AI Magazine Volume 15 Number 1 (1994) (© AAAI)

PI-IN-A-BOX
A Knowledge-Based System for Space Science Experimentation

Richard Frainier, Nicolas Groleau, Lyman Hazelton, Silvano Colombano, Michael Compton, Irving Statler, Peter Szolovits, and Laurence Young

■ The principal investigator (PI)-IN-A-BOX knowledge-based system helps astronauts perform science experiments in space. These experiments are typically costly to devise and build and often are difficult to perform. Further, the space laboratory environment is unique; ever changing; hectic; and, therefore, stressful. The environment requires quick, correct reactions to events over a wide range of experiments and disciplines, including ones distant from an astronaut's main science specialty. This environment suggests the use of advanced techniques for data collection, analysis, and decision making to maximize the value of the research performed. PI-IN-A-BOX aids astronauts with quick-look data collection, reduction, and analysis as well as equipment diagnosis and troubleshooting, procedural reminders, and suggestions for high-value departures from the preplanned experiment protocol. The astronauts have direct access to the system, which is hosted on a portable computer in the Space Lab module. The system is in use on the ground for mission training and was used in flight during the October 1993 space life sciences 2 (SLS-2) shuttle mission.

The critical resource in astronaut-tended flight experiments is time. The lack of time affects both preflight training for, and in-flight operation of, the experiment. This difficulty with time is currently true with the Space Shuttle Program and will persist with the advent of Space Station *Freedom* operations. Another key factor in space experimentation is the use of fixed experiment protocols. This major constraint severely limits the ability of an earth-bound scientist to change the course of an experiment even when the data and current situation clearly indicate that it would be scientifically more valuable to do so.

The principal investigator (PI)-IN-A-BOX knowledge-based system helps scientist-astronauts do better science in space given fairly severe time constraints and the need for them to work in areas outside their specialty. The goal is to help the astronaut become a scientific collaborator with the ground-based principal investigator who designed the experiment. The system facilitates increasing both (1) the level of astronaut-investigator collaboration and (2) the quality of the science performed in space by sharing observations with the astronaut about the quality and the importance of the data as they are being collected in flight. This system has the potential to fundamentally change the way crew members interact with ground-based investigators in the space station era.

In this article, we present a logical overview of the system; continue with a description of our first area of application; explain the technical details of the current implementation; and, finally, share some development philosophy used to manage this multiyear project. This system continues previous work described in Young et al. (1989), Haymann-Haber et al. (1989), and Frainier et al. (1990).

Functional Overview

The PI-IN-A-BOX system has several modules (figure 1). Together, they allow diagnosis of data-collection problems, hypothesis monitoring and formulation (limited to an analysis of interestingness in the initial system), determination and scheduling of the experiment's steps, and general-purpose help for the astronaut-user.

The *data-acquisition module* (DAM) and the *data-quality monitor* (DQM) acquire data from the experiment (displayed in real time),

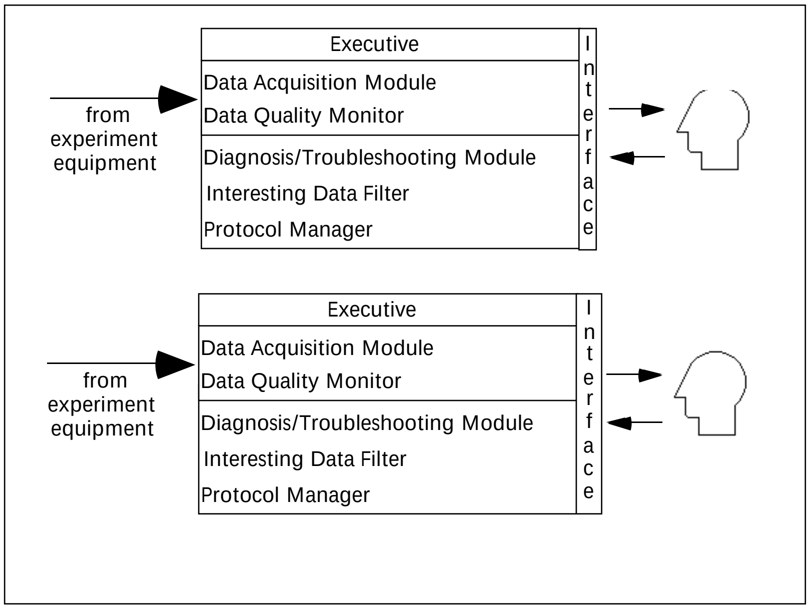


Figure 1. Major Modules.

extract parameters from the data, and interpret them. DQM also analyzes the data to determine quality with respect to the experimental apparatus and provides results to the *diagnosis and troubleshooting module* (DTM).

DTM helps the user isolate and recover from experiment data-collection problems. It suggests tests to isolate equipment faults. It also presents recommendations based on a computation of problem severity and possible recovery strategies with respect to remaining experiment session time (that is, the system can actually recommend that troubleshooting not be performed).

The *interesting data filter* (IDF) *module* monitors data from the experiment passed to it by DAM. IDF analyzes the data to determine their fit with preflight hypotheses. The fit can be either statistical or heuristic. Deviations are reported as interesting. These deviations are defined as needing confirmation, even if not part of the original fixed protocol. Once confirmed, they cease to be interesting.

The *protocol manager* (PM) *module* generates the best-possible experimental protocol for use at any given time in the experiment. It also displays information to, and accepts information from, the user. Corresponding to these two major functions, PM has two logical components: the scheduling component, called the *protocol suggester* (PS), and the human-computer interface (HCI) component, called the *session manager* (SM).

PS creates a new experimental protocol on request. A request from the user is likely when (1) there is a predicted shortage of time, possibly need to cut steps; (2) there is a predicted excess of time, possibly need or have opportunity to add steps; and (3) the experiment is giving interesting data, possibly need to substitute steps that will collect more information about the interesting data.

SM displays the current state of the experiment, including progress against the protocol and elapsed times as well as the history of other sessions occurring earlier in the mission. SM also displays procedural step-by-step checklists of experiment steps to be performed within the experiment by the user. SM updates the current protocol and elapsed times automatically and in response to user editing. SM also offers a scratch pad to allow users to record their observations. Users can perform the following actions using SM: (1) display the status of the current session, including a list of completed steps, the current step, and all pending steps as well as temporal information about the session and the current step; (2) display alternative protocols; (3) display the history of other sessions occurring earlier in the mission (this history is a list of all completed steps, including the experimental conditions used for each step); (4) display experiment checklists for a given experiment step; (5) edit the current protocol and all temporal information known and used by the system; (6) replace the current protocol with any of the other available protocols; (7) order a new set of protocols for consideration (by calling PS); and (8) initiate an equipment troubleshooting session.

Finally, an *executive module* controls module activation and focus of attention. It is also used to augment the operating system environment, if necessary, for a particular host central processing unit (CPU).

One other module is planned for future versions of the system: an *experiment suggester* (ES). ES will work in conjunction with IDF. Given a new hypothesis from IDF, ES will generate a set of tests that can be used to investigate the new hypothesis.

