A FRAMEWORK FOR AUTONOMOUS SPACE ROBOTIC OPERATIONS

Erick Dupuis, Canadian Space Agency, Canada
Régent L'Archevêque, Canadian Space Agency, Canada
Pierre Allard, Canadian Space Agency, Canada
Ioannis Rekleitis, Canadian Space Agency, Canada
Eric Martin, Canadian Space Agency, Canada

1 INTRODUCTION

Over the last few decades, robots have played an increasingly important role in the success of space missions. The Shuttle Remote Manipulator System (also known as Canadarm) on the Space Shuttle has enabled the on-orbit maintenance of precious space assets such as the Hubble Space Telescope. On the International Space Station (ISS), the Canadarm 2 has been a crucial element in all construction activities. Its sibling, Dextre, will be essential to the maintenance of the ISS. In the context of planetary exploration, robotics has also played a central role on most landed missions. The rovers "Spirit" and "Opportunity" are vibrant examples of how robots can allow scientists to make discoveries that would be otherwise impossible.

In light of the missions currently being planned by space agencies around the world, the coming years will only show an increase in the number and the criticality of robots in space missions. Examples include DARPA's Orbital Express mission [32] and DLR TECSAS [28] in the area of satellite-servicing. Similarly, in planetary exploration, robotics will play a critical role

in the Mars Phoenix mission [27], NASA's Mars Science Laboratory [14] and ESA's Exo-Mars [29].

One of the current drawbacks of space robots is that their operation is very human-intensive. On manned platforms such as the Space Shuttle and the ISS, the baseline for operations requires the involvement of an astronaut for control and monitoring and the support of ground support personnel to monitor the operations in real-time. Furthermore, all operations carried-out by these robotic systems are thoroughly verified in simulation prior to their execution in space. Scenarios are run to verify all possible combinations of nominal and anomalous conditions. The verification process spans over several months, taking nominally more than one year. Every hour of operation requires thousands of hours of involvement from support personnel in the planning, verification and execution phases [25]. The advent of ground control for the Canadarm2 on the ISS has freed up the crew from having to perform all robotic operations. Ground-based operators now have the capability to uplink individual commands for automatic execution on the ISS. However, ground control has not reduced the level of effort required for robotic operations: the process to plan, conduct and support robotic operations remains exactly the same.

In the case of robotic planetary exploration missions, the safety constraints are somewhat more relaxed than for manned spaceflight. However, the constraints associated with the communications between the robot on the surface of a remote planet and Earth-based operators are much more stringent. For example, in the case of Mars exploration missions, the round-trip time delays are on the order of 10 to 40 minutes. Because of the geometric configuration of the planets and of the communications assets, it is not uncommon to have situations where there are communication windows of only one hour over a period of 12 hours. In addition, the robot must operate in an environment that is not known a-priori. Environment models are built on the fly and are made available to the operations teams every cycle. The turn-around time for a typical planning cycle from environment modelling, operation planning, verification and upload is on the order of 12 hours.

Despite this, the operations philosophy is based in similar principles to the one used on the International Space Station. Missions like Mars Pathfinder as well as the Mars Exploration Rovers make extensive use of simulations in virtual environments to rehearse operations in order to ensure success of the operations planned for the next cycle. This still requires relatively detailed modelling of the environment and extensive modelling of the behaviour of the robot in this environment. The result is a hectic schedule where dozens of contributors work on shifts to ensure round-the-clock operations during the primary mission.

One of the main consequences of the current operational philosophy is that the cost of operations is extremely high due to the necessity for intensive ground personnel support. For manned programs such as the ISS, the planning cycle is so long that there is virtually no flexibility to adapt to changing circumstances.

So far very little is implemented in terms of autonomous decision-making capability. Operations are based on pre-planned, pre-verified command scripts. When situations requiring some sort of decision are encountered, the robot must usually stop and wait for a human operator to intervene. For example, in the case of the rovers "Spirit" and "Opportunity", once an interesting geological feature has been identified, it takes at least two command cycles (of 12 hours each) to go apply an instrument to it [24]. The scientific return on investment is therefore severely limited by the limited on-board autonomy capability.

Over the last several years, several architectures have been proposed to address problems similar to those associated with space robotics. Among the earlier approaches was the subsumption architecture by R. Brooks [3] where different control laws were used in increasing complexity. Alliance [23] was an extension of subsumption to control multiple robots. SRI

proposed SAPHIRA [16] as the main architecture for the robots from ActivMedia company (manufacturers of the Pioneer mobile robots). From academic labs different open source packages have been put forward addressing the issues of robot control. It is worth noting CARMEN [20], RoboDaemon [5] and the most widely spread combination of Player/Stage/Gazebo [9][30] . The above packages provided different simulators and a central control unit.

In the space community, several architectures have been developed to address the issues associated with ground control. However, since the usual mode of operation for space robotics in the past has been teleoperation with direct operator control or supervision, most of the approaches have not focused on the implementation of autonomy. In the late 1990's, the Canadian Space Agency (CSA) and their industrial partner, MD Robotics, have developed the Intelligent, Interactive Robotic Operations (IIRO) framework [7] , which allowed the teleoperation of remote equipment in operational settings. The Remote Operations with Supervised Autonomy (ROSA) architecture was a follow-on to IIRO and addressed the issue of scripted control and basic autonomy requirements [6][8]. ROSA has been used as the basis for the development of the ground control station for the robotic elements on the Orbital Express satellite-servicing mission .

Similar architectures were developed in Europe at the same time. The Modular Architecture for Robot Control (MARCO) developed by DLR addresses similar issues in the context of teleoperation, ground control, and telepresence [4] . The MARCO architecture and its relatives have been used on several missions including ROTEX and ETS-7. Two other architectures were also developed under the leadership of the European Space Agency: FAMOUS [10] and DREAMS also concentrated on the issues associated with the teleoperation of robots in space. In both cases, special attention was dedicated to the issues surrounding planning, verification and execution of command sequences. Similarly, in the US, the JPL has developed a set of tools for rover ground control from planning to post-flight analysis. This tool, called RSVP (Rover Sequencing and Visualization Program) has been used with the Mars exploration rovers [19].

Despite the wealth of research in autonomous robotics and in control architectures for space robots, relatively little has been done to address specifically the needs of autonomous space robots. NASA/JPL have been developing/proposing three different architectures for applications with a higher degree of autonomy in the last few years: FIDO, CAMPOUT and CLARAty. FIDO [1] is a three-layer software architecture for real-time control of single rover systems equipped with scientific payloads. CAMPOUT [15] is a control architecture for the real time operation of multiple rovers. CLARAty [21][31] is a proprietary architecture that is becoming a requirement for many missions. The main goal of CLARAty is to provide a systematic framework for treating all the different vehicles/robots/instruments involved in Mars missions. CLARAty is object oriented and according to the publications provides a high degree of reusability. In a similar effort, the Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) has developed a software architecture for autonomy [1]. This architecture is composed of two main levels: a decision level whose role is to make plans and supervise their execution and a functional execution level whose role is to carryout low-level actions. The work of the LAAS is documented in this book in the chapter by R. Chatila et. al.

