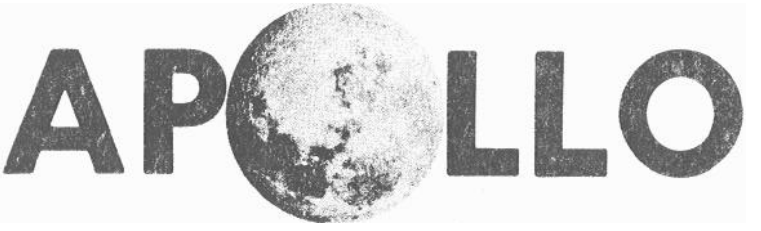
TA165
.M41
.I591

MASSACHUSETTS INSTITUTE OF TECHNOLOGY



GUIDANCE, NAVIGATION AND CONTROL

E-2475

A COMPREHENSIVE DIGITAL SIMULATION FOR THE VERIFICATION OF APOLLO FLIGHT SOFTWARE

by

F. K. Glick
S. R. Femino

JANUARY 1970

MIT CHARLES STARK DRAPER LABORATORY

CAMBRIDGE MASSACHUSETTS 02139

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE, NAVIGATION AND CONTROL

Approved: J. H. Laning Jr. Date: 2/3/70
J. H. LANING, JR, DIRECTOR, DIGITAL COMPUTATION
CHARLES STARK DRAPER LABORATORY

Approved: D. G. Hoag Date: 4 Feb 70
D. G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: Ralph R. Ragan Date: 4 Feb 70
R. R. RAGAN, DEPUTY DIRECTOR
CHARLES STARK DRAPER LABORATORY

to be presented at AIAA 8th
Aerospace Sciences Meeting,
New York, New York
January 21, 1970

E-2475

A COMPREHENSIVE DIGITAL SIMULATION FOR THE VERIFICATION OF APOLLO FLIGHT SOFTWARE

by

F.K. Glick
S.R. Femino

JANUARY 1970

MIT CHARLES STARK DRAPER LABORATORY
CAMBRIDGE. MASSACHUSETTS, 02139

ACKNOWLEDGEMENT

This report was prepared under DSR Project 55-23870, sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NAS 9-4065 with the Charles Stark Draper Laboratory, Massachusetts Institute of Technology, Cambridge, Mass.

The publication of this report does not constitute approval by the National Aeronautics and Space Administration of the findings or the conclusions contained therein. It is published only for the exchange and stimulation of ideas.

E-2475

A COMPREHENSIVE DIGITAL SIMULATION FOR THE VERIFICATION OF APOLLO FLIGHT SOFTWARE

ABSTRACT

The flight software for the on-board Apollo Primary Guidance, Navigation and Control System provides attitude and trajectory control, recursive navigation, targetting, inertial subsystem moding and alignment, prelaunch checkout, crew interaction and telemetry processing for each spacecraft. To develop and flight-qualify this complex software package, an all-digital, instruction-by-instruction simulation of the on-board computer's operation has been implemented, along with a powerful set of software diagnostics. This computer simulation is coupled to a comprehensive digital simulation of the computer's environment, including the sensor systems. The simulator is described and design considerations, advantages and limitations are discussed. A simulated powered flight maneuver is compared with telemetry obtained during the actual performance of the maneuver in flight. The telemetry data revealed a potentially significant effect which had not been modeled. As a result the environment simulation was upgraded to more accurately represent the flight environment.

By: F.K. Glick
S.R. Femino
January 1970

634477

I. Introduction

The central component of the on-board Primary Guidance, Navigation and Control Systems for Apollo spacecraft is the Apollo Guidance Computer (AGC).[1] Two nearly identical computers are used per mission, one each in the command and lunar modules. The task of producing and verifying the flight software for these machines was quite formidable, requiring an average of sixty man-years per AGC program.[2] The problem was not simply one of guaranteeing that the multiprogrammed flight software possessed an internal integrity; the problem was also to develop confidence that the AGC programs would properly guide, navigate and control many configurations of untried spacecraft. One of the more effective tools developed to accomplish this task, the Apollo All-Digital Simulator, is described in this paper.

II. The Apollo Guidance Computer

It is first appropriate to briefly describe the Apollo Guidance Computer (AGC). This processor is a general purpose, sequential, digital machine. Its word length is 16 bits including parity. The random access memory consists of 2048 words of destructive, read-write memory, called erasable, and 36,864 words of non-destructive read-only memory called fixed memory. The memory cycle time (MCT) is slightly less than 12 microseconds, with two MCTs required to execute an add and four MCTs to execute a multiply. As is typical of real-time control computers, this machine has a set of special input-output channels with which it controls the spacecraft and observes the state of its environment. The interrupt structure consists of ten program interrupts with associated priorities, which are used for program control transfers. In addition, twenty-six counter interrupts with associated priorities allow for input-output servicing. The instruction set consists of forty-two regular instructions and nine involuntary instructions. Figure 1 indicates the number and diversity of systems with which the AGC interacts.

III. Simulator Design

A number of factors associated with the Apollo guidance, navigation and control problem indicated that an all-digital simulation of the on-board processors would be a useful tool. First, the major flight software developments required would proceed in parallel with the hardware development, which made it improbable that all the interacting systems would be available before the qualification and delivery of the flight software. Furthermore, other systems with which the software interacts were rapidly evolving and most were unavailable even in preproduction configurations

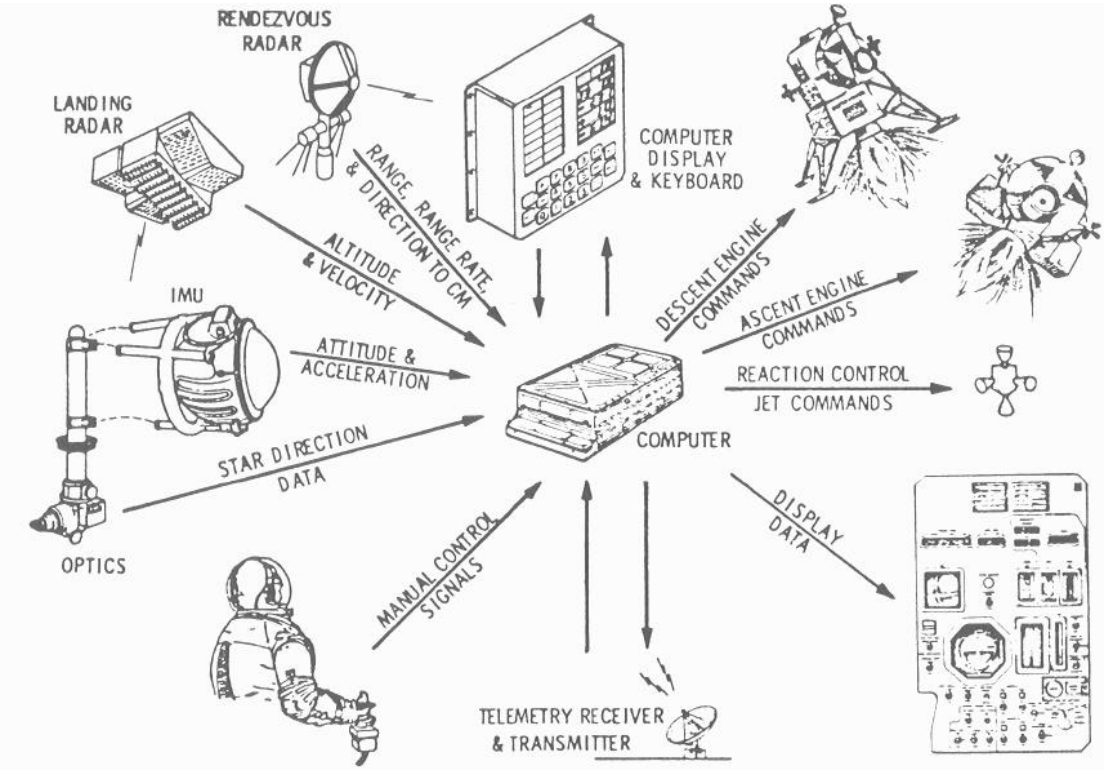


Fig. 1 The Lunar Module Computer Interfaces

for use in software verification. Also, the fixed memory feature of the AGC required that the flight software qualification must be completed prior to the assembly of the flight memory. Thus, some type of model was required. Another factor, which made instruction-by-instruction simulation on a high speed, ground-based computer practical, was the relatively long memory cycle time of the AGC. Also, the fact that many nominal environment conditions such as zero-g are difficult or impossible to reproduce in the laboratory, or are impractical to reproduce for large scale software testing, indicated the desirability of an all-digital simulation.

