a differentiating filter $\frac{s}{s+1}$ to this feedback signal. This helps reducing the overshoot when the aircraft captures the reference glide path. The differentiating filter is engaged as soon as the autopilot is turned on, to ensure that the transients during the first couple of seconds after engaging the filter do not affect the glideslope performance. The filter provides a good approximation of the actual time-derivative due to its very small phase-lag. Both ε_{gs} and the distance from the aircraft to the glideslope reference plane d_{gs} are measured positive if the aircraft flies above the reference glide path.

There are two different phases during a glideslope approach:

1. *Glideslope Armed.* This phase is engaged as soon as the approach mode is selected by the pilot. The longitudinal autopilot mode in which the aircraft flew before selecting the approach mode, usually ALH, will be maintained until the aircraft has reached the glideslope reference plane.
2. *Glideslope Coupled.* This phase is initiated as soon as the aircraft passes the glideslope reference plane for the first time. In this phase the control laws of the GS mode take over the longitudinal guidance task of the autopilot.

The block diagram of the GS Coupled mode is shown in figure 11.6. Notice that the feedback

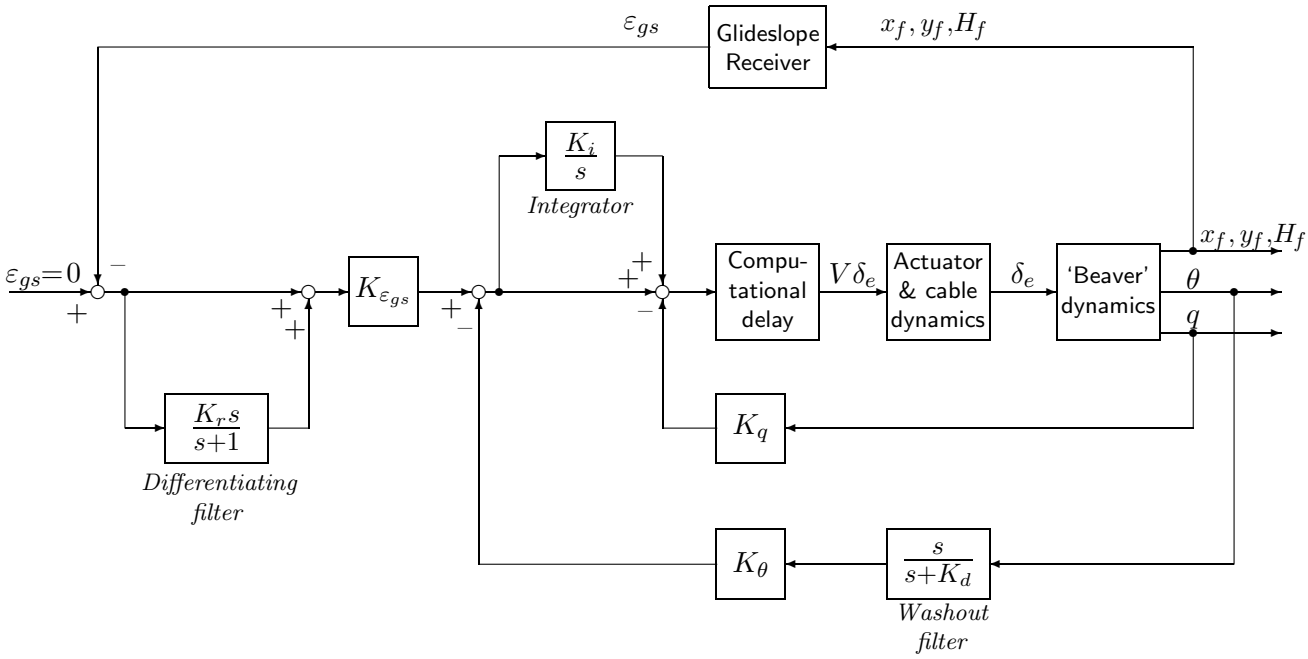


Figure 11.6: Block-diagram of the Glideslope Coupled mode

signals θ and q are measured by means of on-board sensors which have not been drawn in figure 11.6. The signal from the glideslope receiver actually depends upon the geographical position of the aircraft relatively to the runway, which in figure 11.6 has been interpreted as a feedback of the coordinates x_{ref} and y_{ref}, and the height H_{ref} to the glideslope receiver. The gains from this block-diagram depend upon the airspeed V, see table 11.3 at the end of this section. In addition, the gain $K_{\varepsilon_{gs}}$ is reduced as the aircraft nears the runway threshold in order to compensate for the increasing sensitivity in the measurements of the ε_{gs}-signal. The closer the aircraft flies with respect to the glideslope transmitter, the larger the angle ε_{gs} becomes for a constant distance to the glideslope reference line. Since this effect is comparable with an increasing $K_{\varepsilon_{gs}}$, it can be compensated by reducing this gain. If $K_{\varepsilon_{gs}}$ is kept constant the system would become unstable if the distance to the glideslope transmitter is reduced below a certain limit value.

For some runways it is possible to compute the distance to the runway threshold by means of information from the Distance Measurement Equipment (DME), but in general the distance to the glideslope transmitter must be obtained in a different way. For the 'Beaver' autopilot it was not possible to use DME information at all due to hardware limitations. Therefore, the distance to the glideslope transmitter was computed as a function of the altitude, which yields a satisfactory approximation if the deviation from the nominal glide-path is small:

$$R \approx \frac{H_{ref}}{\sin|\gamma_{gs}|} \quad \left(= \sqrt{{H_{ref}}^2 + {R_{gs}}^2}\right) \tag{11.1}$$

where R is the three-dimensional distance to the transmitter, H_{ref} is the height *above the airfield*, and γ_{gs} is the flight-path angle which an aircraft that flies along the nominal glide path would have. R_{gs} is the (unknown) ground-distance to the glideslope transmitter. It is common practice to use a radio-altimeter to determine H_{ref} for such gain-scheduling purposes. For the 'Beaver' aircraft we had to use the pressure altitude above sea level, corrected for the elevation of the airfield itself. This works well as long as the elevations of all relevant airfields are available in some kind of database within the Flight Control or Flight Management Computer.

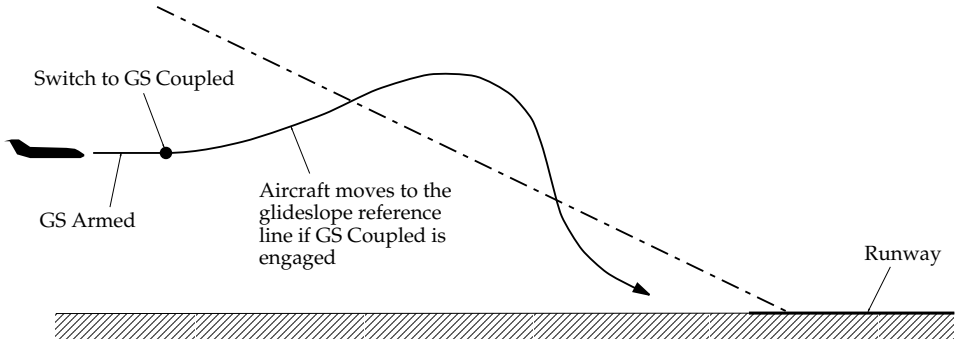


Figure 11.7: Response of the aircraft if the Coupled phase is entered too soon

The mode controller takes care of switching from *Armed* to *Coupled*. It constantly evaluates a switch-criterion which checks if the aircraft crosses the glideslope reference line for the first time. Although this yields a small overshoot in the error angle ε_{gs}, the response of the aircraft is far more desirable than the response shown in figure 11.7, which would have occurred if the Coupled mode would have been switched on earlier. The differentiating filter of the Glideslope Coupled mode also helps in achieving a satisfactory transition from Glideslope Armed to Glideslope Coupled. After switching from Armed to Coupled the autopilot will remain in Glideslope Coupled mode unless the pilot completely de-selects the Approach mode. The Glideslope Coupled control laws only work correctly if the pilot reduces power manually after intercepting the glide path in order to maintain the desired approach speed. See section 3.4.1 for more details about the glideslope signals.

11.3.5 Longitudinal part of the Go Around mode

The Go Around mode is used to cancel an approach. It has both longitudinal and lateral control loops. The longitudinal part of the GA mode in fact is a special case of the PAH mode. It uses the same control logic and inserts a step-wise increase in reference pitch angle of 10°. The pilot must manually apply full engine power. Notice that this makes the Go Around mode potentially very dangerous: if the pilot does *not* apply full power, the aircraft will stall! Therefore the current longitudinal Go Around mode should be equipped with more safety measures to become actually useful. See section 11.4.5 for a description of the lateral part of the GA mode.

11.4 The lateral autopilot modes

11.4.1 Roll Attitude Hold mode with turn-coordinator

The Roll Attitude Hold mode (RAH) is the basic lateral autopilot mode. Its main purpose is to serve as inner-loop for the other lateral autopilot modes, but it is also possible to use the RAH mode separately, for instance to control the roll angle by means of side-stick inputs (fly-by-wire control). The deviation of the actual roll angle from the desired roll angle is fed back to the ailerons via a proportional, integrating controller. The integrator ensures that the desired roll angle is actually reached without a remaining steady-state error. The RAH control loops are equipped with a *turn-coordinator*, which fulfills two functions: it must *(i)* suppress the sideslip angle in turns with appropriate deflections of rudder and ailerons, and *(ii)* suppress the adverse

yaw which occurs when a turn is initiated by deflecting only the ailerons.[1]

1. In a coordinated turn it is necessary to apply both aileron and rudder deflections, which must have opposite signs (ref.[15]). The deflections of the ailerons and rudder depend on the true airspeed and the yaw rate or roll angle (ref.[29]). Using linearized models of the 'Beaver' from ref.[26] these deflections were determined for a number of different airspeeds as a function of the yaw rate. Non-linear simulations in SIMULINK were used to further fine-tune the aircraft responses. The resulting control structure almost completely eliminates the sideslip-angle in turns. The turn-coordinator relations were added to the basic roll angle controller by means of airspeed-dependent factors *dar* and *drr* which, when multiplied with the yaw rate r, determine the turn-coordination corrections to the aileron and rudder deflections δ_a and δ_r.
2. The suppression of the adverse yawing due to aileron deflection is based upon the following relation which is valid only in coordinated turns:

$$r = \frac{g}{V} \sin \varphi \tag{11.2}$$

   If the yaw rate is too small, that is: if $\frac{g}{V} \sin \varphi > r$ the turn obviously is not coordinated and it is necessary to apply a larger deflection of the rudder. If $\frac{g}{V} \sin \varphi < r$, the aircraft is yawing too fast and a smaller deflection of the rudder is required. Therefore the factor $\frac{g}{V} \sin \varphi$ has been fed back to the rudder channel via the factor K_r.

The resulting deflection of the ailerons is equal to the sum of the deflection needed according to the actual control loop of the RAH mode and the deflection needed to maintain a zero value of the sideslip-angle. Rudder deflection is determined by the sum of the deflection for sideslip suppression and the corrections to suppress adverse yawing.

Figure 11.8 gives the block-diagram of the RAH mode with turn-coordinator. The feedback-signals are obtained by means of on-board sensors which have not been drawn in this block-diagram. The gains and correction factors in figure 11.8 all depend upon the true airspeed V; see table 11.3 at the end of this chapter.

11.4.2 Heading Hold/Heading Select mode

The Heading Hold / Heading Select mode (HH) is used to maintain or select a certain heading of the vehicle. It uses the yaw angle as feedback-signal. The difference between the desired yaw angle and the actual yaw angle determines the magnitude of the roll angles needed to turn the aircraft to the desired heading. (Note: in practice the pilot would rather want to control the *azimuth angle* χ in stead of the yaw angle ψ, but since the sideslip angle β is kept minimal the two angles are practically the same. However, the pilot himself must make the proper corrections for the drift-angle due to wind!) With regard to the control laws there is no difference between Heading Hold and Heading Select. They both use the RAH control loops with turn-coordinator as inner-loops, hence the HH mode fulfills a lateral *guidance* task.

Figure 11.9 shows the resulting block-diagram for the HH mode. The feedback-signals are

[1] 'Adverse yaw' arises from the difference in drag of the down-aileron compared to the drag of the up-aileron. The sign of the stability-derivative $C_{n_{\delta_a}}$ depends mainly on the rigging of the ailerons, their profile drag characteristics, and the angle of attack of the airframe. Aileron deflections can also produce side forces on the vertical tail which can become important contributors to $C_{n_{\delta_a}}$ (ref.[20]). A negative value of $C_{n_{\delta_a}}$ causes the aircraft to yaw initially in a direction opposite to that desired by the pilot; the resulting movement is therefore called *adverse* yaw.

obtained by means of on-board sensors which have not been drawn in this block-diagram. All gains are functions of the true airspeed; see table 11.3 at the end of this chapter.

11.4.3 Lateral part of the Approach mode: Localizer

In the Approach mode the 'Beaver' autopilot uses localizer signals from the Instrument Landing System (ILS) for guiding the aircraft to the runway centerline. A description of the ILS system can be found in section 3.4.1. The vertical approach guidance is performed by the Glideslope mode, as shown in section 11.3.4.

The Localizer mode (LOC) uses a feedback signal of the angle Γ_{loc} between the line through the aircraft and the localizer transmitter and the centerline of the runway and its time-derivative. In the flight-tests this signal proved to contain so much noise that an estimation of the angle Γ_{loc} was made by comparing the *heading* of the aircraft with the runway heading. A 'mix' of this estimated error angle and the measured angle was used in the feedback-loop in order to get a satisfactory behavior despite the noise while maintaining enough robustness to account for crosswind. This refinement will not be taken into account here.

There are two phases in the lateral approach guidance:

1. *Localizer Armed.* In this phase the autopilot keeps using the lateral autopilot mode controlling the aircraft (usually Heading Hold) until the aircraft comes near enough to the localizer reference plane. Exactly how soon the Localizer Coupled phase is entered is determined by the rate of change of the angle Γ_{loc} in the Localizer switch-criteria.
2. *Localizer Coupled.* This phase is entered if Γ_{loc} and $\dot{\Gamma}_{loc}$ satisfy the following switch-criteria:

$$K_{\dot{\Gamma}_{loc}}\Gamma_{loc} + \Gamma_{loc} > 0 \quad \wedge \Gamma_{loc} < 0 \tag{11.3}$$

or:

$$K_{\dot{\Gamma}_{loc}}\Gamma_{loc} + \Gamma_{loc} < 0 \quad \wedge \Gamma_{loc} > 0 \tag{11.4}$$

where Γ_{loc} is positive if the aircraft flies at the right-hand side of the localizer reference plane while heading *towards* the runway. There are *two* criteria, because it must be possible to approach the localizer reference plane from two sides.

In the 'Beaver' autopilot, the time-derivative of the localizer signal Γ_{loc} is approximated with a differentiating filter $\frac{s}{s+1}$. This filter is enabled as soon as the autopilot is turned on, to make sure that its transient effects have been died out if the LOC mode is actually switched on. This filter has a very small phase lag; therefore it approximates the actual time-derivative quite well. Without the $\dot{\Gamma}_{loc}$-feedback the aircraft tends to fly to the localizer reference plane as fast as possible, thereby neglecting the wish to make a smooth interception of the centerline. This has been illustrated in figure 11.10.

There is a fixed relation between the lateral distance to the centerline d_{loc}, the angle Γ_{loc}, and the distance to the Localizer transmitter R_{loc}. If the aircraft nears the runway the latter distance will decrease. If d_{loc} is kept constant while reducing R_{loc}, the angle Γ_{loc} will *increase*. This increases the sensitivity of the LOC mode, which leads to instability of the control mode when the aircraft comes too close to the runway. In a similar way as for the Glideslope mode, this problem has been solved by reducing the gain $K_{\Gamma_{loc}}$ as a function of the factor:

$$\frac{H_{ref}}{\sin(|\gamma_{gs}| + \varepsilon_{gs})} + x_{loc} \quad \left(\approx \sqrt{{R_{loc}}^2 + {H_{ref}}^2}\right) \tag{11.5}$$

which is approximately equal to the three-dimensional distance from the aircraft to the localizer transmitter. H_{ref} is the height of the aircraft above the field, x_{loc} is the distance between the

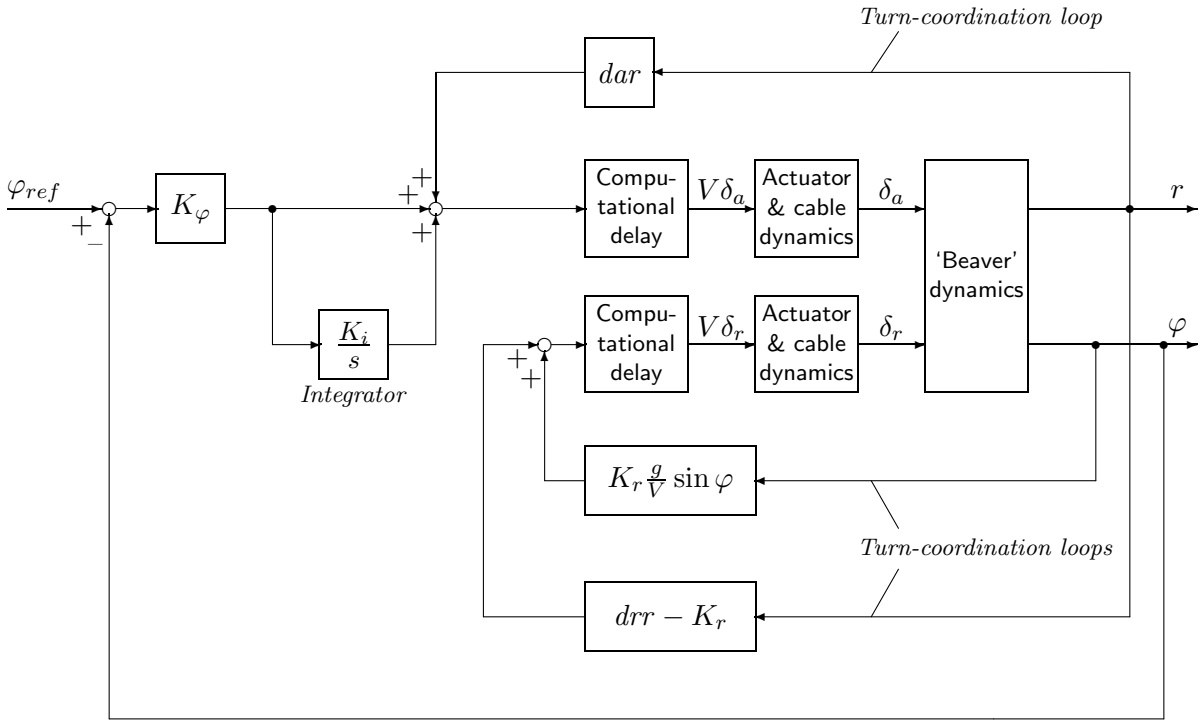


Figure 11.8: Block-diagram of the Roll Attitude Hold mode and turn-coordinator

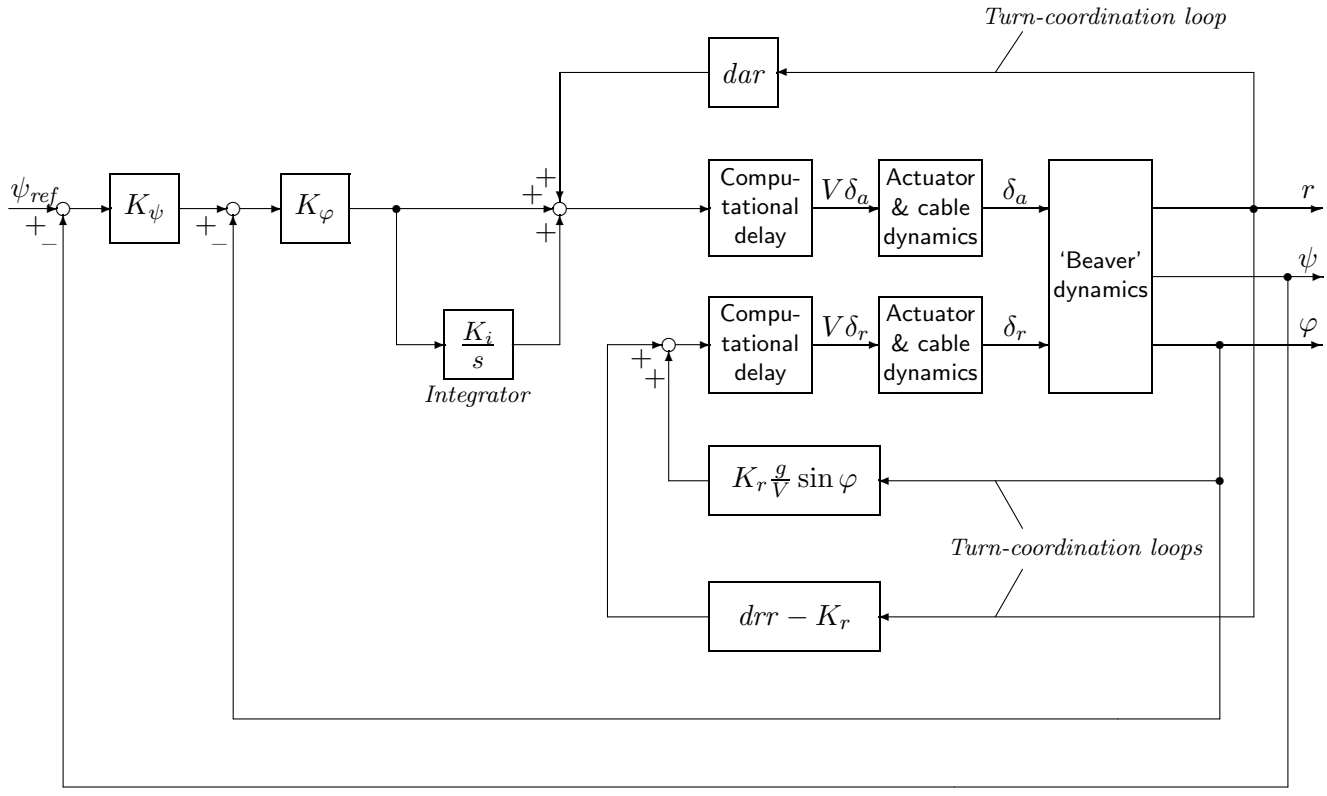


Figure 11.9: Block diagram of the Heading Hold / Heading Select mode with turn-coordination

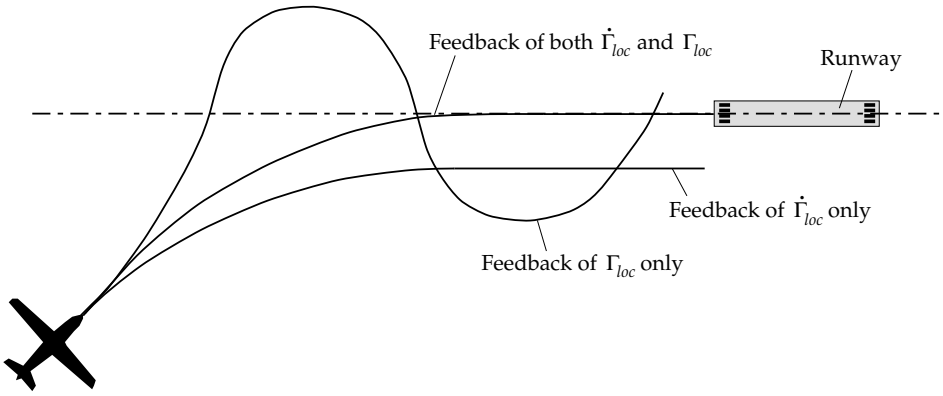


Figure 11.10: Using Γ_{loc}, $\dot{\Gamma}_{loc}$, or a combination of both signals as feedbacks for the LOC mode

runway-threshold and the localizer transmitter, γ_{gs} is the nominal glide-path angle, and ε_{gs} is the error angle between the line through the aircraft and the glideslope transmitter and the nominal glideslope reference line, which is measured by the glideslope antenna on board the aircraft. For the 'Beaver' aircraft the height H_{ref} was determined from the pressure altitude above sea level, corrected for the elevation of the airfield itself. This makes it necessary to have information about the elevation of the runway available in the Flight Control Computer or Flight Management Computer of the aircraft.

The block-diagram of the Localizer Coupled mode is shown in figure 11.11. The gains from this diagram depend upon the true airspeed; see table 11.3 at the end of this chapter. Notice that the feedback signal φ is measured by means of an on-board sensor which has not been drawn in figure 11.11. The signal from the localizer receiver actually depends upon the geographical position of the aircraft relatively to the runway, which in figure 11.11 has been interpreted as a feedback of the coordinates x_{ref} and y_{ref}, and the height H_{ref} to the localizer receiver.

11.4.4 VOR navigation mode

In the Navigation mode (NAV), the aircraft is guided along a VOR-bearing which is selected by the pilot. For this purpose, the angle Γ_{VOR} between the desired VOR-bearing and the bearing on which the aircraft actually flies is used as a feedback signal. In principle, the NAV mode functions in a similar way as the LOC mode, the control structure of the LOC mode is not suitable for VOR navigation, because of the small values of Γ_{VOR} at large distances from the VOR beacon. The heading ψ is used as a feedback signal in stead of the rate of change of Γ_{VOR}, because at a large distance the angle Γ_{VOR} is too small to obtain an accurate value of its time-derivative. The combination of Γ_{VOR} and ψ for the NAV mode has a similar effect as the combination of Γ_{loc} and $\dot{\Gamma}_{loc}$ for the LOC mode for creating a better interception of a radial (see figure 11.10).

Just like the approach modes, the NAV mode has two different phases:

1. *Navigation Armed.* This phase is engaged as soon as the NAV mode is turned on. The lateral autopilot mode in which the aircraft was flying before the pilot selected the NAV mode, usually Heading Hold, is maintained until the aircraft flies through the selected VOR bearing. The mode controller then automatically switches to NAV Coupled.

2. *Navigation Coupled.* This phase is engaged if the aircraft passes the reference VOR bearing for the first time (then the error angle $\Gamma_{VOR} = 0$). The NAV Coupled mode makes the aircraft turn towards the VOR bearing and follow the selected reference line.

The Navigation Coupled control law uses the RAH mode with turn-coordination as inner loops. The signal Γ_{VOR} passes a first-order filter which eliminates the high-frequency components from the VOR signal for noise suppression. In stead of the signal $\dot{\Gamma}_{VOR}$, the heading ψ is coupled back. This loop is basically equal to the outer-loop of the Heading Hold / Heading Select mode, but it contains an additional washout filter that makes it possible to 'crab' along a VOR bearing if there is a side-wind component. This is due to the fact that the washout filter eliminates the low-frequency components from ψ, which naturally includes a constant crab-angle.

The block-diagram of the Navigation Coupled mode is shown in figure 11.12. Notice that the feedback-signals ψ and φ are obtained by means of on-board sensors which have not been drawn in this block-diagram. The gain values from this diagram are scheduled as a function of the airspeed. See table 11.3 at the end of this chapter for a comprehensive list. Contrary to the LOC mode, it was not possible to schedule the gains as a function of the distance to the VOR mode. In theory this would have been possible by applying DME information, since many VOR and DME stations are co-located at the same position. For the 'Beaver' project this was not possible due to hardware limitations. Therefore, the sensitivity of the NAV control law increases if the aircraft nears the beacon and decreases if the aircraft moves away from the beacon. If the distance to the VOR beacon is too small, the NAV mode becomes unstable. The gains were selected such that the system will become unstable if the aircraft enters the 'cone of silence' when cruising at an altitude of 1500 feet. This cone of silence is an area where the VOR signals cannot be received accurately, see section 3.4.2 and ref.[3].

11.4.5 Lateral part of the Go Around mode

The lateral part of the Go Around mode, which is activated if the pilot wants to cancel an approach, is effectively a special case of the Roll Attitude Hold mode. It uses the RAH control structure to maintain a roll angle of zero degrees, hence it serves as a wing-leveler for a canceled approach. See section 11.3.5 for a description of the longitudinal part of the GA mode.

11.5 Turn-compensation

11.5.1 Introduction

The assumption that the longitudinal and lateral motions of the aircraft are independent is no longer valid if the aircraft has a non-zero roll angle, because the lift force decreases if the pilot does not apply an appropriate deflection of the *elevator*. Although the aircraft has been equipped with an Altitude Hold control mode, it is still necessary to compensate for this effect in order to obtain a quicker reaction to the tendency to loose height. A compensation in the Pitch Attitude Hold mode also proved to be useful in order to obtain a smoother ride of the aircraft. In addition to the compensation for the loss of lift in turns, a correction of the measured pitch rate is also necessary.

