A98-31444 ICAS -98-1,1,2

A DISTRIBUTED APPROACH TO THE DESIGN OF A REAL-TIME ENGINEERING FLIGHT SIMULATOR

D J Allerton

College of Aeronautics
Cranfield University, UK

Abstract

Engineering simulators are used in the design and evaluation of aircraft systems. This paper describes the design of an engineering simulator based on commercial off-the-shelf equipment. An array of PCs, coupled via Ethernet cards, is used to sustain an update rate of 50 Hz for all the primary elements of a real-time simulation.

The paper describes the organisation of the simulator software to provide a rapid prototyping environment for the design of flight control laws, aircraft displays and avionics systems. The aircraft displays are based on standard SVGA cards and the methods to provide realistic aircraft displays for primary flight instruments, engine instruments and navigation displays are outlined. Data is recorded and displayed at 50 frames per second, generating 1.5 M bytes of data per minute which can be analysed on-line or saved for subsequent off-line analysis. The paper describes a method to acquire and record essential flight data using the XMS memory of a PC.

The papers includes examples of real-time aircraft displays and aircraft responses for several aircraft to illustrate the effectiveness of this approach to engineering simulation. The paper concludes with a summary of the overall performance of the simulator in meeting an overall real-time iteration rate of 50 Hz for flight models and aircraft displays.

Introduction

Flight simulation has been used in pilot training for civil and military organisations for over twenty years. In this role, emphasis is given to simulator fidelity[1] to ensure a positive transfer of training from the simulator to the aircraft. In the last fifteen years, there have been considerable advances in avionics for both civil and military aircraft including the integration of high speed databuses, the provision of EFIS displays on the flight deck, the introduction of new navigation systems (e.g. GPS) and the development of fly-by-wire flight control systems.

Flight testing for a new aircraft is both time consuming and expensive. While the purpose of flight testing is to establish aircraft performance and stability throughout the flight envelope, an additional role of flight testing is the evaluation and certification of aircraft systems and avionics. Data from flight tests is used to confirm the effectiveness of aircraft systems in meeting a functional specification and to assess the effect of these systems on pilot performance.

It is clear that a considerable part of the design and evaluation of avionics equipment can be undertaken in a flight simulator[2], provided that the fidelity of the simulator is appropriate to the evaluation task and that data can be derived from simulator based trials to evaluate criteria required for the validation and certification of aircraft systems. In this role, airborne flight trials are used to validate the results of simulation studies.

While engineering simulators may contain software and systems which are also found in full mission simulators, the main requirement is to provide a rapid prototyping environment to be able to configure the simulator (and the simulator software) in a simple and straightforward manner appropriate to a wide range of tasks. A second requirement is to reduce the cost of simulation equipment in an engineering flight simulator.

Up to the last few years, the computers used in real-time flight simulation were restricted to high performance mini computers and special-purpose cards. However, the performance of PC systems in recent years has increased to a level where commercial off-the-shelf PCs can be used to provide the computing performance required for real-time flight simulation. In addition, the low cost of peripheral equipment for the PC can lead to a cost effective solution in real-time simulation, using commercial off-the-shelf hardware.

There are two areas where PC systems can be exploited in engineering simulation. Firstly, PC graphics performance may be sufficient for real-time aircraft displays. In an engineering simulator, the aircraft displays only require sufficient functionality and fidelity to enable a pilot to undertake flight test procedures. Secondly, processing capacity need not be restricted to a single processor. By connecting PCs via Ethernet, data can be shared and transferred between processors at data rates which sustain real-time iteration rates in excess of 50 Hz.

21st ICAS Congress
13-18 September 1998
Melbourne, Australia

While the raw processing afforded by a distributed array of PCs may provide the basis for a real-time engineering simulator, consideration must also be given to the design environment to allow an engineer to configure the simulator for specific tasks. This requirement is influenced by the provision of a modular architecture of software modules which can be configured in a straightforward manner. These requirements extend to flight modelling, aircraft displays, navigation systems, the integration of avionics sensors and equipment and also to flight data recording from simulator trials.

This paper describes an engineering flight simulator developed in the College of Aeronautics at Cranfield University, where it has been used to support teaching and research at postgraduate level in avionics, flight control system design and human factors studies. The aim of this programme is to exploit standard PC equipment by means of an integrated set of software modules which can be rapidly configured to provide a flight test environment.

System Organisation

The primary elements of a flight simulator are shown in Figure 1. The flight model provides the focal point of the simulation, responding to pilot inputs and instructor commands and generating flight data for the other simulator modules. For an engineering simulator, the motion platform is usually omitted. Previously, there has been a natural partitioning of systems such that the dynamic real-time models for aircraft dynamics, engine models, navigation systems, avionics and aircraft displays are combined in one system; the visual system is provided as a specific item and the instructor station is a separate computer system and user interface[3].

In addition to the software modules, databases are provided for the flight model data, for the navigation systems and also for the visual system. These provide aircraft specific data for modelling the airframe and engine dynamics, details of navigation aids and airfields and also topological terrain data needed for the visual system. Considerable care is required in the formulation of these databases and in the selection of axes systems such that the dynamic motion, navigation displays and the visual system are correctly aligned.

For a real-time engineering simulator, all the simulator modules are slaved to a common frame rate or iteration rate, typically 50 Hz. This implies that all the code within each module is executed within the period of a single frame (with a small margin to avoid exceeding the frame rate) and that all data transfers are completed within this frame. In practice, data used in a simulator may be acquired and processed by one module and then passed to other modules. Considerable care is needed to ensure that latencies of data transfers through the simulator are kept to a minimum[4]. For example, data from an inceptor is used in the computation of the aircraft motion and the aircraft position and attitude and then passed to the visual system which may in turn, take several frames to compute the displayed image.

Software modules in an engineering simulator can be organised at three levels:

- at the lowest level, the modules are represented as procedures; data is passed between modules either in the parameter list of a procedure or via an area of shared (or common) memory.
- at a higher level, software modules can be developed for specific applications. The visibility of procedures and data structures can be defined to provide communication between modules and to restrict access to specific code or data. Data sharing between modules can be provided by parameter passing or shared memory protocols, including semaphore mechanisms. Generally, these modules are contained within a single processor.
- at the highest level, modules can be bound to a processor; access to code and data structures is via message passing between processors[5].

In the past, modules were allocated to processors subject to the constraints of code size and storage for data structures. With the reduction in the cost of memory devices, the main criteria for binding processes to processors is now strictly data throughput and the need to meet the overall processing requirements.

