A Testbed for the MSX Attitude and Tracking Processors

Daniel S. Wilson

The Midcourse Space Experiment (MSX) spacecraft employs infrared, ultraviolet, and visible light sensors to collect images and spectrographic signatures on a variety of targets, especially missiles, other satellites, and auroral phenomena. These instruments are fixed in the satellite body and face in a common direction. Their fields of view vary, with some of them being quite small ($1 \times 3°$). Thus, as MSX travels in its orbit, the entire satellite must be precisely rotated to keep the target in view. In fact, a pointing accuracy requirement of 0.1° was imposed on the satellite's design. To help meet this requirement, APL designed and built an elaborate testbed for the key elements of the MSX attitude and tracking subsystems. The testbed provides the means to assess the performance of these subsystems through simulations of missile encounters and other data collection events.

INTRODUCTION

A robot and a satellite such as the Midcourse Space Experiment (MSX) have much in common. The currently accepted definition of a robot is a programmable machine that, through the application of computer technology, exhibits a high degree of flexibility or adaptability.[1] That is, a robot, in the course of performing its programmed function, can autonomously adapt its behavior (programmed motion) in response to certain changes in its environment. Without the ability to adapt, the machine is merely a mechanical device. Thus, for a machine to be a robot it must be able to "perceive" (via sensor hardware), "think" (via computers), and move (via actuators) without continuous human intervention. The MSX satellite, likewise, has the ability to sense its surroundings, determine a course of action, and then autonomously perform the action in the context of an assigned task.

The task assigned to MSX is the collection of sensor data on various targets. To maximize the scientific value of the data, MSX must track the target and maintain it within 0.1° of the instruments' common boresight. Today, such a pointing accuracy is routinely achieved by many astronomy satellites, but such satellites have an advantage over MSX: their "targets" (typically stars) are inertially fixed. Some of the MSX scenarios, in contrast, require the satellite to cartwheel to keep the

target missile in the field of view. For the MSX to satisfy the 0.1° pointing accuracy requirement while performing such aerobatics is a considerable challenge for spacecraft designers. The attitude and tracking subsystems of the MSX spacecraft are designed to meet this requirement. This article discusses how to test and validate these subsystems before their deployment into space.

Designing a full-scale test environment in which MSX could practice tracking intercontinental ballistic missiles would require a large amount of space, a zero-gravity environment, and an enormous budget. Obviously, what is needed instead is a testbed that can "fool" the spacecraft's attitude and tracking elements into "thinking" that they are in orbit performing a real target-tracking scenario. This requirement was the charter for the testbed. The testbed simulator was used in all phases of testing, from unit level to end-to-end spacecraft level, to ensure that the system would meet its tracking specifications.

The concept of building a testbed to validate satellite-borne computers and software is not new. In fact, APL built testbed simulators for several earlier satellites involved in tracking moving targets.[2] The MSX testbed simulator, however, is much more complex. It requires more sensor models, more environmental models, more computing power, and much greater fidelity than the earlier testbeds.

The role of the testbed simulator in the MSX project must be understood in the context of the satellite's design. Therefore, a brief overview of the attitude and tracking subsystems of MSX is presented first in this article followed by a discussion of the requirements for the testbed simulator. The architecture and implementation of the simulator are also explained, followed by a discussion of results and future uses of the testbed.

OVERVIEW OF MSX ATTITUDE AND TRACKING SUBSYSTEMS

The Attitude Subsystem

The MSX attitude subsystem consists of a suite of attitude sensors, four reaction wheels, three magnetic torquing rods, and the attitude processor (AP)—a 16-bit computer programmed in the Ada language to perform autonomous, onboard attitude determination and control. The attitude sensor suite consists of a digital Sun sensor, two Earth horizon sensors, a three-axis magnetometer, two ring laser gyroscopes, and a star camera. The AP performs attitude control via the reaction wheels by independently commanding the torque to be applied by each of four wheel motors. The wheels may be spun in either direction at speeds up to 4500 rpm. By trading angular momentum between the satellite body and the wheels, the spacecraft can be turned in any direction. The magnetic torquing rods are used to dump excess angular momentum from the wheels. By energizing one or more of these mutually orthogonal rods at the appropriate point in the orbit, the AP sets up an interaction of the magnetic fields created by the rods with that of the Earth. The result is a torque on the satellite that can be used to hold the satellite stationary while the reaction wheels are slowly decelerated.

The AP is the "brain" of the attitude subsystem. Its software uses the measurements provided by the attitude sensors to determine the current orientation, i.e., attitude, of the satellite axes relative to a standard inertial frame of reference. The AP software then calculates the necessary control signals to the reaction wheels to adjust the satellite to the desired attitude. This attitude is calculated in one of several ways, depending on the mode commands from the satellite operators on the ground. There are two primary attitude modes. The first is the park mode, where the telescopes are pointed away from the Sun and Earth, while keeping the unblanketed side of the satellite rolled away from the Sun. The other is the track mode, where the satellite is pointed at the attitude requested by the tracking processor (TP).

The AP has some secondary functions as well. The MSX satellite is equipped with two solar panels that rotate about a common axis, and an X-band downlink antenna mounted on a gimballed arm. The AP flight software is responsible for keeping the solar panels rotated toward the Sun and the antenna pointed at the ground station during passes over APL's MSX Tracking Station.

The Tracking Subsystem

The MSX tracking subsystem consists mainly of the S-band beacon receiver, portions of the Ultraviolet and Visible Imagers and Spectrographic Imagers (UVISI) instrument, and the TP—another 16-bit computer programmed in Ada to perform autonomous, onboard target tracking. During a data collection event, the AP and TP engage in an electronic dialog using dedicated electrical interfaces to perform the desired spacecraft pointing. The AP provides the TP with the current estimates of MSX's orbit (position, velocity, and acceleration) and attitude parameters (orientation, angular velocity, and angular acceleration). The TP, in turn, provides the AP with the currently desired attitude parameters. These data are exchanged twice per second.

Some of the targets to be tracked and imaged are equipped with radio beacons to maintain compliance with the ABM treaty (1972), which prohibits unaided

acquisition and tracking of missiles from space. The MSX spacecraft's beacon receiver acquires such targets at long range (up to 8000 km) and provides tracking signals (azimuth and elevation angles and angle rates) to the TP.[3]

Some targets will be tracked with the aid of UVISI. Among the components of UVISI is an image processor, which can analyze the output of any of the four UVISI imagers and then provide the TP with the coordinates in the imager's frame of up to 23 detected objects (mostly stars).[4] The TP uses motion analysis and *a priori* knowledge of the target to distinguish the desired target from other objects.

