Aircraft Control Toolbox
Learning Edition

Aircraft Control Toolbox Learning Edition Tutorial (November 2004)

Printing History

July 2002 First Printing
January 2004 Second Printing
November 2004 Thirs Printing

Princeton Satellite Systems, Inc.
33 Witherspoon Street
Princeton, New Jersey 08542

Technical Support/Sales/Info:http://www.psatellite.com

Table of Contents

Chapter 1 Introduction 11

Key Features 12
Aircraft Properties 12
Control Design 13
Graphics and Simulation 14

Chapter 2 Fundamentals 15

Aircraft Properties Databases 16
Organizing Your Scripts 17
Functions 17

Chapter 3 Getting Help 19

MATLAB Help 20
Demos 21
File Help 23
- Introduction 23
- The List Pane 25
- Edit Button 26
- The Example Pane 26
- Run Example Button 26
- Save Example Button 27
- Help Button 27
- Quit 27

Searching in File Help 27
- Find 27
- Find All Button 28

Search Headers Button 28
Search String Edit Box 28
Graphical User Interface Help 28
Technical Support 29

Chapter 4 Structures 31

Data Structures 32
Cell Arrays 33
Classes 34

Chapter 5 Simulation 37

Simulation 38
Introduction 38
Aspects of Simulation Models 38
Linear 40
Nonlinear 41
Creating an Interactive Simulation 42
Customizing a Simulation 48

Chapter 6 Graphics 51

GUIs 52
Plotting 57

Chapter 7 Designing Controllers 59

Using the block diagram 60
Linear Quadratic Control 60
Single-Input-Single-Output 64

Eigenstructure Assignment 68

Chapter 8 Implementing Controllers 71

A General Interface 72
Closed Loop Control 76
Introduction 76
Sensor Input 77
Actuator Model 77
Control Law 77
Pilot Input 83
Control Implementation 84

Chapter 9 References 87

About the References 88
Reference Books 88
Papers 89

List of Figures, Tables, and Listings

Chapter 1 Introduction 11

Figure 1-1 Aircraft properties displayed from a database 13
Figure 1-2 Control GUI 14

Chapter 2 Fundamentals 15

Table 2-1 Aircraft Properties 16
Figure 2-1 F-16 Properties 16

Chapter 3 Getting Help 19

Figure 3-1 DemoPSS 21
Figure 3-2 Opening CASDesign 22
Figure 3-3 Results of `ADSim` 23
Figure 3-4 AircraftLE Folder 24
Figure 3-5 The file help GUI 24
Figure 3-6 Selecting from the alphabetical display 25
Figure 3-7 Using the hierarchical list 26
Figure 3-8 Search Results 27
Figure 3-9 Help GUI 28
Figure 3-10 PSS web technical support 29

Chapter 4 Structures 31

Figure 4-1 Object oriented programming terms 35

Chapter 5 Simulation 37

Table 5-1 Models 38
Listing 5-1 `Fly.m initialization` 42
Listing 5-2 `Fly.m control initialization` 43
Listing 5-3 `Fly.m initializing the state vector` 44
Listing 5-4 `Fly.m Getting the linearized model` 45
Listing 5-5 `Fly.m setting up the HUD` 45
Listing 5-6 `Fly.m setting up the aircraft 3D display` 45
Listing 5-7 `Fly.m initializing ACPlot.m` 46
Listing 5-8 `Fly.m initializing the time display` 46
Listing 5-9 `Fly.m the simulation loop.` 47
Listing 5-10 `Fly.m simulation control` 48
Listing 5-11 `Fly.m plotting` 48
Listing 5-12 `Adding Actuator Dynamics` 48
Listing 5-13 `The actuator model` 49

Chapter 6 Graphics 51

Figure 6-1 HUD.m 52
Figure 6-2 TimeGUI.m 53
Figure 6-3 DrawAC.m 54
Listing 6-1 `HUD.m` 55
Listing 6-2 `TimeGUI.m` 56
Listing 6-3 `DrawAC.m` 57
Listing 6-4 `ACPlot.m` 57
Listing 6-5 `StateSpacePlot.m` 58

Chapter 7 Designing Controllers 59

Figure 7-1 Block diagram 60
Listing 7-1 `Listing` 61
Figure 7-2 Step response 62
Figure 7-3 LQ GUI 63
Figure 7-4 Step response from the GUI 64
Figure 7-5 SISO inputs 65
Figure 7-6 MapIO 66
Figure 7-7 SISO step response 67
Listing 7-2 `CCVDemo` 68
Figure 7-8 Eigenstructure design GUI 69
Figure 7-9 Step response with eigenstructure assignment 70

Chapter 8 Implementing Controllers 71

Listing 8-1 `AircraftControl.m` 72
Listing 8-2 `Initialization` 73
Listing 8-3 `Update` 74
Table 8-1 Excerpts from `ACControl` 75
Figure 8-1 Control and aircraft response 76
Listing 8-4 `F16Actuator.m` 77
Figure 8-2 Pitch Axis Control Augmentation System 78
Listing 8-5 `Setting up the F16 model` 79
Listing 8-6 `Setting the initial state` 80
Listing 8-7 `Extracting the plant model for the design` 81
Listing 8-8 `State space simulation` 82
Figure 8-3 Step response 83
Listing 8-9 `Pilot pitch rate input` 83
Listing 8-10 `Initialization` 84
Listing 8-11 `Update` 85
Figure 8-4 HUD after the pitch rate has been entered 86
Figure 8-5 Rate response to command 86

Chapter 9 References 87

CHAPTER 1

INTRODUCTION

CHAPTER 1

Introduction

This chapter provides a brief introduction to the Aircraft Control Toolbox. The Aircraft Control Toolbox, for use with MATLAB®, provides you with all of the tools needed to design and test control systems for aircraft—all within the MATLAB environment.

1.1 Key Features

The Aircraft Control Toolbox provides a comprehensive set of functions including:

- aircraft dynamics modeling including flexibility, actuator, sensor and engine dynamics,
- nonlinear models for military and commercial aircraft including subsonic and supersonic aircraft with all data contained in a convenient database format,
- aircraft control system design and analysis including classical, eigenstructure assignment, output feedback and many other design methodologies,

The Aircraft Control Toolbox allows you to design and test control systems in a matter of hours, not days or weeks. You can simulate any kind of aircraft. Changes are easy to make and you have excellent visibility into the resulting software.

Prototyping your control systems and simulation models will reduce both development time and cost. MATLAB frees you from the expensive edit/compile/link cycle because it is interpretive and fully interactive.

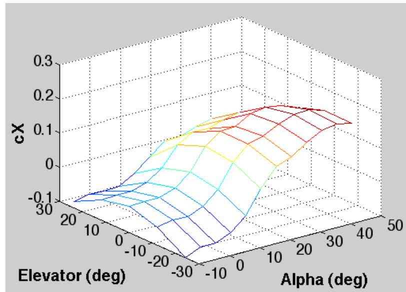
1.2 Aircraft Properties

Aircraft properties are easily accessible to speed your design work. The following plot shows the x-axis aerodynamic force coefficient for a simplified F-16 model obtained by typing

```
F16('cx coeff')
```

You can build your own databases to hold aircraft data using the F-16 database as a template. This way all of your data is organized in one place.

Figure 1-1 Aircraft properties displayed from a database



Many models are included. For example, if you type

```
DC8('inertia')

ans =

     3090000           0       28000
           0     2940000           0
       28000           0     5580000
```

You get the inertia of a DC-8 aircraft.

1.3 Control Design

The toolbox provides a variety of design tools. The toolbox makes use of a state space class that contains all of the information about a state space model including the matrix data, the type of state space system (continuous or otherwise), as well as the names of all of the inputs, outputs and states.

The toolbox offers many control design tools, including

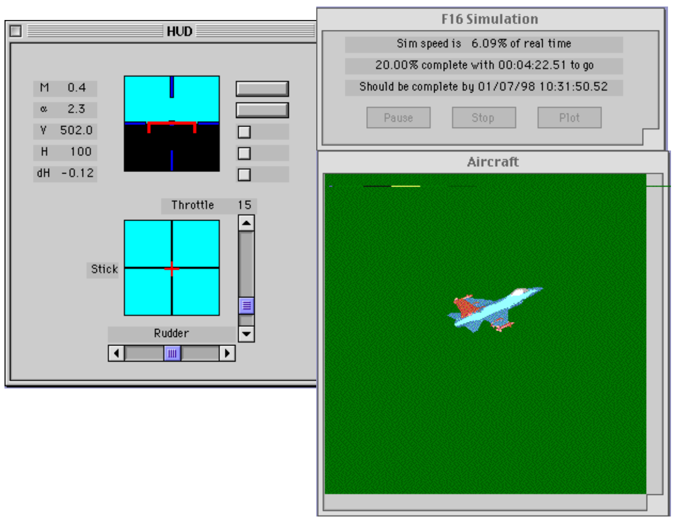
- Output feedback
- Single-input single-output
- Linear quadratic regulator
- Tracking control

- LQ/LTR
- Eigenstructure assignment

1.4 Graphics and Simulation

The toolbox allows you to fly any of your designs using its graphical user interface. The interface is shown below.

Figure 1-2 Control GUI



Using the controls, you can fly your aircraft like any other flight simulator. In this simulator, however, the dynamics are extremely accurate. The nonlinear simulation allows you to add flexible aircraft components, sensor and actuator dynamics, engine dynamics and disturbance dynamics. States for inertia, mass and center-of-gravity are included for vehicles in which the mass properties change significantly. The simulation uses an ellipsoidal earth model so you can simulate aircraft from the ground up into space.

The simulation function will also automatically linearize the nonlinear dynamics and generate a statespace model. In addition, a trimming algorithm is included which can trim your aircraft in a variety of flight modes.

CHAPTER 2

FUNDAMENTALS

This chapter gives you some basic information about the Aircraft Control Toolbox, including: how to use the built-in databases, how the functions are designed, and an introduction to coordinate frames and attitude kinematics.

2.1 Aircraft Properties Databases

All aircraft properties are stored in databases that can be accessed through text-based commands. The toolbox comes with a predefined database of aircraft properties. The following table lists all of the databases included in the toolbox.

Table 2-1 Aircraft Properties

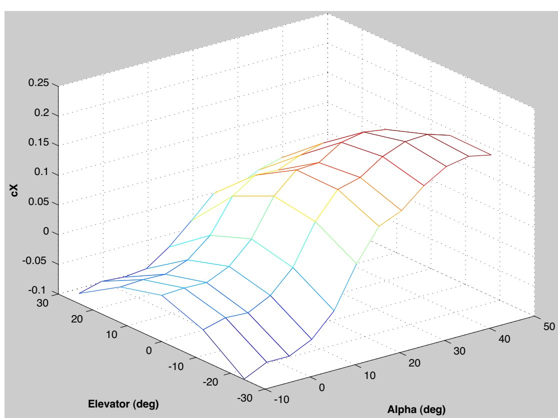
File Name	Type	Description
F16.m	Nonlinear	Tables of aerodynamic coefficients for a simplified F16 model.
DC8.m	Stability Derivatives	Longitudinal and lateral dynamics.

All properties in databases are accessed by passing a text string to the database.

To plot an F16 statespace model type:

```
F16('cx coeff')
```

Figure 2-1 F-16 Properties



2.2 Organizing Your Scripts

It is important to organize your scripts carefully in order to make them readable and easy for other people to use.

The scripts supplied with this package are always organized as:

```
Header

%----------------------------------------------
inr = F16('cx coeff');

%----------------------------------------------
Your design code
```

Since repetitively accessing a database can cause your simulation to be slow, it is recommended that you define local variables to contain the constants or database items.

Variables should always have meaningful names. We recommend the C convention:

```
word1Word2Word3
```

where the beginning of each word after the first is capitalized. If a word is abbreviated the first letter is still capitalized. For example

```
rPM
```

is revolutions per minute. Meaningful variable names reduce the need for comments.

Function names should always begin with a capital letter to distinguish functions from variables. The standard MATLAB functions do not follow this convention.

2.3 Functions

Many of the functions in the toolbox will produce a plot if it is called with no output arguments. In some cases, you do not need any input arguments to get useful plots due to built in default values for the inputs.

Many of the functions in the toolbox are compatible with MATLAB 4.x or earlier. However, many of the newer functions make extensive use of data structures and are only compatible with versions 5.x or newer. We recommend that you get the latest version of MATLAB since in the future we will make even more extensive use of data structures and other object oriented features.

CHAPTER 3

GETTING HELP

Getting Help

This section shows you how to use the help systems built into PSS Toolboxes. There are five sources of help. The first is the standard MATLAB help, the second is the demo functions, the third is the file help function, the fourth is the graphical user interface help system and the fifth is online resources.