The First Domain: Vestibular Physiology

The system was first used in conjunction with a life sciences experiment in vestibular physiology known as the *rotating-dome experiment.* It was devised by Laurence Young, who is the director of the Man-Vehicle Laboratory at the Massachusetts Institute of Technology. The experiment is conducted by one crew member while another crew member acts as subject. The purpose of the experiment is to understand the human oculovestibular system and its relationship to the phenomenon of space motion sickness. During the experiment, the subject's visual field is filled by a dome. The dome, which contains a constellation of dots, is rotated at various speeds and directions. Gazing into the field of rotating dots induces *vection*, or the sensation that the subject, not the dots, is rotating. Voluntary and involuntary reactions to the vection are measured. There are typically two or three inflight sessions, each involving two to four crew members. An experiment session consists of equipment setup, equipment electric checkout, several experiment runs on the first subject, introduction of follow-on subjects, and equipment shutdown and stowage. The experiment has been flown in space three times in the past,[1] and it will fly again this year (1993). [The computer system flew with experiment on shuttle mission STS-58 in October 1993. – *Ed.*]

When performed in space, this experiment generates five analog data channels. Although the astronauts could monitor two of the five channels in real time on a small oscilloscope, they are not as expert as the ground-based investigator at validating or reacting to the data. Investigators can monitor all the channels, but the experiment data are subject to delays and outages as they are passed from the shuttle to relay satellites to and through the ground-based telecommunications system. Further, the investigators are limited in their ability to change the course of the experiment during an hour-long session. In fact, they are limited in their ability to change the course of the experiment during any remaining inflight experiment sessions.

PI-IN-A-BOX has direct access to all five data channels and performs quick-look validation and analysis in real time. The analysis is driven by heuristics compiled from the investigators, and the results are communicated to the astronaut performing the experiment. Thus, the system provides the astronaut with advice on how to best use the precious time allocated to the experiment. This advice is based on the preflight plan, modified by all the events that have occurred in flight, including a record of the experiment apparatus' performance, the list of crew members who have already performed the experiment, and indications made by the analyzed data about each of these subjects. Specific advice includes (1) recommendations on accepting a degradation of the experiment's data collection or spending time to repair a problem (if repair is elected, a step-by-step diagnosis-repair plan is offered to the user), (2) advice on the order in which to test subjects and the order of individual test steps for a given subject, and (3) alerts about analyzed data that appear to be of particularly high value (interesting data).

Other features allow the review of previously completed portions of the experiment and facilitate planning or replanning future experiment sessions. Finally, there are features that provide reminders on setting up and using the experiment apparatus.

Some Typical Scenarios

Let us assume that it is now two days after lift-off. The first session involves two astronauts who will alternate as subject and operator. The system has been set up for the first session, but there is a problem. An electric connection at the junction of two cables is damaged. The problem affects one of the two electromyography data channels. The experiment setup is on time, but the problem must be addressed. Further, there will be a voice and data outage (loss of signal [LOS]) commencing in 5 minutes that will last 20 minutes. Without PI-IN-A-BOX, the crew would typically attempt to repair the apparatus and then ask the ground crew for advice if the effort were unsuccessful. If LOS was in effect, the advice would not arrive until after the 20-minute blackout. With PI-IN-A-BOX, the crew could ask for a recommendation at any time. In this situation, even if the system had a repair procedure available, it would recommend not spending time repairing the low-priority channel but, instead, using this time to get data from the scheduled subjects.

Let us now assume that the astronauts declined the recommendation and spent 20 minutes at the repair. They are now part way through the experiment protocol and 15 minutes behind schedule. The astronauts realize that they are going to have to cut the experiment short. Without PI-IN-A-BOX, the crew would typically work as long as they could and then cut the last steps of the protocol. In this case, the entire second subject would be eliminated. With PI-IN-A-BOX, the crew could again ask for a recommendation. Here, the system would recommend cutting the last experiment condition for the first subject and the first experiment condition for the second subject and then continuing with the rest of the experiment. This recommendation accounts for the various setup times and the scientific importance of the experiment steps, realizing that a lengthy setup was

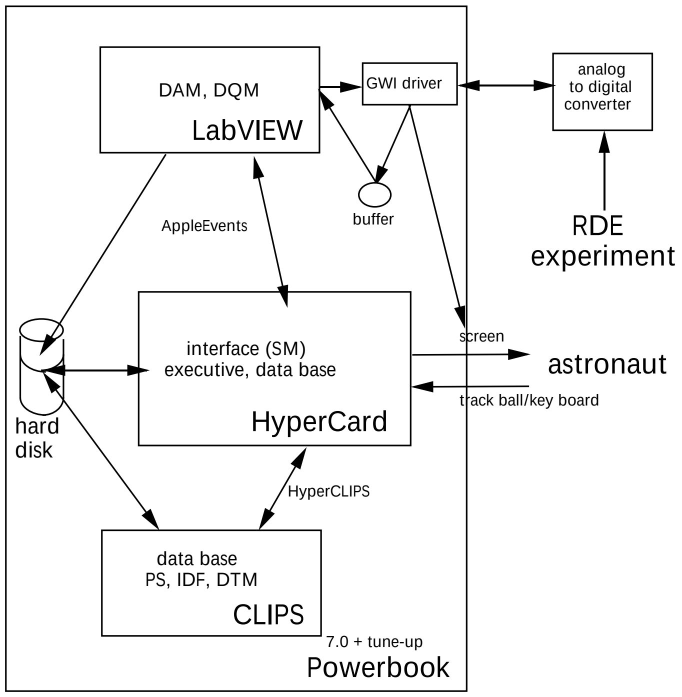


Figure 2. PI-IN-A-BOX *Physical-Logical Mapping.*

required for two low-priority steps. Eliminating both the setup and the steps saved 13 minutes and increased the coverage of the first session.[2]

Current Implementation

As fielded, the system runs on a single MACINTOSH POWERBOOK 170, which hosts all six modules. There is one other piece of hardware, an external (GW Instruments) analog-to-digital converter connected to the POWERBOOK's small computer system interface port. The POWERBOOK is fitted with 8 megabytes (MB) of random-access memory (RAM) (the maximum available on this model) and a 40-MB internal hard drive.[3] Three main software tools were used: CLIPS (C language intelligent production system), LABVIEW, and HYPERCARD.

CLIPS, available from COSMIC, the National Aeronautics and Space Administration's Software Technology Transfer Center, serves as the inference engine for the application. This OPS-style expert system shell is used by the system for schedule repair (PS), diagnosis (DTM), and symbolic analysis (IDF). Key factors leading to its use included low cost, availability of source code (facilitating tool extension and customization), excellent support, continuous upgrades, a strong user group, and widespread use.

LABVIEW, available from National Instruments, controls data collection, reduction, validation, and archival. It is a graphic data flow language (software from pictures) with a mouse-oriented developer's interface and excellent browsers. These features facilitated rapid development and code reuse. Another key feature was its support for data acquisition and analysis in a single package. Other key factors leading to its use included excellent support, continuous upgrades, availability of run-time version, a strong user group, and widespread use.

HYPERCARD (available from Apple Computer) is used for HCI, overall data management within the system, and module activation. Its procedural scripting language and part-whole object hierarchy facilitated the rapid-prototyping style of HCI construction essential to our system's development. Other key factors leading to its use included low cost, widespread use, and good technical support from Apple.

The three tools communicate with each other using APPLE EVENTS and HYPERCLIPS. HYPERCLIPS, a set of two simple one-way drivers, was developed by our team for HYPERCARD-CLIPS interapplication communication. APPLE EVENTS is an interapplication communication feature of the system 7 version of the MACINTOSH operating system. CLIPS and LABVIEW do not communicate directly with each other. The system currently uses all 8 MB of RAM. Virtual memory was tested and rejected because of the associated performance penalty. HYPERCARD and LABVIEW are each allocated 2500 kilobytes (KB) of RAM, CLIPS is allocated 1800 KB, and the remainder is used by the computer's operating system. The system files currently occupy about 11 MB of hard-disk storage.[4]

The mapping between hardware-software and each logical module of the system, as implemented for the rotating-dome experiment, is seen in figure 2. An obvious characteristic of our architecture is the need to make complicated technical trade-offs when building systems that integrate different tools and solve real problems.