The TP flight software provides the planners of a data collection event with a choice of eight pointing types. Commands are transmitted by ground operators to the TP giving it such information as the start and stop times of the data collection event, the pointing type, characterization parameters, selected roll option, and selected scan pattern. For example, if the target is a two-stage missile, the set of characterization parameters would include two sets of coefficients for the curves that approximate the predicted flight path. These command parameters represent the "programming" of the "robot" for a given task.

Other supported pointing types include reference objects (small calibration spheres ejected from MSX itself), satellites, stars, auroral bright spots, Earth-bound objects, Earth-limb pointing, and azimuth/elevation pointing in the local horizontal/local vertical reference frame.[5]

The TP flight software combines the available sensor measurements (from the beacon receiver or UVISI) with knowledge of the current MSX attitude (supplied by the AP) and the *a priori* target estimates (from the uplinked commands) to produce, via a Kalman filter, refined estimates of the target trajectory. The TP software then calculates the new desired pointing state (attitude, angular velocity, and angular acceleration) and sends it to the AP. If the target missile's path deviates from the *a priori* estimate because of a late launch or non-nominal booster performance, the TP software adapts to the situation. Likewise, if a sensor blackout occurs, the TP flight software extrapolates the target state information to provide a continuous sequence of desired attitude states to the AP.

A key consideration in the MSX attitude and tracking subsystems is data latency. The MSX sensor platform is moving at more than 7 km/s and rotating at more than 1.5°/s, and the target is moving at comparable speeds. Therefore, all sensor measurements must be time-tagged, and measurements and estimates must be propagated to a common time before they are incorporated into the various tracking and steering algorithms.

REQUIREMENTS FOR THE TESTBED SIMULATOR

The primary requirement for the testbed simulator is to provide a tool to support the development and checkout of the flight software residing in the TP and AP. An analysis of the needs of the AP and TP flight software development teams yielded the following system requirements:

1. Include hardware equivalent to the computer hardware in the actual TP and AP so that any differences are transparent to the flight software. Thus, the testbed simulator includes engineering models of the TP and AP. A testbed such as this is known as a hardware-in-the-loop simulator.
2. Connect directly to the flight computers (or their facsimiles) via the normal flight interfaces. The testbed simulator should present the same electrical interfaces to the AP and TP as in the flight configuration. This requirement is called first-circuit emulation.
3. Provide a configuration for testing the AP and the TP separately as well as simultaneously, and several special configurations for testing larger strings of subsystems.
4. Model all other spacecraft subsystems relevant to both the AP and the TP. These models are listed in the first two sections of the boxed insert.
5. Model the space environment as it is perceived by the spacecraft subsystems and the AP and TP. These models are listed in the last section of the boxed insert. They are considered the testbed simulator's "truth" models and "close the loop" between the AP/TP and the sensor models. For example, when the AP outputs a nonzero voltage on one of the reaction wheel torque interfaces, the spacecraft attitude dynamics model calculates the resulting change in orientation and angular velocity. The attitude model then provides these inputs to the sensor models. The sensor models generate new outputs to the AP, showing that the gyros registered the attitude motion, the Sun moved in the Sun sensor's field of view, the Earth moved in the horizon scanners' fields of view, etc.
6. Operate in real time. This requirement follows from the requirement to interface with the real flight software while it is running in the flight computers. Although fast-forward and rewind features would have been desirable, the flight software did not support these modes, so neither could the testbed simulator.
7. Simulate various target types that were determined by an analysis of mission science objectives (Table 1).

8. Provide the capability to record in disk log files all inputs and outputs of the systems under test (AP, TP, or both). These files are necessary to support efficient debugging of the flight software. In addition, the outputs of the truth models must be logged. Furthermore, the testbed simulator tool set must include tools to examine and plot data from the simulation log files.
9. Provide the ability to view the current log data while the simulation is in progress, since it is wasteful to run a 45-min simulation to completion if the test failed in the first 10 min.This capability is also needed for efficient debugging of flight software.
10. Provide the capability to drive stimulators for the spacecraft's battery-charging electronics, the Sun sensor, the horizon scanners, the magnetometer, the beacon receiver, and the UVISI imagers.

Figures 1 and 2 show typical test configurations for the AP, TP, and testbed simulator. Figure 1 shows the AP and TP being tested in a laboratory bench setting. The testbed provides models of all the missing spacecraft subsystems with interfaces to the AP and TP. Figure 2 shows the AP and TP being tested after installation on the spacecraft. The testbed simulator models the reaction wheels, attitude sensors, target, and the spacecraft environment; the testbed also drives stimulators for the beacon receiver and the UVISI instrument.

REQUIRED TESTBED MODELS

Attitude processor testbed models

Command subsystem
Data handling (telemetry) subsystem
Sun sensor
Earth horizon sensors (two independent units)
Three-axis magnetometer
Three-axis ring laser gyroscopes (two independent units)
Star camera
Reaction wheels (four independent units)
Magnetic torquing rods (three mutually orthogonal units)
Solar panel drives (two independent units)
X-band antenna gimbal drives (two independent units)
Tracking processor

Tracking processor testbed models

Command subsystem
Data handling (telemetry) subsystem
Attitude processor
Beacon receiver
UVISI image processor
Onboard Signal and Data Processor, the image processor associated with the Spatial Infrared Imaging Telescope III (SPIRIT III)
Space-Based Visible telescope

Environmental models

Sun position
Moon position
Star positions and magnitudes (about 12,600 stars included)
Earth's magnetic field
MSX spacecraft orbit
MSX spacecraft attitude dynamics
Targets

TESTBED SIMULATOR ARCHITECTURE AND IMPLEMENTATION

The ideal architecture for a complex real-time application such as the testbed simulator would consist of many fast microcomputers sharing a common pool of memory. Each of the testbed simulator's models would have its own dedicated microcomputer. By running in parallel, the set of microprocessors would enable the testbed to emulate the simultaneous interaction of the various satellite subsystems. Unfortunately, such an architecture would require more than 25 microcomputers for all the models and support functions (data logging, real-time display, etc.). Interfacing this many microcomputers seamlessly would be technically challenging as well as expensive (especially in 1989, when the testbed simulator architecture was selected).

