Onboard Autonomy on the Three Corner Sat Mission

S. Chien, B. Engelhardt, R. Knight, G. Rabideau, R. Sherwood
Jet Propulsion Laboratory
California Institute of Technology

E. Hansen, A. Ortiviz, C. Wilklow, S. Wichman
University of Colorado, Space Grant College

Keywords: planning, autonomy, scheduling, robust execution, constellation, teamwork, coordination

Abstract

Three Corner Sat (3CS) is a mission of 3 university nanosatellites scheduled for launch in late 2002. The 3CS mission will utilize significant autonomy to perform onboard science data validation and replanning. The 3CS mission will use onboard science data validation, responsive replanning, robust execution, and anomaly detection based on multiple models. Demonstration of these capabilities in a flight environment will open up tremendous new opportunities in space-borne science and space exploration that would be unreachable without this technology.

1 Introduction

The Three Corner Sat (3CS) mission is a University nanosat mission consisting of three coordinated satellites. 3CS will be launched from the Space Shuttle cargo bay via the Air Force Research Laboratory's Multi-Satellite Deployment System (MSDS) in late 2002, and will use extensive autonomous flight software. This software will enable significantly increased science, relating to the 3CS science goals of imaging earthborne clouds from low earth orbit. The 3CS mission also represents significant outreach, as the mission and spacecraft are designed, built and operated almost entirely by students at the University of Colorado, Arizona State University, and New Mexico State University.

This paper focuses on the onboard autonomy capability that will be used for the 3CS mission. The 3CS autonomy capability (threecornersat.jpl.nasa.gov) includes: the SCL robust execution system, the CASPER continuous planning system, an onboard science data validation module, the SELMON anomaly detection and isolation system, and a basic spacecraft coordination package.

The first element of 3CS autonomy is the Spacecraft Command Language (SCL) used for robust execution. In 1997, SCL was flown by CSGC on the DATA-CHASER shuttle payload [Chien et al. 1999]. SCL has also flown onboard several missions including Clementine and FUSE (see www.sclrules.com). SCL provides a rule and script-based procedural language for encoding robust execution procedures as well as basic coordination constructs such as locking, blocking, and run-time resource management. SCL will be used to demonstrate low-level autonomy including: event-driven execution, local retries, low-level fault responses, and command validation.

The second element of 3CS autonomy is the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) [Chien et al. 2000] (casper.jpl.nasa.gov) onboard planning software. CASPER will demonstrate onboard continuous planning to enable the 3CS constellation to respond to mission anomalies, mission opportunities, as well as onboard evaluation of science data. Onboard planning enables integration of the planning process with execution to provide feedback.

The third element of autonomy onboard 3CS is science data validation. Because the 3CS spacecraft will be tumbling, many science images may be of outer space, the Sun, or with the Earth in only a small portion of the image. Onboard data validation will use heuristic methods to estimate the utility of science images. The CASPER onboard planner will then use these utility scores in developing future operations plans. This science information will be used to discard images of lowest utility, prioritizing downlink to send the best images first, and making plans to acquire more science images if storage and other operations constraints allow.

The fourth element of 3CS autonomy is the SELMON monitoring system. SELMON uses empirically derived error bounds to enable context-

sensitive anomaly detection. SELMON will be used to derive error limits and anomaly detection to assist in monitoring the performance of the 3CS constellation.

Finally, 3CS will be flying basic spacecraft coordination software. This software will resolve the leadership election problem, in which reliably one spacecraft is selected as the lead spacecraft that will perform centralized control of the three spacecraft constellation.

The remainder of this paper is organized as follows. First we describe the basic elements of the Three Corner Sat Mission. Next we describe the onboard science data validation algorithms. We then describe the robust execution (SCL) and onboard replanning (CASPER) capabilities. Next the SELMON onboard monitoring and teamwork capabilities are described. Finally, we describe mission status, related work, and future work.

2 The Three Corner Sat Constellation Mission

The Three Corner Sat constellation will consist of three satellites flying in a formation that degrades as the mission continues. The spacecraft will be deployed from the Space Shuttle in September 2002. The mission length is expected to be approximately three months, dependent on atmospheric drag that will eventually cause the satellites to de-orbit.

