B

Flight Simulation

B.1 Introduction

This is the documentation for the flight simulation code available at the companion web site to this book, at www.wiley.com/go/durham/aircraft_control_allocation. The original code is named ADMIRE (Aero-Data Model In a Research Environment). It was created by the Swedish Defence Research Agency (Svensk Försvarets forskningsinstitut, FOI). At the time of this writing version 4.1 of ADMIRE was available at http://www.foi.se/en/Our-Knowledge/Aeronautics/Admire/Downloads/.

Several modifications to the original code have been made, primarily at the flight control system level. Mostly this amounted to the incorporation of a nonlinear dynamic inversion flight control law, and the implementation of various control allocation algorithms referred to in the text.

The high-level source of the modified code in MATLAB® at the companion web site is a Simulink® model named admire_sim_NDI.mdl. This version has been tested with the Student Version of MATLAB® R2014a running on a Macintosh computer.

B.2 Modifications

B.2.1 Three of the top-level blocks have been left almost completely unaltered:

B.2.1.1 Total Computer Delay/Transport Delay Version

This block models time delay within the digital computer that implements the flight control algorithms. In the ADMIRE simulation, this delay is 20 ms, which represents two frames of delay for a computer running at 100 Hz.

B.2.1.2 Saturators, Rate Limiters, and Actuators

This block models the actuator dynamics and incorporates rate and position limiting. The throttle steady state `tss` is not rate limited. Also, the landing gear command, thrust vectoring, and atmospheric disturbances do not go through the delay or actuator blocks.

Aircraft Control Allocation, First Edition. Wayne Durham, Kenneth A. Bordignon and Roger Beck.

Companion website: www.wiley.com/go/durham/aircraft_control_allocation

B.2.1.3 Aircraft Response

This block models the response of the vehicle to the inputs. The `Uncertain Parameters` block, which contained 25 constant blocks with variables that are currently set to zero, has been replaced with a single constant block of zeros. This will affect functionality only if users wish to investigate the effects of altering the uncertain parameters. Variable names have been added on the input ports.

B.2.2 Minor modifications consist of the new `Pilot` and `Sensors` blocks.

B.2.2.1 Pilot

The `Pilot` block is set up to handle simple time histories of pilot inputs that will be biased about the values need for trim. This is similar to the original ADMIRE implementation.

B.2.2.2 Sensors

A `Sensors` block was added at the bottom of the diagram in the feedback path to modify the sensed heading angle. The `mod` (modulus) command was used to keep the heading in the range of principal values ($0 - 2\pi$ rad) for realism in the waypoint-following part of the control law.

B.3 NDI_CLAW

The major modification consists of a block containing a nonlinear dynamic inversion control law (NDI_CLAW) that completely replaces the original ADMIRE control law. For the naming of new variables, we have tried to follow a convention of `Name_units`. To limit names to containing only alpha numeric characters and underscores, we use abbreviations such as *mps* for meters per second.

The primary components of NDI_CLAW are the following blocks:

B.3.1 NDI_CLAW/Rate Transition

There is a rate transition block to take the sensor data from the continuous simulation and make it usable by the control law that is modeled as a discrete system. This is done to emulate a digital computer used for the flight control system. A rate of 100 Hz ($dt = 0.01$ s) was chosen because it is a reasonable rate commonly used in modern flight control computers.

B.3.2 NDI_CLAW/PILOT_Mod

This block converts the pilot inputs into usable commands. Currently, this is done with a simple constant scale factor (gain block). For a real aircraft, there likely would be some gradients and command shaping to improve the handling qualities.

B.3.3 NDI_CLAW/INPUT

This block takes the data from the `PILOT` and `SENSORS` blocks and puts all the incoming information on a single data bus. This is done to help clean up the diagrams. The block clearly identifies data that comes from sources that are external to the flight computer. The block attempts to emulate what an actual computer would do processing the input data. Several scopes have been added to display data the control law is using. Users may wish to add other scopes, or remove these to make the simulation run faster.

B.3.4 NDI_CLAW/MissionManager

This block generates commands for the autopilot modes of the control law. Note that these commands may or may not be used, depending on the values of other variables in the other parts of the control law. The default configuration will fly a square pattern at different altitudes and velocities over 1000 s of simulation time. This was done to verify that the control law could control the aircraft for the entire envelope of valid data used in the ADMIRE simulation.