Other factors were related to the flight software itself. First, the software development staff would he working in parallel on two independent programs per mission, one for the command module computer and one for the lunar module computer. Each of these programs would be developed within exceedingly tight schedule constraints. Because of the parallel nature of these efforts it was desirable to have facilities which could operate in a parallel mode and he easily reconfigured as testing requirements changed. Furthermore, it was desirable to have a facility which could operate with a minimum of working flight software so that programmers could operate and unit-test small modules without a fully checked out AGC program in which to imbed their modules. Finally, it was obvious that a large amount of

time would be consumed in debugging the flight software. This dictated that the potential exist for increasingly powerful diagnostics. Both on-line and post-run editing and analysis of results were essential. The capability to restart any simulation from an intermediate point was very desirable. Coupled with this capability, a guarantee of absolute reproducibility of an earlier simulation was very attractive. Absolute reproducibility is very difficult to provide, but experience has shown that it is extremely useful.

Simulator Design Principles

The above considerations led to the following set of design principles and guidelines for the development of the Apollo All-Digital Simulator. They are listed in the order of their influence on the ultimate simulator configuration.

1. The purpose of the Apollo All-Digital Simulator would be to test the integrity and performance of the flight software, not to verify spacecraft performance or crew procedures.
2. No assumptions would be made about the flight software and its operation except that it was intended to operate on the AGC. The hardware systems would be simulated with no consideration given to the intent of the flight programs. Data collection and modeling should be carried out independently of similar activities for the flight software.
3. Instruction processing would be identical to the processing in the flight computer. If not, the differences would be well understood. Obvious exceptions would be made to provide certain diagnostic capabilites.
4. The guidance computer environment would be virtually the same as the real computer's environment at the interface of the computer with that environment. If not, the differences would be within the quantization levels of the analog-to-digital convertors.
5. All simulations would be bit-for-bit reproducible in both the guidance computer simulation and the environment.
6. Certain randomness should be introduced by default if exact reproducibility was not specifically desired. This would encourage testing of different program branching and timing.
7. Anomalous behavior of the flight software should be made readily apparent.
8. The simulator would provide accurate machine duty cycle information.
9. The simulator would consist of a single set of routines which would provide all the capabilities required by all users. This would simplify simulator support and configuration control.
10. The general run-time efficiency of the simulator would be such as to allow exhaustive testing of the flight software.

Simulator Structure

The resulting all-digital simulator is presently operating on an IBM Model 360/75 with one-million bytes of core. The system configuration is shown schematically in Fig. 2. The simulator operates entirely within the confines of this structure. No special purpose devices or modifications to the host machine are required. The host machine also serves as a general purpose computation facility.

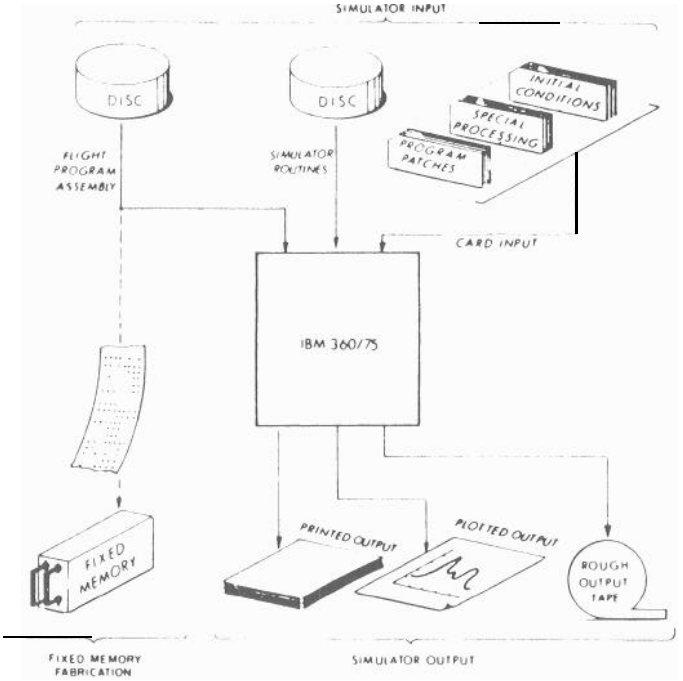


Fig. 2 Simulation system schematic

The flight program assembly which the simulator is to execute is specified by the user and obtained from a library of current programs kept on a disc file. It is important to note that the punched tape used to fabricate the fixed memory is obtained automatically from the same source. Thus the simulator will execute the same code as will be implemented by the read-only memory.

A rough output tape is identified with each simulation. This tape is used for recording snapshots of the complete state of the simulation. These snapshots, taken at selected intervals, contain sufficient information to restart a simulation and proceed in a manner bit-for-bit identical to the previous simulation. Data for post-run editing is also stored on this tape. In addition, printed and plotted output can be requested.

The structure of the simulator is shown in Fig. 3. The instruction simulator is a routine coded in 360 Basic Assembly Language. This routine simulates the memory and instruction processing of the flight computer. It is coupled to its environment through an interface routine, the COMMUKIC ATOR. This routine filters communication between the simulated guidance computer and the simulated environment. The intent of the routine is to minimize control transfers from the instruction simulation to the environment simulation.

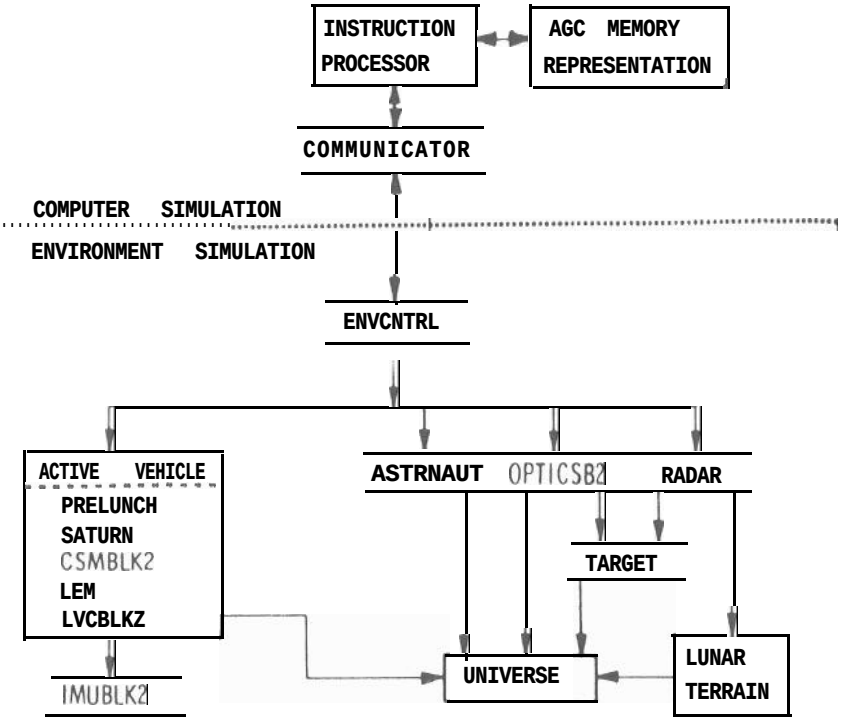


Fig. 3 Simulator routine structure

The environment is coded in an algebraic compiler language called MAC.[3] The environment will be discussed in later sections. First the instruction simulator will be described.