Each of the three spacecraft has as its primary science instrument two fixed cameras. These cameras will be used to take images of the earth with the intention of capturing images of clouds. In order to simplify the mission and costs, the spacecraft will be tumbling (e.g., not attitude stabilized or controlled).

The 3CS spacecraft are lightweight nanosatellites each weighing approximately 15 kg. The exterior envelope of the structure is a six-sided disk structure consisting of tubular supports and machined end caps to hold the bulk of the loading. Figure 1 shows the 3 spacecraft stack before deployment and Figure 2 shows a single spacecraft in the deployed state.

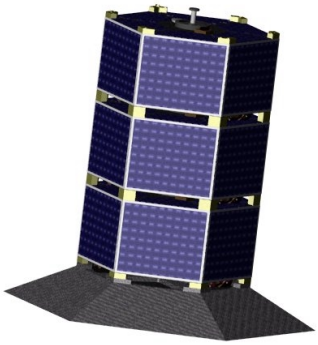

Figure 1: 3-Spacecraft Stack Prior to Deployment

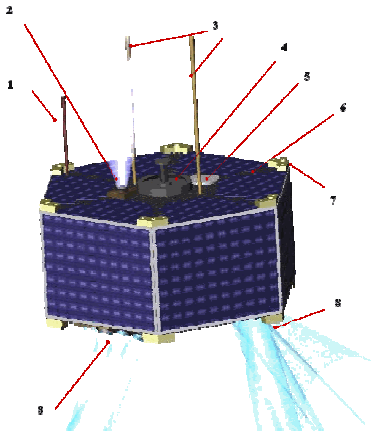

Figure 2: Single Spacecraft after Deployment

The exterior of the spacecraft is hexagonal with solar panels completely covering all sides except the top and bottom. The solar array panels are mounted on thin aluminum sheets that are attached to the exterior of the frame. All components are attached to aluminum honeycomb plates, which fasten to the main frame via slide-in interface brackets, and/or standard socket head cap screws. The batteries are stored in the middle of the structure to avoid an unbalanced inertial configuration. The batteries cells are housed in an eccofoam and aluminum structure attached in a manner to stiffen the component panels from harsh vibration environments.

The onboard flight processor is a 40 MHz PowerPC 825 that is not radiation hardened. The flight processor will be running the VxWorks real-time operating system with 16 MB of non-volatile RAM and 16 MB of dynamic RAM. This available memory must hold the operating system kernel, all of the traditional flight software (to manage the power, camera, and communication subsystems), the autonomy flight software, and all science data (camera images).

3 Onboard Science Data Validation Algorithms

The first element of onboard decision-making is onboard science data validation. Because the 3CS spacecraft are not attitude controlled (e.g.tumbling), their rotation will be characterized and modeled using information derived from the solar panel power generation. However, because of the difficulty of accurately modeling this motion, it is expected that many science images will contain little or no portions of the Earth. In order to detect these situations, 3CS will fly software to evaluate science images. The simplest way to detect these cases is to compress the images. Images of constant dynamic range (e.g. almost

all zeros caused by missing the Earth entirely) will compress almost completely (e.g. to very small size). Computing an average will distinguish all zeros from all maximum values. Alternatively more complex algorithms to find the limb of earth (the extremely bright crescent of the edge of the Earth facing the Sun) could be used. A range of these algorithms is currently being considered and tested on existing images deemed to be similar to 3CS mission data.

4 Robust Execution Software

SCL [SCL, 2001] is a Commercial off the shelf (COTS) software package whose core is a powerful scripting language. It enables a control system to process data input from multiple sources representing the state of the system and to take action based on either directed or detected system state.

SCL technology has been in development since 1988. SCL was proven in its successful flight on board Clementine-1 and ROMPS in 1994. Because it was first developed for an embedded flight environment, SCL has a small memory footprint and low computational demands.

The SCL language combines the features of a traditional scripting language, such as systems test and operations language (STOL), with the logic capture capability of a rule-based expert system and the functionality of a database definition language.

Using the SCL language—including scripts, rules and constraints, data formats, templates and definitions—a system designer can fully define all data and processes used by the control system. The system can take action based on time, operator directive, detected system state, or any combination of these. In addition, a user can dynamically reconfigure the system by modifying, in real-time, the SCL language constructs that specify what actions should take place under what circumstances.