In parallel with these efforts, the Canadian Space Agency has been developing the Autonomous Robotics and Ground Operations (*ARGO*) software suite. *ARGO* provides a framework for the integration of the space operations process from planning to post-flight analysis. The objective of *ARGO* is to reduce operational costs and increase efficiency by providing operator aids and permitting the implementation of a level of autonomy appropriate to the application.

The target applications of the *ARGO* framework [18] cover the full spectrum of autonomy from supervisory control such as might be expected for ISS robotics to more autonomous

operations such as would be encountered in planetary exploration missions. It also covers the full range of space robotic applications from orbital manipulators to planetary exploration rovers. One of the important features of the *ARGO* framework is that it does not provide a universal architecture for ground control and autonomy. Instead, *ARGO* provides a set of toolboxes that can be assembled in a variety of manners depending on the application and its requirements. To facilitate the re-use of software, the design is modular and portable to the maximum extent possible.

Several toolboxes already exist within the *ARGO* framework. The central element of the *ARGO* framework is the *Cortex Toolbox*, which provides a set of tools to implement on-board autonomy software based on the concept of hierarchical finite state machines. The *Cortex Toolbox* allows an operator to graphically generate the behaviours to be implemented on the remote system. It automatically generates the code to be uploaded and it can be used to debug and monitor the execution of the autonomy software on-line and off-line.

A *Re-configurable Ground Control Station* has been implemented, which allows operators to easily build an operator interface to send commands and monitor telemetry. This interface is intimately tied to the *Remote and Log Toolboxes*. Using this tool, an operator can generate a control station in a few minutes and even customize the control station on the fly. Finally, the *Remote Toolbox* is used to distribute capabilities among different platforms. It is the central mechanism used for remotely accessing devices and connecting to telemetry streams.

2 CONCEPT OF OPERATION

The basic premise behind the design of *ARGO* is the integration into a single environment of the operations process from planning through verification to execution and post-flight analysis. *ARGO* provides a set of tools that can be connected in context-dependent configurations. For example, to plan and verify command sequences, the command and telemetry interfaces can be connected to a virtual environment composed of a simulation model of the system to be controlled along with a graphical rendering of the system and its environment. At run-time, the same command and telemetry interfaces get connected seamlessly to the real system in space to upload the pre-verified command scripts. After execution, it is possible to connect the telemetry interface to the logged telemetry files to conduct off-line post-flight analyses.

One key feature of *ARGO* is the capability to implement varying levels of autonomy as appropriate for the target application. One of the key determining factors is whether the information being fed back to the operator is still current or has gone stale. Autonomy is not required when the operator has adequate, up-to-date information and the ability to intervene in a timely manner. However, if the operator only receives out-of-date information or loses the ability to intervene, then he is not capable of making timely decisions and some amount of local decision-making capability is required at the remote site. Factors that can influence the level of autonomy required include long time delays, low communication bandwidths, intermittent windows of communication, poor situational awareness and a dynamic environment.

For example, in the presence of communication links with low latency, high bandwidth and high reliability, it is possible to maintain the operator in the loop for every decision. Primitive commands can be sent to the remote robot one at a time and confirmations can be requested from the operator before any command is executed. In such a case, the operator can intervene and over-ride the robot in case of anomaly.

In contrast, a rover on the surface of another planet is subject to communications with long delays, narrow bandwidth and frequent blackouts. The operator has relatively poor situational awareness because of the bandwidth limitations and no ability to intervene in a timely manner because of the long delays and frequent blackouts. The environment is unstructured and, in some

cases, could even be unknown to the operator. In this case, it is preferable to implement some autonomous decision-making capability. Relying on the operator for every decision will result in long idle times between communication windows while the robot is waiting for instructions.

To implement such a variety of levels of autonomy, the *ARGO* framework makes use of concepts such as command scripts and autonomous behaviours. Command scripts are files in which sequences of commands are recorded for automatic execution. The scripts are built using the finite state machine formalism, which is a convenient powerful way to represent logical rules. Commands are represented as discrete states that are linked by event-driven state transitions. In this context, states represent individual actions to be performed by the robot and transitions are events that cause the script to leave its current state to move into the next one. Typical state transitions can include external events such as sensor values and operator inputs, as well as internal events such as completion of the previous command or failure to complete it. The finite state machine formalism allows the implementation of decision-based branching as well as loops in the script. The simplest incarnation of a command script is a linear series of primitive commands for the robot to execute from start to finish.

Autonomous behaviours are built using the same methodology as scripts. In fact, behaviours can be seen as sub-scripts that can be invoked to handle some pre-determined conditions that the robot is expected to face during its mission. Whereas scripts are specific to one particular scenario, behaviours are libraries of action sequences to be undertaken by the robot under triggering conditions. Behaviours, therefore, provide the robot with some amount of reactive autonomy to make decisions and take action on a determined set of events or conditions.

Behaviours can be hierarchical in nature: i.e. the states in the finite state machine composing a behaviour can themselves be behaviours. Behaviours provide the capability for operators to generate much more compact command scripts since complex operation sequences can be encapsulated in a single command that can be invoked at different times in a script or even by another behaviour.

The fact that *ARGO* treats behaviours in the same manner as command scripts allows the operator to program, verify and uplink autonomous behaviours in the same development environment that is used for operations planning. Thus, *ARGO* truly provides an integrated environment for all operations-related issues from design and testing of autonomous behaviours to planning, verification and execution of command scripts.

3 THE *ARGO* TOOLBOXES

Given the diversity of applications targeted by the *ARGO* framework, it was recognized very early that “one solution cannot fit all”. The design philosophy behind *ARGO* is therefore to provide a set of toolboxes that can be assembled as needed to suit the operational needs of the target application. In order to facilitate the re-use of these software toolboxes from one application to the next, the design is modular and portable to the maximum extent possible. This is accomplished by using recognized standards such as the Java programming language, the Eclipse development environment, the XML standard and UML software development methodologies. These facilitate software portability and have accelerated the development cycle. All of the toolboxes interact via the *Remote toolbox* (described later), which enables seamless integration and easy reconfigurability.

A lot of attention has been dedicated to ensuring the usability of the toolboxes: User interfaces are designed keeping in mind that the operator should concentrate on solving operational problems instead of developing code. Graphical programming aids provide an intuitive interface for developing scripts and behaviours as well as for assembling graphical user interfaces for the reconfigurable ground control station (*RGCS*). The *RGCS*, itself, makes

extensive use of graphical tools for easy monitoring of telemetry. All command and telemetry channels are automatically identified in a database built by the *Remote toolbox* to ease the process of assembling the application-specific software infrastructure.