IV. The Instruction Simulator

As Fig. 2 indicates, the simulator operates on three types of card input: flight program patches, requests for special processing and initial condition specifications.([4]) These are processed during the loader phase of the simulation. Since the simulator has access to the assembly's symbol table, the patching can be accomplished symbolically rather than by direct address.

With these inputs the loader completes its representation of the AGC memory. Each AGC word is represented by a full IBM 360 word of four bytes. The right half-word either contains the AGC word corresponding to the given AGC location or data sufficient to find the AGC word. The left half-word contains the branch address of the code required to process the instruction.

After the initialization of the environment (described below), the simulator enters its execution phase. During this phase the instruction simulator sequences through the instructions of the flight program. The sequence of operations at each instruction begins with a check for possible AGC interrupts. If an interrupt has occurred the simulator stores those registers saved by the AGC hardware and branches to the appropriate interrupt processor in the flight program. If no interrupt is pending, the simulator tests the need for special processing of this AGC instruction. The special processing could be one or more of a large class of special requests which were attached to this location by the loader. The loader attaches flags to a location to trigger special processing in response to a user-supplied instruction to do so via the control cards mentioned above. Special processing in this context includes the following functions:

1. The generation of a line of AGC program instruction trace.
2. The dumping of user-specified registers of memory.
3. The dumping of environment state variables.
4. The change of any AGC register to a user-specified value.
5. The generation of an on-line message that the particular instruction has been executed and the AGC time of its execution.
6. The initiation of an involuntary AGC program restart.
7. The recording of a snapshot of the simulator state from which a subsequent simulation can be initiated.
8. A check for the accessing of an unused AGC memory location.
9. Run termination.

These functions can be initiated by the accessing of any memory location to which they are attached. Furthermore, they can be made conditional upon the number of accesses of the location to which they are attached or to the contents of any AGC register. If special processing is required, a branch is made based on the branch address stored with the AGC memory representation.

When all the processing required is accomplished, the simulator checks for pre-instruction overflow and executes a set of host machine instructions which leave the simulated AGC special registers and memory in the state they would have after

the execution of the instruction in the actual AGC. Then certain post-execution processing is completed, including the completion of the line of trace if trace is active, post-execution overflow checks, and the updating of the AGC clock with the number of memory cycles appropriate for the instruction just executed.

The simulator contains an ordered sequence of time-initiated events. These events include time-triggered special processing or time-stops set by the environment simulation. After the AGC clock is incremented, the time associated with the first event is tested against the AGC clock. If the event is due, it is initiated at this time and the remaining events are pushed up in the stack. The simulator then proceeds to the next AGC instruction. Where a trade-off exists between speed in processing AGC instructions and speed in processing special requests, instruction processing is made speedier at the expense of infrequently occurring special processing. That is, the simulator is optimized for those instructions which require no special processing.

In this way, the instruction simulator continues through the flight program instructions until one of two events occur. One event is the incrementing of AGC time beyond a time-stop set by the environment portion of the simulator. At this point, control is passed to the environment for whatever reason control is required there.

The second event is the execution of any RGC instruction which accesses a sensor input register. At this point, the contents of that register must be updated by a valid simulated input for that AGC time. To accomplish this, the instruction simulator relinquishes control to the COMMUNICATOR. The COMMUNICATOR supplies the sensor input, updated to the proper AGC time with an extrapolation of data provided by the environment at its last update. This data is called a recipe. Associated with each recipe is the AGC time at which it was generated and the AGC time at which the extrapolation may be in error by one quantum in the input register. This is the expiration time of that data. Also associated with each recipe is an invalidation flag set by the instruction simulator. The simulator sets these flags whenever the AGC program issues a command to the environment which places the validity of the recipe in question. For example, the turning on of a reaction control jet would invalidate the vehicle attitude extrapolation. These invalidation flags are reset after the environment has had an opportunity to act on the commands and include their effect in the extrapolation data or expiration time. Before supplying the input register contents from the recipe, the COMMUNICATOR determines if the data is valid. If either the invalidation flag is set or the expiration time has passed, the COMMUNICATOR calls the environment to update to the present AGC time and refresh the recipes.

Similarly, after its update, when control returns from the environment to the COMMUNICATOR, the newly-refreshed data is used to provide the input register contents. With the input register contents updated to the AGC time, control is returned to the instruction simulator. Processing of instructions then resumes using the sensor input. It continues until the next accessing of a sensor input register by the AGC program. This structure assures that all sensor inputs to the flight program are current at the time they are accessed and that they incorporate the effect of all previous AGC output activity. It also avoids unnecessary updating of sensor inputs which are not being used.

Several features of the instruction simulation bear reiteration. The flight program code is executed directly; there is no modeling of the AGC program. Special processing can be initiated at any program instruction or memory access. The entire simulation is paced by AGC time. All input is valid at the time it is read by the flight software but superfluous environment updates are avoided.

V. The Environment Simulation

The environment portion of the Apollo All-Digital Simulator has two functions. First, it provides all AGC sensor and discrete input, including astronaut interaction. Second, it maintains a standard against which the flight software performance can be judged. The use of the simulated environment as a standard warrants elaboration.

Many of the tasks required of the flight software involve the estimation of the state of its physical environment. Quantities which the AGC programs estimate include spacecraft attitude and trajectory, target vehicle trajectory, gravity, sensor errors and lines of sight tovarious landmarks or celestial bodies. The measurements made and processing done to refine these estimates are in general imperfect. The imperfections can arise from numerous sources: the processes being measured often only indirectly reflect the quantities being estimated; the sensor is, in general, not perfect; the finite precision available to represent data introduces errors; and finally, programming errors can lead to subtle or gross errors in an estimate.

All of the above error-sources are present in the All-Digital Simulator. However, the environment portion of the simulator has available or can generate the “true” value of the quantity being measured and the “true” value of the quantity being estimated. For example, the environment can compute the “true” altitude of the lunar module above the simulated lunar surface. This altitude can be compared with the AGC -estimated altitude. While it is the case that the “true” quantities are represented in the environment simulation with finite precision mathematical models,

the precision is far greater than the AGC provides; and the models, which are discussed below, are more comprehensive than those used in the flight programs. Thus the "true" environment quantities provide a standard against which to compare AGC estimates made by the flight software.

There is a second sense in which the environment provides a standard against which to judge AGC program performance. The environment monitors and reacts to the output from the AGC program. The environment tests this output against constraints imposed by the systems to which that output is directed. When critical out-of-specification behavior is detected the fact is flagged for subsequent diagnosis and correction.

The environment consists of a controlling routine and a set of model routines. Their organization is shown in Fig. 3. The controlling routine (ENVCNTRL) is responsible for updating all relevant routines in the proper sequence and preventing unnecessary or redundant computation.