11.5.2 Correction of the pitch rate in turns

If the aircraft turns with a roll angle φ and yaw rate r, the pitch rate gyro will measure a value $q_{tot} = q + r\cos\varphi$ in stead of the actual pitch rate q, as is illustrated in figure 11.13. In this figure,

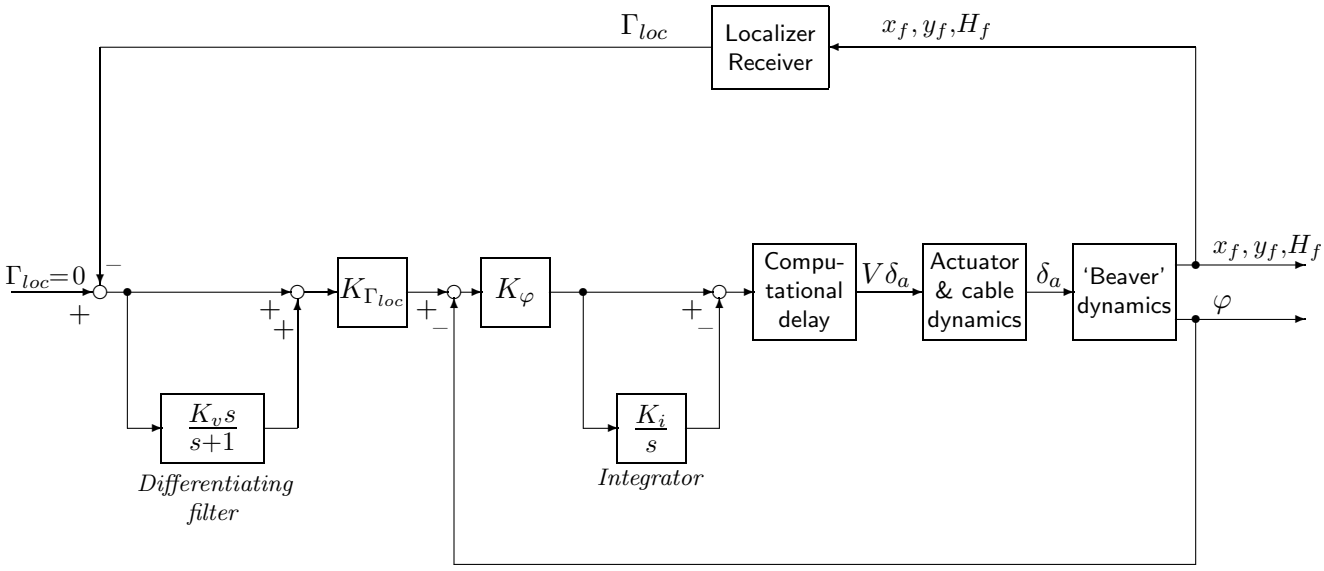


Figure 11.11: Block diagram of the Localizer mode (turn-coordination not shown)

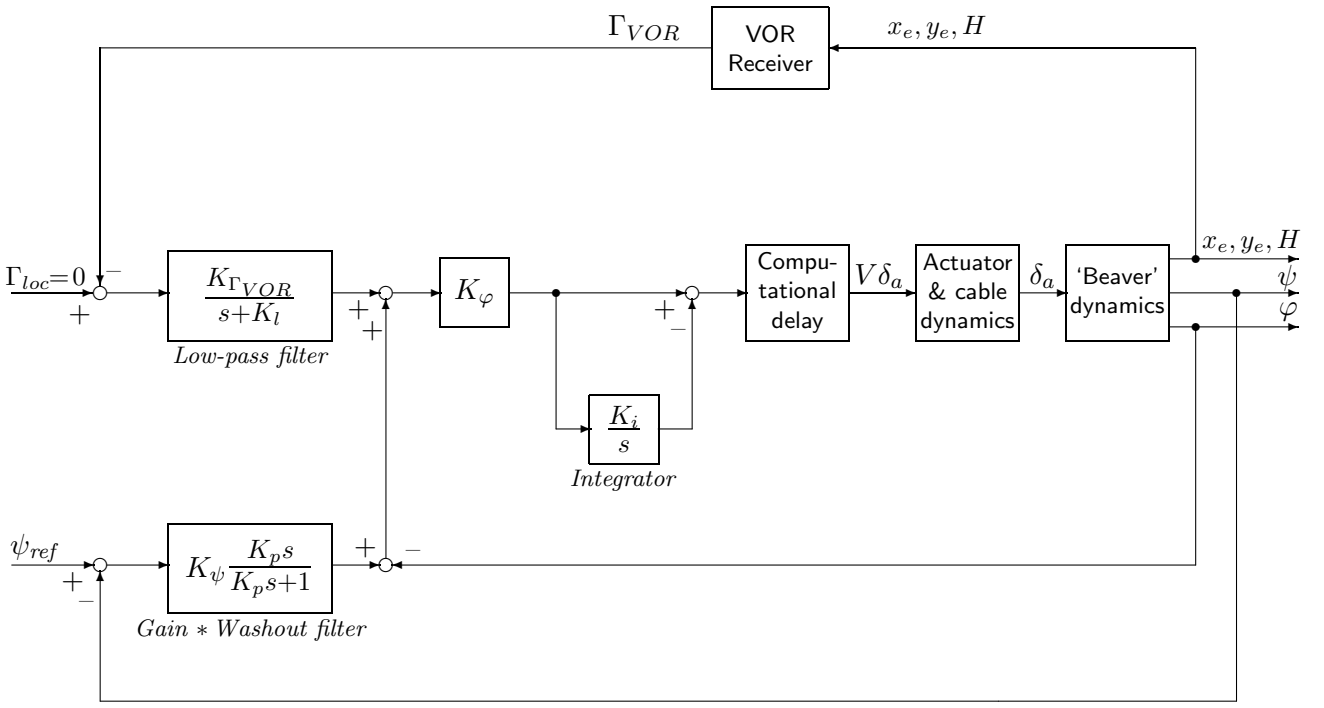


Figure 11.12: Block diagram of the VOR Navigation mode (turn-coordination not shown)

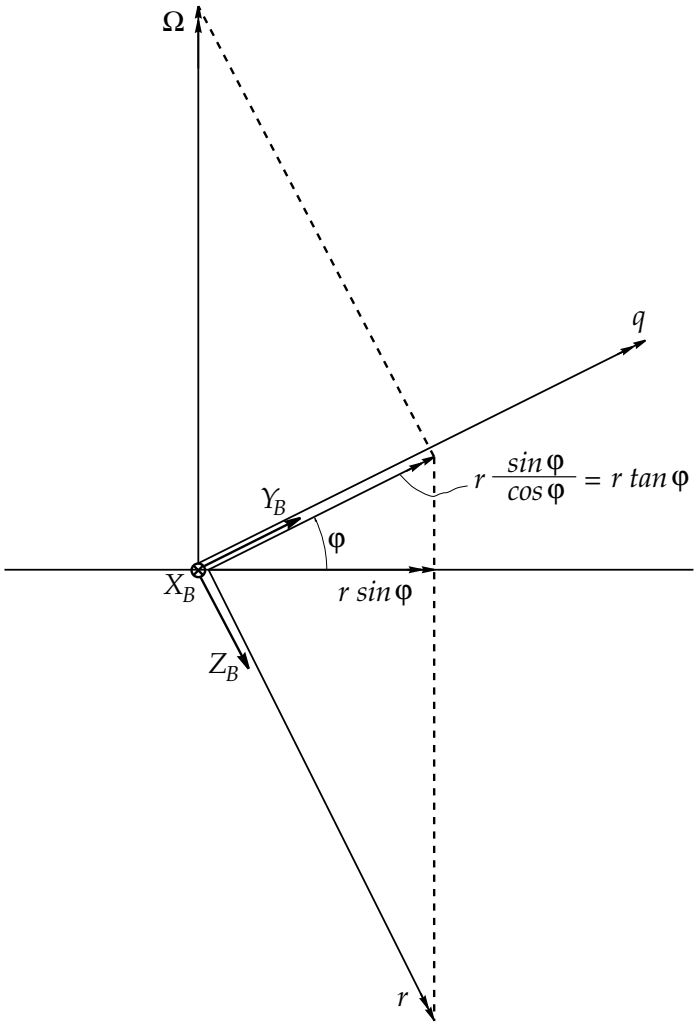


Figure 11.13: Contribution $r \tan \varphi$ to measured pitch rate for $\varphi \neq 0$

Ω is the angular velocity about the Z_V-axis. The additional component $r \cos \varphi$ should not be fed back to the q-loop of the longitudinal autopilot modes, because it has nothing to do with the longitudinal motions of the aircraft. If the measured pitch rate q_{tot} is not corrected properly, the term will yield a *positive*-valued contribution to the elevator deflection which results in an unwanted nose-down command.

11.5.3 Correction for the loss of lift in turns

For roll angles $\varphi \neq 0$ the lift force in Z_V-direction decreases, as shown in figure 11.14. In a horizontal, symmetrical, stationary flight condition the lift L is equal to the total weight W of the aircraft. If the aircraft has a roll angle φ, the total lift force must be increased to $L + \Delta L$ in order to maintain a lift component along the Z_V-axis that equals W. From figure 11.14 we can deduce that:

$$\begin{aligned} L' &= L \cos \varphi \\ L - L' &= L\,(1 - \cos \varphi) \\ \Delta L \cos \varphi &= L\,(1 - \cos \varphi) \end{aligned} \tag{11.6}$$

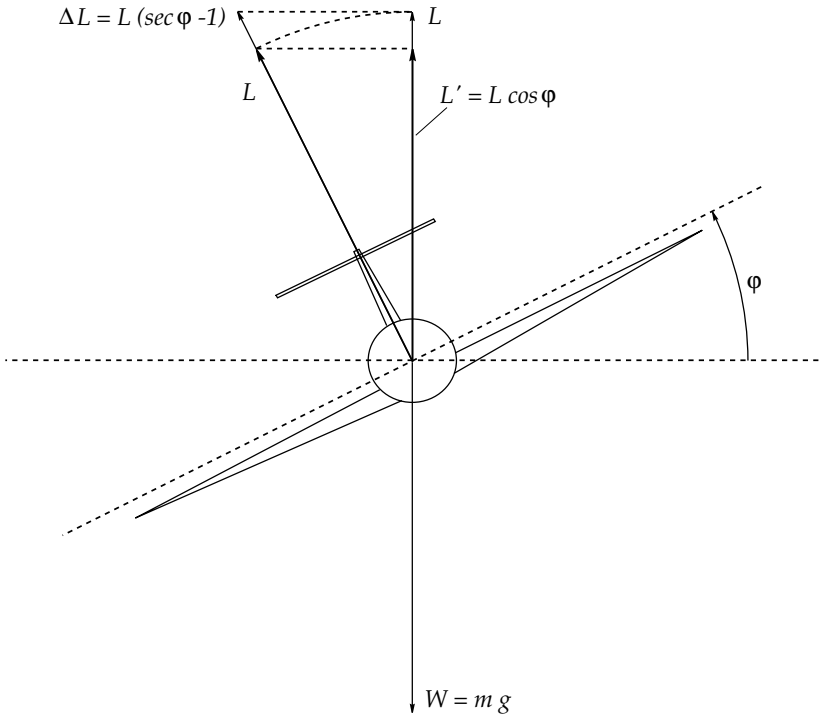


Figure 11.14: Loss of lift force for non-zero roll angle

So the required extra lift for a non-zero roll angle becomes:

$$\Delta L = L\left(\frac{1}{\cos\varphi} - 1\right) = L\left(\sec\varphi - 1\right) \tag{11.7}$$

To compensate for this loss of lift, a negative (= upward) contribution to the elevator deflection is needed. For the 'Beaver' aircraft, this contribution initially leads to a further decrease in the lift force due to the negative value of the stability derivative $C_{Z_{\delta_e}}$, but this is compensated due to the fact that the aircraft will rotate to a larger angle of attack. The overall effect is therefore a positive contribution to the lift-force. The additional deflection of the elevator needed to compensate for the loss of lift-force in turns can be written as:

$$\Delta\delta_e = K_{tc}{}^*(\sec\varphi - 1) \tag{11.8}$$

11.5.4 Total turn-compensation

It is convenient to express the compensation for the decreasing lift force in terms of a correction of q, since this makes it easier to combine this compensation with the correction for $r\tan\varphi$. The total correction to obtain the required pitch rate q from the measured value q_{tot} now becomes:

$$q = q_{tot} - (r\tan\varphi + K_{tc}\left(\sec\varphi - 1\right)) \tag{11.9}$$

where $K_{tc} = -K_{tc}{}^*/K_q$. The values of the gain K_{tc} were obtained by means of non-linear simulations of the aircraft flying under Pitch Attitude Hold or Altitude Hold control in combination with large roll-attitude commands for the Roll Attitude Hold mode for three different initial values of the airspeed. Table 11.3 shows the resulting gains as a function of the airspeed. The turn-compensation for the ALH mode is stronger than the compensation for PAH, i.e. K_{tc} is larger for the ALH mode than for PAH, because of the different functions of these two longitudinal modes:

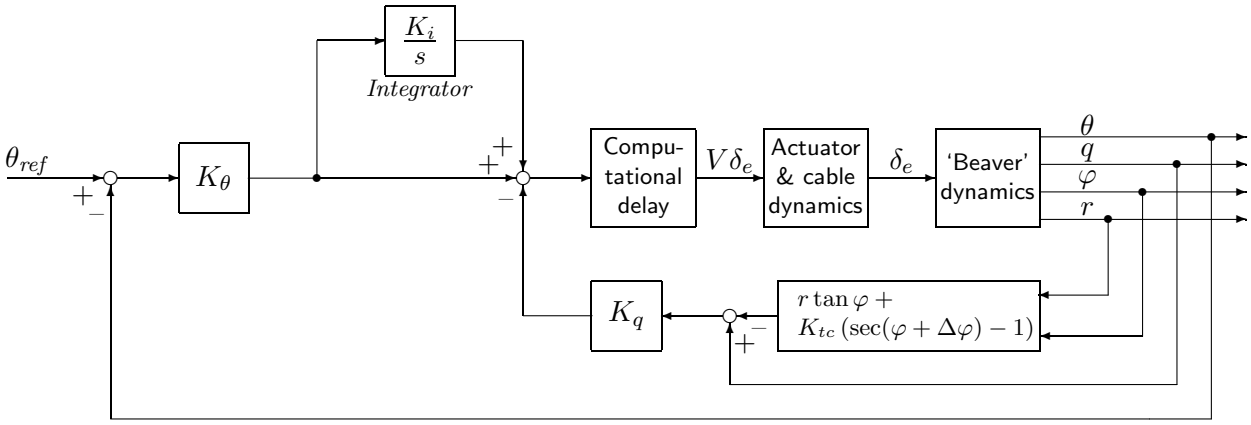


Figure 11.15: Block-diagram of the PAH mode with turn-compensation

- The ALH mode is designed to maintain a certain reference altitude. If the actual altitude differs from the desired value a pitch-up or pitch-down command is initiated by the outer-loop of the ALH control law. The turn-compensation for the ALH mode has therefore been optimized to minimize variations in altitude if the aircraft starts to roll. However, this implies that appropriate pitch-up or pitch-down commands have to be given which results in a non-constant value of the pitch angle.
- The PAH mode is designed to maintain a constant pitch angle. The turn-compensation for this mode helps to minimize variations in pitch angle response to rolling motions of the aircraft. If the compensation in PAH mode would have been equal to the ALH mode the initiation of turns would have yielded pitch-up commands which are desirable for maintaining a constant altitude, but undesirable for maintaining a constant pitch angle. In practice the PAH mode will be used separately only in combination with side-stick control, i.e. in fly-by-wire mode, where the pilot will generate the appropriate pitch commands to compensate for a loss of *altitude* while the PAH turn-compensation takes care of compensating for changes in pitch angle.

Non-linear simulations of the 'Beaver' aircraft revealed a noticeable asymmetrical behavior, which made it difficult to obtain satisfactory turn-compensation behavior for both right and left turns (corresponding with positive and negative values of the roll angle). For this reason, it was considered to use a small offset in φ, which would result in a somewhat different correction of the pitch rate in comparison to equation (11.9):

$$q = q_{tot} - (r\tan\varphi + K_{tc}\,(\sec(\varphi+\Delta\varphi)-1)) \tag{11.10}$$

where $\Delta\varphi$ is the offset in the roll angle. After introducing this offset, it turned out that any improvement in the *altitude* responses led to larger differences in *pitch angle* responses for left and right turns, and vice-versa. In the final autopilot system the offset value was therefore only applied for the ALH mode where some differences in pitch angle responses for right and left turns were considered to be tolerable, taking into account that this mode could not be controlled directly via a side-stick. For the PAH mode the best compromise between matching altitude responses and matching pitch angle responses turned out to be no offset at all. Table 11.3 at the end of this chapter shows the value of the offset-angle for the ALH mode as a function of the true airspeed. Figure 11.15 shows the resulting block-diagram of the Pitch Attitude Hold mode with turn-compensation, which also serves as inner-loop for other longitudinal autopilot

Signal	Lower boundary	Upper boundary
θ	-10 $[deg]$	+20 $[deg]$
φ	-45 $[deg]$	+45 $[deg]$
p	-12 $[deg\,s^{-1}]$	+12 $[deg\,s^{-1}]$
q	-12 $[deg\,s^{-1}]$	+12 $[deg\,s^{-1}]$
r	-12 $[deg\,s^{-1}]$	+12 $[deg\,s^{-1}]$

Table 11.1: Signal boundaries used by the mode controller

modes after adding a washout filter in the θ-loop (see section 11.3.2).

11.6 The signal limiters

For reasons of safety, the mode controller of the experimental 'Beaver' autopilot, used during the flight tests constantly checked the magnitude of the feedback signals. If these values would exceed certain upper or lower boundary values, the autopilot would automatically switch off. These boundary values have been listed in table 11.1. In order to make sure that the aircraft will never exceed these limits, the control laws have been equipped with signal limiters which were tuned by means of non-linear simulations of the PAH and RAH modes. In the flight tests these limitations proved to be sufficient to keep the aircraft within the safety margins from table 11.1. Apart from the signal limiters in the inner-loops and the safety checks in the mode controller, the output signals to the control surfaces were also limited, allowing only practically feasible control surface deflections. In the simulations, the outputs from the control laws lay always within the practical range of the control surfaces.

In the PAH mode and the inner loops of the ALH, ALS, GA, and GS modes the following limitations were necessary:

- The reference pitch angle θ_{ref} had to be limited to the maximum value from table 11.1 minus a small margin to make sure the autopilot would not switch off.
- The difference pitch angle $\theta_{ref} - \theta$ had to be limited to make sure that the pitch rate q remained between the upper and lower limits from table 11.1. This limiter was tuned by examining q-responses to large block-shaped pitch commands in PAH mode.
- The output signal from the integrator block had to be limited in order to prevent *integrator windup*. Without such a limitation, it is possible that the output signal from the integrator block becomes larger than the maximum feasible elevator deflection if the signal $\theta_{ref} - \theta$ has a value unequal to zero that remains either positive or negative during a large period of time. Without an anti-windup limiter, it can take a long time before the integrator is 'unloaded', which degrades the autopilot performance.

In the RAH mode and the inner loops of the HH, LOC, and NAV modes the following limitations were necessary:

- The reference roll angle was limited in order to prevent excessive loss of lift force and extreme values of the load factor in turns. Also the roll angle limitation made it possible to turn without increasing the engine power, except for turns at very low velocities. The resulting limits were tighter than the roll angle limits from table 11.1.

Signal	Lower limit	Upper limit
θ_{ref}	$-8°$	$+18°$
$\theta_{ref} - \theta$	$-2\ V$	$+2\ V$
Output of PAH integrator	$-10\ V$	$+10\ V$
φ_{ref}	$-30°$	$+30°$
$\varphi_{ref} - \varphi$	$-3.75\ V$	$+3.75\ V$
Output of RAH integrator	$-1\ V$	$+1\ V$

Table 11.2: Signal limiters from the PAH and RAH loops

- The difference $\varphi_{ref} - \varphi$ had to be limited in order to keep the roll rate p within the range from table 11.1. This limiter was tuned by examining p-responses to large block-shaped roll commands in RAH mode.
- The output from the integrator had to be limited in order to prevent integrator wind up. The output value had to be large enough to suppress the influence of the engine upon the lateral motions of the aircraft (the 'Beaver' exhibits a pronounced asymmetrical behavior in open-loop responses which needs to be suppressed by the RAH loops), but it had to be limited to get a reasonably small overshoot when the reference *heading* in Heading Select mode was reached. Without the anti-windup integrator, the overshoots would have been unacceptably large due to the considerable time it would then take to 'unload' the integrator.

It was not necessary to include additional signal limiters to the outer loops, because all autopilot modes make use of the same inner-loops. Therefore, if the aircraft remains within the safe region defined by table 11.1 for PAH and RAH modes, it can never exceed these limitations in other modes whose outer-loops effectively create command signals for the PAH and RAH inner-loops. The resulting limiter values for the 'Beaver' autopilot are listed in table 11.2. The values of the reference angles are measured in degrees, while the other signals are converted to Volts. This is due to the fact that the actuator deflections are functions of input Voltages. The maximum allowable values of the command signals to the actuators are plus or minus 10 Volts, due to physical limitations of the control system hardware. The resulting maximum and minimum deflections of the elevator, rudder, and ailerons are smaller than the deflections which can be applied by a human pilot.

Longitudinal Autopilot Modes:

PAH	K_θ	=	$-0.001375\,V^2 + 0.1575\,V - 4.8031$	$[V\ deg^{-1}]$
	K_q	=	$-0.000475\,V^2 + 0.0540\,V - 1.5931$	$[Vs\ deg^{-1}]$
	K_i	=	0.5	$[s^{-1}]$
	K_{tc}	=	$0.02865\,V^2 - 1.7189\,V + 53.7148$	$[V]$
	$\Delta\varphi$	=	0	$[deg]$
ALH	K_H	=	$-0.00010\,V^2 + 0.015\,V - 0.5975$	$[Vm^{-1}]$
	K_d	=	$-0.0025\,V + 0.2875$	$[s^{-1}]$
	K_{tc}	=	$1.7189\,V + 14.3239$	$[V]$
	$\Delta\varphi_{tc}$	=	2	$[deg]$
	Other gains & coefficients: see PAH			
ALS	$K_{\dot{H}}$	=	$-0.0003875\,V^2 + 0.04025\,V - 1.1041$	$[Vs\,m^{-1}]$
	K_c	=	0.15	$[s^{-1}]$
	K_d	=	$-0.0025\,V + 0.2875$	$[s^{-1}]$
	Other gains & coefficients: see PAH			
GS	$K_{\varepsilon_{gs}}$	=	$(0.00575\,V^2 - 0.63\,V + 18.00625)\left(\frac{H-H_{RW}}{\sin\lvert\gamma_{gs}\rvert} + x_{gs}\right)\frac{1}{1000}$	$[V\ deg^{-1}]$
	K_d	=	$-0.0025\,V + 0.2875$	$[s^{-1}]$
	K_r	=	1	$[\,-\,]$
	Other gains & coefficients: see PAH			

Lateral Autopilot Modes:

RAH	K_φ	=	$0.000975\,V^2 - 0.108\,V + 2.335625$	$[V\ deg^{-1}]$
	K_i	=	0.25	$[s^{-1}]$
	K_r	=	-4	$[Vs\ deg^{-1}]$
	dar	=	0.165	$[Vs\,m^{-1}]$
	drr	=	$-0.000075\,V^2 + 0.0095\,V - 0.4606$	$[Vs\ deg^{-1}]$
HH	K_ψ	=	$0.05\,V - 1.1$	$[deg\ deg^{-1}]$
	Other gains & coefficients: see RAH			
LOC	$K_{\Gamma_{loc}}$	=	$(0.00775\,V^2 - 0.76\,V + 15.75625)\left(\frac{H-H_{RW}}{\sin\lvert\gamma_{gs}\rvert} + x_{loc}\right)\frac{1}{1000}$	$[deg\ deg^{-1}]$
	K_v	=	10	$[\,-\,]$
	Other gains & coefficients: see RAH			
NAV	$K_{\Gamma_{VOR}}$	=	$0.05375\,V^2 - 6.825\,V + 153.03125$	$[deg\ deg^{-1}]$
	K_ψ	=	$0.05\,V - 1.1$	$[deg\ deg^{-1}]$
	K_l	=	1	$[s^{-1}]$
	K_p	=	15	$[s]$
	Other gains & coefficients: see RAH			

Table 11.3: Gain-factors and coefficients of the control laws as a function of airspeed and altitude

Chapter 12

'Beaver' autopilot – implementation in FDC 1.2

12.1 Introduction

There are several ways to implement control laws of an Automatic Flight Control System in SIMULINK. In practice there will be a gradual increase in model complexity during the AFCS design process. In this chapter two different implementation methods will be described. The first method is to obtain a literal 'translation' of the block-diagrams from chapter 11 in SIMULINK. Due to the straightforward manner of implementing the control laws, this method is very suitable for quick evaluations, but the resulting SIMULINK systems are rather inflexible and do not match the software structure needed for actual application in a Flight Control Computer. This method will be demonstrated for the basic control modes of the 'Beaver' autopilot in section 12.2. The second method integrates all control laws in one subsystem, yielding a structure like figure 3.1, which makes it easier to convert graphical block-diagrams to a high-level programming language for use in a real-time flight-simulator or in the FCC's of the actual aircraft. In section 12.3 this second method will be demonstrated for the complete 'Beaver' autopilot.

12.2 Implementing separate control laws in SIMULINK

12.2.1 Structure of the control-law simulation models

The first steps in a control system design usually require application of *linear* control system design tools such as the control system design toolboxes of MATLAB. The linear results must be validated by means of non-linear simulations, which should take place directly after finishing the linear system design in order to provide direct feed-back of possible errors in the design at an early stage of the design process. Within the MATLAB environment this can be achieved by converting the original block-diagrams of the control laws, e.g. the block-diagrams from chapter 11, into corresponding graphical SIMULINK models.

During this conversion it is important to remember that the linear system analysis is based upon *small perturbations* of signals around their nominal values, while the non-linear analysis is based upon the true values of those signals. Consider for instance simulations of a pitch-attitude control law. With a linear small-perturbations model such simulations are straightforward, because all output signals from the control laws represent deviations from the nominal elevator deflections, while the signals obtained from the linear aircraft model correspond to deviations from the initial flight condition. For this reason the desired change in pitch angle may be entered *directly* into the pitch controller when using a linearized aircraft model. With

a non-linear model this signal must be superimposed upon the initial value of the pitch angle before it can be compared with the true value of the pitch angle extracted from the aircraft model. Moreover, the change in elevator deflection according to the pitch controller must be added to the initial elevator deflection before it is sent to the non-linear aircraft model. This initial value is in general not equal to zero! In general, for a non-linear simulation it is necessary to add the outputs from control laws to the initial values of the control inputs before they may be entered into the aircraft model, while the initial values of the outputs from the aircraft model must be subtracted from the current outputs before sending them to the control laws.

12.2.2 SIMULINK implementation of the Pitch Attitude Hold mode

Figure 12.1 shows the SIMULINK implementation of the Pitch Attitude Hold mode of the 'Beaver' autopilot, which was obtained by means of a 'literal translation' of the PAH block-diagram from figure 11.3. This SIMULINK system is called PAH. Figure 12.2 shows the signal manipulations required to convert the inputs and outputs from the non-linear aircraft model to small perturbation signals for the PAH control law and vice versa, as implemented within the subsystem Beaver dynamics. The core of this subsystem is an S-function block which calls the system Beaver (see chapter 5).

The aircraft model has twelve input signals, of which the first four are aerodynamic control inputs, the fifth and the sixth signals are engine inputs, and the others are wind and turbulence inputs. These inputs are gathered in the vectors $\mathbf{u}_{aero}$, $\mathbf{u}_{prop}$, and $\mathbf{u}_{wind}$, respectively. The first element of $\mathbf{u}_{aero}$ represents the elevator deflection δ_e, which is equal to the initial value $\delta_e(0)$ *plus* the computed change in elevator deflection $\Delta\delta_e$, obtained from the PAH control law. Since the other aerodynamic and engine control inputs are not used by the PAH loop, they remain equal to their initial values. Moreover, the wind and turbulence inputs are zero in order to simulate a no-wind condition. Thus, on the input side of the S-function block we add a vector $[\,\Delta\delta_e\,,\,0\,,\,0\,,\,0\,]^T$ to the *initial* aerodynamic input vector $\mathbf{u}_{aero}(0)$, called *uaero0* in figure 12.2. This result is *Muxed* with the initial value of the engine input vector $\mathbf{u}_{prop}(0)$, called *uprop0* in figure 12.2, and the atmospheric disturbance vector $\mathbf{u}_{wind}$, which equals zero in figure 12.2 due to no wind. This results in the S-function input vector for the system Beaver, see equation (10.2) from chapter 10.

On the output side of the S-function block the output vector is first *Demuxed* in two parts: the state-vector $\mathbf{x}$ and a vector containing the remaining output signals. The initial value of the state vector $\mathbf{x}_0$ is subtracted from $\mathbf{x}$ to obtain the deviations from the initial flight-condition for the PAH control law. A second Demux block divides the resulting deviations vector into twelve scalar signals, of which the PAH mode only needs the fifth and seventh elements, corresponding to $\Delta\theta = \theta - \theta_0$ and $\Delta q = q - q_0$ respectively. The Outport and Inport blocks in the subsystem Beaver dynamics transfer the signals $\Delta\delta_e$, $\Delta\theta$, and Δq to/from the top-level of the system PAH.

12.2.3 SIMULINK implementation of the Roll Attitude Hold mode

In a similar way the Roll Attitude Hold mode and turn-coordinator of the 'Beaver' autopilot were implemented in SIMULINK. Figure 12.3 shows the SIMULINK equivalent of the block-diagram from figure 11.8, called RAH. The block Beaver Dynamics again is a subsystem which now contains signal manipulations for the RAH mode, shown in figure 12.4.

In figure 12.5 the Pitch and the Roll Attitude Hold modes have been combined in one block-diagram. This scheme also includes the turn-compensation loop in the pitch-channel, which was described in section 11.5. This SIMULINK model is called PAHRAH. It is obvious that the model structure in this case already is starting to become complicated, even though this diagram is still limited to the two basic modes of the 'Beaver' autopilot. Adding the guidance

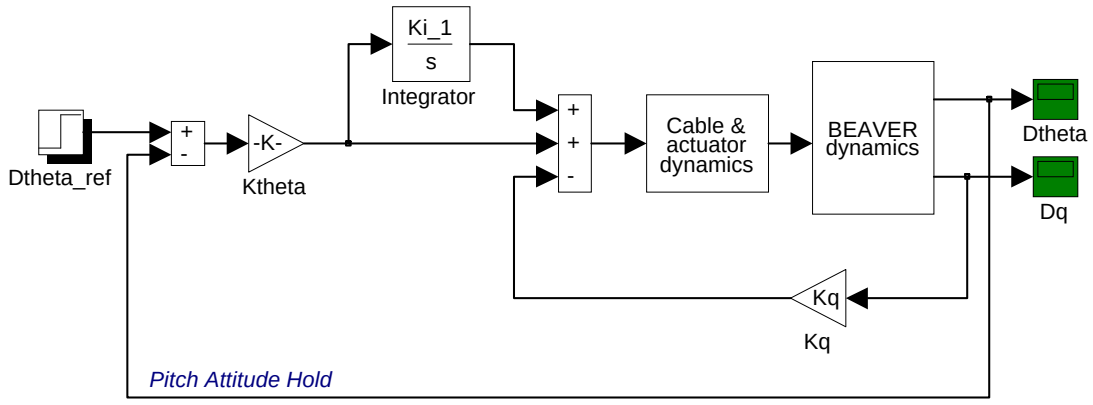


Figure 12.1: Block-diagram representation of the PAH mode in SIMULINK

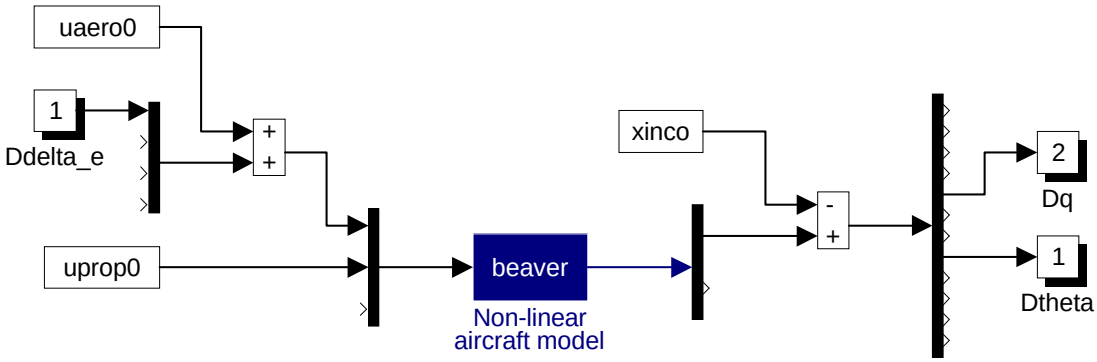


Figure 12.2: Signal manipulations for the PAH mode in a non-linear simulation model

loops (Altitude Hold, Altitude Select, Heading Hold) to this structure is feasible when more parts of the system are 'grouped' together in subsystems, but it is hard to keep the complexity of the system under control. Remember that the system from figure 12.5 does not yet contain signal limiters, continuous gain-scheduling functions, feedback of the pitch and roll rates to the actuator & cable models, and turn-compensation loops! Implementation of the radio-navigation modes (Glideslope, Localizer, and Navigation) will be even more complicated because then additional models for the generation of the radio signals are needed. For this reason, 'literal translations' of the block-diagrams from chapter 11 are only usable for the *basic* autopilot modes. The next steps in the control-law design process require a better structured simulation model, like the model from section 12.3. For a smooth transition between linear design and non-linear analysis it is recommended to use both kinds of models: the simplified models of the basic modes already yield useful results (as will be demonstrated in the next section for the PAH and RAH modes of the 'Beaver' autopilot), while the sophisticated simulation structure from later design phases is especially useful for the transition to full flight simulation and the FCC's of the real aircraft. The next section describes how to apply the block-diagrams from figures 12.1, 12.3, and 12.5 in practice for non-linear simulations.