3.1 The *Cortex* Autonomy Toolbox

The central element of the *ARGO* framework is the *Cortex Toolbox*, which is used to implement command scripts and sets of reactive behaviours. *Cortex* has been developed in light of the fact that the development of such behaviour sets rapidly becomes labour intensive even for relatively simple systems when using low level programming languages, thus making reusability very difficult if not impossible. *Cortex* is based on the Finite State Machine (FSM) formalism, which provides a higher-level way of creating, modifying, debugging, and monitoring such reactive autonomy engines. Some advantages of this representation are its intuitiveness and the ease with which it can be graphically constructed and monitored by human operators.

The concept of hierarchical FSM allows a high-level FSM to invoke a lower-level FSM. This provides the capability to implement hierarchical task decomposition from a high-level task into a sequence of lower-level tasks. If the FSM is implemented in a modular fashion, it allows the implementation of the concept of libraries that provide the operator with the re-use of FSM from one application to another.

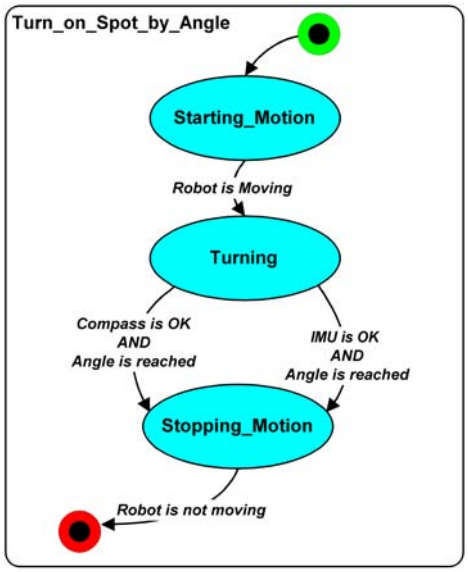


Figure 1 - A Simple Finite State Machine

In general, FSMs are used to represent a system using a *finite* number of configurations, called *states*, defined by the system parameters or its current actions. In the FSM shown in *Figure 1*, the states are **Starting_Motion**, **Turning,** and **Stopping_Motion.** In this case, actions are defined within the state, and they occur during state entry, re-entry and exit. For example, in *Figure 1*, when entering the **Turning** state, the current robot azimuth angle could be recorded to serve as a starting point for the destination angle computation.

The system can *transition* from one state to another based on its current state, conditions and outside events. Conditions on transitions are often referred to as *Guards*, and are implemented as statements that can be evaluated as being either true or false. Outside events, called *Triggers,* make the FSM evaluate its transition's guards and enable a transition to occur. In *Figure 1*, the system will transition from **Starting_Motion** to **Turning** once the robot motion is confirmed, or from **Turning** to **Stopping_Motion** if the compass is functioning correctly and its

readings confirms the robot has turned by the specified angle. In this particular FSM, no specific Triggers have been defined, so any Trigger (such as a periodic "CLOCK" event) will make the FSM evaluate its transitions.

A set of states connected together by transitions forms a state *machine*. In *Figure 1*, the **Turn_on_Spot_by_Angle** block is an FSM that implements a behaviour that has a mobile robot turn on the spot. The FSM also contains parameters it uses to make decisions (such as the current robot heading and the commanded angle of the turn) and paramaters on which it acts (the robot itself).

In *Figure 1*, **Starting_Motion** is represented as a single state. However the logic to be used in this state could be complex. In order to represent the state logic, the user can decide to use a sub-FSM in its place. The result is a *Hierarchical Finite State Machine* (HFSM) as shown in *Figure 2*.

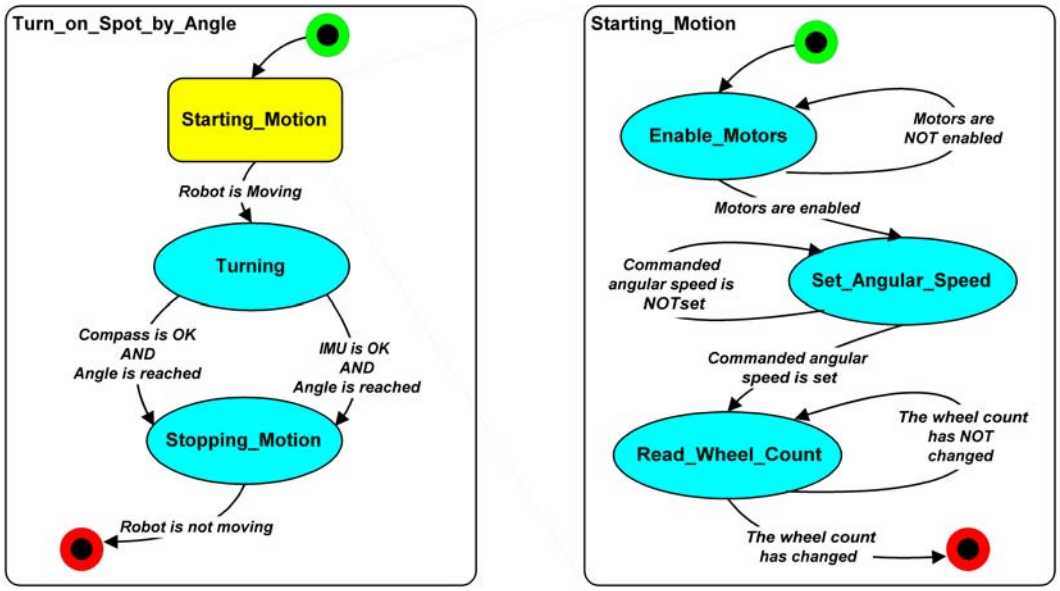


Figure 2 - A Hierarchical Finite State Machine

In this case, the **Starting_Motion** sub-FSM can be made modular by specifying the robot on which it acts as an input parameter. Defining parameters required or produced by a FSM defines its interface to the outside world thus allowing its reuse in various higher levels FSMs, an approach that has been successfully used in software libraries for years. This is the strategy used by *Cortex* to provide FSM modularity and reusability.

Graphical User Interface

The use of an intuitive graphical representation of FSM (see *Figure 3* and *Figure 4*) by *Cortex* allows the developer/operator to concentrate on the problem to be solved instead of concentrating on the programming skills to implement the solution.

The use of hierarchical FSM can address a wide spectrum of autonomy levels, from simple sense and react behaviours relying on sensor input to high-level mission scripts. For example, *Cortex* can be used to implement an emergency stop when an obstacle is detected by a sensor, as well as the higher level behaviour that will have the robot follow the obstacle edge until it can resume its original trajectory toward a user specified destination.