An alternate design would use a single, large computer and a multitasking operating system to simulate the simultaneous operation of many subsystems by time-slicing and interrupt techniques. Each of the models would be implemented as a separate task. If the computer were fast enough to handle the worst-case collision of tasks (when a large number of tasks all need to be serviced by the computer at the same time) and still maintain the illusion of simultaneity, then this

Table 1. Target types provided by the testbed simulator.

Target type	Modeling method
Missile (from launch to impact)	Table of Earth-fixed vectors
Missile that deploys a secondary object	Two tables of Earth-fixed vectors
Calibration sphere ejected from MSX	Spacecraft frame ejection velocity vector and an orbit integrator
Another satellite	Initial position and velocity vectors and an orbit integrator
Earth-fixed object with optional random walk	Latitude–longitude–altitude and standard Earth model
An object fixed in spacecraft-centered local vertical frame	Orbit integrator and constant azimuth and elevation angles
Inertially fixed object	Constant position vector

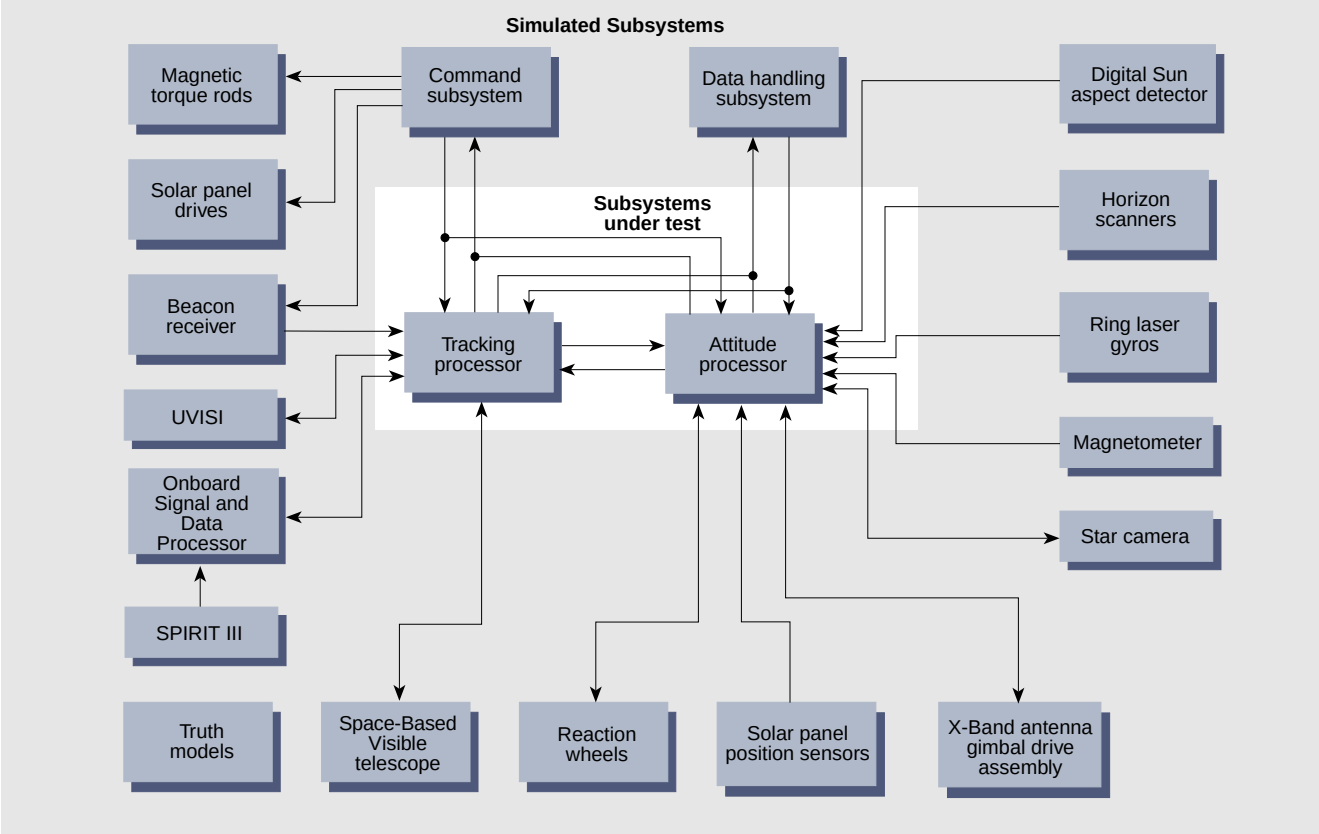

Figure 1. Testbed simulator concept. The tracking processor (TP) and attitude processor (AP) are the spacecraft subsystems being tested. The testbed provides models of all the other spacecraft subsystems having interfaces to the TP and AP. Note the presence of the truth models: the Sun, Moon, magnetic field, orbit, attitude, and target models.

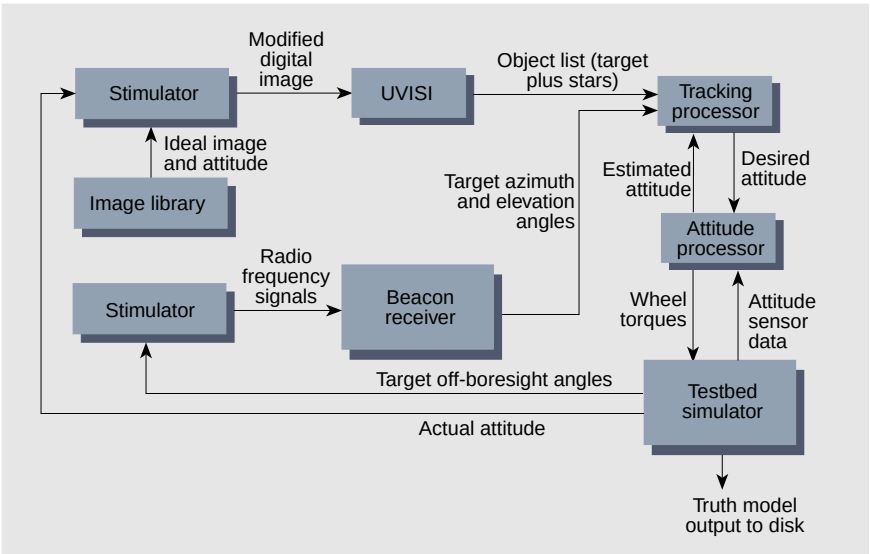

Figure 2. Testbed simulator configuration for closed-loop spacecraft tests. The real spacecraft subsystems replace the testbed's models for the beacon receiver, UVISI, and command and data handling (telemetry) subsystems.

architecture would seem relatively risk-free because the computer hardware and multitasking operating system could be obtained as commercial, off-the-shelf (COTS) equipment. However, with this approach, the computer resources could be found deficient after the simulator is implemented.