The active vehicle routine simulates the spacecraft whose flight program is being executed by the instruction simulator. It simulates all spacecraft systems influencing the flight software. It provides the attitude and trajectory perturbations caused by contact forces and moments acting on the vehicle. Since the spacecraft configurations are quite distinct, there are five active vehicle routines from which the proper routine is selected: a prelaunch routine (PRELUNCH), a launch vehicle routine (SATURN), a command and service module routine (CSMBLK2), a lunar module routine (LEM) and an entry vehicle routine (LVCBLK2).

The inertial measurement unit is simulated by a separate routine (IMUBLK2). This routine uses the attitude and trajectory profile computed by the active vehicle routine to simulate the stable platform's gimbal motion and sensed acceleration. After a simulated analog-to-digital conversion, these become sensor inputs to the AGC program.

The astronaut routine (ASTRNAUT) simulates the interaction of the crew with the AGC. This includes such activity as datainput via a simulated keyboard interface, decision making and branching based on AGC output displays, taking of navigation sightings on simulated celestial bodies and manual maneuvering of the spacecraft. These activities are controlled by statements input to the simulator at the beginning of the run and there is no on-line human interaction with the simulation.

The physical phenomena external to the spacecraft, such as gravity and atmospheric effects, are provided by a single routine (UNIVERSE). This routine also performs the free-fall trajectory integration. It incorporates at intervals the contact-force-generated trajectory perturbations to generate the active vehicle state vector.

The passive vehicle routine (TARGET) integrates the free-fall trajectory of the vehicle whose computer is not simulated in the given simulation. This state vector is required to compute simulated optical and radar measurements made on that vehicle. A common gravity routine is used for the active and passive vehicle trajectory integration.

The optical system routine (OPTICSB2) simulates the optical lines of sight of the scanning telescope and sextant. These devices operate under computer control and provide sensor inputs to the AGC program.

The radar systems routine (RADAR) simulates the interaction of the AGC with the rendezvous and landing radars of the lunar module or the VHF ranging system of the command module. These systems provide numerous sensor inputs to the flight software and are, in turn, controlled by the flight software.

The routine which simulates the deviation of the lunar surface from a smooth sphere (LUNAR.TERRAIN) is used by the landing radar model to compute altitude measurements for the AGC. The routine is also used to define the altitude of the surface at lunar touchdown, since in general the exact point of touchdown is not known prior to the simulation. This guarantees that the measured altitude is consistent with the altitude at which the vehicle lands.

This environment simulation has an initialization phase during which it processes input data supplied by the user. This phase establishes the initial conditions from which the environment state will evolve. At this time the user selects and tailors the models to test specific AGC program routines or features. Nearly all environment statevariables have analogous quantities in the AGC memory. In general these analogous quantities will not be identical because of the desire or necessity of simulating initial condition errors. Each environment routine is called during the initialization phase. The initial state is computed and discrete inputs to the instruction simulator are set. The environment is then ready for its execution phase and control returns to the instruction simulator.

As indicated above, the AGC instruction simulator proceeds instruction-by-instruction through the flight program until it arrives at a time-stop set by the environment or an instruction which accesses a sensor input register. The environment's time and state remain as they were at the last call from the instruction simulator. As the instruction simulator proceeds through the program, it will, in general, set commands to the environment. For example, it may turn on a reaction control jet or turn off an engine gimbal actuator. Since the effect of these commands is not sensible to the AGC program until it again accesses a sensor, the instruction simulator can proceed asynchronously some distance ahead of the environment in time. The only consideration given the environment when commands are generated is that the invalidation flags described above are set for all sensors whose recipe may be invalidated by the command being issued.

All of the output activity or commands generated by the AGC program are buffered with the time of their generation until the time of the next environment update. When a sensor input is required and the sensor recipe has been invalidated by an AGC command or its expiration time, control is transferred to the environment. The environment must thenupdate its time and state to the time at which the sensor input is required. In so doing, the environment carefully incorporates the effects of all the commands set by the flight program.

The ENVCNTRL routine causes each environment routine to be called in turn to update to the present AGC time. Each routine reads the list of buffered commands and incorporates each command relevant to it at the proper time. When it reaches the time at which the sensor input is required, it files the recipe data and an expiration time for each recipe. Control then returns to the ENVCNTRL routine which calls any other routine for which a buffered command existed or whose sensor input was required.

In the course of these updates any routine may determine that an input should be made to the AGC program at some future time. For example, the IMUBLK2 routine may wish to set a gyro fail indication in the future. To do this, the routine desiring to stop the instruction simulator simply files a time-stop. These stops are ordered and the lowest future time is passed to the instruction simulator and becomes its time-stop for a return to the environment. This procedure allows environment-initiated inputs to be made available to the AGC program at the proper time. No assumptions are made about how or when the AGC program processes its input.

When all such routines have been updated, the environment is said to have updated to AGC time and control is returned to the instruction simulator. The instruction simulator then accesses the sensor input which caused the environment to be updated and receives an input which is valid at the time it was accessed. At this point, AGC instruction processing commences again and the procedure is repeated.

Modeling Philosophy

It is appropriate to describe the modeling philosophy and some of the models which have evolved for this simulator. As indicated above, a key role is played by the sensor devices. By sensor device, we mean any device which observes or measures the state of the environment and inputs data to the flight program. This definition is intended to include the crew as a sensor. All new information which becomes available to the AGC program after the initialization of the AGC memory must pass through a sensor. The sensors thus filter all inputs to the AGC program.

In modeling physical phenomena we could deterministically simulate the phenomena by integrating the equations of motion of the process and passing the resulting response through the sensor. Alternately we could input to the sensor, signals which are statistically similar to those generated by the physical process. The random processes generated in this way do not require integration of the equations of motion governing the process. Both methods of modeling are used in the Apollo All-Digital Simulator. Basically low frequency processes are simulated by numerical integration of their differential equations of motion. The effect of propellant slosh is simulated by direct integration of two second-order equations for each tank. Higher frequency processes and processes whose forcing functions are not explicitly known or are not directly controlled by the AGC program are simulated by random processes. These processes are generated by sequences of random numbers. In some cases these sequences are passed through shaping filters to introduce correlation from sample to sample. In other cases the samples used are essentially uncorrelated. For example, the mechanical vibration of the spacecraft structure at frequencies higher than approximately six hertz is simulated by passing an uncorrelated random sequence through a second-order difference equation. The resulting random process is input directly to the analog-to-digital convertors used to sense body attitude.

Since the guidance computer is a digital machine of finite word length, there is inevitably a quantum level below which the AGC program cannot represent input data. This quantum level, in the case of each sensor, is taken as a bound on the accuracy required in the environment.

Environment Models

We will now describe some of the models used in the environment routines. The active vehicle routines are designed to be a skeleton framework for calculating the vehicle response to contact accelerations. Each vehicle routine is independent of any specific mission requirements or data. This facility is obtained by storing all mission-specific data in data files. The data files typically contain about 1000 data words, including such parameters as mass property tables, engine coordinates, time delays, thrust and ISP tables, slosh parameters, and bending coefficients. A data file from which the vehicle program reads the necessary parameters is selected when the simulation is initiated. This method allows different missions or off-nominal testing to be simulated by simply changing the data file. The use of one routine to model a vehicle for all missions represents a substantial savings in man-hours required for updating and maintenance.