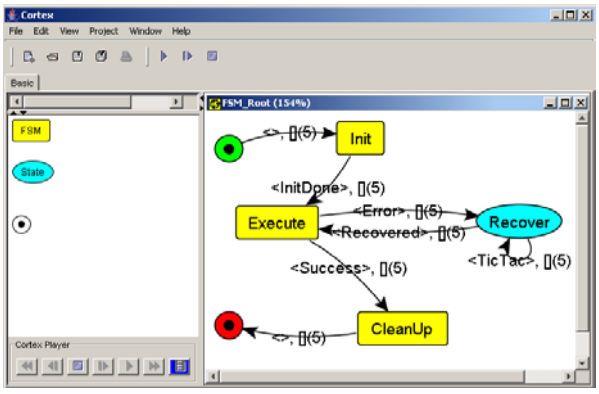


Figure 3 – Cortex GUI: FSM Editing

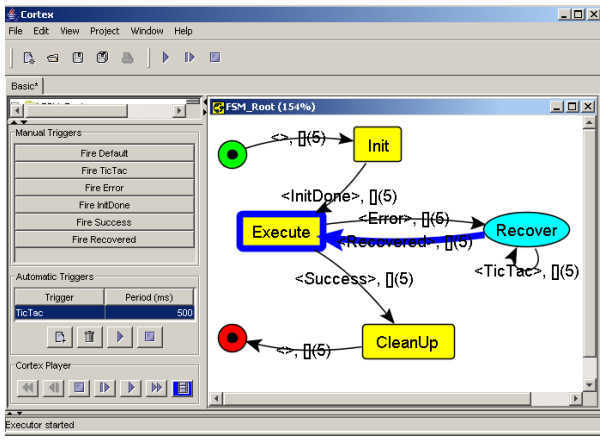


Figure 4 – Cortex GUI: FSM Monitoring

FSM are assembled graphically using States, sub-FSM, junctions (a construct used to combine transitions) and transitions (*Figure 3*). The operator can provide JAVA code snippets for state actions and transition's guard expressions and can define the inputs, local variables, and output parameters of each sub-FSM. The user can also assign a priority to each transition to force the order in which their guards are tested during execution. Constructs from other FSM can also be graphically "cut-and-pasted" to the current FSM to allow for the reuse of existing FSMs.

FSM Implementation Generation

The current implementation of *Cortex* provides an automatic code generator that produces JAVA source code to implement the FSM defined by the user. The code generator provides FSM topology checking to detect unreachable states and transition loops. A compiler module then compiles the code, reports problems, and helps the operator localise errors by highlighting FSM components where code is in error. At the moment, *Cortex* does not yet perform advanced forms of formal system verification such as temporal or logical checks. Such verification mechanisms will eventually become essential for the verification and validation of the generated software though methods other than extensive simulation.

Remote Operations and Monitoring

Once the FSM code has been successfully generated and compiled, the operator can monitor its execution on line, as well as interact with the FSM execution by firing events, as shown in *Figure 4*. The FSM graphical representations are animated in real-time to provide a monitoring mechanism to the operator/developer. This includes highlighting of current state and the transitions that led to that state, as well as the status of the FSM parameters.

The *Cortex* GUI can record all FSMs events in a file. This allows for a step-by-step playback of the FSM execution to assist in the analysis and debugging of FSMs execution. The playback supports all of the on-line graphical animation.

Cortex fully integrates the *Remote toolbox* functionalities. This allows the *Cortex* GUI and the FSMs to be distributed on different machines, which, in turn, allows FSM to be monitored and controlled remotely.

The distribution can also be performed at the FSM/sub-FSM level. Such a feature supports the distribution of sub-FSMs on different computers, a capability that can be used in multiple robots scenarios.

3.2 Reconfigurable Ground Control Station

The second key component of the *ARGO* framework is the Reconfigurable Ground Control Station (*RGCS*). The *RGCS* has been developed in reaction to the fact that projects in remote systems or autonomous systems all require a control station for the operator and that the development of this station is typically a very tedious, lengthy process that re-develops the same capability every time. To overcome this problem, the *RGCS toolbox* offers a set of standard graphical user interface (GUI) tools that readily connect with a remote system through the *ARGO* framework. It connects with remote systems through a set of commands and telemetry interfaces. The list of available control capabilities and telemetry signals is built automatically since the *RGCS* connects to the database that is used to generate the on-board software of the remote system. The *RGCS* palette contains a list of graphical widgets used to send commands and to display telemetry. The operator can then build a GUI rapidly by clicking and dragging items on a control panel and linking them to the resources being controlled or sending telemetry. ***Figure 5*** shows a simple implementation of a command and telemetry panel for a manipulation application.