The architecture we chose for the testbed simulator is a hybrid of the two extremes in the preceding discussion. That is, we procured a few microcomputers and a commercial multitasking operating system and linked them to a minicomputer using COTS shared-memory technology.

The testbed simulator architecture is illustrated in Fig. 3. The minicomputer used was a Digital Equipment Corporation MicroVAX 3800, running the VAX/VMS operating system. This computer is capable of about one million floating point operations per second (1 mflops).

A Motorola Corporation MVME 133A single-board computer was selected as the microcomputer. This board includes a 20-MHz Motorola 68020

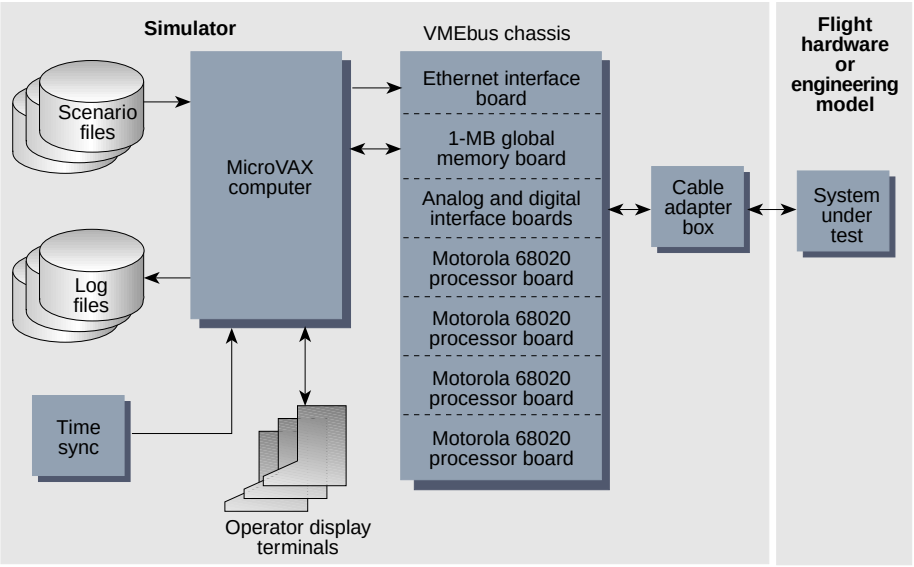

Figure 3. Testbed simulator architecture. The models run on the four Motorola processors. The support functions of startup, scenario control, data logging, and display run on the MicroVAX minicomputer. The cable adapter boxes provide the interconnections to the attitude processor, tracking processor, and any stimulators being driven by the testbed simulator.

microprocessor, 68881 math coprocessor, 1 MB of random access memory (RAM), timers, serial ports, and industry standard VMEbus[6] interface circuitry. The 68020/68881 combination was benchmarked at about 0.07 mflops—not fast by today's standards, but not slow for early 1989. The board's 32-bit architecture made it especially attractive. We estimated that four such boards per testbed simulator would be sufficient for our processing load. The single-board computers used up only 4 of the 21 slots in the VMEbus chassis that we selected, so there was room for expansion, if necessary.

Because of the first-circuit emulation requirement imposed on the testbed simulator, and the unique nature of many of the spacecraft electrical interfaces, we had to design and fabricate a number of custom digital interfaces between the testbed simulator and the AP and TP. Using an industry standard such as the VMEbus aided this task since tools and parts for rapid hardware prototyping in this environment were mature and widely available. This choice also enabled us to use COTS boards for the 16 analog-to-digital and digital-to-analog interface circuits required. Thirty-nine custom electrical interfaces representing 13 unique types were designed and fabricated. Most of these custom interfaces were "smart" interfaces in that they employed embedded programmable logic state machines or 8-bit microcontrollers to perform input/output services. These features enabled the models running in the Motorola processors to be applied solely to the application message level of the interface protocols. All of the lower levels of the protocols were implemented in hardware and firmware on the interface boards. Without these smart interfaces, the simulator's four Motorola processors would never have been able to handle the processing load.

A Q-bus-to-VMEbus adapter from the BIT3 Corporation provided the required shared memory between the Motorola boards and the VAX minicomputer. (The Q-bus is the main data path in a MicroVAX computer.) This product includes a board containing 1 MB of RAM accessible simultaneously from both the VAX and the Motorola computers. This dual-port memory board, together with some custom designed buffer management software, enabled all of the computers to work together almost seamlessly.

The software architecture of the testbed simulator cannot be separated from the hardware architecture. The two were conceived as a unit. Ada was the natural choice for programming language given our sponsor, the Department of Defense, and the nature of our application. Ada had already been selected for most of the satellite's onboard software. Several Ada compiler vendors offered dual-host development systems for the VAX/Motorola MVME-133A combination. In fact, this combination seemed to be the industry standard among Ada cross-developers, which strongly influenced our selection of the MVME-133A microcomputer. We selected the Ready Systems Comprehensive Ada Development System as our Ada compiler. With this tool set, we write Ada programs on the VAX, which can be run on either the VAX or the Motorola microcomputers.

Our software design called for each model to be implemented as a task running on one of the microcomputers in the VMEbus chassis. The support functions of scenario creation, startup/control, data logging, real-time display, report generation, and data plotting were assigned to the VAX minicomputer. Intertask communication was handled by buffer queues and semaphore-protected buffers on the shared memory board. VAX/VMS served as the multitasking operating system for the minicomputer, and the Ready Systems ARTX (Ada Real Time Executive) served that role on the Motorola microcomputers.

We recognized early on that the testbed simulator would be used in a variety of test configurations (AP only, TP only, etc.). Therefore, we established the design goal of being able to switch test configurations

without having to plug and unplug interface cables. This goal led to the concept of the cable adapter boxes shown in Figs. 3 and 4. The boxes are equipped with a few ganged switches to enable the user to select test configurations simply and reliably.