As indicated above, the active vehicle routine is responsible for simulating the motion of the spacecraft sensor station in response to contact forces and moments applied to thevehicle. The models required to do this are different for a prelaunch, Earth-fixed mode than for a flight mode. In the planetary-fixed cases the spacecraft is rotated with respect to an inertial frame at Earth rate. The sensed accelerations due to centripetal and gravitational accelerations are computed. Gravity in this case includes the non-spherical terms of the Earth's field and the attractions of the Sun and Moon. In addition, the launch vehicle centerline can be displaced from local vertical to simulate the effect of a mean wind deflection or asymmetric heating of the launch vehicle. A time-varying deflection of the launch vehicle simulates zero-mean, wind-induced sway of the booster. The launch vehicle response to this forcing function is that of a damped second-order linear system. The random forcing function is generated by passing a piecewise-constant random function whose amplitude from sample to sample is normally distributed, through a first-order linear system. The time constant of this system is chosen so that the correlation of the output is similar to that found in wind gust distributions. Two independent processes are generated, one for each horizontal direction.

In the flight mode the active vehicle routine uses significantly different models. The six-dimensional rigid body equations of motion are numerically integrated. The driving terms for these equations arise from several sources. During atmospheric flight, the aerodynamic forces and moments are simulated. The UNIVERSE routine contains a representation of the atmosphere from which it computes dynamic pressure, dynamic pressure rate and Mach number. The active vehicle routine computes the appropriate aerodynamic coefficients. In the case of

the entryvehicle these coefficients are stored internally at specific angles of attack and Mach number. A parabolic interpolation is made to obtain coefficients at intermediate angles of attack and Mach numbers. These then allow the computation of aerodynamic forces and moments on the vehicle.

If the simulated spacecraft is outside the atmosphere, it will normally experience free-fall conditions punctuated by short periods of thrusting flight. In this context thrusting includes all maneuvers which cause the trajectory to deviate from a free-fall trajectory; that is, attitude control thrusting as well as trajectory control thrusting.

The reaction control system is used to provide contact acceleration for controlling vehicle attitude. This system of thrusters is normally controlled by the AGC digital autopilot. The environment responds to individual thruster on-off commands. It models each jet as a constant thrust motor. The application and removal of thrust are delayed from the simulated electrical on-off signals originating in the AGC. The lengths of these delays are chosen to match the build-up and tail-off transients typical of these thrusters. The propellant usage is modeled with a constant flow-rate during thrusting and a propellant penalty for each firing. The penalty accounts for unburned propellants lost in each firing. Either on or off failures of any jet can easily be induced. In some vehicle configurations the plumes from some of the reaction control thrusters impinge on the spacecraft structure, The effect of this impingement is important for two reasons: it reduces the effectiveness of the thrusters; and extended impingement can damage the structure. The effect is modeled by adding impingement forces and moments to the net force and moment when an impinging thruster is on.

Vehicle trajectory control is normally provided by the main propulsion system. The model of the LM Descent Propulsion System is given here as an example. The main engine is throttleable and can be gimbaled within a small region about the vehicle centerline. The application or removal of the primary thrust is delayed from the simulated AGC on-off commands by the amount required to match the impulse due to typical build-up or tail-off transients. The engine throttle can be controlled by commands from the AGC and/or inputs from the simulated astronaut's hand controller. The sum of the commands from these two sources represents the commanded thrust level. The simulated actual thrust output of the engine follows this commanded thrust with a first-order lag. Propellant consumption is modeled by using a linear interpolation on a table of specific impulse versus engine thrust. The gimbaled engine is positioned by two actuators which allow the thrust vector to be pointed at thevehicle center-of-gravity. Extension or retraction of the actuators,

as commanded by the AGC, is translated into a simulated angular rotation of the engine about its mount. The motion of the engine due to structural compliance of the engine mount during thrusting is simulated as a rotation, proportional to thrust magnitude, of the thrust vector with respect to the vehicle centerline. Constant initial thrust vector alignment errors due to mechanical tolerances are also simulated, for both engine alignment and throat geometry errors.

Additional contact accelerations occur during ascent engine thrusting immediately after separation from the descent stage, due to plume impingement on the descent stage. The build up and decay of these staging forces and torques are modeled by two sets of second-order polynomial time series.

Propellant typically accounts for over half the vehicle's total weight. Thus, the sloshing of propellant in the tanks has a significant effect on the vehicle dynamics. Propellant sloshing is modeled by either a linear or non-linear two-dimensional mass-spring oscillator. Either model can be selected, based on a choice between the faster computer execution time of the linear model and the better fidelity of the non-linear model. The linear slosh model consists of a simulated slosh mass sliding on a frictionless plane which is perpendicular to the tank centerline. The mass is constrained by a linear spring which is attached at the centerline of the tank. The non-linear system is modeled as a simulated mass rolling in a paraboloid while constrained by a cubic spring.[5] The origin of the paraboloid is the spring attachment point of the linear model, but the spring itself is free to slide along the axis of symmetry of the paraboloid which is parallel to the tank centerline. Both models have one of these lightly-damped oscillators for each propellant tank. The mass of the sloshing fluid, the frequency of the oscillator, and the point of attachment of the spring along the tank centerline (or the origin of the paraboloid, for the non-linear model) are all functions of the propellant remaining in the tank. They are determined by linear interpolation on tabulated data, which differs for each tank geometry and propellant density. For small disturbances, the two models are nearly equivalent, but for large disturbances the linear model allows slosh forces and moments to increase without limit, giving results which cannot be realistically interpreted, whereas the geometry of the non-linear model better approximates the physical limits of actual propellant sloshing in tanks.

The simulated variation of vehicle mass properties due to propellant consumption, change in vehicle configuration, or propellant shift between tanks is handled in a low frequency loop. The vehicle center-of-gravity and inertia are recomputed after the expenditure of every 50 pounds of propellant, by interpolation between tabulated values of these parameters as a function of vehicle mass. A

particular set of these tables is selected, depending upon the vehicle configuration being simulated. For certain configurations, other mass property effects are also modeled, such as shifting of propellant between two coupled tanks.

The principle sensor used by the AGC program for vehicle attitude and trajectory measurements is the inertial measurement unit (IMU).[6] This unit includes a gimbaled inertial platform, gyroscopically stabilized in three axes with three integrating accelerometers mounted on the platform. Each of the three gimbals has a torquer which is used to coarse align the platform to a desired orientation. These torquers are commanded by the AGC program through a digital-to-analog convertor. Each gimbal has resolvers which drive the analog-to-digital convertors used to encode the gimbal angle. In addition, each gyro has a torquer which can be commanded by the flight program. All of these elements are simulated by the IMU routine. The routine maintains an orthogonal transformation matrix which defines the stable platform orientation with respect to an inertial frame. The active vehicle routine supplies an orthogonal transformation matrix defining vehicle orientation with respect to the same reference inertial frame. Using these two matrices, the transformation matrix defining vehicle attitude with respect to the platform system is computed. From this matrix the IMU routine extracts euler rotations corresponding to the platform gimbal angles. It also obtains vehicle attitude rate and acceleration from the active vehicle routine. With these it computes gimbal rate and gimbal acceleration. These are then used as input to the analog-to-digital convertor which encodes the gimbal angles for the AGC program. The encoders are highly non-linear devices whose amplitude and phase characteristics depend on the amplitude, frequency and phase of the input signals. It is quite important that these encoders be modeled in detail. Failure to include their effect in the control system design could lead to an instability in one or more of the digital attitude control loops.

As these analog-to-digital convertors generate the sensor inputs used by the autopilot for attitude determination, it is appropriate to introduce the effects of high frequency structural vibrations at this point. This is done by generating a discrete random process by passing a normally distributed, uncorrelated random sequence through a second-order difference equation which acts as a shaping filter responding to an essentially-white input spectrum. The coefficients of the difference equation are chosen so as to produce a power spectral density at the filter output which closely corresponds to the spectra measured at the IMU station in actual spacecraft vibration tests. The outputs of the shaping filters, one for each gimbal axis, are then added to the low frequency body attitude motion which is obtained by direct numerical integration. The analog-to-digital convertor model then operates on the sum of these two inputs.