For each module, it is possible to predict the worst case processing needed for computations required in any frame. However, there are two timing considerations which influence the overall processing throughput. Firstly, the processor speed (particularly to execute floating-point arithmetic operations) and the size of code will determine the upper bound of the processing rate. Secondly, the amount of data transferred between modules and the overhead of any transfer mechanisms will also influence the overall processing. For example, the overhead to access shared data in a sequential processor may be negligible whereas data provided once per frame from another processor may not be available until a few milliseconds into a frame.

Ethernet

Ethernet has become an accepted standard for local area communications over the last five years and, as a consequence, a range of low-cost network cards have been produced for PCs and workstations. In addition to this hardware development, communications protocols have been devised and approved for use with Ethernet cards. However, the majority of applications for local area networks are not constrained by real-time considerations,

rather, emphasis is given to data integrity. While the data integrity of Ethernet is excellent, protocols are used to guarantee data transfers, including re-try mechanisms.

Ethernet is based on packet transfers. In other words, the benefits of Ethernet are realised by transmitting blocks of data rather than individual bytes of data. The minimum packet size is 46 bytes and the maximum packet size is 1500 bytes, per transfer. The data is encoded and transmitted serially using coaxial cable which provides a reasonable level of noise immunity. Although Ethernet affords a data transfer rate of 10M bits per second, this is only the data rate over a cable between adjacent terminals. In practice, the actual data rate between terminals (or nodes) is considerably less than this for a number of reasons:

- contention for the network - transfers may be delayed until the bus is available
- time to formulate the packet for transfer
- time to receive and copy the packet and also to copy the packet.

The requirements of a local area network are quite different from real-time simulation: delays of several seconds can be tolerated, data integrity can be improved by re-transmitting corrupted data and the network bus loading will vary according to the number of users and the volume of traffic. This first point is fundamental to Ethernet. To transmit packets, it is necessary to check the result of the transmission. As Ethernet is based on collision sensing, simultaneous transfers over the bus are detected and data is re-transmitted after a short period. This re-try mechanism is provided at the card level and the applications software is isolated both from the transfer protocol and the hardware bus interface.

This arrangement is totally unsuited to a distributed approach to real-time flight simulation where data must be guaranteed to be transferred within a very short period during every frame. Nevertheless, Ethernet does offer several advantages for real-time simulation:

- the data rate of 10M bits per second allows large blocks of data to be transferred between computers
- the cost of an Ethernet card is less than $50
- higher performance Ethernet cards are available providing 100M bits per second data rates
- the bus integrity is very high (the bit error is of the order 10^{-5} bits per second)
- public domain Ethernet packet drivers are available.

The major problem is that bus transfers must be deterministic in real-time applications. Although re-try mechanisms can sustain very high throughput on lightly loaded networks, it is difficult to ensure worst-case transfer rates under all conditions. However, this problem is simplified for flight simulation because the sequence of transfers is known and typically, there are only a few hundred variables to transfer during any frame.

The problem of bus contention is overcome by ensuring that only one terminal transmits at any time. This arrangement is similar to avionics buses such as Mil-Std 1553B, where the bus is multiplexed by the terminals and bus ownership is controlled by a dedicated bus controller. This is not strictly necessary in flight simulation, as the sequence of bus transfers is likely to be repeated in every frame. The completion of individual transfers can be used to signal the availability of the bus for the next transfer. This scheme is aided by the provision of a 'promiscuous' mode in Ethernet, whereby all terminals are able to monitor all bus transfers, affording a broadcast mode.

As each packet contains not only the data required for a bus transfer, but also the node addresses of the transmitter and receiver, it is straightforward to monitor the sequence of transfers and to follow a predetermined sequence of bus transfers. This assumes a set of co-operating processes. In a dedicated engineering simulator, where the processors and processes are simulator modules, it is reasonable to assume that all modules fulfil an agreed and synchronous set of transfers.

In performance tests, back-to-back transfers of a maximum block length resulted in an overall data throughput of 2.4M bits per second which includes formation of the packet at the transmitter, checking and re-transmission of the packet at the receiver. These tests were performed using two 33 MHz 486 PCs and a 16-bit non-optimising compiler with run-time checks enabled.

The applications software for the simulator interfaces to a low-level packet driver obtained from the Crynwr public domain packet driver collection which implements the PC/TCP protocols[6], which are supported by the majority of Ethernet card vendors. These low-level packet drivers provide a common set of access primitives at the Medium Access Control (MAC) layer[7].

The packet driver is invoked via a software interrupt to establish a link between an applications program and the packet driver and also to write packets. Received packets are copied to a user defined region which is also established by means of a software interrupt. In the case of real-time simulation, it is assumed that packets will be transmitted and received in sequence at the start of each frame. Consequently, it is straightforward to sequence the writing and reading of packets.

Typically, a 500 byte packet is transferred in approximately 2 ms. This enables the flight model to broadcast essential flight data to other systems, in the form of a packet containing over 100 floating-point values, at the start of every frame. Packets arriving from

other processors are queued as background tasks and extracted as required during the frame. Initially, each processor interrogates a common file containing the list of transfers per frame and constructs the list of packet transfers to be executed in a pre-defined order every frame. Once the initial links are established with the packet driver, calls are invoked to read or write packets, with the source and destination Ethernet addresses appended to the data packet for each transfer.

The packet format is defined as a Modula-2 data structure and is used by all modules; it specifies the location of variables in the packet, their size and type. All modules use the same packet format (and conventions for units) and it is straightforward to alter this packet definition for all modules and simply recompile the simulator modules, avoiding any explicit re-ordering of the packet contents.

In practice, the use of a standard Ethernet packet format offers a number of advantages:

- it provides a local interchange 'standard' between software modules, independent of the platform or programming language - modules simply read or write packets in the prescribed format
- partitioning of modules or re-allocation of modules to processors can be achieved by adding a few extra Ethernet transfers
- the visibility of a module can be defined at the processor level so that shared data is only available via Ethernet transfers and private code or data, local to a processor, is hidden from other processors.

Flight Modelling

Flight modelling underpins an engineering flight simulator. The aircraft dynamics are computed at a sufficiently fast iteration rate that the aircraft motion computed by the simulation is continuous; in particular, the pilot in-the-loop cannot discern any discontinuities in the solution of the equations of motion. In practice, this requirement means that the equations of motion for the airframe aerodynamics and the engines must be solved at a minimum iteration rate of 50 Hz.

