Adjustably Autonomous Multi-agent Plan Execution with an Internal Spacecraft Free-Flying Robot Prototype

Gregory A. Dorais and Keith Nicewarner[*]
NASA Ames Research Center
MS 269-2, Moffett Field, CA, 94035, USA
gdorais@arc.nasa.gov, knicewarner@arc.nasa.gov

Keywords: robot planning and execution, applications of planning and scheduling, reactive planning, mixed-initiative planning, multi-agent planning, adjustable autonomy

Abstract

We present a model-based, adjustably autonomous multi-agent architecture with monitoring, planning, diagnosis, and execution elements. We discuss an internal spacecraft free-flying robot prototype controlled by an implementation of this architecture and a ground test facility used for development. In addition, we discuss a simplified environment control life support system for the spacecraft domain also controlled by an implementation of this architecture. We discuss adjustable autonomy and how it applies to this architecture and its user interface, which provides the user situation awareness of both autonomous systems and enables the user to dynamically edit the plans prior to and during execution as well as control these agents at various levels of autonomy. This interface also permits the agents to query the user or request the user to perform tasks to help achieve the commanded goals. We describe a sample scenario where these two agents and a human interact to cooperatively detect, diagnose and recover from a simulated spacecraft fault, and conclude by describing the individual components in the autonomy architecture.

Introduction

Spacecraft mobile robots offer the potential to increase the capability, productivity, and duration of space missions while decreasing mission risk and cost. They can perform a number of functions, both inside and outside the spacecraft, from relatively simple tasks, such as remote sensing and providing crew support, to more complex tasks, such as performing maintenance and in-situ construction.

Onboard planning and robust plan execution are key technologies necessary to robustly achieve increasingly ambitious mission goals for longer time periods with less ground support than traditionally required. The “Ambitious Spacecraft” proposed in [1] represents a simplified domain that captures several important aspects of long-term planning applicable to space-based observatories (either orbiting or fly-by) where the dynamics of the environment and the interaction with human operators occur infrequently.

In this paper we focus on a different domain where a space-based robot is an active assistant to a human astronaut, ground personnel, or other onboard autonomous systems. As with the ambitious spacecraft, instrument and motion requirements, as well as the need to optimize time and scarce resources, are still present. However, the robot must be able to deal with the variability of an engineered environment occupied by humans and has to be able to quickly and safely modify its plans as needed.

In a research effort to study how free-flying mobile robots can support operations inside manned spacecraft, in particular the International Space Station (ISS), NASA is developing a series of prototypes, called Personal Satellite Assistants (PSAs), each having increased remote sensing and navigation capabilities [2]. Three of these robots are being operated in ground test facilities and the fourth is under development.

One of the predominant challenges to deploying PSAs is to reduce the need for direct operator interaction. Teleoperation is often not practical due to the communication latencies incurred because of the distances involved and in many cases a crewmember would directly perform a task rather than teleoperate a robot to do it. We have developed an adjustably autonomous control system, based on the IDEA architecture [3], that integrates a constraint-based plan database (EUROPA) [4], deliberative planner, reactive planner, path planner, and a diagnosis engine (Livingstone 2) [5]. We have used this control system to command PSAs in simulation as well as the physical prototypes in test facilities.

This paper briefly presents a PSA prototype and one of the test facilities. We discuss adjustable autonomy and the adjustably autonomous control system and its user interface. We conclude by examining a test scenario where a PSA, a simulated environmental control life support system, and a human cooperatively detect, diagnose and recover from a simulated spacecraft fault.

PSA Prototype

A 6-DOF PSA Model 2 prototype was developed and is shown in Figure 1. The Model 2 is 12” in diameter (the targeted flight model diameter is 8”) and capable of position and velocity estimation and motion in 6-DOF (X, Y, Z, yaw, pitch, roll). 6-DOF position and velocity estimation is achieved using multiple stereo-pair cameras (between 1-4 pairs). Propulsion and attitude control in 6-DOF are achieved using 6 ducted-fan pairs.

[*] QSS Group, Inc.