Display Capability

Initially, we planned to provide real-time displays containing numeric data only, that is, no graphics. However, after seeing a demonstration of a three-dimensional perspective animation of the MSX spacecraft running on a personal computer, we added a personal computer to the testbed simulator's equipment suite. Then, after some tailoring, we made a graphical display, called SeeMSX, a standard part of the testbed simulator's display capability. A sample frame from a typical scenario is shown in Fig. 5. Having this "chase plane" view of the satellite during simulations has been invaluable. The chase plane view is what would be seen from the starboard window of a plane flying alongside the spacecraft (traveling to the left), parallel to the ground.

The SeeMSX display contains a wealth of information in a compact, color-coded format. The center portion of the display (Fig. 5) contains a line drawing of the spacecraft body showing several key features of the satellite design: the beacon receiver antenna deck (the red rectangle on one end of the spacecraft), the two solar panels (the large blue rectangles; the panels appear gray if the side coated with solar cells faces away from the viewer), and the X-band antenna (the red *T* on the base of the satellite).

The Sun geometry portion of the display (upper left corner) informs the viewer of the position of MSX in its orbit relative to the shadowed portion of the orbit. The line rotates like the second hand on a watch, making it easy to tell when eclipse is imminent or half an orbit away.

The reaction wheel portion of the display (lower right corner) gives speed and commanded torque for each of the four reaction wheels. The estimated power consumed by torquing the wheels is also shown as a narrow white bar along the right edge of the wheel display.

The boresight portion of the display (upper right corner) is the most useful part of the display for assessing the pointing performance of the tracking system. The viewpoint is that of an observer located inside the spacecraft, looking out along the boresight of one of the spacecraft telescopes. The display software places a white circle (with an inverted *T* on it) on the target. The green circle represents a hypothetical field of view of radius 0.1°. If the target is more than 0.1° but less than 1°, then the radius is changed automatically to 1°, and the field of view is shown as a yellow circle instead of green. If the target moves more than 1° away from the boresight, the radius switches automatically to 10° and the color switches to red. The green-yellow-red/good-fair-poor color scheme is used also on some of the numeric fields of the display, notably the solar panel and X-band antenna error angles.

In addition to pointing the telescopes at the target, the flight software is commanded to maintain a particular roll angle orientation around the line of sight to the target. The roll angle error can be approximated in the SeeMSX display by examining the white spot and noting the tilt of the base of the inverted *T*. The roll angle error is displayed more accurately in the bar graph just to the left of the boresight display. This bar graph is automatically scaled and color-coded in the same manner as the boresight circle.

The SeeMSX display has been so successful as a visualization aid that the software that created it was enhanced to accept real-time spacecraft telemetry and then installed in APL's MSX Mission Control Center. Thus, every time MSX passes over the APL ground station, operators in the Mission

Figure 4. Testbed simulator laboratory. The two identical copies of the testbed are arranged back-to-back in this view. From foreground to background, the visible elements of the testbed are the two cable adapter boxes, the VMEbus chassis, display terminals, and MicroVAX computer cabinets.

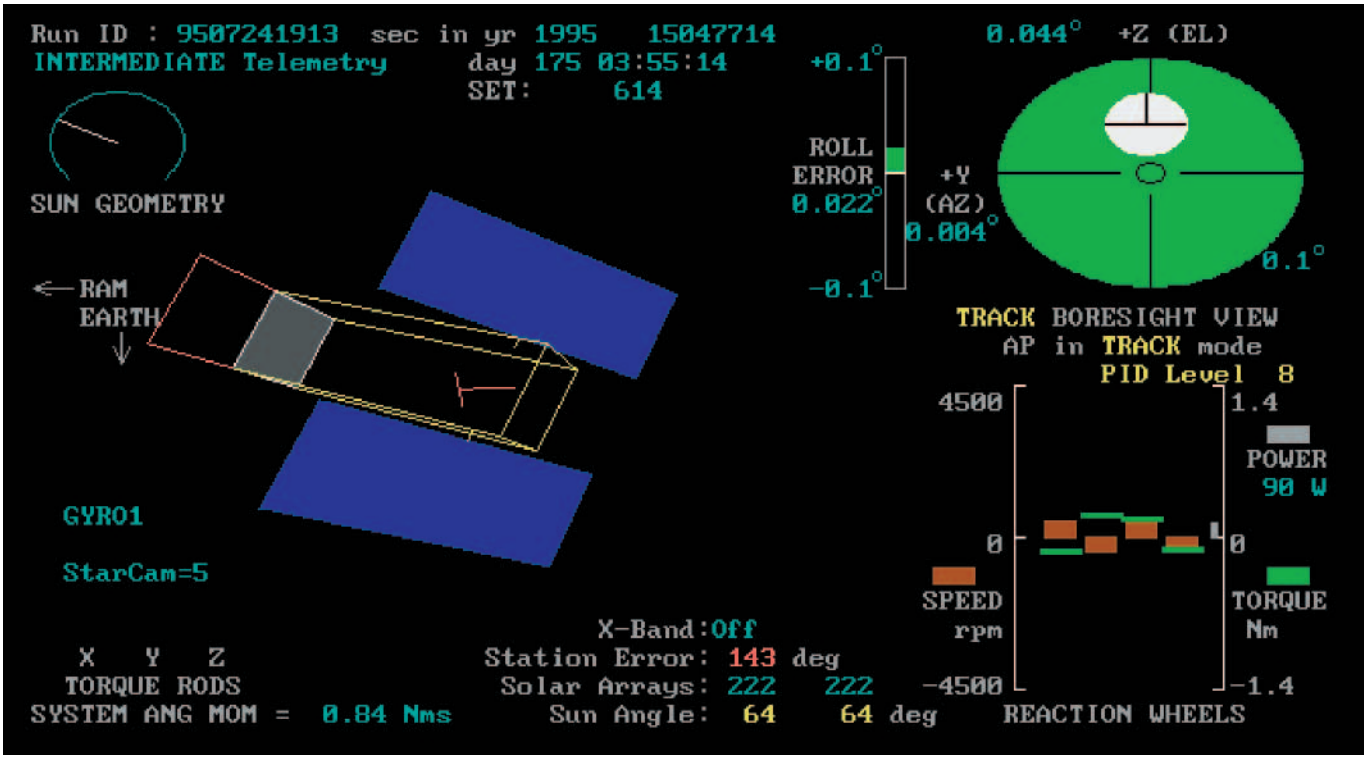

Figure 5. Sample SeeMSX display showing MSX tracking a target almost perfectly. The spacecraft is pictured in the local vertical frame, and the Earth is directly below the spacecraft, which is traveling to the left (in the "ram" direction). The green circle shows the current azimuth (AZ) and elevation (EL) error angles relative to the target (the white circle).