HCI, called SM in our system, was built as two HYPERCARD stacks. One stack contains 26 cards. Fourteen of these cards serve as a persistent database (and are not viewed directly), and 12 are used for display. The second stack contains 16 cards that display pictures, line drawings, and real-time data traces.

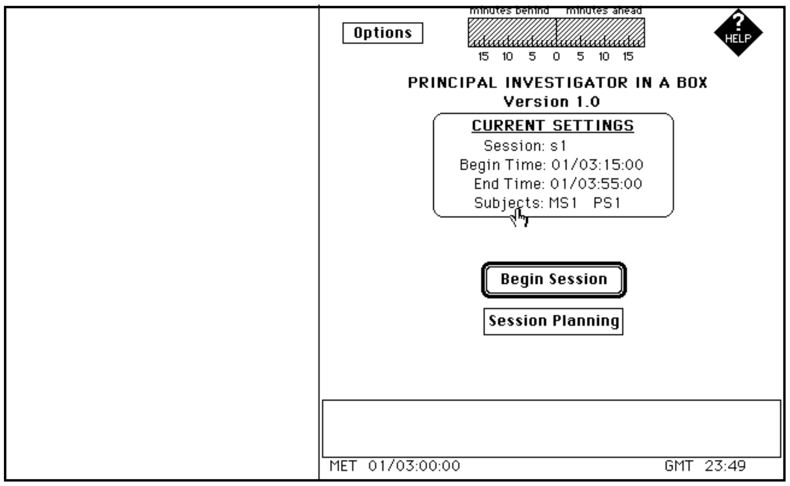
The database is divided between HYPERCARD and CLIPS. The HYPERCARD-resident portion consists of 14 cards. Most of these cards are used to store data from the experiment: One card is used for each step of the experiment that generates data. One card is used to display summary results of previous experiment sessions. Two cards are used to store data used globally by several of the modules. The CLIPS-resident portion consists of about 120 base facts. This number increases as the mission progresses, and experiment history is generated. During the typical operation of PS, about 200 facts are in the database at any one time (out of 120 base facts and 400 generated facts).

An Illustration of the System in Use

Assume that the operator has just set up the laptop computer and analog-to-digital converter in anticipation of an experiment session. The system automatically loads after the MACINTOSH powers up. Two minutes and 20 seconds later, startup is complete. The system has selected the next scheduled session and presents overview information, such as the scheduled start time, the end time, and subjects (figure 3). In this example, there are two subjects: mission specialist 1 (MS1) and payload specialist 1 (PS1). If the begin time or end time has recently changed, the user communicates the currently scheduled time to the system by clicking on the item to be updated. There is also the ability to similarly change or edit the subject list as well as other options. When the current settings are completely correct, the user proceeds by clicking on Begin Session.

The system prepares for the next action—a functional checkout of the experiment's electrical output. Notice that because the EMG (electromyogram) functional check should be performed, an illustration is displayed to help assure correct electrode placement (figure 4).

The LABVIEW-based DQM is used for this checkout and autocalibration of the experiment apparatus (these checks typically occur after equipment setup and before each new subject enters the experiment, although they can be performed optionally at any time during a session). The system displays a list of the signals to be checked, with an arrow pointing to the currently checked signal. As each 10-second check occurs, a real-time trace of the signal is displayed (figure 5). Five signal traces are displayed from top to bottom on the left side of the screen: Joystick, Biteboard, Right-EMG, Left-EMG, and Tachometer. The *tachometer* is a heartbeat trace that is only two pixels high. Note that in figure 5,

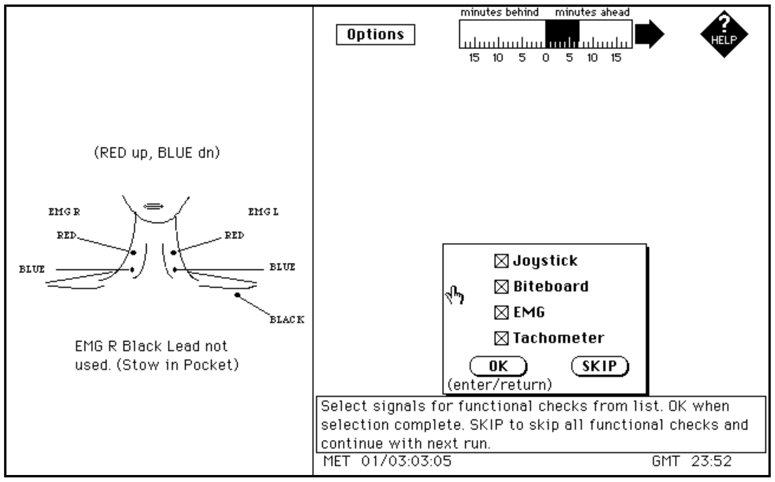


PRINCIPAL INVESTIGATOR IN A BOX
Version 1.0

CURRENT SETTINGS
Session: s1
Begin Time: 01/03:15:00
End Time: 01/03:55:00
Subjects: MS1 PS1

MET 01/03:00:00 GMT 23:49

Figure 3. Session Startup.

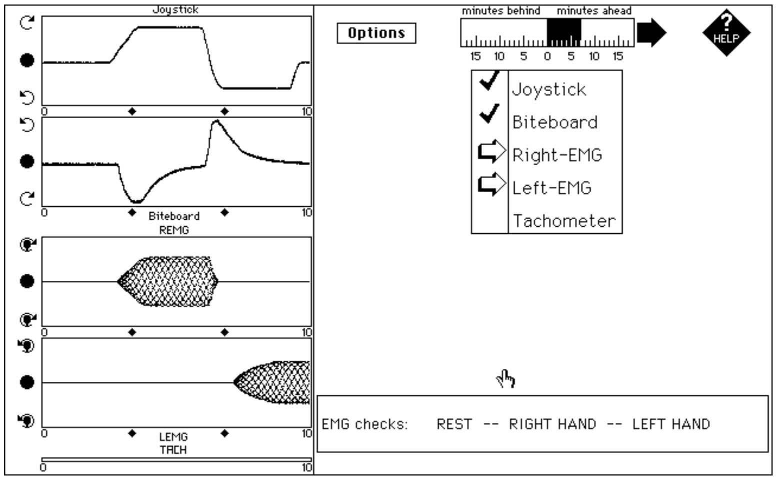


EMG R Black Lead not used. (Stow in Pocket)

Select signals for functional checks from list. OK when selection complete. SKIP to skip all functional checks and continue with next run.

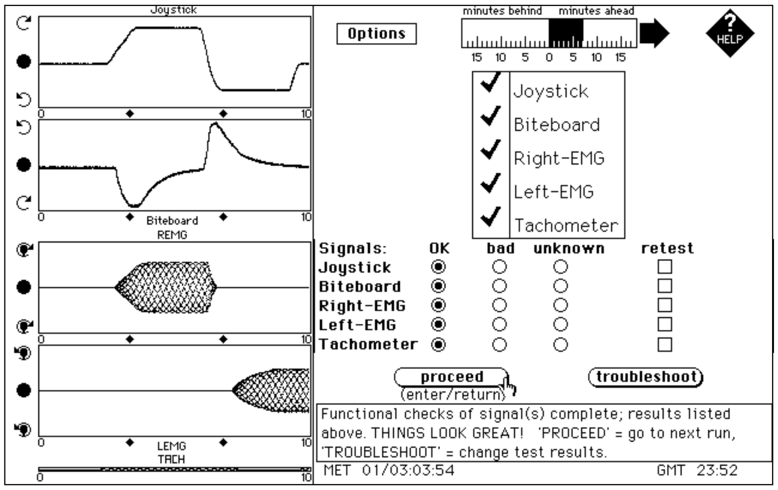
MET 01/03:03:05 GMT 23:52

Figure 4. Functional Check Preparation.