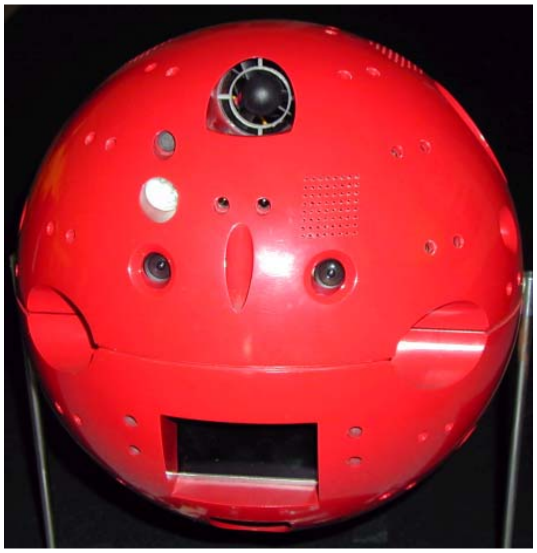
Figure 1 - PSA Model 2

The Model 2 has an LCD located at the center of its front lower hemisphere. The LCD can be used to display data generated locally as well as data received via its wireless network, e.g., text terminals, images, schematics, videos, and teleconferencing. The location of these and additional components are depicted in Figure 2.

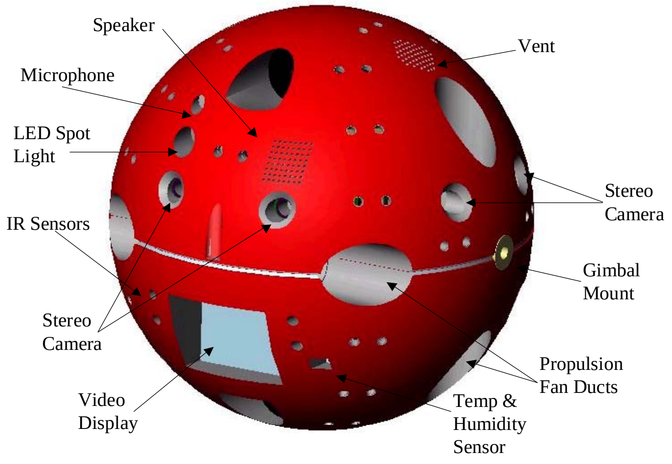

Figure 2 - PSA Model 2 Annotated Drawing

Micro-gravity Test Facility

To test the Model 2 on Earth, a micro-gravity test facility was developed. The facility is roughly 36’ long, 13’ wide, and 8’ high. It contains a full-scale mockup of the ISS U.S. Lab module. The facility consists of a 3-DOF (X,Y,Z) bridge-crane-like mechanism that supports a passive gimbal that mounts the PSA, which permits free spinning in yaw and pitch. The PSA Model 2 is shown in the facility in Figure 3.

The facility can be operated in several modes: position, velocity, force, and follow. Follow mode is the most significant and enables us to simulate micro-gravity.

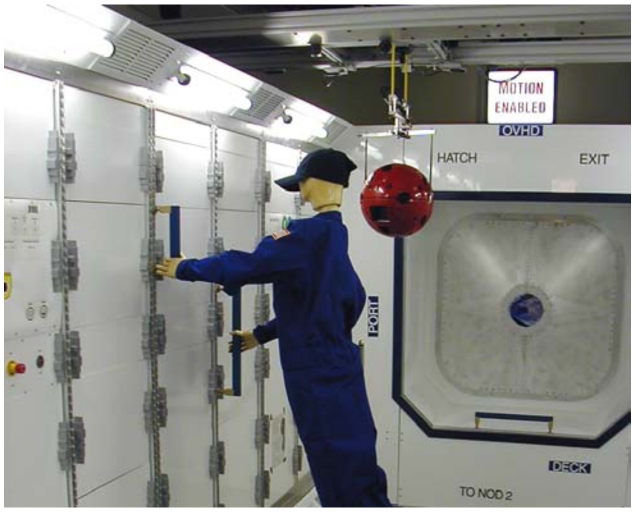

Figure 3 - PSA Model 2 in Micro-gravity Test Facility

Sensors located on the trolley and gimbal sense translation forces (X,Y,Z) acting on or generated by the gimbal payload. These sensor signals are interpreted by the crane motors as force commands and move the payload accordingly. The Z-axis signal is offset to cancel the force of gravity. The result is that the payload “floats” within the facility, moving at a relatively constant velocity, and accelerates appropriately when the payload is subjected to a force. When the PSA Model 2 is the payload, its fan power is sufficient to propel it throughout the facility as if it was in a micro-gravity environment.