SCL scripts, rules, and databases are reusable across a family of control systems. For example, for the Far Ultraviolet Spectroscopic Explorer (FUSE) mission, SCL models were reused: on the payload flight processor, in the payload integration and test system, and at the satellite control center. On 3CS, the same core set of rules and scripts is used for prototyping, testing, as well as in the ground software and flight software.

The SCL database contains records defining all data items that characterize a given system. Some examples include sensor measurements, actuator states, and derived data. Each record includes the current value of the data item and various attributes that are accessible by other SCL modules, scripts, and rules. Certain attributes can be "set" to cause a rule to trigger or to control actuators. Special record types specify how a data item is to be extracted from an input data stream, such as for telemetry processing.

The SCL DataIO module acquires data in real-time from external sources, updates the database, converts the data to engineering units, and filters, smoothes and archives the data if desired. DataIO also performs limit checking, and notifies the Real Time Engine (RTE) if any data value change exceeds a user-defined threshold. The SCL RTE is the inference engine for the underlying expert system, the command interpreter, and the script scheduler and execution manager. SCL scripts and rules are compiled by the SCL compiler and loaded to the RTE in the form of an SCL project. This project provides the RTE with the procedural, scheduling, and event-capturing elements of the system. The RTE captures all SCL database updates and processes the corresponding rules associated with a particular database item. The SCL compiler also supports real-time communication between an external operator and the RTE and SCL database via a "command line" interface.

SCL is used as the "glue" for the 3CS flight software, and integrates the autonomy software with the communications, power, and science payload software. Software components (external to SCL) communicate with the SCL RTE using a Message Software Bus. SCL can send a "notify" message to an external "listen" application when an SCL script or rule changes the state of an actuator. SCL also provides a constraint checking capability that allows the RTE to impose pre-conditions on command execution. This enables SCL to avoid potential failure scenarios, or to automatically execute pre-command configuration scripts.

SCL's mixture of procedural, time-based, and event-based programming using scripts and rules provides a rich environment for the design of autonomous applications. SCL scripting automates routine tasks and ensures consistently correct command sequences. Using rules that monitor and reason based on system state information, SCL can detect and react to events, such as anomalies, faster than human operators. SCL executes both time-based and event-based operations simultaneously in real-time.

For the 3CS mission, SCL is used to integrate the flight software components. SCL scripts will be used onboard to perform activities such as imaging, managing the inter-satellite communications link, coordinating ground/spacecraft communications opportunities, managing the power subsystem, and performing onboard housekeeping (such as data and event logging).

SCL rules and constraints encode the interdependencies between various activities being

performed onboard each of the 3CS spacecraft. For example, if onboard sensors indicate that a spacecraft is low on power, onboard planning and execution will be used to reduce the use of the power-intensive transceivers. Additionally, power availability will be monitored to appropriately postpone imaging opportunities until the spacecraft has reached a nominal power state. SCL scripts have the capability to check state and resource availability so as to only execute if they do not endanger the spacecraft. SCL will also be used to monitor and log spacecraft telemetry points as well as to execute the actual data dumps for downlinks during communications opportunities.

5 CASPER Planning Software

3CS will use onboard continuous planning software. This software will be used to generate mission plans to manage the science and engineering activities of the spacecraft. Specifically, 3CS will be flying the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) continuous planning system.

Traditionally, the majority of planning and scheduling research has focused on a batch formulation of the problem. In this approach, when addressing an ongoing planning problem, time is divided up into a number of planning horizons, each of which lasts for a significant period of time. When one nears the end of the current horizon, one projects what the state will be at the end of the execution of the current plan (see Figure 3). The planner will then generate a plan for the new horizon using a user-defined set of goals and the expected initial state. As an example of this approach, the Remote Agent Experiment operated in this fashion (Jonsson et al. 2000).

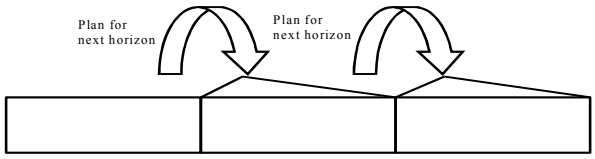

Figure 3: Traditional Batch Plan then Execute Cycle

This approach has a number of drawbacks. In this batch oriented mode, typically planning is considered an off-line process, which requires considerable computational effort, and there is a significant delay from the time the planner is invoked to the time that the planner produces a new plan.[1] If a negative event occurs (e.g., a plan failure), the response time until a new plan is generated may be significant. During this period the system being controlled must be operated appropriately without planner guidance. If a positive event occurs (e.g., a fortuitous opportunity, such as activities finishing early), again the response time may be significant. If the opportunity is short lived, the system must be able to take advantage of such opportunities without a new plan (because of the delay in generating a new plan). Finally, because the planning process may need to be initiated significantly before the end of the current planning horizon, it may be difficult to project what the state will be when the current plan execution is complete. If the projection is wrong the plan may have difficulty.