The inertial platform is nominally fixed with respect to inertial space. In reality, drifts in the gyros cause it to deviate from this nominal alignment. These errors are simulated by three drift terms: a constant bias drift about each gyro's input axis, an acceleration-sensitive drift about its input axis and an acceleration-sensitive drift about its spin axis. These errors can be compensated by gyro-torquing, which is done under AGC program control. To accomplish gyro-torquing the flight program issues pulse trains of varying lengths to the individual torquers. The response of the gyro torquers depends on the length of the pulse train and the amount of torquing activity which has preceded it. This is simulated in the IMU routine by a time-varying pulse scaling. The time-varying scaling is itself changed as a function of the history of torquing activity. This provides a satisfactory model of both short-range and long-range effects of gyro-torquing.

The simulation of the three integrating accelerometers is accomplished by resolving the integrated contact acceleration obtained from the active vehicle routine onto the input axes of the accelerometers and converting this velocity change into a pulse output from each instrument. The integrating accelerometers are also imperfect instruments. Their errors are simulated by unique bias and scale factor errors for each accelerometer. The bias error causes a constant output error at all levels of acceleration input. The scale factor error causes the pulse output to be in error by a fraction of the acceleration applied to the instrument.

The IMU routine computes accelerometer and gimbal recipes which are available to the COMMUNICATOR routine. These are used to provide the inputs to the instruction simulator.

In addition to the dynamic interface with the flight program, the inertial measurement unit receives discrete moding commands from the AGC program. The IMU routine simulates this discrete interface. It tests the commands originating in the AGC program to detect violation of hardware constraints. It also returns those IMU moding and status discretes which are available to the flight software.

The problems of rendezvous and lunar landing require relative position and velocity sensor inputs to the AGC program. These measurements are provided by the rendezvous and landing radars in the lunar module and the VHF ranging system in the command module.

The rendezvous radar consists of an antenna assembly and an electronics assembly. The antenna is mounted on a two-gimbaled mount. These gimbals are gyroscopically stabilized and instrumented with resolvers and gyro torquers in a

manner similar to the gimbals of the inertial measurement unit. The resolver outputs are encoded by analog-to-digital convertors to provide the AGC program with angular position of the antenna bore sight. The gyro torquers are used by the flight program to direct the antenna bore sight along the AGC-estimated line of sight to the target vehicle.

The electronics assemhly generates the transmitter signal and attempts to acquire a return signal from the target vehicle transponder. When the transponder signal is detected, tracking loops are closed. The electronics assembly then assumes control of the antenna assembly and generates antenna-pointing error signals from the microwave return. These drive the antenna stabilization loops so as to keep the antenna pointed toward the transponder. Once the transponder has been acquired, status discretes are returned to the guidance computer indicating that range and range rate measurements are available. These measurements are then requested by the AGC program and the information incorporated, after filtering, into the AGC state vector estimates.

The simulation of the above functions of the rendezvous and landing radars is accomplished by the RADAR routine. This routine has four modes of operation. One of these is a trivial mode simulating periods when the radars are inactive or not under AGC control. In this case the radars are simply not simulated. The remaining three modes are search-designate, tracking and self-test. In the search-designate mode the two antenna stabilization loops are simulated in detail. These loops are modeled by two coupled third-order systems which are driven by AGC program-generated pulse trains and by rotational motion of the simulated spacecraft. The resulting antenna gimbal angles are encoded by adetailed simulation of the analog-to-digital convertor. In the search-designate mode, the angular excursion of the antenna is checked against specified limits corresponding to mechanical stops on the antenna mount. When the excursion reaches these limits the routine holds the angular position fixed at the limit until acombination of vehicle motion and AGC program positioning commands drive the angle away from the simulated stops.

When the AGC program issues a “self-track enable” discrete, the RADAR routine attempts to switch to its tracking mode. It causes the target vehicle state vector to be updated using the TARGET routine. It then determines: if the line of sight to the target is within a two-degree cone centered on the antenna bore sight; if the range to the target is less than 400 nautical miles; if the range rate is less than 1500 feet per second; and if a user-specified time delay has elapsed since the “self track enable” discrete was set. If all of these conditions are met the routine switches to its tracking mode. The conditions represent specified operational capabilities of the radar system.

On entering the tracking mode the RADAR routine returns a "data good" discrete to the guidance computer indicating that range and range rate information is now available. In this mode the antenna stabilization loops are no longer simulated in detail. It is assumed that the radar can hold the beam center along the line of sight to the transponder. The simulated beam center differs from the mechanical bore sight by a fixed angular bias whose magnitude is adjustable. In this mode the encoding of the antenna gimbal angle is simulated by a simple truncation of the gimbal angle to the nearest convertor bit. When in the tracking mode, the RADAR routine responds to AGC program requests for range or range rate readings. These readings are obtained by updating both active and passive vehicle state vectors to the time of the reading and computing the range or range rate. The quantity requested is properly scaled, quantized and returned to the read register of the simulated guidance computer where it is available to the instruction simulator. The routine monitors the various conditions which must be satisfied if acquisition is to be maintained. If any are violated the routine removes the "data good" discrete and returns to its search-designate mode.

The self-test mode is entered when the ASTRNAUT routine switches a simulated "radar test" panel switch. In this mode the RADAR routine allows test range and range rate readings to be made without the tracking conditions being satisfied.

The landing radar system of the lunar module provides measurements of slant range and three components of velocity with respect to the lunar surface. To generate these measurements, the radar transmits four beams to the surface. As in the case of the rendezvous radar simulation, the RADAR routine tests each beam for the satisfaction of certain tracking conditions. These conditions are functions of range andvelocity along the beam with respect to the surface and the angle of incidence of the beam. Only when the conditions are met will the RADAR routine allow the flight program to obtain valid readings. All appropriate status and moding discretes are simulated.

An interesting feature of the landing radar simulation is the representation of the lunar surface features. The simulation of these effects is quite important, since the altitude deviations caused by surface features may be viewed as noise which couples strongly into both the trajectory and attitude control problems. To represent these deviations in three dimensions, the altitude of the surface is tabulated as a function of range along several radials originating at the nominal landing site. To find the altitude at any intermediate point, the radial distance of the point from thenominal landing site is found. The two adjacent stored radials and the azimuths of the point with respect to these radials are found. A linear interpolation is made

to find the altitude at the appropriate range along each adjacent radial, as illustrated in Fig. 4. Knowing the altitude on each radial at the range of interest, a third linear interpolation based on azimuth is made to find the altitide at the point of interest.