Some other constant blocks have been left in the mission manager. These can be used, for example, to stay at the trim condition (`x0bare`) or go to the first altitude and velocity in the tables. To use them, replace the blocks `ALT_table` and `V_table`. To change the altitude and velocity commands, edit the following variables in the initialization file `INIT_NDI.m`:

Altcmd_t Vector of time (s)
Altcmd_v Vector of altitude commands (m)
Vcmd_t Vector of time (s)
Mcmd_v Vector of Mach number
a Vector of the speed of sound at the commanded altitude (m/s) ($Vcmd_v = Mcmd_v . * a$).

The bank-angle command will do left and right turns of increasing amplitude. This is currently not being used, because the roll response is defaulted to generate bank-angle commands based on waypoints. To change the values of the commanded bank angle, edit the following variables in the initialization file `INIT_NDI.m`:

PHIcmd_t Vector of time (s)
PHIcmd_v Vector of bank angle commands (°).

B.3.4.1 NDI_CLAW/MissionManager/WaypointNavigation

This block is a simple attempt at an autopilot feature that works with one of the roll axis options to follow waypoints (X and Y earth axes). The output of this block is: the current X and Y command values (`Xcmd` and `Ycmd`), the previous commanded values (`Xcmdm1` and `Ycmdm1` abbreviations of `Xcmd` minus one and `Ycmd` minus one), and a maximum bank angle to be used, which is based on the g available at the current flight condition.

There is an attempt to start a turn prior to the waypoint, so that a constant-radius turn will end with the aircraft on the new heading.

Some of the known problems with this navigation system are: insufficient error checking, and incorrect following of waypoints placed too close or requiring very tight turns. The default path is a simple square pattern. The waypoints may be changed by editing the following variables in the initialization file `INIT_NDI.m`

Xwpt X distance (m), north, initially $\begin{bmatrix}0 & 4000 & 4000 & 0 & 0\end{bmatrix} * 8$
Ywpt Y distance (m), east, initially $\begin{bmatrix}0 & 0 & 4000 & 4000 & 0\end{bmatrix} * 8$

The factor of eight may be adjusted if the closeness of the waypoints results in turns that exceed the capabilities of the airplane.

B.3.5 NDI_CLAW/DynamicInversionControl

This is the top level of the dynamic inversion control law. It is broken into six main subsystems:

B.3.5.1 NDI_CLAW/DynamicInversionControl/Auto_Throttle

The throttle is not commanded by the control allocation. Instead, it was implemented as an energy-based control scheme. It uses commanded altitude and velocity to make an energy command, `Ecmd`. The sensed altitude and velocity are used to calculate the current energy state `E`. A proportional-integral control law is used to generate throttle commands to drive the energy state to the desired values. The integration is turned off if the error is very large, to prevent the system from overshooting due to integrator build-up. There is a table look-up to reduce the throttle gain when the afterburner is on ($\texttt{tss} > 0.8$) because the afterburner is much more effective than the non-afterburner throttle.

There is also a block `Get_UseSplit`. When `UseSplit` is true, the preferred values change to bias the inboard elevons up and the outboard elevons down. This effectively makes them act as a speed-brake to help decelerate the vehicle.

B.3.5.2 NDI_CLAW/DynamicInversionControl/CMD

The `CMD` (commands) block converts the pilot and mission manager data into commands for the regulator blocks. At the time of this writing, this block is mostly a pass-through that does not have great effect. The altitude command is rate-limited to avoid over-large commands to the auto-throttle. There are unconnected doublet and time history blocks that may be included to observe the aircraft responses to simple commands. To use them, disconnect the input to a gain-of-1 block, and connect the desired input.

B.3.5.3 NDI_CLAW/DynamicInversionControl/REG

These are the regulator blocks that generate the desired accelerations. They are separated into roll, pitch and yaw.

NDI_CLAW/DynamicInversionControl/REG/RollReg
There are three different ways to generate the desired roll acceleration, which are selectable by changing the variable `Rmode` in the initialization file `INIT_NDI.m`. Note that by the logic of the `If` blocks, only the code in the `true` condition runs.

Rmode = 1: Roll-rate Command: This block uses simple proportional control of roll rate. This mode is typical of a pilot flying and using the lateral stick to generate roll-rate commands. There is a command gain that is a function of Mach number. This functionality is typical of high-speed fighter aircraft. Although it is not currently connected, it was included so that users may experiment with different values.