Adjustable Autonomy

One of the challenges we face in designing the autonomous control system for this robot is to enable the user to control the robot, or in some cases to be instructed, at the command level most effective to accomplish the desired tasks. For some tasks, the system should be able to run without user intervention. For other tasks, direct control of the robots’ velocity and position is desired. For many tasks, what is desired is some operation point between these two extremes. We refer to the capability to dynamically select an operation point between these two extremes as adjustable autonomy. The four areas of a system that affect how “adjustable” its autonomy is are: data reporting, decision-making, sensing, and actuation.

Data Reporting

The goal of data reporting is to provide relevant information that provides the user situation awareness (particularly if the user had not been monitoring the system) of the environment, the robot, as well as the autonomous system, in order to understand the system behavior, predict its future behavior, and assess how the user may change its behavior. Too much data can overwhelm the user and make it difficult to find the relevant information. However, if too little data is provided, the user will not be able to properly assess and command the system. Moreover, no one level of data is adequate. In some cases more is needed and other cases less is better. In the PSA autonomous control system, data is provided to the user by four methods:

real-time displays—including multiple onboard camera views, a third-person simulated perspective of the PSA in its environment, and sensor signals graphed over time.

synthesized spoken language—it has the advantage of drawing the user's attention but generally must be used sparingly. The user can select which messages should be announced.

textual messages and an event log—this also can be controlled by the user selecting which messages can be sent and by selecting a log level for events.

the PSA itself—including its position, motion, and its LCD display. We are in the process of adding a laser pointer to the PSA so it can be used to point for the benefit of the user.

Decision-making
Decision-making is essentially the ability for the user to command the system. Currently, there are five methods to command the system:

teleoperation—a GUI and two 3DOF joysticks (one for controlling translation and the other for orientation) enable direct control. Automatic obstacle avoidance and station keeping is supported.

plan editing—the plan may be edited even while it is being executed. Decision types include: goal insertion & removal, goal schedule restriction & relaxation (undo restriction), and goal argument restriction & relaxation.

decision assignment—the user can determine which decisions can be made by the autonomous system and which the user must make. Each decision can be set to one of the following levels:

- prevent command – the command is not permitted
- suppress: make decision without any notification
- exec notify: make decision and send user message when executed
- plan notify: make decision and send user message when planned and when executed
- request approval - default positive: ask user permission, no decision prior to timeout interpreted as approval.
- request approval - default negative: ask user permission, no decision prior to timeout interpreted as denial.
- prevent decision - permit command

spoken language commands—currently this method is limited, but there are situations where it is useful since the user need not be near the GUI and it enables hands-free commanding.

direct robot interaction—includes physically moving the robot, directing it to avoid or follow the user or an object, or commanding it using human gestures it recognizes (not yet implemented).

Sensing
The sensing area of an adjustable autonomous system refers to the ability of the system to use a person as virtual sensor or for the person to otherwise alter the system's perceived state of the environment or itself. For example, the system may decide to ask a person if a hatch is open rather than making the trip to check. Similarly, the PSA may check if the hatch is open by attempting to pass through it and infer that it is closed because it was blocked. However, the hatch may be open, but was blocked because a person was there. The person could correct the incorrect inference made by the PSA by informing the system that the hatch was open thus permitting it to generate plans than involve passing through the hatch.

Actuation
Like the sensing area, the actuation area refers to those actions that can be done by a user. Examples of such actions are requesting the user to open a closed hatch or performing a maintenance task on the PSA.

Our hypothesis is that by enabling the user to vary the level of control in the four areas of data reporting, decision-making, sensing, and actuation over an extended period of operation, the system can be effective at cooperatively achieving a wide-variety of tasks.