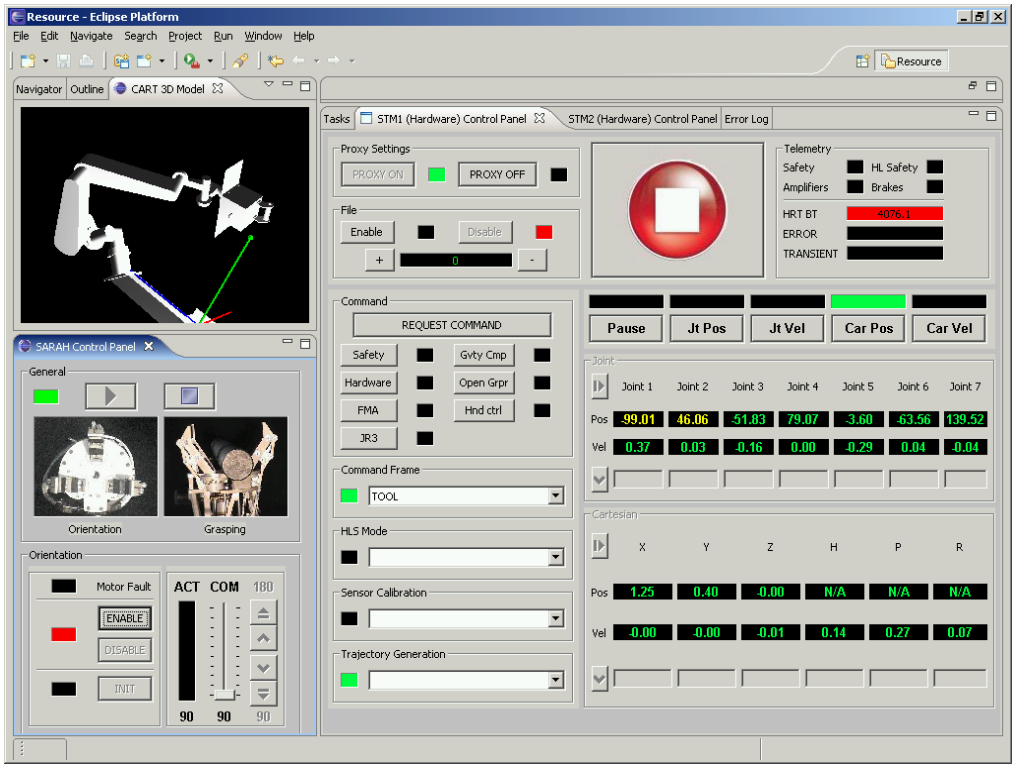


Figure 5 - Snapshot of the Remote Ground Control Station

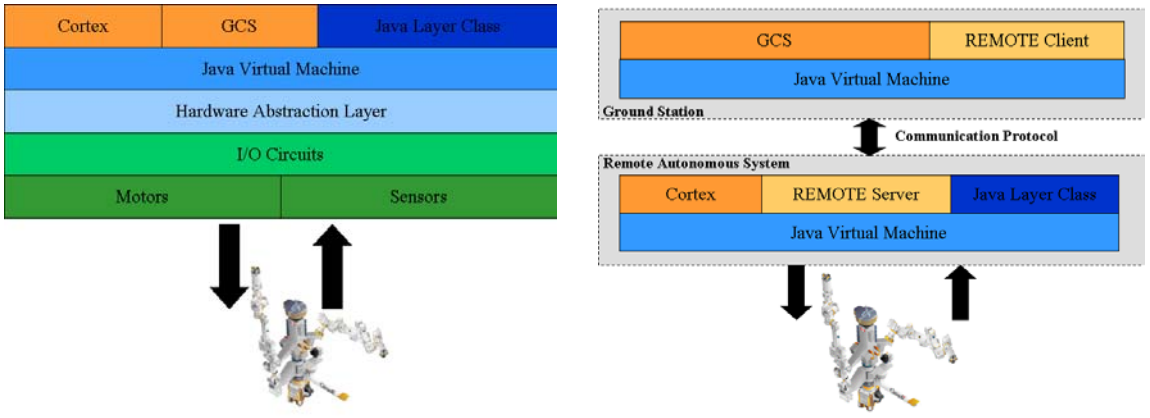
The addition of context dependent panels allows the entire operations process to be run on a single control station from mission planning to verification, execution and post-mission analysis. The operator can set up context-dependent control pages connected to different telemetry sources or commands sinks. For example, in the verification phase, it is possible to send commands to a simulator of the system and simulate its response before uploading the command to the real system. In the execution context, the commands are then sent to the real autonomous system and the telemetry is displayed on the station. The same interface can then be used in post-mission analysis context to display telemetry from a log file to analyse in detail the execution of a past mission. The addition of a graphical rendering application provides a powerful interface for easy interaction with the remote robot.

3.3 The *REMOTE Toolbox*

The *REMOTE* [1] *Toolbox* is the glue that is used to integrate the toolboxes available in the *ARGO* framework. It provides a set of standard Application Programming Interfaces (API) unifying the methods used to send commands and receive telemetry between sub-systems. The *Remote Toolbox* can be used to connect a ground station to a remote robot, to integrate inter-acting sub-systems on the remote robot or even to link ground stations together. It provides generic, reliable and manageable tools for remote-system control.

Any system that can be locally controlled through a Java class layer can be remotely controlled through the *Remote* API. Its modular construction allows easy interfacing with any communication protocol or media. The *Remote* API allows the software developer to make abstraction of the fact that the software is distributed over several processes and platforms. Each interface is coded with an abstraction layer that is handled by the *Remote* API through a Java Layer.

Figure 6 shows how the Remote Toolbox allows the developer to move easily from a local implementation to a remote implementation. In the latter case, *Remote* client-server pairs are created to handle the exchange of commands and telemetry in a manner that is completely transparent to the operator and the developer.



(a) Local Execution (b) Remote Execution

Figure 6: Remote Architecture Usage Sample

[1] Remote stands for Remote Execution and Monitoring of Objects for Teleoperation Environments

The *Remote* API has been developed as a multi-user, multi-target system and provides mechanisms for conflict-resolution among users. It offers monitoring and administration tools, which allow some users to observe and eventually interfere with other users' actions, without compromising system stability. Extensive error-tracking and execution-feedback features are also part of this toolbox, as well as abort mechanisms.

The *Remote toolbox* provides a set of meta-commands to manage and monitor the submission/execution queue of commands to a remote device. In particular it allows the user to cancel previously submitted commands that are queued for execution, to monitor the progress of commands, from acknowledgement to completion, and to be alerted on any error or abnormal situation that occurs during the execution process. It allows safe multi-user operation, together with priority mechanisms that prevent conflicts. It also provides both blocking and non-blocking operation, as well as batch-mode command envoy. Finally, *Remote* also provides advanced scheduling and prioritizing mechanisms.

The toolbox provides default TCP-IP and Local communication protocols. It also offers the developer the option to provide his own protocol (e.g. Serial, UDP). First tests have been recently performed with SCPS[2]-TP technology. SCPS defines a set of revisions to the Internet protocols to enable them to operate properly over stressed communications links. It extends normal TCP/IP to become an 'upgraded' TCP for space.

4 SAMPLE CASES

The *ARGO* framework has been applied to a few reference cases typical of space robotics applications. Two examples are described in the following section. The first scenario is a satellite servicing application in Low-Earth orbit. This is representative of most robotic manipulation tasks in Earth orbit where the environment is known and structured but it is dynamic since the satellite to be captured is in free flight. Bandwidth limitations and communication dropouts dominate the quality of the communication link. The satellite-servicing scenario is implemented on a robotic test-bed in laboratory settings.

The second scenario is a planetary exploration scenario where a rover autonomously explores the surface of a remote planet. In this case, the communications are subject to long delays and narrow communication windows. The operator must uplink a command script for a long period between communication opportunities and cannot intervene during operations. This scenario is implemented on a laboratory rover used in an outdoor test facility that is representative of the Martian topography.

4.1 On-Orbit Servicing Scenario

The first category of robotic missions driving the requirements of *ARGO* is autonomous on-orbit servicing of failed/failing spacecrafts. The commercial viability of such operations will require the usage of an efficient operations planning, verification and execution process. In addition, some basic autonomy capabilities will be required to perform the capture of the client satellite to be serviced.

Such capabilities will be demonstrated during the TECSAS mission [28] currently planned for 2009 by the Deutsches Zentrum für Luft und Raumfahrt (DLR). TECSAS is aimed at the in-flight demonstration of key technologies for on-orbit servicing. The mission consists in sending a servicer satellite equipped with a robotic arm to capture a client satellite on which maintenance

[2] Space Communications Protocols Specification

operations will be performed on orbit. It will demonstrate the capability of rescuing a scientific micro-satellite with failed control system and prolonging its operational life in space.

The Canadian Space Agency will participate in the TECSAS mission. One of its key contributions to the mission is the utilisation of the *ARGO* Toolboxes for the autonomous capture of an uncooperative satellite. In this context, an uncooperative satellite is defined as a satellite that was not designed specifically for servicing and/or that does not participate actively to the rendezvous and capture.