Rmode = 2: Bank-angle Command: This mode emulates an autopilot flying bank-angle commands. There is proportional control of roll rate (similar to `MODE=1`), with an outer loop that has proportional-integral-forward path control of bank angle.

Rmode = 3: Cross-track Command: This mode is used when using the waypoints from the mission manager. It uses proportional control of roll rate and bank angle. The bank-angle command comes from two different blocks. Initially, the bank angle command is the maximum bank angle (with appropriate sign) until the aircraft is on the correct heading. Once the proper heading is achieved, cross-track error and cross-track error rate are used to calculate the required bank angle. This block may create undesired results if the waypoints are not well spaced with an easy-to-follow path. In that case, the airplane may fly in circles or deviate into an undesired location.

NDI_CLAW/DynamicInversionControl/REG/PitchReg
There are three different ways to generate the desired pitch acceleration, which are selectable by changing the variable `Pmode` in the initialization file `INIT_NDI.m`. Note that the way the `If` blocks are implemented, only the code in the true condition runs.

Pmode = 1: Pitch-rate Command: This mode is simple proportional control of pitch rate. This is similar to the roll-rate command system. For low-speed flight, this mode may be preferable to pilots. At higher speeds, pilots would more likely prefer *g*-command for the pitch control inceptor. The current model does not have a *g*-command system.

Pmode = 2: Flight-path Angle Command: This mode performs proportional control on pitch rate and proportional-integral control on flight path angle. Some UAVs have the remote pilot/operator pitch stick command flight path angle.

Pmode = 3: Altitude Command: This mode is used to control altitude with pitch. There is proportional control of pitch rate and flight path angle. Flight path angle command is generated from proportional and integral control on altitude.

NDI_CLAW/DynamicInversionControl/REG/YawReg
This block generates the yaw acceleration command based on a desired sideslip angle. The controller is proportional-integral-forward path on sideslip angle, and proportional on yaw

rate. The `n1` (-1) block converts a sideslip rate command into a yaw rate command, $r \approx -\dot{\beta}$. Without yaw thrust vectoring the vehicle is yaw-control limited. Different allocation methods seem to be able to achieve roughly the same maximum sideslip angle, but if one uses more of the AMS the goal can be achieved faster.

B.3.5.4 NDI_CLAW/DynamicInversionControl/AERO_OBM

This block calculates the nominal accelerations, the control effectiveness matrix, and the limits on the aerodynamic effectors. The `AeroLimits` block finds the limits on the aerodynamic effectors based on Mach number per the ADMIRE documentation. There are terms for all 16 inputs to the `ADMIRE_main` program that can act as control effectors with the control law. The 16 input values are:

drc Right canard
dlc Left canard
droe Right outboard elevon
drie Right inboard elevon
dlie Left inboard elevon
dloe Left outboard elevon
dr Rudder
dle Leading edge flaps
ldg Landing gear
tss Thrust command
dty Yaw thrust vectoring
dtz Pitch thrust vectoring
u_dist Forward velocity disturbance
v_dist Side velocity disturbance
w_dist Vertical velocity disturbance
p_dist Roll rate disturbance

Both linear and nonlinear versions of the OBM have been implemented.

Linear model version
The model was linearized in a flight condition with nominal trim effectors $\mathbf{u}_{nom}$:

$$\begin{aligned} \mathbf{u}_{cmd} &= \mathbf{u}_{nom} + B^{-1}(\dot{\mathbf{x}}_{des} - \dot{\mathbf{x}}_{nom}) \\ \dot{\mathbf{x}}_{nom} &= A\mathbf{x} + B\mathbf{u}_{nom} \end{aligned} \tag{B.1}$$

The A and B matrices come from the ADMIRE linearization tool.