Sample Multi-Agent Mission Test Scenario

In this section we discuss a scenario, summarized in Figure 4, in which the PSA, an autonomous Environmental Control Life Support System (ECLSS) agent, and a crewmember participate in the diagnosis of and recovery from an ISS module fault. In this scenario, ECLSS is autonomously controlled by a high-level autonomous system similar to the one used by the PSA as shown in Figure 5 to be discussed in the following section (the main difference is that ECLSS does not use a path planner). The scenario has two variations depending on the cause of the initially sensed anomaly. This scenario is used to demonstrate:

- Integrated Vehicle Health Management
- Cooperative multi-agent planning and execution
- Generation and execution of a near-optimal 6-DOF route plans
- Stereo vision-based 6-DOF localization and map registration

The scenario begins with the PSA station keeping at its dock when a fixed temperature sensor at rack 5, locker 1 in the ISS US Lab module signals a high temperature to the ECLSS. The ECLSS attempts to diagnose the problem and is not able to determine whether the sensor is defective or if the station system is actually overheating without additional information. In our case, we have specified that each case is equally likely. So in order to disambiguate the system state, ECLSS commands the PSA agent to go to the fixed sensor location and verify the temperature at that location by sending the PSA agent a sense-at-location goal. The PSA agent then reactively deliberates (i.e., the reactive planner calls the deliberative planner in response to the new goal). The deliberative planner decomposes the goal into a move-to subgoal followed by a subgoal to maintain position while the temperature is sensed. The move-to subgoal then decomposes into a path-planning subgoal followed by an

Step	Agent	Scenario Step Description
1	ECLSS	Detects a high heat signal from a fixed ISS node sensor. Fixed sensor health or heat source unknown.
2	ECLSS	Commands PSA to verify the temperature at that location
3	PSA	Generates plan upon receipt of the command to go to the fixed sensor location and measure temperature
4	PSA	Starts executing plan
5	PSA	Moves to fixed sensor and begins collecting temperature data and sending it to ECLSS
		Variation A: Fixed rack sensor failed high, rack lockers nominal
6a	ECLSS	Determines fixed sensor is faulty and uses PSA sensor as temporary sensor.
7a	ECLSS	Requests crewmember to repair sensor
8a	Crewmember	Repairs fixed sensor and notifies ECLSS
9a	ECLSS	Requests PSA to measure temperature to validate fixed sensor, which signals actual temperature
10a	ECLSS	Infers problem resolved and commands PSA to return to docking bay
11a	PSA	Returns to docking locker recharge
		Variation B: Fixed rack sensor healthy, one rack locker overheating
6b	ECLSS	Determines fixed sensor is accurate.
7b	ECLSS	Commands PSA to locate heat source.
8b	PSA	Searches region for heat source and determines maximum heat is at location of locker X.
9b	PSA	Sends locker location and its temperature to ECLSS.
10b	ECLSS	Determines locker can be powered down and turns off power to locker. Temperature declines.
11b	ECLSS	Requests PSA to verify temperature has declined
12b	PSA	Sends locker temperature to ECLSS
13b	ECLSS	Releases PSA to perform previously scheduled tasks
14b	PSA	Returns to docking locker to recharge

Figure 4 – Sample PSA Test Mission Scenario

execute path subgoal. All of these goals are flexibly scheduled. When the path-planning goal is executed, a path from the current location to the desired location is generated, where a path consists of a sequence of waypoints that avoids known obstacles and no-fly zones. When the path is scheduled to execute, the PSA agent sends it to the PSA subsystem, which executes each waypoint. As needed, the trajectory between waypoints is dynamically changed to avoid obstacles detected en route. When it arrives at the destination, the PSA subsystem confirms that the path was completed, or in a failure case notifies the PSA agent that the path cannot be achieved. PSA agent then commands the PSA subsystem to station keep for a period while the PSA measures the temperature. After that period, the top-level PSA agent sense-at-location goal completes by returning the sensed temperature to ECLSS. ECLSS then compares the two sensor readings. The two cases where they agree or disagree are listed below. The preceding activity is summarized by steps 1-5 in Figure 4.

If the PSA and ECLSS temperature sensors disagree, the ECLSS state estimator infers that the fixed temperature sensor has failed and requests that a crewmember repair it by sending a message to the ECLSS operator user interface requesting the repair and waits for confirmation that the crewmember has repaired the sensor. When the crewmember confirms, the fixed sensor value returns to nominal. Meanwhile, ECLSS tells PSA to measure the temperature again at the same location and compares the return value to the value read from the fixed sensor. Since they now agree, the ECLSS state estimator infers that the fixed sensor is healthy and the PSA is commanded to its dock completing the scenario (steps 6a-11a).