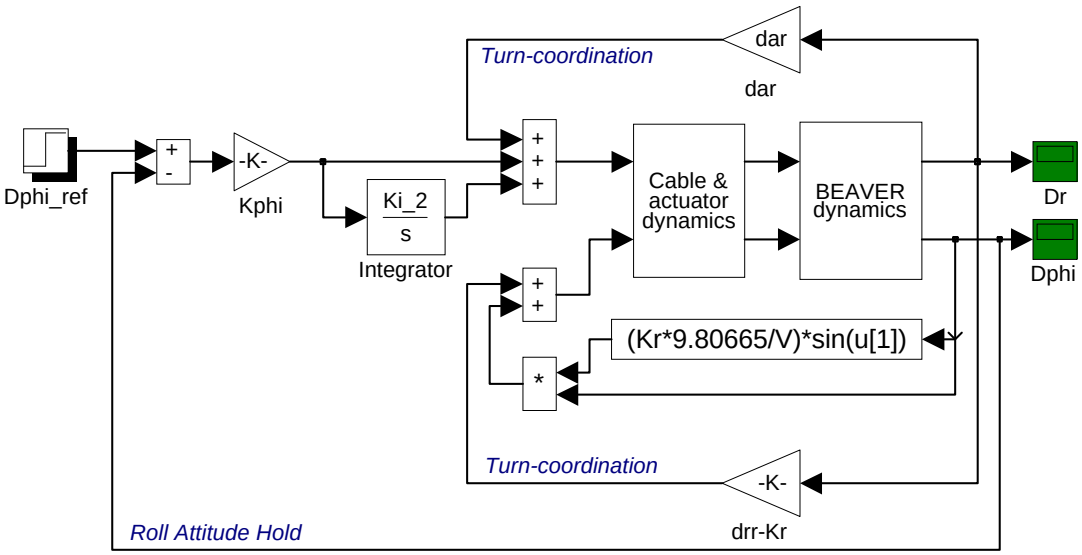


Figure 12.3: Block-diagram representation of the RAH mode in SIMULINK

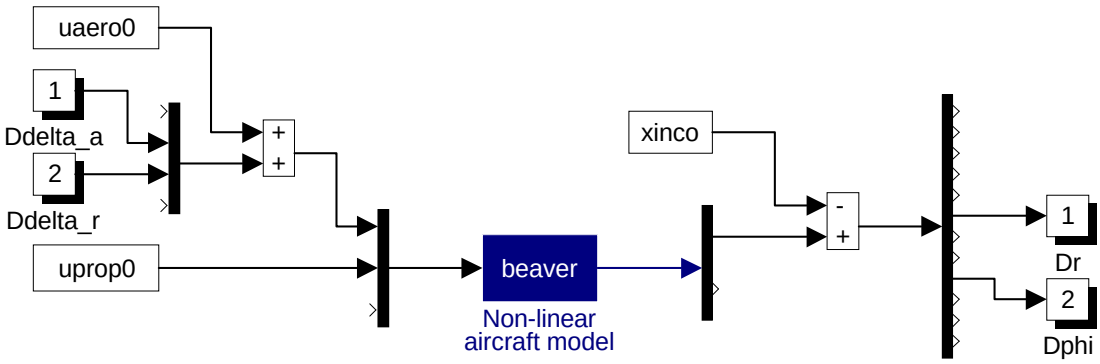


Figure 12.4: Signal manipulations for the RAH mode in a non-linear simulation model

12.2.4 Using the PAH and RAH simulation models in practice

The SIMULINK systems PAH, RAH, and PAHRAH can be opened by typing `pah`, `rah`, or `pahrah` at the MATLAB command-line. It is also possible to use the MATLAB macro APMENU, which opens up a user-menu from which to choose the appropriate autopilot simulation model. Before starting a simulation with these systems it is necessary to initialize the system parameters and initial conditions, which is simplified by means of the MATLAB macro PRAHINIT. This routine can be started by typing `prahinit` at the command-line, or by double-clicking the magenta initialization button at the bottom of the block-diagrams. Figure 12.6 shows the user menu from PRAHINIT. All menu items should be clicked in the indicated order, and all questions which appear in the MATLAB command-window should be answered. Item 5 ('Fix states') is optional; it can be skipped if one does not want to artificially fix state variables of the aircraft model to their initial values during a simulation.

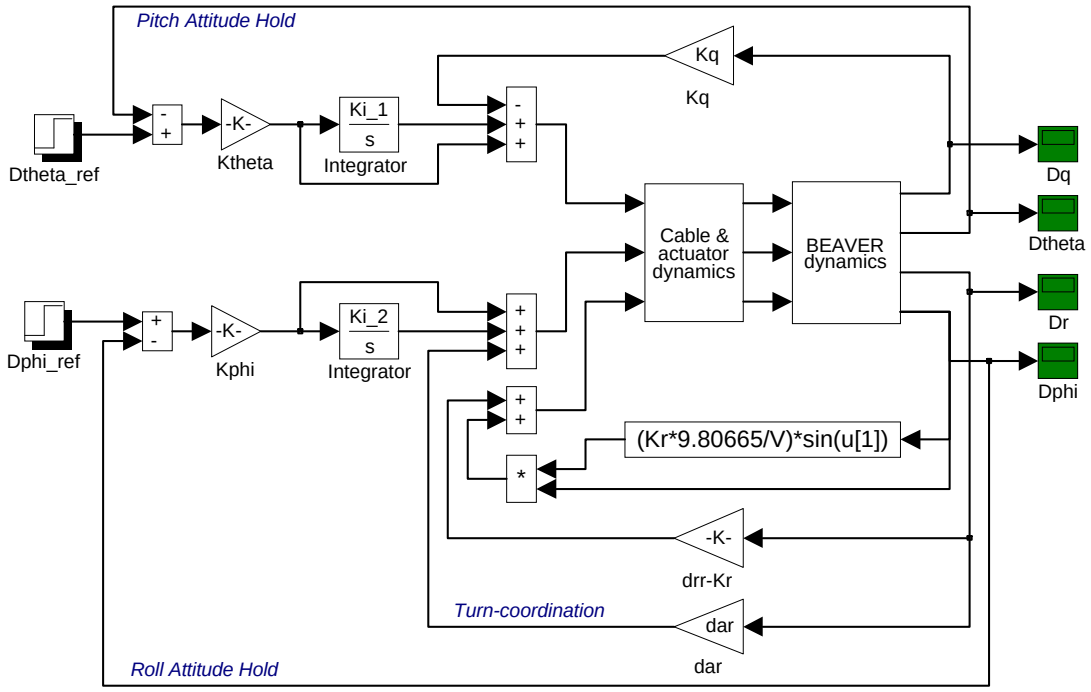


Figure 12.5: Block-diagram representation of the PAH *and* RAH modes in SIMULINK

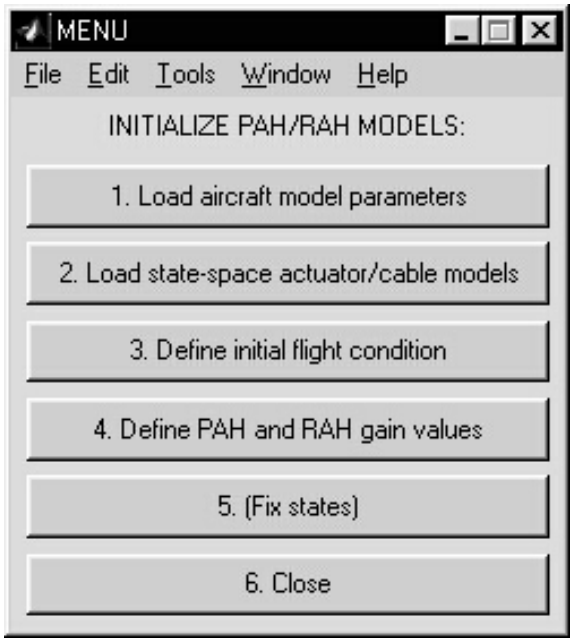


Figure 12.6: Initialization menu for the systems PAH, RAH, and PAHRAH

The system initialization thus requires the following steps to be taken:

1. First, the parameters for the non-linear aircraft model must be loaded from the file AIRCRAFT.DAT. This is done by clicking item number 1, which will start the load routine LOADER. You will be asked to specify the directory in which this file can be found; usually the default directory will be correct. If the load routine can't find AIRCRAFT.DAT it will ask whether or not it should run MODBUILD to create the file. This question will also be asked if you specify a wrong directory, so before running MODBUILD you should check the directory name again. If you return to the main menu without AIRCRAFT.DAT having been loaded the model parameters will not be present in the workspace, so you will encounter an error message if you still try to start a simulation.
2. The second item in the main menu must be clicked to load the state-space matrices of the cable & actuator models for the elevator, ailerons, and rudder. There are three models available, valid for an airspeed of 35, 45, or 55 ms^{-1}. Choose the one that lies most closely to the airspeed for which you want to make a simulation (variations in actuator & cable parameters due to changes in the airspeed *during* simulations will not be taken into account). The actuator model for $V = 45\ ms^{-1}$ is usually good enough.
3. The third item in the main menu must be clicked to define the initial value of the input and output vectors of the non-linear aircraft model. There are two options: to run the aircraft trim-routine ACTRIM or to load the initial flight condition from file by means of the MATLAB macro INCOLOAD. For a description of the trim-routine, consult section 8.2. If you select the second option, a menu will be shown for asking you what to load (choose *load trimmed flight condition*), and you will be asked to specify the directory name (again the default directory will usually be right), the filename (e.g. CR4520 for a symmetrical steady-state trimmed flight condition at an airspeed of 45 ms^{-1} and an altitude of 2000 ft), and the extension (use the default extension .TRI for a trimmed-flight condition obtained with ACTRIM).
4. The fourth item must be clicked to define the gain values for the Pitch and Roll Attitude Hold modes. Since the systems PAH, RAH, and PAHRAH do not contain continuous gain-scheduling functions, it is necessary to enter a value for the airspeed for which the gains are determined. This value of the airspeed should be chosen as close as possible to the mean value of the airspeed anticipated for the simulation. Usually, quite accurate results can be obtained by selecting the *initial* airspeed as reference value for the gains.
5. Finally, the fifth item can be clicked if for one reason or another it is necessary to artificially fix some state variables of the aircraft model to their initial values. This option will be used later on in an example. See also section 9.5. Under normal circumstances item 5 can be skipped.

After having defined all system parameters, it is possible to define the reference input signals. In the systems PAH, RAH, and PAHRAH the default pitch-command equals a 1° step-input, while the default roll-command is equal to a 10° step-input. Double-clicking the step-signal blocks on the left hand side of the systems makes it possible to change the magnitude of these signals.[1] Replacing the step-signal blocks by other kinds of input-signal generators makes it possible to change the shape of the input signals if desired.

[1] The factor $\pi/180$ in these blocks is used to transform all angles from degrees to radians.

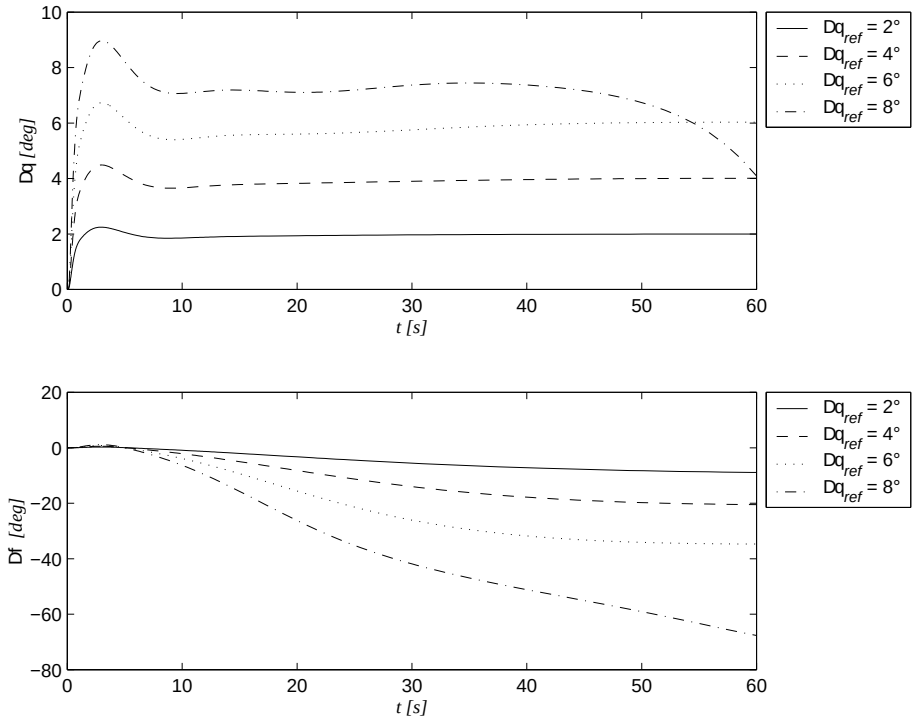


Figure 12.7: Pitch and roll-responses without wing-leveler

Examples:

Use the system PAH to perform simulations of block-shaped pitch commands, ranging from 1° to 8°. First run PRAHINIT, then enter the desired reference pitch angle in the input block on the left hand side of the system PAH. Do not use item 5 of the initialization menu yet. Increasing the reference step in pitch angle eventually yields an unstable pitch-response, which is caused by longitudinal-lateral cross-coupling effects in combination with a large decrease in airspeed (the engine power remains constant). This is shown in figure 12.7. For large *pitch*-commands, the *roll* angle will increase considerably, bringing the aircraft in a spiral dive. (The results of each simulation can be plotted by means of the MATLAB macros RESULTS and RESPLOT, which should be run in this particular order; see sections 9.4.1 and 9.4.2 for more details.) Now re-run PRAHINIT, this time selecting item 5 in the user-menu to artificially fix the asymmetrical motion variables to their initial values. This time the instability in the pitch-response will not occur, even for much larger reference pitch commands. This proves that the increasing roll angle was indeed the cause of the instability in the pitch angle. Now repeat these simulations with the system PAHRAH, using a reference roll-command of 0° to suppress the increase of the roll angle. Run item 5 of PRAHINIT again and select the option '*Don't fix any states*' in the user-menu in order to allow the asymmetrical state variables to vary freely this time. Now the instability in the pitch response will be suppressed by the Roll Attitude Hold mode which serves as a 'wing leveler', keeping the roll angle equal to zero. Figure 12.8 presents the results. These simulations make clear that in practice the two basic modes of the autopilot should not be applied separately to avoid dangerous situations. Of course there is no operational need for separate use of either the symmetrical or asymmetrical autopilot modes, but this analysis is still useful since it provides more insight in the overall dynamics of the system and the validity of the linear results over a large range of pitch commands.

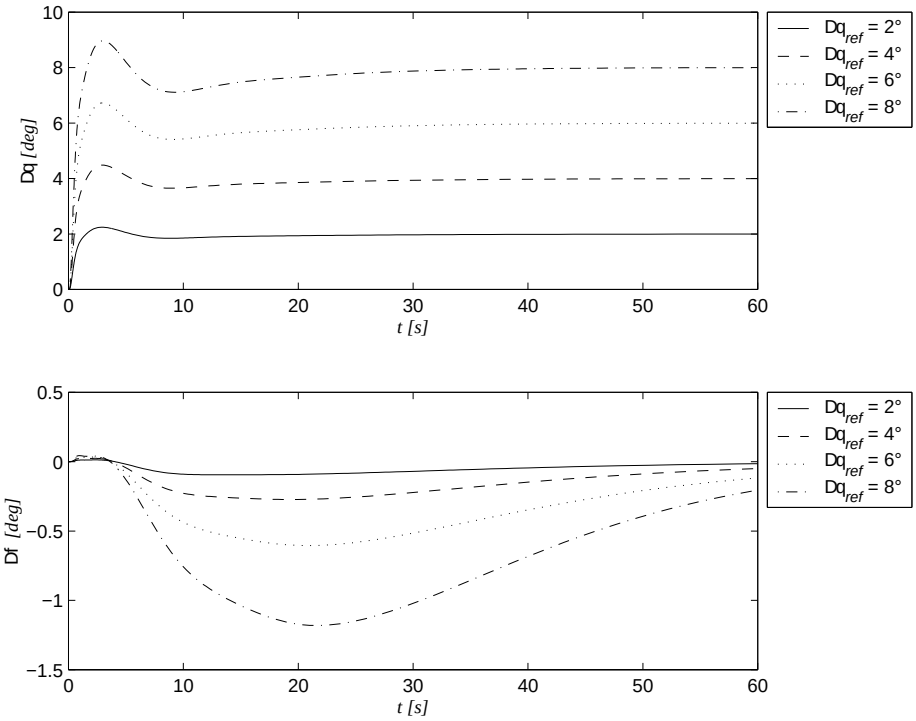


Figure 12.8: Pitch and roll-responses with wing-leveler

12.3 Integral autopilot simulation model

12.3.1 General structure of the autopilot simulation model

While the basic modes of the 'Beaver' autopilot could be implemented quite easily in SIMULINK systems that resembled the block-diagrams from chapter 11, it is quite difficult to maintain a clear structure of the simulation model if the outer loops are also implemented in this way. This is especially true when items such as on-line gain-scheduling, signal limiters, turn-compensation, atmospheric turbulence, and radio-navigation signals are to be implemented. Also, conversion from the simulation models to real-time software routines is quite difficult to achieve with the model structure from section 12.2. The implementation of the *complete* 'Beaver' autopilot therefore has been based upon the model structure from figure 3.1, which combines all control logic into one subsystem in stead of constructing the control laws 'around' the aircraft model like the block-diagrams from chapter 11. Reconstruction of these block-diagrams required a repositioning of the different elements to obtain a clear input/output structure for the control logic. A division between symmetrical and asymmetrical control laws was made, and the control function of the inner-loops was separated from the guidance task of the outer-loops.

The systems APILOT1, APILOT2, and APILOT3 contain the resulting SIMULINK implementation of the 'Beaver' autopilot model. APILOT3 is the most comprehensive version, APILOT2 is a simplified model that does not take into account external atmospheric disturbances and noise in radio-navigation signals, and APILOT1 contains neither atmospheric disturbances, nor radio-navigation blocks. In this section APILOT3 will be used as a guideline to explain the general structure of the autopilot simulation model, but APILOT1 and APILOT2 are better suited for practical purposes, because they require less computing time and memory due to the larger feasible step-size for the numerical integrations. For on-line help about these models, enter `type apilot.hlp` at the MATLAB command-line.

Figure 12.9 shows the top-level of APILOT3; compare this with figure 3.1. This system can be opened by typing `apilot3` at the command-line (type `apilot1` or `apilot2` to open the simplified models). The models can also be accessed by means of the MATLAB macro APMENU, which reveals a graphical user-menu for choosing the appropriate autopilot model. The the top-level of APILOT3 contains the following subsystem blocks:

- Beaver Dynamics links the non-linear aircraft model Beaver to the autopilot simulation model by means of an S-function block,
- Symmetrical autopilot modes and Asymmetrical autopilot modes contain the control laws,
- Actuator & cable dynamics contains linear state-space models of the dynamics of the actuators and the cables from the actuators to the control surfaces, as used in the 'Beaver' test aircraft,
- Computational delay & limiters takes into account the computational delay in the evaluation of the control laws and the input limitations of the actuators,
- Mode Controller and Reference Signals define switch-settings and reference values used by the control laws,
- Wind & Turbulence determines the components of wind and atmospheric turbulence in the aircraft's body-axes, along with the time-derivatives of these values,
- VOR and ILS determine radio-navigation signals for the navigation and approach modes, using the models from chapter 7,
- Sensors gathers other sensor characteristics and is used to subtract the initial conditions from the S-function outputs that leave the system Beaver (this is necessary, because the autopilot control laws are based upon *deviations* from the initial values of the S-function outputs while the aircraft model itself uses the full signals),
- Add initial inputs is used to add the initial values of the control inputs to the changes in control surface deflections according the control laws (again: the aircraft model is based upon the full signals; the control laws are based upon deviations from the initial values).

Notice that only three control inputs have been coupled to the system Beaver; the autopilot does not manipulate the engine inputs and flap setting. But APILOT3 already has been prepared to accommodate possible autopilot enhancements which do control those variables, e.g. an autothrottle system that controls the engine in order to maintain a constant airspeed.

For on-line help about the autopilot models, enter `type apilot.hlp`. Information about the color scheme from these models can be found in COLORS.HLP (type `type colors.hlp` at the command-line).

12.3.2 Implementation of the symmetrical autopilot modes

Figure 12.10 shows the internal structure of the block Symmetrical Autopilot Modes. The control laws themselves have been implemented separately in the subsystems ALH, ALS, GS, and PAH, which correspond with Altitude Hold, Altitude Select, Glideslope Coupled, and Pitch Attitude Hold modes respectively. The PAH mode serves both as an independent autopilot mode and as *inner-loop* controller for the ALH, ALS, and GS modes. The additional washout-filter which was needed in the θ-loop for the ALH, ALS, and GS modes (see sections 11.3.2, 11.3.3, and 11.3.4) has been included in the subsystem PAH with a switch to de-activate it for the PAH mode itself. The gain-block 1/Ktheta is a correction needed to use the PAH control law as inner-loop,

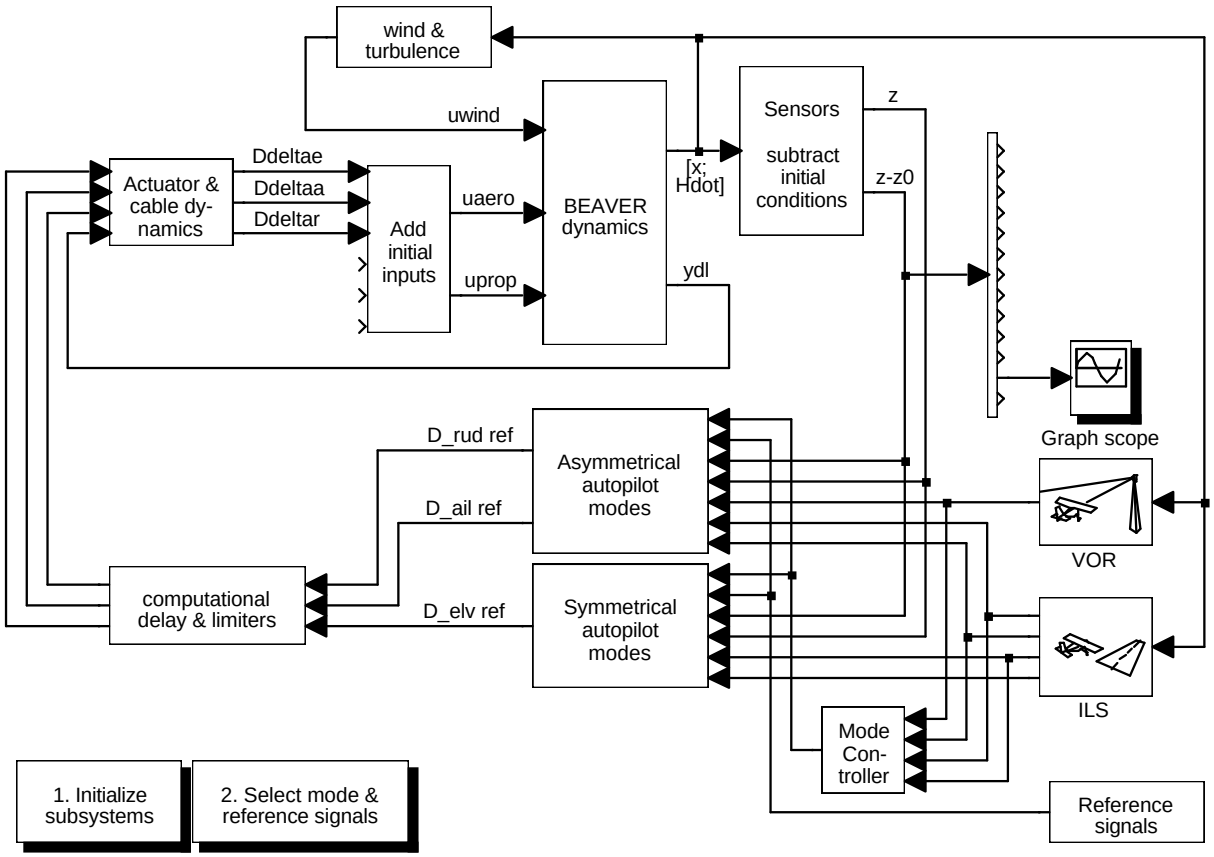


Figure 12.9: Top-level of the autopilot simulation model

taking into account the different position of the gain K_θ in the ALH, ALS, and GS modes compared to the PAH mode itself (compare figures 11.4, 11.5, and 11.6 with figure 11.3). In figure 12.10 the results from the blocks ALH, ALS, and GS are added together; switches inside these blocks determine whether the outputs from these blocks are taken into account or not (if a control mode is not active its output is set to zero). The settings of all switches are determined by the subsystem Mode Controller in the top-level of APILOT3 (see section 12.3.4); the block Select symm. mode-controller signals extracts the relevant switch-settings for the symmetrical autopilot modes. It also extracts the reference input signals for the symmetrical modes, which are determined by the block Reference signals in the top-level of APILOT3. The block Select symm. control variables extracts the variables used by the symmetrical control laws to determine the change in elevator deflection from the output vectors from the aircraft model, $\mathbf{z}$ and $\mathbf{z} - \mathbf{z}_0$, including signals for the gain-scheduling functions. See section 12.3.6 for the definitions of these vectors.

12.3.3 Implementation of the asymmetrical autopilot modes

The general structure of the block Asymmetrical Autopilot Modes from the top-level of APILOT3 is shown in figure 12.11. The individual asymmetrical control laws have been implemented in the subsystems HH, NAV, LOC, and RAH, which correspond to the Heading Hold, Navigation, Localizer, and Roll Attitude Hold modes, respectively. The RAH mode serves both as independent autopilot mode and as the inner-loop controller for the HH, NAV and LOC modes. The outputs from the blocks HH, NAV, and LOC are added together; switches inside these blocks

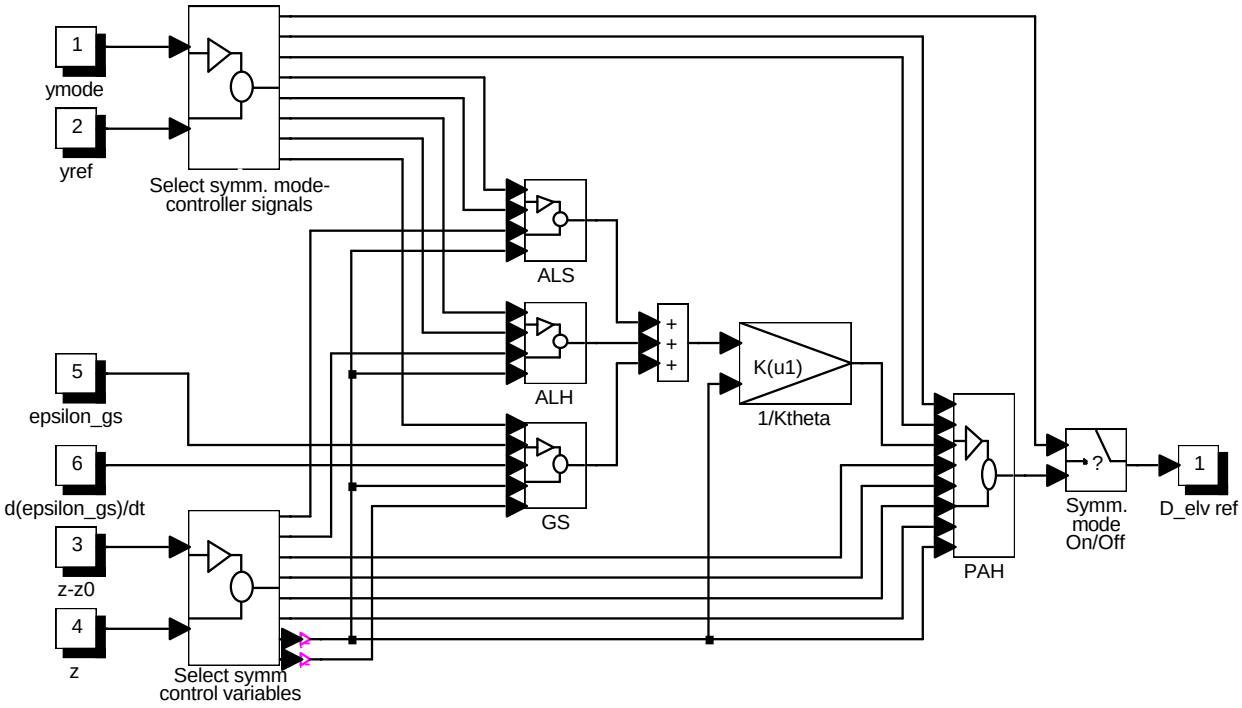


Figure 12.10: Implementation of the symmetrical autopilot modes

determine whether or not the outputs from these blocks are taken into account. The settings of all switches for the asymmetrical modes are extracted from the block Mode Controller (see section 12.3.4) by the block Select asymm. mode-controller signals, which also extracts the reference input signals for the asymmetrical modes from the outputs of Reference signals. The block Select asymm. control variables extracts the variables used by the asymmetrical control laws to determine the change in aileron and rudder deflection from the vectors $\mathbf{z}$ and $\mathbf{z} - \mathbf{z}_0$, including the signals for the gain-scheduling functions. See section 12.3.6 for the definition of the vector $\mathbf{z}$.

12.3.4 Implementation of the Mode Controller

Figure 12.12 shows the subsystem Mode Controller. It contains switch-criteria for the Glideslope, Localizer, and Navigation modes, as described in sections 11.3.4, 11.4.3, and 11.4.4. The switch-criteria themselves are contained in the MATLAB subroutines GSSWITCH, LOCSWTCH, and NAVSWTCH, respectively. The source-codes of these routines, which can be found in the FDC subdirectory APILOT, are self-explaining; compare them with the theoretical description of the switch-criteria from chapter 11. Type `help gsswitch`, `help locswtch`, or `help navswtch` at the command-line for on-line help about these subroutines.

The output signal from the subsystem Mode Controller is the vector $\mathbf{y}_{mode}$, which has ten elements. The first five elements of this vector control the settings of the switches from the subsystem Symmetrical autopilot modes. This part of $\mathbf{y}_{mode}$ is equal to the MATLAB variable *ymod1S* or *ymod2S*, depending upon the desired symmetrical autopilot mode and the output from the switch-function GSSWITCH. The last five elements of $\mathbf{y}_{mode}$ define the switch-settings for the subsystem Asymmetrical autopilot modes. This part of the vector is equal to the variable *ymod1A* or *ymod2A*, depending upon the asymmetrical autopilot mode and the output from the switch-function LOCSWTCH or NAVSWTCH. The variables *ymod1S*, *ymod2S*, *ymod1A*, and *ymod2A* are obtained from the MATLAB workspace by means of Constant blocks. They can be

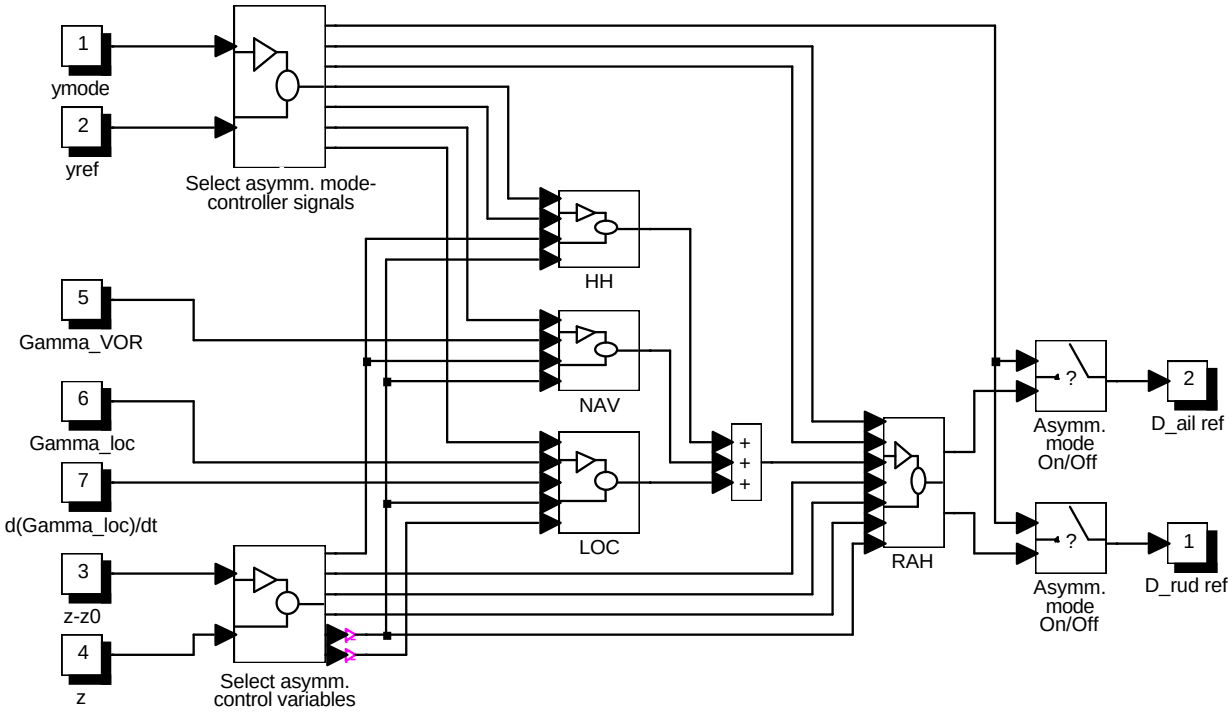


Figure 12.11: Implementation of the asymmetrical autopilot modes

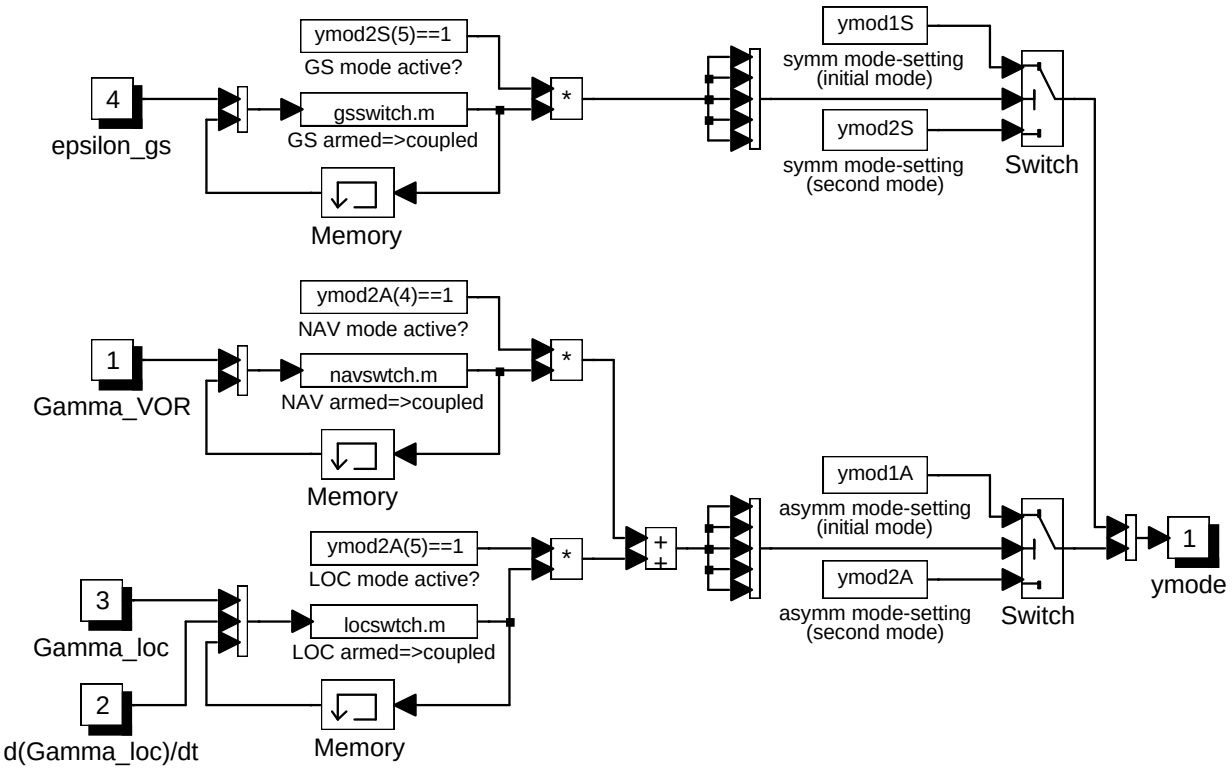


Figure 12.12: Implementation of the Mode Controller

set manually or by running the MATLAB routine APMODE (type `apmode` at the command-line or double-click the block Select mode & reference signals). The extensions S and A in the names of these variables refer to symmetrical and asymmetrical autopilot modes, respectively. The numbers *1* and *2* refer to an initial and a final mode-definition, which is used for defining *Armed* and *Coupled* phases in control modes, respectively. See section 12.4.1 for the exact definitions of these variables.