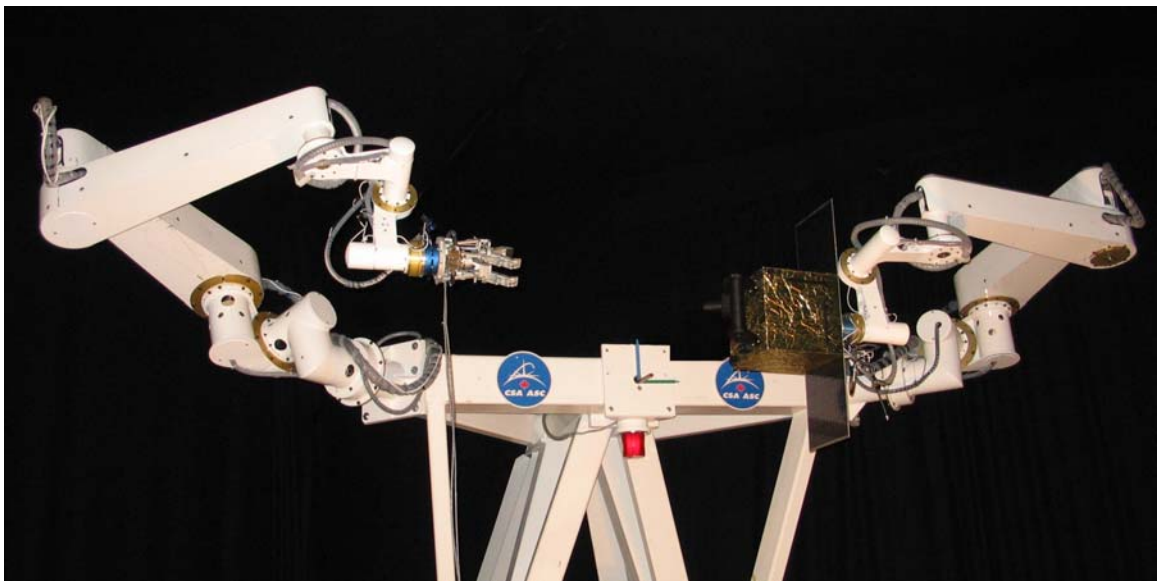
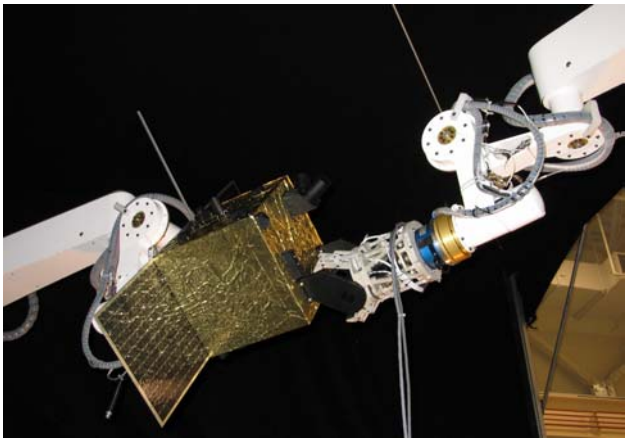


Figure 7:CSA Automation Robotics Testbed

In preparation for TECSAS, the *ARGO* technologies are validated in the laboratory on the CSA Automation and Robotics Test-bed (CART). This test-bed, shown in Figure 7, is composed of two 7-degree-of-freedom (DOF) manipulators. One of the manipulator arms is equipped with a mock-up of the client satellite in 2/3^{rds} scale. The manipulator is used to emulate the motion of a tumbling satellite, maintaining constant angular momentum. The other manipulator emulates the motion of the arm on the servicer satellite and is used for the capture.

The robotic end-effector on the servicer arm is a SARAH hand developed at Laval University [17]. SARAH, shown in Figure 8(a), is a 10-DOF adaptable under-actuated hand. The hand, because of its capability to adapt mechanically to different shapes, does not require a specialized grapple fixture. An active vision system, the Laser Camera System (LCS) of Neptec shown in Figure 8(b), is used to visually acquire the pose of the client satellite in order for the manipulator to perform appropriate manoeuvres.



(a) (b)

Figure 8: (a) The SARAH Hand grasping the target satellite; (b) Laser Camera System (LCS)

The overall implementation of this On-Orbit Servicing (OOS) demonstration on the CART test-bed is presented in Figure 9. The overall control architecture of the two robotic arms is implemented in Matlab/Simulink. The execution code is automatically generated using the Real-Time Workshop toolbox of Matlab and is compiled and run on a cluster of Pentium IV computers operating under the real-time QNX environment.

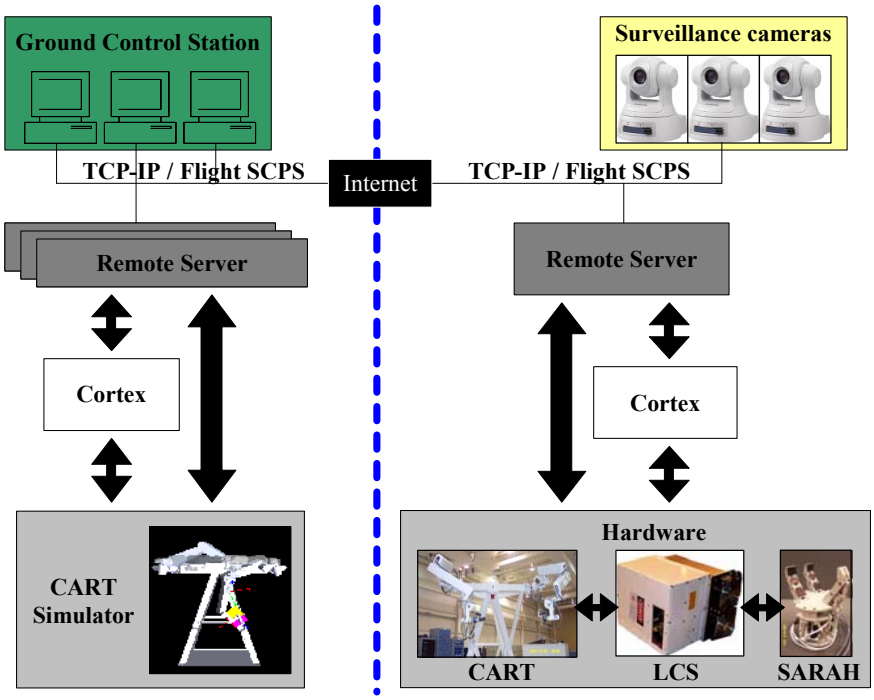


Figure 9: Overall implementation of the CART testbed

The *Cortex Toolbox* is used to implement the behaviours required for the autonomous capture of the client satellite. On TECSAS, the operator will be responsible for the planning and execution of the long-range rendezvous of the two spacecrafts. The autonomy engine will take control when the two spacecraft are distant by a few meters. It will be responsible for performing the final approach of the servicer spacecraft to the client, deploying the manipulator arm and performing the capture of the slow spinning/tumbling client satellite.