The computing tasks to be performed in the solution of the equations of motion are four-fold:

- derivation of the aerodynamic and propulsive forces and moments on the aircraft
- computation of the linear and angular accelerations on the aircraft
- computation of the linear and angular velocities of the aircraft, in the appropriate frame of reference
- computation of the aircraft position and attitude relative to the visual and navigation databases

The aerodynamic forces and moments are defined in terms of the aerodynamic and engine data for the aircraft[8]. Typically, aerodynamic data is available in the form of data tables or graphs for the aerodynamic derivatives[9]. Often, these variables are defined in terms of two or three variables, for example, $C_{m\alpha}$ (the pitching moment with respect to the angle of attack) is typically defined as a function of the angle of attack (α) and also includes the effects of mach number, flap setting and undercarriage position. It is possible to pre-process the data to simplify table look-up functions or polynomial fits to the data, however the computation of each aerodynamic derivative requires table access and interpolation[10], which may prove time consuming.

As a processor is bounded by its clock speed, considerable attention is given to the organisation of aerodynamic data and the real-time computation of the forces and moments. This constraint also applies to the numerical integration of the accelerations to derive velocities and the subsequent integration of aircraft velocities to derive position and attitude. In practice, first order integration methods (forward Euler) provide sufficient accuracy and stability in the solution of the equations of motion, although care is needed to ensure this approximation does not introduce instability into specific modes of motion.

The majority of the aerodynamic data is defined in terms the aircraft stability axes. For engine data, forces and moments are normally computed with respect to the body axes. Figure 2 shows the organisation of the equations of motion used in real-time simulation. In addition to the computation of the aerodynamic derivatives and the integration of variables, aircraft data is computed in the most appropriate axes and then transformed for computation in other axes[11]. The computation constraints of real-time computing also impact on these axes transformations. The computation of the Euler parameters, the formulation of the Direction Cosine Matrix needed to transform between body axes and the earth frame also require arithmetic operations from the limited real-time budget available per frame.

Weather data derived from an atmospheric model, is defined in terms of the Euler axes. In the solution of the navigation equations, it is possible to transform aircraft motion from a 'flat earth' or an Euler co-ordinate framework to spherical co-ordinates; it is straightforward to compute relative bearing and distances using spherical co-ordinates. More commonly, as the tasks undertaken in an engineering simulator are unlikely to include flight distances exceeding several hundred miles, the navigation system is organised as a set of flat earth 'tiles'. As the aircraft manoeuvres over the surface of the earth, the nearest airfield is selected from the navigation database and aircraft motion is transformed to the runway axis system. A similar transformation is applied to the co-ordinate system of the visual database.

In most simulators, the pilot inputs are acquired as analogue data from the control column, rudder pedals, brakes, flaps, undercarriage lever, engine levers and associated knobs, levers and cockpit switches. For a PC engineering flight simulator, an industrial I/O card was selected, providing 32 analogue and 32 digital channels and 12-bit bit analogue to digital conversion (ADC) with an overall sampling rate of 1600 Hz. The card is programmed to interrupt on completion of each sample. The converted value is stored in a shared buffer and accessed by the flight model software as required. In this mode, autonomous sampling is provided by responding to an interrupt as a background task. After each interrupt, the converted value is stored and the ADC card is set to convert an analogue value for the next of the 32 channels.

There is an overhead associated with each interrupt. The interrupt handler is written in Modula-2. For each interrupt, all the machine registers are saved, the interrupt is processed, the converted value is written to a shared region, the I/O card is reset for the next conversion, the registers are restored and the processor returns from the interrupt, to resume the main task. If the overhead for each interrupt is 50 μs, say, then 1.6 ms is lost per frame on interrupt handling. There is a clear trade-off between latency (the delay in converting an analogue input and using it in the equations of motion) and the interrupt processing overhead. However, this scheme does ensure that there are no delays in waiting for the conversion of analogue data and that an acceptable proportion of the frame is dedicated to data acquisition.

Aircraft Displays

In flight training simulators, aircraft displays are based on replication of the flight deck environment and often use simulated electro-mechanical instruments. The requirement for an engineering simulator is different; the displays should be functionally correct and support an update rate which is matched to the flight model iteration rate. However, the aircraft displays do not need to have a high level of resemblance to specific displays and, for most applications, it is appropriate to emulate aircraft displays by means of real-time computer graphics.

For the PC, although a number of high performance graphics cards are available, it is possible to exploit features of the SVGA for the generation of aircraft displays which are inappropriate to other applications of real-time graphics. An SVGA graphics card affords: frame store management for the video access logic, a range of display resolutions and colours, colour palette selection and video refresh rates in excess of 70 Hz to avoid display flicker. However, there is no hardware support for vector generation, character generation, in-filling of regions, graphics transformations, hidden surface elimination or the clipping of graphical objects.

For aircraft displays, it is necessary to animate the display at 50 Hz by re-drawing objects which include vectors (lines), characters and in-filled regions. In addition, there is a requirement to provide real-time in-fill for instruments such as an attitude indicator and also hidden surface elimination, for example, where an instrument failure flag may drop and obscure a pointer.

A standard SVGA display with a resolution of 640:480 pixels, has a pixel refresh rate of the order 15 million pixels per second at 50 Hz. However, pixel rendering can only be achieved by the processor writing individual pixels into the SVGA memory. In practice, the maximum rendering rate that can be achieved with a standard SVGA display is approximately 2 million pixels per second. This problem is ameliorated by only modifying the display when there is a change to an aircraft display during one frame. For most aircraft instruments, the major part of the instrument is static so that only the pointers or symbols are updated, reducing the overall rendering rate to a manageable level.

A specific set of graphics primitives were written to support vector generation or character generation. For vector generation, straight line segments are generated by means of Bresenham's algorithm[12]. As each pixel of the approximated straight line is computed, it is written to the frame store. Therefore, the limiting speed of vector generation is influenced by the processor performance and the bus interface between the processor and the video memory on the SVGA graphics card.

Boeing 747-200 and Boeing 747-400 displays are shown in Figures 4 and 5 respectively. Figure 6 illustrates the update rate for the worst case situation for the Boeing 747-200 display. The drawing rates for a 33 MHz 486 are shown in Figure 7, giving vector generation rates between 1 and 2μs per pixel. If the average length of lines drawn for an aircraft display is 100 pixels, then only 100 lines can be drawn per frame at 50 Hz, allowing for each line to be erased before it is drawn in a new position. A margin is also required to ensure that the frame time is never exceeded. The graph in Figure 8 shows the distribution of vector lengths for the first 1000 updates of the Boeing 747-200 display. It is clear that the majority of vectors are less than 30 pixels in length giving an average rendering rate of 1.5 μs per pixel.