EMG checks: REST -- RIGHT HAND -- LEFT HAND

Figure 5. Functional Check Operation.



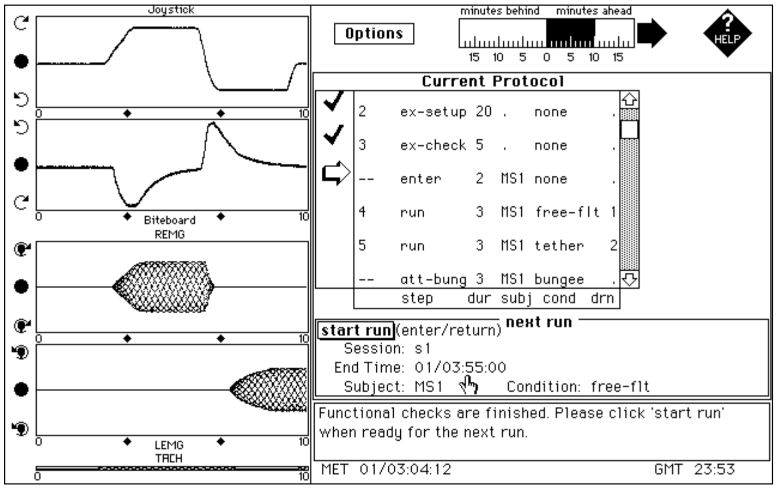
✓ Joystick
✓ Biteboard
✓ Right-EMG
✓ Left-EMG
✓ Tachometer

Signals:	OK	bad	unknown	retest
Joystick	◉	○	○	☐
Biteboard	◉	○	○	☐
Right-EMG	◉	○	○	☐
Left-EMG	◉	○	○	☐
Tachometer	◉	○	○	☐

Functional checks of signal(s) complete; results listed above. THINGS LOOK GREAT! 'PROCEED' = go to next run, 'TROUBLESHOOT' = change test results.

MET 01/03:03:54 GMT 23:52

Figure 6. Functional Check Results.



Current Protocol

```
✓  2    ex-setup 20  .    none      .
✓  3    ex-check 5   .    none      .
⇨  --   enter    2   MS1  none      .
   4    run      3   MS1  free-flt  1
   5    run      3   MS1  tether    2
   --   att-bung 3   MS1  bungee    .
        step     dur subj cond      drn
```

next run
Session: s1
End Time: 01/03:55:00
Subject: MS1 Condition: free-flt

Functional checks are finished. Please click 'start run' when ready for the next run.

MET 01/03:04:12 GMT 23:53

Figure 7. Main Session Display.

the Right-EMG and Left-EMG signals are displayed simultaneously as part of the single EMG check; so, there are two arrows indicating the current signal.

DQM performs one of the more interesting and challenging tasks in our system. It was not obvious at first how to interpret and react to a 10-second slice of analog data. It was quickly realized during integration testing that the experiment hardware electric specifications were not by themselves sufficient to determine if a signal channel was functioning correctly. We settled on partitioning the 10-second test into 3 segments: a rest-condition segment, a full–positive-deflection segment, and a full–negative-deflection segment. These segments are identified by applying mathematical filtering and differentiation operations. Specific parameters are then calculated for each segment. The analysis of these three segments is combined with known operational and faulty states of the hardware to finally determine the health of a given signal channel. When a signal is identified as operationally OK, the check also serves as a calibration of the channel and is used by DAM during its run-data analysis. Then there is a problem; DQM does not always indicate a unique fault. Resolution must then occur later in a troubleshooting session.

Upon completion of all requested checks, a summary of the results is displayed (figure 6). In this example, the experiment apparatus is functioning correctly. If a problem had been seen, then an automatic troubleshooting session (DTM) would have been initiated and a recommendation prepared for the user. If troubleshooting is pursued, a series of interactions would guide the repair effort.

Shown next is the main display screen (figure 7) with a summary of the current experiment protocol: (step) types, step (dur)ation in minutes, (subj)ect, experiment (cond)ition, and so on. An arrow indicates the step currently being performed; check marks indicate completed steps; and, finally, pending steps are listed below the current step. In this case, MS1 is entering the dome to be tested in the free-float condition. This information is echoed in the Next Run display area just below the current protocol listing and can be changed or edited by darkening the item in need of update.

If the user wants to check the procedures associated with the current, completed, or upcoming step, he/she merely clicks on the step in the Current Protocol window. A copy of the checklists and procedures is then displayed for review. These checklists exist as paper documents. They can easily be converted to PICT files, which can then be displayed by HYPERCARD. Procedures controlling the use of the experiment apparatus are managed with a configuration-control system. Changes to procedures are made by an engineering change order (ECO). These changes must be incorporated into the system. The ease of conversion to PICT files greatly facilitates maintenance of the procedures within the PI-IN-A-BOX system.

Real-Time Data Collection and Analysis

PI-IN-A-BOX is a real-time knowledge-based system. It must receive, analyze, and then act on the data that are generated by the experiment

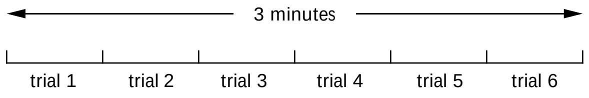


Figure 8. A Rotating-Dome Experiment Run.

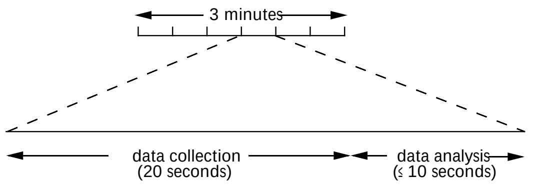
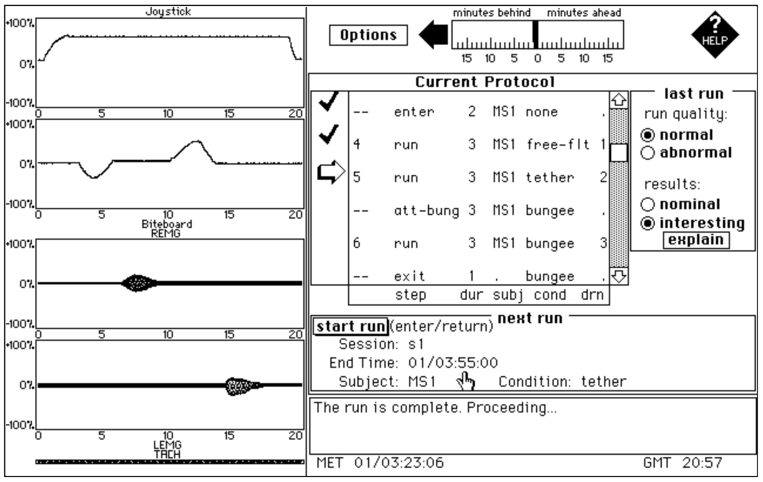


Figure 9. A Rotating-Dome Experiment Trial.



Current Protocol

	step	dur	subj	cond	drn
✓ --	enter	2	MS1	none	.
✓ 4	run	3	MS1	free-flt	1
⇨ 5	run	3	MS1	tether	2
--	att-bung	3	MS1	bungee	.
6	run	3	MS1	bungee	3
--	exit	1	.	bungee	.

last run
run quality:
● **normal**
○ **abnormal**
results:
○ **nominal**
● **interesting**
explain

next run
start run (enter/return)
Session: s1
End Time: 01/03:55:00
Subject: MS1   Condition: tether

The run is complete. Proceeding...