The signs of the output signals from the MATLAB functions GSSWITCH, LOCSWTCH, and NAVSWTCH determine whether the corresponding control laws are in Armed or Coupled phase. Consider for example the glideslope switch-function GSSWITCH. As long as its output value is positive, the corresponding Switch block on the right-hand side of the system Mode Controller will pass-through the symmetrical mode-setting vector for the Armed phase, *ymod1S*. If the output from GSSWITCH becomes negative, the mode-setting vector for the Coupled phase, *ymod2S*, will be passed instead. In order to ensure that the glideslope mode will not switch back from Coupled to Armed, a Memory block is used to store the previous output from GSSWITCH. The same technique has been applied for the Localizer and Navigation modes, using the routines LOCSWTCH and NAVSWTCH, respectively. Notice that while this switching method makes it possible to switch between an *initial* and *second* phase of the symmetrical and asymmetrical control modes, it does not allow more than one mode-switching action per simulation run. Of course it is possible to extend the system Mode Controller with other switches, using new variables such as *ymod3S*, *ymod4S*, etc. but this will always limit the number of switching actions to a *finite* value. For simulations of the 'Beaver' autopilot one switching action for the symmetrical and asymmetrical modes is sufficient. Mode Controller does not take into account mode-selections by the pilot; it only represents the *automatic* switching actions from the Mode Controller software in the real aircraft.

Although the switch-criteria functions are evaluated regardless of which control mode is active, their outputs are only passed through if the corresponding control mode is actually selected by the user at the start of a simulation. If the control mode is not selected, the outputs from the switch-functions are multiplied by zero, which causes the Switch blocks on the right hand side of the subsystem Mode Controller to pass through the first mode-setting vector (being *ymod1S* or *ymod1A*). This is the only mode-setting vector for all modes which don't have a separate Armed phase; for those modes the variables *ymod2S* and *ymod2A* are treated as dummy variables. In the subsystem Mode Controller this has been achieved by multiplying the outputs from GSSWITCH, LOCSWTCH, and NAVSWTCH with the output from a logical function that is zero when the corresponding control law is not active.

12.3.5 Implementation of atmospheric disturbances

Figure 12.13 shows the structure of the block Wind & Turbulence from APILOT3. It contains a model of atmospheric turbulence, based upon Dryden filters with velocity-dependent coefficients, and a wind-profile for the Earth's boundary layer. These subsystems (turb2 and BLwind) can both be found in the wind and turbulence library WINDLIB, which has been described in chapter 6. The time-derivative block du/dt is used to obtain time-derivatives of the body-axes wind-velocity components, extracted from BLwind, which are needed as inputs to the equations of motion. The time-derivatives of the velocity components due to atmospheric turbulence are computed within the subsystem turb2. See section B.3 in appendix B for more details about the equations of motion in non-steady atmosphere.

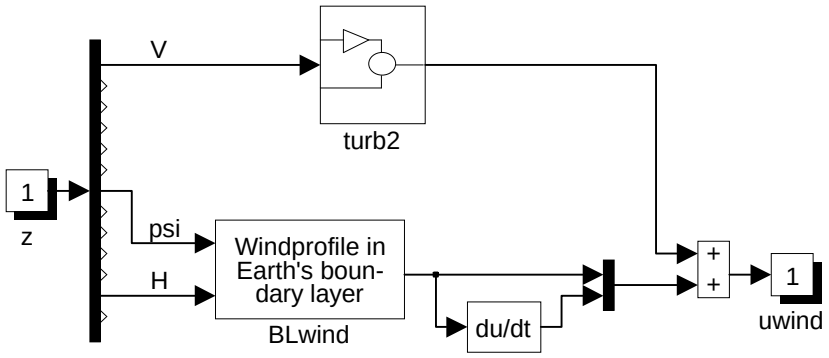


Figure 12.13: Internal structure of the subsystem Wind & Turbulence

12.3.6 Blocks to obtain small-deviation signals from the aircraft model

Due to the fact that the control laws are based upon small deviation models of the aircraft motions some signal manipulations are needed on the input and output sides of the non-linear aircraft model. On the input side the initial values of the control surface deflections must be added to the deviations of these values according to the control laws. This is done in the masked subsystem Add initial inputs, whose contents are displayed in figure 12.14. The three unconnected input ports correspond with the flap deflection, the engine RPM, and the manifold pressure of the engine. The current version of the autopilot does not use these potential control inputs to the aircraft model, although it is theoretically possible to do so. For instance, the engine inputs can be manipulated by an autothrottle system, and the flap extension can be used for gust-alleviation purposes.

On the output side of the aircraft model, the difference between the initial and current value of the output vector $\mathbf{z}$ is determined in the subsystem Sensors/Subtract initial conditions. The vector $\mathbf{z}$ is defined as:

$$\mathbf{z} = \left[\begin{array}{cccccccccccc} V & \alpha & \beta & p & q & r & \psi & \theta & \varphi & x_e & y_e & H & \dot{H} \end{array} \right]^T$$

This vector is practically equal to the state vector $\mathbf{x}$ from the aircraft model Beaver (see chapter 5), with two exceptions:

1. $\mathbf{z}$ contains the rate of climb or descent $\dot{H}$ as a thirteenth element,
2. $\mathbf{z}$ has *passed* the sensor models from the subsystem Sensors.

Because of this second difference, the output vector from the block Beaver Dynamics has been denoted as $[\mathbf{x}; \dot{H}]$ (written as $[x;\ Hdot]$ in the graphical system from figure 12.9) in stead of $\mathbf{z}$. The initial value of the vector $\mathbf{z}$ is called $\mathbf{z}_0$. In the graphical autopilot systems these vectors have been written as z and $z0$, respectively.

Figure 12.15 shows the structure of the subsystem Sensors. On the left-hand side of this system, the initial conditions are subtracted from the aircraft model outputs. The resulting vector is then *Demuxed* into separate scalar elements, which are sent to the corresponding sensor models. Here it is assumed that the sensor models are based upon small deviation signals too, otherwise the original outputs from the non-linear aircraft model should be used. On the right-hand side of the block-diagram, the outputs from the sensor models are *Muxed* again into one vector, which will be used as basic input vector for the control laws. Since these small-deviation signals are not suited for gain-scheduling purposes, a second vector is constructed. Here, the initial conditions are added again to the small deviation signals which leave the sensor blocks.

The difference between this signal and the original output vector from the non-linear aircraft model is completely caused by the sensor dynamics.

12.3.7 Additional blocks on the input side of the aircraft model

The masked subsystem Computational delay & limiters takes into account the time-delay which is caused by the sampling and computing actions of the Flight Control Computer, and the hardware-related limitation of the command signals to the actuators. In the FCC of the 'Beaver', the input signals for the control laws were sampled thirty times per second. Since the software is fast enough to evaluate all control laws within one sample-interval, the total time-delay equals approximately 0.03 *sec*.[1] The maximum allowable magnitude of the input signals to the actuators is equal to plus or minus 10 V. This imposes a hard limit to the magnitude of the *output* signals from the Flight Control Computer. For the 'Beaver' autopilot every individual contribution to the command signals for the actuators has been carefully balanced by means of internal limiters within the FCC software in order to keep the FCC outputs within this 10 V range. Yet, in order to cover for possible errors in the sizing of the internal limiters from the subsystems Asymmetrical autopilot modes and Symmetrical autopilot modes, the 10 V limit has also been implemented separately in the subsystem Computational delay & limiters.

The block Actuator & cable dynamics contains the dynamic models of the actuators and the cables to the control surfaces. Here, simplified linear second-order state-space models with two inputs and one output are used. For the model of the elevator actuator these inputs are the commanded elevator deflection and the non-dimensional pitch rate, the aileron actuator model uses the commanded aileron deflection and the non-dimensional roll rate, and the rudder actuator model uses the commanded rudder defection and the non-dimensional yaw rate. The non-dimensional angular velocities, which take into account the effect of rotational movements upon the aerodynamic effectiveness of the control surfaces, are extracted from the non-linear aircraft model. They are contained in the S-function output vector of Beaver, see equation (10.1) from chapter 10.

12.3.8 Additional blocks on the output side of the aircraft model

As shown in figure 12.15, the subsystem Sensors contains models of sensor dynamics. Actually this is only partially true, because most of the sensor blocks are still empty due to a lack of suitable models; the subsystem Sensors merely provides an easy way of extending the simulation model should better sensor models become available. Currently Sensors only contains:

- a time-delay of 0.4 seconds in the airspeed signal, which takes into account the time needed for changes in air-pressure to travel from the Pitot-tube to the airdata computer,
- a similar time-delay for the altitude signal, a quantizer which takes into account a Least Significant Bit of 4 *ft* in the measured altitude, and a Moving Average Filter which was implemented in the autopilot software to smoothen-out the altitude signal,
- a time-delay of 0.8 *sec* in the time-derivative of the altitude signal, which takes into account the transport delay in the Pitot-static system and the computational delay caused by the determination of the time-derivative of the altitude from different altitude samples within the airdata computer.

[1] For reasons of computing speed APILOT3 does not actually use sampled signals. It is possible to simulate the influence of sampling by means of a Zero Order Hold filter in the subsystem Sensors, but at a sampling rate of 30 *Hz*, the influence of sampling proved to be negligible. The influence of the computational delay is not very pronounced either, but contrary to the ZOH filter, this effect could be simulated easily without any penalties in computing speed.

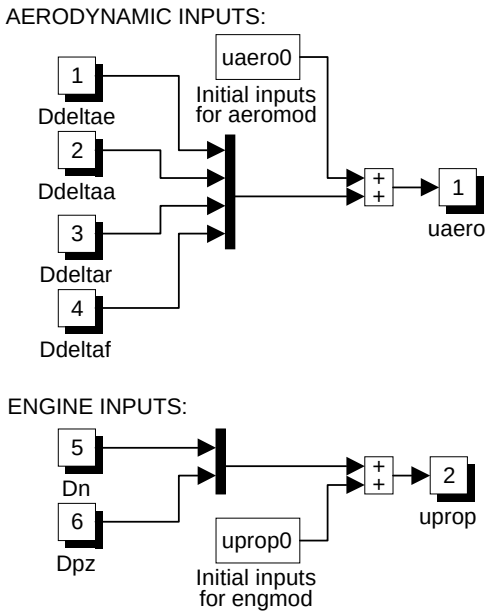


Figure 12.14: Internal structure of the subsystem Add initial inputs

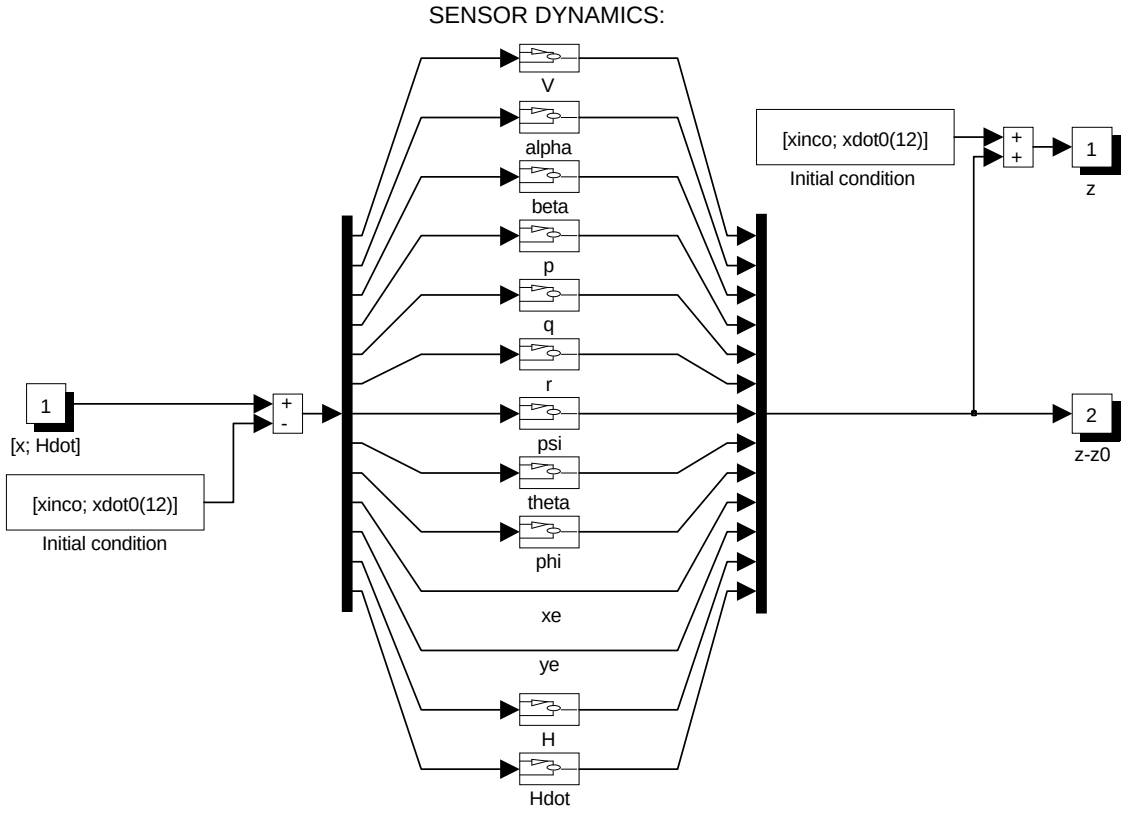


Figure 12.15: Internal structure of the subsystem Sensors/Subtract initial conditions

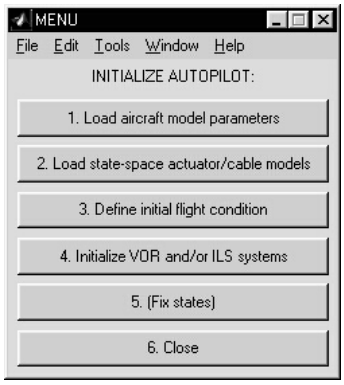


Figure 12.16: Main menu of the autopilot initialization routine APINIT

The altitude sensor block contains a 3-switch, generated by the MATLAB routine NSWITCH (see section 9.7.3), which makes it possible to select one of *three* different altitude outputs: *(i)* a continuous signal which has passed the 0.4 *sec* time-delay only; *(ii)* a discontinuous signal which has also passed the 4 *ft* quantizer block; *(iii)* a discontinuous signal which has been smoothened out by the Moving Average filter after passing the quantizer block first. Which option is chosen depends upon the value of the Constant block that is connected to the 3-switch. This can be changed after double-clicking the altitude sensor block within the subsystem Sensors/Subtract initial conditions.

12.4 Performing simulations with the autopilot models

12.4.1 Autopilot model initialization

Before starting a simulation of one of the autopilot models it is necessary to define all parameters from these systems. Two MATLAB routines to assist the user in this initialization task:

1. APINIT. This routine sets the initial values of the control signals and initializes the aircraft model, actuator and cable models, and radio navigation models. Moreover, it contains a 'shortcut' to the routine FIXSTATE that makes it possible to artificially fix one or more state variables from the aircraft model, e.g. to neglect longitudinal-lateral cross-coupling effects (see section 9.5). APINIT can be started by typing `apinit` at the command-line or by double-clicking the button-block Initialize subsystems in the top-level of the autopilot simulation model. Figure 12.16 shows the main menu from APINIT. The user should walk through the menu-options in the appropriate order to make sure all system parameters are properly set in the MATLAB workspace. Item 5 is optional and has therefore been put between brackets.
2. APMODE. This routine helps setting the longitudinal and lateral autopilot modes and the reference inputs. These reference signals define the shape of the *deviations* from the initial values of the input signals. Currently, it is possible only to specify step-shaped input values. If this step is set to zero, we can analyze the effectiveness of the 'Hold' modes of the autopilot under influence of external disturbances and longitudinal-lateral coupling effects. Non-zero steps are used to obtain step-responses. Simulating other types of input signals requires manual editing of the SIMULINK models of the autopilot. It is recommended always to use APMODE for setting the autopilot mode, because this ensures that only legitimate combinations of autopilot modes are selected. APMODE can be started by

Element number:	Symmetrical mode switches (*ymodS*):	Asymmetrical mode switches (*ymodA*):
1	Symmetrical mode On/Off	Asymmetrical mode On/Off
2	Symmetrical outer-loops On/Off	Asymmetrical outer-loops On/Off
3	ALH mode On/Off	HH mode On/Off
4	ALS mode On/Off	NAV mode On/Off
5	GS mode On/Off	LOC mod On/Off

Table 12.1: Definition of the elements from *ymodS* and *ymodA* (1 = On, 0 = Off)

typing `apmode` at the command-line or by double-clicking the button-block Select mode & reference signals in the autopilot simulation model. The user must specify the symmetrical autopilot mode and corresponding reference input signal and the asymmetrical autopilot mode and reference input. APMODE does not set the reference inputs for the approach and navigation modes; those definitions will be defined by clicking item 4 of the main menu from APINIT.

The switch-settings that define which autopilot mode is activated are stored within the MATLAB workspace in the variables *ymod1S*, *ymod1A*, *ymod2S*, and *ymod2A*. These variables are all defined by APMODE. The extensions *S* and *A* refer to symmetrical and asymmetrical autopilot modes, respectively, while the numbers *1* and *2* specify which phase the autopilot mode is in: *1* = *Armed*, i.e. the first phase of the control mode, and *2* = *Coupled*, i.e. the second phase.[1] The switch-setting variables are vectors with five elements which all have a value of either 0 or 1, which determine whether a switch should be 'opened' or 'closed'. Table 12.4.1 shows the meaning of all elements from *ymodS* and *ymodA*. In order to get valid combinations of control laws, APMODE only allows the following definitions of *ymodS* and *ymodA*:

Symmetrical:	*ymod1S (Armed):*	*ymod2S (Coupled):*
PAH:	[1 0 0 0 0]	[0 0 0 0 0]
ALH:	[1 1 1 0 0]	[0 0 0 0 0]
ALS:	[1 1 0 1 0]	[0 0 0 0 0]
GS:	[1 1 1 0 0]	[1 1 0 0 1]
Asymmetrical:	*ymod1A (Armed):*	*ymod2A (Coupled):*
RAH:	[1 0 0 0 0]	[0 0 0 0 0]
HH:	[1 1 1 0 0]	[0 0 0 0 0]
NAV:	[1 1 1 0 0]	[1 1 0 1 0]
LOC:	[1 1 1 0 0]	[1 1 0 0 1]

Moreover, APMODE determines valid combinations of symmetrical and asymmetrical modes, which both may have an independent Armed and Coupled phase. The Go-Around mode can be simulated by selecting the Pitch and Roll Attitude Hold modes and entering the appropriate reference values with APMODE ($\Delta\theta = 10°$, $\varphi = 0°$). For the Armed phase of the Glideslope mode, the ALH control law is used, while the Localizer and Navigation modes use the HH control law

[1] Although Armed and Coupled phases only apply to the radio-navigation modes, the variables *ymod2S* and *ymod2A* must always be defined in the MATLAB workspace when running simulations of the autopilot (except for the simplified model APILOT1 which does not contain radio-navigation models altogether). If they are not needed, their elements are set to zero by APMODE. Leaving them undefined would yield an error when starting a simulation.

in Armed phase.

The reference inputs to the basic control laws, being PAH, ALH, ALS, RAH, and HH, are stored in the variables *yrefS*, and *yrefA*. These variables are defined as follows:

$$\begin{aligned} yrefS &= [\Delta\theta_{ref}\ \Delta H_{ref}\ \Delta\dot{H}_{ref}]^T \\ yrefA &= [\Delta\varphi_{ref}\ \Delta\psi_{ref}]^T \end{aligned}$$

All elements from these vectors are constants, which implies that the simulation model only allows step-shaped changes in θ, H, $\dot{H}$, φ, and ψ to be defined (all at $t = 0$). If you want to change the shape of these test inputs, you must edit the block Reference Signals within APILOT1, APILOT2, or APILOT3.

The parameters for the Glideslope, Localizer, and Navigation modes are defined separately by APINIT, which calls the routines ILSINIT and VORINIT. Type `help ilsinit` or `help vorinit` at the command-line for more information.

12.4.2 Examples of non-linear autopilot simulations

In this section some examples of autopilot simulation experiments will be demonstrated in order to show how to use the systems APILOT1 to APILOT3 in practice. Since this report does not intend to fully cover the behaviour of the 'Beaver' autopilot itself, the reader is referred to refs.[22] and [29] for detailed information about the 'Beaver' autopilot project. Ref.[29] describes the linear design of the control laws, and ref.[22], treats the non-linear evaluations of the autopilot along with the very first version of the FDC toolbox which has now become obsolete.

Example: step response of the PAH mode

Suppose we want to analyze the response of the 'Beaver' to a step-shaped pitch-command for the PAH mode, similar to the analysis from section 12.2.2. Since we will not take into account wind, turbulence, and sensor models and we don't need the radio-navigation signals, it is possible to use the system APILOT1 for this simulation. This model contains complete implementations of all autopilot modes except approach and navigation. Type `apilot1` at the command-line to open APILOT1. Before starting a simulation, double-click the two button-blocks at the bottom side of the block-diagram, or type `apinit` and `apmode` at the command-line. From the user-menu of APINIT choose options 1 to 3 and answer the questions appearing in the command-window. Option 4 is not necessary because APILOT3 does not contain VOR or ILS models; option 5 is skipped because we don't want to fix any state variables of the aircraft model. After clicking button 2 select the actuator models for an airspeed of 45 ms^{-1}. After clicking button 3, load an initial *trimmed* flight condition from the file CR4520.TRI within the FDC subdirectory DATA. From the user-menu of APMODE select the Pitch Attitude Hold mode with a step input $\Delta\theta_{ref}$ equal to 1° and the Roll Attitude Hold mode with a reference value $\Delta\varphi_{ref} = 0$. If you now start the simulation, a time-trajectory will appear in the figure-window, generated by the Graph scope[1]. By default this scope will display θ, but you may connect it to another output line if you wish. See the definition of the S-function output vector in equation (10.1). By default, 60 seconds of flight will be simulated. After the simulation, run the routines RESULTS and RESPLOT respectively to get an overview of the results. Again it is useful to figure out how large the pitch step may be without the response becoming unstable. Compared with the earlier result from figure 12.8 the feasible $\Delta\theta$ range is now smaller, which is probably due to the influence of the limiters and/or continuous gain scheduling functions.

[1] You may get a warning message about an unconnected output line in APILOT1. Disregard this message; the diagram was purposely designed in this way.

Example: step response of the ALH and RAH modes with and without turn-compensation

As described in section 11.5 a turn-compensation loop was added to the pitch channel in order to minimize the influence of the bank angle upon the pitch angle in PAH mode and the altitude in ALH mode. The fine-tuning of the gains for these loops was done by means of series of non-linear simulations; this process has been treated in more detail in ref.[22]. Here we will briefly analyze the influence of the turn-compensation logic upon altitude responses due to roll angle commands. Open the system APILOT1 and initialize its subsystems by double-clicking the first button-block or running APINIT. Remember that it is not necessary to initialize the VOR and/or ILS systems when using APILOT1. Now double-click the second button-block or run APMODE. Select the Altitude Hold mode with reference input $\Delta H_{ref} = 0$ in combination with the Roll Attitude Hold mode with reference input $\Delta\varphi_{ref} = 30°$. Run the simulation, then run RESULTS and save the altitude response by typing: `time1 = time; H1 = H;` at the command-line. This is the altitude response *with* turn-compensation. Next, enter the system Symmetrical autopilot modes, double-click PAH, and delete the output line from the block Turn Compensator. Restart the simulation and run RESULTS again. If you now type: `plot(time,H,time1,H1)`, the altitude responses with and without turn-compensation will be displayed. Figure 12.17 shows the results expressed in feet. Obviously, the turn-compensator helps reducing the initial change in altitude that occurs after initiating a turn with a roll angle of 30°. Note: the gains for the turn-compensation were carefully fine-tuned for both left and right turns up to a level where any improvement in altitude response for left turns yielded a deterioration of the responses for right turns and vice versa. Since the responses were judged to be good enough, more sophisticated 'asymmetrical' turn-compensation loops were not taken into account during the design process.

Example: simulation of the Approach mode

For simulations of the approach mode it is necessary to specify the parameters of the radio-navigation models as well before starting a simulation. The user must make sure that the aircraft is in the vicinity of the Localizer and Glideslope reference planes at the beginning of a simulation, because the LOC and GS modes will not couple unless the aircraft passes these reference planes. For this reason, it is also required to select an appropriate initial heading and altitude for ILS capture. Let's take a closer look at this problem. First open the system APILOT2. This system does not take into account wind, turbulence, and sensor noise, but it does contain the required VOR and ILS models. Double click button 1 to initialize the subsystems of APILOT2. Click the appropriate buttons in the user menu to load the aircraft model parameters, load the actuator & cable models, load the initial flight condition from file (in this case, load CR4520.TRI from the default FDC data-directory), and initialize the ILS system. The following parameters must be entered at the command-line for the ILS initialization: runway height, initial X-distance from the aircraft to the runway, initial Y-distance from the aircraft to the runway, glideslope angle, runway heading, X-distance from the runway threshold to the localizer antenna, X-distance from the runway threshold to the glideslope antenna, and Y-distance from the runway centerline to the glideslope antenna. Here, enter 12000 m for the initial X-distance from the aircraft to the runway, and use default values for all other ILS parameters. Also use default values for the VOR system (the VOR parameters are used here as dummy variables only). Since the initial flight condition from the file CR4520.TRI sets the initial altitude to 2000 ft, the X-distance of 12000 m puts the aircraft in the vicinity of the glideslope reference plane. In this case the initial Y-position puts the aircraft *on* the extended runway centerline, and the initial heading corresponds with the reference runway heading ($\psi = \psi_{RW} = 0\ deg$).

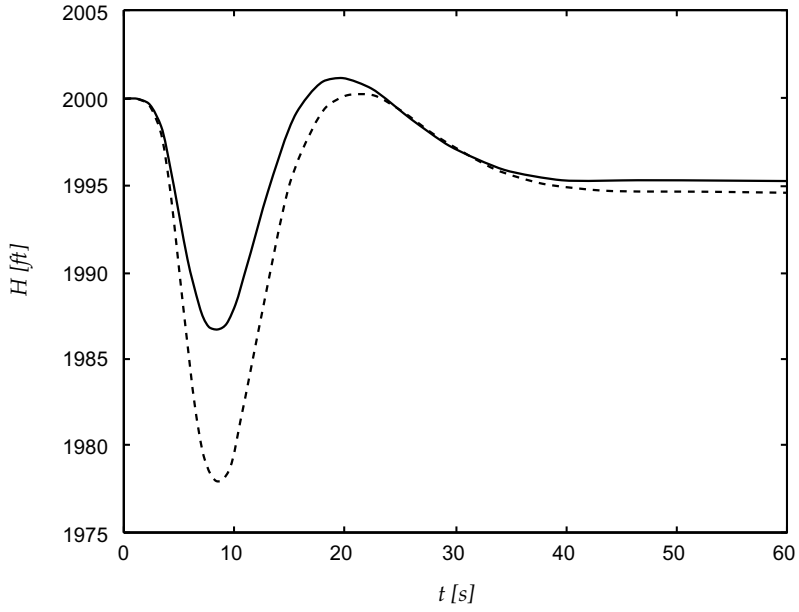


Figure 12.17: H-response to command $\Delta\varphi = 30°$ (— with, ... without turn-compensation)

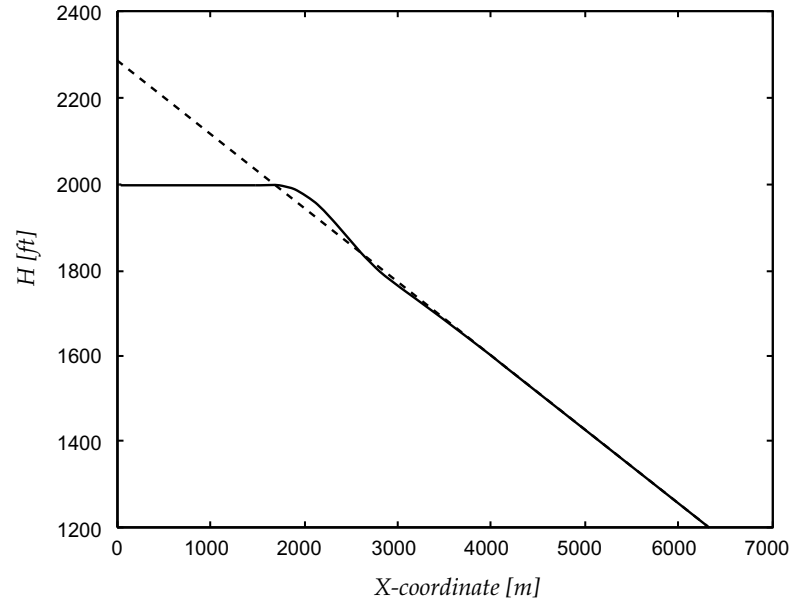


Figure 12.18: Altitude response when intercepting glideslope (... : reference line)

Now double-click the other button block, or run APMODE directly from the command-line and specify the *Glideslope* and *Localizer* modes to be used. You may now start the simulation. By default the Graph Scope block is coupled to the altitude. After approximately 8 seconds, the aircraft will capture the glideslope. Run RESULTS after the simulation has finished and plot the altitude against the coordinate x_e to see the glide-path. Figure 12.18 shows the resulting altitude response expressed in [*ft*]. It is also useful to plot the results which are stored in the matrix *yils*; enter `type ils.hlp` at the command-line for the definition of this matrix, or see table E.5 in appendix E. If you plot the airspeed V you will notice an considerable increase in airspeed when the aircraft starts to descend on the glideslope. This is due to the fact that the powersetting is not changed automatically for the descent. In the real aircraft the pilot would reduce power, which can be roughly simulated by *fixing* the airspeed to its initial value. Use option 5 of the initialization menu to do so, or type `fixstate` at the command-line. Select the option *Fix arbitrary states* and specify the vector `[1]` at the command-line to fix the first state variable, being the airspeed. See also section 9.5.