For character generation, the font for a specific character set is stored and accessed on a bit per pixel basis. There is no requirement to scale the fonts and for the majority of cases, only the visible part of each character is copied. However, a font of character of 16:8 pixels requires approximately 50 μs to render each character in the worst case for a 33 MHz 486. Although there is no arithmetic computation for character generation, time is taken in accessing the appropriate bits of each font character.

In addition to the requirements to render vectors and characters, the attitude indicator imposes considerable processing overhead from the time required to render the blue and brown in-filled regions. Handley and Allerton[13] have shown that the amount of rendering can be reduced significantly by computing the difference between successive frames and in-filling the respective regions as a series of triangles.

One further problem, which is specific to real-time aircraft displays, is that aircraft instruments are 'layered' in the sense that objects of one colour may overlay other objects in the display. For example, the pointer of an altimeter passes over the barometric pressure setting digital read out. For several instruments, there may be three or four levels of object occlusion. However, there is only a limited set of primary colours used in aircraft displays and the priorities of these objects are known. A further problem is that re-drawing objects may interfere with other objects in a display. For example, to reposition a pointer, it is necessary to erase the pointer and re-draw it at a new position. This operation must be performed without erasing underlying objects.

In the SVGA mode, each pixel of the display is represented by a single byte of eight bits. This byte is accessed by the video generation logic and is treated as an eight bit address which maps to 256 possible colours defined in the colour palette. However, the mapping (or encoding) of frame store pixels to a specific colour can be defined by setting the contents of the colour palette registers. Each specific code of the 256 possible values is defined in terms of the red, green and blue colour content, giving a selection of 2^{18} possible colours.

More importantly, this scheme allows the designer to define the mapping between logical colours and physical colours, simply by appropriate initialisation of the colour palette. It is possible to define 8 primary colour layers and allocate these layers to specific bits in the frame store bytes. For example, green might be allocated to the third bit. It is then possible to draw and erase in green by setting and clearing the third bit of each byte representing the pixels of a vector or character. This operation can be performed during vector generation or character generation by means of simple Boolean logic operations in writing to the frame store.

One further benefit of this approach is that it is straightforward to define the resultant colour for overlapping regions, for example, where a pixel is set to green for one object but the same pixel is set to red, say, for another object. The resultant colour (which corresponds to the logical addition of two colours) can be defined to establish the priority of red over green or vice-versa. In addition, this scheme provides extra colours, for example, if red and green never overlap, a third colour, purple say, could be allocated to this bit pattern. Drawing and erasing in purple would then set or clear both bits represented by this colour.

A limited amount of clipping is required in aircraft displays. For example, the pitch lines of an attitude indicator are clipped to the circular bezel of the display or the rolling digits of an engine display are clipped to the window over the digits. While it is possible to clip vectors and characters to irregular regions, this organisation of colour planes lends itself to clipping by overlaying irregular objects over a regular clipping region. For example, the attitude indicator shown in Figure 4 has regions of blue and brown and also white pitch lines clipped to the circular bezel. Clipping to the bezel is achieved by clipping to a rectangular region bounding the blue and brown in-filled regions and the pitch lines and overlaying the grey bezel region such that brown, blue or white lines under the grey region also map to grey. The characters in the airspeed window of the Boeing 747-400 display in Figure 5 are also clipped by this method.

Although SVGA does not strictly support real-time graphics, the features afforded by the graphics organisation of SVGA allow many of the attributes of aircraft displays to be implemented in a straightforward and logical manner. The software developed for the engineering flight simulator has adopted the VESA[14] graphics protocols to provide a high degree of portability for different SVGA cards. The low-level graphics primitives needed for vector generation and character generation were implemented in 8086 assembler. The primitives for colour selection, font access and initialisation of the colour palette were written in Modula-2. For either the 640:480 or the 1024:768 SVGA modes, each pixel is represented by a single byte and the VESA interface provides the necessary control to access the SVGA display pages.

Flight-test Environment

The major purpose of an engineering simulator is to evaluate systems or algorithms, typically for pilot in-the-loop studies. This evaluation phase requires the capture of flight data during flight test experiments and also the recording and display of flight test data. One option is to record only the flight data specific to the test. A more flexible option is to record all the flight data, also allowing data analysis to be undertaken off-line.

During each frame, the flight model software generates a 512 byte packet which contains aerodynamic data, engine data, navigation data and systems data. At this rate, 1.5M bytes of data is generated per minute. For the PC running under DOS, although 640K bytes of memory is allocated for code and data storage, the remainder of the PC memory can be accessed via the Extended Memory

System (XMS) functions. This enables programs to exploit the large memory space available in the current generation of PCs. The XMS functions provide dynamic allocation of memory on a block basis. In addition, XMS functions are accessed via a handle provided by DOS rather than DOS interrupts, which enables XMS functions to be invoked in an interrupt service routine. By aligning the XMS block size with the Ethernet packet size, it is possible to copy Ethernet packets directly to the XMS memory.

As Ethernet packets are received, they are copied to the next available block in XMS memory by the Ethernet interrupt handler as a raw Ethernet packet, i.e. no processing or reformatting is performed on the packet. This approach offers the additional benefit that the disk files can be accessed off-line to emulate flight trials data.

In the flight test environment, flight data packets are available in three forms: the current packet is available during the current frame, recent packets are available by reading the appropriate XMS blocks and thirdly, in order to avoid filling the XMS memory, blocks can be spooled from the XMS region to disk as a background task. This method of flight data recording, using standard PC technology, provides a virtual memory flight data recording capability. A specific frame is accessed from the Ethernet buffer, the XMS memory or disk. No data packets are lost as each packet is written to XMS memory once per frame when the foreground task is interrupted to process each arriving Ethernet packet.

Results

An engineering flight simulator has been developed using three standard 486 PCs to provide real-time simulation with an iteration rate of 50 Hz, giving a 20 ms frame rate and a worst-case margin of 5 ms per frame. With the exception of a few graphics primitives, the software was written in Stony Brook 16-bit Modula-2 running under the DOS operating system. The non optimising version of the compiler was used, with all run-time checks enabled.

Figure 3 shows the overall organisation of the simulator. Each PC is a 66 MHz 486 with a standard NE-2000 compatible Ethernet card. A signal conditioning card was interfaced to an Advantech PCL-812 I/O card. The only non-proprietary card is a programmable sound generation card which replicates a range of aircraft sounds including engines, slipstream, gear rumble, warnings and idents etc.