MET 01/03:23:06   GMT 20:57

Figure 10. Postrun Display.

quickly enough to be of use to the user. Each data-producing step in the vestibular physiology experiment consists of six 30-second data-producing trials (figure 8).

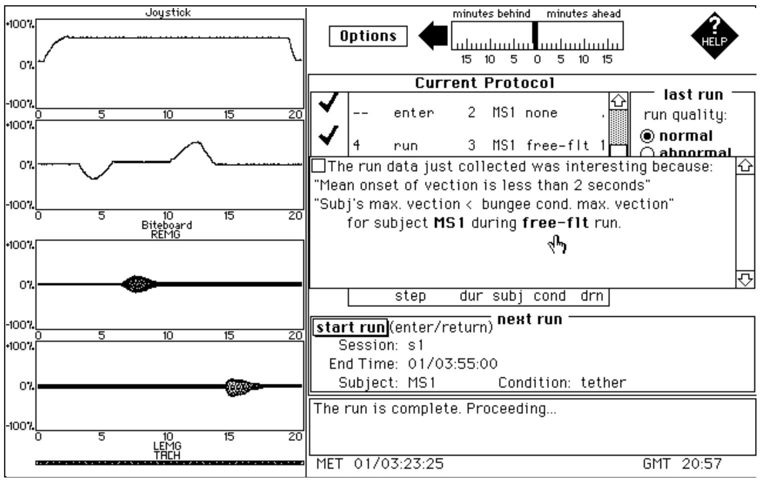
Each trial has 20 seconds of data gathering and real-time display followed by 10 seconds of rest (figure 9). The 20-second data-collection and data-display period is controlled by a LABVIEW-compatible driver supplied by GW Instruments. All 5 data channels are sampled at a rate of 225 hertz. During this time, the driver used for data acquisition monopolizes the POWERBOOK (even the mouse is not tracked). The following 10 seconds are shared by LABVIEW and HYPERCARD. DAM performs data analysis, reduction, parameter extraction, and archival. It also communicates results to HYPERCARD for use in alerts and for postrun analysis (if DAM determines that a critical signal has malfunctioned during a trial, then the run might be halted for troubleshooting).

The system performs several actions after a run of six trials. It first checks that at least five of the six trials in a run have been completed successfully. If so, the run is labeled "nominal." It then checks the data for agreement with the current hypotheses (IDF). The IDF module currently consists of about three dozen CLIPS rules. Heuristics currently used to determine interestingness focus on the presence, onset latency, and intensity of vection. These domain-specific heuristics include the following: (1) the onset of vection is interesting if it is consistently less than two seconds; (2) early in the flight, maximum vection is interesting if it is consistently greater than 90 percent; (3) the number of dropouts experienced by a subject is interesting if it is consistently low (0) under tactile conditions; and (4) an experiment run is interesting if maximum vection under tactile conditions is consistently greater than maximum vection under nontactile conditions.

After the IDF module runs, the system prepares for the next run. Summary information is displayed for the user to review. A possible postrun display is seen in figure 10. In this case, the run was normal, and the data were interesting. An explanation of the interestingness is available for review if desired. In this example, the run was interesting for two reasons: The sensation of vection began quickly, but it had a low maximum compared to that predicted by the results of the same subject's earlier runs (see figure 11).

Between-Run Options

The user can invoke a variety of options (available from the Options pull-down menu) at this time. One option is to exploit the observed interestingness by asking the system for a better plan for session completion (experiment protocol). These protocols take into account the time remaining in the current session as well as which subjects were tested in previous in-flight sessions and what their results were. One of the two resulting suggestions is the *proposed protocol*. This protocol observes session time constraints. The other suggestion is the *optimal protocol*. This protocol relaxes the time constraint slightly to offer a focused plan with minimal negative impact on the mission time line. The time needed to suggest these new protocols is usually less than 30 seconds. The PS module con-



```
✓  --   enter   2  MS1  none
✓  4    run     3  MS1  free-flt  1

☐ The run data just collected was interesting because:
"Mean onset of vection is less than 2 seconds"
"Subj's max. vection <  bungee cond. max. vection"
     for subject MS1 during free-flt run.

  Session:  s1
End Time:  01/03:55:00
  Subject:  MS1        Condition:  tether

The run is complete. Proceeding...

MET  01/03:23:25                         GMT  20:57
```

Figure 11. A Brief Explanation of Interesting Data.

trolling the content of the two suggested protocols consists of about 200 rules.

The experiment protocol is conceptualized as follows: There are a number of experiment sessions in a mission. Each session is conducted in accordance with its protocol. The protocol consists of a number of blocks. There is one block for each subject, a block for experiment setup, and a block for experiment stowage. Each block has a number of steps. The setup and store blocks are straightforward, but the optimum subject block order (and run ordering within a given subject block) depends on a complex interpretation of previous mission history. The first task is to determine which experiment steps should be performed and assign a (science) priority to each. The block ordering is determined next, after which, step ordering within a block is determined. Following the step ordering, minor setup tasks are inserted. Finally, the result is checked against current time constraints.

Heuristics currently used to determine new protocols include the following:

Get at least some data on every subject.

Complete the data collection using a full set of experiment conditions on at least one subject.

Each subject has a set of run conditions to be performed. Some of these conditions might be performed multiple times. Always give a subject's as-yet-untested run conditions priority over duplicate run conditions that have been tested at least once.

Do not schedule bungee runs if the joystick signal is bad.

Schedule runs with similar experiment conditions together.

If the bungee is set up, schedule bungee runs first; if the bungee is not set up, schedule runs that do not need the bungee first.

After some data have been obtained, if there are interesting data to be confirmed on a subject, then give preference to this subject.

If a subject is currently set up in the test apparatus, schedule the remaining steps for this subject first.

If no subject is set up, give first preference to a subject with unfinished tests from an earlier session, and give next preference to a subject who was giving interesting data.

If there were interesting data on a subject and an experiment condition from the current session, and the subject and the condition have not been repeated, then rerun the subject and the condition.

If there were interesting data on a subject and an experiment condition from the current session, and this condition was again run today but was again not found to be interesting, then run the subject in this condition one more time.

If there were interesting data on a subject and an experiment condition from the current session, and this subject and condition were again run today, but the condition was not run on another subject, then run the condition on another subject.

If there were interesting data to be confirmed on a subject and an experiment condition from a previous session, and this condition was not run today, then run the subject in this condition.

If there were interesting data to be confirmed on a subject and an experiment condition from a previous session, and this condition was run today but was again not found to be interesting, then run the subject in this condition one more time.

If more time is needed than is currently allowed, cut the least desirable (from a science standpoint) step.

As this list suggests, it is difficult for an astronaut (or even an investigator) under time pressure to keep these heuristics, together with their relative priority, in short-term memory and apply them correctly when rescheduling a session protocol.