3.1 MATLAB Help

You can get help for any function by typing

```
help functionName
```

For example, if you type

```
help c2dzoh
```

you will see the following displayed in your MATLAB command window:

```
-----------------------------------------------------------------
    Create a discrete time system from a continuous system
    assuming a zero-order-hold at the input.
    Given
    .
    x = ax + bu

    Find f and g where
     x(k+1) = fx(k) + gu(k)

-----------------------------------------------------------------
    Form:
    [f, g] = C2DZOH( a, b, T )
-----------------------------------------------------------------
    ------
    Inputs
    ------
    a                    Plant matrix
    b                    Input matrix
    T                    Time step
    -------
    Outputs
    -------
    f                    Discrete plant matrix
```

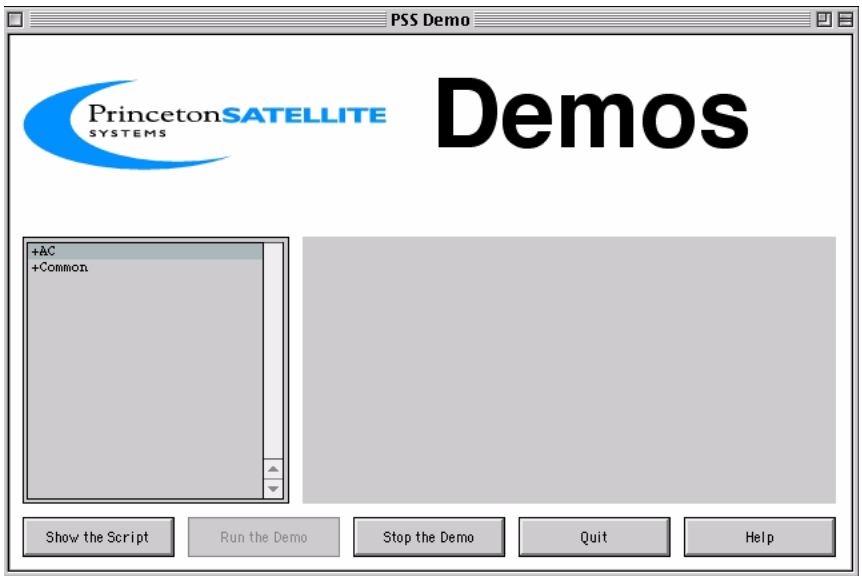
```
    g                        Discrete input matrix

----------------------------------------------------------------
```

All PSS functions have the standard header format shown above.

3.2 Demos

If you type `DemoPSS` you will see the following GUI.

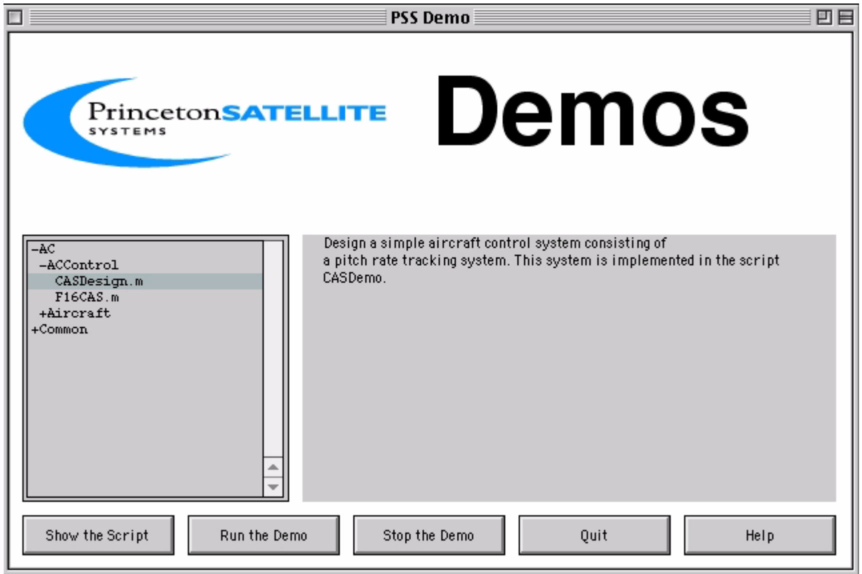
Figure 3-1 DemoPSS



The list on the left-hand-side is hierarchical. The GUI checks to see which directories are in the same directory as `DemoPSS` and lists all directories and files. This allows you to add your own directories and demo files.

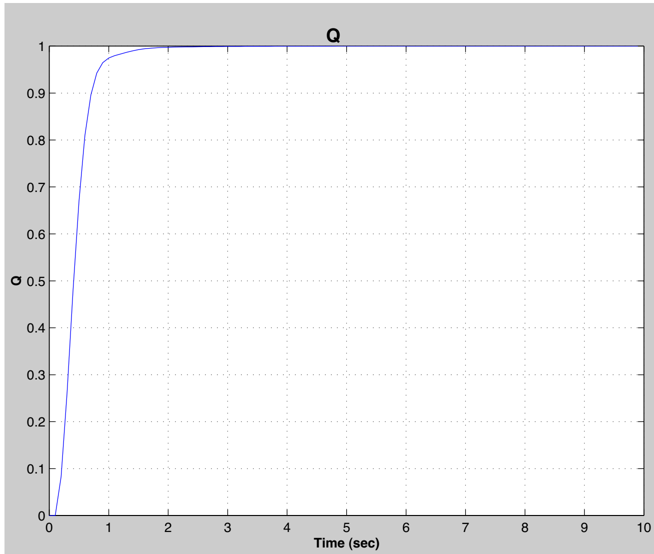
Click on +AC to open the directory. The + sign changes to – and the list changes. Then click on +ACControl.

Figure 3-2 Opening CASDesign



If you would like to look at, or edit, the script, hit "`Show the Script`." Select `CASDesign.m` and hit `"Run The Demo."` If you let it run to completion, several plots will appear. The following is the last plot.

Figure 3-3 Results of `ADSim`



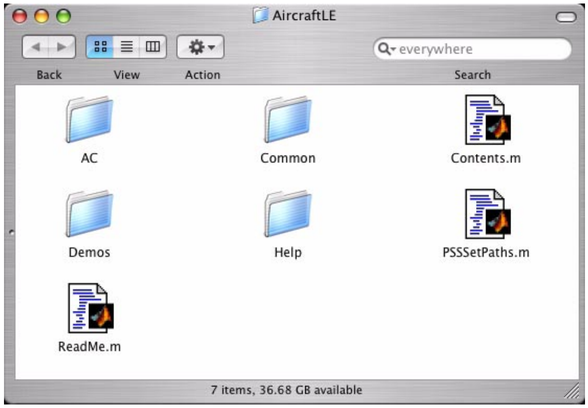
3.3 File Help

3.3.1 Introduction

The FileHelp function gives you access to the headers of all of the functions in the toolbox. You can browse the headers and try out examples associated with each function. You can also edit the examples, create new examples and save them into the help database.

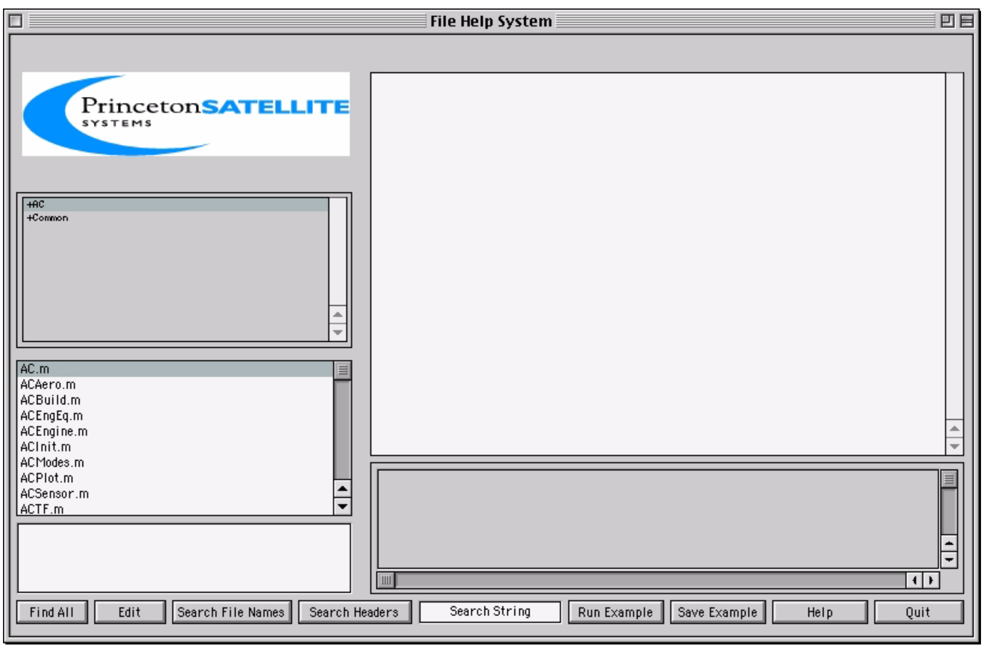
As a reference, the PSSToolboxes folder looks like the following figure.

Figure 3-4 AircraftLE Folder



When you type FileHelp the FileHelp GUI appears.

Figure 3-5 The file help GUI

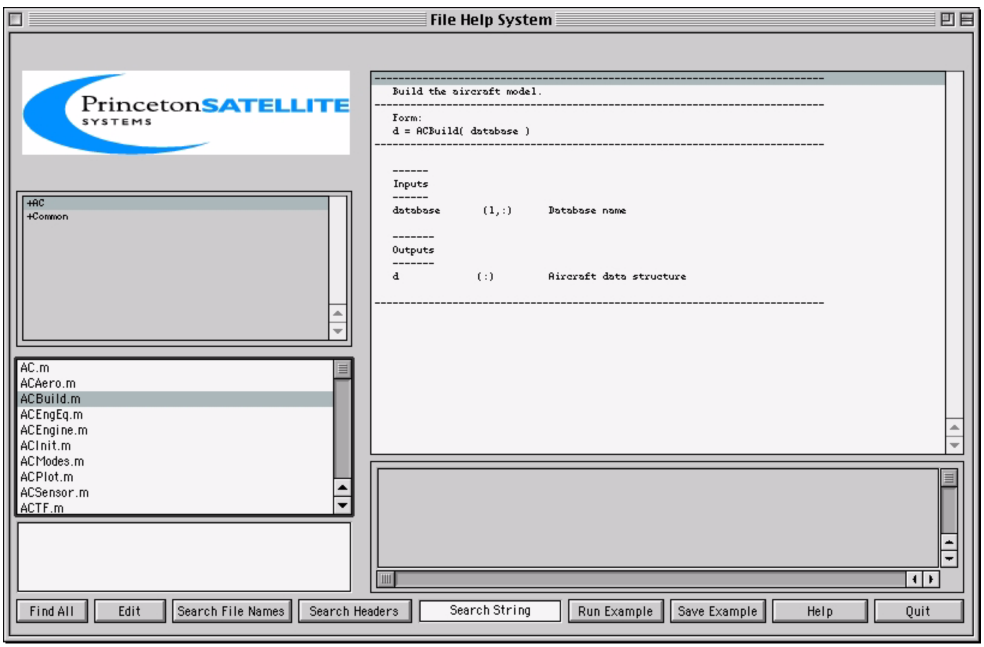


There are five main panes in the window. On the left hand side is a display of all files in the toolbox arranged in the same hierarchy as the PSSToolboxes folder. Below it is a list of all files in alphabetical order. On the right-hand-side is the header display pane. Immediately below the header display is the editable example pane. To its left is a template for the function. You can cut and paste the template into your script or function.

3.3.2 The List Pane

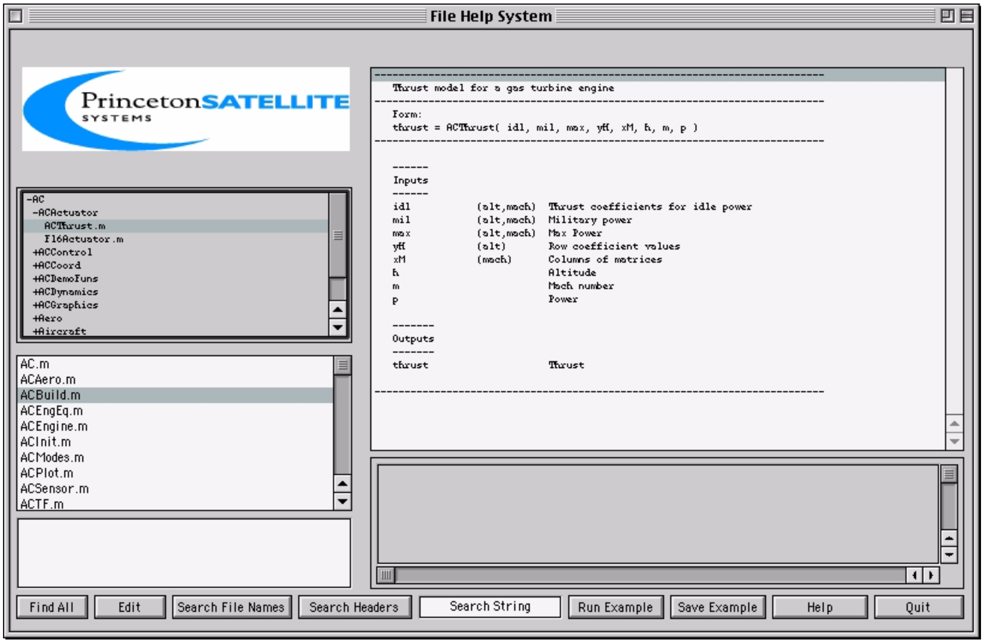
If you click a file in the alphabetical list, the header will appear in the header pane. This is the same header that is in the file. The headers are extracted from a.mat file so changes you make will not be reflected in the file. If the file is a script, a template will not appear, as is the case for this file.

Figure 3-6 Selecting from the alphabetical display



You can also use the hierarchical list. Any name with a + or - sign is a folder. Click on the folders until you reach the file you would like. When you click a file, the header and template will appear.