A number of flight models have been developed for civil and military aircraft including piston-engine aircraft, turbo-prop aircraft and jet aircraft. Full six degree-of-freedom flight models have been developed and fully integrated with a proprietary navigation database and a commercial visual system. The flight models have been validated by flight testing by qualified test pilots and also by comparison with aircraft flight test data.

Figure 9 shows the short period response for a Handley Page Jetstream-100 with Aztazou turbo-prop engines to a manual elevator input of 10 degrees for 1 second. Figure 10 shows the long period phugoid response for a Boeing 747-200 with Pratt and Whitney JT9D engines and Figure 11 shows the Dutch roll response for the same aircraft to a manual rudder input of 30 degrees for 3 seconds (yaw damper engaged). These results were generated directly by the flight test recording module.

One specific application of the engineering simulator has been the design and development of flight control systems[15]. The organisation of the flight model software allows a new model to be developed in a few days. The equations of motion are common to all flight models and the aerodynamic data, which is defined in a consistent form, has to be implemented using table look-up and interpolation from aerodynamic tables. As an example of the efficiency of this approach, Gautrey[16] implemented a full flight model of the Airbus A300-600R in one day and coded an auto-throttle and a C* flight control law in two days. Flight test data was obtained from pilot in-the-loop studies within three days of obtaining the simulator data package for the aircraft.

A range of real-time graphics library primitives have been developed to provide the instrumentation in both standard cockpit configurations and also EFIS displays. The displays cover primary flight displays, navigation displays and engine instruments. By careful exploitation of the SVGA graphics architecture, it is possible to provide displays with sufficient resolution and an update rate of 50 Hz. These library primitives enable displays to be configured in a few hours for specific aircraft. In addition, the modular structure of the displays software allows each display to be tested off-line in terms of functionality and also worst-case real-time performance, prior to installation in the simulator.

The overall timing for the Boeing 747-200 flight model, using the instrument display shown in Figure 4, is given in Figure 6. This confirms that the flight model and engine model occupies less than 1 ms of each frame. Generally, the instrument displays occupy 3 ms to 5 ms of each frame. The occasional excursions to an overall processing time of 14 ms is caused by the compass cards of the HSI and the RMI instruments. The worst-case update rate for this display, using a 66 MHz Pentium processor with an ATI Mach32 SVGA card, was recorded at 140 Hz.

Conclusions

This paper has shown that it is possible to implement a real-time flight simulator, based on several standard PCs coupled with Ethernet and to obtain the same level of fidelity as simulators developed using high performance mini-computers. The simulator has an update rate in excess of 50 Hz using commercial off-the-shelf PC equipment.

SVGA provides sufficient resolution to implement real-time displays of aircraft instrument displays including EFIS displays. In order to achieve this performance, a real-time graphics library was developed which is optimised for SVGA. In addition, the problems of updating displays, colour plane organisation and colour prioritisation are solved by mapping the logical colour plane organisation using the SVGA colour palette.

The PC, combined with a low-level Ethernet interface, also provides a real-time flight test environment to acquire simulator data from trials at rates up to 1.5M bytes per minute. Data is streamed to XMS memory and copied to disk without loss of data providing a high volume and high throughput data recording capability with standard PC equipment.

The engineering simulator has been validated by a number of flight trials and performance tests to confirm the overall processing. All this has been achieved using software written in Modula-2 for the PC running under DOS. Further improvements in real-time performance can be achieved in several ways:

- the use of Ethernet allows the system to be partitioned to balance the processor loading, allocating modules to additional processors without a major reorganisation of the simulator software
- at present, the simulator is based on standard 66 MHz PCs. A factor of at least five would be achieved by moving to Pentium processors
- all the Modula-2 software has been developed using a non-optimising 16-bit compiler with run-time checks invoked. A further factor of two to three is possible by using an optimising compiler and suppressing run-time checks.

The simulator provides the basis of a real-time engineering simulator using an off-the-shelf PC implementation with Ethernet and SVGA displays. In this form, it is straightforward to develop new models and to evaluate flight control systems, avionics, displays and navigation laws by means of the high performance data recording facility. The modular structure of the simulator allows the software to be used directly for real-time applications and to be re-used for off-line system development.

References

1. International Standards for the Qualification of Airplane Flight Simulators, The Royal Aeronautical Society, July 1996.
2. Allerton D J, Avionics, Systems Design and Simulation, *The Aeronautical Journal*, Vol. 100, No. 1000, pp. 439-448, October 1996.
3. Allen L, Evolution of Flight Simulation, *AIAA Conf. Flight Simulation and Technologies*, AIAA-93-3545, August 1993.
4. Katz A, Allen D M and Dickson J S, Synchronisation and Time Tagging in Distributed Real-time Simulation, *AIAA Conf. Flight Simulation Technologies*, AIAA-89-3300, p259, 1989.
5. Valentino G J, Shared-memory Networks (SMNs): A Key Enabling Technology for Distributed Real-time Systems, *AIAA Conf. Flight Simulation Technologies*, AIAA-94-3427-CP, 1994.
6. Ethernet PC/TCP Packet Driver Specification Version 1.09, FTP Software Inc., Sept. 1989.
7. Halsall F, Introduction to Data Communications and Computer Networks, Addison-Wesley, 1985.
8. Rolfe J M and Staples K J, Flight Simulation, Cambridge University Press, 1986.
9. Smetana F, Computer Assisted Analysis of Aircraft Performance, Stability and Control, McGraw-Hill, 1984.
10. Sinnett M K, Steck J E, Selberg B P and Oetting R B, An Alternate Approach to Table Look-up Routines for Real-time Digital Flight Simulation, *AIAA Conf. Flight Simulation Technologies*, AIAA-89-3310, p336, 1989.
11. Baarspul M, Proc. A Review of Flight Simulation Techniques, *Progress in Aerospace Sciences*, Vol. 22, pp. 1-20, Pergamon Press, 1990.
12. Bresenham J, E, Algorithm for Computer Control of a Digital Plotter, *IBM Systems Journal*, 4(1), pp. 25-30, 1965.
13. Handley S J and Allerton D J, The Design of an EFIS Attitude Indicator for a Flight Simulator, *19th ICAS Congress*, AIAA, Anaheim, pp. 282-290, September 1994.
14. VESA BIOS Extension (VBE) Core Functions Standard, Version 2.0, *Video Electronics Standards Association*, November 1994.
15. Green B, ADA Autocoding and Object-based Software Design - An Application for Real-time Flight Controls Simulation, *AIAA Conf. Flight Simulation Technologies*, AIAA-94-3408-CP, 1994.
16. Gautrey J, Robust Flight Control Design in a Computational Control Aircraft Engineering Environment, *5th Garteur Flight Mechanics Action Group FM(AG08)*, Toulouse, November 1995.