However, if the PSA and ECLSS temperature sensors agree, then the ECLSS state estimator infers that a nearby locker is overheating, but which one is unknown. ECLSS gives a goal to the PSA agent to direct it to locate the source of the heat. The PSA agent decomposes this goal to send a command to the PSA subsystem that causes it to execute its heat source seeking behavior. This behavior has the PSA first spin fully around, scanning the environment with its thermal imager. Once the scan is complete, the PSA points to the largest magnitude heat source and moves toward it. When the PSA gets as close as it can to the heat source, the PSA agent completes the sense-at-location goal by returning the location and temperature measurement to ECLSS. In our case, the heat source is actually in a locker in a neighboring rack, which the ECLSS state estimator infers. ECLSS then commands the system operating in the locker to power-off, which reduces the (simulated) heat in the area. The ECLSS fixed sensor then reads a nominal temperature. ECLSS sends the PSA agent a goal to measure the temperature again to verify the temperature is nominal. The PSA measures the temperature and returns the temperature to ECLSS. ECLSS infers that the locker temperature is nominal and releases the PSA from further requests. The PSA then returns to its dock completing the scenario (steps 6b-14b).

PSA Autonomy Framework

An autonomy framework has been developed and is depicted in Figure 5 [6]. Care was taken to design and implement this framework so that it is applicable to a wide range of free-flying vehicles.

The user can issue commands to the PSA through the Crew GUI. Also, the user can issue verbal commands to the PSA and receive spoken notifications generated by the PSA via a headset. Other external systems, including other PSAs, can directly and simultaneously issue commands to the PSA, which will attempt to resolve any conflicts. Finally, the PSA itself can generate commands in keeping with its high-level goals and periodic task schedule.