Figure 3-7 Using the hierarchical list



3.3.3 Edit Button

This opens the MATLAB edit window for the function selected in the list.

3.3.4 The Example Pane

This pane gives an example for the function displayed. Not all functions have examples. The edit display has scroll bars. You can edit the example, create new examples and save them using the buttons below the display. To run an example, push Run Example button.

You can include comments in the example by using the percent symbol.

3.3.5 Run Example Button

Run the example in the display. Some of the examples are just the name of the function. These are functions with built-in demos. Results will appear either in separate figure windows or in the MATLAB Command Window.

3.3.6 Save Example Button

Save the example in the edit window. Pushing this button only saves it in the temporary memory used by the GUI. You can save the example permanently when you Quit.

3.3.7 Help Button

Open the help system.

3.3.8 Quit

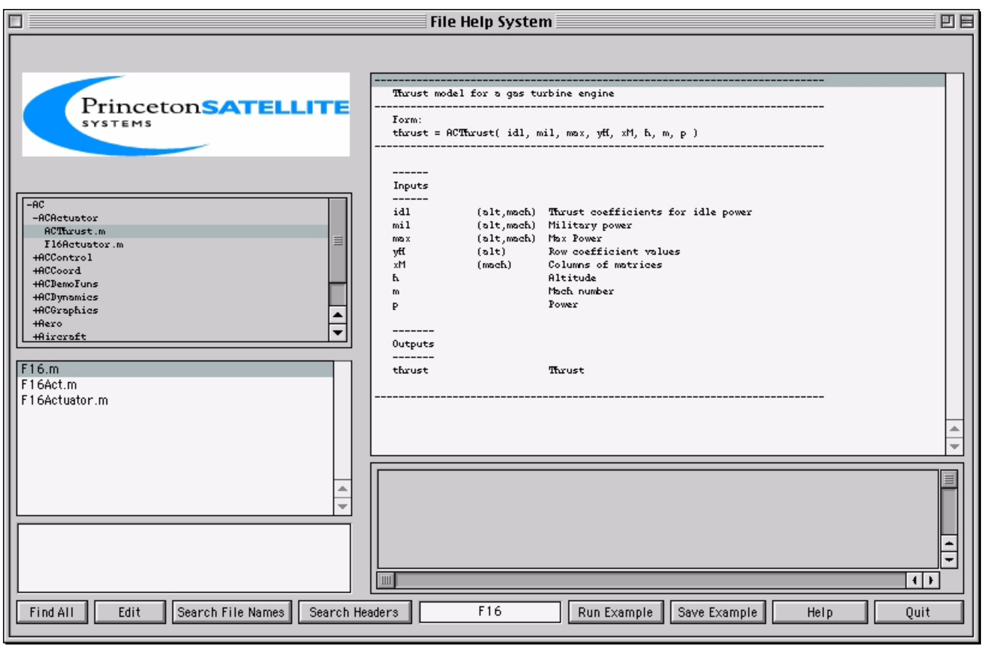
Quit the GUI. If you have edited an example, it will ask you whether you want to save the example before you quit.

3.4 Searching in File Help

3.4.1 Find

Type in a name in the edit box and push Search File Names.

Figure 3-8 Search Results



All files with "Att" appear in the alphabetical list.

3.4.2 Find All Button

Find all returns to the list of the functions.

3.4.3 Search Headers Button

Search headers for a string. This function looks for exact, but not case sensitive, matches. The file display displays all matches. A progress bar gives you an indication of time remaining in the search.

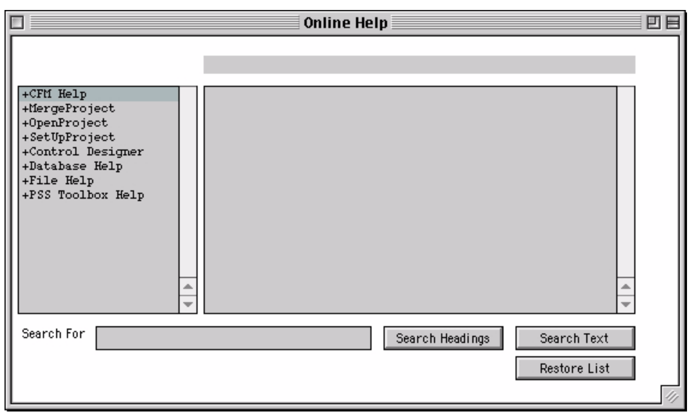
3.4.4 Search String Edit Box

This is the search string. Spaces will be matched so if you type "attitude control" it will not match "attitude control" (with two spaces.)

3.5 Graphical User Interface Help

Each graphical user interface (GUI) has a help button. If you hit the help button a new GUI will appear.

Figure 3-9 Help GUI



You can access on-line help about any of the GUIs through this display. It is separate from the file help GUI described above.

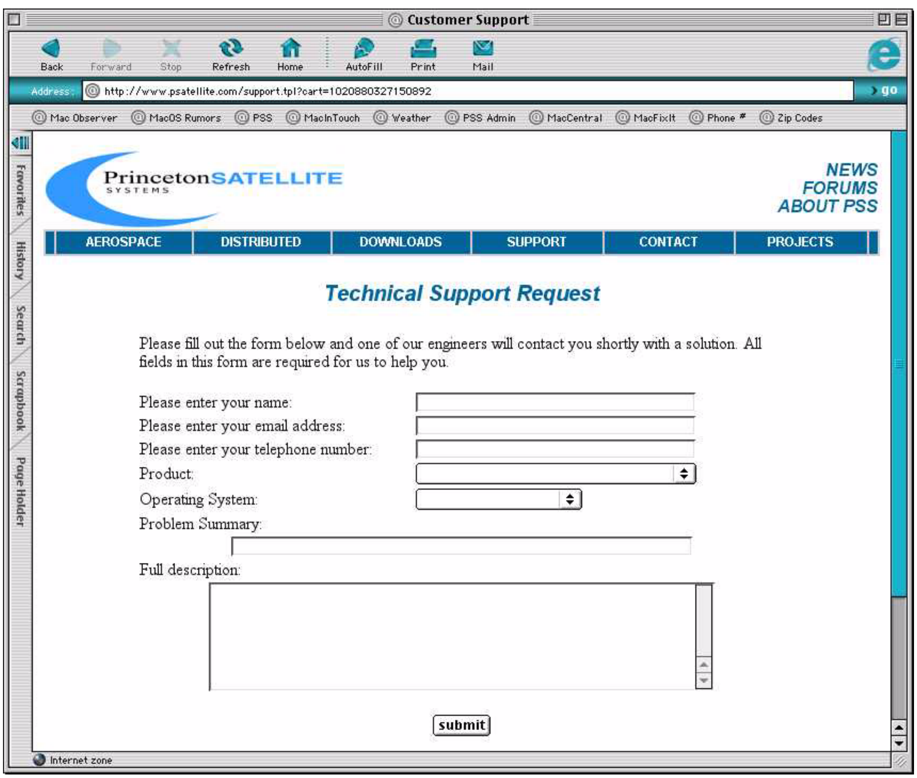
The display is hierarchical. Any list item with a + or – in front is a directory. + means the directory list is closed, – means it is open. Clicking on a directory name toggles the directory open or closed. If you click on a file name in the list you will get a text display in the right-hand pane.

You can either search the headings or the text by entering a text string into the Search For edit box and hitting the appropriate button. Restore List restores the list window to its previous configuration.

3.6 Technical Support

Contact support@psatellite.com for email technical support. You can submit technical questions and search the database of technical questions on the PSS website www.psatellite.com. The Tech Support page is shown in Figure 3-10.

Figure 3-10 PSS web technical support



CHAPTER 3

Getting Help

CHAPTER 4

STRUCTURES

CHAPTER 4

Structures

MATLAB 5.x and 6.x have a number of useful new data types. These are used extensively in the toolbox. All of the CAD and GUI functions use them.

This chapter discusses data structures and cell arrays and gives some tips on how to use them.

4.1 Data Structures

Data structures allow you to collect disparate data elements into a single variable. For example, suppose you needed to pass the name of a sensor and its unit boresight vector to a function. You might write

```
u = [1;0;0];
name = 'Sensor A'
x = Sensor( name, u );
```

With data structures you can write

```
a = struct( 'u' , [1;0;0], 'name' , 'Sensor A'  )
```

or

```
a.u = [1;0;0];
a.name = 'Sensor A'
```

instead. Now your function call is

```
x = Sensor( a );
```

You can imagine how much more convenient passing a data structure is than passing a long list of inputs.

If you type b = a;

b will be

```
b.u = [1;0;0];
b.name = 'Sensor A'
```

You cannot add data structures or use arithmetic operations on them.

You can have an array of data structures. For example:

```
u(1).a = eye(3);
u(1).b = rand(3,4)
u(2).a = 6*eye(3);
u(2).b = rand(3,4);
```

4.2 Cell Arrays

A cell array is an array in which any element can contain any other type of data structure. For example, you could implement the above data structure with a cell array

```
b{1} = [1;0;0];
b{2} = 'Sensor A' ;
```

Unlike data structures, you can concatenate cell arrays. The following

```
c = {a{:} b{:}}
```

would give you

```
c{1} = [1;0;0];
c{2} = 'Sensor A' ;
c{3} = [1;0;0];
c{4} = 'Sensor A' ;
```

You can perform operations on cell contents. For example

```
a{1} = diag([1 2 3]);
b{1} = [1;1;1];
```

when multiplied together give

```
[1;2;3];
```

Cell arrays are a convenient way of storing strings. For example you could write

```
a = ['First String      ' ;' Second Long String' ];
```

being careful to make sure they were the same length, or write

```
a{1} = 'First String' ; a{2} = 'Second String' ;
```

`uicontrol` functions will often take cell arrays of strings. It is convenient to lump `uicontrol` properties into a cell array:

```
v = { 'parent', h.fig, 'fontunits', 'pixels', 'fontsize',
12, 'horizontalalignment', 'left' };
```

and call `uicontrol` as

```
uicontrol( v{:}, ...
```

4.3 Classes

Classes are a form of data structure in which both the data and the operations that can be done on the data are defined together. The toolbox includes several classes. For example you might define a class called `names`. You could then create the method "+" which would overload the MATLAB "+" function so that if you let

```
a = names( 'Emily' );
b = names( 'Stuart' );
```

then

```
c = a + b;
```

would be the same as

```
c = names( 'Emily Stuart' )
```

`names` would be the class constructor and "+" is a method that overloads "+".

An important aspect of a class is that you cannot get access to the internal data structure from outside of a class method. This allows the class designer control access to the data so that the user cannot use it in an incorrect manner. Object oriented design terms related to classes are listed in the following table.

Figure 4-1 Object oriented programming terms

Term	Definition
class	A data structure definition and functions that operate on that data structure.
constructor	A method that creates an object of type class.
instance	A variable of type class. In Matlab if you type x = 2 you create an instance of class double.
method	A function that is part of a class.
object	An instance of a class. When you type x = 2, you create an object of class double.
overloading	Giving a meaning to an operation that is specific to a class. For example, in the statespace class + means parallel connection of state space systems.
polymorphism	When a function behaves differently for different types of inputs.

CHAPTER 4

Structures

CHAPTER 5

SIMULATION

CHAPTER 5

Simulation

This chapter describes how to use the Aircraft Control Toolbox to build simulations of your Aircraft.

5.1 Simulation

5.1.1 Introduction

When we talk about simulation, it is convenient to break it into two categories, linear and nonlinear. Aircraft dynamics are inherently nonlinear, and most aircraft actuators and sensors are nonlinear. Nonetheless, it is usually possible to linearize the dynamics and devices about some operating point where, in a sufficiently restricted region, the system behaves linearly. This is the basis for the linear control laws developed in this toolbox. The toolbox uses the function AC.m for all aircraft simulations. With appropriate plug-in functions it can perform very sophisticated simulations of anything from a biplane to a single stage to orbit launch vehicle.

5.1.2 Aspects of Simulation Models

Aircraft simulations can range from simple three degree of freedom longitudinal dynamics models to models that incorporate the dynamics of moving parts, aeroelasticity, dynamical engine models, pilot dynamics and so forth. There are two major tools for simulation in the toolbox. One is to use the statespace models for linear simulations. The other is to use the nonlinear simulation, AC.m.

Two convenient statespace simulation tools are Step.m and IC.m. The first does step responses and the second does responses to an initial state vector. Another useful tool is MRS.m which computes mean squared responses of a system to noise inputs.

The following table lists different features that simulation models can have and shows which ones are available in AC.m.

Table 5-1 Models

Feature	In AC?	Description	Uses
Rigid Body (6-DOF)	4	Three rotational and three translational degrees of freedom. Six kinematic states (seven if quaternions are used) are needed.	All aircraft
Flat Earth	4	Constant gravity. No earth curvature.	Most aircraft
Ellipsoidal Earth	4	Includes rotation of the earth and altitude dependent gravity.	Launch vehicles

Table 5-1 Models

Feature	In AC?	Description	Uses
Rotating parts	4	Spinning parts, such as gas turbines or a gatling gun on an A-10.	All aircraft with engines.
Actuator Dynamics	4	Nonlinear models that relate commanded thrust, aileron angle, etc. to the actual angle. Accommodates lags, delays, limits.	All aircraft but not always necessary for preliminary designs.
Sensor Dynamics	4	Nonlinear models that relate measured quantity to its measurement.	See above
Flex	4	Bending of wings, etc.	Important for evaluating aeroelastic effects.
Time varying inertia and mass	4	As fuel is consumed the inertia and mass change.	Launch vehicles.
Inertia and mass of moving parts		On some aircraft (and on boosters with large gimballed nozzle assemblies) the dynamics of moving parts can be significant.	Light aircraft and some boosters.
Detachable parts		Bombs and missiles.	Military aircraft.
Thermal effects		Interaction of heating and aerodynamics	Supersonic aircraft.