Chapter 13

Recommendations for future FDC releases

13.1 Transforming the toolbox to a central model library

The current toolbox provides ample possibilities for the users to adapt and enhance the models and tools. Combining all those new developments in a central model and tool library can quickly enhance the flexibility and power of the FDC toolbox, provided such a library is well maintained. Although every individual user is free to do whatever he wants with the FDC models and tools, all entries to a central model library should comply to certain rules, in order to prevent compatibility problems. That was also the main reason for issuing the license agreement from section 1.3. If there is one central model library to distribute the 'official' FDC toolbox and its enhancements, it will be much easier to support the toolbox and to determine who is responsible for compatibility problems and bugs of each distributed FDC version.

An example of such a central program library is the distribution of all TEX-related programs via the *Comprehensive TEXArchive Network (CTAN)* on Internet.[1] On a smaller scale, the same sort of program distribution via an ftp-site could be applied to the FDC toolbox and its future enhancements. There should be one or more 'librarians' which are ultimately responsible for deciding which entries are included to the main FDC distribution and which are not. This also makes it easier to respond to user-comments, while the user himself can easier issue own contributions to the FDC toolbox. It depends upon the user responses to this version of the toolbox whether such an ftp-site will ever be realized.

13.2 Porting SIMULINK models to other computer platforms

In theory it is possible to use an automatic code-generator for transforming graphical SIMULINK systems to a subroutine written in a high-level program language that can easily be ported to other systems. One application where this could be particularly useful is for transferring control laws from an off-line SIMULINK-based simulation environment to on-line flightsimulation and the Flight Control Computers (FCC's) of the test aircraft. Automating these conversions will drastically reduce the chances of making errors. It will ensure that the control laws used in flight are exactly the same as the control laws designed within the SIMULINK environment.

Figure 13.1 shows what this process would look like. In the early stages of the AFCS design process, the control laws are implemented as graphical block-diagrams in the fashion of

[1] This network is a set of fully-mirrored ftp-sites which provide up-to-date TEX-related software on the World Wide Web (`http://www.ora.com/homepages/CTAN-web`).

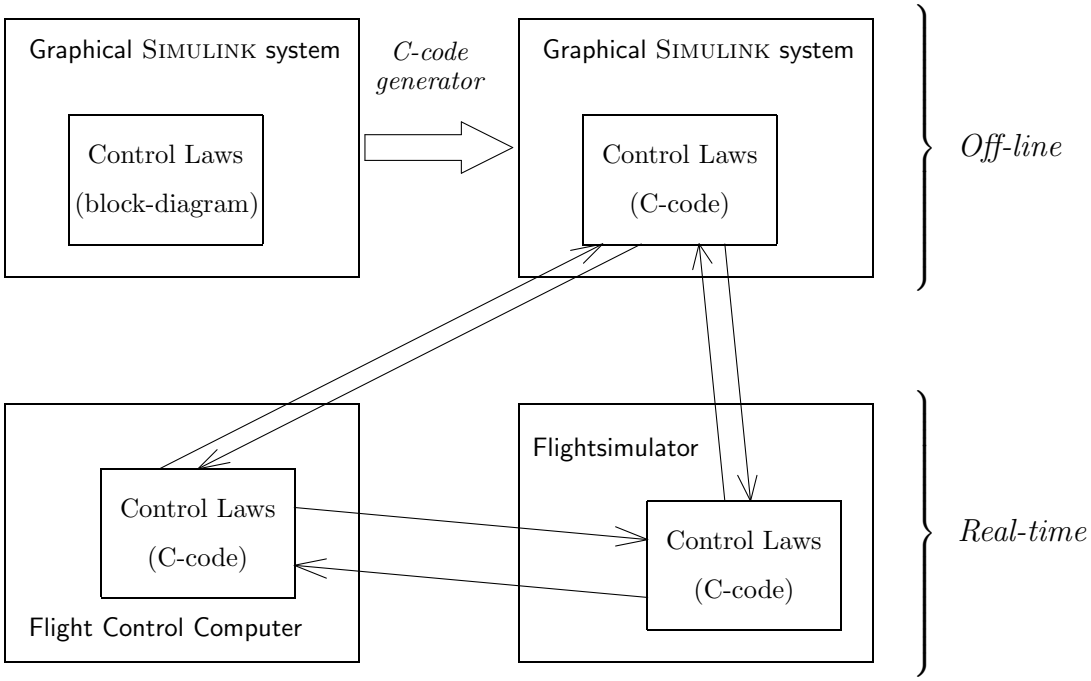


Figure 13.1: 'Portable' control laws

section 12.2. Next, the separate control laws are integrated within a block-diagram structure as treated in section 12.3, taking into account all possible external errors, sensor dynamics, actuator dynamics, discretization effects, time-delays, etc. This type of block-diagram represents the ultimate off-line simulation model of the AFCS. At this stage it becomes necessary to make the step towards *on-line* simulation. To achieve this, it is necessary to convert the part of the block-diagram that represents the control laws to a high-level language subroutine. For reasons of computing speed it is probably best to use the C language or take an object-oriented approach with $C++$ (in figure 13.1 it has been assumed that the block-diagrams are converted to C). Before porting this to the on-line flightsimulator or FCC's of the aircraft the resulting program should be verified thoroughly to ensure that the results match those of the graphical block-diagram. The best way to do this is to link the C programs to the original SIMULINK simulation model, thus replacing the graphical block-diagram representation of the AFCS by the new C subroutine. In SIMULINK this is possible by means of so-called MATLAB-executables (MEX-files).

Porting the C subroutine to the on-line flightsimulator is possible only if the simulator software contains interface-routines for linking external subroutines to the main simulation program. It is essential that the input/output relations between the simulator software and the subroutine with the AFCS control laws are clearly defined and 'compatible' with the off-line SIMULINK model. Should the SIMULINK model use different units of measurements, reference frames, etc., it is necessary to put an appropriate interfacing program between the AFCS software and the aircraft model within the simulator software. Obviously these considerations also apply to the implementation of AFCS software within the FCC's of the test aircraft.

Another problem that has to be solved is the pilot-interface. Obviously, the pilot must be able to enter the AFCS mode and reference values. It is recommended to use computer displays to represent the status of the AFCS in the aircraft or simulator, because that provides

a large flexibility with regard to the display formats. Furthermore, the pilot needs a flexible, standardized input device to enter settings and reference values to the AFCS. Such standardized display and input devices make it possible to test different experimental AFCS designs in the on-line flightsimulator and in real flight in parallel, without changing the hardware every time a new system is developed. In the off-line environment it is relatively easy to emulate such input/output devices from the flight-simulator and the test aircraft.

Theoretically we could also go one step beyond the automatic code generation for control laws only, because an automatic code generator can also be applied to the *complete* simulation model (i.e. aircraft model + external disturbances + AFCS control laws + sensor dynamics + actuator dynamics + ...). This opens the possibility to port a complete simulation model from SIMULINK to the on-line flightsimulator. However, there are a number of fundamental problems to be solved for achieving that level of portability. First of all, the SIMULINK environment is by itself not suited for real-time simulations while the on-line flightsimulator *must* have real-time capabilities. Using the converted SIMULINK model as a subroutine of the flightsimulator thus requires careful control of the timing of the program. Secondly, the interfacing problem becomes even more pronounced if the complete aircraft model is ported since the simulation model needs to be coupled to the software subroutines which control the display, flight instruments, motion system, etc. The interfacing problem is enlarged by the enormous flexibility of SIMULINK. Only by making solid guidelines for the input/output structure of the graphical systems it will be possible to achieve this level of portability. Thirdly, it will still be necessary to extract the AFCS control laws separately for implementation within the FCC's of the aircraft. All these problems eventually can be solved, but it will require quite some programming efforts plus careful application of hardware interface devices. Such extensions to the research environment can probably best be achieved with help from experienced programmers and hardware developers. For this reason it seems recommendable to start with a more limited approach, taking care of the portability of the control laws first.

13.3 Other possible improvements for future FDC releases

FDC 1.2 provides the basis for future implementations of other aircraft models and their application to a large variety of research and design tasks. However, since many of its subroutines are still heavily leaning towards the original 'Beaver' model, the implementation of other aircraft models is not as straightforward as it should be. The current tools and models are quite flexible, but not yet flexible enough. Also, the structure of some programs has become somewhat incomprehensible because of the increasing complexity due to new options and safeguards for the users, despite the highly improved on-line help facilities and documentation within the source-codes themselves. The basic programming efforts for new FDC versions will therefore be aimed at improving the *structure* of the MATLAB programs and increasing their *flexibility* to cope with changes in the SIMULINK systems. For the graphical systems themselves it is also necessary to increase the flexibility of the package, which in particular will require a major overhaul of the first level of the non-linear aircraft model.

Some other improvements under consideration for future FDC releases are listed below:

- In order to keep the toolbox easily accessible to the users, a further improvement of the user-interface will have to be considered. All user-menu's should be brought to the MATLAB 4.x standard, which should completely eliminate the necessity of using the MATLAB command window for entering system parameters, directory names, etc. when using FDC

tools. In the current version it is too often necessary to switch from graphical user-menus to the text-oriented command window, and vice versa.

- It should be possible to define units of measurements for the different variables from the FDC models and analytical tools as *global* definitions. Currently, the user is often forced to stick to the S.I. units which may be rather inconvenient in some instances (e.g. in aviation the altitude is often expressed in $[ft]$ in stead of $[m]$, and ILS deviations are expressed in *'dots'* in stead of $[\mu A]$).
- Future versions of the toolbox should contain more flexible and user-friendly tools to process simulation data (i.e. generalized replacements for RESULTS and RESPLOT).
- The on-line help-texts should be further developed to a windows-oriented format such as the 'MATLAB Expo' demonstration. Of course, a hypertext-oriented help environment would be even better (it may be a good idea to use a HTML-based help environment, because Internet browsers are nowadays very common for PC's *and* workstations).
- It may be a good idea to couple the VOR and ILS models to animation routines that display typical cockpit instruments, such as a Radio Magnetic Indicator, Track Deviation Indicator, or Horizontal Situation Indicator (see ref.[3]). The current output signals from these models tend to be rather abstract in comparison with the readings from cockpit instruments, which may be a problem in cases where it is necessary to consult pilots. Other animation routines may also be useful, for instance to visualize the simulated motions of the aircraft.

At the moment of writing some of these improvements were already being implemented, so you may find some additional, undocumented options for the FDC toolbox at your FDC installation diskette. Consult the file README1.TXT for the latest information!

Appendix A

Symbols and definitions

In this appendix, the symbols used in this report will be defined, along with the frames of reference and sign conventions. Due to the very large number of variables used in this report, it sometimes has been necessary to use the same symbols for different variables. In practice, this is not a serious problem since the meaning of a particular symbol usually follows directly from the context in which it is used. In cases where confusion can arise, some symbols have been overlined to make the necessary distinction. For the units of measurement in most cases the S.I. conventions are used in order to prevent confusion. Only in a few cases where equations from other literature has been used the units of measurements may differ from the S.I. units. A list of variable names and acronyms has been included in appendix E. Appendix C defines the parameters of the mathematical model of the aircraft; appendix D contains the parameters for the SIMULINK implementation of this model.

A.1 List of symbols

a	[ms^{-1}]	speed of sound
$a_{x,k}$	[g]	kinematic acceleration along X_B-axis
$a_{y,k}$	[g]	kinematic acceleration along Y_B-axis
$a_{z,k}$	[g]	kinematic acceleration along Z_B-axis
A_x	[g]	output of accelerometer (specific force) in c.g. along X_B-axis
A_y	[g]	output of accelerometer (specific force) in c.g. along Y_B-axis
A_z	[g]	output of accelerometer (specific force) in c.g. along Z_B-axis
b	[m]	wing-span
$\overline{c}$	[m]	mean aerodynamic chord
CD	[rad]	course datum (selected VOR bearing)
C_1	[–]	engine model parameter
C_2	[–]	engine model parameter
C_l	[–]	non-dimensional moment about X_B-axis (rolling moment)
C_m	[–]	non-dimensional moment about Y_B-axis (pitching moment)
C_n	[–]	non-dimensional moment about Z_B-axis (yawing moment)
C_X	[–]	non-dimensional force along X_B-axis
C_Y	[–]	non-dimensional force along Y_B-axis
C_Z	[–]	non-dimensional force along Z_B-axis
$\overline{D}$	[N]	total aerodynamic drag
d_{gs}	[m]	distance from aircraft to nominal glideslope line
d_{loc}	[m]	distance from aircraft to extended runway centerline
dpt	[–]	$\frac{\Delta p_t}{\frac{1}{2}\rho V^2}$ = non-dimensional pressure increase across propeller
F_B		body-fixed reference frame
F_E		Earth-fixed reference frame

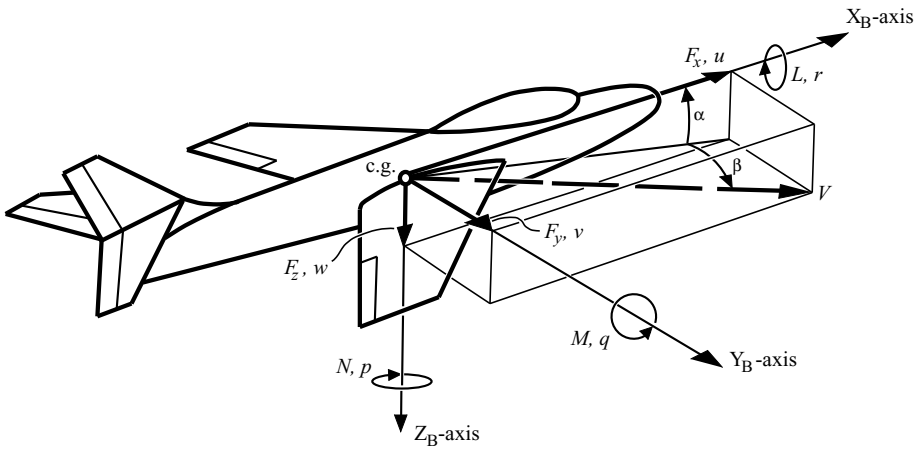


Figure A.1: Definitions of (angular) velocity components p, q, and r, angle of attack α, sideslip angle β, and external forces & moments F_x, F_y, F_z, L, M, and N

F_M		measurement reference frame
F_R		special body-fixed reference frame for the 'Beaver'
F_S		stability reference frame
F_V		vehicle-carried vertical reference frame
F_W		flight-path reference frame
F_x	[N]	total external force along X_B-axis, see figure A.1
F_y	[N]	total external force along Y_B-axis, see figure A.1
F_z	[N]	total external force along Z_B-axis, see figure A.1
fpa	[g]	flight-path acceleration
g	[ms^{-2}]	acceleration of gravity
h	[m]	pressure altitude
h	[s]	step-size for numerical integration
H	[m]	geopotential altitude
H_f	[m]	height above aerodrome level
H_{RW}	[m]	runway elevation above sea level
i	[–]	counter or iteration number
i_{gs}	[μA]	glideslope current (ILS)
I_i	[kgm^2]	inertia parameter (i = 1, 2, …, 6, see appendix B, table B.2)
i_{loc}	[μA]	localizer current (ILS)
I_x	[kgm^2]	moment of inertia along X_B-axis
I_y	[kgm^2]	moment of inertia along Y_B-axis
I_z	[kgm^2]	moment of inertia along Z_B-axis
J_{xy}	[kgm^2]	product of inertia in X_BY_B-plane
J_{xz}	[kgm^2]	product of inertia in X_BZ_B-plane
J_{yz}	[kgm^2]	product of inertia in Y_BZ_B-plane
k	[–]	time-step for discrete systems, $t \equiv kt_s$
$K_{...}$		gain factors for autopilot model, see table 11.3
L	[Nm]	total rolling moment, see figure A.1
$\overline{L}$	[N]	total aerodynamic lift
L_g	[m]	general scale length for atmospheric turbulence
L_u	[m]	scale length for turbulence velocity along X_B-axis
L_v	[m]	scale length for turbulence velocity along Y_B-axis
L_w	[m]	scale length for turbulence velocity along Z_B-axis
m	[kg]	mass of the aircraft

M	[–]	Mach number
M	[Nm]	total pitching moment, see figure A.1
M_a	[$kg\ kmol^{-1}$]	molecular weight of the air
M_0	[$kg\ kmol^{-1}$]	molecular weight of the air at sea level
n	[RPM]	engine speed
N	[Nm]	total yawing moment, see figure A.1
p	[$rad\,s^{-1}$]	angular rate of roll, see figure A.1
P	[$Nm\,s^{-1}$]	engine power
P_l, P_m, P_n, P_{pp}, P_{pq}, P_{pr}, P_{qq}, P_{qr}, P_{rr}		inertia parameters for p-equation (see appendix B, table B.2)
p_s	[Nm^{-2}]	ambient or free-stream pressure
p_z	[$''Hg$]	manifold pressure
q	[$rad\,s^{-1}$]	angular rate of pitch, see figure A.1
q_c	[Nm^{-2}]	impact pressure
q_{dyn}	[Nm^{-2}]	dynamic pressure
Q_l, Q_m, Q_n, Q_{pp}, Q_{pq}, Q_{pr}, Q_{qq}, Q_{qr}, Q_{rr}		inertia parameters for q-equation (see appendix B, table B.2)
r	[$rad\,s^{-1}$]	angular rate or yaw, see figure A.1
R	[$JK^{-1}kg^{-1}$]	R_a/M_0, specific gas constant of air
R_a	[$JK^{-1}kmol^{-1}$]	universal gas constant
R_c	[–]	Reynolds number with respect to $\overline{c}$
R_e	[m^{-1}]	Reynolds number per unit length
R_{gs}	[m]	ground distance from aircraft to glideslope transmitter (ILS)
R_{loc}	[m]	ground distance from aircraft to localizer transmitter (ILS)
R_l, R_m, R_n, R_{pp}, R_{pq}, R_{pr}, R_{qq}, R_{qr}, R_{rr}		inertia parameters for r-equation (see appendix B, table B.2)
S	[m^2]	wing area
S_{gs}	[$\mu A\,rad^{-1}$]	sensitivity of the glideslope system (ILS)
S_{loc}	[$\mu A\,rad^{-1}$]	sensitivity of the localizer system (ILS)
t	[s]	time
t_s	[s]	sampling time (step-width for discrete systems)
T	[K]	ambient or free-stream temperature
T_t	[K]	total temperature
u	[ms^{-1}]	velocity component along X_B-axis
u_g	[ms^{-1}]	component of the turbulence velocity along X_B-axis
u_w	[ms^{-1}]	wind velocity component along X_B-axis
u_{we}	[ms^{-1}]	wind velocity component along X_E-axis
v	[ms^{-1}]	velocity component along Y_B-axis
v_g	[ms^{-1}]	component of the turbulence velocity along Y_B-axis
v_w	[ms^{-1}]	wind velocity component along Y_B-axis
v_{we}	[ms^{-1}]	wind velocity component along X_E-axis
V	[ms^{-1}]	true airspeed, see figure A.1
V_c	[ms^{-1}]	calibrated airspeed
V_e	[ms^{-1}]	equivalent airspeed
V_w	[ms^{-1}]	wind velocity
$V_{w_{9.15}}$	[ms^{-1}]	wind velocity at 9.15 m altitude (reference velocity)

$V\Delta\delta_a$	[V]	command signal to actuators (change in deflection of the ailerons)
$V\Delta\delta_e$	[V]	command signal to actuators (change in elevator deflection)
$V\Delta\delta_r$	[V]	command signal to actuators (change in rudder deflection)
w	[ms^{-1}]	velocity component along Z_B-axis
w_1, w_2, w_3	[$-$]	independent white noise signals
w_g	[ms^{-1}]	component of the turbulence velocity along Z_B-axis
w_w	[ms^{-1}]	wind velocity component along Z_B-axis
w_{we}	[ms^{-1}]	wind velocity component along X_E-axis
W	[N]	aircraft weight
X_a	[N]	aerodynamic force along X_B-axis
x_e	[m]	X-coordinate in Earth-fixed reference frame F_E
x_f	[m]	X-coordinate in runway-fixed reference frame F_F
X_{gr}	[N]	gravity force along X_B-axis
X_p	[N]	propulsive force along X_B-axis
Y_a	[N]	aerodynamic force along Y_B-axis
y_e	[m]	Y-coordinate in Earth-fixed reference frame F_E
y_f	[m]	Y-coordinate in runway-fixed reference frame F_F
Y_{gr}	[N]	gravity force along Y_B-axis
Y_p	[N]	propulsive force along Y_B-axis
Z_a	[N]	aerodynamic force along Z_B-axis
z_e	[m]	Z-coordinate in Earth-fixed reference frame F_E
z_f	[m]	Z-coordinate in runway-fixed reference frame F_F
Z_{gr}	[N]	gravity force along Z_B-axis
Z_p	[N]	propulsive force along Z_B-axis
α	[rad]	angle of attack, see figure A.1
β	[rad]	sideslip angle, see figure A.1
γ	[rad]	flight-path angle
γ	[$-$]	ratio of specific heats of air
γ_{gs}	[rad]	nominal glide-path angle on final approach (ILS glide-path)
Γ_{gs}	[rad]	angle between localizer reference plane and the line through the ground position of the aircraft and the *glideslope* antenna
Γ_{loc}	[rad]	angle between localizer reference plane and the line through the ground position of the aircraft and the *localizer* antenna
Γ_{VOR}	[rad]	angle between selected and actual VOR radial
δ_a	[rad]	deflection of ailerons ($\delta_a = \delta_{a_{right}} - \delta_{a_{left}}$)
δ_e	[rad]	deflection of elevator
δ_f	[rad]	deflection of flaps
δ_r	[rad]	deflection of rudder
Δ		increment
Δp_t	[Nm^{-2}]	increase of total pressure over the propeller
$\Delta\delta_a$	[rad]	change in deflection of the ailerons
$\Delta\delta_e$	[rad]	change in elevator deflection
$\Delta\delta_r$	[rad]	change in rudder deflection
ε_{gs}	[rad]	glideslope error angle above/below the nominal ILS glide-path
θ	[rad]	pitch angle
λ	[Km^{-1}]	temperature gradient ($\partial T/\partial h$)
μ	[rad]	aerodynamic angle of roll
μ	[$kg\,m^{-1}s^{-1}$]	dynamic viscosity
ρ	[$kg\,m^{-3}$]	air density
σ		standard deviation
σ_{gs}	[μA]	standard deviation of glideslope noise
σ_{loc}	[μA]	standard deviation of localizer noise
σ_u	[ms^{-1}]	standard deviation of turbulence velocity in X_B-direction
σ_v	[ms^{-1}]	standard deviation of turbulence velocity in Y_B-direction
σ_w	[ms^{-1}]	standard deviation of turbulence velocity in Z_B-direction

φ	[rad]	roll angle
Φ	[rad]	bank angle
χ	[rad]	azimuth angle
ψ	[rad]	yaw angle
ψ_w	[rad]	wind direction (wind from the North: $\psi_w = \pi$)
ψ_{RW}	[rad]	runway heading
ω	[$rad\,s^{-1}$]	angular frequency
ω_0	[$rad\,s^{-1}$]	natural frequency of undamped system
ω_n	[$rad\,s^{-1}$]	natural frequency of damped system
Ω	[$radm^{-1}$]	spatial frequency

A.2 Vectors

$\mathbf{a}$	body-axes acceleration vector
$\mathbf{C}_{aero}$	vector with non-dimensional aerodynamic force and moment coefficients
$\mathbf{C}_{prop}$	vector with non-dimensional engine force and moment coefficients ('propulsive')
$\mathbf{F}$	resulting force vector acting on rigid body ($\mathbf{F} = [F_x\ F_y\ F_z]^T$)
$\mathbf{h}$	resulting angular momentum of rigid body about c.g. ($\mathbf{h} = [h_x\ h_y\ h_z]^T$)
$\mathbf{M}$	resulting moment vector about c.g. of rigid body ($\mathbf{M} = [L\ M\ N]^T$)
$\mathbf{r}$	position vector
$\mathbf{u}_{...}$	input vectors
$\mathbf{V}$	true airspeed vector
$\mathbf{V}_w$	wind velocity vector
$\mathbf{x}$	state vector
$\mathbf{y}_{...}$	output vectors
$\mathbf{\Omega}$	rotational velocity vector

A.3 Matrices

A	system matrix of linear state-space system
B	input matrix of linear state-space system
C	output matrix of linear state-space system due to $\mathbf{x}$
D	output matrix of linear state-space system due to $\mathbf{u}$
$T_{P\rightarrow Q}$	transformation matrix from a reference frame F_P to a reference frame F_Q
Θ	transformation matrix for first Euler rotation from F_E to F_B
Φ	transformation matrix for second Euler rotation from F_E to F_B
Ψ	transformation matrix for second Euler rotation from F_E to F_B

A.4 Functions

$\mathbf{f}(t)$	general vector-equation for the time-derivatives of the state variables
$\mathbf{g}(t)$	general vector-equation for the output variables
$H(\omega)$	frequency response of forming filter
$S(\omega)$	power spectral density function
$S(\Omega)$	power spectral density function

A.5 Indices and subscripts

0	nominal value
0	value at sea level
a	ailerons
a	relative to the surrounding atmosphere (used for velocity components)

aero	aerodynamic forces and moments or force and moment coefficients
c.g.	center of gravity
dpt	stability derivative with respect to non-dimensional pressure increase across propeller
e	elevator
e	relative to Earth axes (used for velocity components)
f	flaps
f	referenced to runway-fixed reference frame F_F, F stands for 'field'
grav	gravity force components
gs	related to glideslope deviation
k	'kinematic' (used for accelerations)
loc	related to localizer deviation
p	stability derivative with respect to non-dimensional rolling speed
q	stability derivative with respect to non-dimensional pitching speed
r	stability derivative with respect to non-dimensional yawing speed
r	rudder
RW	runway, used for defining runway parameters
prop	engine forces and moments or force and moment coefficients ('propulsive')
VOR	related to VOR signals
w	wind velocity and wind velocity components along body axes
wind	force components due to non-steady atmosphere
we	wind velocity components along Earth axes
α	stability derivative with respect to angle of attack
α^2	stability derivative with respect to α^2
α^3	stability derivative with respect to α^3
$\alpha\delta_f$	stability derivative with respect to $\alpha\delta_f$
$\alpha\, dpt^2$	stability derivative with respect to $\alpha\, dpt^2$
$\alpha^2 dpt$	stability derivative with respect to $\alpha^2 dpt$
β	stability derivative with respect to sideslip angle
β^2	stability derivative with respect to β^2
β^3	stability derivative with respect to β^3
$\dot{\beta}$	stability derivative with respect to non-dimensional sideslip *rate*
δ_a	stability derivative with respect to deflection of ailerons
$\delta_a\alpha$	stability derivative with respect to $\delta_a\alpha$
δ_e	stability derivative with respect to deflection of elevator
$\delta_e\beta^2$	stability derivative with respect to $\delta_e\beta^2$
δ_f	stability derivative with respect to deflection of flaps
δ_r	stability derivative with respect to deflection of rudder
$\delta_r\alpha$	stability derivative with respect to $\delta_r\alpha$

A.6 Abbreviations

AFCS	Automatic Flight Control System
ALH	Altitude Hold mode of autopilot
ALS	Altitude Select mode of autopilot
CACSD	Computer Aided Control System Design
CD	Course Datum
c.g.	center of gravity
DHC	De Havilland of Canada Ltd.
DME	Distance Measuring Equipment
DUT	Delft University of Technology
FCC	Flight Control Computer
FDC	Flight Dynamics and Control
GA	Go Around mode of autopilot
GS	Glideslope mode of autopilot

HH	Heading Hold / Heading Select mode of autopilot
ILS	Instrument Landing System
LOC	Localizer mode of autopilot
NAV	VOR Navigation mode of autopilot
PAH	Pitch Attitude Hold mode of autopilot
ODE	Ordinary Differential Equation
RAH	Roll Attitude Hold mode of autopilot
STOL	Short Take-off and Landing
TAS	True Airspeed
VOR	Very high frequency Omnidirectional Range
VTOL	Vertical Take-off and Landing

A.7 Reference frames and sign conventions

A.7.1 Definitions

The definitions of the reference frames used within this report are given below. The reference frames F_M and F_R have been added to this list, although they will be used only in table C.2 in appendix C. The equations for translational and angular velocities are referenced to the body axes F_B. The aircraft *attitude* is defined by the Euler angles ψ, θ, and φ, for which purpose the vehicle-carried vertical reference frame F_V is introduced. The aircraft *position* is defined with respect to the Earth-fixed reference frame F_E.

Measurement reference frame F_M: This is a *left-handed* orthogonal reference frame that is used for correcting stability and control derivatives of the aircraft if the c.g. position differs from the one used during the flight tests. For the 'Beaver' aircraft, the origin O_M lies in a point, resulting from the perpendicular projection of the foremost point of the wing chord, parallel to the $O_BX_BZ_B$-plane (ref.[26]). The $X_MO_MZ_M$-plane coincides with the $O_BX_BZ_B$-plane. The positive X_M-axis points backwards, the positive Y_M-axis points to the left, and the positive Z_M-axis points upwards.

Body-fixed reference frame F_B: This is a right-handed orthogonal reference system which has its origin O_B in the center of gravity of the aircraft. The $X_BO_BZ_B$ plane coincides with the aircraft's plane of symmetry if it is symmetric, or it is located in a plane, approximating what would be the plane of symmetry if it is not (ref.[9]). The X_B-axis is directed towards the nose of the aircraft, the Y_B-axis points to the right wing (starboard), and the Z_B-axis points towards the bottom of the aircraft.

Special body-fixed reference frame for the 'Beaver', F_R: This reference frame is defined specifically for the 'Beaver' aircraft. It is identical to F_B with one exception: its origin O_R is placed in a body-fixed reference point, which has been selected to coincide with a c.g. position that was actually used during one flight. It has the following coordinates in F_M: $x = 0.5996\,m$, $y = 0\,m$, $z = -0.8815\,m$, see ref.[26]. F_R is used only to define the moments and products of inertia for the aircraft condition on which the aerodynamic model is based (see table C.2 in appendix C).

Stability reference frame F_S: This is a special body-fixed reference frame, used in the study of small deviations from a nominal flight condition. The reference frames F_B and F_S differ in the orientation of their respective X-axes. The X_S-axis is chosen parallel to the projection of the true airspeed vector $\mathbf{V}$ on the $O_BX_BZ_B$-plane (if the aircraft is symmetric this is the plane of symmetry), or parallel to $\mathbf{V}$ itself in case of a symmetrical nominal flight condition. The Y_S-axis coincides with the Y_B-axis.

Flight-path or wind reference frame F_W: This reference frame, also called the wind reference frame, has its origin in the c.g. of the aircraft. The X_W-axis is aligned with the velocity vector of the aircraft and the Z_W-axis coincides with the Z_S-axis.

Earth-fixed reference frame F_E: This reference frame, also called the topodetic reference frame (ref.[9]), is a right-handed orthogonal system which is considered to be fixed in space. Its origin can be placed at an arbitrary position, but will be chosen to coincide with the aircraft's center of gravity at the start of a flight test manoeuvre. The Z_E-axis points downwards, parallel to the local direction of gravity. The X_E-axis is directed to the North, the Y_E-axis to the East.

Vehicle-carried vertical reference system F_V: This reference system has its origin at the c.g. of the aircraft. The X_V-axis is directed to the North, the Y_V-axis to the East, and the Z_V-axis points downwards (along the local direction of gravity). These reference axes are always parallel to the Earth-fixed reference axes, although the origin O_V moves relatively to the Earth-fixed reference frame.

A.7.2 Relationships between the reference frames

In figure A.2 the relationship between the Earth-fixed and vehicle-carried vertical reference systems is shown. F_E and F_V differ only in the position of their respective origins. The relationship between the vehicle-carried vertical and body-axes is shown in figure A.3. The Euler angles ψ, θ, and φ define the orientation of F_B with respect to F_V, hence they define the *attitude* of the aircraft with respect to the Earth's surface. The transformation matrices that express each of the Euler rotations separately are:

$$\Psi = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

$$\Theta = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{A.2}$$

$$\Phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{bmatrix} \tag{A.3}$$

The total transformation matrix from F_V to F_B then becomes:

$$\begin{aligned} T_{V\to B} &= \Phi\cdot\Theta\cdot\Psi = \\ &= \begin{bmatrix} \cos\psi\cos\theta & \sin\psi\cos\theta & -\sin\theta \\ \cos\psi\sin\theta\sin\varphi - \sin\psi\cos\varphi & \sin\psi\sin\theta\sin\varphi + \cos\psi\cos\varphi & \cos\theta\sin\varphi \\ \cos\psi\sin\theta\cos\varphi + \sin\psi\sin\varphi & \sin\psi\sin\theta\cos\varphi - \cos\psi\sin\varphi & \cos\theta\cos\varphi \end{bmatrix} \end{aligned} \tag{A.4}$$

so the relation between a vector $\mathbf{y}_B$ in the body reference frame and $\mathbf{y}_V$ in the vehicle-carried vertical reference frame is:

$$\mathbf{y}_B = T_{V\to B}\cdot\mathbf{y}_V \tag{A.5}$$

The orientation of the flight-path axes with respect to the vehicle-carried vertical axes can also be expressed in terms of Euler angles, denoted by χ, γ, and μ. This is shown in figure A.4.