However, in an onboard planning context, the planner is an embedded entity that makes the batch-oriented model of planning inappropriate. Specifically, such an embedded planner must be anytime and responsive. It must be anytime in that at any point in time there must be an executable plan. This means that generative planning techniques are less suitable because they do not have the "anytime" property.

To achieve a higher level of responsiveness in a *dynamic planning* situation, we utilize a *continuous planning* approach and have implemented the CASPER continuous planning system (Chien et al., 2000). Rather than considering planning a batch process in which a planner is presented with goals and an initial state, the planner has a current goal set, a plan, a current state, and a model of the expected future state. At any time an incremental update to the goals, current state, or planning horizon (at much smaller time increments than batch planning)[2] may update the current state of the plan and thereby invoke the planner process. This update may be an

[1] As a data point, the planner for the Remote Agent Experiment (RAX) flying on-board the New Millennium Deep Space One mission (Jonsson et al 2000) takes approximately 4 hours to produce a 3 day operations plan. RAX is running on a 25 MHz RAD 6000 flight processor and uses roughly 25% of the CPU processing power. While this is a significant improvement over waiting for ground intervention, making the planning process even more responsive (e.g., on a time scale of seconds or tens of seconds) to changes in the operations context, would increase the overall time for which the spacecraft has a consistent plan. As long as a consistent plan exists, the spacecraft can keep busy working on the requested goals and hence may be able to achieve more science goals.

[2] For the spacecraft control domain we are envisioning an update rate on the order of tens of seconds real time.

unexpected event or simply time progressing forward. The planner is then responsible for maintaining a consistent, satisficing plan with the most current information. This current plan and projection is the planner's estimation as to what it expects to happen in the world if things go as expected. However, since things rarely go exactly as expected, the planner stands ready to continually modify the plan. From the point of view of the planner, in each cycle the following occurs:

- Changes to the goals and the initial state first posted to the plan,
- Effects of these changes are propagated through the current plan projections (including conflict identification)
- Plan repair algorithms[3] are invoked to remove conflicts and make the plan appropriate for the current state and goals

This approach is shown in below in Figure 4. At each step, the plan is created by using incremental replanning from:

- The portion of the old plan for the current planning horizon;
- The change (Δ) in the goals relevant for the new planning horizon;
- The change (Δ) in the state; and
- The new (extended) planning horizon

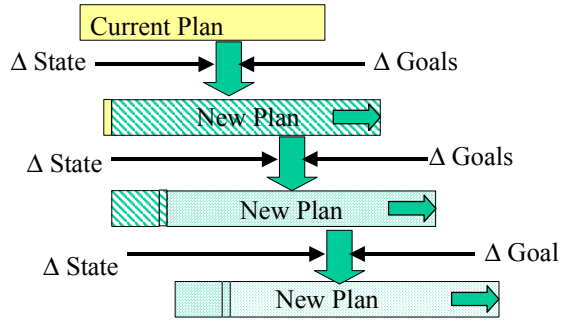

Figure 4: Continuous Planning Incremental Plan Extension

For the 3CS mission, CASPER will manage the near-term mission plan in between daily uplink and downlink passes. This involves tracking the engineering, communications, and science activities and modifying them in response to execution feedback. The most significant of these will be replanning of science observations based on onboard data assessment. Using a science score computed for each image onboard, CASPER will prioritize images. CASPER will then decide whether or not to discard images with little or no Earth in the frame, and plan for new images in an attempt to maximize the science return. While replanning, CASPER will need to account for available system resources: imaging opportunities, available power and energy, and available memory. CASPER will also need to manage setup procedures for downlink, uplink, command load, and imaging activities.

Within the available flight processor resources, CASPER is expected to be able to respond to activity and state updates on the 10 to 60 second timescale. This will enable more recent information regarding the execution status of activities as well as monitored state and resource values to influence planning.