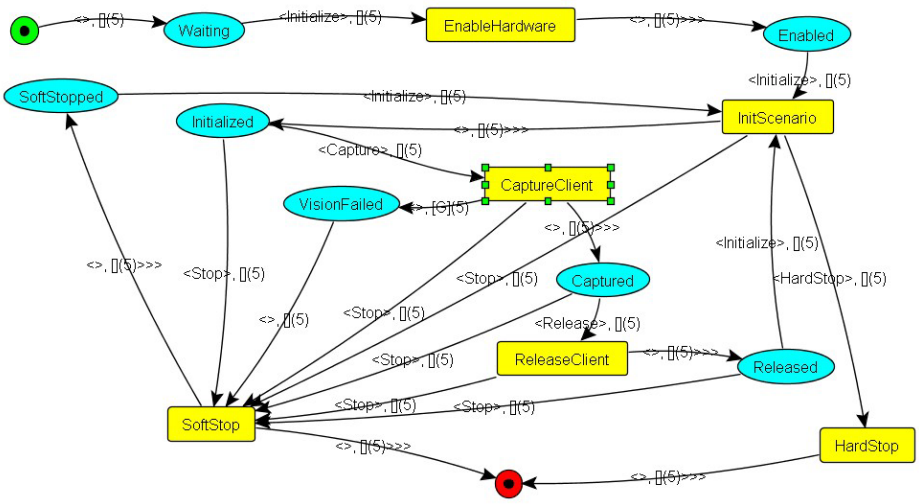


Figure 10: ***The Cortex engine for the capture scenario in the CSA lab.***

Transitions between phases of the operation are triggered by sensory events. The *Cortex* engine considers anomalies such as the possibility of the client spacecraft to drift out of the capture envelope of the manipulator (through translation or rotation), blinding of the vision sensor or loss of sight, reduction of the safe distance between the two manipulators below an acceptable limit, or failed capture which results in the client satellite being sent into a tumble. Figure 10, presents a *Cortex* implementation of the later phases of a typical OOS after an autonomous far rendezvous has been performed. Following a modified version of the basic behaviour pattern, the operator has overriding control of pausing the process using a soft stop and then restarting it, or terminating the process by using a hard-stop. The actual capture sequence is itself a state machine of three stages (approach, align, and capture).

In this architecture, the Ground Control Station is first used to develop the command scripts and autonomous behaviours using the *Cortex toolbox*. In this context, the *RGCS* is connected to a simulator of the remote system, which includes the dynamics of the two satellites, the robot arm and the vision system. The simulator is then used to validate the command scripts and the autonomous behaviours associated with each phase of the mission. Depending on the predictability of the parameters triggering transitions in the *Cortex* engine, some portions can be simulated in a very deterministic manner. For example the manipulation of Orbital Replacement Units (ORU) is performed in a very controlled and static environment, the client satellite itself, and is subject to very little uncertainty. On the other hand, other portions of the mission, such as the capture of the client satellite, will be subject to factors such as illumination conditions and initial conditions on satellite attitude, which are much more unpredictable. The validation of the portions of the command script and autonomous behaviours associated with these events will therefore require validation using a broader range of parameters to systematically verify the plausible outcomes of the mission.

After verification, the *RGCS* is reconfigured to connect to the real system. The validated *Cortex* behaviours and scripts are then uploaded to the system for execution. Depending on the communication windows, it is possible to require operator confirmation before the performance of key steps in the command script. The synergy of the *Cortex* autonomy engine and the *Remote toolbox* allow for operator intervention at different phases of an operation without hindering the autonomy nature of the operation. For example, operator confirmation could be requested before starting the final capture sequence. This approach would simplify the interactions required

between the attitude and orbit control systems of the two spacecrafts by leaving the coordination to the ground control personnel.

4.2 Planetary Exploration Scenario

The second category of missions driving the requirement of the *ARGO* framework is planetary exploration-class robotic missions. The current approach used for planning, verifying and executing operations of planetary rovers relies extensively on large teams of operators on the ground to plan the navigation and scientific operations. Because of the long time delays and narrow communication windows, command scripts must keep the robot busy for long periods of time and operators cannot intervene during the command execution. Increasing the autonomy of planetary rovers offers the potential to greatly enhance the scientific return on investment by increasing the productivity of the robot.

To validate the *ARGO* tools for this category of missions, a capability for autonomous long-range rover navigation has been implemented using a laboratory rover and a terrain representative of the Martian surface. In this case, long-range navigation is defined as navigation of the rover to a final destination located beyond its visual horizon. The rover must therefore be able to navigate through an environment in which operations have not been deterministically validated by simulation.

The Mobile Robotics Test-bed (MRT) used for long-range navigation experiments is based on a Pioneer P2-AT robot (see *Figure 11*). The sensor suite of the MRT includes wheel encoders for odometry, a six-axis Inertial Measurement Unit (IMU), a digital compass for azimuth as well as sonars and optical proximity sensors for obstacle and hole detection. A scanning lidar sensor (ILRIS 3D unit from Optech) is used to obtain detailed terrain models for path planning. The experiments are conducted in the Canadian Space Agency's Mars emulation terrain, a facility emulating several different kinds of Martian topography (plains, hills, canyons, rock fields, craters) over an area of 30 meters by 60 meters (see *Figure 12*).

The proposed approach to long-range navigation requires 3D environment sensing, feature-based localization, inertial navigation and guidance and map-based path planning, all under FSM based reactive behaviours.

Figure 11 - The Mobile Robotics Test-bed

Figure 12 - The Mars Emulation Terrain

As in the On-Orbit Servicing scenario, *Cortex* is used to implement all autonomous behaviours.

Figure 13 shows the high level decomposition of the behaviour required for navigation over the horizon. Every box represents a hierarchical finite state machine (FSMs) that is in itself decomposed into smaller finite state machines, each one implementing simpler behaviours.

The default entry point for this behaviour is a call to the lower-level behaviour used to scan the terrain using the lidar sensor thus obtaining a dense 3D surface scan. The Scan behaviour is also used to generate a triangular mesh of the recently scanned terrain model. Nominally, the next action taken by the FSM is to call the Localize behaviour to position the robot in the world model using an iterative closest point algorithm [11]. This behaviour also merges the recently scanned terrain model with any a-priori terrain models obtained from previous observations. The next step is then to plan a path towards the final destination using a graph search algorithm on the triangular mesh.

The Navigate behaviour decomposes into lower level behaviours responsible for executing the planned path and reacting to the presence of obstacles. Several events can cause transitions to occur in the High-Level Navigation behaviour. Examples of such events include the detection of obstacles, the presence of a high terrain uncertainty index, or failure of any of the lower level behaviours to complete successfully. For simplicity of illustration, the FSM shown in *Figure 13* has been depicted without most of the transitions associated with error conditions.

Each of the lower level behaviours is typically based on the behaviour template shown in *Figure 14*. This template is mostly used for behaviours that run in a continual fashion (as opposed to a series of discrete events). It provides the standard modes in which the behaviours can be at any time (e.g. Initialising, Running, Pause, Recovering from error, etc.) along with the transitions that can be expected between these standard modes.