The relationships between the body, flight-path, and stability reference frames are shown in figure A.5. These three reference systems all have their origin in the aircraft's center of gravity.

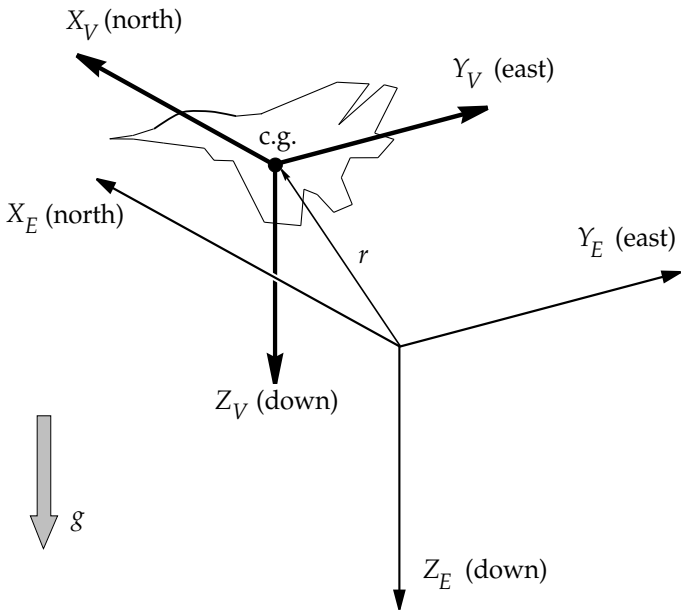


Figure A.2: Relationship between the vehicle-carried vertical reference frame F_V and the Earth-fixed reference frame F_E

The X_W-axis is aligned with the velocity vector of the aircraft. The orientation of the flight-path axes with respect to the body-fixed reference frame is defined by the angle of attack α and the sideslip angle β. The stability reference system is displaced from the flight-path axes by a rotation β and from the body axes by a rotation $-\alpha$.

A.7.3 Sign conventions for deflections of control surfaces

Figure A.6 shows the positive directions of control surface deflections. The positive elevator deflection is measured downwards; a positive value of δ_e results in a pitch-down moment to the aircraft. The deflections of the rudder and ailerons are positive if they force the aircraft to move to the left. If one aileron deflection has a positive sign, the other one consequently is negative. The 'total' aileron deflection is defined as: $\delta_a = \delta_{a_{right}} - \delta_{a_{left}}$. The flap angle is positive if the flaps deflect downwards (which they always do), similar to the elevator deflection. A positive value of δ_f results in an increase in lift and drag of the aircraft.

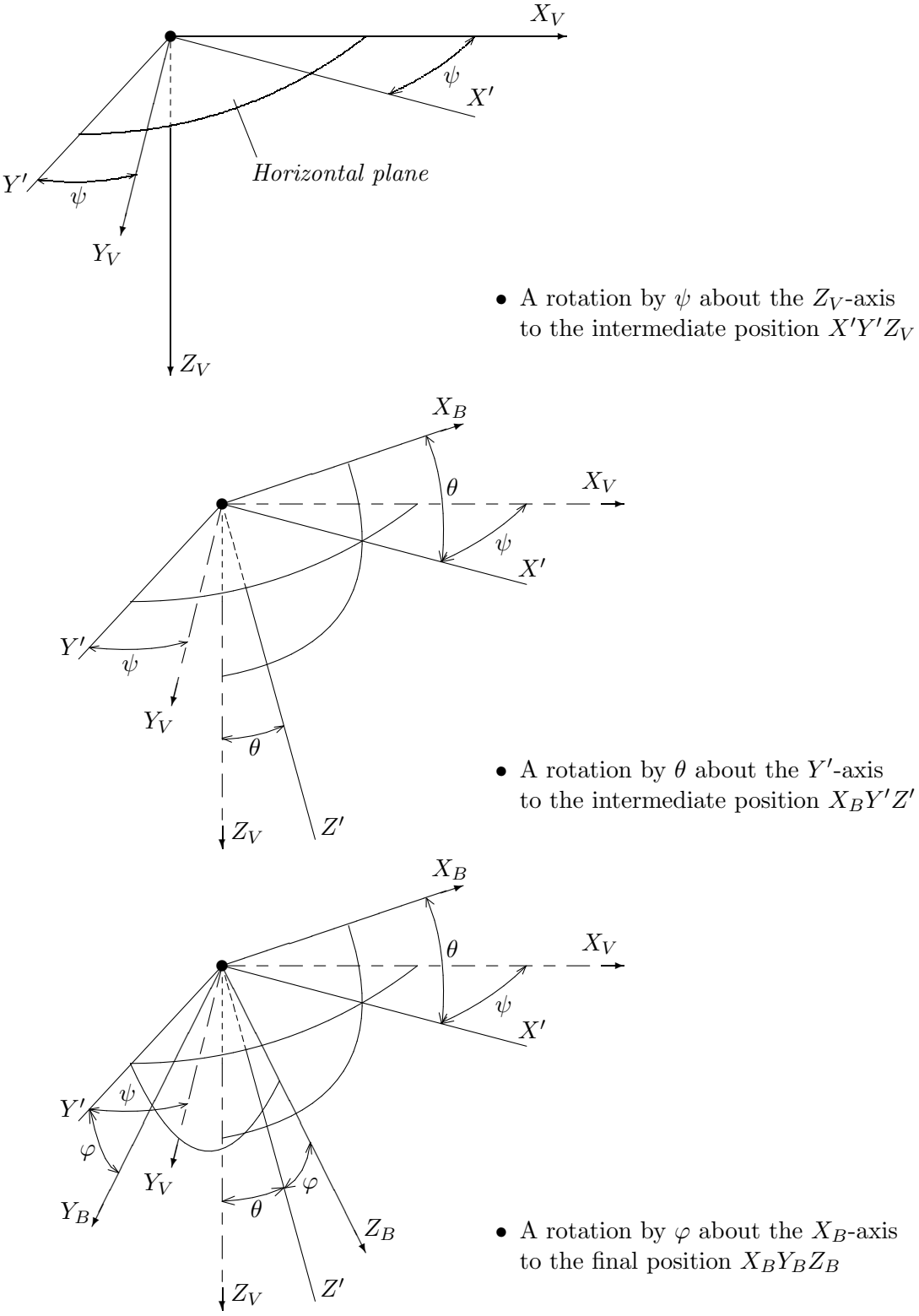
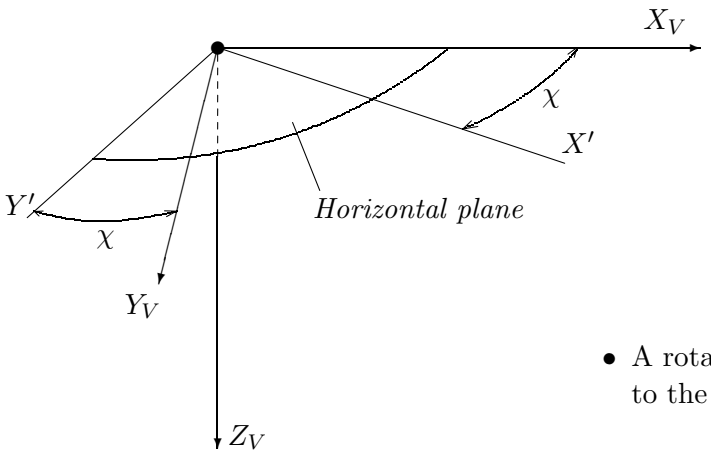
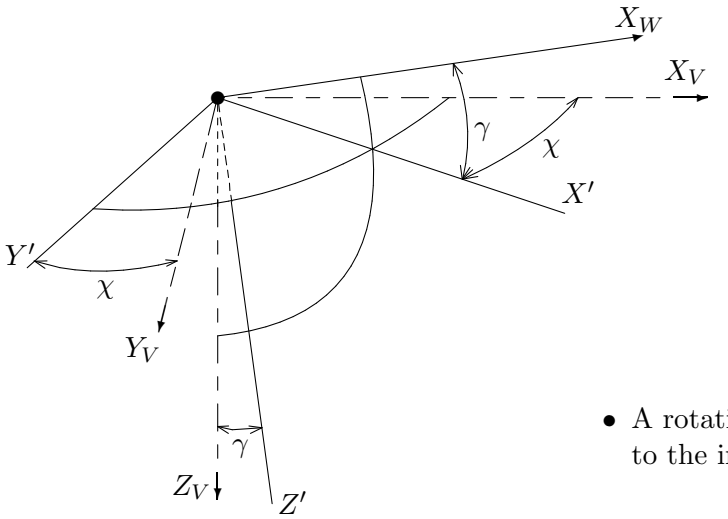


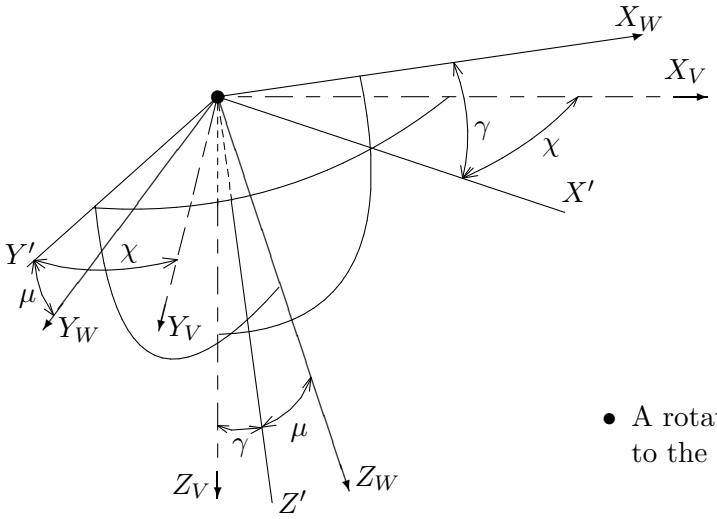
Figure A.3: Relationship between the vehicle-carried vertical reference frame F_V and the body-fixed reference frame F_B



- A rotation by χ about the Z_V-axis to the intermediate position $X'Y'Z_V$



- A rotation by γ about the Y'-axis to the intermediate position $X_W Y' Z'$



- A rotation by μ about the X_W-axis to the final position $X_W Y_W Z_W$

Figure A.4: Relationship between the vehicle-carried vertical reference frame F_V and the flight-path reference frame F_W

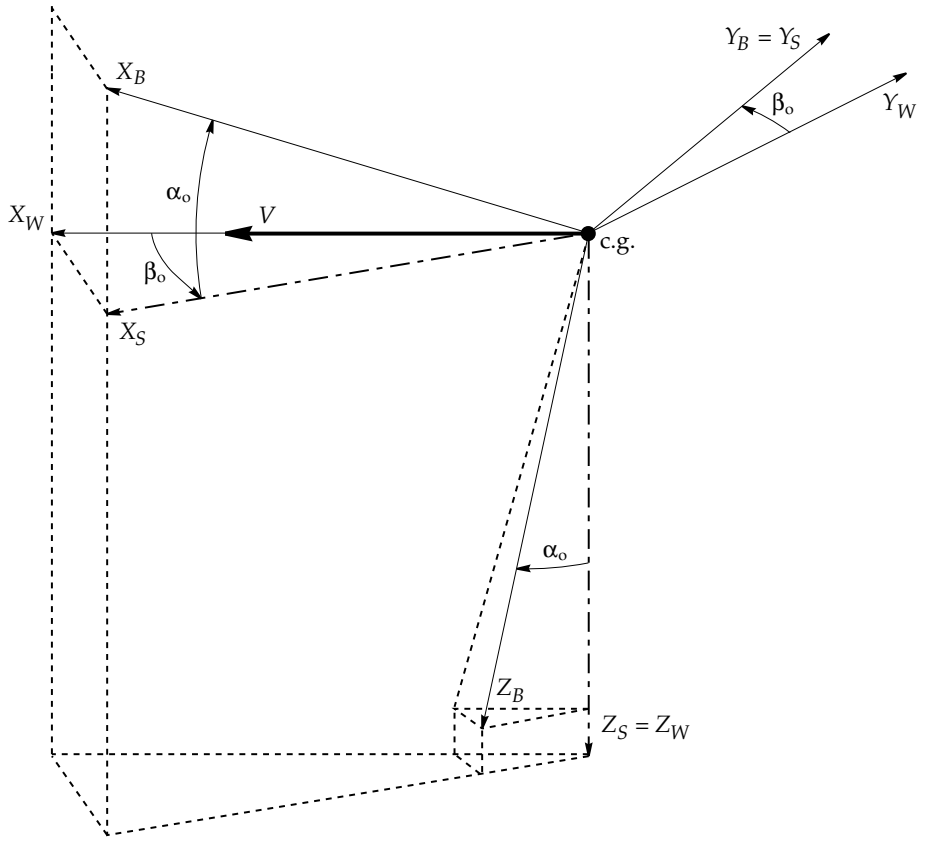


Figure A.5: Relationship between the body-fixed reference frame F_B, flight-path reference frame F_W and stability reference frame F_S

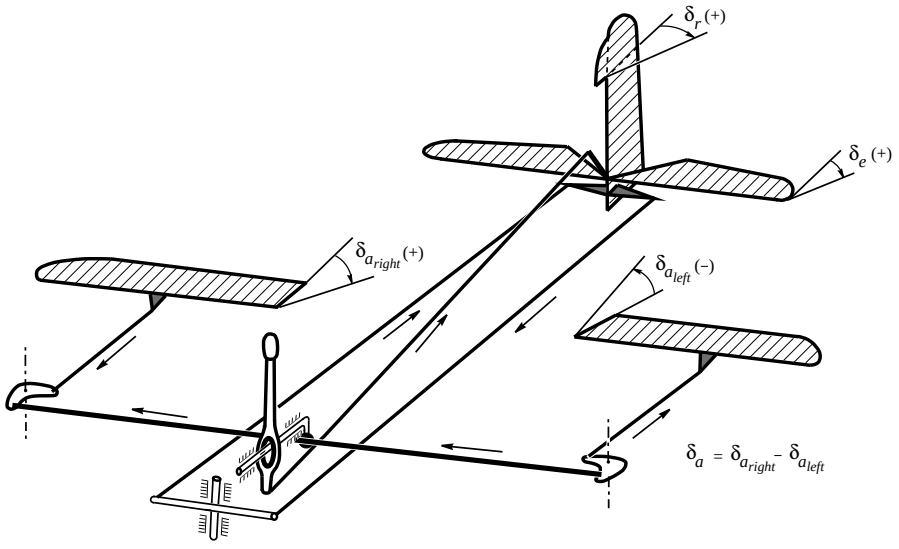


Figure A.6: Sign conventions for control surface deflections

Appendix B

General rigid-body equations of motion

In this appendix, the derivation of the rigid-body equations of Newton will be given. This appendix has been included since a good understanding of Newtonian mechanics is very important for any discussion about the derivation of the equations of motion of an aircraft. One should particularly take notice of the assumptions which shall be made during this derivation. For more details regarding the equations of motion the reader is referred to refs.[9], [10], [15], [19], or [20] (to name just a few).

B.1 Linear and rotational velocity equations in body-axes

B.1.1 General force equation for a rigid body

Consider a mass point δm that moves with time-varying velocity $\mathbf{V}$ under the influence of a force $\mathbf{F}$. Both $\mathbf{V}$ and $\mathbf{F}$ are measured relatively to a right-handed orthogonal reference frame OXYZ. Applying Newton's second law yields:

$$\delta\mathbf{F} = \delta m \cdot \dot{\mathbf{V}} \tag{B.1}$$

Applying this equation to all mass points of a rigid body and summing all contributions across this body yields:

$$\sum \delta\mathbf{F} = \sum \delta m \frac{d\mathbf{V}}{dt} = \frac{d}{dt}\sum \mathbf{V}\delta m \tag{B.2}$$

Let the center of gravity of the rigid body have a velocity $\mathbf{V}_{c.g.}$ with components u, v, and w along the X, Y, and Z-axes of the right-handed reference frame. The velocity of each mass point within the rigid body then equals the sum of $\mathbf{V}_{c.g.}$ and the velocity of the mass point with respect to this center of gravity. If the position of the mass point with respect to the c.g. is denoted by the vector $\mathbf{r}$, the following vector equation is found:

$$\mathbf{V} = \mathbf{V}_{c.g.} + \dot{\mathbf{r}} \tag{B.3}$$

therefore:

$$\sum \mathbf{V}\delta m = \sum(\mathbf{V}_{c.g.} + \dot{\mathbf{r}})\delta m = m\mathbf{V}_{c.g.} + \frac{d}{dt}\sum \mathbf{r}\delta m \tag{B.4}$$

In this equation, m denotes the total mass of the rigid body. In the center of gravity we can write:

$$\sum \mathbf{r}\,\delta m = 0 \tag{B.5}$$

so the equation for the total force $\mathbf{F}$ acting upon the rigid body, becomes:

$$\mathbf{F} = m\dot{\mathbf{V}}_{c.g.} \tag{B.6}$$

B.1.2 General moment equation for a rigid body

The moment $\delta\mathbf{M}$, measured about the center of gravity, is equal to the time-derivative of the angular momentum of the mass point relative to the c.g.:

$$\delta\mathbf{M} = \frac{d}{dt}(\mathbf{r} \times \mathbf{V})\delta m = (\dot{\mathbf{r}} \times \mathbf{V})\delta m + (\mathbf{r} \times \dot{\mathbf{V}})\delta m \tag{B.7}$$

where:

$$\dot{\mathbf{r}} = \mathbf{V} - \mathbf{V}_{c.g.} \tag{B.8}$$

and:

$$(\mathbf{r} \times \dot{\mathbf{V}})\delta m = \mathbf{r} \times \delta\mathbf{F} = \delta\mathbf{M}_{c.g.} \tag{B.9}$$

In this equation $\delta\mathbf{M}_{c.g.}$ denotes the moment of the force $\delta\mathbf{F}$ about the center of gravity. The angular momentum of the mass point relative to the c.g. will be denoted by $\delta\mathbf{h}$, which is defined as: $\delta\mathbf{h} \equiv (\mathbf{r} \times \mathbf{V})\delta m$. Writing this out yields:

$$\delta\mathbf{M}_{c.g.} = \delta\dot{\mathbf{h}} - (\mathbf{V} - \mathbf{V}_{c.g.}) \times \delta m = \delta\dot{\mathbf{h}} + \mathbf{V}_{c.g.} \times \mathbf{V}\delta m \tag{B.10}$$

The contributions of all mass points are once again summed across the whole rigid body, yielding:

$$\sum \delta\mathbf{M}_{c.g.} = \frac{d}{dt}\sum \delta\dot{\mathbf{h}} + \mathbf{V}_{c.g.} \times \sum \mathbf{V}\delta m \tag{B.11}$$

The equation for the resulting moment $\mathbf{M}_{c.g.}$ about the c.g. then becomes:

$$\mathbf{M}_{c.g.} = \dot{\mathbf{h}} \tag{B.12}$$

where $\mathbf{h}$ denotes the resulting angular momentum of the body about the center of gravity.

B.1.3 Angular momentum around the center of gravity

Consider a rigid body with angular velocity $\boldsymbol{\Omega}$, with components p, q, and r about the X, Y, and Z axes of the right-handed reference frame respectively:

$$\boldsymbol{\Omega} = \mathbf{i}\,p + \mathbf{j}\,q + \mathbf{k}\,r \tag{B.13}$$

where $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are unity vectors along the X, Y, and Z-axes. The total velocity vector of a mass point of a rigid body that both translates and rotates becomes:

$$\mathbf{V} = \mathbf{V}_{c.g.} + \boldsymbol{\Omega} \times \mathbf{r} \tag{B.14}$$

hence, the angular momentum of the rigid body about the c.g. can be written as:

$$\mathbf{h} \equiv \sum \delta\mathbf{h} = \sum \mathbf{r} \times (\mathbf{V}_{c.g.} + \boldsymbol{\Omega} \times \mathbf{r})\delta m = \sum \mathbf{r} \times \mathbf{V}_{c.g.}\delta m + \sum \mathbf{r} \times (\boldsymbol{\Omega} \times \mathbf{r})\delta m \tag{B.15}$$

The first term of the right hand side of equation (B.15) is equal to zero:

$$\left(\sum \mathbf{r}\delta m\right) \times \mathbf{V}_{c.g.} = 0 \tag{B.16}$$

and for the second term we can write:

$$\sum \mathbf{r} \times (\boldsymbol{\Omega} \times \mathbf{r})\delta m = \sum\{\boldsymbol{\Omega}(\mathbf{r} \cdot \mathbf{r}) - \mathbf{r}(\boldsymbol{\Omega} \cdot \mathbf{r})\}\delta m = \sum\{\boldsymbol{\Omega}\|\,\mathbf{r}\,\|^2 - \mathbf{r}\,(\boldsymbol{\Omega} \cdot \mathbf{r})\}\delta m \tag{B.17}$$

Substitution of $\mathbf{r} = \mathbf{i}\,x + \mathbf{j}\,y + \mathbf{k}\,z$, (B.16), and (B.17) in equation (B.15) yields:

$$\mathbf{h} = \boldsymbol{\Omega}\sum(x^2 + y^2 + z^2)\delta m - \sum \mathbf{r}\,(px + qy + rz)\delta m \tag{B.18}$$

symbol	definition
I_{xx}	$\sum(y^2+z^2)\,\delta m$
I_{yy}	$\sum(x^2+z^2)\,\delta m$
I_{zz}	$\sum(x^2+y^2)\,\delta m$
J_{xy}	$\sum xy\;\delta m$
J_{xz}	$\sum xz\;\delta m$
J_{yz}	$\sum yz\;\delta m$

Table B.1: Moments and products of inertia

The components of $\mathbf{h}$ along the X, Y, and Z axes will be denoted as h_x, h_y, and h_z respectively, yielding:

$$\begin{aligned} h_x &= p\sum(y^2+z^2)\delta m - q\sum xy\,\delta m - r\sum xz\,\delta m \\ h_y &= -p\sum xy\,\delta m + q\sum(x^2+z^2)\delta m - r\sum yz\,\delta m \\ h_z &= -p\sum xz\,\delta m - q\sum yz\,\delta m + r\sum(x^2+y^2)\delta m \end{aligned} \tag{B.19}$$

The summations appearing in these equations are defined as the inertial moments and products about the X, Y, and Z axes respectively; see table B.1.[1] Using these definitions, equations (B.19) can be written in vector notation as a product of the inertia tensor $\mathbf{I}$ with the angular velocity vector $\mathbf{\Omega}$:

$$\mathbf{h} = \mathbf{I}\cdot\mathbf{\Omega} \tag{B.20}$$

where $\mathbf{I}$ is defined as:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -J_{xy} & -J_{xz} \\ -J_{yx} & I_{yy} & -J_{yz} \\ -J_{zx} & -J_{zy} & I_{zz} \end{bmatrix} \tag{B.21}$$

B.1.4 General equations of motion for a rigid body

When we choose a reference frame fixed to the body ($OXYZ = OX_BY_BZ_B$) the inertial moments and products from the equations (B.19) become constants. The reference frame itself then rotates with angular velocity $\mathbf{\Omega}$. For an arbitrary position vector $\mathbf{r}$ with respect to the body reference frame we can then write:

$$\dot{\mathbf{r}} = \frac{\partial \mathbf{r}}{\partial t} + \mathbf{\Omega}\times\mathbf{r} \tag{B.22}$$

Applying equation (B.22) to the general force and moment equations (B.6) and (B.12), we find:

$$\mathbf{F} = m\left(\frac{\partial \mathbf{V}_{c.g.}}{\partial t} + \mathbf{\Omega}\times\mathbf{V}_{c.g.}\right) \tag{B.23}$$

and:

$$\mathbf{M}_{c.g.} = \frac{\partial \mathbf{h}}{\partial t} + \mathbf{\Omega}\times\mathbf{h} = \frac{\partial(\mathbf{I}\cdot\mathbf{\Omega})}{\partial t} + \mathbf{\Omega}\times(\mathbf{I}\cdot\mathbf{\Omega}) \tag{B.24}$$

These two vector-equations form the basis for the development of the general rigid-body dynamic model used in the FDC toolbox. The linear and angular accelerations can be moved to the left

[1] The summations across the body actually have to be written as integrals, but that further refinement has been omitted here.

hand side of equations (B.23) and (B.24), yielding:

$$\frac{\partial \mathbf{V}_{c.g.}}{\partial t} = \frac{\mathbf{F}}{m} - \mathbf{\Omega} \times \mathbf{V}_{c.g.} \tag{B.25}$$

$$\frac{\partial \mathbf{\Omega}}{\partial t} = \mathbf{I}^{-1} \left(\mathbf{M} - \mathbf{\Omega} \times \mathbf{I} \cdot \mathbf{\Omega} \right) \tag{B.26}$$

These equations can be written-out into their components along the body-axes, yielding:

$$\begin{aligned} \dot{u} &= \frac{F_x}{m} - qw + rv \\ \dot{v} &= \frac{F_y}{m} + pw - ru \\ \dot{w} &= \frac{F_z}{m} - pv + qu \end{aligned} \tag{B.27}$$

and:

$$\begin{aligned} \dot{p} &= P_{pp}\, p^2 + P_{pq}\, pq + P_{pr}\, pr + P_{qq}\, q^2 + P_{qr}\, qr + P_{rr}\, r^2 + P_l\, L + P_m\, M + P_n\, N \\ \dot{q} &= Q_{pp}\, p^2 + Q_{pq}\, pq + Q_{pr}\, pr + Q_{qq}\, q^2 + Q_{qr}\, qr + Q_{rr}\, r^2 + Q_l\, L + Q_m\, M + Q_n\, N \\ \dot{r} &= R_{pp}\, p^2 + R_{pq}\, pq + R_{pr}\, pr + R_{qq}\, q^2 + R_{qr}\, qr + R_{rr}\, r^2 + R_l\, L + R_m\, M + R_n\, N \end{aligned} \tag{B.28}$$

P_{pp}, P_{pq}, ... R_n are inertia coefficients derived from the matrix multiplications involving the inertia tensor $\mathbf{I}$; they have been listed in table B.2. $\mathbf{V}_{c.g.}$, $\mathbf{\Omega}$, $\mathbf{F}$, and $\mathbf{M}$ are defined as:

$$\begin{aligned} \mathbf{V}_{c.g.} &= \mathbf{i}\, u + \mathbf{j}\, v + \mathbf{k}\, w \\ \mathbf{\Omega} &= \mathbf{i}\, p + \mathbf{j}\, q + \mathbf{k}\, r \\ \mathbf{F} &= \mathbf{i} F_x + \mathbf{j} F_y + \mathbf{k} F_z \\ \mathbf{M} &= \mathbf{i} L + \mathbf{j} M + \mathbf{k} N \end{aligned}$$

These equations describe the motions of any rigid body relatively to the Earth under the following restrictive assumptions:

1. the body is assumed to be rigid during the motions considered,
2. the mass of the body is assumed to be constant during the time-interval in which its motions are studied,
3. the Earth is assumed to be fixed in space, i.e. its rotation is neglected,
4. the curvature of the Earth is neglected.

The latter two assumptions were made in the definition of the inertial reference frame in which the motions of the rigid body are considered. If the equations are to be applied to a moving vehicle, the description of the vehicle motion under assumptions 3 and 4 are accurate for relatively short-term guidance and control analysis purposes only. The assumptions do have practical limitations when very long term navigation or extra-atmospheric operations are of interest, see ref [20].

The forces and moments along the body-axes of the rigid body can be separated into different components. For an aircraft, the most important contributions are the gravity forces, aerodynamic forces and moments, and propulsive forces and moments. Sometimes other contributions must be taken into account, e.g. ground-forces which are encountered when the aircraft is taxiing. Those additional components will be disregarded throughout this report, i.e. the model from this report will represent an *in-flight* model of the aircraft. Therefore we can write:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} X_{gr} + X_a + X_p \\ Y_{gr} + Y_a + Y_p \\ Z_{gr} + Z_a + Z_p \end{bmatrix} \tag{B.29}$$

symbol	definition
$\|I\|$	$I_{xx}I_{yy}I_{zz} - 2J_{xy}J_{xz}J_{yz} - I_{xx}{J_{yz}}^2 - I_{yy}{J_{xz}}^2 - I_{zz}{J_{xy}}^2$
I_1	$I_{yy}I_{zz} - {J_{yz}}^2$
I_2	$J_{xy}I_{zz} + J_{yz}J_{xz}$
I_3	$J_{xy}J_{yz} + I_{yy}J_{xz}$
I_4	$I_{xx}I_{zz} - {J_{xz}}^2$
I_5	$I_{xx}J_{yz} + J_{xy}J_{xz}$
I_6	$I_{xx}I_{yy} - {J_{xy}}^2$
P_l	$I_1 \ / \ \|I\|$
P_m	$I_2 \ / \ \|I\|$
P_n	$I_3 \ / \ \|I\|$
P_{pp}	$-(J_{xz}I_2 - J_{xy}I_3) \ / \ \|I\|$
P_{pq}	$(J_{xz}I_1 - J_{yz}I_2 - (I_{yy} - I_{xx})I_3) \ / \ \|I\|$
P_{pr}	$-(J_{xy}I_1 + (I_{xx} - I_{zz})I_2 - J_{yz}I_3) \ / \ \|I\|$
P_{qq}	$(J_{yz}I_1 - J_{xy}I_3) \ / \ \|I\|$
P_{qr}	$-((I_{zz} - I_{yy})I_1 - J_{xy}I_2 + J_{xz}I_3) \ / \ \|I\|$
P_{rr}	$-(J_{yz}I_1 - J_{xz}I_2) \ / \ \|I\|$
Q_l	$I_2 \ / \ \|I\|$
Q_m	$I_4 \ / \ \|I\|$
Q_n	$I_5 \ / \ \|I\|$
Q_{pp}	$-(J_{xz}I_4 - J_{xy}I_5) \ / \ \|I\|$
Q_{pq}	$(J_{xz}I_2 - J_{yz}I_4 - (I_{yy} - I_{xx})I_5) \ / \ \|I\|$
Q_{pr}	$-(J_{xy}I_2 + (I_{xx} - I_{zz})I_4 - J_{yz}I_5) \ / \ \|I\|$
Q_{qq}	$(J_{yz}I_2 - J_{xy}I_5) \ / \ \|I\|$
Q_{qr}	$-((I_{zz} - I_{yy})I_2 - J_{xy}I_4 + J_{xz}I_5) \ / \ \|I\|$
Q_{rr}	$-(J_{yz}I_2 - J_{xz}I_4) \ / \ \|I\|$
R_l	$I_3 \ / \ \|I\|$
R_m	$I_5 \ / \ \|I\|$
R_n	$I_6 \ / \ \|I\|$
R_{pp}	$-(J_{xz}I_5 - J_{xy}I_6) \ / \ \|I\|$
R_{pq}	$(J_{xz}I_3 - J_{yz}I_5 - (I_{yy} - I_{xx})I_6) \ / \ \|I\|$
R_{pr}	$-(J_{xy}I_3 + (I_{xx} - I_{zz})I_5 - J_{yz}I_6) \ / \ \|I\|$
R_{qq}	$(J_{yz}I_3 - J_{xy}I_6) \ / \ \|I\|$
R_{qr}	$-((I_{zz} - I_{yy})I_3 - J_{xy}I_5 + J_{xz}I_6) \ / \ \|I\|$
R_{rr}	$-(J_{yz}I_3 - J_{xz}I_5) \ / \ \|I\|$

Table B.2: Definition of inertia coefficients

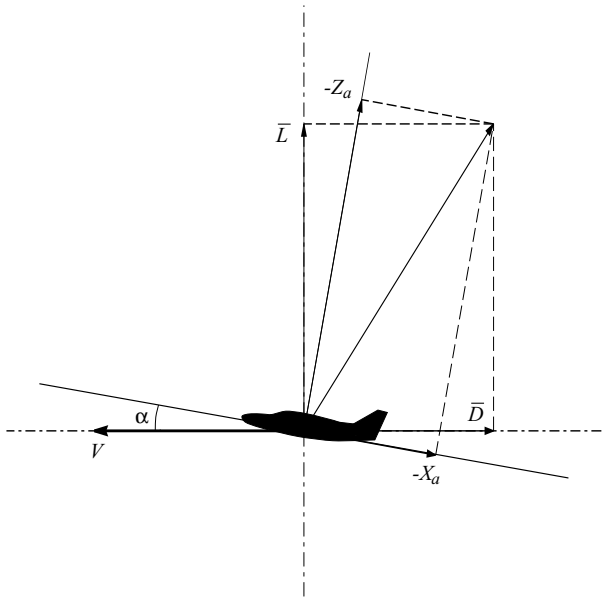


Figure B.1: Relationship between aerodynamic forces in flight-path and body-axes

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} L_a & + & L_p \\ M_a & + & M_p \\ N_a & + & N_p \end{bmatrix} \tag{B.30}$$

The index gr denotes gravity, a denotes aerodynamics, and p denotes propulsion. In section B.3, a fourth element will be added to the body-axes forces in order to account for a non-steady atmosphere.