Control Center will be able to assess the overall health of the spacecraft attitude and tracking systems with a quick glance at the SeeMSX display.

Some Testbed Models

Decisions had to be made regarding the level of fidelity of the truth models. Standard satellite orbit models, for example, range from low-fidelity, computationally simple models, to high-fidelity, computationally intensive models. The guiding principle in selecting the proper level of fidelity was that the models had to be just good enough to assess the performance of the flight software. For the orbit model, we chose to use the J2 gravity model (Earth oblateness) with no disturbance forces such as aerodynamic drag or solar radiation pressure. Over a few hours at an altitude of 900 km, drag has a negligible effect on MSX. The gravitational force was integrated via a fourth-order Runge–Kutta numerical integrator with a 0.5-s step size.

The spacecraft attitude truth model presented additional choices. We chose to model the reaction wheel torques, magnetic torque, gravity gradient torque, and the torque due to the venting of cryogenic gas from the SPIRIT III telescope. Again, we chose to ignore aerodynamic and solar pressure torques. The spacecraft attitude dynamics are calculated by inserting the modeled torques into Euler's equation, as applied to a system with four reaction wheels,[7] solving for the vehicle acceleration, and then integrating over time. Because some of the attitude sensors (star camera and gyros) provide precision attitude measurements to the AP at high rates (9 Hz for the star camera, 20 Hz for the gyros), we chose to use a 50-ms step size in the attitude model, together with a second-order Runge–Kutta integrator.

The star camera model is among the most complex of the testbed simulator models. The MSX star camera includes an embedded processor that manages five independent star-tracking windows that can be commanded by the AP to acquire and then autonomously track any five stars available in the camera's field of view. The camera sends a serial digital message nine times per second to the AP. This message contains the current coordinates and magnitude (brightness) of the stars being tracked by the five windows. The AP employs an onboard star catalog of about 9,200 stars and pattern-matching software to determine attitude from the star camera's messages with an accuracy of a few arcseconds. The onboard catalog was constructed from standard star catalogs by culling out faint stars in regions of high star density (i.e., in the galactic plane) and stars with near neighbors.

The testbed simulator model includes the software logic necessary to emulate the camera's functionality.

To realistically model the behavior of the camera, the simulator employs a star catalog of about 12,600 stars. This catalog is complete down to stars of magnitude 6.4. Therefore, this catalog, unlike the artificially thinned catalog embedded by the AP, is representative of the natural distribution of stars in the sky. The model then adds Gaussian noise to the star measurements in a way that increases with spacecraft slue rate. The model also employs a variable brightness threshold that causes the camera to drop track on fainter stars as the camera boresight approaches bright objects such as the Sun, Moon, or Earth. To run at 9 Hz, the star camera model required an entire Motorola processor board dedicated to it.

Connections to Other Stimulators

The testbed simulator was conceived primarily as a test tool for the AP and TP. However, with its truth models, the testbed has the necessary information to also drive stimulators for a variety of related subsystems. For example, throughout a simulation, the testbed simulator "knows" where the spacecraft, Sun, and Earth are. It "knows" the current spacecraft attitude and the rotation angles of the solar panels. With this information, the testbed simulator can calculate the amount of energy available from the solar panels. Thus, the testbed simulator was designed to make this calculation and write a message once per second to a stimulator attached to the spacecraft's battery-charging electronics. Similar techniques were employed to make the testbed simulator drive stimulators for the Sun sensor, horizon scanners, magnetometer, beacon receiver, and UVISI. These features allowed the testbed simulator to put major strings of spacecraft subsystems through realistic mission scenarios.

The testbed simulator's ability to drive stimulators for the beacon receiver and UVISI is noteworthy. In the case of the beacon receiver, the testbed simulator uses its orbit, target, and attitude models to calculate the target's azimuth and elevation angles relative to the center of the beacon receiver's field of view. These data are sent to a computer-controlled stimulator that puts the corresponding phase shifts into the radio-frequency signals being inserted into the beacon receiver's antenna leads. The beacon receiver's electronics then "see" the target exactly where the testbed simulator says it is. The beacon receiver's embedded computers then calculate the target azimuth and elevation angles, and send this information to the TP, which in turn tries to drive the azimuth and elevation angles to zero by generating a new desired attitude for the AP. In response, the AP torques the reaction wheels to rotate the vehicle accordingly. Since the reaction wheels are being emulated by the testbed simulator, we have a closed-loop, nearly end-to-end simulation of the MSX tracking system.

For the UVISI imagers, the details are more complicated, but the effect is the same. In a typical missile-tracking scenario, the imagers are expected to see a complex scene containing star streaks, the Earth limb, and the rocket plume. If we assume that the MSX orbit, the target trajectory, and the tracking and attitude performance are perfectly nominal, then we can assume that the target would be in the exact center of the image. Using such logic, all of the digital images for a particular scenario can be simulated in advance and saved in a library.

But in any given simulation, the tracking and attitude performance are not perfect, resulting in the target being somewhat off-center relative to the UVISI imager field of view. Since our goal is to assess tracking and attitude performance, we cannot assume that the target is in the center of each image. We solved this problem by generating the images slightly larger than the actual imager field of view. Then, during a simulation, the testbed simulator sends the current attitude to a computer-controlled stimulator for UVISI. The stimulator compares the current attitude with the nominal attitude used to generate the image for the current time. If the difference is nil, then the stimulator slices out the center portion of the oversized image and injects it into the UVISI flight electronics, just as if it came from the focal plane unit. However, if the attitude difference indicates that the target is slightly off to the right, then the stimulator slices out a portion of the image slightly to the left of center. Thus, this design permits nearly end-to-end simulation of MSX tracking scenarios involving the UVISI imagers.

RESULTS

The testbed simulator began to be used by the AP and TP flight software development teams at about the midpoint of their implementation phase. Prior to that, the flight software developers tested their software using the custom test equipment suite used by the flight hardware development team. As the flight (and testbed) software matured, the testbed simulator's ability to drive realistic mission scenarios increasingly made it the method of choice for testing. Hundreds of simulations were run during the latter half of the flight software implementation period. These runs typically employed the testbed's AP-only and TP-only modes.

Flight Software Testing

Most of the flight software defects found with the testbed simulator were, as expected, due to programmer or analyst error. Nevertheless, a handful of errors were traced to defects in the Ada compiler being used by the flight software team. That is, the program contained a correct sequence of statements in the Ada language, but

the compiler did not translate the statements into a correct sequence of machine language statements. Such problems remind us that although the use of COTS products greatly reduces risk, risk cannot be eliminated entirely.