Nonlinear model version
The nonlinear model replaces the linearized A and B matrices with table look-ups of the aerodynamic data.

$$\begin{aligned} \mathbf{u}_{cmd} &= \mathbf{u}_{nom} + B^{-1}(\dot{\mathbf{x}}_{des} - \dot{\mathbf{x}}_{nom}) \\ \dot{\mathbf{x}}_{nom} &= \mathbf{f}(\mathbf{x}, \mathbf{u}_{nom}) = \mathbf{f}(\mathbf{x}, \mathbf{u}) - \Delta\mathbf{f} \end{aligned} \tag{B.2}$$

The actual accelerations are:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \ \mathbf{u}) = \begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} \tag{B.3}$$

The accelerations were developed in Chapter 2, Eq. (2.12). ADMIRE documentation adopts the more compact notation of the first edition of Stevens and Lewis (1992):

$$\begin{aligned} \dot{p} &= (C_1 r + C_2 p)q + C_3 L + C_4 N \\ \dot{q} &= C_5 pr - C_6(p^2 - r^2) + C_7 M \\ \dot{r} &= (C_8 p - C_2 r)q + C_4 L + C_9 N \end{aligned} \tag{B.4}$$

The moments are total moments, aerodynamic and propulsive. The subscripted C terms are calculated in the initialization file `INIT_NDI.m` from

$$\begin{aligned} \Gamma &= I_{xx} I_{zz} - I_{xz}^2 \\ \Gamma C_1 &= (I_{yy} - I_{zz}) - I_{xz}^2 \\ \Gamma C_2 &= (I_{xx} - I_{yy} + I_{zz}) I_{xz} \\ \Gamma C_3 &= I_{zz} \\ \Gamma C_4 &= I_{xz} \\ C_5 &= (I_{zz} - I_{xx})/I_{yy} \\ C_6 &= I_{xz}/I_{yy} \\ C_7 &= 1/I_{yy} \\ \Gamma C_8 &= I_{xx}(I_{xx} - I_{yy}) + I_{xz}^2 \\ \Gamma C_9 &= I_{xx} \end{aligned} \tag{B.5}$$

The `Aircraft Response` block provides as outputs the rates and moment coefficients. These are used to calculate the body axis accelerations. These accelerations include the effects of the effectors being at their current deflections. If we calculate the control commands as they change from the current positions, then we can use the actual accelerations as the nominal accelerations:

$$\dot{\mathbf{x}}_{nom} = \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \ \mathbf{u}) \tag{B.6}$$

If we calculate the control commands as the change from some nominal position, which is not the current effector deflections, then we need to subtract off a component that is the difference between the total acceleration and the acceleration with the effectors at their nominal locations.

Consider the linear case:

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B\mathbf{u} = \mathbf{f}\,(\mathbf{x}, \ \mathbf{u}) \\ \dot{\mathbf{x}}_{nom} &= A\mathbf{x} + B\mathbf{u}_{nom} \\ \dot{\mathbf{x}}_{nom} &= A\mathbf{x} + B\mathbf{u} - (B\mathbf{u} - B\mathbf{u}_{nom}) = \mathbf{f}\,(\mathbf{x}, \ \mathbf{u}) - \Delta\mathbf{f} \end{aligned} \tag{B.7}$$

To estimate $\Delta\mathbf{f}$ for the nonlinear model, we run the ADMIRE nonlinear aerodynamic model with the controls at their current deflections and their nominal deflections and take the difference.

To calculate the B matrix for the nonlinear OBM, we call the aerodynamic model with one effector disturbed by first a small positive and then a small negative deflection from the nominal position and do a simple difference to get a slope. This can be time consuming to do it for all seven aero effectors every frame, so we only update one column of B each frame to speed up the simulation. The B matrix is initialized to the values for the trim condition.

We have implemented B matrix calculations for only the aerodynamic effectors (the first seven in the list above). For the other effectors, the B matrix is initialized at the trim values, but never updated. If we really want to experiment with the other effectors, we should stay near the trim condition until we implement a way to calculate their B-matrix terms as flight conditions change.

B.3.5.5 NDI_CLAW/DynamicInversionControl/Unom

There are various options for nominal effectors, $\mathbf{u}_{nom}$:

1. Set the nominal values to zero:

$$\mathbf{u}_{nom} = \mathbf{0} \tag{B.8}$$

2. Trim the aircraft and use the trimmed values:

$$\mathbf{u}_{nom} = \mathbf{u}_{trim} \tag{B.9}$$

3. Use the last computed command:

$$\mathbf{u}_{nom} = \mathbf{u}_{cmd} z^{-1} \tag{B.10}$$

4. Use the last computed command sent through an estimated effector transfer function:

$$\mathbf{u}_{nom} = \mathbf{u}_{cmd} z^{-1} G_{act}(z) \tag{B.11}$$

5. Use sensors to determine the current values of the effectors:

$$\mathbf{u}_{nom} = \mathbf{u}_{sensed} \tag{B.12}$$

All of these options are available except the last.