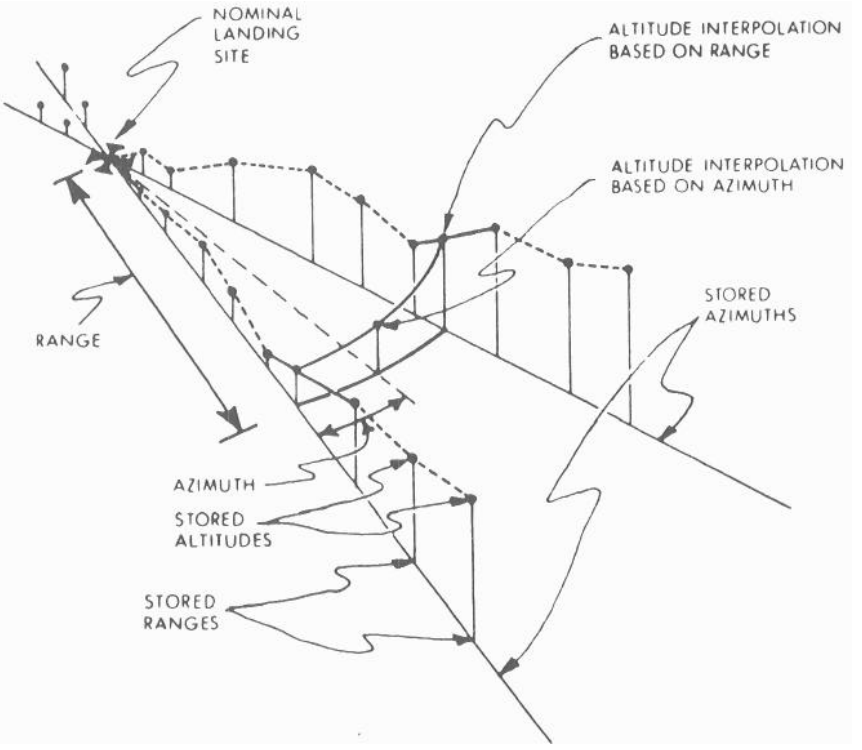


Fig. 4 Lunar surface representation

The problem is further complicated by the fact that the point of intersection of a beam with this surface representation is not explicitly known. Thus an algorithm to step down the beam, searching for the intersection, is provided. Data representing the surface surrounding each potential landing site can easily be generated. In this way all the potential interaction of the flight program with the surface features may be investigated.

A number of physical phenomena, external to the spacecraft, are simulated by a single routine called UNIVERSE. The primary function of this routine is to maintain the activevehicle trajectory in the form of a six-dimensional state vector: three components each for position and velocity. The routine computes the free-fall state vector by a direct numerical integration of the gravitational accelerations acting on the spacecraft. To these are added the state perturbations due to contact accelerations acting on the spacecraft. As indicated, these are generated by the

active vehicle routine. The perturbations are incorporated in the state vector approximately every two seconds or whenever a precise state vector is required. The gravitational effects simulated are the spherical principle body attraction and the non-spherical principle body perturbations. The spherical Sun and other-body attractions are included.

The Sun and Moon positions required for the gravity computations are obtained by interpolation from stored ephemerides. Also generated, on request from stored data, is the position of any of the 37 navigation stars used for Apollo missions. The routine computes planetary-inertial transformations for both the Earth and Moon. Amodelof the Earth's atmosphere based on the U.S. Standard Atmosphereof 1962 [7] is used with the active vehicle state vector to compute dynamic pressure and Mach number, which in turn determine the aerodynamic forces acting on the vehicle.

This summary of some of the significant models included in the Apollo All-Digital Simulator has been presented to illustrate the breadth and detail of the environment simulation.

VI. Simulator Validation

Much of the simulation capability described above was developed before flight testing began. With the advent of Apollo missions it became possible to verify the fidelity of the simulation against data obtained by telemetry during a flight. The results of one such exercise are presented in the following section.

To verify the fidelity of the Apollo All-Digital Simulator, certain key variables such as accelerometer output or inertial platform gimbal angles were compared with the samequantities telemetered to the ground during actual flights. A maneuver of particular interest was a long burn of the lunar module descent engine, made during the Apollo IX mission. During this burn the command and service modules were docked with the lunar module.

In attempting to duplicate the burn, both the simulated guidance computer and its environment were initialized to the best available estimates of the conditions that actually existed prior to the maneuver. The simulated computer's erasable memory was set to values corresponding to those loaded into the actual computer just before launch. These included initial mass of the LM and CSM, accelerometer bias terms, and the launch time. The AGC estimate of the vehicle's state vector and the inertial-to-stable-member transformation matrix were obtained from telemetry from the actual computer during the mission. Because telemetry data

represented the best estimate of the vehicle state, it was also used to initialize the environment's positionvector, velocity vector, and IMU orientation. The simulated astronaut's inputs to the AGC through the keyboard were also matched to the telemetry data. The balance of the environment's initialization was completed using the pre-flight estimates of the mission performance parameters. These items included the mass properties, engine parameters and modeling coefficients of the LM's data file.

All variables for which telemetry data was available were compared. Excellent agreement was obtained even during the transient period at the start of the burn. Figure 5 shows a time history of the inner gimbal of the stable platform during the attitude transient. This trace corresponds roughly to the yaw attitude of the vehicle.

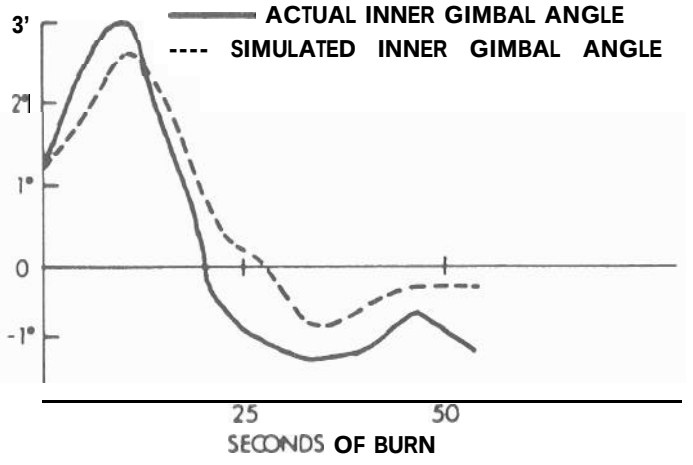


Fig. 5 Initial attitude transients

The agreement between simulated and actual inner gimbal angle is typical of that obtained for other variables. There was however a significant discrepancy in the outer IMU gimbal angle, which is illustrated in Fig. 6. With the prevailing stable

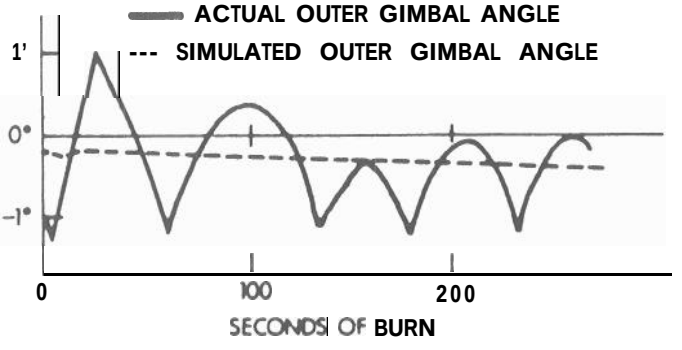


Fig. 6 Comparison of outer gimbal angles for original simulation

platform alignment, this gimbal roughly reflected vehicle rotation about its thrust axis. The actual gimbal angle was repeatedly driven against the lower limit of the digital autopilot deadband. Reaction control jets were then fired briefly by the autopilot to drive thevehicle attitude error toward zero, resulting in parabolic curves which are indicative of the response to a constant moment about the outer gimbal axis. This behavior is clearly not present in the simulator. The effect causes a non- zero average attitude error as opposed to the zero or very small average attitude error one would expect from a more symmetric limit cycle. This could be significant in situations such as the lunar landing where great pointing accuracy is required.

After verifying the initialization parameters and other aspects of the simulation, a fairly extensive investigation was launched to determine the source of the torque. Since engine rig tests had not been designed to measure torques about the thrust axis, the manufacturer could not offer any hardware test data. However, the most likely source of the torque was believed to be swirling of the exhaust gases in the engine nozzle, which had been observed in other smaller engines.

