Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS) and the American Institute of Aeronautics and Astronautics, Inc.

A98-31557

ICAS-98-3,7,3

A TOOLSET FOR THE DESIGN OF AUTONOMOUS UAV SYSTEMS

F Valentinis
Wackett Aerospace Centre

W A Belton
Wackett Aerospace Centre

J Kneen
Dept. of Computer Systems Engineering

C Bil
Wackett Aerospace Centre

Royal Melbourne Institute of Technology
GPO Box 2476V, Mebourne, Vic 3001, AUSTRALIA

Abstract

Designers face many unique challenges in the development of autonomous UAV systems, many of which are not catered for using conventional aircraft design methodologies and tools. This paper presents a toolset, called "Wackett Aerospace Centre Uav Design Toolset" (WACUT), which provides the UAV designer with a means to evaluate and optimise a UAV system design. The toolset provides an indication of the impact of changing one of a full range of system variables from aircraft configuration and geometry through to autonomous control structure. It has proven invaluable in the design phase of an autonomous UAV system currently under development by the Sir Lawrence Wackett Aerospace Centre, with support from the Commonwealth Scientific and Industrial Research Organisation (CSIRO). The proposed flight vehicle system is being targeted specifically to atmospheric research applications.

Introduction

Atmospheric research applications present autonomous UAV system designers with a variety of challenges. Most noteworthy is the requirement for the vehicle to make course corrections based on results from real time analysis of atmospheric data, and to do so for long periods of time. This data may consist of wind direction and static pressure as well as results of on line gas sample analysis results.

With support from the CSIRO Division of Atmospheric Research, the Wackett Aerospace Centre at RMIT is developing an autonomous UAV System for long endurance atmospheric research applications. Various airframe and system configurations have been considered [8], [11]. A toolset has been developed to aid in this design process.

This paper discusses some of the aspects which differentiate autonomous UAV system design from the design of conventional aircraft or even remotely piloted UAVs. It describes a design toolset which takes these aspects into account, greatly aiding the overall design process. Finally, it presents some examples showing how the toolset can be used in the design of a UAV used in atmospheric research applications.

A particular focus of the paper is the UAV design and optimisation environment WACUT, which couples a fault tolerant control systems implementation platform, known as Coral, with a non linear six degree of freedom flight simulator. The flight simulator relies on a set of characteristics determined by a flexible parameter estimation code suitable for use with low Reynolds number flight vehicles. Using this system, changes can quickly be made to either the airframe or controller, then the resulting design can be fed into a full mission simulation, from which a set of performance measures can be derived.

Building an Intelligent, Autonomous Vehicle System.

Fully autonomous operation places unique requirements on the operation of a flight vehicle. Fault tolerance in particular is of paramount importance even in missions involving high degrees of uncertainty. These requirements are particularly evident in the case of vehicles designed for autonomous atmospheric research applications, where long endurance in often extremely hazardous environments is a critical requirement.

These unique requirements significantly complicate the design and implementation of all aspects of fully autonomous UAVs, from the airframe through to avionics systems. As proposed mission complexity increases, so too inevitably comes a significant increase in the complexity of a given vehicle's control regime, and the importance of matching an airframe design to the mission becomes more critical.

Airframe Design

The challenge of designing and optimising airframes intended for fully autonomous vehicle systems can be quite different to the problem of designing full sized manned

Copyright © 1998 by ICAS and AIAA. All rights reserved

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS) and the American Institute of Aeronautics and Astronautics, Inc.

aircraft. The first and foremost difference comes from the absence of human factors in the vehicle.

While this absence is in many ways advantageous, as it means that either more payload or fuel carrying capacity can be added to the vehicle, in general it means that the design will be considerably smaller than an average passenger carrying airframe.

For this reason the majority of UAVs operate in a rather low range of Reynolds numbers (from as low as 250,000 to 2,000,000). The vast majority of sizing and design analysis tools are designed for manned aircraft, and do not produce accurate results when presented with UAV design problems. This in itself leads to difficulties in both the design, and more particularly, the design analysis process.

The solution to this dilemma is to use more advanced analysis techniques for design validation such as CFD codes (9), and wind tunnel models (6) - but neither of these provide rapid feedback to the designer indicating the effects of changing critical design variables - an imperative requirement in the UAV airframe optimisation process.

A solution to this problem, implemented as an intrinsic part of the WACUT toolset, was to develop a flexible parameter estimation code based on an extended, nonlinear lifting line scheme scheme. The code produces results for all common UAV configurations, and supports a concise input format. The code, called the Aerodynamic Analysis Program (AAP), provides sufficient accuracy for use in the airframe optimisation process and returns prompt results.

A secondary consideration in the autonomous UAV airframe design process comes about from the fact that the vehicle is constantly under the control of an automatic control system. Variation of control parameters, and especially modification to higher level control structures will dramatically affect the performance of the vehicle in meeting mission objectives.

A particular example of this is the incorporation of thermalling algorithms into a control system, enabling the vehicle to reduce throttle and glide when possible, and thus increase range. When such techniques are used, a critical question becomes how much do such techniques increase range, and exactly how do changes in parameters such as aspect ratio affect total range when such a technique is used.

The close coupling of the automatic control system design process to the airframe design process is beneficial, and therefore support of this from any autonomous UAV design analysis toolset is very useful.

Avionic Systems Design

The requirement for UAVs to operate autonomously in complex and harsh environments inevitably leads to a requirement for quite large and complex avionic systems for which implementation can be quite difficult, particularly in a redundant, fault tolerant fashion. Reducing complexity at all levels in the avionic systems design process is therefore a goal worth highlighting in the case of fully autonomous UAV systems.

This requirement suggests that the development of tools and systems which simplify the implementation process at all levels is highly desirable. The process of developing any modern digital flight control system can be broken into a hierarchy of systems, each of which have associated with them unique challenges.