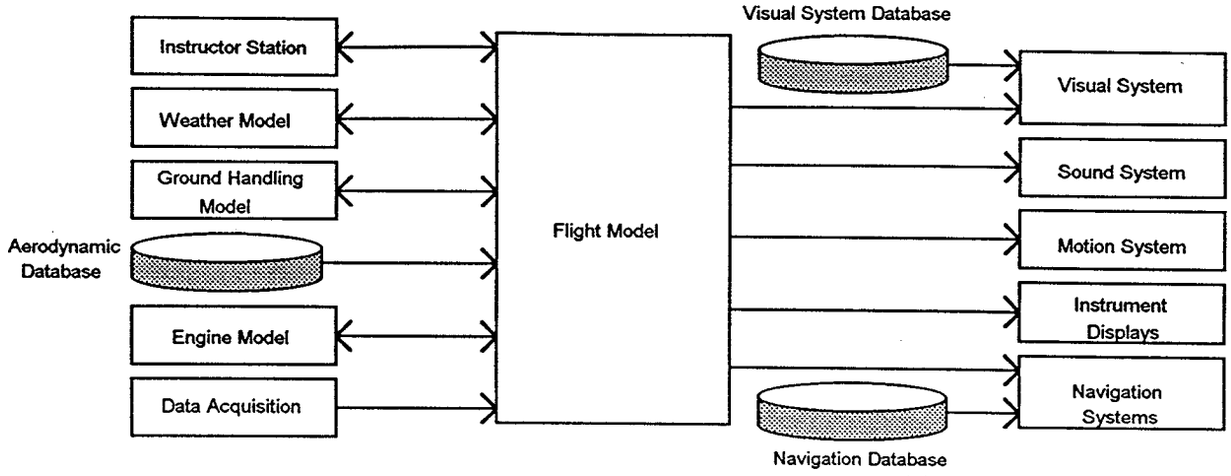

Figure 1 Simulator Organisation

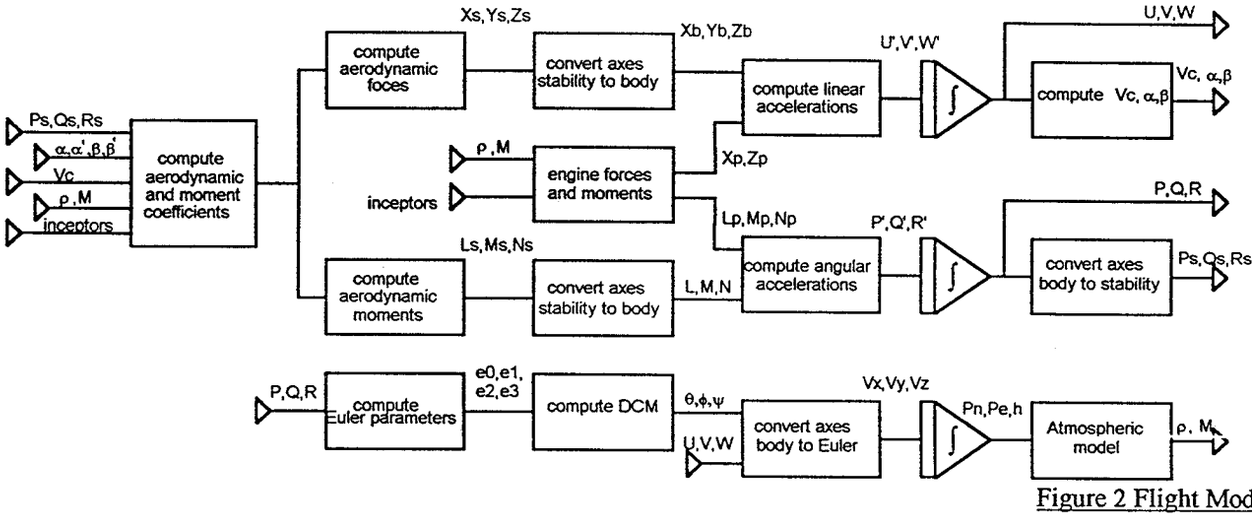

Figure 2 Flight Model

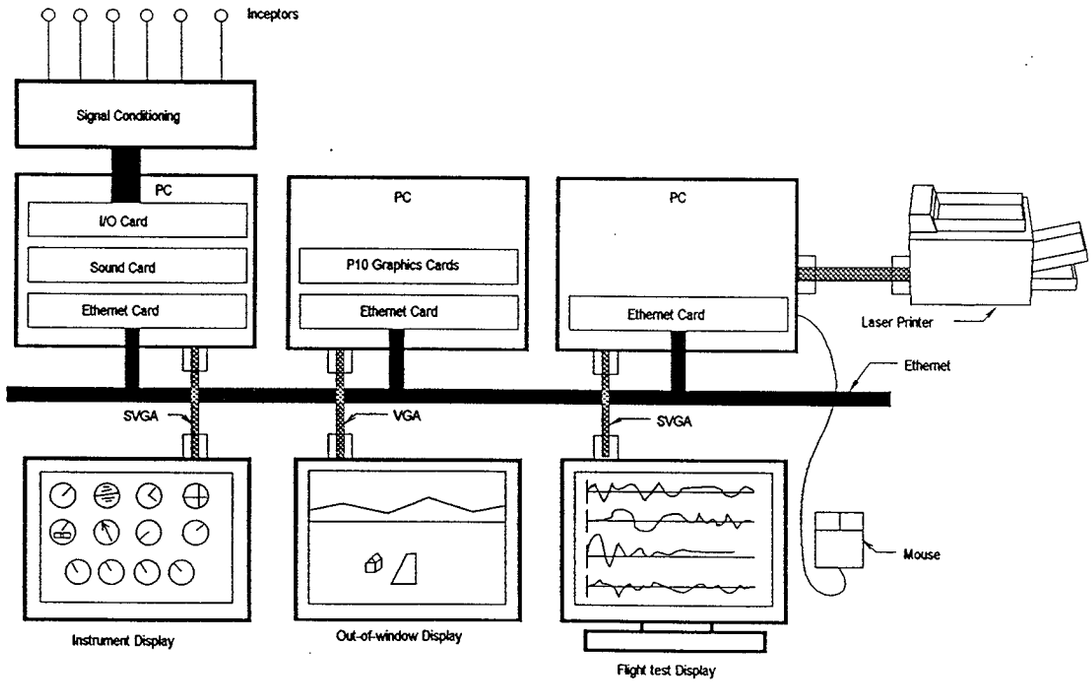

Figure 3 System Architecture

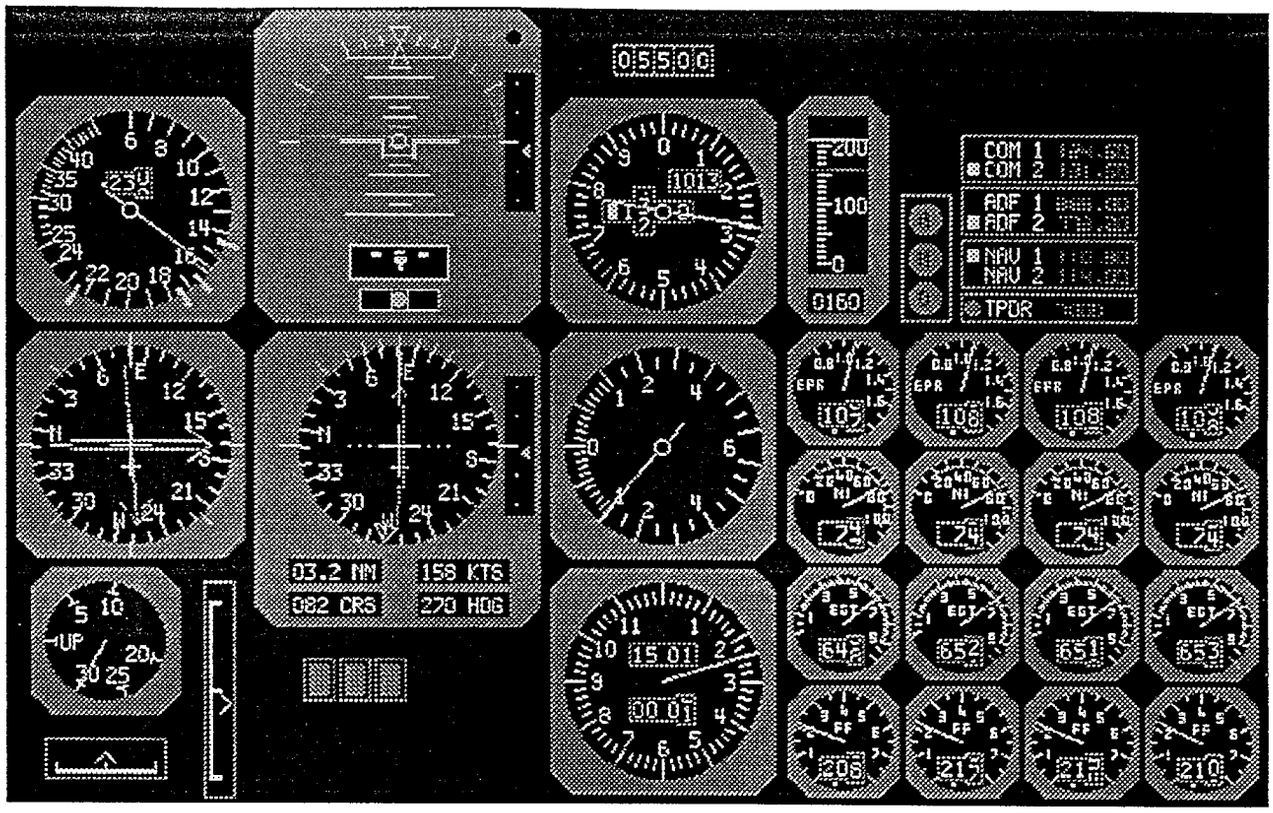

Figure 4 Boeing 747-200 Display