Troubleshooting

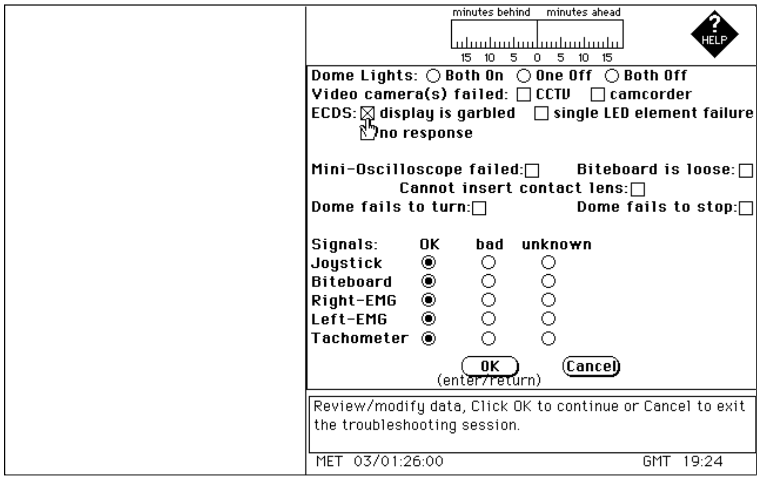
Another option is to invoke a manual troubleshooting session (DTM). The user has a chance to indicate a variety of observed problems, as seen in figure 12. In this case, the user has indicated that the ECDS (experiment-control data system) display associated with another on-board data-gathering computer is garbled. The system responds by recommending reinitializing ECDS (figure 13) and displays the relevant portion of the control panel to aid recall (figure 14). A further example is seen in figures 15 and 16. Here, a bad EMG functional check leads to an investigation of the EMG connectors and, later, to a replacement of the EMG amplifier batteries. Additional functional checks are occasionally part of the troubleshooting process.

The DTM module consists of about 200 CLIPS rules. The CLIPS rules work with procedural code in HYPERCARD to guide the user through troubleshooting and repair. The overall flow involves the formulation of a repair recommendation based on the current experimental situation (the recommendation could entail forgoing a lengthy repair, for example). If troubleshooting continues, the system traverses a graph step by step until either the problem is repaired, the user terminates the session, or the system has no further advice to offer. Steps include displaying pictures (figure 14), line drawings (figure 15), and repair instructions (figure 16); conducting functional checks; and making observations for the system (figure 15).

Several other between-run options can be invoked from the Options pull-down menu. These options include the use of a notepad, the review of the history of previously completed sessions, and the use of a manual editing facility for the current protocol. Finally, as mentioned earlier, the procedures associated with any step can be brought up for review.

System Construction Philosophy

The system makes maximum use of commercial off-the-shelf software to leverage programming effort and avoid reinventing the



Dome Lights: ○ Both On ○ One Off ○ Both Off
Video camera(s) failed: ☐ CCTV ☐ camcorder
ECDS: ☒ display is garbled ☐ single LED element failure
☐ no response

Mini-Oscilloscope failed: ☐ Biteboard is loose: ☐
Cannot insert contact lens: ☐
Dome fails to turn: ☐ Dome fails to stop: ☐

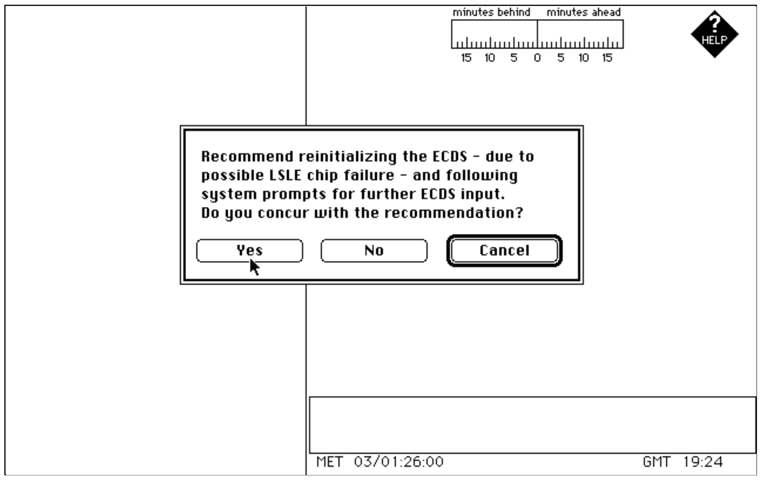
Signals:	OK	bad	unknown
Joystick	●	○	○
Biteboard	●	○	○
Right-EMG	●	○	○
Left-EMG	●	○	○
Tachometer	●	○	○

OK (enter/return) Cancel

Review/modify data, Click OK to continue or Cancel to exit the troubleshooting session.

MET 03/01:26:00 GMT 19:24

Figure 12. Selecting Manual Troubleshooting.

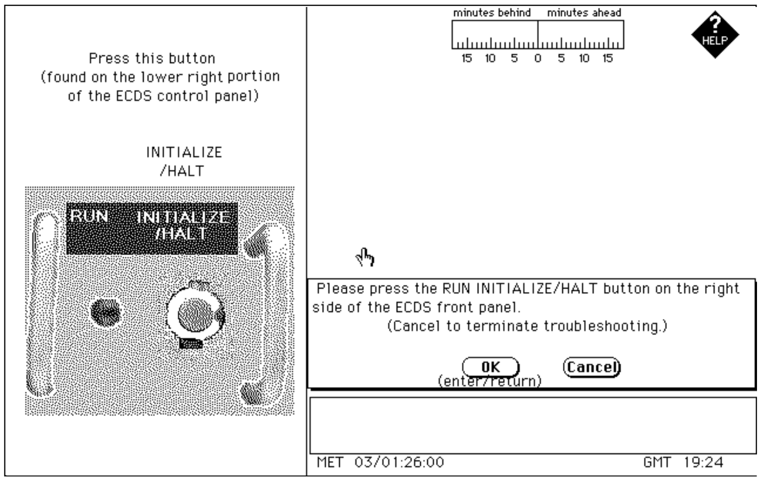


Recommend reinitializing the ECDS - due to possible LSLE chip failure - and following system prompts for further ECDS input.
Do you concur with the recommendation?

Yes No Cancel

MET 03/01:26:00 GMT 19:24

Figure 13. DTM Recommendations.



Press this button (found on the lower right portion of the ECDS control panel)

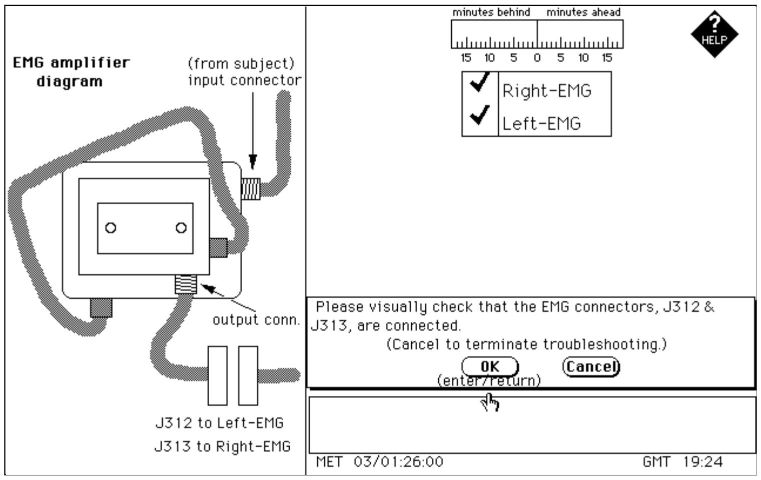
INITIALIZE /HALT

Please press the RUN INITIALIZE/HALT button on the right side of the ECDS front panel.
(Cancel to terminate troubleshooting.)