Two demos show how to use AC.m with the F16.m database. The first is CTSim.m which simulates a coordinated turn. The second is Fly.m which lets you fly the F16 using the heads up display, HUD.m. The steps you take to set up a simulation are:

- Trim the model using ACTrim.m
- Initialize the model data structures and state vector using ACBuild.m and ACInit.m.
- Run AC.m.
- Get plot results with ACPlot.m

5.1.3 Linear

5.1.3.1 Creating a State Space System

If you have your model in transfer function form it can be converted to state space form using

```
[a,b,c,d] = ND2SS( num, den );
```

`num` can have more than one row. To make it of type statespace

```
g = statespace( a, b, c, d );
```

If you have a nonlinear system expressed in the form

$$\dot{x} = f(x, u)$$

and f is a Matlab function in the form

```
xDot = F(x,u);
```

then

```
[a,b] = Jacobian('f',x,u);
```

5.1.3.2 Zero Order Hold

The simplest way to simulate a continuous time system is to discretize it using the zero order hold. This toolbox gives two ways to do this. One is the standard zero order hold

```
[aD,bD] = C2DZoh(a,b,T);
```

and the simulation is

```
x = aD*x + bD*u;
y = c*x + d*u;
```

The second is the delta form of the zero order hold

```
[aD,bD] = C2DelZoh(a,b,T);
```

and the simulation is

```
x = x + aD*x + bD*u;
y = c*x + d*u;
```

These approximations assume that the input is held constant over the interval `T`.

5.1.3.3 First Order Hold

An alternative approach is to discretize the system using a first order hold. This approximation assumes that the input varies linearly from step `k` to step `k + 1`.

```
[aD,bD,cD,dD] = C2DFOH(a,b,T);
```

and the simulation is

```
x = aD*x + bD*u;
y = cD*x + dD*u;
```

5.1.4 Nonlinear

The toolbox provides several functions for nonlinear simulations. These functions do not vary the step size automatically or perform any error testing. One has to be careful since a large integration time step can introduce instabilities or artificial damping into systems.

The Aircraft Control Toolbox also provides a variable step size routine, RK45, and Euler, a first order method.

Given the function

```
xDot = Fun(x,t,p1,p2...p10)
```

and time step `h` use either

```
x = RK2( 'Fun', x, h, t, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10 )
```

or

```
x = RK4( 'Fun', x, h, t, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10 )
```

`t` (time) and `p1` through `p10` are optional arguments. If you need more than 10 optional arguments you can pack `p1` through `p10`. For example if you need to pass two inertia matrices

```
p1 = [inertia1,inertia2];
```

5.2 Creating an Interactive Simulation

Fly.m is a complete, nonlinear, interactive simulation that uses all of the toolbox GUIs to allow you to fly an F-16.

In this section we walk through the script Fly.m and explain in detail how it works. A summary of how to set up simulation scripts has already been given above so we will jump right into the details.

Listing 5-1 Fly.m initialization

```
% Clean up
%---------
close all
clear all

% Global for the time GUI
%------------------------
global simulationAction
simulationAction = ' ';
% Global for the HUD
%-------------------
global hUDOutput
hUDOutput = struct('pushbutton1',0,'pushbutton2',0,'checkbox1',0,...
                   'checkbox2',0,'checkbox3',0);
% F16 database
%-------------
d                = ACBuild('F16');
d.theta0         = 0;
d.wPlanet        = [0;0;0];
d.actuator.name  = [];
d.aero.name      = 'ACAero';
d.engine.name    = 'ACEngine';
d.rotor.name     = [];
d.sensor.name    = 'ACSensor';
d.disturb.name   = [];

% Load the standard atmosphere
%----------------------------
load -ascii AtmData;

d.atmData        = AtmData;
d.atmUnits       = 'eng';
```

In Listing 5-1 on page 42 we clean up the workspace, define a global variable for TimeGUI.m and build the aircraft data structure, d. The name fields are names of functions that implement different aspects of the model. ACAero.m, ACEngine.m and ACSensor.m are models included with the toolbox. You can write your own models and use AC.m as long as you adhere to the input/output conventions for each of the functions. Type "help AC" for more information.

The last code loads data for the standard atmosphere and specifies the units as English (ft.).

The following code initializes the controls. These values trim the aircraft.

Listing 5-2 Fly.m control initialization

```
% Control
%--------
d.control.throttle  =  0.1485;
d.control.elevator  =  -1.931;
d.control.aileron   = -7e-8;
d.control.rudder    =  8.3e-7;
```

The state vector is specified in terms of angle-of-attack (alpha), sideslip (beta), and total velocity (vT). These are converted into a body state vector by the function VTToVB. The cG, inertia and mass are also states and are specified. The simulation uses quaternions and QECI converts the initial euler angles and position vector to the quaternion from ECI to the body frame. The engine model has a single state and it is found by ACEngEq which takes the aircraft data structure (which contains the control) and finds the engine equilibrium state at that control setting. There are no actuator, sensor, flex or disturbance states so they are set to empty matrices.

The initial time is specified and the state vector, x, of type acstate is created using the constructor acstate.

Simulation

The time step is step to 0.01 sec and the number of integration steps are computed.

Listing 5-3 Fly.m initializing the state vector

```
% Initial state vector
%---------------------
alpha     = 0.03936;
beta      = 4.1e-9;
vT        = 502;
v         = VTToVB( vT, alpha, beta );
cG        = [0.3;0;0];
r         = [2.092565616797901e+07+100;0;0];
eulInit   = [0;0.03936;0];
q         = QECI( r, eulInit );
w         = [0;0;0];
wR        = 160;
engine    = ACEngEq( d, v, r );
mass      = 1/1.57e-3;
inertia   = [9497;55814;63100;0;-982;0];
actuator  = [];
sensor    = [];
flex      = [];
disturb   = [];

% Initial time and state
%-----------------------
t = 0;
x = acstate( r, q, w, v, wR, mass, inertia, cG, engine, actuator, sensor,
flex, disturb );

% Initialize the model
%---------------------
dT   = 0.1;
nSim = 20/dT;
```

The linearized plant model is computed, just for information purposes. ACModes extracts the standard aircraft rigid body modes. ACModes only works if the aircraft is flying straight and level.

Listing 5-4 Fly.m Getting the linearized model

```
d    = ACInit( x, d );
gLin = AC( x, 0, 0, d, 'linalpha');
aC   = get( gLin, 'a' );

% Display aircraft rigid body modes
%-----------------------------------
ACModes( gLin );
```

Setting up the displays is discussed in the graphics section. The settings for the control maximums

Listing 5-5 Fly.m setting up the HUD

```
% Set up the HUD
%---------------
dHUD.atmData  = AtmData;
dHUD.atmUnits = 'eng';

cHUD.control     = d.control;
cHUD.elevatorMax = 90;
cHUD.aileronMax  = 90;
cHUD.rudderMax   = 90;
hHUD = HUD( 'init', dHUD, x, [], cHUD );
```

is used to translate mouse movement into control.

Listing 5-6 Fly.m setting up the aircraft 3D display

```
% Set up the aircraft display
%-----------------------------
load gF16
hF16 = DrawAC( 'init', gF16, x );
```

Simulation

Plotting is initialized by specifying the names of plots. ACPlot.m lists all available plots.

Listing 5-7 Fly.m initializing ACPlot.m

```
% Initialize the plots
%---------------------
plots = [ 'Euler angles       ';...
          'Quaternion         ';...
          'Quaternion NED To B';...
          'Angular rate       ';...
          'Position ECI       ';...
          'Velocity           ';...
          'Alpha              ';...
          'Rudder             ';...
          'Throttle           ';...
          'Aileron            ';...
          'Elevator           '];

dPlot = ACPlot( x, 'init', plots, d, nSim, dT, nSim );
```

The time display is discussed in the graphics section.

Listing 5-8 Fly.m initializing the time display

```
% Initialize the time display
%----------------------------
tToGoMem.lastJD         = 0;
tToGoMem.lastStepsDone = 0;
tToGoMem.kAve           = 0;
ratioRealTime           = 0;
nTTGo                   = 10;
[ ratioRealTime, tToGoMem ] =  TimeGUI( nSim, 0, tToGoMem, 0, dT, 'F16 
Simulation' );
```

The first section of the simulation loop updates the time display periodically. The next sections update the HUD and extract the control settings. Data storage for the plots is done next. The 3D display is updated and then the simulation state is updated.

Listing 5-9 Fly.m the simulation loop.

```
for k = 1:nSim

 % Display the status message
  %---------------------------
  if( floor(k/nTTGo) == k/nTTGo )
     [ ratioRealTime, tToGoMem ] = TimeGUI( nSim, tToGo Mem, ratioRealTime, dT );
  end

  % HUD information
  %----------------
  hHUD = HUD( 'run', dHUD, x, hHUD, cHUD );

  % Controls
  %---------
  d.control  = hHUD.control;

  % Plotting
  %---------
  dPlot = ACPlot( x, 'store', dPlot, d.control );

  % 3D Display
  %-----------
  hF16 = DrawAC( 'run', gF16, x, hF16 );

  % The simulation
  %---------------
  x   = AC( x, t, dT, d );
  t   = t + dT;
```

This code shows the end of the simulation loop. This code implements commands from

Listing 5-10 Fly.m simulation control

```
   % Time control
   %-------------
   switch simulationAction
     case 'pause'
       pause
       simulationAction = ' ';
     case 'stop'
       return;
     case 'plot'
       break;
   end
   HUDCntrl;
 end
```

TimeGUI.m.

The final snippet is the plotting code.

Listing 5-11 Fly.m plotting

```
 % Create the plots
 %-----------------
 ACPlot( x, 'plot', dPlot );
```

Run Fly.m to see how it all works!

5.3 Customizing a Simulation

You can add sensor, actuator and flex dynamics to the simulation by plugging in your own routines. For example, the script CResponse.m shows the aircraft response to a variety of control inputs. The script CActuator.m is the same script but with first order actuator dynamics added. Two things are needed to add actuator dynamics. The first is a few changes to CResponse.m shown in Code Sam-

Listing 5-12 Adding Actuator Dynamics

```
 d.actuator = struct('name','F16Act','aileron',2,'elevator',2,'rudder',2);
 actuator   = [d.control.elevator;d.control.aileron;d.control.rudder];
```

ple (5-12). The first line creates a data structure for the data needed by the actuator model. In this case, the actuators are modeled as first order lags. The first member of the structure is the name of the function that models the actuator. The last three members are the break frequencies for each actuator model. The second line initializes the actuator state to the current value of the controls.

The next part is the actuator model shown in Code Sample (5-13). `x` is the actuator part of the state

Listing 5-13 The actuator model

```
function [dX, control] = F16Act( x, controlInput, actuatorData )
control.throttle = controlInput.throttle;
control.elevator = x(1);
control.aileron  = x(2);
control.rudder   = x(3);
dX    = zeros(3,1);
dX(1) = actuatorData.elevator*(controlInput.elevator - x(1));
dX(2) = actuatorData.aileron *(controlInput.aileron  - x(2));
dX(3) = actuatorData.rudder  *(controlInput.rudder   - x(3));
```

vector, initialized above. `controlInput` is the control data structure, used to initialize the actuator state vector above, and `actuatorData` is the actuator data structure, `d.actuator`.

CHAPTER 5

Simulation

CHAPTER 6

GRAPHICS

CHAPTER 6

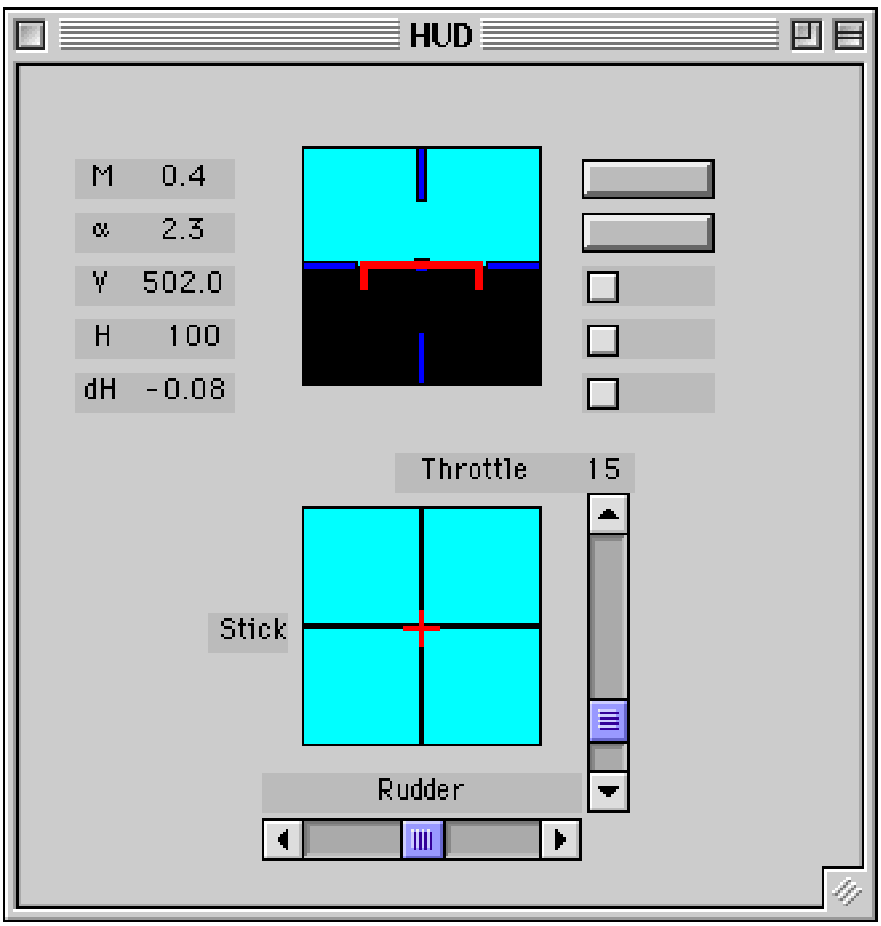
Graphics

This chapter describes how to use the Aircraft Control Toolbox graphics.