CASPER will be integrated with the SCL execution system allowing for tight feedback from SCL rules and scripts to be reflected and acted upon within the CASPER plans.

The CASPER system is currently being deployed in a suite of applications including spacecraft control, autonomous ground communications stations, uninhabited aerial vehicles, and industrial control.

There is significant interest in using CASPER to provide onboard planning for single rover and multi-rover formations. In collaboration with the Long Range Science Rover effort, CASPER is being integrated with onboard control software for the Rocky 7 and Rocky 8 prototype planetary rovers [Volpe et al. 2000]. In this application, CASPER generates validated rover-command sequences for Rocky 7 based on high-level science and engineering activities. Once a plan has been generated it is continuously updated during plan execution to correlate with sensor and other feedback from the environment so that the planner may be responsive to unexpected changes.

CASPER has also been used in research demonstrations of spacecraft constellation autonomy (Barrett 1999, Barrett 2000) and rover swarms (Chien et al. 2000). In these efforts, CASPER is used in a distributed fashion to coordinate a team of rovers or spacecraft in achieving planetary science goals (for the remainder of this discussion we presume a distributed rover model but analogous efforts are underway involving distributed spacecraft). For this application, a distributed version of CASPER was developed where it is assumed each rover has an onboard planner, which allows rovers to plan for themselves and/or for other rovers. Each onboard planner generates a rover command sequence for achieving science goals and also performs execution monitoring and dynamic re-planning when necessary. This distributed planning environment is part of a multi-rover execution architecture being developed at JPL that integrates a number of systems including the

[3] For details on the state/resource representation or the repair methods see (Rabideau et al. 1999).

ASPEN planning and scheduling system, a machine-learning data analysis system, Rocky 7 rover-control software, and a multi-rover simulation environment (Estlin et al. 1999).

CASPER is also being deployed to automate ground communications stations (Fisher et al. 2001) and uninhabited aerial vehicles (Colgren et al. 2000).

6 SELMON Monitoring Software

The SELective MONitoring system (SELMON) (Doyle et al. 1993, Doyle 1995) is a generalized software-based monitoring system that uses multiple anomaly models to identify and isolate phenomena. SELMON improves upon existing monitoring methods in two areas:

Anomaly Detection- SELMON employs several techniques for recognizing abnormal behavior, going beyond the traditional methods of limit sensing (comparing sensor values to predefined alarm thresholds) and discrepancy detection (compares sensor values to predictions from a simulation). SELMON detects anomalies that traditional software-based methods fail to detect, making the anomaly detection process more complete and removing this burden from the operators.

Attention Focusing- Once an anomaly has been detected, SELMON determines how much of the system being monitored has been affected. This kind of information can be critical in the first few moments of an emergency, when several sensors reporting the same anomaly may lead to operator confusion and delayed response.

For the 3CS mission, current plans are to fly several anomaly detection methods onboard the spacecraft with more analysis to take place on the ground (where more memory and computational power are available). Based on availability of resources, this decision will be evaluated closer to launch.

Using SCL in conjunction with SELMON will enable further insight into system performance by evaluating summary data produced through the use of these tools. SELMON profiling will be used to monitor several aspects of system performance including: communication system performance, power system performance, as well as thermal characteristics of the spacecraft. SELMON will also be applied to tracking data (derived externally) and used to analyze the trajectory and orbit decay to assist in projecting mission lifetime, thermal, and communications characteristics.

The pairing of SCL and SELMON allows for a system that can be incrementally automated based on actual mission performance, experience gained during the mission and increased confidence in the automation of a given function rather than automating functions prior to launch based on predicted behavior.

7 Teamwork and Coordination

While the 3CS mission utilizes three spacecraft, in order to simplify the mission as much as possible, 3CS uses a simple coordination scheme based on centralized control. One spacecraft is designated the master spacecraft that will maintain the operations plan for all three spacecraft.

However, there still remains the problem of reliably electing a leader. Using a single static leader (e.g. designated prior to launch) is undesirable because if that spacecraft loses some key functionality it would mean the end of the mission. This general leadership election problem has been analyzed in depth (Tushar et al, 1996a, 1996b). Current plans are to implement a basic hard-coded solution to the three spacecraft problem in the onboard flight software.