B.3.5.6 NDI_CLAW/DynamicInversionControl/ControlAllocation

At the top level, the control allocation is broken into three subsystems: `PreProcess`, `GetU` and `PostProcess`.

NDI_CLAW/DynamicInversionControl/ControlAllocation/PreProcess
This block sets the inputs for the control allocation routines. `GetLimits` will find the current allowable change in effectors based on the nominal effector values. If the variable `UseRL` is set to 1 in the initialization file, then the commands will be limited to move no more than the rate limits allow each frame. The `INDX` vector is used to make effectors part of the 'active' set, which is determined by the control allocation routine. The default setting is that the seven aerodynamic surfaces are 'active' and the nine other effectors are 'inactive'. We recommend leaving the other effectors 'inactive' for now, although one may want to turn on the thrust vectoring.

NDI_CLAW/DynamicInversionControl/ControlAllocation/GetU
This block is where the control allocation problem, as defined in Section 4, is solved using the methods in Chapter 6 and Appendix A. There currently are six different methods to choose from. A method may be selected by changing the value of the variable `CAmethod` in the initialization file `INIT_NDI.m`.

The seven methods are:

MakeSquare (Option 0): This is the ganging allocation method described in Sections 6.4 and 8.5. A square matrix, as in Eq. (6.1), results. If the values are commanded to exceed their limits, the commands are clipped to the position limits. This option currently will not enforce rate limits on the commands.

WeightedPseudoClipped (Option 1): This method is a variant of the weighted pseudo-inverse described in Section 6.5.4 and in Eq. (6.35) (page 85). The weighting matrix is defined in the initialization file `INIT_NDI.m`.

If the values are commanded to exceed their limits, the commands are clipped to the position limits. This option currently will not enforce rate limits on the commands.

WeightedPseudoScaled (Option 2): This method is the same as the previous, except that if the values are commanded to exceed their limits, the commands are uniformly scaled so that all commands are at or within their limits.

DirectAllocation (Option 3): The direct allocation method was described in Section 6.6.2.

CGI (Option 4): This method implements the scaled cascaded generalized inverse described in Section 6.5.5.2. It is the basis for the 'effector blender' algorithm used in the X-35 control law (Bordignon, 2002).

VJA (Option 5): This is an implementation of Banks' method described in Section 6.8. The name of the module `VJA` is from Banks' original description of the procedure as the 'Vertex Jumping Algorithm'.

LP (Option 6): This is an implementation of the linear programming methods described in Appendix A. The variable `LPmethod` is used to select between several algorithms. `LPmethod` should be an integer between 0 and 5:

0 DB_LPCA Dual branch control allocation
- Linear program objective error minimization branch (1-norm)
- Control error minimization (1-norm)

1 DBinf_LPCA Dual branch control allocation
- Linear program objective error minimization (1-norm)
- Control error minimization (inf-norm)

2 DP_LPCA Direction preserving control allocation linear program

3 DPscaled_LPCA Direction preserving control allocation linear program
- Reduced formulation (solution scaled from boundary)

4 MO_LPCA Mixed optimization (single branch) control allocation linear program
- Objective error minimizing
- Control error minimizing

5 SB_LPCA Single branch control allocation linear program
- Direction preserving
- Control error minimizing

NDI_CLAW/DynamicInversionControl/ControlAllocation/PostProcess
This block adds the nominal effector values, adds a component in the null space to drive towards a preferred solution, and merges the 'active' and 'inactive' effector commands to get the 16 total commands that feed the ADMIRE simulation. This is the null-space restoring described in Section 7.4.4.

If the reader wants to investigate what happens if there is no null space restoring, delete the line coming out of the `DriveToPreferred` block.

References

Aerodata Model in Research Environment (ADMIRE), Ver. 3.4h, Swedish Defence Research Agency (FOI), Stockholm, Sweden, 2003.

Stevens, BL and Lewis, FL 1992 *Aircraft Control and Simulation,* 1st edn. John Wiley & Sons, pp. 80–81.

Bordignon, K and Bessolo, J 2002 'Control allocation for the X-35B,' AIAA 2002-6020 in *2002 Biennial International Powered Lift Conference and Exhibit, 5-7 November 2002, Williamsburg, Virginia*.