At the top level of the hierarchy sits the controller software, which generally runs under the control of a software runtime environment. This environment in turn is under the control of an operating system, which is executed on a redundant computer platform.

The process of designing autonomous flight controls involves numerous experts, the expertise of which must be focused at different levels of this hierarchy. A controls engineer, for example will develop a controller, using any of many techniques available, then pass the results of his work, in the form of a specification to a software engineer. The given specification is then implemented using a particular programming language, which is most often Ada 95 in the case of air vehicle systems.

This software is run via a runtime executive which handles such tasks as redundant instantiation, and this in turn relies on the operating system which in turn requires the electronics computer hardware to be running at an appropriate level of performance. These latter lower level systems are also quite complex, and are generally developed by separate teams.

Given that controllers are normally defined in terms of flow models described by block diagrams, the process of converting a controller specification into a program can often be laborious and unintuitive for the uninitiated. The main reason for this stems from the fact that modern structured languages such as Ada are designed for implementing procedural algorithms, and not necessarily the connectionist flow oriented models represented by block diagrams containing mathematical relationships.

This incompatibility in approach leads to the unnecessary difficulty of performing a specification conversion to match the programming approach of interest. For a complex smart controller designed to handle missions involving high degrees of uncertainty, this can lead to errors being introduced into the design process.

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS) and the American Institute of Aeronautics and Astronautics, Inc.

A key tool in the WACUT toolkit, called Coral, overcomes this incompatibility in approach via a programming language, which follows the block oriented paradigm used in controls design. The Coral compiler produces a pcode binary which can be run on any platform supporting the Coral runtime executive, ie a PC, network of workstations, or dedicated avionics hardware. When run on a parallel processing system, Coral also automates the process of redundantly implementing a given system, further reducing the work required to implement an avionic system.

A further difficulty which arises in the development of avionic systems relates to the basic issue of processor compatibility. In the past, the use of processors supporting instruction sets defined by standards such mil-std-1750 guaranteed that a single software development toolset could be used in a single avionics systems development project.

In order to reduce cost, modern civil systems, such as the Airbus A320 [(7)] system, however, uniformally use standard off shelf processors wherever possible. Selecting a single processor family for use across an entire flight control system may not be ideal, however.

Support for deployment of diverse processors in a single flight control system can be quite advantageous, particularly later in the life cycle of a system when the original processor used in a design may no longer be the optimal choice. Using arrays of diverse processors also has the advantage that different processor module designs are sufficiently dissimilar to minimise risk of identical faults appearing across different designs.

A major disadvantage of such an architecture is however, that barriers of incompatibility are introduced which, in a modular, distributed environment, may inhibit a runtime executive from loading code into certain processor modules.This could, for example prevent one processor module from taking over a task from a damaged processor in the event of a fault.

These compatibility barriers make the design of a fault tolerant software backbone more difficult as it locks resources away from a systems designer. Systems based on Coral, on the other hand, execute a pcode based binary format. Any processor running a native version of the Coral runtime is therefore compatible with the Coral binary format.

The effect of this is that these barriers of compatibility are removed from the system, allowing any system process to use any resource in the digital avionic system.

A UAV Systems Development Toolset

The WACUT toolset couples an efficient aerodynamic analysis code to a flight simulation environment which can be linked to a control system developed in the Coral programming language.

The aerodynamic code, known as AAP is designed from the ground up to work with Coral, and its native output format is actually a Coral subprogram.

This subprogram can be compiled in as part of a unified controller and flight simulator pair, resulting in a means to directly test the impact of small design changes on a UAV systems ability to satisfy performance objectives.

An almost limitless variety of missions can be programmed into the controller, allowing great flexibility in creating test cases for the complete Autonomous UAV system.

A post processor can be used to filter and tabulate the output of a simulation run so that concise plots can be generated which can be used as a basis of judging the ability of a vehicle system to perform in particular conditions.

Figure 1 shows the way the various tools and systems interface to form the complete toolset.

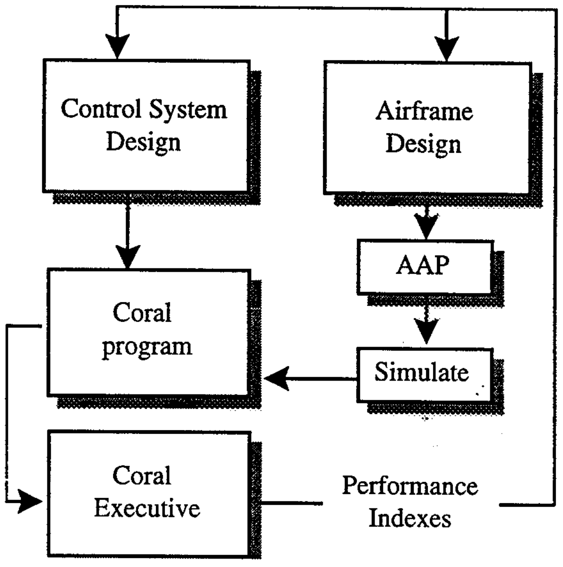

Figure 1: The WACUT Toolset.

Intelligent Controls Design Tools

At the core of the WACUT toolset is the Coral system, designed to simplify the process of developing fault tolerant digital control systems for autonomous vehicles. Coral consists of a programming tool set including a specialised programming language and a runtime executive.

The executive can either be run on a single workstation, a network of workstations, or on a specialised redundant flight computer. One such specialised computer is the PACER, which was designed at the Wackett Aerospace

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS)
and the American Institute of Aeronautics and Astronautics, Inc.

Center as a test platform for Coral. By deploying Coral on a platform such as the PACER, the development of a fault tolerant system can be almost totally automated, and the task of programming is simplified in comparison with the use of conventional programming languages, such as Ada. The syntax and structure of the Coral language corresponds closely to the structure of the conventional block diagrams used by controls Engineers.