8 Mission Status and Schedule

The 3CS mission is scheduled for launch in late 2002, with the software design, development, and integration already underway. The development schedule includes integration milestones continuously throughout the spring and summer of 2001. As part of this effort, the CASPER software is already being integrated with the SCL execution system. Additionally a 3CS testbed has been developed at the University of Colorado, where the individual pieces of the autonomy software are in the process of being integrated.

9 Related Work and Future Work

One related autonomy flight was the Remote Agent Experiment (RAX) [RAX, 1999]. The 3CS autonomous spacecraft mission differs from RAX in several ways. First, 3CS will be flying autonomy software for the entire mission duration (expected to be approximately 3 months), whereas RAX controlled the Deep Space One Spacecraft twice for approximately two days each. Second, 3CS will demonstrate closing the loop with an onboard science element as well as demonstrating onboard automation of engineering functions, whereas RAX did not involve science components. However, the Deep Space One Spacecraft was considerably more complex than the 3CS spacecraft. Finally, the constituent autonomy software modules used by each mission are different. RAX demonstrated Mode Identification and Reconfiguration using Livingstone and Burton, robust

execution using the Execution Support Language (ESL), and onboard planning using the Remote Agent Planner and Scheduler.

PROBA[PROBA] is a European Space Agency mission that will demonstrate onboard autonomy. PROBA will be launching in 2001.

Techsat-21 is a United States Air Force Mission launching in 2004. This mission will demonstrate the Autonomous Sciencecraft Constellation concept [Chien et al 2001, ASC 2001], using the CASPER planning software and SCL robust execution software. ASC will demonstrate the onboard science concept using a much more capable three spacecraft constellation. In addition to CASPER and SCL, ASC will use the Object Agent software to perform autonomous formation flying and maneuvering. In addition, ASC will use the Livingstone 2 and Burton software for model-based mode identification and reconfiguration.

10 Conclusions

The 3CS mission will use significant onboard autonomy including: robust execution using SCL, onboard planning using CASPER, onboard data validation, and onboard anomaly detection using SELMON. This exciting new mission will validate key autonomous systems technologies for future missions furthering the long-term quest for more capable, autonomous space systems for the 21st century.

11 Acknowledgements

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

12 References

Autonomous Sciencecraft Constellation, http://asc.jpl.nasa.gov

S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D. Tran, "ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling," *SpaceOps*, Toulouse, France, June 2000.

"R. Colgren, P. Schaefer, R. Abbott, H. Park, A. Fijany, F. Fisher, M. James, S. Chien, R. Mackey, M. Zak, T. Johnson, and S. Bush, Technologies for Reliable Autonomous Control (TRAC) of UAVs," Proceedings of the 19th Digital Avionics Systems Conference (DASC), Philadelphia, PA, 7-13 October 2000.

The PROBA Onboard Autonomy Platform, http://www.estec.esa.nl/proba/

R. Doyle et al., "Focused Real-time Systems Monitoring based on Multiple Anomaly Models," *Working Notes of the 7th International Workshop on Qualitative Reasoning about Physical Systems*, Eastsound, WA, May 1993.

R. Doyle, "Determining the Loci of Anomalies Using Minimal Causal Models," *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 1821-1827, August 1995.

F. Fisher, B. Engelhardt, R. Knight, C. Wilklow, S. Chien, T. Estlin, "CLEaR : Closed Loop Execution and Recovery," Proceedings of the IEEE Aerospace Conference (IAC), Big Sky, MT, March 2001.

Interface & Control Systems, Spacecraft Command Language, http://www.sclrules.com.

A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith, "Planning in Interplanetary Space: Theory and Practice," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge*, CO, April 2000.

G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," *International Symposium on Artificial Intelligence Robotics and Automation in Space*, Noordwijk, The Netherlands, June 1999.

The Remote Agent Experiment, http://rax.arc.nasa.gov.

D.C. Tushar, S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," Journal of the ACM, 43:2, March 1996, 225-267.

D.C. Tushar, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," Journal of the ACM, 43:4, July 1996, 685-722.

R. Volpe, T. Estlin, S. Laubach, C. Olson, B. Balaram, "Enhanced Mars Rover Navigation Techniques," IEEE International Conference on Robotics and Automation (ICRA 2000), San Francisco, CA, April 24-28, 2000.