6.1 GUIs

The toolbox has three GUI windows that you can use in your simulations. Each GUI has an initialization function call format and a run-time function call format. The three GUIs are shown in the following figures. The first is HUD.m a "Head-Up Display" that allows you to control your aircraft

Figure 6-1 HUD.m



model. It can be used with any simulation. It has an airplane mode and a helicopter mode. You move the sliders for pedal and throttle and move the box in the lower display by clicking on the new desired location. For an airplane this causes the ailerons or elevators to move. The numerical displays on the left are Mach number, angle of attack in degrees, velocity, altitude and altitude rate. The two push buttons and three checkboxes can be assigned names and functions by the user.

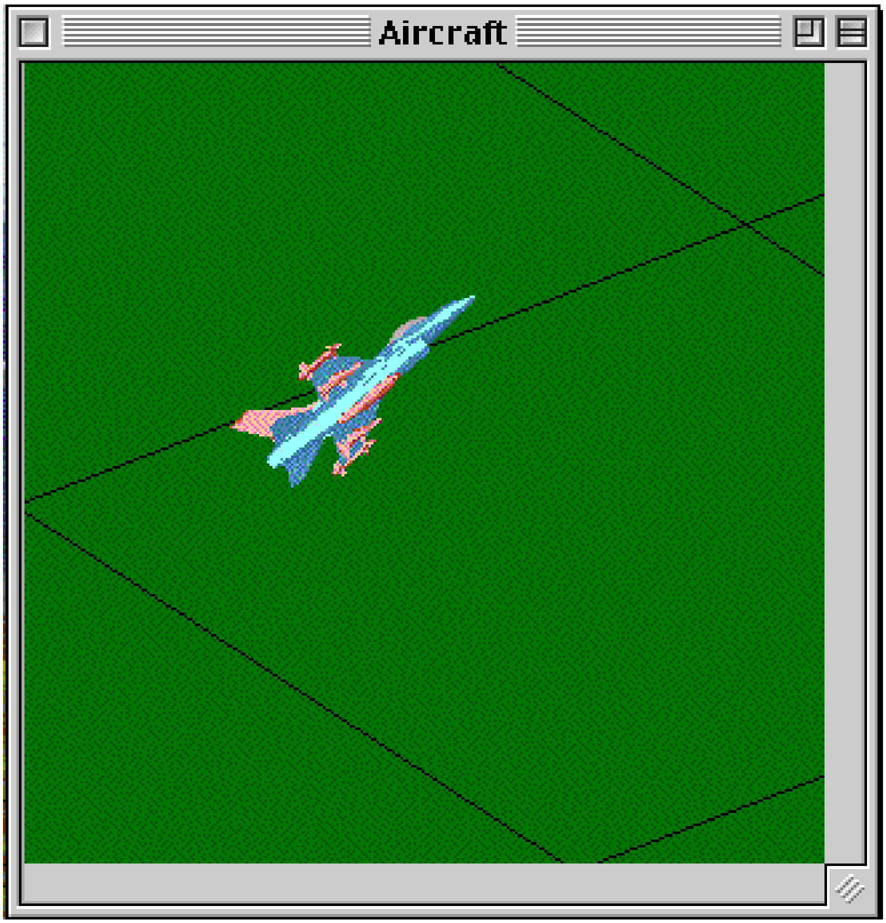
The second is TimeGUI.m which lists time statistics and allows you to control your simulation. By

Figure 6-2 TimeGUI.m

pushing one of the three buttons you can stop the simulation, pause, or exit the simulation loop. If you use one of the toolbox plotting routines, exiting will cause all existing data to plot.

The last is the aircraft display, DrawAC.m which gives you a 3-dimensional picture of what your aircraft is doing. Any aircraft model can be loaded into the display. The toolbox supplies a prepro-

Figure 6-3 DrawAC.m



cessed F-16 model as an example.

The following demos show you how to write the code in each case. All are excerpts from the demo

Listing 6-1 HUD.m

```
% Global for the HUD
%--------------------
global hUDOutput
hUDOutput = struct('pushbutton1',0,'pushbutton2',0,'checkbox1',0,...
                   'checkbox2',0,'checkbox3',0);

% Set up the HUD
%---------------
dHUD.atmData  = AtmData;
dHUD.atmUnits = 'eng';

cHUD.control     = d.control;
cHUD.elevatorMax = 90;
cHUD.aileronMax  = 90;
cHUD.rudderMax   = 90;
hHUD = HUD( 'init', dHUD, x, [], cHUD );
for k = 1:nSim
  % HUD information
  %----------------
  hHUD = HUD( 'run', dHUD, x, hHUD, cHUD );

  % Controls
  %---------
  d.control  = hHUD.control;
  HUDCntrl;
end
```

Fly.m. The dHUD and cHUD structures set up the HUD. dHUD has a third field, .type, that lets you select helicopter or aircraft mode. If omitted, HUD defaults to aircraft. HUDCntrl updates the state of the push buttons

The TimeGUI function uses a global variable, `simulationAction`, to communicate with the

Listing 6-2 TimeGUI.m

```
global simulationAction
simulationAction = ' ';

tToGoMem.lastJD        = 0;
tToGoMem.lastStepsDone = 0;
tToGoMem.kAve          = 0;
r     = 0;
nTTGo = 10;
[ r, tToGoMem ] =  TimeGUI( nSim, 0, tToGoMem, 0, dT, 'F16 Simulation' );
for k = 1:nSim

  if( floor(k/nTTGo) == k/nTTGo )
    [r, tToGoMem ] =  TimeGUI( nSim, k, tToGoMem, r, dT );
  end

  switch simulationAction
    case 'pause'
      pause
      simulationAction = ' ';
    case 'stop'
      return;
    case 'plot'
      break;
  end
end
```

script. It is the <u>only</u> global variable used in the toolbox.

`gF16` in the following demo contains the data structures for the F-16 3D model.

Listing 6-3 DrawAC.m

```
% Set up the aircraft display
%----------------------------
load gF16
hF16 = DrawAC( 'init', gF16, x );
for k = 1:nSim
  % 3D Display
  %-----------
  hF16 = DrawAC( 'run', gF16, x, hF16 );
end
```

6.2 Plotting

The toolbox has two plotting functions ACPlot.m and StateSpacePlot.m. The former is for use with the acstate class and the latter with the statespace data class. The following demo from Fly.m shows how to use ACPlot.m. The variable "plots" contains the names of the desired plots. This example

Listing 6-4 ACPlot.m

```
% Initialize the plots
%---------------------
plots = [ 'Euler angles       ';...
          'Quaternion         ';...
          'Quaternion NED To B';...
          'Angular rate       ';...
          'Position ECI       ';...
          'Velocity           ';...
          'Alpha              ';...
          'Rudder             ';...
          'Throttle           ';...
          'Aileron            ';...
          'Elevator           '];

dPlot = ACPlot( x, 'init', plots, d, nSim, dT, nSim );
for k = 1:nSim
      dPlot = ACPlot( x, 'store', dPlot, d.control );
end
```

will plot data on every pass through the loop but you can control that using the inputs to ACPlot.

StateSpacePlot.m is similar. You must combine the outputs from HUD.m into u in this example.

Listing 6-5 StateSpacePlot.m

```
dPlot   = StateSpacePlot( 'init', plots, nSim, nSim );
for k = 1:nSim

  % Controls
  %---------
  u = [hHUD.control.collective;...
          hHUD.control.longitudinalCyclic;...
          hHUD.control.lateralCyclic;...
          hHUD.control.rudder];
  dPlot = StateSpacePlot( 'store', x, [], u, dPlot );
end
```

CHAPTER 7

DESIGNING CONTROLLERS

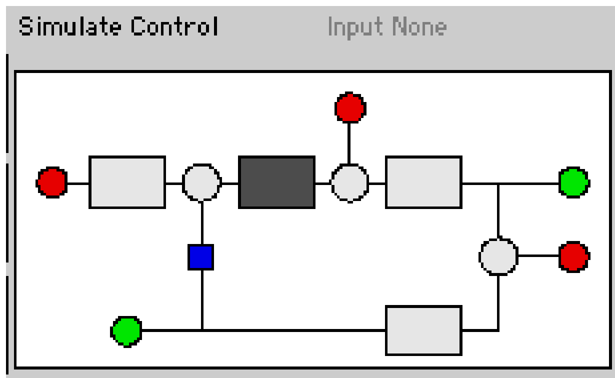
Designing Controllers

This chapter shows how to design controllers using the ControlDesignPlugin. The three major methodologies are discussed, Linear Quadratic, Eigenstructure assignment and Single-Input-Single-Output. This section focuses on how to use the Control Designer GUI.

7.1 Using the block diagram

The block diagram from the control designer GUI is shown in the following figure.

Figure 7-1 Block diagram



When you select a block, all operations (including all of the simulation buttons, loading and saving) apply only to that block. To work with the entire diagram click the highlighted block so that none are highlighted. The blue box opens and closes the control loops. When it is blue (the default) the system is closed. To open the loops, click the box.

The red circles are inputs and the green are outputs. When you are working with the entire system you can select the input and output points by clicking on the red and green circles. The red circle on the left is the command input, the one on the top is the disturbance input and the one on the right is the noise input. The green output on the right is the state output and the green output on the left is the measurement output.

7.2 Linear Quadratic Control

In this example we will design a compensator for a double integrator using full-state feedback. A double integrator's states are position and velocity. For full-state feedback, both must be available.

This example is automated using the `LQFullState.m`.

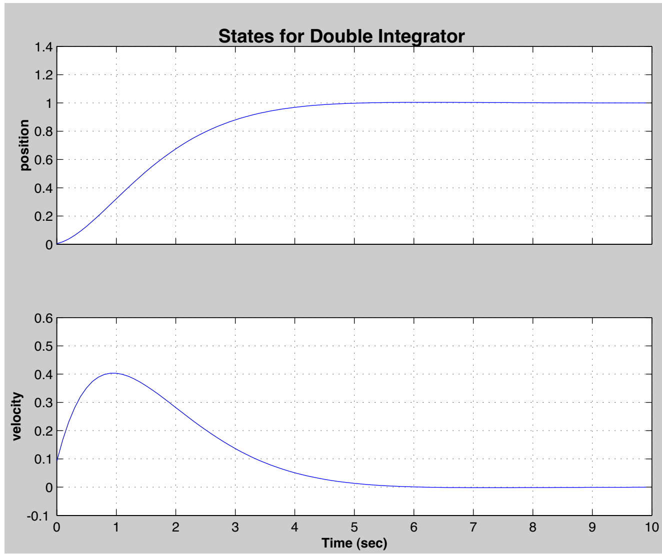
Listing 7-1 Listing

```
a          = [0 1;0 0];
b          = [0;1];
c          = eye(2);
d          = [0;0];

g          = statespace( a, b, c, d, 'Double Integrator',...
            {'position', 'velocity'}, 'force', {'position', 'velocity'}
);

save( 'DoubleIntegrator', 'g' );

q          = eye(2);
r          = 1;

w.q        = q;
w.r        = r;

gC         = LQC( g, w, 'lq' );
k          = get( gC, 'd' );

[a,b,c,d] = getabcd( g );
inputs     = get( g, 'inputs' );
inputs     = strvcat( inputs, 'pitch rate' );
g          = set( g, a - b*k*c, 'a' );
Step( g, 1, 0.1, 100 );
```

The script sets values for the controller design matrices. As you can see, you can also use `LQC.m` outside of the design GUI. This script also creates the plant model, `DoubleIntegrator.mat`.