In order to simplify the notations for the remainder of this appendix and the other chapters in this report, the velocity vector $\mathbf{V}_{c.g.}$ will simply be denoted as $\mathbf{V}$. The body-axes components of this vector are u, v, and w, respectively, and the length of this vector is denoted as V.

B.2 Using flight-path axes for describing linear motions

B.2.1 Why flight-path axes?

In aerodynamic problems it is more convenient to use the true airspeed V, angle of attack α, and sideslip angle β in stead of the linear velocity components u, v, and w along the body-axes of the aircraft. Since V, α, and β can be described in terms of u, v, and w, and vice-versa, both sets of variables can be applied for solving the equations of motion, i.e. both sets can be used as *state variables* for our rigid-body model. In practice, the flight-path variables V, α, and β are best suited for simulation purposes for two reasons:

1. From a physical point of view it is logical to express the aerodynamic forces and moments in terms of V, α, and β. For simulation purposes it is required to write the linear force equations as a set of explicit Ordinary Differential Equations, i.e. all time-derivatives should be put on one side of the equations and all other terms on the other side. This may be difficult to achieve since the aerodynamic forces and moments may depend upon $\dot{\alpha}$ and $\dot{\beta}$ while $\dot{\alpha}$ and $\dot{\beta}$ themselves are not available until *after* the force equations have

been evaluated. In practice it is customary to assume a linear relation between these time-derivatives and the aerodynamic forces, which makes it relatively simple to convert the force equations to explicit ODE's (this has been demonstrated in section 3.2.3 for the 'Beaver'), provided the equations are written in terms of α and β in stead of v and w.

2. It is better to use the flight-path variables in order to obtain a higher accuracy for the numerical computations. For agile aircraft which have an upper limit of the pitch rate q of about 2 $rad\ s^{-1}$ and which fly at high airspeeds (e.g. $V = 600\ ms^{-1}$), the term $u\,q$ in equation (B.28) may become as large as 120 g! On the other hand, the factor F_z/m, which represents the normal acceleration due to the external force along the Z_B-axis (primarily gravity and aerodynamic lift) has an upper-limit of only a few g's. Hence, artificial accelerations of much greater magnitude than the actual accelerations of the aircraft are introduced in the equations for u, v, and w, because of the high rotation rates of the body-axes. In practice this means less favorable computer scaling and hence poorer accuracy for a given computer precision if the simulation model is based upon u, v, and w in stead of V, α, and β (ref.[11]).

B.2.2 Transforming forces and velocities from body to flight-path axes

The following derivation of the V, α, and β-equations is largely based upon ref.[9]. The body-axes velocity components are equal to:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = V \begin{bmatrix} \cos\alpha\cos\beta \\ \sin\beta \\ \sin\alpha\cos\beta \end{bmatrix} \tag{B.31}$$

Hence:

$$V = \sqrt{u^2 + v^2 + w^2} \tag{B.32}$$

$$\alpha = \arctan\left(\frac{w}{u}\right) \tag{B.33}$$

$$\beta = \arctan\left(\frac{v}{\sqrt{u^2 + w^2}}\right) \tag{B.34}$$

Sometimes the aerodynamic forces and moments are expressed in terms of aerodynamic lift $\overline{L}$, drag $\overline{D}$, and sideforce $\overline{Y}$ in stead of the body-axes force-components X_a, Y_a, and Z_a. In that case it is necessary to apply the proper axis-transformation to these forces (see figure B.1):

$$\begin{bmatrix} X_a \\ Y_a \\ -Z_a \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} \overline{D} \\ \overline{Y} \\ \overline{L} \end{bmatrix} \tag{B.35}$$

Notice the minus sign for the aerodynamic force component along the Z_B-axis, which is due to the fact that the positive Z_B-axis points *downwards*. See also section A.7.2 in appendix A.

B.2.3 Derivation of the $\dot{V}$-equation

From equation (B.32) it can be deduced that:

$$\dot{V} = \frac{u\dot{u} + v\dot{v} + w\dot{w}}{V} \tag{B.36}$$

Substituting the definitions (B.31) for u, v, and w, and canceling terms yields:

$$\dot{V} = \dot{u}\cos\alpha\cos\beta + \dot{v}\sin\beta + \dot{w}\sin\alpha\cos\beta \tag{B.37}$$

If we substitute equations (B.28) for $\dot{u}$, $\dot{v}$, and $\dot{w}$, the terms involving the vehicle rotational rates p, q, and r turn out to be identically zero and the resulting equation becomes:

$$\dot{V} = \frac{1}{m}\left(F_x \cos\alpha \cos\beta + F_y \sin\beta + F_z \sin\alpha \sin\beta\right) \tag{B.38}$$

B.2.4 Derivation of the $\dot{\alpha}$-equation

Differentiating equation (B.33) with respect to the time yields:

$$\dot{\alpha} = \frac{u\dot{w} - \dot{u}w}{u^2 + w^2} \tag{B.39}$$

Using equation (B.31) we can re-write the denominator of this equation:

$$u^2 + w^2 = V^2 - v^2 = V^2(1 - \sin^2\beta) = V^2 \cos^2\beta \tag{B.40}$$

Substituting the u and w-relations from equation (B.31) and equation (B.40) into equation (B.39) yields:

$$\dot{\alpha} = \frac{\dot{w}\cos\alpha - \dot{u}\sin\alpha}{V\cos\beta} \tag{B.41}$$

Substituting for $\dot{u}$ and $\dot{w}$ (see equations (B.28)) and rewriting terms yields:

$$\dot{\alpha} = \frac{1}{V\cos\beta}\left\{\frac{1}{m}(-F_x \sin\alpha + F_z \cos\alpha) + pv\cos\alpha + qu\cos\alpha + qw\sin\alpha - rv\sin\alpha\right\} \tag{B.42}$$

Using equations (B.31) for u, v, and w, we find:

$$\dot{\alpha} = \frac{1}{V\cos\beta}\left\{\frac{1}{m}\left(-F_x \sin\alpha + F_z \cos\alpha\right)\right\} + q - (p\cos\alpha + r\sin\alpha)\tan\beta \tag{B.43}$$

B.2.5 Derivation of the $\dot{\beta}$-equation

Differentiating equation (B.34) with respect to the time yields:

$$\dot{\beta} = \frac{\dot{v}(u^2 + v^2) - v(u\dot{u} + w\dot{w})}{V^2\sqrt{u^2 + w^2}} \tag{B.44}$$

From equations (B.31) the following relations can be derived:

$$\begin{aligned} u^2 + w^2 &= V^2 \cos^2\beta \\ uv &= V^2 \sin\beta\cos\beta\cos\alpha \\ vw &= V^2 \sin\beta\cos\beta\sin\alpha \end{aligned} \tag{B.45}$$

These values substituted in equation (B.44) yield:

$$\dot{\beta} = \frac{1}{V}(-\dot{u}\cos\alpha\sin\beta + \dot{v}\cos\beta - \dot{w}\sin\alpha\sin\beta) \tag{B.46}$$

Substituting for $\dot{u}$ and $\dot{w}$ (see equations (B.28)) yields:

$$\begin{aligned} \dot{\beta} \;=\; & \frac{1}{V}\left\{\frac{1}{m}\left(-F_x\cos\alpha\sin\beta + F_y\cos\beta - F_z\sin\alpha\sin\beta\right) + qw\cos\alpha\sin\beta + \right. \\ & \left. -\; rv\cos\alpha\sin\beta + pw\cos\beta - ru\cos\beta + pv\sin\alpha\sin\beta - qu\sin\alpha\sin\beta\right\} \end{aligned} \tag{B.47}$$

If we substitute equations (B.31), many terms can be canceled and we find:

$$\dot{\beta} = \frac{1}{V}\left\{\frac{1}{m}\left(-F_x\cos\alpha\sin\beta + F_y\cos\beta - F_z\sin\alpha\sin\beta\right)\right\} + p\sin\alpha - r\cos\alpha \tag{B.48}$$

B.3 Equations of motion in non-steady atmosphere

The equations of motion are valid only if the body-axes velocity components are measured with respect to a non-rotating system of reference axes having a constant translational speed in inertial space. Under the assumptions 3 and 4, mentioned in section B.1.4, it is possible to select a reference frame that is fixed to the surrounding atmosphere as long as the wind velocity vector $\mathbf{V}_w$ is constant. In that case, the components u, v, and w of the velocity vector $\mathbf{V}$ express the aircraft's velocity with respect to the surrounding atmosphere. If the wind velocity vector $\mathbf{V}_w$ is not constant during the time-interval over which the motions of the aircraft are studied it is not possible to fix the reference frame to the surrounding atmosphere. This happens for instance during the approach and landing of an aircraft, because the wind velocity changes with altitude. Again using assumptions 3 and 4 of section B.1.4, the most obvious choice of the reference frame in this case turns out to be the Earth-fixed reference frame F_E (ref.[14]).

In the subsequent part of this section the subscripts a and e will be used to denote velocities with respect to the surrounding atmosphere and the Earth, respectively. We can write:

$$\mathbf{V}_e = \mathbf{V}_a + \mathbf{V}_w \tag{B.49}$$

or:

$$\begin{aligned} u_e &= u_a + u_w \\ v_e &= v_a + v_w \\ w_e &= w_a + w_w \end{aligned} \tag{B.50}$$

where u_a, v_a, and w_a are the body-axes components of $\mathbf{V}_a$, u_e, v_e, and w_e are the body-axes components of $\mathbf{V}_e$, and u_w, v_w, and w_w are the body-axes components of $\mathbf{V}_w$ along the body-axes of the aircraft. The force equations now become:

$$\mathbf{F} = m\left(\frac{\partial \mathbf{V}_e}{\partial t} + \mathbf{\Omega} \times \mathbf{V}_e\right) \tag{B.51}$$

In order to compute the aerodynamic forces and moments, it is necessary to know the values of V_a (the true airspeed), α, and β.[1] Rewriting equation (B.51) yields:

$$\frac{\partial \mathbf{V}_e}{\partial t} = \frac{\mathbf{F}}{m} - \mathbf{\Omega} \times \mathbf{V}_e \tag{B.52}$$

For the individual components along the body-axes we thus find:

$$\begin{aligned} \dot{u}_e &= \frac{F_x}{m} - qw_e + rv_e \\ \dot{v}_e &= \frac{F_y}{m} + pw_e - ru_e \\ \dot{w}_e &= \frac{F_z}{m} - pv_e + qu_e \end{aligned} \tag{B.53}$$

In a manner analogous to sections B.2.3, B.2.4, and B.2.5 expressions for the time-derivatives of V, α, and β can be found:

$$\begin{aligned} \dot{V}_a = {} & \frac{1}{m}\left(F_x \cos\alpha\cos\beta + F_y \sin\beta + F_z \sin\alpha\sin\beta\right) + \\ & - \left(qw_w - rv_w + \dot{u}_w\right)\cos\alpha\cos\beta + \left(pw_w - ru_w - \dot{v}_w\right)\sin\beta + \\ & - \left(pv_w - qu_w + \dot{w}_w\right)\sin\alpha\cos\beta \end{aligned} \tag{B.54}$$

[1] Notice that expressions (B.31) to (B.34) remain valid if V_a is substituted for V.

$$\begin{aligned}
\dot{\alpha} &= \frac{1}{V\cos\beta}\left\{\frac{1}{m}\left(-F_x\sin\alpha + F_z\cos\alpha\right) + \right. \\
&\quad \left. - \left(pv_w - qu_w + \dot{w}_w\right)\cos\alpha + \left(qw_w - rv_w - \dot{u}_w\right)\sin\alpha\right\} + \\
&\quad + q - \left(p\cos\alpha + r\sin\alpha\right)\tan\beta \qquad \text{(B.55)} \\
\dot{\beta} &= \frac{1}{V}\left\{\frac{1}{m}\left(-F_x\cos\alpha\sin\beta + F_y\cos\beta - F_z\sin\alpha\sin\beta\right) + \right. \\
&\quad + \left(qw_w - rv_w + \dot{u}_w\right)\cos\alpha\sin\beta + \left(pw_w - ru_w - \dot{v}_w\right)\cos\beta + \\
&\quad \left. + \left(pv_w - qu_w + \dot{w}_w\right)\sin\alpha\sin\beta\right\} + p\sin\alpha - r\cos\alpha \qquad \text{(B.56)}
\end{aligned}$$

The differences between these expressions and equations (B.38), (B.43), and (B.48) can be modeled by adding a 'wind component' to the forces along the aircraft's body-axes. The resulting force components along body-axes then become:

$$\begin{array}{rcllll}
F_x &=& X_a &+ X_p &+ X_{gr} &+ X_w \\
F_y &=& Y_a &+ Y_p &+ Y_{gr} &+ Y_w \\
F_z &=& Z_a &+ Z_p &+ Z_{gr} &+ Z_w
\end{array} \qquad \text{(B.57)}$$

where X_w, Y_w, and Z_w represent *corrections* to the body-axes forces due to non-steady atmosphere, determined by the following equations:

$$\begin{aligned}
X_w &= -m\left(\dot{u}_w + qw_w - rv_w\right) \\
Y_w &= -m\left(\dot{v}_w - pw_w + ru_w\right) \qquad \text{(B.58)} \\
Z_w &= -m\left(\dot{w}_w + pv_w - qu_w\right)
\end{aligned}$$

Due to these additional force components the responses of V, α, and β in non-steady atmosphere differ from the responses in steady atmosphere. The aerodynamic forces and moments are functions of V, α, and β, which implies that these forces and moments also differ from the results that would have been obtained in steady atmosphere. If the wind velocity or direction changes quickly, for instance in atmospheric turbulence, the aerodynamic model itself will need to be enhanced with additional terms to account for aerodynamic lags, e.g. the *gust penetration effect*, which is caused by the finite dimensions of the aircraft. This effect is described and modeled for linearized aircraft models in ref.[21]. It will not be taken into account in this report, however.[1] In section 3.3 some common methods to model the atmospheric turbulence velocity components along the body-axes of the aircraft are outlined. This section also describes the transformation from Earth to body-axes which is necessary to convert wind given with respect to the Earth to wind velocity components along the aircraft's body-axes.

Corrections (B.59) again contain terms involving the vehicle angular velocities p, q, and r. Unlike the term $u\,q$, the magnitude of $u_w\,q$ will not become very large when compared to the normal acceleration F_z/m, because the maximum wind velocity usually is relatively small, compared to the airspeed. This is especially true for cases where the aircraft is manoeuvring

[1] In ref.[21] the responses of an aircraft to atmospheric turbulence are modeled by adding 'gust corrections' to u, α, and β. This means that α and β themselves are no longer measured relatively to the surrounding atmosphere, as was defined within this report. Furthermore, different editions of ref.[21] having different equations for modeling the gust penetration effect have been published, but it is not clear which version is the best. Modeling the gust penetration effect requires knowledge about contributions of certain specific parts of the airframe to the stability derivatives; knowledge which is not readily available for the 'Beaver' aircraft. It is possible to approximate these contributions, but that introduces errors. In addition, the expressions from ref.[21] have to be adapted for the non-linear equations used in this report. For these reasons the modeling of turbulence responses in the fashion of ref.[21] was considered to be beyond the scope of this report. But even with this limitation of the aerodynamic model taken into account, the general equations of motion for non-steady atmosphere derived in this section are still valid and the responses of V, α, and β still implicitly affect the aerodynamic forces and moments.

heavily, yielding large values of q, e.g. in case of dog-fights between fighter aircraft or aerobatic manoeuvres. Large wind velocities are common at higher altitudes, but there manoeuvring is limited. Hence, expressions (B.59) will not give any problems due to computer precision, as discussed in section B.2.

B.4 Kinematic relations

So far we have derived differential equations for the true airspeed, angle of attack, sideslip angle, and the rotational velocity components. However, to solve the equations of motion it is also necessary to know the attitude of the aircraft relatively to the Earth and the altitude of the aircraft, because some contributions to the external forces and moments depend upon those quantities. Moreover, it is useful to know the coordinates of the aircraft with respect to the Earth-fixed reference frame in order to be able to simulate navigational tasks. The attitude of the aircraft with respect to the Earth is defined by the Euler angles ψ, θ, and φ, see figure A.3 in appendix A. The kinematic relations which determine the time-derivatives of these Euler angles are given by the following formulas:

$$\begin{aligned} \dot{\psi} &= \frac{q\sin\varphi + r\cos\varphi}{\cos\theta} \\ \dot{\theta} &= q\cos\varphi - r\sin\varphi \\ \dot{\varphi} &= p + (q\sin\varphi + r\cos\varphi)\tan\theta = p + \dot{\psi}\sin\theta \end{aligned} \tag{B.59}$$

The position of the aircraft with respect to the Earth-fixed reference frame is given by the coordinates x_e, y_e, and z_e, defined by the following equations:

$$\begin{aligned} \dot{x}_e &= \{u_e\cos\theta + (v_e\sin\varphi + w_e\cos\varphi)\sin\theta\}\cos\psi - (v_e\cos\varphi - w_e\sin\varphi)\sin\psi \\ \dot{y}_e &= \{u_e\cos\theta + (v_e\sin\varphi + w_e\cos\varphi)\sin\theta\}\sin\psi + (v_e\cos\varphi - w_e\sin\varphi)\cos\psi \\ \dot{z}_e &= -u_e\sin\theta + (v_e\sin\varphi + w_e\cos\varphi)\cos\theta \end{aligned} \tag{B.60}$$

These equations are a result of the following transformation:

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{z}_e \end{bmatrix} = T_{B\to E} \cdot \begin{bmatrix} u_e \\ v_e \\ w_e \end{bmatrix} \tag{B.61}$$

where $T_{B\to E} = T_{B\to V} = {T_{V\to B}}^{-1}$ is the transformation matrix from F_B to F_E, see the definition in section A.7.2 of appendix A. Often, the altitude of the aircraft is used in stead of the coordinate z_e. The relationship between the time-derivatives of H and z_e is simple:

$$\dot{H} = -\dot{z}_e \tag{B.62}$$

Note: the positive Z_E-axis points *downwards*.

B.5 Resulting dynamic model

In the previous sections we have derived *twelve* scalar differential equations, namely: three force equations, three moment equations, and six kinematic relations. These equations can be combined in *one* non-linear vector equation:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{F}_{tot}(t), \mathbf{M}_{tot}(t)\right) \tag{B.63}$$

with $\mathbf{x}$ defined as:

$$\mathbf{x} = [\ V\ \ \alpha\ \ \beta\ \ p\ \ q\ \ r\ \ \psi\ \ \theta\ \ \varphi\ \ x_e\ \ y_e\ \ H\ ]^T \tag{B.64}$$

Equation (B.63) represents the resulting dynamic model of the rigid body. It has been elaborated further in chapter 3.

Appendix C

Definition of the parameters of the 'Beaver' model

This appendix contains the definitions of the parameters used by the non-linear dynamical model of the 'Beaver' and it lists some general data of this aircraft. The dynamical model itself has been described in chapter 3. Appendix D will show how these parameters were implemented in the FDC toolbox itself.

Manufacturer	De Havilland Aircraft of Canada Ltd.
Serial no.	1244
Type of aircraft	Single engine, high-wing, seven seat, all-metal aircraft
Wing span b	14.63 m
Wing area S	23.23 m^2
Mean aerodynamic chord $\overline{c}$	1.5875 m
Wing sweep	0°
Wing dihedral	1°
Wing profile	NACA 64 A 416
Fuselage length	9.22 m
Max. take-off weight	22800 N
Empty weight	14970 N
Engine	Pratt and Whitney Wasp Jr. R-985
Max. power	450 Hp at $n = 2300$ RPM, $p_z = 26'' Hg$
Propeller	Hamilton Standard, two-bladed metal regulator propeller
Diameter of the propeller	2.59 m
Total contents of fuel tanks	521 l
Contents fuselage front tank	131 l
Contents fuselage center tank	131 l
Contents fuselage rear tank	95 l
Contents wing tanks	2 x 82 l
Most forward admissible c.g. position	17.36% $\overline{c}$ at 16989 N; 29.92% $\overline{c}$ at 22800 N
Most backward admissible c.g. position	40.24% $\overline{c}$

Table C.1: General aircraft data of the DHC-2 'Beaver', PH-VTH

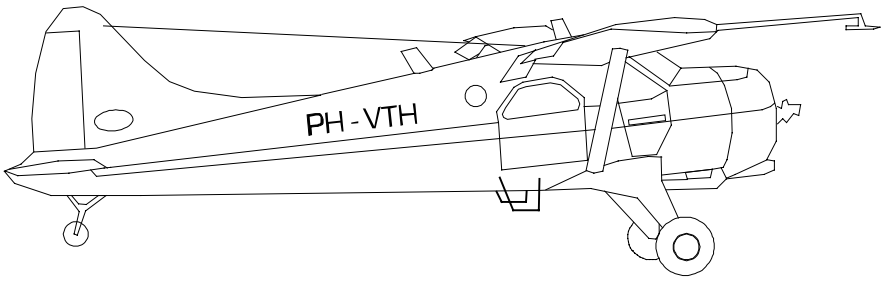


Figure C.1: The De Havilland DHC-2 'Beaver' aircraft

$x_{c.g.}$	=	0.5996	$[m]$	in F_M
$y_{c.g.}$	=	0.0	$[m]$	in F_M
$z_{c.g.}$	=	-0.8851	$[m]$	in F_M
I_x	=	5368.39	$[kg\,m^2]$	in F_R
I_y	=	6928.93	$[kg\,m^2]$	in F_R
I_z	=	11158.75	$[kg\,m^2]$	in F_R
J_{xy}	=	0.0	$[kg\,m^2]$	in F_R
J_{xz}	=	117.64	$[kg\,m^2]$	in F_R
J_{yz}	=	0.0	$[kg\,m^2]$	in F_R
m	=	2288.231	$[kg]$	
h	=	1828.8	$[m]$	$(= 6000\ [ft])$
ρ	=	1.024	$[kg\,m^{-3}]$	

Table C.2: Aircraft data on which the aerodynamic model of the 'Beaver' is based

C_X		C_Y		C_Z	
parameter	value	parameter	value	parameter	value
0	-0.03554	0	-0.002226	0	-0.05504
α	0.002920	β	-0.7678	α	-5.578
α^2	5.459	$\frac{pb}{2V}$	-0.1240	α^3	3.442
α^3	-5.162	$\frac{rb}{2V}$	0.3666	$\frac{q\bar{c}}{V}$	-2.988
$\frac{q\bar{c}}{V}$	-0.6748	δ_a	-0.02956	δ_e	-0.3980
δ_r	0.03412	δ_r	0.1158	$\delta_e\beta^2$	-15.93
δ_f	-0.09447	$\delta_r\alpha$	0.5238	δ_f	-1.377
$\alpha\delta_f$	1.106	$\frac{\dot{\beta}b}{2V}$	-0.1600	$\alpha\delta_f$	-1.261

C_l		C_m		C_n	
parameter	value	parameter	value	parameter	value
0	0.0005910	0	0.09448	0	-0.003117
β	-0.06180	α	-0.6028	β	0.006719
$\frac{pb}{2V}$	-0.5045	α^2	-2.140	$\frac{pb}{2V}$	-0.1585
$\frac{rb}{2V}$	0.1695	$\frac{q\bar{c}}{V}$	-15.56	$\frac{rb}{2V}$	-0.1112
δ_a	-0.09917	δ_e	-1.921	δ_a	-0.003872
δ_r	0.006934	β^2	0.6921	δ_r	-0.08265
$\delta_a\alpha$	-0.08269	$\frac{rb}{2V}$	-0.3118	$\frac{q\bar{c}}{V}$	0.1595
		δ_f	0.4072	β^3	0.1373

Table C.3: Coefficients in the aerodynamic model of the 'Beaver' (TAS-range: 35-55 ms^{-1})

C_X		C_Y		C_Z	
parameter	value	parameter	value	parameter	value
dpt	0.1161	$-$	$-$	dpt	-0.1563
$\alpha \cdot dpt^2$	0.1453				

C_l		C_m		C_n	
parameter	value	parameter	value	parameter	value
$\alpha^2 \cdot dpt$	-0.01406	dpt	-0.07895	dpt^3	-0.003026

$$dpt \equiv \frac{\Delta p_t}{\frac{1}{2}\rho V^2} = C_1 + C_2 \frac{P}{\frac{1}{2}\rho V^3} \quad \text{with:} \quad \begin{cases} C_1 &= 0.08696 \\ C_2 &= 191.18 \end{cases}$$

Table C.4: Coefficients in the engine forces & moments model of the 'Beaver' (35-55 ms^{-1})

Appendix D

FDC implementation of the aircraft parameters

D.1 How to define the parameters in the MATLAB workspace

The SIMULINK models from FDC 1.2 use data from the MATLAB workspace in order to define the model parameters. This data is of course highly dependent of the aircraft under consideration. For the 'Beaver' model, most parameters have been stored in three parameter matrices and one parameter vector. The analytical routines from SIMULINK and the FDC toolbox, described in chapter 8, can only be applied to the system Beaver if these parameters are present in the MATLAB workspace. It is possible to load them from the file AIRCRAFT.DAT by means of the routine LOADER (section 9.3.1). The file AIRCRAFT.DAT itself can be created with the routine MODBUILD (section 9.2). The definitions of the parameter vector and matrices for the system Beaver will be given below. See appendix C for the numerical values of all parameters and for some general information about the 'Beaver' aircraft.

If you plan to implement a model of another aircraft within the FDC structure, you are free to use your own data-formats for all aircraft-dependent blocks within the systems. However, unless you plan to enhance the model with options for non-constant mass and/or mass-distribution properties, it is recommended *not* to alter the definitions of the matrices *GM1* and *GM2* in which the basic geometrical properties, the mass, and the inertia parameters of the aircraft are defined. This is because those two matrices are used by aircraft-*independent* subsystems as well. With regard to these parameters, the best way of implementing a new aircraft model is to insert the appropriate values in the routine MODBUILD by editing its source-file MODBUILD.M in the FDC subdirectory AIRCRAFT and then run this routine to update *GM1* and *GM2*. The many comment-lines within the source code will guide you through this process.

D.2 Definition of the parameter matrices for the system Beaver

Aerodynamic model

The stability and control derivatives used by the aerodynamic model of the 'Beaver' aircraft have been combined in the parameter matrix AM. The definition of AM is given in table D.2. The numerical values of these coefficients can be found in table C.3 from appendix C; they have been implemented in MODBUILD to generate the datafile AIRCRAFT.DAT. Equation (3.13) in section 3.2.2 describes the general polynomial structure of the aerodynamic model of the 'Beaver'.

Engine forces & moments model

The coefficients used by the model of the engine forces and moments of the 'Beaver' aircraft have been combined in the parameter matrix *EM*. The definition of *EM* is given in table D.2. The numerical values of these coefficients can be found in table C.4 from appendix C; they have been implemented in MODBUILD to generate the datafile AIRCRAFT.DAT. Equation (3.17) in section 3.2.2 describes the general polynomial structure of the engine forces and moments model of the 'Beaver'.

Weight & balance and geometrical data

In the aircraft model from FDC 1.2 it is assumed that the mass of the airplane and its mass-distribution remain constant during the motions of interest. These values can therefore be regarded as *parameters* for the equations of motion. The parameter vector *GM1* contains the mass, moments and products of inertia, and three geometrical properties, being: the mean aerodynamic chord, wing span, and wing area. See appendix C for the numerical values of these properties. The parameter matrix *GM2* contains the inertia coefficients from table B.2 in appendix B. The routine MODBUILD contains the definitions of these coefficients for the 'Beaver' aircraft and the general (!) inertia equations from table B.2. It stores the resulting parameter vectors in the datafile AIRCRAFT.DAT. Table D.3 gives the definitions of *GM1* and *GM2*.

$$
AM = \begin{bmatrix}
C_{X_0} & C_{Y_0} & C_{Z_0} & C_{l_0} & C_{m_0} & C_{n_0} \\
C_{X_\alpha} & 0 & C_{Z_\alpha} & 0 & C_{m_\alpha} & 0 \\
C_{X_{\alpha^2}} & 0 & 0 & 0 & C_{m_{\alpha^2}} & 0 \\
C_{X_{\alpha^3}} & 0 & C_{Z_{\alpha^3}} & 0 & 0 & 0 \\
0 & C_{Y_\beta} & 0 & C_{l_\beta} & 0 & C_{n_\beta} \\
0 & 0 & 0 & 0 & C_{m_{\beta^2}} & 0 \\
0 & 0 & 0 & 0 & 0 & C_{n_{\beta^3}} \\
0 & C_{Y_p} & 0 & C_{l_p} & 0 & C_{n_p} \\
C_{X_q} & 0 & C_{Z_q} & 0 & C_{m_q} & C_{n_q} \\
0 & C_{Y_r} & 0 & C_{l_r} & C_{m_r} & C_{n_r} \\
0 & 0 & C_{Z_{\delta_e}} & 0 & C_{m_{\delta_e}} & 0 \\
C_{X_{\delta_f}} & 0 & C_{Z_{\delta_f}} & 0 & C_{m_{\delta_f}} & 0 \\
0 & C_{Y_{\delta_a}} & 0 & C_{l_{\delta_a}} & 0 & C_{n_{\delta_a}} \\
C_{X_{\delta_r}} & C_{Y_{\delta_r}} & 0 & C_{l_{\delta_r}} & 0 & C_{n_{\delta_r}} \\
C_{X_{\alpha\delta_f}} & 0 & C_{Z_{\alpha\delta_f}} & 0 & 0 & 0 \\
0 & C_{Y_{\delta_r\alpha}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & C_{l_{\delta_a\alpha}} & 0 & 0 \\
0 & 0 & C_{Z_{\delta_e\beta^2}} & 0 & 0 & 0 \\
0 & C_{Y_\beta} & 0 & 0 & 0 & 0
\end{bmatrix}^T
\begin{matrix}
\mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \\ \mathbf{10} \\ \mathbf{11} \\ \mathbf{12} \\ \mathbf{13} \\ \mathbf{14} \\ \mathbf{15} \\ \mathbf{16} \\ \mathbf{17} \\ \mathbf{18} \\ \mathbf{19}
\end{matrix}
$$

$$
\begin{matrix}
\mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6}
\end{matrix}
$$

Table D.1: Coefficients of the aerodynamic model of the 'Beaver'

$$
EM = \begin{bmatrix}
C_{X_{dpt}} & 0 & C_{Z_{dpt}} & 0 & C_{m_{dpt}} & 0 \\
0 & 0 & 0 & 0 & 0 & C_{n_{dpt}} \\
C_{X_{\alpha\, dpt^2}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & C_{l_{\alpha^2\, dpt}} & 0 & 0
\end{bmatrix}^T
\begin{matrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \end{matrix}
$$

$$
\begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \end{matrix}
$$

Table D.2: Coefficients of the engine forces & moments model of the 'Beaver'

$$
GM1 = \begin{bmatrix} \overline{c} & b & S & I_{xx} & I_{yy} & I_{zz} & J_{xy} & J_{xz} & J_{yz} & m \end{bmatrix}^T \quad \mathbf{1}
$$

$$
\begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} \end{matrix}
$$

$$
GM2 = \begin{bmatrix}
P_l & P_m & P_n & P_{pp} & P_{pq} & P_{pr} & P_{qq} & P_{qr} & P_{rr} \\
Q_l & Q_m & Q_n & Q_{pp} & Q_{pq} & Q_{pr} & Q_{qq} & Q_{qr} & Q_{rr} \\
R_l & R_m & R_n & R_{pp} & R_{pq} & R_{pr} & R_{qq} & R_{qr} & R_{rr}
\end{bmatrix}^T
\begin{matrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \end{matrix}
$$

$$
\begin{matrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} \end{matrix}
$$

Table D.3: 'Beaver' geometry and mass-distribution

Appendix E

Definitions of variables and acronyms from FDC 1.2

E.1 Variables and acronyms from the graphical models

A large number of acronyms has been used in the graphical models of FDC 1.2 in order to clarify the meaning of all signal lines. These acronyms are also used within the on-line helpfiles (*.HLP). The following subsections explain the meaning of these variables and acronyms. See appendix A for the meaning of the symbols themselves. The variables from the different MATLAB subroutines from FDC 1.2 are explained within the source-codes of the subroutines themselves. Section E.2 defines the input and output variables for the non-linear aircraft model and radio navigation models. The model parameters for the SIMULINK implementation of the non-linear aircraft model have been defined in appendix D.