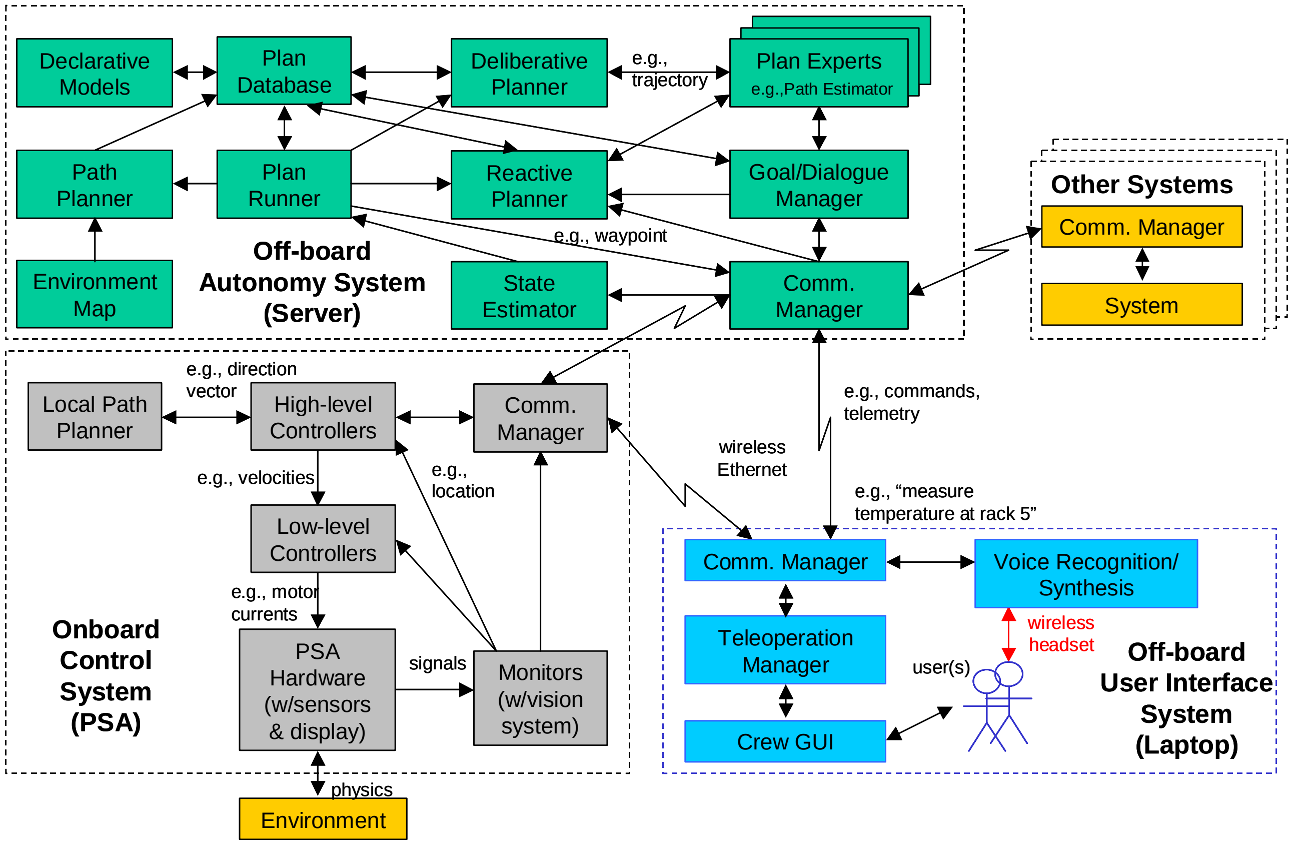

Figure 5 - PSA Autonomy Framework

The PSA autonomy framework is comprised of a number of control elements, which are represented as boxes in Figure 5. The current implementation is distributed over three processors, as indicated by the dashed boxes, which are connected by wireless Ethernet. Each of these three subsystems and the control elements it contains is briefly discussed below. Note that the framework design and many of its elements draw their heritage from the model-based, goal-achieving, temporally-flexible NASA "Remote Agent" autonomy software flight-validated on the Deep Space One spacecraft in 1999 [7].

Onboard Control System Elements

The onboard control system is responsible for sensing, sensor analysis (e.g., object and fault recognition), state estimation (e.g., position estimation), hardware actuation (e.g., motor currents), and real-time reactive control (e.g., obstacle avoidance), generally with sub-second latency. This system is designed to enable local operation of the PSA even when communication with the off-board system is lost, which may occur during a flight emergency.

Local Path Planner—generates a trajectory between two waypoints that takes into account locally sensed obstacles When given a third waypoint, the trajectory passes through the second waypoint without stopping. The local path planner performs limited trajectory repair in case of a path plan failure, e.g., blocked path.

High-level controllers—translates the trajectory into a sequence of 6-DOF (position, velocity, and acceleration) setpoints for the low-level controllers.

Low-level controllers—translates the setpoints into motor force commands to achieve the specified PSA motion.

PSA Hardware—the sensors and actuators with their associated drivers. These include fan motor controllers, stereo cameras, environment sensors, proximity sensors, and an LCD.

Monitors—signal processing loops that abstract the data generated by the sensors. They run from being as simple as indicating that a proximity sensor has triggered to continually calculating 6-DOF positions and velocities by fusing the stereo camera, 6-DOF inertial sensors, and proximity sensors.

Communication Manager—responsible for managing message traffic and executing appropriate message handlers. Serves same role in both off-board systems.

Off-board Autonomy System

The off-board autonomy system is responsible for high-level autonomous control including inter-agent communication and coordination (including humans), goal management, decomposing high-level tasks (planning) into commands that can be executed by the onboard control system, e.g., waypoint commands, constraining task times (scheduling), command sequencing (plan execution), and reasoning about sensor data provided by the onboard control system, e.g., for diagnosis, and for plan repair, e.g., onboard control system is unable to achieve a waypoint. Architecturally, this system could be integrated onboard the PSA.

Declarative Models—contains the library of constraints used by the Plan Database that define a set of coordinated state machines. A constraint may simply specify that Task A must precede Task B by at least 10 seconds but not more than 20 seconds. The constraint may also functionally relate the parameters of tasks A and B as well as specify preconditions as to when it applies.

Plan Database—contains the plan being executed and is responsible for automated sub-goaling of tasks, i.e., determining the set of sub-tasks required to achieve a task, and for maintaining flexible plans, i.e., the propagation of valid task variable domains that are minimally restricted without violating a constraint. This has been implemented using the EUROPA plan database developed at the NASA Ames Research Center. EUROPA is a derivative of the model-based, temporally-flexible Remote Agent Plan Database described in [4], an earlier version of which was demonstrated on Deep Space One [7]. The plan database represents a temporal, constraint-based network of tokens that defines the past, the present, and flexibly-defined future states and actions of the system. Each token represents the "state" of a state variable for a period of time and the tasks that achieve or determine this state. Each token defines a start, end, and duration temporal variable, each with an upper and lower bound, as well as the procedure (predicate and arguments) invoked when the token is "executed." The plan database supports multiple timelines with constraints on and between tokens. If none of the constraints are violated for a given instantiation of the plan database, the database is defined to be consistent.