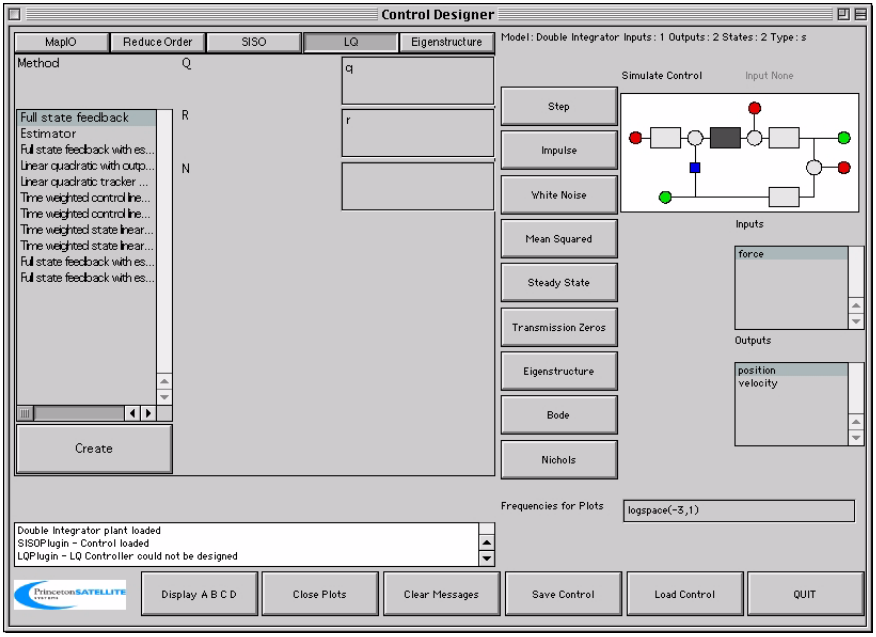
Run the script and you will get the plot.

Figure 7-2 Step response



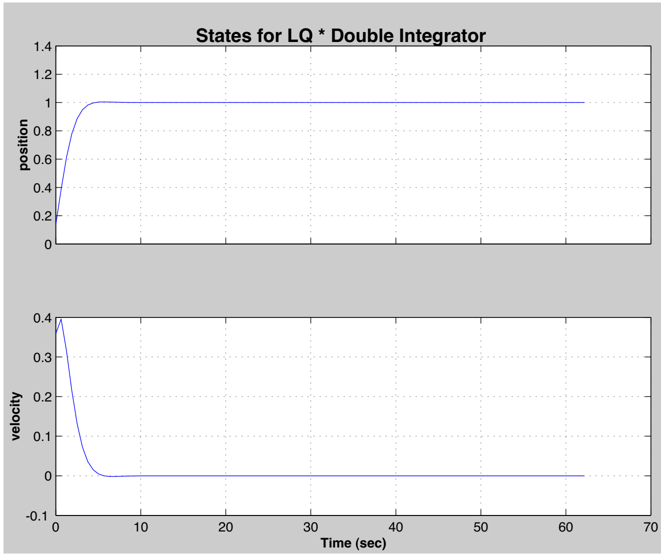
Now type `ControlDesignPlugin`. Select the plant and load in DoubleIntegrator.mat. Select the control and then select the LQ tab. Select full state feedback. Enter q and r into the corresponding input fields. The display will look as follows. Push create. The values for q and r are read in from the workspace. This eliminates the need to type in potentially large matrices. When you read in a controller these matrices are stored in the workspace.

Figure 7-3 LQ GUI



Next click the control block so that you get the whole system. It will unhighlight. You can now do a step response by pushing step.

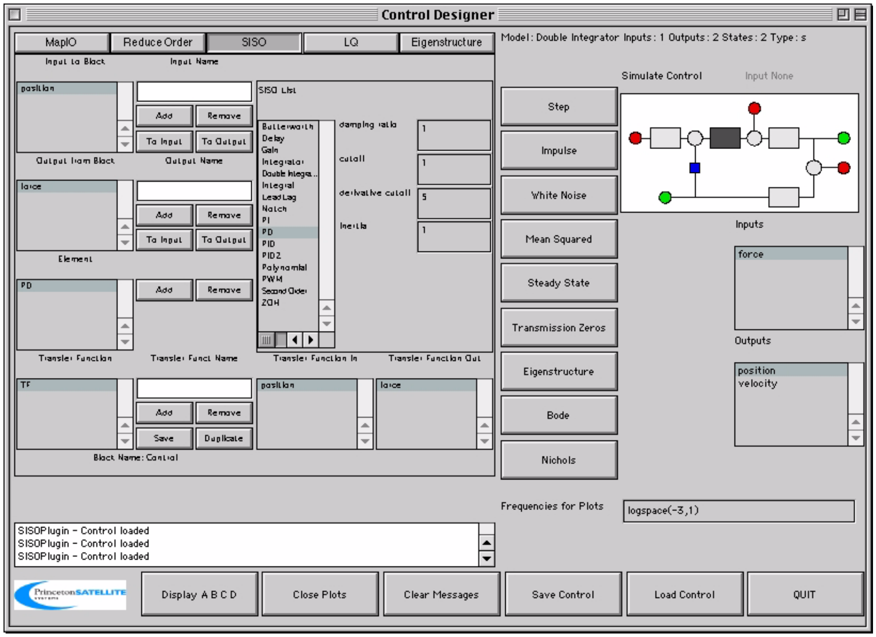
Figure 7-4 Step response from the GUI



7.3 Single-Input-Single-Output

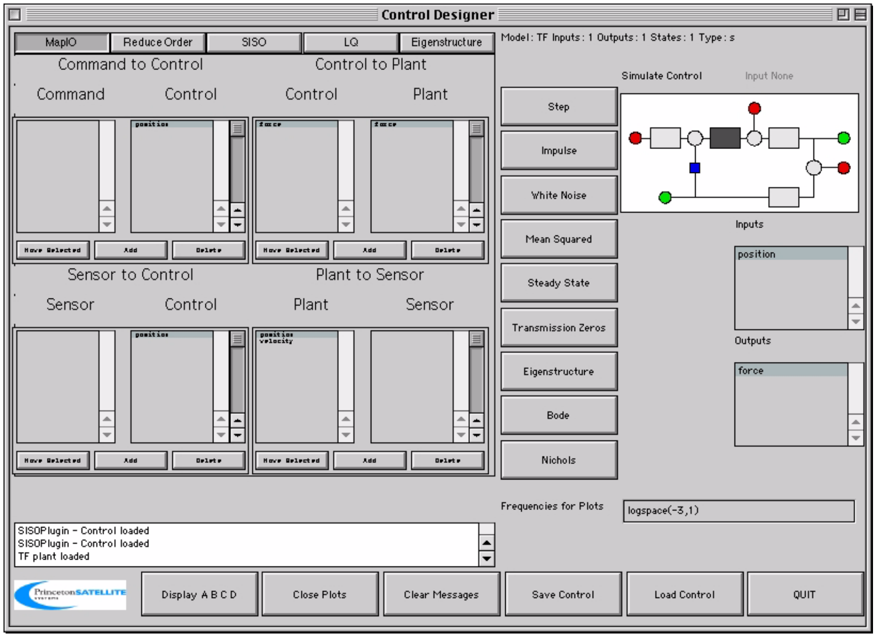
Close and reopen the GUI and load in the double integrator plant. Next select the control block and the SISO tab. Add the input position and output force. Then add a transfer function TF. Push the button to position the transfer function input and force the output. Now select TF and click PD in the SISOList. The GUI will look like the following.

Figure 7-5 SISO inputs



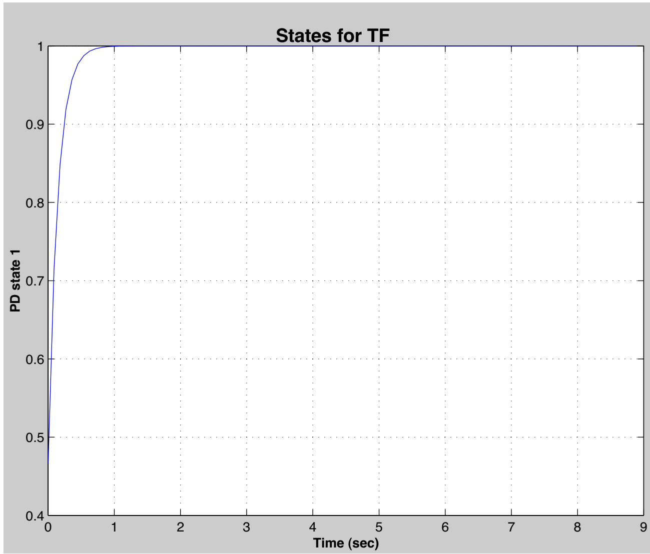
Hit the save button under the transfer function heading. Select the MapIO tab. You will see that the inputs and outputs of the plant and controller are aligned properly.

Figure 7-6 MapIO



Under plant to sensor click velocity and hit remove since it is not used by the SISO controller. When removed, velocity is prefixed by a star to indicate that it is part of the plant but unused. Click the control box to select the whole plant and hit step. You will see the following step response.

Figure 7-7 SISO step response



7.4 Eigenstructure Assignment

Run the script CCVDemo. This script generates the inputs for the eigenstructure assignment exam-

Listing 7-2 CCVDemo

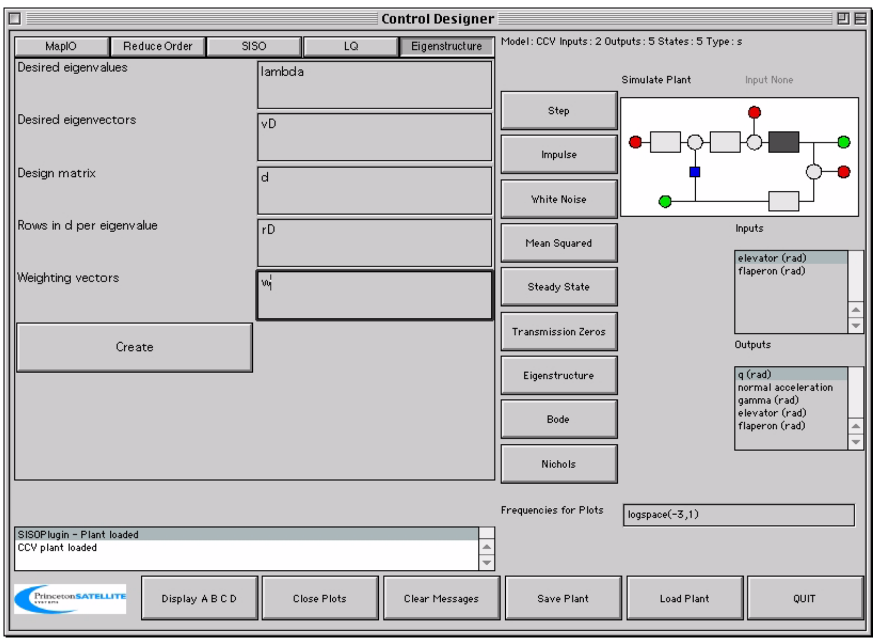
```
% Plant matrix
%-------------
g = CCVModel;

% Desired eigenvalues and eigenvectors
%------------------------------------
lambda = [ -5.6 + j*4.2; -5.6 - j*4.2; -1.0;...
           -19.0; -19.5];
vD = [ 1-j  1+j  0  1  1;...
      -1+j -1-j  1  0  0;...
       0    0    0  0  0];

% We really want to decouple gamma
%---------------------------------
w  = [ 1    1    1  1  1;...
       1    1    1  1  1;...
       100  100  1  1  1];

% The design matrix.
%----------------------------------------
d  = [eye(3),zeros(3,2);... % Desired structure for eigenvector 1
      eye(3),zeros(3,2);... % Desired structure for eigenvector 2
      0 1 0 0 0;...         % Desired structure for eigenvector 3
      0 0 1 0 0;...         %
      0 0 0 1 0;...         % Desired structure for eigenvector 4
      0 0 0 0 1];           % Desired structure for eigenvector 5

% Rows in d per eigenvalue
% Each column is for one eigenvalue
% i.e. column one means that the first three rows of
% d relate to eigenvalue 1
%---------------------------------------------------
rD = [3,3,2,1,1];

% Compute the gain and the achieved eigenvectors
%-----------------------------------------------
[k, v] = EVAssgnC( g, lambda, vD, d, rD, w );
```

ple. The model is already stored in `CCVModel.mat`.

lambda gives the desired eigenvalues, something that would be specified for simple pole placement. `vD` are the desired eigenvectors which we can assign because we are using multi-input-multi-output control. The weighting matrix shows how important each element of the desired eigenvector is to the control design. Notice that the length of each eigenvector in `vD` is not the length of the state. This is because we don't care about most of the eigenvector values. The matrix d is used to relate the desired eigenvector matrix to the actual states. `rD` indexes the rows in `d` to the eigenvalues. Each row relates vD to the plant matrix. For example, rows 7 and 8 relate column 3 in vD to the plant. In this case, vD(1,3) relates to state 2 and vD(2,4) relates to state 3.

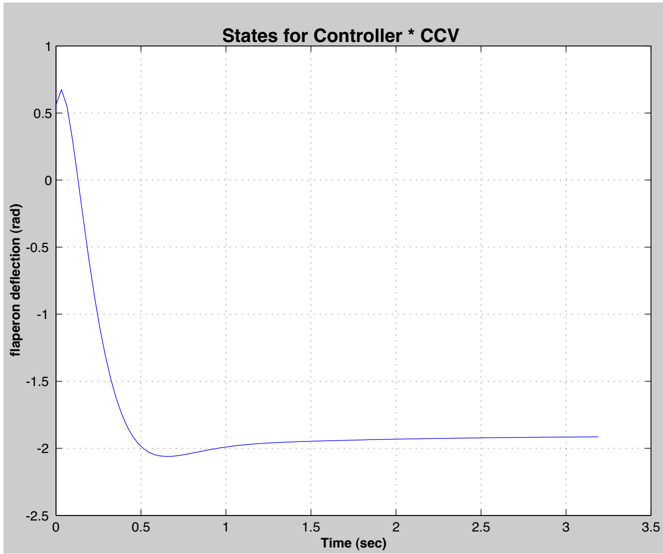
Now open ControlDesignPlugin. Click on the plan box and load CCVModel.mat. Now click on the Eigenstructure tab and enter lambda, vD, d, rD and w into the corresponding spots. The GUI will look as follows.