Through the development of Coral, the usual controls design implementation loop has been considerably shortened . Using conventional computer platforms and programming approaches, the controls engineer would design and test the system in a simulated environment, then pass the design to software and electronics hardware design teams who then perform the implementation.

With this new approach, the controls designer can perform a simple translation from a block diagram to a program in the Coral language, then directly execute the resulting program on a custom built flight control computer such as the PACER, a PC or a network of workstations. The block diagrams can include transfer function blocks as well as fuzzy logic blocks.

Coral

The Coral programming language is based loosely on the COREL language [11],which is also aimed at real time parallel control applications. Programs written in Coral are compiled into a machine independent pcode format, ie a compact binary format which can be read into a virtual machine and executed via interpretation. Compiled programs are encoded in such a way that they can be broken up and automatically distributed across a parallel processor array, such as the PACER or a network of workstations. If possible, the entire processor array is used in the implementation of a particular controller.

As an extension of this capability, this distribution of function across the array can be done redundantly, so that multiple, redundant copies of controller blocks are placed on different processors. The runtime executive then takes care of any voting and execution of user supplied failure detection codes, which must be supplied as part of the Coral program describing the controller.

A series of small examples are presented in the following section to give the reader a contextual feel for the structure of the language. The first three examples define a series of reusable blocks - each of which are used in the final example, which links the former three to form a program which outputs a performance index indicating how well a given vehicle will track a given pitch hold command when under the control of a given controller.

The examples form a basic introduction to the most elementary features of the language only, and should not be seen as a complete description of its functionality. Readers interested in learning more about Coral should consult [10].

Coral Programs consist of a network of blocks, which are connected to one another via a series of linking expressions. Links can take many forms, including mathematical equations, fuzzy logic rules, and conditional expressions. Each block has associated with it a series of ports, to which links are connected, and internally may contain any number of nodes, which act as intermediate connection points, and further blocks, all of which are also joined using linking expressions. Figure 2 shows the major components of a Coral program.

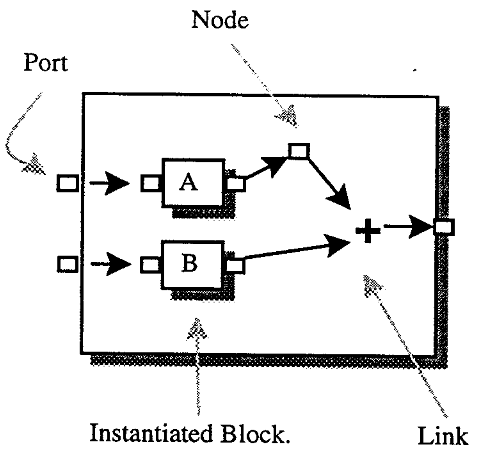

Figure 2: Constituent parts of a Coral program.

One of the most fundamental requirements of any digital systems development toolset is facilitation of difference equation implementation.These equations are used primarily for the implementation of transfer functions in the Z domain, but also for certain numerical methods, including numerical integration algorithms.

For the first example, a block type is created which implements a fourth order Adams Bashford integrator. Once a block type is created, it may be reused through instantiation in any other block type which is defined.

In this example, all inputs and outputs are logged for four time steps, hence the (4) after the name of the block type. To access previous values of an input, the @(t-x) suffix is added to a variable, where the x indicates the time lag required. The variables "t" and "dt" give the current time and the current time step, respectively. These four facilities allow any digital controller to be implemented.

```
#import <Coral.cor>

block type AB4_Integrator(4):SISO
```

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS) and the American Institute of Aeronautics and Astronautics, Inc.

```
{
  output:= input +
          dt/24.0*(55.0*input-
          59.0*input@(t-1)+
          37.0*input@(t-2 -
          9.0*input@(t-3));
}
```

Note that the block type created,AB4_Integrator, inherits its behaviour from type SISO. SISO, one of many predefined blocks provided by the Coral standard library, simply declares two port types, called input and output. Since the block AB4_Integrator inherits its behaviour from SISO, it too has these two ports. For this reason, no ports are defined in this example.

Ground up fuzzy logic support is an important element of Coral , as it provides a basis for the implementation of high level control behaviour using linguistic rule structures.

The following example shows fuzzy logic processing in action. It begins with the standard block and block declaration header, but rather than immediately declaring ports, it declares two specialized port types.

Port types are used to define sequences of valid fuzzy sets - each of which is characterised by a trapezoidal membership function. The functions are described by listing four numbers which give the vertex positions of each trapezoid at the 0,1,1 and 0 membership points on the function, respectively.

```
block type Longitudinal_Controller
{
  port type Angle_Error_Type
  {
    NL -PI  ,-PI  ,-0.30,-0.17;
    NM -0.30,-0.17,-0.17,-0.05;
    NS -0.10,-0.05,-0.05, 0.00;
    ZE -0.05, 0.00, 0.00, 0.05;
    PS  0.00, 0.05, 0.05, 0.10;
    PM  0.05, 0.17, 0.17, 0.30;
    PL  0.17, 0.30, PI  , PI  ;
  };

  port type Elevator_Deflection_Type
  {
    NL -0.70,-0.52,-0.52,-0.35;
    NM -0.52,-0.35,-0.35,-0.17;
    NS -0.35,-0.17,-0.17,-0.00;
    ZE -0.05, 0.00, 0.00, 0.05;
    PS  0.00, 0.17, 0.17, 0.35;
    PM  0.17, 0.35, 0.35, 0.52;
    PL  0.35, 0.52, 0.52, 0.70;
  };

  ports:

    Deflection:Elevator_Deflection_Type;
    Error:Angle_Error_Type;

  {
     if (Error = PL) Deflection = PL;
     if (Error = PM) Deflection = PM;
     if (Error = PS) Deflection = PS;
     if (Error = ZE) Deflection = ZE;
     if (Error = NS) Deflection = NS;
     if (Error = NM) Deflection = NM;
     if (Error = NL) Deflection = NL;
  }
}
```

The types Angle_Error_Type and Elevator_Deflection_Type are used in declaring the two ports used in the Longitudinal_Controller block type. Port types can be declared and used with any port, not just ports which are used for fuzzy logic processing, however the set values are ignored for all processing other than fuzzy logic processing.