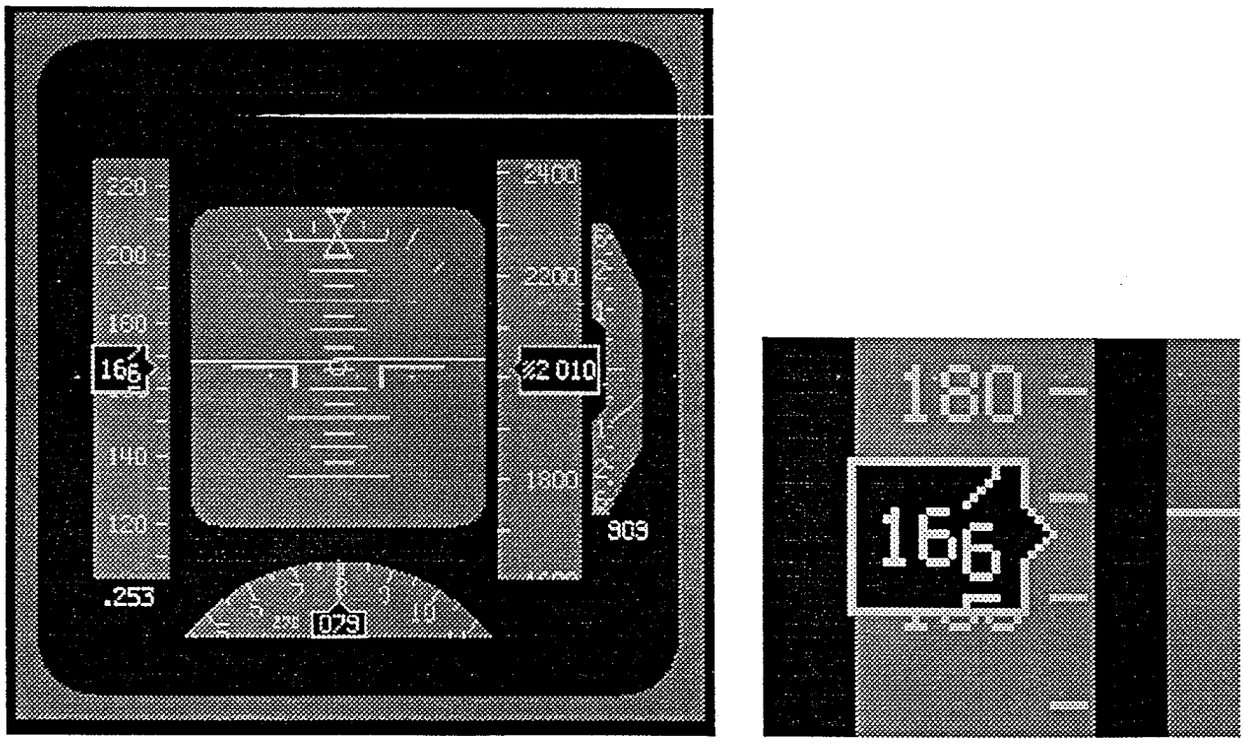

Figure 5 Boeing 747-400 Display

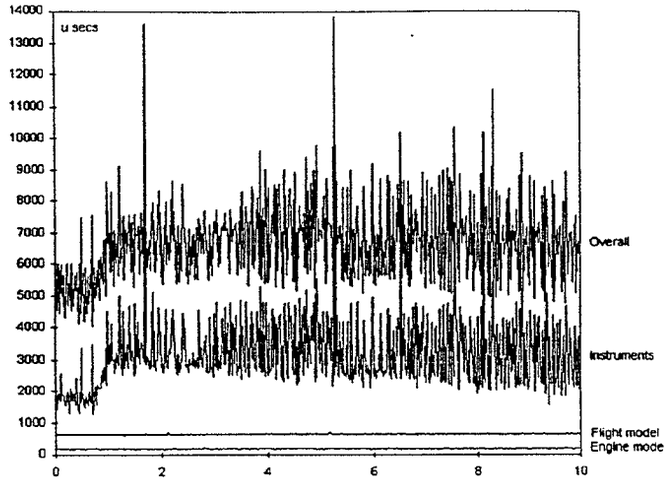

Figure 6 System Timing

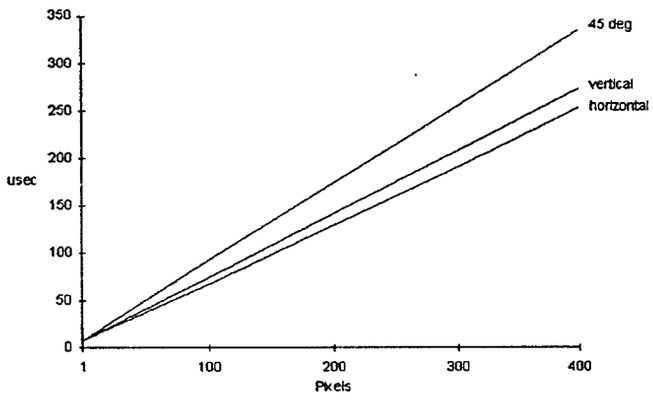

Figure 7 Pixel Drawing Rate

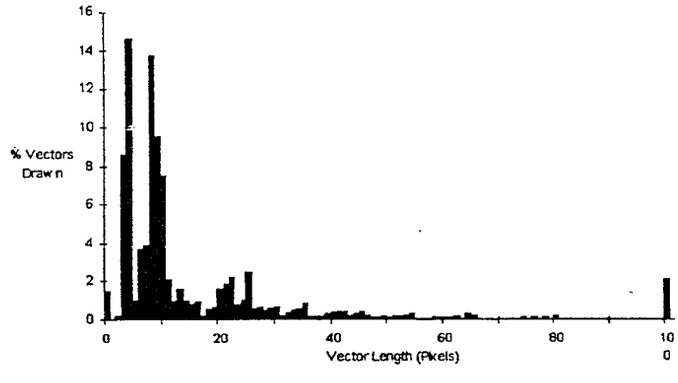

Figure 8 Distribution of Vector Lengths