Figure 7-8 Eigenstructure design GUI



Push Create. Next push Step. You will see the following plot.

Figure 7-9 Step response with eigenstructure assignment



CHAPTER 8

IMPLEMENTING CONTROLLERS

CHAPTER 8

Implementing Controllers

This chapter shows how to implement controllers in the nonlinear simulation.

8.1 A General Interface

The function `AircraftControl.m` provides a general interface that can be used to structure your control system. The following listing shows the entry point for `AircraftControl.m`

Listing 8-1 AircraftControl.m

```
y = AircraftControl( action, d )

persistent s

switch action
  case 'initialize'
    s = Initialize( d );

  case 'update'
    [y, s] = Update( s, d );
end
```

`s` is used for global memory. Notice that s is always returned from the internal functions. `d` is passed to the function to initialize it. `y` is the output of the controller and `s` is the updated memory.

This version of AircraftControl just sends commands open loop to the aircraft. The initialization function is shown below.

Listing 8-2 Initialization

```
function s = Initialize( d )

s.actuatorName = d.actuatorName;
s.control      = d.control;

switch d.actuatorName
  case 'elevator'
    s.cDS.dT        = 0.5;
    s.cDS.magnitude = 2;
    s.cDS.init      = d.control.elevator;

  case 'throttle'
    s.cDS.dT        = 3;
    s.cDS.magnitude = 0.1;
    s.cDS.init      = d.control.throttle;

  case 'aileron'
    s.cDS.dT        = 2;
    s.cDS.magnitude = 5;
    s.cDS.init        = d.control.aileron;

  case 'rudder'
    s.cDS.dT        = 0.5;
    s.cDS.magnitude = 2;
    s.cDS.init      = d.control.rudder;

  otherwise
    error([d.actuatorName 'is not available'])
end
```

The name of the actuator to be used is being passed to this routine. Details for the actuation of the actuator are given in each case statement.

The update function is called each time step and is shown below.

Listing 8-3 Update

```
function [y, s] = Update( s, d )

% This is just to test the actuators
%-----------------------------------
switch s.actuatorName
  case 'elevator'
    s.control.elevator  = CInputs( d.t, 1, s.cDS, 'doublet' );
  case 'throttle'
    s.control.throttle  = CInputs( d.t, 1, s.cDS, 'doublet' );
  case 'aileron'
    s.control.aileron   = CInputs( d.t, 1, s.cDS, 'doublet' );
  case 'rudder'
    s.control.rudder    = CInputs( d.t, 1, s.cDS, 'doublet' );
end

y = s.control;
```

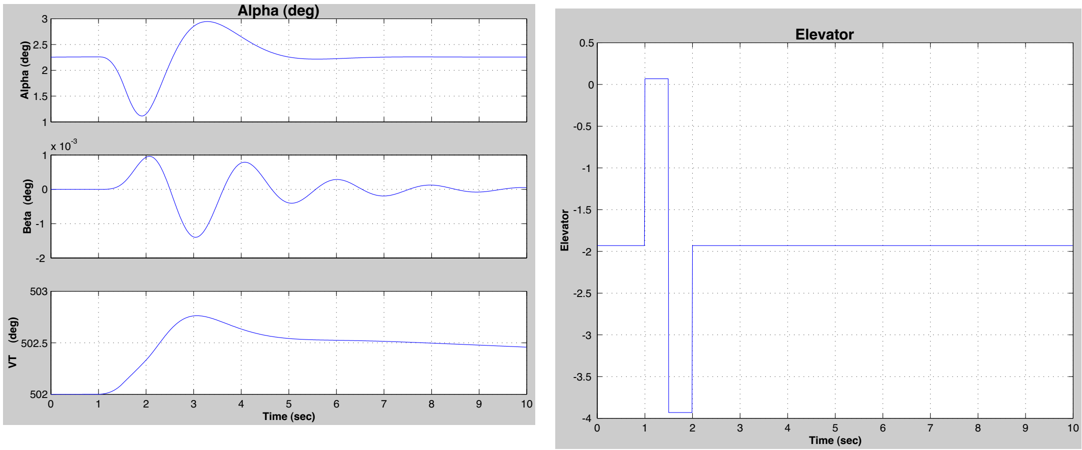
The data structure `s.cDS` is passed to `CInputs.m` which generates the control signature. The output is the datastructure `s.control`. This function is shown as implemented in the `ACControl.m` demo. The following listing shows relevant excerpts from that script.

Table 8-1 Excerpts from `ACControl`

```
% Control
%--------
d.control.throttle  =   0.1485;
d.control.elevator  =  -1.931;
d.control.aileron   =  -7e-8;
d.control.rudder    =   8.3e-7;

% Set up the control inputs
%--------------------------
AircraftControl( 'initialize', struct( 'actuatorName', actuatorName,
'control', d.control) )

for k = 1:nSim

  % Controls
  %---------
  d.control = AircraftControl( 'update', struct( 't', t, 'sensor', ACSensor(
x, d, 'meas' ) ) );
```

The control and response are shown in the following figure.

Figure 8-1 Control and aircraft response



8.2 Closed Loop Control

8.2.1 Introduction

`AircraftControl.m` can be easily modified to do closed loop control. This example is based on [Ref. 9-1] Example 4.5-1, a pitch rate control augmentation system. Note that in the reference the authors implement the pitch augmentation system as an analog system.

There are four parts to this problem

- Sensor input
- Actuator Model
- Control law
- Pilot input
- Control implementation

In this case we are using the elevator as the actuator. Our inputs are the pitch rate and angle of attack.

Our new control function is called AircraftControlCAS.m. The demo is `F16CAS.m`. The control design script is CASDesign.m.

8.2.2 Sensor Input

The sensors are available from the function `ACSensor.m`. You will use sensor outputs 5, `alpha` or angle-of-attack, and 3, `q` or pitch rate. This sensor model does not include any dynamics.

8.2.3 Actuator Model

The new actuator model is in F16Actuator.m shown in the following listing.

Listing 8-4 F16Actuator.m

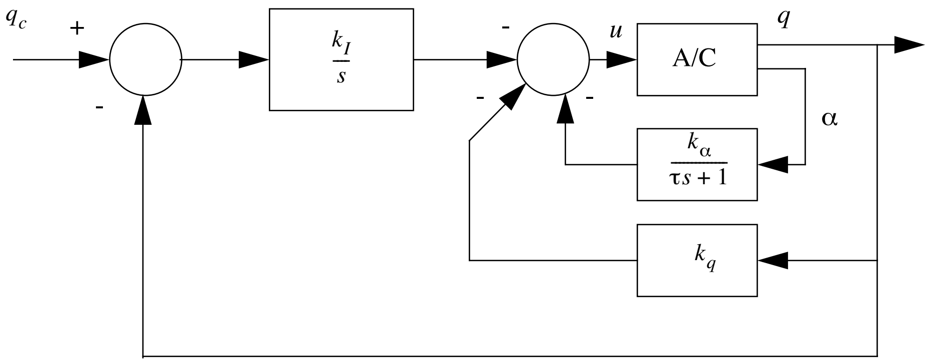
```
[dX, control] = F16Actuator( x, control, d )

dX = [...
      (control.throttle - x(1))/d.throttleLag;...
      (control.elevator - x(2))/d.elevatorLag;...
      (control.aileron  - x(3))/d.aileronLag;...
      (control.rudder   - x(4))/d.rudderLag];

control.throttle = x(1);
control.elevator = x(2);
control.aileron  = x(3);
control.rudder   = x(4);
```

Each actuator is modeled as a simple lag. `dX` is the derivative vector and the control output is now the state `x` which is the filtered control input.

8.2.4 Control Law

The controller, consisting of an integrator outer loop and two proportional inner loops is shown in the following block diagram. Notice that the error between the command and measured pitch rate is integrated while the pitch rate, and not the pitch rate error, is fed back through a proportional loop.

Figure 8-2 Pitch Axis Control Augmentation System



The measured pitch rate is subtracted from the commanded pitch rate and integrated in the outer loop. The inner loop consists of two loops, an alpha and a pitch rate loop. The control law is

$$u = -\left(\frac{k_I}{s}(q_C - q) + k_q q + \frac{k_\alpha}{\tau s + 1}\alpha\right) \quad [8\text{-}1]$$

This controller is demonstrated in the script `CASDesign.m`. The F-16 model is augmented with elevator dynamics represented by a simple lag.

When designing you need to

- set up the model
- set the initial state
- set the initial settings of the actuators
- linearize the model
- do your control design
- simulate

The script CASDesign.m does these things. The control design part is limited to using the gains from the reference. The script does a state-space simulation of the controller and the dynamics as a final check on the response.

The first three steps are the same in the design scripts and the simulation scripts. The simulation scripts also usually linearize the model to extract the aircraft modes.

Setting up the model is shown in the following listing.

Listing 8-5 Setting up the F16 model

```
% F16 database
%-------------
d               = ACBuild('F16');
d.theta0        = 0;
d.wPlanet       = [0;0;0];
d.actuator.name = 'F16Actuator';
d.aero.name     = 'ACAero';
d.engine.name   = 'ACEngine';
d.rotor.name    = [];
d.sensor.name   = 'ACSensor';
d.disturb.name  = [];

% Load the standard atmosphere
%----------------------------
d.atmData       = load('AtmData');
d.atmUnits      = 'eng';

% Actuator dynamics
%------------------
d.actuator.throttleLag = 4.9505e-02;
d.actuator.elevatorLag = 4.9505e-02;
d.actuator.aileronLag  = 4.9505e-02;
d.actuator.rudderLag   = 4.9505e-02;
```

The data structure entries with the `.name` fields are the names of the plugin functions, such as the `F16Actuator` described above. If there is no plugin you enter `[]`.

The initial state is loaded as shown in the following listing.

Listing 8-6 Setting the initial state

```
% Control settings
%-----------------
d.control.throttle  =   0.1385;
d.control.elevator  =  -0.7588;
d.control.aileron   =  -1.2e-7;
d.control.rudder    =   6.2e-7;

% Initial state vector Corresponding to Nominal in
% Table 3.4-3 p. 139 of the reference
%-------------------------------------------------
altitude  = 100;
alpha     = 0.03691;
beta      = -4.0e-9;
theta     = 0.03991;
vT        = 502;
v         = VTToVB( vT, alpha, beta );

cG        = [0.35;0;0];

r         = [2.092565616797901e+07+altitude;0;0];

eulInit   = [0;theta;0.00];

q         = QECI( r, eulInit );
w         = [0;0;0];

wR        = 160;
engine    = ACEngEq( d, v, r ); % Engine state
mass      = 1/1.57e-3;
inertia   = [9497;55814;63100;0;-982;0];
actuator  = [0;0;0;0];
sensor    = [];
flex      = [];
disturb   = [];

% Initial time and state
%-----------------------
x         = acstate( r, q, w, v, wR, mass, inertia, cG, engine, actuator,
sensor, flex, disturb );
```

We only want to work with the longitudinal dynamics for q and alpha. Extracting those state space matrices is shown in the following listing.

Listing 8-7 Extracting the plant model for the design

```
% Generate the state space model
%------------------------------
stateName.actuator = {'Throttle Lag', 'Elevator Lag', 'Aileron Lag', 'Rudder Lag'};
d                  = ACInit( x, d, stateName );
g                  = AC( x, 0, 0, d, 'linalpha');
aC                 = get( g, 'a' );
cC                 = get( g, 'c' );
bC                 = get( g, 'b' );

kLon         = [10 11 5 8 26];
kLonAQ       = [11 8 26];
kAlphaSensor = 5;
kQSensor     = 3;
kElevator    = 2;

disp('The state space matrices for just alpha and q')
a    = aC(kLonAQ,kLonAQ);
b    = bC(kLonAQ,kElevator);
c    = cC(kAlphaSensor,kLonAQ); % alpha only
```

The script doesn't actually do the design, it just uses the gains in the reference and checks eigenvalues.

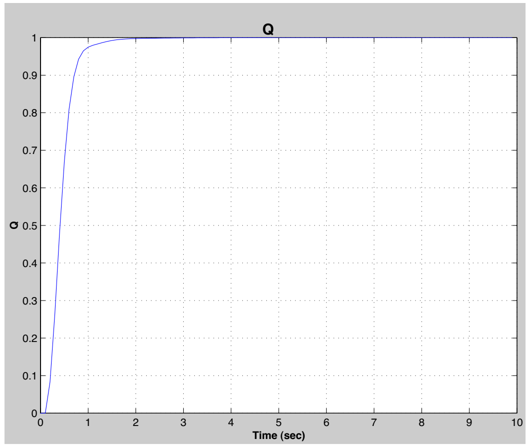
The state space simulation code is shown below.