The example continues by listing sequences of fuzzy logic rules defining the function of the controller.

As well as allowing for fuzzy controls design and digital implementation of SISO transfer functions, Coral also supports modern controls implementation through its comprehensive support of matrix math operations.

```
#import <Coral.cor>
#import <AB4_Integrator.cor>

block type Longitudinal_Model
{
  ports:
    x:Real[4];
    u:Real;
  nodes:
    A: Real[4][4];
    B: Real[2][4];
  {
    A=[ [-0.02, 7.32,-9.81, 0.00],
        [-0.72,-102 , 0.00, 26.3],
        [ 0.00, 0.00, 0.00, 1.00],
        [-0.00,-8.12, 0.00,-1.19]];

    B=[ [ 0.00],
        [-3.99],
        [ 0.00],
        [-10.5] ];

    x := AB4_Integrator(A*x+B*u);
  }
}
```

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS)
and the American Institute of Aeronautics and Astronautics, Inc.

This example shows how each Coral port can have a series of channels associated with it, in much the same way as arrays can be implemented in conventional programming languages. These channels can be used to store the elements of matrices. All common matrix operations are supported by Coral .

Note how the AB4_Integrator block, defined in a previous example, is reused in the construction of this block. The Coral language supports the use and reuse of block types via a function call style syntax, in which blocks are created, and linked to the parent block by linking the default 'output' port of the block to the port receiving the function call return value (x in this case), and linking the port(s) listed in the function call argument to the input ports of the new block. This 'function call' style of block creation is the first of two ways block types in Coral programs can be used in other blocks.

Blocks can be defined and saved to individual files, then called upon as building blocks. The following example does just that, using the second method of block type usage, by building a block which uses the blocks defined earlier.

The example creates an instance of the Longitudinal_Controller and Longitudinal_Model blocks, via a declaration in the "blocks" section of the block type declaration, and an integrator using the function call style syntax introduced earlier. The ports of these various blocks are then linked to create a system consisting of a controller linked to a longitudinal aircraft model. The controller is given a command to pitch the aircraft to 0.5 radian, and the integral of the error from the plant model is used as a simple performance index.

```
#import <Coral.cor>
#import <AB4_Integrator.cor>
#import <Longitudinal_Model.cor>
#import <Longitudinal_Controller.cor>

block type Experiment

  ports:
    Error,Total_Error:Real;
  blocks:
    Controller:Longitudinal_Controller;
    Model:Longitudinal_Model;
  {
    when (t < 5)
    {
      Model.u:=Controller.Elevator_Deflection;
      Error:=0.5-Model.x[3];
      Controller.Angle_Error := Error;
      Total_Error:=AB4_Integrator(abs(Error));
    }
  }
}
```

Ports within blocks instantiated within the host block are referred to by giving the block name, followed by a period and the port name . The first expression line of the block therefore connects the 'X' port of the Filter to the T port of the host block.

Most of the expressions in the expression block of this example perform a direct link function, with the exception of the "when" block. A when block is used to make a group of expressions dependent on a particular boolean condition. In this case, the performance index is calculated for five seconds of operation only.

Runtime Executive

The Coral runtime executive executes Coral programs once they have been compiled into the Coral binary format which is effectively a processor independent pcode.

The system is designed to operate on loosely coupled parallel processor arrays,but the executive hides the parallel nature of the target system from the controls designer/user, and gives the system the feel of a single machine. The executive orchestrates operation of the system and is responsible for both redundant loading and execution of programs on the parallel system. It is also responsible for system monitoring and reconfiguration in the event that one or more processor modules fail.

In a multiprocessor system, each processor must execute a copy of the Coral executive. Given that the Coral pcode is processor independent, once a processor system is running the Coral executive, it may connect to any processor pool of Coral processors. The types of processors or systems do not matter at all. For ground based simulation, therefore, a heterogeneous network of varied systems may be used.

The runtime is highly portable, and requires only the most basic of operating system services. A scheduler, for example is not required, nor is operating system support for any form of multitasking or threading - the executive provides all of these services.

The PACER

The PACER is a proof of concept flight computer designed with execution of Coral based programs in mind. It has been designed specifically for real time intelligent flight controls research, and provides a platform for the implementation of control systems optimised using the WACUT toolset.

The PACER is essentially a parallel processor array organised as a loosely coupled series of independent modules. The primary system module is the processor modules which are connected together via a series of communications ports. The number of modules is not limited, so whilst a small, non fault tolerant UAV controller can

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS) and the American Institute of Aeronautics and Astronautics, Inc.

be created using a single module, very large and complex fault tolerant systems may be created by linking many modules together.

The current configuration houses 3 processor modules each of which include a Motorola 68360 microcontroller, operating at approximately 5 VAX MIPS. Each module includes 4MB of DRAM, and 512 KB of FLASH memory for non volatile on line data storage. The modules are upgradeable to 32MB of DRAM.

Each module is connected to four separate busses through which data is transferred at a rate of 2M bits per second. The four links ensure that the modules are connected in a redundant mesh structure, in which intermodule communications are possible even in the event of a link failure.

Each of the modules of the PACER may be connected to an optional IO interface card, which provides high speed 16 bit data acquisition and motor control functions, effectively forming a link from the computer to the airframe.