OK (enter/return) Cancel

MET 03/01:26:00 GMT 19:24

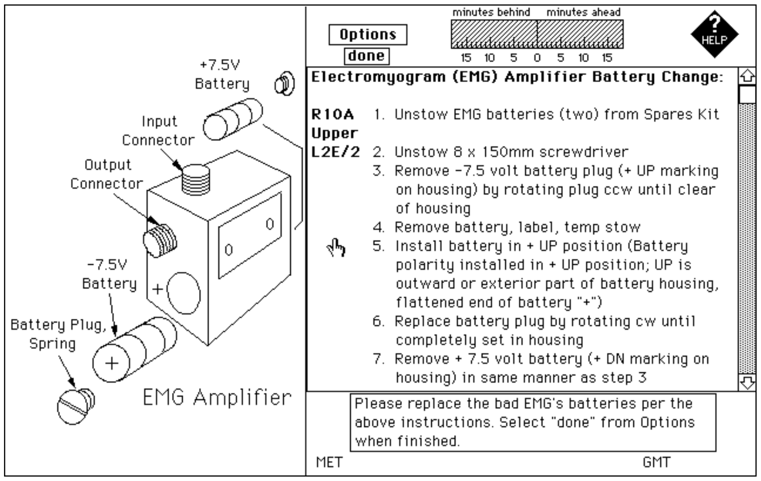
Figure 14. DTM Instructions.



Please visually check that the EMG connectors, J312 & J313, are connected.
(Cancel to terminate troubleshooting.)

MET 03/01:26:00 GMT 19:24

Figure 15. DTM Requests for Information.



Electromyogram (EMG) Amplifier Battery Change:

R10A Upper L2E/2
1. Unstow EMG batteries (two) from Spares Kit
2. Unstow 8 x 150mm screwdriver
3. Remove -7.5 volt battery plug (+ UP marking on housing) by rotating plug ccw until clear of housing
4. Remove battery, label, temp stow
5. Install battery in + UP position (Battery polarity installed in + UP position; UP is outward or exterior part of battery housing, flattened end of battery "+")
6. Replace battery plug by rotating cw until completely set in housing
7. Remove + 7.5 volt battery (+ DN marking on housing) in same manner as step 3

Please replace the bad EMG's batteries per the above instructions. Select "done" from Options when finished.

Figure 16. DTM Repair Instruction.

wheel. Some custom coding was done when necessary. There are clear advantages to this arrangement: Development started against a target machine that was not yet available, continued on a first-generation offering, and was delivered on a second-generation machine. Thus, users have the high performance of (and excitement of using) current hardware, and developers are able to take advantage of the increased hardware and software capabilities of the host machine. This approach to system development is important because application software and the programming paradigms it is based on change more slowly than operating systems, which, in turn, change more slowly than CPUs;[5] so, by developing from the application software tool downward toward CPU, we can deliver up-to-date hardware and software. Perhaps just as important, this approach has allowed better maintainability and a cheap upgrade path by leveraging the efforts of the technical staff members at Apple, Claris, National Instruments, and the Johnson Space Center Software Technology Branch. For example, if we had to incorporate a complete interapplication communications facility between CLIPS, HYPERCARD, and LABVIEW, it would have been more expensive and less robust, and it would probably have locked us out of routine software upgrades. Instead, we experimented with incremental enhancements that we could discard or retain as more capable software became available or as the system's operational requirements changed. Another maintenance win occurs when team members occasionally and inevitably leave: The learning process is quicker for the new member who is already familiar with widely used products such as CLIPS, LABVIEW, and HYPERCARD.

A modified version of the spiral model of software development was followed for most of the major modules. This style of software development is a *requirement-discovery style*, where a rough specification is used to guide knowledge engineering and rapidly construct a prototype. The prototype is then demonstrated internally (development team and end users) and externally (colleagues and upper management). Comments are then fed back into the specification. This process leads to a follow-on version of the module after another iteration of knowledge engineering and rapid prototyping. We believe this approach to be superior to the waterfall model of software development for knowledge-based systems for two major reasons. First, there is no reasonable way to determine a specification that is detailed enough to guide a multiyear effort. It was only after coding, demonstrating, and using prototypes that differences between how the task was described and how the task was performed were resolved.[6] Second, the knowledge in the knowledge-based system is not static. As a result of the SLS-1 shuttle mission in June 1991, significant changes were made not only in the experiment and its approach but also in our conception of what the PI-IN-A-BOX system could best do. Our approach is not perfect. One potential problem is determining when to stop the development cycle. In our case, firm milestones were associated with the flight schedule that provided the necessary constraints. Another potential problem is maintenance of requirements and test documents. Because each development cycle modifies the system's requirements, these changes must be captured and reflected in the test plan and other documents. We found these issues to be minor frustrations compared to the benefits of our software development scheme.

An interesting approach to validating a module is *cross-prototyping,* which was used on one of the modules. One team member built a prototype of the module based on the official production system (OPS) style of representation. When this team member left the project, maintenance and extension of the module was given to a new member. The new member was initially much more familiar with object-frame representations than with the OPS paradigm. After understanding the purpose and current requirements of the module, a new prototype was quickly built in IntelliCorp's KEE and then translated to the PARMENIDES frame tool and FRULEKIT OPS-style rule system (both from Carnegie Mellon University) running over MACINTOSH Allegro Common Lisp. This approach provided performance comparisons between the two implementations and led to the discovery and elimination of several subtle bugs in the module.

Benefits of the Knowledge-Based System

The main benefit of this system is the maximization (or, certainly, an increase in) the scientific quality of data from experiments performed by humans in space. This benefit, in turn, increases the value of the research performed. The increased value comes from increased crew productivity. This increased productivity has two dimensions: First, time is not spent on unproductive tasks after equipment failures. Second, reactions to the scientific consequences of already-gathered data are improved. Although caution should be exercised in generating dollar figures, we estimate savings of $6,000 for each astronaut science–hour (based on a 20-percent crew productivity increase from operational use and a conservative figure of $30,000 an hour of crew time on the space station).

Future Directions

Use of the system in support of space shuttle mission SLS-2 will continue through 1993. The team is also working to identify follow-on experiments for future missions. After one or two experiments, we hope to know enough to create a general-purpose tool to aid science experiments in space or, indeed, in any situation where quick-look analysis can be used to guide the focus of attention for the remainder of a limited scientific observation period. Examples under consideration include other life science experiments, materials science experiments, atmospheric studies, and plasma physics. Although the system yields maximum benefit when applied to a particular experiment, there is value in adding just the DAM, DQM, and DTM modules to major equipment or facilities on the space station (for example, a centrifuge and gas-grain simulation facility). The addition of these three modules without reference to a particular set of experiments would still allow for the use of a general-purpose monitoring, diagnosis, and repair system over the life (20+ years) of the orbiting equipment. In this case, a repair recommendation is weakened to the extent that experiment-specific data-collection heuristics and history are not available.

Alternative Approaches

There are alternatives to the in situ knowledge-based approach described in this article. A more traditional approach would be to add more ground support people at one of the existing sites with voice and data channel links. They would do the analysis and present results and recommendations to the scientist. The traditional AI approach would be to add LispM (in place of people) in the back room of shuttle operations with a data link to the scientist. Both approaches are critically hindered by the key problem of any ground-based solution: delays and outages in receiving experiment data and transmitting solutions to the crew. We feel that space experimentation requires careful consideration of the trade-offs between these approaches and on-board intelligence.

Conceptually related work is associated with a space shuttle–based cryogenic experiment—SHOOT (superfluid helium on-orbit transfer). Two systems, AFDEX and CMS, facilitate the conduct of SHOOT (Raymond 1989; Shapiro and Robinson 1989). The AFDEX rule-based system is designed to provide intelligent process control, diagnosis, and error recovery. This system is hosted on a 80386-based GRID laptop and will be located on the space shuttle's aft flight deck. The AFDEX system software is a combination of CLIPS and C code. AFDEX is capable of autonomous (closed-loop) control of the experiment but is planned for use under astronaut control. The command and monitoring system (CMS) is designed to provide near–real-time monitoring and control of the SHOOT experiment from earth by the investigator. CMS is hosted on a MACINTOSH II computer. The system software is written in Apple's MPW(C). It is anticipated that most (roughly 80 percent) of the time, the SHOOT experiment will be under ground control and the remainder under astronaut control.