Deliberative Planner—schedules outstanding tasks, and the related sub-tasks generated by the plan database, as well as makes decisions regarding constraining the domains of task variables to achieve specified goals during a specified period of time. This element is implemented by a variation of the Remote Agent Model-based Planner/Scheduler described in [4] and as specified by the Intelligent Distributed Execution Agent (IDEA) architecture [3]. More specifically, the Deliberative Planner (DP) is responsible for generating a consistent, flexible plan in the plan database given a start and end horizon time bound, an initial state of the timelines at the start time, and a set of goals. A flexible plan is loosely defined as a set of timelines, each consisting of tokens on each timeline, token order constraints that prevent overlapping tokens on the same timeline, and token procedure variable constraints. Plan flexibility is characterized by the set of decisions yet to be made that result in a consistent plan. A plan identification function is used to determine which of the outstanding decisions must be made in order to have a valid plan. The search process and decision selection priorities are determined in part by user-defined heuristics. Complex plans can require considerable computation time. The proper set of heuristics can dramatically reduce the time required. The DP is called to initialize the plan database and also is called during plan execution as specified by the plan being executed. It is typically called to plan for a period of significant duration sufficiently in the future such that the DP will complete prior to the start time of this period, but not so far in the future that the initial state at the future start horizon is not known with high confidence.

Reactive Planner—responsible for insuring that the Plan Database is in a state such that the tasks to be executed at a specified time are unambiguous. It has been implemented as described in [3]. In many respects, as implemented the Reactive Planner (RP) is very similar to the DP described above, although that not need be the case. The salient differences between the two planners are:

- the RP reasons over a shorter, more immediate time horizon, typically ending just after the current execution time
- the RP plan identification function is more restrictive so decisions that were postponed by the DP must now be made; the time allocated for planning is relatively very short, typically less than a few seconds, and cannot be exceeded without a fault
- in the event of a plan deliberation or execution failure, the RP repairs the plan locally or if necessary generates a standby plan to safe the PSA and call the DP. Plan repair may be necessary for several reasons including tasks completing too late or too early, task return state variables posted to the Plan Database make it inconsistent, and new tasks have been added to the Plan Database for immediate execution that cause a conflict.

Plan Experts—computational procedures, called by a planner, that return information used by the planner to make planning decisions, typically regarding token variable values. For example, a route planner expert is called by either the deliberative or reactive planner to determine the time, route, and energy required to move between two points in the environment or to cover a certain space. The route planner expert has access to a global map that can be updated with sensed obstacles. A route plan request is typically made by the deliberative planner as part of developing the initial plan, but may also be called by the reactive planner to develop an alternate route if necessary, e.g., the route is blocked or there is insufficient energy to complete the current plan. In addition, a user may initiate a request to answer a hypothetical question about a particular goal.

Plan Runner (command sequencer)—executes tokens in the plan database at the appropriate time. Executing a token involves calling the procedure with its arguments defined by the token, updating the plan database with the token return values when the procedure terminates, constraining the plan database so that planners only have limited ability to change the past, and calling the Reactive Planner, as described above, as needed to update the plan database. The plan runner implemented is described in more depth in [3].

State Estimator—abstracts and infers a consistent set of state variables with respect to a system model given the discrete and continuous sensor data provided over time. Some of these state variables, such as the health of a sensor, may not be directly measurable. To accomplish this we are using the model-based L2 state estimation system, which is based on algorithms described in [5] and is an extension of the

Livingstone system that was a component of the Remote Agent [7]. In certain instances it may be necessary to infer that a sensor is not healthy in order to achieve a set of state values that are consistent with the system model. In other cases it may be necessary to collect additional data to disambiguate between conflicting possible inferences for given sensor data.