Flight Vehicle Simulation.

Flight Control Systems implemented via Coral programs can be tested using a flight simulator code, which is also implemented in the Coral language. The Coral runtime executive may be run on a PC in addition to over an RTOS on an embedded target such as the PACER, therefore rapid prototyping and optimisation is possible by controlling an airframe in simulation.

The simulator supports full six degree of freedom operation, and achieves nonlinear operation by switching between banks of flight derivatives. A nonlinear gust model is also provided.The vehicle simulation code is generic in nature, and takes its derivative information from a Coral block which can either be written by hand, or generated automatically by the AAP program.

Aerodynamic Analysis Program

The Aerodynamic Analysis Program (AAP) was developed to estimate an aircrafts aerodynamic load distributions as well as its aerodynamic derivatives. Although not developed exclusively for UAV's, its development has been centered around their design requirements.

The aerodynamic load distributions are determined by the Aerodynamic Load Distribution Program (ALDP) at given design points.The aerodynamic derivatives, determined using the Aerodynamic Derivatives Program (ADP), are used to assess the aircraft handling properties.

Figure 3 indicates the overall AAP structure. The independence of the ALDP and ADP is evident, allowing them to run in conjunction or individually.

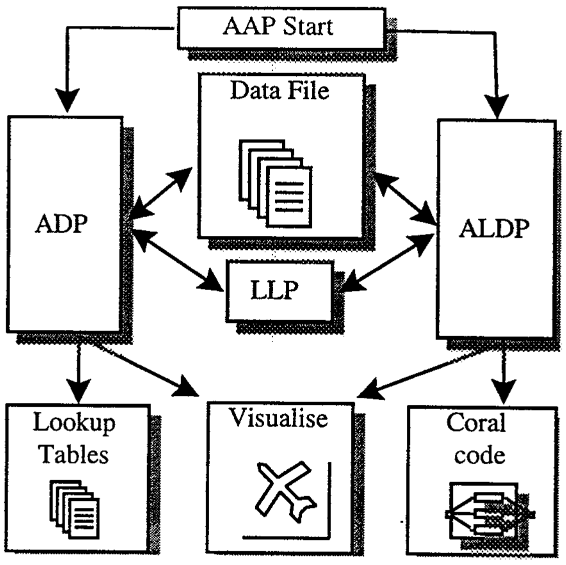

Figure 3: AAP Structure

The entire AAP has been developed using Matlab 4.2 and at this stage has only been implemented in that environment.

Aerodynamic Load Distributions

The Aerodynamic Load Distribution Program (ALDP) primary role is to provide aerodynamic distributions for the various aircraft components. At this stage two component types are supported; surfaces and bodies, analysis of which is undertaken by a modified lifting line method and a slender body method [4] respectively.

Nonlinear Lifting Line Method Traditional Implementation of the Lifting Line Method (1) [3] assumes that the lift-curve-slope is linear ($a = a_{linear}$), thus an error is introduced in the post linear segment of the lift-curve as is evident in figure 4. This is a reasonable approximation for applications where Reynolds Number is high. As Reynolds number decreases the region where a is linear decreases, though the non-linear region remains relatively constant.Thus the result is an increase in the region where error is present with the decrease in Reynolds Number.

$$\frac{ca}{8s}(\alpha - \alpha_0) = \sum_{n=1}^{m} \sin(n\theta)(1 + n\frac{ca}{8.s.\sin(\theta)}) \qquad (1)$$

To reduce this error the monoplane equation is modified as per (2) so that C_L and its corresponding a ($a = a_{local}$) are used rather than computing the C_L on the fly from a_{linear} and a0. This reduces the errors in the post-linear region.

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS)
and the American Institute of Aeronautics and Astronautics, Inc.

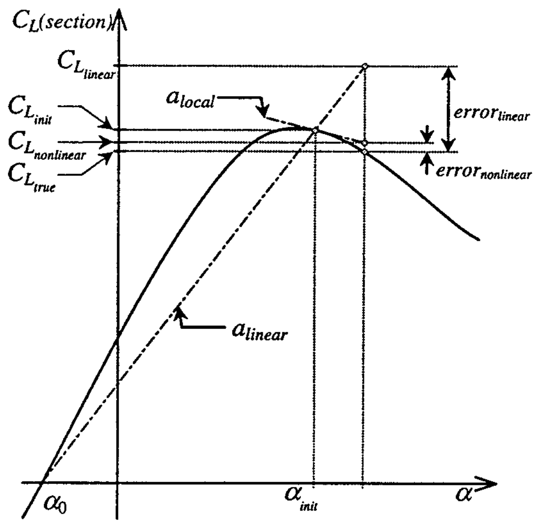

Figure 4: Nonlinear Lift Curve Slope

$$\frac{c}{8s}C_l = \sum_{n=1}^{m} \sin(n\theta)(1 + n(\frac{ca_{local}}{8s\sin(\theta)})) \qquad (2)$$

The traditional LLM had been further extended to allow for wing sweep and dihedral. This has been achieved by applying a transformation to the physical wing, as shown in figure 5. The physical wings quarter chord is 'rolled out' to form a 1-D lifting line, i.e. no sweep or dihedral. Local velocity, α and β are also transformed across the span to compensate for the 'roll out'. Once the lifting line has been analysed using the modified LLP a further transformation is applied to resolve all forces to the stability axis.