As the AP and TP flight software neared completion, the engineering models of the AP and TP flight computers were mated to one another on a laboratory bench and then tested as a unit with the testbed simulator. Next, the prototype beacon receiver electronics were added, along with their own testbed equipment. As described in the preceding section, the testbed simulator was linked to the beacon receiver's testbed equipment (i.e., stimulator) to allow testing of closed-loop missile tracking. Then the prototype UVISI image processor was added, along with its own testbed equipment, and more closed-loop missile tracking scenarios were conducted.

After the flight hardware for the AP and TP was integrated with the spacecraft, one of the two copies of the testbed simulator was relocated to the integration facility, and simulations with flight hardware in the loop began. These tests included the following:

1. Spacecraft/booster separation and initial attitude acquisition, including detumbling from non-nominal separation, deploying the solar panels, and orienting the panels to the Sun
2. Initial X-band antenna deployment and the first few station passes
3. Reaction wheel momentum dumping using the gravity gradient torque and using the magnetic torque rods
4. AP response to attitude sensor failures
5. Tracking of missiles, ejected reference objects, and other satellites
6. Earth-limb and celestial background scans
7. Thermal/vacuum performance

Missile Tracking Scenarios

For the MSX tracking subsystems, the closed-loop missile tracking scenarios were the most interesting and challenging spacecraft tests. One such test, MSX Dedicated Target (MDT) V, involved tracking a missile from horizon to horizon with both the beacon receiver and one of the UVISI imagers. The geometry of the encounter required the spacecraft's pitch rates to reach 1.3°/s at the point of closest approach between the missile and MSX. The high angular velocity stresses the AP's attitude determination algorithm, its attitude control algorithm, and the TP's trajectory determination algorithm. Several variations of this scenario were run:

1. Beacon receiver tracking only
2. UVISI imager tracking only
3. Simultaneous beacon receiver and UVISI imager tracking
4. Late missile launch
5. Intentional beacon receiver loss of track and reacquisition in the middle of the run

One of the more interesting issues that surfaced during tests of the MDT V scenario was a problem with stars falling into the target tracking gate. Recall that the UVISI image processor sends a message to the TP each 0.5 s containing a list of up to 23 objects detected in its field of view. The target is presumably in the list somewhere; the rest of the objects are probably stars. The TP uses its current estimate of the target position and velocity, together with the current estimate of attitude and angular velocity from the AP, to identify the target among the list of objects. Objects in the list that appear inertially fixed (but moving in the imager frame) are probably stars; an object that appears to be moving inertially (but fixed in the imager frame) is probably the target. The TP draws a box, called a gate, around the portion of the imager frame where the target is expected to be. The MDT V tests with the UVISI imagers revealed that in some image compaction modes of the imager, the TP's target gate was too large, leading to a high probability that a star would fall in the target gate. The TP was observed to track the star for a few seconds before returning to the real target. The analysis of this anomaly led to the alteration of the interface between UVISI and the TP. The UVISI image processor flight software was modified to include the compaction factor in its message to the TP. The TP flight software was modified to dynamically change the size of the target gate based on the current UVISI image compaction factor. The result was improved target tracking performance with the UVISI imagers.

The high rotation rates demanded by the MDT V scenario helped the team find other problems in the flight software, too. For example, we found that the attitude determination algorithms, which performed superbly at modest rates, broke down badly at high rates. One of the testbed simulator's analysis tools generated a report and graphs showing the difference between the AP's attitude estimate, as reported in the telemetry stream, and the testbed's attitude truth model. This tool highlighted the rate dependence in the AP's attitude error. By experimenting on the testbed with various simulated attitude sensor failures, we were able to show that the errors were in the AP's processing of gyro data. After the flight algorithms were modified to correctly compensate for the latency in the gyro data, the testbed's analysis tool showed that the performance of the AP's attitude determination software was acceptable even at rates higher than those in the MDT V scenario.

Figure 6 presents testbed simulator data from one of the MDT V scenarios. In this variant, the TP's closed-loop tracking algorithm employed data from both the beacon receiver and the narrow-field visible light

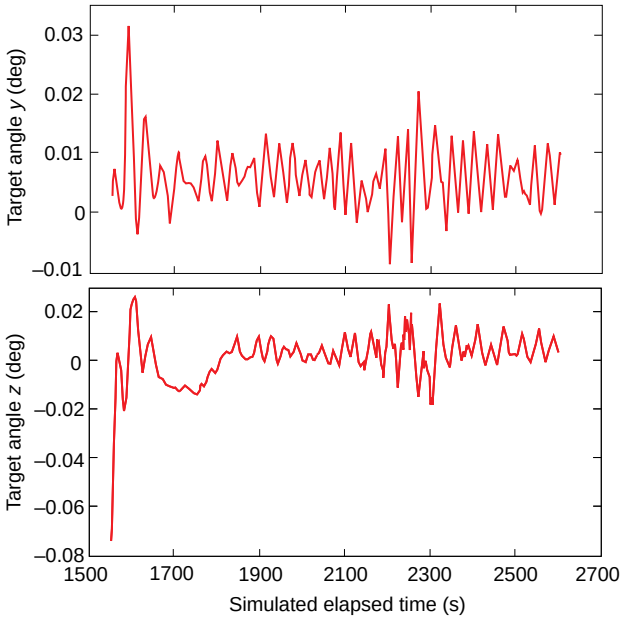

Figure 6. Pointing errors from a closed-loop missile tracking simulation performed during spacecraft integration testing. The horizontal axis shows elapsed time in seconds since the start of the simulation. The closest approach of spacecraft and missile occurred at 2250 s. The pointing specification is ±0.1°.

imager in UVISI. (Blending data from diverse sensors to track an object is sometimes called sensor fusion.) The graphs are constructed from log files of the testbed's truth models, specifically, the MSX orbit, attitude, and target models. The graphs show, as a function of time, the target line-of-sight error. The target is supposed to be maintained exactly on the spacecraft x axis during this event. Line-of-sight errors show up as small offsets along the spacecraft y and z axes. The larger transient on the left edge of the graphs is the result of attitude settling from a previous large-angle maneuver. The tracking errors are noticeably larger around 2250 s. This time corresponds to the closest approach of the missile to MSX. The graphs show that the tracking system satisfied the MSX pointing specification of 0.1° with better than 70% of the error budget to spare. The oscillations apparent in the graphs are artifacts of the beacon receiver stimulator; the resolution of the phase angle that can be specified in the stimulator is about 0.04°. This coarse resolution causes discontinuous jumps in the target location. The TP's Kalman filter, which has not been tuned to eliminate this artifact, follows the sensor data. (The UVISI stimulator's resolution is about 0.005°.)