Conclusion

PI-IN-A-BOX is a unique knowledge-based system for aiding scientific experimentation in space. We used AI (symbolic reasoning), formerly AI (advanced object-oriented HCI), and non-AI (data acquisition and analysis) techniques to build a useful system. Our framework will be expanded and generalized into a tool to aid the investigations that will occur on Space Station *Freedom* in the latter part of this decade.

Acknowledgments

We would like to thank the other members of the PI-IN-A-BOX team, past and present, for their efforts, especially Jeffery Shapiro, Guido Haymann-Haber, Chih Chao Lam, and Donald Kaiser. Thanks as well for the technical assistance provided by the NASA Johnson Space Center Software Technology Branch, National Instruments, and Apple Computer, especially Gary Reily, Chris Culbert, Rob Dye, John Hanks, John Fowler, Don Powers, Dave Nagel, and Tom Mayer. Thanks also to our reviewers for their helpful comments. Last but not least, thanks to NASA, especially Peter Friedland, Mark Gersh, Henry Lum, and Gregg Sweitek, for management support.

This work was cofunded by the NASA Space Station *Freedom* Advanced Development Program and the Office of Aeronautics, Exploration, and Technology AI Program, to whom we are grateful.

Notes

1. The rotating-dome experiment flew on space shuttle–hosted Space Lab missions SL-1 (1983), D-1 (1985), and SLS-1 (1991).

2. The coverage heuristic states that it is better to have at least some data on each subject than to have a full set of runs on one subject with no data on another.

3. All materials and equipment used on and in the space shuttle require a qualification (analysis and test for safety, reliability, and so on). These tests need to be performed in advance of the mission's critical design review. It was not possible to qualify more recent (and more capable) versions of the POWERBOOK for our target mission.

4. The system files include documentation text files, rule files, permanent data files, other application files, and tool image files.

5. Consider that CLIPS has been around since 1986 (or earlier), HYPERCARD and LABVIEW were first released in 1988, MACINTOSH system 7 was released in 1991, and POWERBOOKS became available in quantity in 1992. The pace of improvement of hardware-based computing power over the last 20 years has been astonishing. Looking ahead to operations on a space station with a projected life of 40 years or more, it is critical not to start with hardware systems that will be obsolete before they are launched.

6. These differences included both the ground-based scientists and the astronauts. In both cases, actual task-performance style was more conservative than the idealized version articulated for the system developers. We feel that the key to a really useful and valuable system lies in aiding actual task performance.

References

Frainier, R.; Groleau, N.; Bhatnagar, R.; Lam, C.; Compton, M.; Colombano, S.; Lai, S.; Szolovits, P.; Manahan, M.; Statler, I.; and Young, L. 1990. A Comparison of CLIPS- and Lisp-Based Approaches to the Development of a Real-Time Expert System. In Proceedings of the First CLIPS Conference, 320–333. Houston, Tex.: National Aeronautics and Space Administration.

Haymann-Haber, G.; Colombano, S. P.; Groleau, N.; Rosenthal, D.; Szolovits, P.; Young, L. R. 1989. An Expert System to Advise Astronauts during Experiments: The Protocol Manager Module. In Proceedings of the Conference on Space Automation and Robotics, Houston, Texas, August.

Raymond, E. 1989. Knowledge-Based Process Control and Diagnostics for Orbital Cryogen Transfer. In Proceedings of the Computers in Aerospace VII Conference, 276–280. Washington, D.C.: American Institute of Aeronautics and Astronautics.

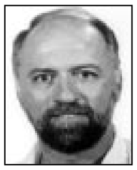
Shapiro, J., and Robinson, F. 1989. Interactive Remote Control for an STS-Based Superfluid Helium Transfer Demonstration. In Proceedings of the Computers in Aerospace VII Conference, 696–704. Washington, D.C.: American Institute of Aeronautics and Astronautics.

Young, L. R.; Colombano, S. P.; Haymann-Haber, G.; Groleau, N.; Szolovits, P.; and Rosenthal, D. 1989. An Expert System to Advise Astronauts during Experiments (presented at the Fortieth Congress of the International Astronautical Federation), IAF-89-033, International Astronautical Federation, Paris, France.

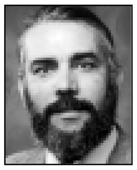
Richard Frainier is a computer scientist for RECOM Technologies, Inc., at the NASA Ames Artificial Intelligence Research Branch in Mountain View, California. His work centers on the engineering of real-time knowledge-based systems and issues

in human-computer interaction.

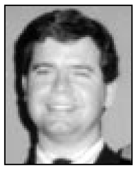
Nicolas Groleau is conducting research in the Artificial Intelligence Research Branch at the NASA Ames Research Center. He obtained his Ph.D. in computer science and civil and environmental engineering from the Massachusetts Institute of Technology in 1992. His current interests include automated scientific discovery, artificial neural net–based time-series prediction, and autonomous scientific research advisers.



Lyman Hazelton is a research scientist in the Center for Space Research at the Massachusetts Institute of Technology. His recent investigations have been on AI-based scheduling and control. He is particularly interested in hybrid knowledge-based systems, knowledge representation, and temporal reasoning. His expertise includes computer science, aeronautical engineering, physics, and electronics.



Silvano Colombano is the project manager for the Astronaut Science Adviser Project. He received an M.A. in physics and a Ph.D. in biophysical sciences from the State University of New York at Buffalo. He has worked in the area of computer science since 1977, first at the Roswell Park Memorial Institute in Buffalo, New York, and then at the NASA Ames Research Center in the Artificial Intelligence Research Branch (since 1988). He began the development work on the astronaut science adviser and has managed the project since 1989.



Michael Compton is a computer scientist in the Artificial Intelligence Research Branch at the NASA Ames Research Center (as a contractor through RECOM Technologies, Inc.), where he develops expert systems for a variety of NASA applications, including space experimentation and work-flow automation. He received a B.S. and an M.S. in computer science from the University of San Francisco and has been active in applied AI since 1983.

Irving Statler is chief of the Aerospace Human Factors Research Division at the NASA Ames Research Center. He manages a program of research ranging from fundamental psychophysiological research and human-performance modeling to human behavior and performance as elements of complex human-machine systems. His personal interest is in the role of the subconscious in cognition.

Peter Szolovits is professor of computer science and engineering at the Massachusetts Institute of Technology (MIT) and head of the Clinical Decision-Making Group within the MIT Laboratory for Computer Science. Szolovits's research centers on the application of AI methods to problems of medical decision making. He has worked on problems of diagnosis of kidney diseases, therapy planning, execution and monitoring for various medical conditions, and computational aspects of genetic counseling. His interests in AI include knowledge representation, qualitative reasoning, and probabilistic inference.

Laurence Young is a payload specialist in training for a space life sciences orbital flight of Space Lab, on leave from his position as professor of aeronautics and astronautics at the Massachusetts Institute of Technology. He has been the principal investigator on four Space Lab missions, beginning in 1976, all for experiments on vestibular effects in space. He conceived of the PI-IN-A-BOX project as a way of using expert systems to assist in the design and conduct of flight experiments and has worked with the NASA Ames Research Center in bringing it to a flight experiment. He is a member of the National Academy of Engineering and the Institute of Medicine of the National Academy of Sciences.