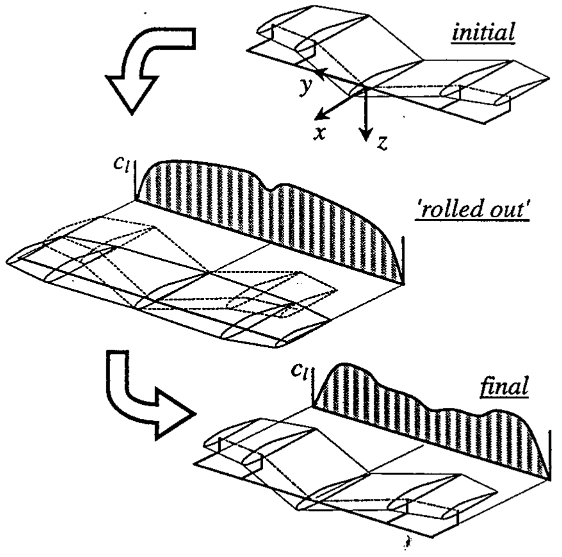

Figure 5: Transformation and 'roll out' of the wing

Eppler PROFILE Program The 2-D data required by the LLM is found using the PROFILE [1] program. PROFILE being deemed more suitable than, the comparable, XFOIL due to lower computation times [5], a factor that is critical for aircraft having many different section / new sections / flap settings.

Airfoil data files, in the form of C_L, a, C_D, C_M and Aerodynamic Centre lookup tables, are produced from the PROFILE output file. These two dimensional lookup tables cover a range of Reynolds Numbers and angles of attack, thus forming a non-linear model of section data. The Reynolds Number and angle of attack ranges have been chosen to cover the range expected by piston-propeller powered and unpowered UAV's.

Slender Bodies The aerodynamic load distribution on the bodies of the aircraft are found by the superposition of two flow types; axisymetric flow around a body of revolution and traverse flow around a body of revolution. The use of this theory results in the assumption that the bodies' cross-section is circular and is pointed. These assumptions limit the application of this theory to providing C_M data and interference effects. Methods more suitable for C_L and C_D distributions are presented by Nielsen [5] and constitute work-in-progress for the AAP.

As with the Lifting Line Method discussed above, allowances have been made for fuselages with centre lines that are not straight, i.e. cambered fuselages. The fuselage centre line is 'rolled out' parallel to the aircrafts x-axis [fig.4]. As with the wings quarter chord the local velocity, alpha and beta are all transformed to compensate for the 'roll out'.

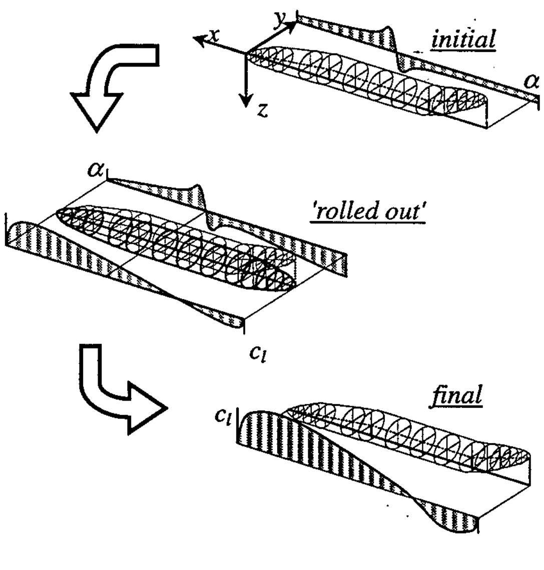

Figure 6: Transformation and 'roll out' of the fuselage

The axial flow velocity components are given by equations 3 and 4.

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS)
and the American Institute of Aeronautics and Astronautics, Inc.

$$q_r(r,x) = \frac{1}{4\pi}\int_0^l \frac{u(x_0).S'(x_0)r(x_0)}{[(x-x_0)^2+r(x_0)^2]^{\frac{3}{2}}}dx_0 \quad (3)$$

$$q_x(r,x) = \frac{1}{4\pi}\int_0^l \frac{u(x_0).S'(x_0)(x-x_0)}{[(x-x_0)^2+r(x_0)^2]^{\frac{3}{2}}}dx_0 \quad (4)$$

The transverse flow velocity components are given by equations 5, 6 and 7.

$$\begin{aligned} q_r(r,\theta,x) =& \frac{1}{4\pi}\int_0^l \frac{\mu(x_0)\sin(\theta)}{[(x-x_0)^2+r(x_0)^2]^{\frac{3}{2}}}dx_0 - \\ & \frac{3}{4\pi}\int_0^l \frac{\mu x_0 \sin(\theta) r(x_0)^2 dx_0}{[(x-x_0)^2+r(x_0)^2]^{\frac{5}{2}}}dx_0 \end{aligned} \quad (5)$$

$$q_\theta(r,\theta,x) = \frac{1}{4\pi}\int_0^l \frac{\mu(x_0)\cos(\theta)}{[(x-x_0)^2+r(x_0)^2]^{\frac{3}{2}}}dx_0 \quad (6)$$

$$q_x(r,\theta,x) = \frac{-3}{4\pi}\int_0^l \frac{\mu(x_0)(x-x_0)r(x_0)\sin(\theta)}{[(x-x_0)^2+r(x_0)^2]^{\frac{5}{2}}}dx_0 \quad (7)$$

Moment Distribution normal to the flow is given by equation 8.

$$M_{normal}(x) = \frac{1}{2}\rho Q_\infty^2 \int_0^{2\pi} (x+RR')\sin(\theta)R(x)C_p\delta x d\theta \quad (8)$$

where:

$$C_p = \frac{-2q_x}{Q_\infty} - \frac{\alpha}{Q_\infty}(q_r sin\theta + q_\theta \cos\theta) - \frac{q_r^2+q_\theta^2+q_x^2}{Q_\infty^2}$$
$$\mu = 2\pi R^2\sqrt{v^2+w^2}$$

u,v and w are local velocity components.

Q_∞ is the local summation of u,v and w.

R is the local body radius.

Component Interference As components are analysed separately interference effects can be incorporated by two different methods. The first is an iterative procedure, the second determines interference conditionally i.e. a given component can only cause interference to specified components.