Goal/Dialogue Manager—acts as an arbiter between the autonomous control system and other agents, including people. It retains state regarding its interaction with the other agents, e.g., recalls the subject of a previous sentence spoken by the user. As an arbiter, this element serves two roles: a goal manager and a dialogue manager. The goal manager essentially acts as a meta-planner for the deliberative planner. As stated above, the deliberative planner requires a start and end horizon time bounds, an initial state of the timelines at the start time, and a set of goals. The goal manager interacts with the user to determine this information. This may include negotiation of goals when all goals are not achievable or supporting mixed-initiative planning for hypothetical situations. The dialogue manager is responsible for acting as an intelligent interface with other agents. When interacting with people, it can converse with a person speaking a restricted natural language, responding as appropriate to spoken commands and queries. It inserts, changes or removes tokens in the Plan Database or responds to user queries by querying the planner experts and Plan Database. Currently, the integrated Dialogue Manager is simplistic. A more sophisticated dialogue manager tested on a stand-alone simulator is presented in [8]. The integration of such a dialogue manager remains as future work.

Off-board User Interface System

The user-interface system enables the user to interact with the PSA by commanding and displaying information. It provides situational awareness, sensor-data views, plan views, and commanding capabilities. This includes interfaces for interactively creating and modifying the plan as well as teleoperation. Our intent is for this interface to support operation at various autonomy levels that can be dynamically changed and range from teleoperation to high-level autonomous control.

Voice Recognition and Synthesis—provides speech-to-text and text-to-speech conversions. The voice recognition subsystem essentially converts an audio signal into a parsed text stream. Conversely, the voice synthesis subsystem essentially converts text commanded by the Dialogue Manager or the Plan Runner into speech via the user headset or remote speakers. We use commercial products to accomplish these tasks and plan to upgrade them as improvements are made.

Teleoperation Manager—executes supported user commands and converts GUI-generated commands into commands executable by the Autonomy system, e.g., plan editing. Also, it supports two force-feedback 3-DOF joysticks or one 6-DOF joystick for teleoperation in position, velocity, or acceleration modes.

Crew GUI—displays the sensor data, renders the PSA in a 3D model of its environment, displays plans, provides plan editors for both PSA task and path plans, and provides for direct commanding of the PSA. Included in the displayed sensor data is the real-time video stream generated by the PSA. In addition, by using a camera mounted on the Crew GUI display, the Crew GUI supports teleconferencing.

Summary

We presented the ongoing research and development effort to design an adjustably autonomous control system for an internal spacecraft free-flying robot prototype, which is also applicable to a wide range of free-flying vehicles. We described a PSA prototype as well as its micro-gravity test facility. We discussed adjustable autonomy and the autonomy framework for intelligent flight vehicle control being developed. A sample mission scenario being used to test the prototype and the autonomous control system was also outlined.

Acknowledgements

We gratefully acknowledge the contributions of the many talented people on and supporting the PSA team. In addition, we acknowledge the support provided by the NASA Cross-Enterprise Technology Development Program, the Computing, Information, and Technology Program, and the Engineering Complex Systems Program.

References

[1] Smith, David E., Frank, Jeremy, and Jonsson, Ari K., "Bridging the gap between planning and scheduling." Knowledge Engineering Review, 15(1):47-83, Cambridge University Press, UK, 2000.

[2] Gregory A. Dorais and Yuri Gawdiak, "The Personal Satellite Assistant: an internal spacecraft mobile monitor." *Procs. of the IEEE Aerospace Conference*, Big Sky, MT, 2003.

[3] Nicola Muscettola et al., "A unified approach to model-based planning and execution." *Procs. of the Sixth International Conference on Intelligent Autonomous Systems*, Venice, Italy, 2000.

[4] Ari K. Jonsson, et al., "Planning in interplanetary space: theory and practice." *Procs. of the 5th Artificial Intelligence Planning and Scheduling Conference*, Breckenridge, CO, 2000.

[5] James Kurien and P. Pandurang Nayak, "Back to the future with consistency-based trajectory tracking." *Procs. of the 17th National Conference on Artificial Intelligence*, Austin, TX, 2000.

[6] Gregory A. Dorais, et al., "An autonomous control system for an intra-vehicular spacecraft mobile monitor prototype." *Procs. of the 7th International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, Nara, Japan, 2003.

[7] Douglas Bernard, et al., "Final report on the Remote Agent experiment." *Procs. of the New Millennium Program DS-1 Technology Validation Symposium*, Pasadena, CA, Feb. 8-9, 2000.

[8] Manny Rayner, Beth Ann Hockey, and Frankie James, "A compact architecture for dialogue management based on scripts and meta-outputs." *Procs. of Applied Natural Language Processing (ANLP)*, 2000.