Listing 8-8 State space simulation

```
dT           = 0.1; % 10 Hz controller works well

[a, b]       = C2DZOH( a, b,       dT );
[aCAS, bCAS] = C2DZOH( aCAS, bCAS, dT );

nSim         = 100;

xPlot        = zeros(1,nSim);

qC           = 1.0;
xCAS         = [0;0];
x            = [0;0;0];
y            = [0;0];

for k = 1:nSim

  xPlot(k) = y(2);

  y    = c*x;
  xCAS = aCAS*xCAS + bCAS*[y(1);y(2) - qC];
  yCAS = -(cCAS*xCAS + dCAS*y);
  x    = a*x + b*yCAS;

end

t = (0:(nSim-1))*dT;

Plot2D( t, xPlot, 'Time (sec)', 'Q' );
```

Note that in the reference the simulations are done with analog control. The resulting step response is shown in the following figure. The controller and the plant are propagated separately. This makes it much easier to go from the linear simulation to the nonlinear simulation.

Figure 8-3 Step response



8.3 Pilot Input

Pilot input can be done in two ways. One is just to pass the desired input into your control function. The second is to customize the HUD. In this example, we need a pitch rate input which is not an available output on the standard HUD. We would like the pilot to be able to select a pitch rate and then command the aircraft.

The pilot input is read in using the following code.

Listing 8-9 Pilot pitch rate input

```
% Pitch rate input
%------------------
pilotPitchRateInput = struct( 'enter', hUDOutput.pushbutton1, 'value',
hHUD.control.text1 );

% Controls
%---------
d.control = AircraftControlCAS( 'update', struct( 't', t, 'sensor', ACSensor(
x, d, 'meas' ), 'pilotPitchRateInput', pilotPitchRateInput ) );
```

8.4 Control Implementation

The controller is implemented int AircraftControlCAS. As with the previous example there are two parts, the initialization and the update.

The initialization is shown in the following listing.

Listing 8-10 Initialization

```
function s = Initialize( d )

kI       =   1.5;
kQ       =  -0.5;
kAlpha   =  -0.08; % Notice this sign!
tauAlpha =   0.1;
s.aCAS   =   [-1/tauAlpha 0;0 0];
s.bCAS   =   [1/tauAlpha 0;0 -1];
s.cCAS   =   [kAlpha kI];
s.dCAS   =   [0 kQ];
s.xCAS   =   [0;0];

[s.aCAS, s.bCAS] = C2DZOH( s.aCAS, s.bCAS, d.dT );
s.control        = d.control; % Nominal settings
s.pilotPitchRateInput = 0;
```

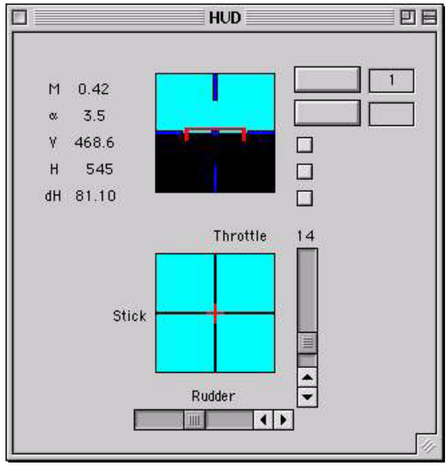
The update is shown below.

Listing 8-11 Update

```
function [y, s] = Update( s, d )

% Pilot input
%------------
if( d.pilotPitchRateInput.enter )
  s.pilotPitchRateInput = d.pilotPitchRateInput.value;
  disp(sprintf('New pitch rate input %12.4f', s.pilotPitchRateInput))
end

% Input
%------
input = [d.sensor.alpha; d.sensor.q];

% Control implementation
%-----------------------
yCAS   = -(s.cCAS*s.xCAS + s.dCAS*input);
s.xCAS = s.aCAS*s.xCAS + s.bCAS*[input(1);input(2) - s.pilotPitchRateInput];

% Output
%-------
s.control.elevator = yCAS;
y                  = s.control;
```

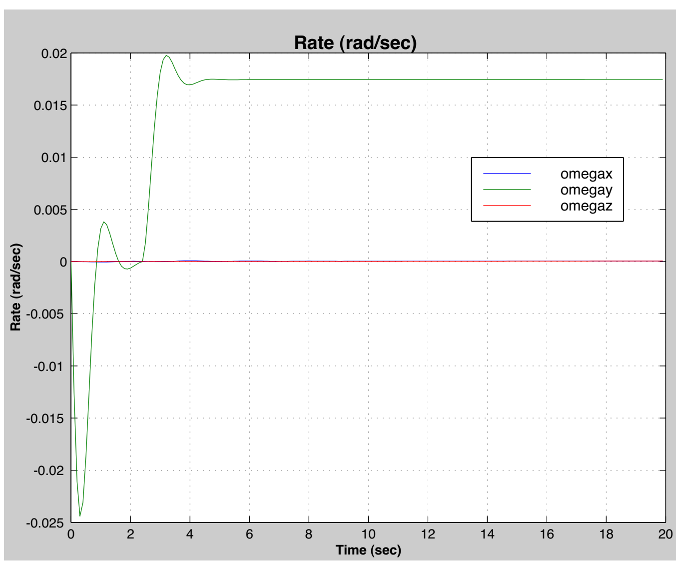
The results are shown in the following plot. A 1 deg/sec pitch rate is commanded using the first button on the HUD. You may need to push the button a couple of times. The line in `disp` above prints into the command window to let you know that the command went through. The HUD looks like the following figure.

Figure 8-4 HUD after the pitch rate has been entered



The rate response is shown below.

Figure 8-5 Rate response to command



CHAPTER 9

REFERENCES

CHAPTER 9

References

This chapter describes the references used in designing this toolbox.

9.1 About the References

References 1-4 are essential references for anyone designing aircraft control systems.

[Ref. 9-1] covers most of the material in this toolbox and explains in detail how to use all of the control and simulation tools. It is an easily accessible text and is very well written. It covers all forms of control design techniques that are applicable to aircraft. It is the ideal companion volume for this toolbox.

[Ref. 9-2] covers the modeling of aircraft in great detail. If you are interested in building your own simulation models, and creating your own properties databases, then this book is an excellent source of information.

[Ref. 9-3] is a classic book with interesting approaches to SISO and MIMO control. It also has a great deal of information on aircraft modeling.

[Ref. 9-4] covers the application of linear quadratic regulator techniques to both aircraft and spacecraft. It is very well written and clearly explains all of the fundamental principles of aerospace control design.

9.2 Reference Books

[9-1] Stevens, B. L. and F. L. Lewis (1992). *Aircraft Control and Simulation*, John Wiley & Sons, New York.

[9-2] Ashley, H. (1974). *Engineering Analysis of Flight Vehicles*, Dover Publications, Inc., New York.

[9-3] McRuer, D., Ashkenas, I., and D. Graham (1971). *Aircraft Dynamics and Automatic Control*, Princeton University Press.

[9-4] Bryson, A. E., Jr. (1994). *Control of Spacecraft and Aircraft*, Princeton University Press, Princeton, New Jersey.

[9-5] Maciejowski, J.M. (1989). *Multivariable Feedback Design*. Addison-Wesley, Reading, MA.

[9-6] Zhou, K., (1998). *Essentials of Robust Control*. Prentice-Hall, New Jersey.

[9-7] Dutton, K., S. Thompson, and B. Barraclough. (1997). *The Art of Control Engineering*. Addison-Wesley, Reading, MA.

[9-8] Abzug, M. J., and E. E. Larrabee. (1997). *Airplane Stability and Control*. Cambridge University Press.

9.3 Papers

[9-9] Andry, A. N., Jr., Shapiro, E.Y. and J.C. Chung, "Eigenstructure Assignment for Linear Systems," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. AES-19, No. 5. September 1983.

[9-10] Hung, Y. S., and MacFarlane A.G.J. (1982). *Multivariable Feedback: A Quasi-classical Approach*. Lecture Notes in Control and Information Sciences, Vol. 40. Berlin: Springer-Verlag.

[9-11] Stein, G. and Athans, M. (1987). The LQG/LTR Procedure for Multivariable Feedback Control Design. *IEEE Transactions on Automatic Control*, AC-32(2), 105-114.

[9-12] Anderson, B.D.O. and Mingori, D.L. (1985). Use of Frequency Dependence in Linear Quadratic Control Problems to Frequency-Shape Robustness. *J. Guidance and Control*, 8(3), 397-401.

[9-13] MacFarlane, A.G.J. and Postlethwaite, I. (1977). The generalized Nyquist stability criterion and multivariable root loci. *Int. J. Control*, 25(1), 81-127.

[9-14] Edmunds, J.M. (1979). Controls system design and analysis using closed-loop Nyquist and Bode arrays. *Int. J. Control*, 30(5), 773-802.

[9-15] Doyle, J.C. and Stein, G. (1981). Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis. *IEEE Transactions on Automatic Control*, AC-26(1), 4-16.

[9-16] Dorato, P. (1987). A Historical Review of Robust Control. *IEEE Control Systems Magazine*, 7(2),44-47.

[9-17] MacFarlane, D.C. and Glover, K. (1989). *Robust Control Design Using Normalized Coprime Factor Plant Descriptions*. Springer-Verlag, Berlin.

[9-18] Doyle, J.C. and G.J. Balas (1990). Identification of Flexible Structures for Robust Control. *IEEE Control Systems Magazine*, 10(4),51-58.

References

[9-19] Fan, M.K.H and Tits A.L. (1988). m-form numerical range and the computation of the structured singular value. *IEEE Transactions on Automatic Control*, AC-33, 284-289.

[9-20] Safonov, M. and Doyle J.C. (1984). Minimizing conservativeness of robustness singular values. *Multivariable Control: New Concepts and Tools* (Tzafestas S.G., ed.), Dordrecht: Reidel, 197-207.

[9-21] Doyle, J.C. (1978). Guaranteed margins for LQG regulators. *IEEE Transactions on Automatic Control*, AC-23, 756-757.

[9-22] Horowitz, I. and Sidi, M. (1980). Practical design of feedback systems with uncertain multivariable plants. *Int. J. Systems Sci.,* 11(7), 851-875.

[9-23] Horowitz, I. (1979). Quantitative synthesis of uncertain multiple input-output feedback system. *Int. J. Control*, 30(1), 81-106.

[9-24] Park, M.S., Chait, Y. and Steinbuch, M. (1994). A New Approach to Multivariable Quantitative Feedback Theory: Theoretical and Experimental Results. *ASME J. DSMC*.

[9-25] Hamel, P.G. (1994). Aerospace vehicle modeling requirements for high bandwidth flight control. *Aerospace Vehicle Dynamics and Control*, Oxford University Press, Oxford, 1-32.

[9-26] Hyde, R.A. and Glover, K. (1994). Flight controller design using multivariable loop shaping. *Aerospace Vehicle Dynamics and Control*, Oxford University Press, Oxford, 81-102.

[9-27] Carr, S.A. and Grimble, M.J. (1994). Comparison of LQG, H_∞ and classical designs for the pitch rate control of an unstable military aircraft. *Aerospace Vehicle Dynamics and Control*, Oxford University Press, Oxford, 103-124.

[9-28] Gribble, J.J., et al. (1994). Helicopter flight control design: multivariable methods and design issues. *Aerospace Vehicle Dynamics and Control*, Oxford University Press, Oxford, 199-224.

Index

Symbols

@acstate, 43

A

AC, 39
ACBuild, 39
ACControl.m, 75
ACEngEq, 43
ACInit, 39
ACModes, 45
ACPlot, 39, 46, 57
ACSensor.m, 77
ACTrim, 39
AircraftControl.m, 72, 76
AircraftControlCAS.m, 77
angle-of-attack, 77
artificial damping, 41

C

C, 17
C2DelZoh, 40
C2DFOH, 41
C2DZoh, 40
CASDesign.m, 77, 78
`CASDesign.m`, 22
`CCVDemo`, 68
`CCVModel.mat`, 69
cell array, 33
CInputs.m, 75
class, 34
    class, 35
    constructor, 35
    instance, 35
    method, 35
    object, 35
    overloading, 35
    polymorphism, 35
ControlDesignPlugin, 60, 62
CTSim, 39

D

data structure, 32
database, 16
DC8, 16
DemoSC, 21
double integrator, 64
double integrator', 60
`DoubleIntegrator.mat`, 61
DrawAC, 54

E

Eigenstructure assignment, 60
eigenvalues, 69
eigenvector, 69

F

F-16, 54, 78
F16, 16, 17
F16Actuator.m, 77
F16CAS.m, 77
FileHelp, 24
first order hold, 41
Fly, 42, 55
full-state feedback, 60

G

global variable, 56
GUI, 52

H

help, 28
help system, 20
HUD, 47
HUD, 52

I

IC, 38
integration time step, 41

J

Jacobian, 40

L

Linear Quadratic, 60
`LQC.m`, 61
`LQFullState.m`, 61

M

Matlab Command Window, 26
MRS, 38

N

ND2SS, 40
nonlinear simulation, 41

O

online help, 28

P

pitch rate, 77
progress bar, 28

Q

QECI, 43

R

RK2, 41
RK4, 41
RK45, 41

S

script, 17
SISO, 64
spacecraft dynamics, 38
statespace, 38
StateSpacePlot, 58
Step, 38

T

Technical Support, 29
TimeGUI, 48, 53, 56

V

VTToVB, 43

Z

zero order hold，40