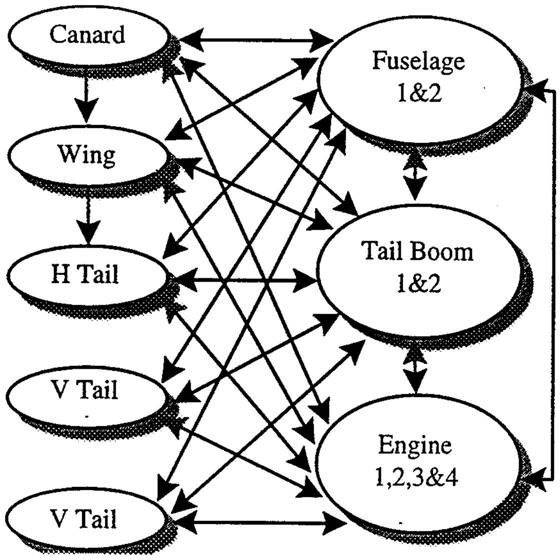

Figure 7: Conditional Interference of Components

The second method, illustrated in figure 7 was chosen primarily due to its significantly lower computational time. One pass is required for this method whereas a minimum of $2(g-1)$ iterations are required for the iterative procedure, where g is the number of groups.

Interference caused by bodies is determined by using equations 3 to 7 above and interference caused by lifting surfaces is determined by integrating the effect of trailing and bound vortices across the span. It should be noted that all interference effects are calculated using the physical, rather than the 'rolled out', location of the components.

Input And Out Format Table 1 outlines the data required by the ALDP for the analysis of an aircraft at any given design point. The same data is required by the ADP and will be discussed in more depth later. Brackets indicate that an array of values is input for a given parameter, to allow for derivative determination.

TYPE/Group	Input Data	
AIRCRAFT	Aircraft Ref.:	[velocity],[altitude],$\delta temp$, [alpha],[beta],[roll rate], [pitch rate],[yaw rate],CG
SURFACE -canard -wing -horizontal tail -vertical tail(1&2)	Surface Ref.:	x,y,z,incidence
	Section Prop:	x,y,z,twist,chord,name
	Control Prop.:	inboard section, outboard section, hinge position, [deflection]
BODY -fuselage(1&2) -tail boom(1&2)	Body Ref.:	x,y,z,incidence
	Section prop:	x,y,z,width,height
ENGINE -engine(1,2,3&4)	Disk Ref.:	x,y,z,incidence,toe, [exit velocity]
	Nacelle Prop.:	x,y,z,width,height

Table 1: Analysis Input Data

A total of 13 groups and 16 associated control surfaces

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS) and the American Institute of Aeronautics and Astronautics, Inc.

are allowed by the AAP (and ALDP), although only the wing group is required for analysis to be performed. Such flexibility allows a wide variety of configurations to be analysed without rewriting any part of the AAP.

Aerodynamic Derivatives

The core of the Aerodynamic Derivative Program is identical to the Aerodynamic Load Distribution Program. The ADP utilises the same input files, analysis procedures and approach to interference but produces quite different results. Whereas the ALDP analyses a single design point, the ADP deals with large amounts of the flight envelope.

Derivative Determination Derivatives are determined using a looping procedure. For instance aerodynamic quantities such as C_L, C_D and C_M are found (in the form of matrices) over a range of velocities, altitudes and angle of attack (in the form of arrays). These quantities are then differentiated with respect to the velocity arrays.

The major penalty inherent in this method is the computation time associated with the extensive utilisation of iteration in the Matlab environment. Execution times in the current Matlab version are in the order of hours to determine a full set of derivatives with input arrays having 10 values i.e. each derivative array is made up of 1000 values. Steps are, however underway to convert this critical code to a compiled language. This will significantly improve performance. Significant speedup with the current implementation can also be achieved by reducing the output table size, but with a corresponding loss of accuracy.

Input and Output The ADP outputs derivative data in two forms; it is written to an aircraft data file in the form of a collection of 2D matrices and is also written to a Coral source code file which can be linked in as part of the WACUT flight simulator.

Test Case: A UAV System for Atmospheric Research

Through cooperation with the CSIRO Division of Atmospheric Research, a series of atmospheric research related missions have been proposed for application of a future RMIT UAV system. These mission scenarios, which are described below, have formed initial test cases for the WACUT system.

CSIRO Missions.

CSIRO's existing air sampling program currently utilises a manned aircraft for air sampling in Bass Straight from Melbourne to Cape Grim in Tasmania. This successful,

Longitudinal
$CL_{MD} = [velocity, altitude]$
$CD_0 = [velocity, altitude]$
$CD_\alpha = [\alpha, velocity, altitude]$
$CD_M = [\alpha, velocity, altitude]$
$CD_Q = [Q, velocity, altitude]$
$CD_{\delta cf} = [\delta cf, velocity, altitude]$
$CD_{\delta wf} = [\delta wf, velocity, altitude]$
$CD_{\delta e} = [\delta e, velocity, altitude]$
$CL_0 = [velocity, altitude]$
$CL_U = [\alpha, velocity, altitude]$
$CL_{d\alpha} = [d\alpha, velocity, altitude]$
$CL_\alpha = [\alpha, velocity, altitude]$
$CL_M = [\alpha, velocity, altitude]$
$CL_Q = [Q, velocity, altitude]$
$CL_{\delta cf} = [\delta cf, velocity, altitude]$
$CL_{\delta wf} = [\delta wf, velocity, altitude]$
$CL_{\delta e} = [\delta e, velocity, altitude]$
$CM_U = [\alpha, velocity, altitude]$
$CM_{d\alpha} = [d\alpha, velocity, altitude]$
$CM_\alpha = [\alpha, velocity, altitude]$
$CM_M = [\alpha, velocity, altitude]$
$CM_Q = [Q, velocity, altitude]$
$CM_{\delta cf} = [\delta cf, velocity, altitude]$
$CM_{\delta wf} = [\delta wf, velocity, altitude]$
$CM_{\delta e} = [\delta e, velocity, altitude]$
Lateral
$Cl_\beta = [\beta, velocity, altitude]$
$Cl_P = [P, velocity, altitude]$
$Cl_R = [R, velocity, altitude]$
$Cl_{\delta a} = [\delta a, velocity, altitude]$
$Cl_{\delta r} = [\delta r, velocity, altitude]$
$Cm_\beta = [\beta, velocity, altitude]$
$Cm_P = [P, velocity, altitude]$
$Cm_R = [R, velocity, altitude]$
$Cm_{\delta a} = [\delta a, velocity, altitude]$
$Cm_{\delta r} = [\delta r, velocity, altitude]$
$Cn_\beta = [\beta, velocity, altitude]$
$Cn_P = [P, velocity, altitude]$
$Cn_R = [R, velocity, altitude]$
$Cn_{\delta a} = [\delta a, velocity, altitude]$
$Cn_{\delta r} = [\delta r, velocity, altitude]$