The exhaust gas swirling was modeled in the simulator as a torque about the thrust axis which was proportional to the thrust. This model was included in the LEM vehicle routine and the simulation was repeated, with the results shown in Fig. 7. The outer gimbal angle plot from the improved simulation agreed approximately in shape, amplitude and period with the actual data from telemetry. The other axes were practically uneffected, and remained in agreement with the telemetry data.

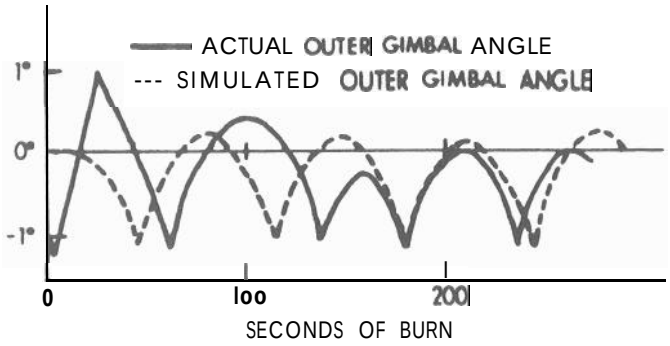


Fig. 7 Comparison of outer gimbal angles for revised simulation

VII. Conclusions

The capabilities described above were not realized without attendant costs. We briefly summarize some of these costs and then several unique advantages of

this approach to simulation. First, because of the all-digital nature of this facility, every system or phenomenon which could influence the flight software had to be modeled. The neglect of a single subtle characteristic could have catastrophic results. Also, translation of all systems and phenomena into digital programs demands a high level of technical competance. The development and maintenance of the simulator has required an engineering and programming effort estimated at 50 man-years expended over a five-year period. Furthermore, the complexity of the resulting simulator makes it difficult for inexperienced personnel to use. A particularly troublesome feature is the lack of any interactive capability, although this has been overcome by the development of an interactive version. Finally, a powerful computation facility is required to operate such comprehensive software.

The limitations of this capability are outweighed by its unique advantages. With the present facility the simulator operates significantly faster than real time in most applications. That is, it typically requires 0.8 second of real time to simulate one second of AGC and environment time. Thus the simulator is not constrained to operate at real time as are many of the alternative simulators. Another important attribute of this type of facility is the exact reproducibility of simulations. Rapid response to new requirements and flexibility are also important features of this simulator.

Although several alternative simulators have been implemented, this simulator remains the pt imary flight software development and verification tool. Without it, the timely production of manned space flight software would have been difficult if not impossible.

References

1. Draper, C. S. et al, Space Navigation, Guidance and C ontrol, Technivision Limited, Maidenhead, England, August, 1966.
2. Guidance System Operations Plan for Manned LM Earth Orbital and Lunar Missions Using Program LUMINARY, MIT/IL R-567, Cambridge, Mass., November, 1968.
3. Users Guide to MAC-360, MIT/IL, Cambridge, Mass., September, 1967.
4. Mimno, P. R., Digital Simulation Manual, MIT/IL R-599, Cambridge, Mass., January, 1968.
5. Bauer, H. F. "Nonlinear Mechanical Model for the Description of Propellant Sloshing", AIAA Journal, Vol. 4, No. 9, pp. 1662-1668, September 1966.
6. Draper, C. S., op. cit.
7. U. S. Standard Atmosphere, 1962, U. S. Weather Bureau, Washington, D. C., December, 1962.

DISTRIBUTION LIST

M. Adams
P. Adler
R. Aiyawar
S. Albert
R. Bairnsfather
A. Bancroft
J. Barker
N. Barnert
R. Battin
C. Beals
W. Bernikowich
L. Berman
H. Blair -Smith
D. Bowler
T. Brand
P. Brennan
N. Brodeur
P. Canepa
G. Cherry
S. Copps
D. Corwin
R. Covelli
M. Cramer
D. Crocker
R. Crisp
W. Daly
S. D'Amore
R. D'Angelo
W. Danforth
S. David
C. Davis
J. Deckert
V. Demery
D. Densmore
L. Drane
V. Dunbar
M. Dunlavey
G. Edmonds
S. Eliassen
A. Engel
D. Estabrook
D. Eyles
P. Felleman
S. Femino (20)
R. Finkelst ein
T. Fitzgibbon
J. Fleming
D. Fraser
J. Gallagher
R. Gilbert
J. Gilmore
C. Gilson
K. Glick (20)
S. Goldberger
J. Goode
L. Goodman
R. Goss
E. Grace
A. Green
D. Grief
D. Gustafson
E.C. Hall
D. Hamilton
M. Hamilton
J. Hand
J. Harper
J. Harrison
R. Haslam
P. Heinemann
D. Hoag
A. Hopkins
D. Hsiung
E. Hughes
R. Ireland
S. Jennings
L. Johnson
M. Johnson
M. Johnston

DISTRIBUTION LIST (Cont)

J. Jones
P. Kachmar
G. Kalan
J. Keenan
D. Keene
J. Kernan
K. Kido
F. Kirven
A. Klumpp
G. Kossuth
B. Kriegsman
A. Laats
J. Laird
J. Laning
L. Larson
R. Larson
G. Levine
R. Lones
G. Mansbach
M. Markovitz
B. McCoy
R. McKern
H. McOuat
R. Metzinger
D. Millard
R. Millard
P. Mimno
D. Moore
J. Morse
E. Mullen
N. Neville
J. Nevins
J. O'Connor
G. Ogletree
E. Olsson
W. Ostanek
P. Peck
A. Penchuk
P. Philliou
N. Pippenger
W. Prince
c. Pu
L. Quagliata
J. Reber
J. Reed
J. Reedy
D. Reinke
P. Roberts
W. Robertson
B. Ross
R. Russell
P. Rye
R. Schlundt
G. Schmidt
R. Scholten
N. Sears
G. Silver
L. Silver
J. St. Amand
W. Stameris
R. Stengel
G. Stubbs
W. Tanner
W. Tempelman
J. Turnbull
J. Vella
P. Volante
J. Vittek
R. Wadsworth
R. Weatherbee
P. Weissman
R. Werner
P. White
R. White
R. Whittredge
P. Wolff
C. Work
S. Zeldin
Apollo Library (2)
MIT/Library (6)

External:

NASA/ RASP0 (1)

AC Electronics (3)

Kollsman (2)

Raytheon (2)

MSC:

National Aeronautics and Space Administration (21&1R)
Manned Spacecraft Center
Houston, Texas 77058
ATTN Apollo Document Control Group (BM 86) (18&1R)
M. Holley (2)
T. Gibson (1)

KSC : (1R)

National Aeronautics and Space Administration
J. F. Kennedy Space Center
J. F. Kennedy Space Center, Florida 32899
ATTN: Technical Document Control Office

LRC : (2)

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia
ATTN Mr. A.T. Mattson

GA: (3&1R)

Grumman Aerospace Corporation
Data Operations and Services, Plant 25
Bethpage, Long Island, New York
ATTN: Mr. E. Stern

NAR: (8&1R)

North American Rockwell, Inc
Space Division
122 14 Lakewood Boulevard
Downey, California 9024 1
ATTN: CSM Data Management
D/ 096-402 AE99

NAR RASPO: (1)

NASA Resident Apollo Spacecraft Program Office
North American Rockwell, Inc.
Space Division
122 14 Lakewood Boulevard
Downey, California 9024 1

GE RASPO: (1)

NASA Daytona Beach Operations
P. 0. Box 2500
Daytona Beach Florida 32015
ATTN: Mr. H. Lyman