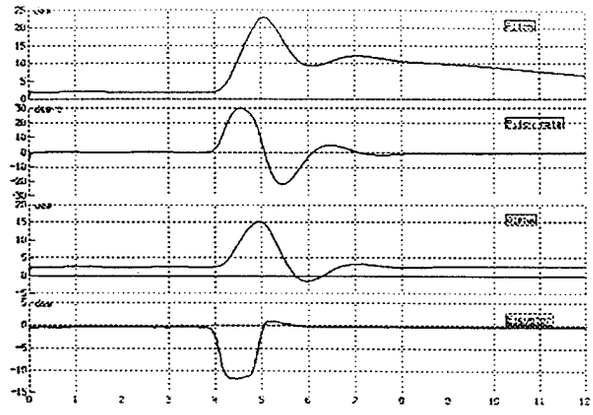

Figure 9 Jetstream-100 Short Period Phugoid Response

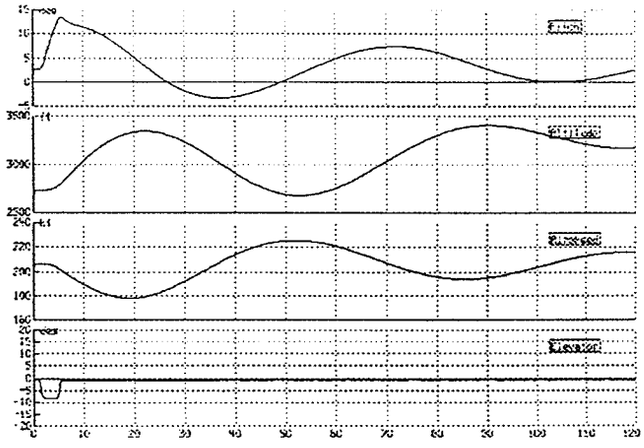

Figure 10 Boeing 747 Long Period Phugoid Response

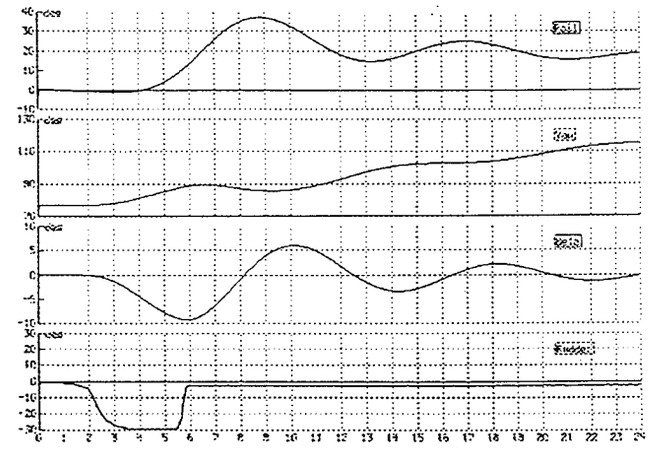

Figure 11 Boeing 747-200 Dutch Roll Response