Other Test Benefits

In addition to finding a few problems in the flight software, the spacecraft-level tests had some collateral benefits. For example, some of the tests gave us the opportunity to try out actual flight command sequences. Numerous cases of inappropriate or omitted commands and incorrect timing between commands were found. In a few instances, the command planner selected the proper command, but the command translation software generated the wrong sequence of bits. One such error caused the least significant bits of certain floating point numbers to be incorrectly encoded in command sequences. These subtle errors could have been easily overlooked if not for the realism provided by the testbed simulator.

The training of mission operators was another collateral benefit of all this testing. The thermal vacuum test, in particular, provided an excellent training opportunity. During this test, the entire spacecraft was enclosed in a chamber capable of creating the environmental conditions that the satellite will experience in space, namely, vacuum and temperature extremes. The test was conducted under realistic, on-orbit conditions, for example, around-the-clock operation, simulated station passes, battery charging and discharging, and temperature cycling. Throughout this test, the testbed simulator controlled the stimulators for the horizon scanners, magnetometer, Sun sensor, and battery charging electronics, and provided simulated star camera, gyro, wheel speed, solar panel position, and X-band antenna position data to the AP. These tests gave operations personnel the chance to see realistic telemetry on their display terminals. They also learned how to recognize and respond to anomalies.

FUTURE USES

Postlaunch Software Development

The MSX operations plan requires the testbed simulator laboratory (the testbed and the engineering models of the AP and TP) to be maintained for the duration of the mission. Should a tracking anomaly occur on orbit, it will be possible to reconstruct the event in the laboratory and then experiment until the cause of the anomaly is isolated. If a "work-around" is proposed to avoid the anomaly, test and verification of the work-around procedure will be performed in the laboratory before it is attempted on the spacecraft.

This capability is not restricted to tracking anomalies, but applies to various other spacecraft anomalies. In many missions, a hardware failure causes the spacecraft operators to come up with creative solutions to accomplish a goal without the failed component. The testbed simulator laboratory provides a setting for safely experimenting with candidate solutions.

Should a flight software change be proposed, either in response to an anomaly or a change in requirements,

the software developers will use the testbed simulator to test and validate the changed software, and then run an extensive set of regression tests to ensure that the change has not inadvertently introduced an error elsewhere in the software. This is the same regression test suite that we already used to validate the final prelaunch flight software programs.

Event Rehearsals

In addition to its use in support of flight software maintenance, the testbed simulator has an evolving role in support of operations planners. The analysts tasked with the translation of experimenters' requests into a scheduled data collection event on the spacecraft have a variety of tools that they routinely use during the planning process. The development of software to convert their planned sequences of spacecraft commands into the format required for testbed simulator scenarios makes it possible to rehearse the planned event on the testbed. Although such rehearsals are not practical for every event due to the real-time nature of the testbed simulator (What value is a 24-hour weather forecast if it takes 24 hours to generate it?), it is useful to have this capability for use with certain high-value or particularly complicated events. Operations planners made extensive use of this rehearsal capability during prelaunch event planning.

CONCLUSION

The investment made in developing the testbed simulator has helped to ensure that the MSX will meet its tracking specifications. This investment will continue to pay dividends throughout the mission life as the testbed simulator supports anomaly investigation, flight software maintenance, and event planning.

REFERENCES

[1] Clarke, R., "Asimov's Laws of Robotics: Implications for Information Technology," *Comput.* **26**(12), 53–61 (1993).

[2] Englar, T. S., Gregorski, F. S., Smith, R. R., Frank, W. A., and Wilson, D. S., "Simulation of the Environment of an Autonomous Satellite to Provide a Testbed for Real-Time Software/Hardware Validation," in *Proc. 1989 Summer Comput. Simul. Conf.*, Austin, TX, pp. 247–253 (1989).

[3] Valverde, C. R., Stilwell, R. K., Russo, A. A., and McKnight, T. R., "The S-Band Beacon Receiver for the Midcourse Space Experiment," *Johns Hopkins APL Tech. Dig.* **15**(1), 67–81 (1994).

[4] Murphy, P. K., Waddell, R. L., and Heyler, G. A., "Image and Track Processing in Space (Part II)," in *Proc. AIAA Comput. Aerosp. 9 Conf.*, San Diego, CA, pp. 586–596 (1993).

[5] Waddell, R. L., Murphy, P. K., and Heyler, G. A., "Image and Track Processing in Space (Part I)," in *Proc. AIAA Comput. Aerosp. 9 Conf.*, San Diego, CA, pp. 576–585 (1993).

[6] *The VMEbus Specification*, VMEbus International Trade Association, Scottsdale, AZ (1987).

[7] Vadali, S. R., and Junkins, J. L., "Optimal Open-Loop and Stable Feedback Control of Rigid Spacecraft Attitude Maneuvers," *J. Astronaut. Sci.* **32**(2), 105–122 (1984).

ACKNOWLEDGMENTS: The testbed simulator was the result of the work of many people. The author would like to express his appreciation to L. Lee Pryor for technical and managerial assistance; Warren A. Frank for architectural contributions; Thomas W. LeFevere for architectural, electrical, and software engineering contributions; Barry N. Brown, John Rodell, and Kenneth E. Williams for analytical support; and Lydia A. Baker, Hugh C. Davey, Michael G. Mohler, and Michael L. Schaeffer for software engineering support. The MSX mission is sponsored by the Ballistic Missile Defense Organization. This work was supported under contract N00039-94-C-0001.

THE AUTHOR

DANIEL S. WILSON is a member of the Senior Professional Staff in APL's Space Department. He received a B.A. degree in mathematics in 1974 from Franklin and Marshall College in Pennsylvania. Mr. Wilson has worked on ground systems in support of numerous NASA and DoD satellites, including the Hubble Space Telescope and the Strategic Defense Initiative's Delta 181 and Delta 183 satellites. His work has been chiefly in the areas of spacecraft attitude determination and control. Mr. Wilson is currently working on a planning simulator to be used in support of NASA's Near Earth Asteroid Rendezvous. His e-mail address is Daniel.Wilson@jhuapl.edu.