Table 2: Aerodynamic Derivative Data

but to date expensive programme has potential to be replaced by a UAV programme $^{(2)}$. Potential cost savings through the use of an UAV may permit the program to be extended. The frequency of flights to Cape Grim would be increased and, if possible, similar profiles flown at other locations. Regions of particular interest are off south-western Western Australia and offshore from Darwin. The two missions currently flown by the CSIRO are a pollution plume detection mission and a continuous carbon dioxide sampling mission.

Plume detection: This mission involves the direct measurement of the cross-section area of the pollution plume emanating from Melbourne, for the purpose of better estimating emissions of a wide range of trace constituents. Such a measurement would probably be conducted in the vicinity of the Baseline Monitoring Station at Cape Grim on the north-western tip of Tasmania. The Baseline Station would provide concentrations of the gases of interest at ground level during the flight.

A WACUT based simulation of this mission is being developed, which links a random pollution spread model to the WACUT flight simulator code, providing a controller with a series of pollution level inputs which can be used

Copyright © 1998, by the International Council of the Aeronautical Sciences (ICAS)
and the American Institute of Aeronautics and Astronautics, Inc.

for tracking and mapping the outside of a pollution plume. The WACUT output from the simulated mission run in this case can be used to determine how closely the vehicle tracks the plume, and hence can be used as a basis for controller and

Continuous carbon dioxide measurements: In a second mission, measurements are taken of the vertical and/or horizontal distribution of the gas in the atmosphere, over large regions. Such data has the potential to provide information about the global carbon cycle and the processes of atmospheric transport (the movement of gases from place to place), which is not available using present measurement strategies.

The important design parameters in this mission type are endurance, and ability to make very tight turns, spiraling upwards. Design evaluation of these traits can be done with a basic WACUT system, without requiring special extensions as per the previous mission. A waypoint sequence is simply programmed into the system and the mission output is observed to determine the vehicle performance.

Future Work

The toolset described currently performs an analysis and simulation function, ie it alone cannot perform any sort of system optimisation on a system design. There are a large number of variables which could be optimised using an automated technique in any UAV system - certainly optimising for all of these parameters at once is quite impractical, especially given the time taken to run an analysis and mission simulation.

It may, however be possible to encapsulate a full toolset run , simulating a restricted manoeuvre into the evaluation function of an optimisation routine. This would allow the designer to optimise a restricted number of variables whilst holding others constant.

Conclusions

A toolset has been created which greatly simplifies the design of autonomous UAV systems. The toolset provides a supporting role in both the airframe and avionic systems design process, reducing the overall time required to design an autonomous UAV system.

The toolset consists of three main components: a flight vehicle aerodynamic parameter estimation code, a flight and mission simulator, and finally a control systems development environment supporting the Coral runtime environment.

The toolset as a whole permits evaluation of an airframe when under the control of a specific control system,and allows the designer to carefully observe the performance of the combined system both in performing manoeuvres and in attempting to satisfy mission goals.

The system is currently being used in the design process of a flight vehicle for atmospheric research, being codeveloped by RMIT and the CSIRO Division of Atmospheric Research.

References

(1) R Eppler and D Somers. A computer program for the design and analysis of low-speed airfoils. Technical Report TM 80210, NASA, 1980.

(2) C Allison et al. Global atmospheric sampling laboratory (gaslab): supporting and extending the cape grim trace gas programs. Baseline Atmospheric Program, 1993.

(3) E Houghton and N Carruthers. Aerodynamics for engineering students. Edward Arnold, third edition edition, 1982.

(4) J Katz and A Plotkin. Low-speed aerodynamics From wing theory to panel methods. McGraw Hill, 1991.

(5) J Nielsen. Missile Aerodynamics. McGraw-Hill, New York, 1960.

(6) A Pope. Wind tunnel testing. Wiley, 1947.

(7) CR Spitzer. Digital Avionics Systems-Principles and Practices. McGraw-Hill, second edition edition, 1993.

(8) LA Thompson and C Bil. Design and flight trials of multi purpose autonomous flight vehicle system. In Proceedings, PICAST2-AAC6, 1995.

(9) R Trofoletto. Estimation of aerodynamic load distributions on the pc9/a aircraft using a cfd panel code. Technical Report DSTO-TR-00XX, DSTO Aeronautical and Maritime Research Laboratory, 1994.

(10) F Valentinis. Coral language specification and user manual. Technical Report 10253, RMIT Aerospace UAV Design office, 1998.

(11) F Valentinis, C Bil, and P Riseborough. Development and trials of an autonomous flight control system for uav's. In Proceedings, ICAS '96, 1996.