E.1.1 Aircraft model (system Beaver)

a	a	
alpha	α	
alphadot	$\dot{\alpha}$	
Ax	A_x	
axk	$a_{x,k}$	
Ay	A_y	
ayk	$a_{y,k}$	
Az	A_z	
azk	$a_{z,k}$	
beta	β	
betadot	$\dot{\beta}$	
Caero	$\mathbf{C}_{aero}$	$= [\, C_{X_a}\ C_{Y_a}\ C_{Z_a}\ C_{l_a}\ C_{m_a}\ C_{n_a}\,]^T$
chi	χ	
Cla	C_{la}	
Clp	C_{lp}	
Cma	C_{ma}	
Cmp	C_{mp}	
Cna	C_{na}	
Cnp	C_{np}	
Cprop	$\mathbf{C}_{prop}$	$= [\, C_{X_p}\ C_{Y_p}\ C_{Z_p}\ C_{l_p}\ C_{m_p}\ C_{n_p}\,]^T$
CXa	C_{Xa}	
CXp	C_{Xp}	
CYa	C_{Ya}	

CYp	$C_{Y\,p}$
CZa	$C_{Z\,a}$
CZp	$C_{Z\,p}$
deltaa	δ_a
deltae	δ_e
deltaf	δ_f
deltar	δ_r
dpt	*dpt*
Fgrav	$\mathbf{F}_{grav} = [\, X_{gr}\; Y_{gr}\; Z_{gr}\,]^T$
fpa	*fpa*
FMaero	$\mathbf{FM}_{aero} = [\, X_a\; Y_a\; Z_a\; L_a\; M_a\; N_a\,]^T$
FMprop	$\mathbf{FM}_{prop} = [\, X_p\; Y_p\; Z_p\; L_p\; M_p\; N_p\,]^T$
Ftot	$\mathbf{F}_{tot} = [\, F_x\; F_y\; F_z\,]^T$
Fwind	$\mathbf{F}_{wind} = [\, X_w\; Y_w\; Z_w\,]^T$
Fx	F_x
Fy	F_y
Fz	F_z
g	g
gamma	γ
In	matrix with time-trajectories of the inputs, see table E.3
L	L
La	L_a
Lp	L_p
M	M
Ma	M_a
Mp	M_p
Mtot	$\mathbf{M}_{tot} = [\, L\; M\; N\,]^T$
mu	μ
n	n
N	N
Na	N_a
Np	N_p
Out	matrix with time-trajectories of the outputs, see table E.4
P	P
p	p
pb/2V	$\frac{pb}{2V}$
pdot	$\dot{p}$
Phi	Φ
phi	φ
phidot	$\dot{\varphi}$
ps	p_s
psi	ψ
psidot	$\dot{\psi}$
pz	p_z
q	q
qc	q_c
qc/V	$\frac{q\bar{c}}{V}$
qdot	$\dot{q}$
qdyn	q_{dyn}
r	r
rb/2V	$\frac{rb}{2V}$
Rc	R_c
rdot	$\dot{r}$
Re	R_e

rho	ρ
T	T
theta	θ
thetadot	$\dot{\theta}$
time	vector containing the time-axis of the simulations
Tt	T_t
u	u
uaero	$\mathbf{u}_{aero} = [\ \delta_e\ \delta_a\ \delta_r\ \delta_f\ ]^T$
udot	$\dot{u}$
ueul	$\mathbf{u}_{eul}$, see chapter 5
upqr	$\mathbf{u}_{pqr}$, see chapter 5
uprop	$\mathbf{u}_{prop} = [\ n\ p_z\ ]^T$
uVab	$\mathbf{u}_{Vab}$, see chapter 5
uw	u_w
uwdot	$\dot{u}_w$
uwind	$\mathbf{u}_{wind} = [\ u_w\ v_w\ w_w\ \dot{u}_w\ \dot{v}_w\ \dot{w}_w\ ]^T$
uxyH	$\mathbf{u}_{xyH}$, see chapter 5
V	V
v	v
Vc	V_c
Vdot	$\dot{V}$
vdot	$\dot{v}$
Ve	V_e
vw	v_w
vwdot	$\dot{v}_w$
w	w
wdot	$\dot{w}$
ww	w_w
wwdot	$\dot{w}_w$
x	$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$
Xa	X_a
xdot	$\dot{\mathbf{x}} = [\ \dot{V}\ \dot{\alpha}\ \dot{\beta}\ \dot{p}\ \dot{q}\ \dot{r}\ \dot{\psi}\ \dot{\theta}\ \dot{\varphi}\ \dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$
xe	x_e
xedot	$\dot{x}_e$
Xgr	X_{gr}
xinco	initial value of $\mathbf{x}$
Xp	X_p
Xw	X_w
Ya	Y_a
yacc	$\mathbf{y}_{acc} = [\ A_x\ A_y\ A_z\ a_{x,k}\ a_{y,k}\ a_{z,k}\ ]^T$
yad1	$\mathbf{y}_{ad1} = \left[\ a\ M\ q_{dyn}\ \right]^T$
yad2	$\mathbf{y}_{ad2} = [\ q_c\ V_e\ V_c\ ]^T$
yad3	$\mathbf{y}_{ad3} = [\ T_t\ R_e\ R_c\ ]^T$
yatm	$\mathbf{y}_{atm} = [\ \rho\ p_s\ T\ \mu\ g\ ]^T$
ybvel	$\mathbf{y}_{bvel} = [\ u\ v\ w\ ]^T$
ydl	$\mathbf{y}_{dl} = \left[\ \frac{pb}{2V}\ \frac{q\bar{c}}{V}\ \frac{rb}{2V}\ \right]^T$
ye	y_e
yedot	$\dot{y}_e$
yeul	$\mathbf{y}_{eul} = [\dot{\psi}\ \dot{\theta}\ \dot{\varphi}\,]^T$
yfp	$\mathbf{y}_{fp} = [\ \gamma\ fpa\ \chi\ \Phi\ ]^T$
Ygr	$Y_g r$
yhlp	$\mathbf{y}_{hlp}$, see chapter 5
Yp	Y_p

ypow	$\mathbf{y}_{pow}$	=	$[\ dpt\ P\ ]^T$
ypqr	$\mathbf{y}_{pqr}$	=	$[\dot{p}\ \dot{q}\ \dot{r}\ ]^T$
yuvw	$\mathbf{y}_{uvw}$	=	$[\ \dot{u}\ \dot{v}\ \dot{w}\ ]^T$
yVab	$\mathbf{y}_{Vab}$	=	$[\dot{V}\ \dot{\alpha}\ \dot{\beta}\ ]^T$
Yw	Y_w		
yxyH	$\mathbf{y}_{xyH}$	=	$[\dot{x}_e\ \dot{y}_e\ \dot{H}\ ]^T$
Za	Z_a		
Zgr	Z_{gr}		
Zp	Z_p		
Zw	Z_w		

E.1.2 Autopilot models (systems APILOT1 to APILOT3)

alpha	α
ALH?	switch signal for Altitude Hold mode
ALS?	switch signal for Altitude Select mode
asymm. mode?	switch signal for asymmetrical autopilot modes
asymm. out. loop?	switch signal for asymmetrical outer-loops
beta	β
CD	Course Datum (VOR reference bearing, see section 3.4.2)
dar	gain factor *dar*, see table 11.3
drr	gain factor *drr*, see table 11.3
D_ail	$\Delta\delta_a$ input to cable & actuator models
D_ail ref	$\Delta\delta_{a,ref}$ (reference value from control laws)
D_elv	$\Delta\delta_e$ input to cable & actuator models
D_elv ref	$\Delta\delta_{e,ref}$ (reference value from control laws)
D_rud	$\Delta\delta_r$ input to cable & actuator models
D_rud ref	$\Delta\delta_{r,ref}$ (reference value from control laws)
Ddeltaa	$\Delta\delta_a$ input to non-linear aircraft model
Ddeltae	$\Delta\delta_e$ input to non-linear aircraft model
Ddeltaf	$\Delta\delta_f$ input to non-linear aircraft model; here: $\Delta\delta_f = 0$
Ddeltar	$\Delta\delta_r$ input to non-linear aircraft model
DHr (...)	reference ΔH (brackets show which of the control laws determined this value)
DHdot r (...)	reference $\Delta\dot{H}$ (brackets show which of the control laws determined this value)
Dn	Δn input to non-linear aircraft model; here: $\Delta n = 0$
Dphir (...)	reference $\Delta\varphi$ (brackets show which of the control laws determined this value)
Dpsir (...)	reference $\Delta\psi$ (brackets show which of the control laws determined this value)
Dpz	Δp_z input to non-linear aircraft model; here: $\Delta p_z = 0$
Dthetar (...)	reference $\Delta\theta$ (brackets show which of the control laws determined this value)
epsilon_gs	ε_{gs}
gamgs	γ_{gs}
Gamma_loc	Γ_{loc}
Gamma_VOR	Γ_{VOR}
GS coupled?	switch signal for Glideslope Coupled mode
H	H
Hdot	$\dot{H}$
HH?	switch signal for Heading Hold mode
HRW	H_{RW} = height of runway above sea level
HVOR	H_{VOR} = height of VOR transmitter above sea level
Kd	gain factor K_d, see table 11.3
Keps_gs	gain factor $K_{\epsilon_{gs}}$, see table 11.3
KGam_LOC	gain factor $K_{\Gamma_{loc}}$, see table 11.3
KGam_VOR	gain factor $K_{\Gamma_{VOR}}$, see table 11.3
KH	gain factor K_H, see table 11.3
KHdot	gain factor $K_{\dot{H}}$, see table 11.3

Ki	gain factor K_i, see table 11.3
Kphi	gain factor K_φ, see table 11.3
Kpsi	gain factor K_ψ, see table 11.3
Kq	gain factor K_q, see table 11.3
Kr	gain factor K_r, see table 11.3
Ktheta	gain factor K_θ, see table 11.3
Kv	gain factor K_v, see table 11.3
LOC coupled?	switch signal for Localizer Coupled mode
NAV?	switch signal for VOR Navigation mode
p	p
phi	φ
psi	ψ
psiRW	ψ_{RW} = heading of the runway centerline
psi_w	ψ_w
q	q
r	r
Rgs	R_{gs}
Rloc	R_{loc}
Sgs	S_{gs}
Sloc	S_{loc}
symm. mode?	switch signal for symmetrical autopilot modes
symm. outer loop?	switch signal for symmetrical outer-loops
theta	θ
uaero	$\mathbf{u}_{aero} = [\ \delta_e\ \delta_a\ \delta_r\ \delta_f\ ]^T$
uaero0	initial value of $\mathbf{u}_{aero}$
uprop	$\mathbf{u}_{prop} = [\ n\ p_z\ ]^T$
uprop0	initial value of $\mathbf{u}_{prop}$
uw	u_w
uwdot	$\dot{u}_w$
uwind	$\mathbf{u}_{wind} = [\ u_w\ v_w\ w_w\ \dot{u}_w\ \dot{v}_w\ \dot{w}_w\ ]^T$
V	V
vw	v_w
vwdot	$\dot{v}_w$
Vw	V_w
ww	w_w
wwdot	$\dot{w}_w$
x	$\mathbf{x} = [\ V\ \alpha\ \beta\ p\ q\ r\ \psi\ \theta\ \varphi\ x_e\ y_e\ H\ ]^T$
xdot0	initial value of $\dot{\mathbf{x}}$
xe	x_e
xgs	x_{gs} = X-distance from runway threshold to glideslope transmitter
xinco	initial value of $\mathbf{x}$
xloc	x_{loc} = X-distance from runway threshold to localizer transmitter
xRW	x_{RW} = initial X-distance from aircraft to runway threshold
xVOR	x_{VOR} = X-coordinate of VOR transmitter
ydl	$\mathbf{y}_{dl} = \left[\ \frac{pb}{2V}\ \frac{q\bar{c}}{V}\ \frac{rb}{2V}\ \right]^T$
ye	y_e
ygs	y_{gs} = Y-distance from runway threshold to glideslope transmitter
yils	$\mathbf{y}_{ils}$, see table E.5
ymod1A	vector with initial asymmetrical mode-switches, see section 12.4.1
ymod2A	vector with second asymmetrical mode-switches, see section 12.4.1
ymod1S	vector with initial symmetrical mode-switches, see section 12.4.1
ymod2S	vector with second symmetrical mode-switches, see section 12.4.1
ymode	vector with symmetrical and asymmetrical mode-switches
yref	vector with symmetrical and asymmetrical reference inputs to the control laws

yrefA	vector with asymmetrical reference inputs to the control laws, see section 12.4.1		
yrefS	vector with symmetrical reference inputs to the control laws, see section 12.4.1		
yRW	Y_{RW}	=	initial Y-distance from aircraft to runway threshold
yvor	$\mathbf{y}_{vor}$, see table E.6		
yVOR	Y_{VOR}	=	Y-coordinate of VOR transmitter
z	$\mathbf{z}$	=	$[\,\mathbf{x}\,,\ \dot{H}\,]^T$ (*after* passing the sensor blocks)
z-z0	$\mathbf{z} - \mathbf{z}_0$, with $\mathbf{z}_0$ the initial value of $\mathbf{z}$		

E.1.3 Radio-navigation models (library NAVLIB)

For more details about the symbols from the following list, see also sections 3.4.1 and 3.4.2 from chapter 3. In particular, consult figures 3.13, 3.14, 3.15, and 3.17.

cat	ILS performance category (1, 2, or 3)		
CD	Course Datum (VOR reference bearing, see section 3.4.2)		
Cone of silence flag	flag, equals 1 if aircraft enters 'cone of silence' above VOR station		
cos_gamgs	$\cos\gamma_{gs}$		
cos_psiRW	$\cos\psi_{RW}$		
dgs	d_{gs}		
D_igs	Δi_{gs}	=	steady state error in nominal glideslope angle
D_iloc	Δi_{loc}	=	steady state error due to misalignment of localizer reference plane
epsilon_gs	ε_{gs}		
gamgs	γ_{gs}	=	nominal glideslope angle
Gamma_gs	Γ_{gs}		
Gamma_loc	Γ_{loc}		
Gamma_VOR	Γ_{VOR}		
GS_flag	flag, equals 1 if glideslope can *not* be received accurately		
H	H		
Hf	H_f		
HRW	H_{RW}	=	height of runway above sea level
HVOR	H_{VOR}	=	height of VOR transmitter above sea level
igs	i_{gs}		
iloc	i_{loc}		
K	general symbol for gain value (used for first-order transfer function)		
KSgs	gain for i_{gs}, used to model deviations in glideslope sensitivity		
KSloc	gain for i_{loc}, used to model deviations in localizer sensitivity		
KVORerr	gain for Γ_{VOR}, used to model steady-state VOR errors		
Lgs	L_{gs}	=	scale-length of glideslope noise
Lloc	L_{loc}	=	scale-length of localizer noise
LOC_flag	flag, equals 1 if localizer can *not* be received accurately		
psi	ψ		
psiRW	ψ_{RW}	=	runway heading
QDR	QDR	=	current VOR bearing
Range flag	flag, equals 1 if distance to VOR transmitter exceeds range of VOR signals		
Rgs	R_{gs}		
Rloc	R_{loc}		
RVOR	R_{VOR}		
RWpos	runway position vector $[\,x_{RW}\,y_{RW}\,H_{RW}\,]^T$		
Sgs	S_{gs}		
sigma_gs	σ_{gs}	=	standard deviation of glideslope noise
sigma_loc	σ_{loc}	=	standard deviation of localizer noise
sin_psiRW	$\sin\psi_{RW}$		
Sloc	S_{loc}		
tan_gamgs	$\tan\gamma_{gs}$		
tau	τ	=	general symbol for time-constant of first-order filters
ToFrom	flag, equals 1 if the aircraft flies *To* the VOR or 0 if it flies away *From* the VOR		

uils	$\mathbf{u}_{ils}$	=	$[\, x_e\ y_e\ H\,]^T$
uVOR	$\mathbf{u}_{VOR}$	=	$[\, x_e\ y_e\ H\,]^T$
V	V		
xe	x_e		
xf	x_f		
xgs	x_{gs}	=	X-distance from runway threshold to glideslope transmitter
xloc	x_{loc}	=	X-distance from runway threshold to localizer transmitter
xRW	x_{RW}	=	initial X-distance from aircraft to runway threshold
xVOR	x_{VOR}	=	X-coordinate of VOR transmitter
ye	y_e		
yf	y_f		
ygs	y_{gs}	=	Y-distance from runway threshold to glideslope transmitter
yils	$\mathbf{y}_{ils}$, see table E.5		
yils1	$\mathbf{y}_{ils1}$	=	$[\, i_{gs}\ i_{loc}\,]^T$
yils2	$\mathbf{y}_{ils2}$	=	$[\, \epsilon_{gs}\ \Gamma_{loc}\,]^T$
yils3	$\mathbf{y}_{ils3}$	=	$[\, x_f\ y_f\ H_f\ d_{gs}\ R_{gs}\ R_{loc}\,]^T$
yils4	$\mathbf{y}_{ils1}$	=	$[\, LOC\_flag\ GS\_flag\,]^T$
yvor	$\mathbf{y}_{vor}$, see table E.6		
yVOR	y_{VOR}	=	Y-coordinate of VOR transmitter
yvor1	y_{vor1}	=	Γ_{VOR}
yvor2	y_{vor2}	=	R_{VOR}
yvor3	$\mathbf{y}_{vor3}$	=	[*Cone of silence flag, Range flag*]^T
yvor4	y_{vor4}	=	*ToFrom*
yRW	y_{RW}	=	initial Y-distance from aircraft to runway threshold

E.1.4 Wind and turbulence models (library WINDLIB)

a0, a1, ...	$a_0, a_1, \ldots$	=	coefficients of transfer function denominator
b0, b1, ...	$b_0, b_1, \ldots$	=	coefficients of transfer function numerator
H	H		
K	general symbol for gain value (used for first-order transfer function)		
Lug	L_{u_g}	=	scale length of longitudinal turbulence
Lvg	L_{v_g}	=	scale length of lateral turbulence
Lwg	L_{w_g}	=	scale length of vertical turbulence
psi	ψ		
psi_w	ψ_w		
sigma_ug	σ_{u_g}	=	standard deviation of longitudinal turbulence
sigma_vg	σ_{v_g}	=	standard deviation of lateral turbulence
sigma_wg	σ_{w_g}	=	standard deviation of vertical turbulence
tau	τ	=	general symbol for time-constant of first-order filters
ug	u_g	=	u_w due to turbulence
ugdot	$\dot{u}_g$		
uw	u_w		
uwdot	$\dot{u}_w$		
uwind	$\mathbf{u}_{wind}$	=	$[\, u_w\ v_w\ w_w\ \dot{u}_w\ \dot{v}_w\ \dot{w}_w\,]^T$
vg	v_g	=	v_w due to turbulence
vgdot	$\dot{v}_g$		
vw	v_w		
Vw	V_w		
vwdot	$\dot{v}_w$		
wg	w_g	=	w_w due to turbulence
wgdot	$\dot{w}_g$		
ww	w_w		
wwdot	$\dot{w}_w$		

E.2 Input/output variables of the simulation models

E.2.1 Aircraft model (system Beaver)

Results which are sent to the workspace

During simulations of systems which call the non-linear aircraft model Beaver, the results are stored in the matrices *In* and *Out* within the MATLAB workspace. These matrices have been defined in tables E.3 and E.4, respectively. Each column of these matrices contains a time-trajectory of a specific input or output signal. These matrices are obtained by combining the input and output signals from the different subsystems of the system Beaver within its top-level. See section 5.1 and the description of Level 1 (the top-level of the system Beaver) in chapter 5. The time-axis itself is stored in a separate vector *time*, to provide the reference base against which the time-trajectories of the inputs and outputs can be plotted. On-line help with regard to these input and output definitions will be displayed in the command window if you enter `type level1.hlp`, `type inputs.hlp`, or `type outputs.hlp`. The routines RESULTS and RESPLOT, which have been described in chapter 9, are available for simplifying evaluations of the simulation results. Type `help results` or `help resplot` for more details.

S-function inputs and outputs

Due to the fact that it is not possible to send vector signals through Inport and Outport blocks in the first level of a graphical SIMULINK system, it was not very practical to match the definition of the matrix *Out* with the output vector that connects the system Beaver to other dynamical systems. For this reason, only a subset of all output signals from Beaver was made available for connecting other systems. In this report, these variables have been referred to as *S-function outputs*. Table E.1 shows the definition of this vector. The definition of the input which is used to connect the system Beaver to other systems does match the definition of the matrix *In*, as shown in table E.2 (compare with table E.3). For on-line help regarding these S-function inputs and outputs enter `type level1.hlp`, `type inputs.hlp`, or `type outputs.hlp` at the command-line. See also figure 5.2 in chapter 5.

E.2.2 Radio navigation models (library NAVLIB)

During simulations, the outputs from the ILS example system ILS example are stored in the matrix *yils*. The outputs from the VOR example system VOR example are stored in the matrix *yvor*. Tables E.5 and E.6 show the definitions of these matrices. The row-numbers are displayed underneath the symbols. See the description of the systems ILS example and VOR example in chapter 7 for more information.

Definition of the 'S-function output vector' (the elements from this vector have all been connected to an Outport block in the first level of Beaver)															
V	α	β	p	q	r	ψ	θ	φ	x_e	y_e	H	$\dot{H}$	$\frac{pb}{2V}$	$\frac{q\bar{c}}{V}$	$\frac{rb}{2V}$
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Table E.1: Definition of the S-function output vector of the system Beaver

Definition of the 'S-function input vector' (the elements from this vector have all been connected to an Inport block in the first level of Beaver)											
δ_e	δ_a	δ_r	δ_f	n	p_z	u_w	v_w	w_w	$\dot{u}_w$	$\dot{v}_w$	$\dot{w}_w$
1	2	3	4	5	6	7	8	9	10	11	12

Table E.2: Definition of the S-function input vector of the system Beaver

Sub-vector	Definition of the matrix *In* (the numbers correspond with the columns, containing the time-trajectories of the specified input variables)					
$\mathbf{u}_{aero}$	δ_e	δ_a	δ_r	δ_f		
	1	2	3	4		
$\mathbf{u}_{prop}$	n	p_z				
	5	6				
$\mathbf{u}_{wind}$	u_w	v_w	w_w	$\dot{u}_w$	$\dot{v}_w$	$\dot{w}_w$
	7	8	9	10	11	12

Table E.3: Definition of the matrix *In*

Sub-vector	Definition of the matrix *Out* (the numbers correspond with the columns, containing the time-trajectories of the specified output variables)											
$\mathbf{x}$	V	α	β	p	q	r	ψ	θ	φ	x_e	y_e	H
	1	2	3	4	5	6	7	8	9	10	11	12
$\dot{\mathbf{x}}$	$\dot{V}$	$\dot{\alpha}$	$\dot{\beta}$	$\dot{p}$	$\dot{q}$	$\dot{r}$	$\dot{\psi}$	$\dot{\theta}$	$\dot{\varphi}$	$\dot{x}_e$	$\dot{y}_e$	$\dot{H}$
	13	14	15	16	17	18	19	20	21	22	23	24
$\mathbf{y}_{bvel}$	u	v	w									
	25	26	27									
$\mathbf{y}_{uvw}$	$\dot{u}$	$\dot{v}$	$\dot{w}$									
	28	29	30									
$\mathbf{y}_{dl}$	$\frac{pb}{2V}$	$\frac{q\bar{c}}{V}$	$\frac{rb}{2V}$									
	31	32	33									
$\mathbf{y}_{fp}$	γ	fpa	χ	Φ								
	34	35	36	37								
$\mathbf{y}_{pow}$	dpt	P										
	38	39										
$\mathbf{y}_{acc}$	A_x	A_y	A_z	$a_{x,k}$	$a_{y,k}$	$a_{z,k}$						
	40	41	42	43	44	45						
$\mathbf{C}_{aero}$	C_{Xa}	C_{Ya}	C_{Za}	C_{la}	C_{ma}	C_{na}						
	46	47	48	49	50	51						
$\mathbf{C}_{prop}$	C_{Xp}	C_{Yp}	C_{Zp}	C_{lp}	C_{mp}	C_{np}						
	52	53	54	55	56	57						
$\mathbf{FM}_{aero}$	X_a	Y_a	Z_a	L_a	M_a	N_a						
	58	59	60	61	62	63						
$\mathbf{FM}_{prop}$	X_p	Y_p	Z_p	L_p	M_p	N_p						
	64	65	66	67	68	69						
$\mathbf{F}_{grav}$	X_{gr}	Y_{gr}	Z_{gr}									
	70	71	72									
$\mathbf{F}_{wind}$	X_w	Y_w	Z_w									
	73	74	75									
$\mathbf{y}_{atm}$	ρ	p_s	T	μ	g							
	76	77	78	79	80							
$\mathbf{y}_{ad1}$	a	M	q_{dyn}									
	81	82	83									
$\mathbf{y}_{ad2}$	q_c	V_e	V_c									
	84	85	86									
$\mathbf{y}_{ad3}$	T_t	R_e	R_c									
	87	88	89									

Table E.4: Definition of the matrix *Out*

Sub-vector	Definition of the matrix *yils* (the numbers correspond with the columns, containing the time-trajectories of the specified output variables)					
$\mathbf{y}_{ils1}$	i_{gs}	i_{loc}				
	1	2				
$\mathbf{y}_{ils2}$	ε_{gs}	Γ_{loc}				
	3	4				
$\mathbf{y}_{ils3}$	x_f	y_f	H_f	d_{gs}	R_{gs}	R_{loc}
	5	6	7	8	9	10
$\mathbf{y}_{ils4}$	*LOC_flag*	*GS_flag*				
	11	12				

Table E.5: Definition of the matrix *yils*, containing outputs from ILS example

Sub-vector	Definition of the matrix *yvor* (the numbers correspond with the columns, containing the time-trajectories of the specified output variables)	
y_{VOR1}	Γ_{VOR}	
	1	
y_{VOR2}	R_{VOR}	
	2	
$\mathbf{y}_{VOR3}$	*Cone of silence flag*	*Range flag*
	3	4
y_{VOR4}	*To/From flag*	
	5	

Table E.6: Definition of the matrix $yVOR$, containing outputs from VOR example

Bibliography

[1] Anon. *Approach and Landing Simulation*. AGARD report 632, Ames, 1975.

[2] Anon. *International Standards and Recommended Practices*. Annex 10, Volume I, Part I: *Equipment and Systems* and Attachment C to Part I, ICAO, Montreal, Canada, 1968.

[3] Abbink. F.J.: *Vliegtuiginstrumentatie I/II*. Lecture Notes D-34 (in Dutch), Delft University of Technology, Faculty of Aerospace Engineering, Delft, 1984.

[4] Bauss, W. (ed.): *Radio Navigation Systems for Aviation and Maritime Use*. AGARDograph 63, Pergamon Press, UK, 1963.

[5] Bosch, P.P.J. van den, Klauw, A.C. van der: *Modelling, Identification, and Simulation of Dynamical Systems*. Lecture notes $l87$, Delft University of Technology, Faculty of Electrical Engineering, Delft, edition 1992.

[6] Brandt, A.P., Broek, P.Ph. van der: *Vliegeigenschappen 2*. Lecture Notes D-34 (in Dutch), Delft University of Technology, Faculty of Aerospace Engineering, Delft, 1984.

[7] Chen, Chi-Tsong: *Linear system theory and design*. Holt, Rinehart and Winston Inc., USA, 1984.

[8] Colgren, R.D.: *A Workstation for the integrated design and simulation of flight control systems*. Lockheed Aeronautical Systems Company, Burbank, California, USA, 1988.

[9] Duke, E.L., Antoniewicz, R.F., Krambeer, K.D.: *Derivation and Definition of a Linear Aircraft Model*. NASA Reference Publication 1207, USA, 1988.

[10] Etkin, B.: *Dynamics of Flight – Stability and Control*. Wiley, New York, USA, 2^{nd} edition, 1982.

[11] Fogarty, L.E., Howe, R.M.: *Computer mechanization of six degree-of-freedom flight equations*. NASA Contractor Report 1344, USA, 1969.

[12] Forsythe, G.E., Malcolm, M.A., Moler, C.B.: *Computer methods for Mathematical Computations*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1977.

[13] Gear, C.W.: *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.

[14] Gerlach, O.H.: *Mathematical model of external disturbances acting on an aircraft during an ILS approach and landing*. Report VTH-159, Delft University of Technology, Faculty of Aerospace Engineering, Delft, The Netherlands, 1970.

[15] Gerlach, O.H.: *Lecture Notes on Aircraft Stability and Control*. Lecture Notes D-26, Delft University of Technology, Faculty of Aerospace Engineering, Delft, The Netherlands, 1981.

[16] Hoogstraten, J.A., Moesdijk, B. van de: *Modular programming structure applied to the simulation of non-linear aircraft models*. In: *IMACS Conference Proceedings on Simulation in Engineering Sciences*, Nantes, France, 1983.

[17] Johnson, W.A., McRuer, D.T.: *Development of a Category II Approach System Model.* NASA Contractor Report 2022, Washington D.C., USA, 1972.

[18] Kendal, B.: *Manual of Avionics.* BSP Professional Books, UK, 2^{nd} edition, 1987.

[19] McLean, D.: *Automatic Flight Control Systems.* Prentice Hall, Hertfordshire, UK, 1990.

[20] McRuer, D., Ashkenas, I., Graham, D.: *Aircraft Dynamics and Automatic Control.* Princeton University Press, Princeton, New Jersey, USA, 1973.

[21] Mulder, J.A., Vaart, J.C. van der: *Aircraft Responses to Atmospheric Turbulence.* Lecture Notes D-47, Delft University of Technology, Faculty of Aerospace Engineering, Delft, The Netherlands, edition 1992.

[22] Rauw, M.O.: *A* SIMULINK *environment for Flight Dynamics and Control analysis – Application to the DHC-2 'Beaver'* (2 parts). Graduate's thesis (not published). Delft University of Technology, Faculty of Aerospace Engineering, Delft, The Netherlands, 1993.

[23] Ruijgrok, G.J.J.: *Elements of Airplane Performance.* Delft University Press, Delft, The Netherlands, 1990.

[24] Stengel, R.F., Sircar, S.: *Computer-Aided Design of Flight Control Systems.* AIAA-91-2677-CP, Princeton, New Jersey, USA, 1991.

[25] Stevens, B.L., Lewis, F.L.: *Aircraft Control and Simulation.* John Wiley & Sons Inc., 1992.

[26] Tjee, R.T.H., Mulder, J.A.: *Stability and Control Derivatives of the De Havilland DHC-2 'Beaver' Aircraft.* Report LR-556, Delft University of Technology, Faculty of Aerospace Engineering, Delft, The Netherlands, 1988.

[27] Tomlinson, B.N., Padfield, G.D., Smith, P.R.: *Computer-Aided control law research – from concept to flight test.* In: *AGARD Conference Proceedings on Computer Aided System Design and Simulation*, AGARD CP-473, London, 1990.

[28] Vegte, J. van de: *Feedback Control Systems.* Prentice Hall, London, UK, 2^{nd} edition, 1990.

[29] Wever, P.N.H.: *Ontwerp en implementatie van de regelwetten van het automatisch besturingssysteem van de De Havilland DHC-2 'Beaver'.* Graduate's thesis (in Dutch, not published), Delft University of Technology, Faculty of Aerospace Engineering, Delft, The Netherlands, 1993.

Index