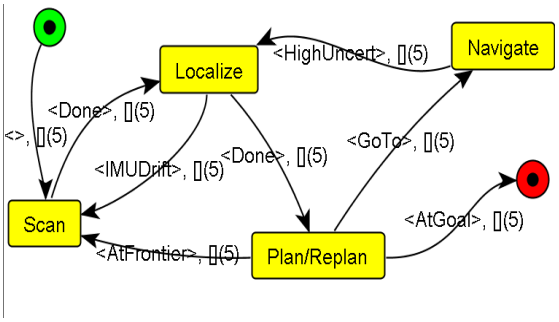


Figure 13 - High-Level Navigation Behaviour

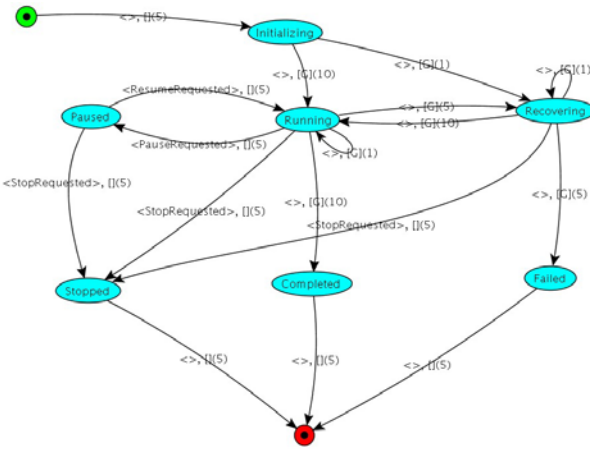


Figure 14 - Basic Behaviour Template

Because the rover nominally operates in an environment that is unstructured and only partially known, verification of behaviours and command scripts cannot be performed in a deterministic fashion before every uplink. It is therefore not necessary in this case to connect the Ground Control Station to a simulator in an attempt to predict the outcome of every operation. Instead, validation is done extensively when the behaviours are developed in an attempt to predict the reaction of the system within certain bounds of environment parameters. The *RGCS* connects to the remote equipment in "batch" mode uplinking behaviours and command scripts in batch and receiving packets of telemetry typically after the completion of the navigation task.

For the planetary exploration scenario, the modularity of the *ARGO* toolboxes allows us to combine different basic behaviours in a variety of ways and, as such, to identify recurrent behaviours. Moreover, the development of a basic library of behaviours enables the rapid prototyping and testing of a variety of algorithms.

5 CONCLUSION

Frameworks such as *ARGO* offer great potential to reduce the cost of operations through integration, and to enable more capable missions through the addition of autonomy. However, several important steps remain before such frameworks are accepted. The biggest problem it faces at the moment is its incompatibility with the current operations paradigms.

The operations culture of space robotics programs has been inherited from past manned programs such as Apollo where the impact of any failure could have been catastrophic on the survival of the crew. The current methodology for space robotic operations on manned programs such as the ISS requires the involvement of a local operator for control and monitoring and the support of ground support personnel to monitor the operations in real-time. All operations carried-out by these robotic systems are thoroughly verified in simulation prior to their execution in space. Scenarios are run to verify all possible combinations of nominal and anomalous conditions in order to guarantee safety and mission success under all foreseeable circumstances.

Increasing the level of autonomy of a space robot implies that some operational decisions will be made locally based on external events such as sensor data, or on internal events such as malfunctions. These events will certainly impact the execution of the sequence of operations. They are difficult to predict in a reliable fashion and, therefore, it will be nearly impossible to determine exactly the sequence of actions to be taken by an autonomous robot in an unknown environment. Task verification using the current paradigm would require predicting the exact sequence of events for nominal and off-nominal scenarios. Thousands of simulations would have to be run to generate all reasonable combinations of conditions to be expected and there would be no way to identify which of these simulation runs would in fact occur during the next operations cycle. At best, one could use a Monte Carlo approach to determine probabilistically the chances of success. This is obviously neither an efficient nor an acceptable way of ensuring dependability.

The operational procedures currently used depend on the deterministic prediction of the behaviour of robots during every operations sequence. This will not be possible for autonomous robots operating in the presence of uncertainty. A plethora of external factors such as environmental conditions, over which operators will have little or no control, will affect the outcome of autonomous sequences. At best, analyses could be made to guarantee the safe operation of the autonomous system and assess the success probability within certain bounded conditions. This will require a shift in the operations paradigm trading a potential increase in the productivity of space robots for the acceptance that less than 100% of planned operation sequences will be conducted with success on the first attempt.

The next steps to gain acceptance for the *ARGO* framework is to perform extensive field-testing. *ARGO* is already central to all robotics laboratory activities led at the Canadian Space Agency and plans are being made for its usage in mining/drilling applications as well as for the control of unmanned vehicles. The true test will be its usage in the TECSAS mission currently being planned by the CSA and DLR. TECSAS will be a satellite servicing technology demonstration mission. It will involve the rendezvous and capture of a client micro-satellite by a servicer equipped with a robotic arm. The current plans of the CSA are to obtain flight heritage for several *ARGO* toolboxes through TECSAS. The target application is the fully autonomous capture of the freely spinning client micro-satellite by the servicer without the necessity for human intervention.

Frameworks like *ARGO* offer tremendous potential to reduce the cost of operations and to increase the productivity of space robots. Such capabilities undoubtedly will have a critical impact on all space robotic operations in the near to mid-term future. In fact, they will be crucial to the successful implementation of the exploration plans being put forward by the major space agencies of the world. The current plans for robotic missions call for ever more ambitious

missions with increasing expectations in terms of productivity and scientific return on investment. In several cases, the key to increasing capability and reducing operational complexity/cost will be to augment the local decision making capability thus reducing the need for active human involvement. Extrapolating costs based on past experience with programs such as the International Space Station, it is clear that a shift will have to take place in the operations philosophy in order to reduce cost and augment capabilities. Similarly, if commercial on-orbit servicing is to become a reality, frameworks like *ARGO* will play a key role in bringing the cost of operations to an acceptable level.

6 REFERENCES

[1] Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F., "An Architecture for Autonomy", International Journal of Robotics Research, Vol. 17, no. 4, April 1998

[2] Baumgartner, E.T., "In-Situ Exploration of Mars Using Rover Systems", Proc. of the AIAA Space 2000 Conference, Long-Beach, CA, USA, September 2000.

[3] Brooks, R. A. "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, Vol. 2, No. 1, March 1986, pp. 14-23; also MIT AI Memo 864, September 1985.

[4] Brunner, B., Landzettel, K., Schreiber, G., Steinmetz, B.M. and Hirzinger, G., "A Universal Task-Level Ground Control and Programming System for Space Robot Applications - the MARCO Concept and its Applications to the ETS-VII Project", Proc. Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS 99), ESTEC, Noordwijk, The Netherlands, pp.217-224, June 1-3 1999.

[5] Dudek, G. and Sim, R., "RoboDaemon - A device independent, network-oriented, modular mobile robot controller", Proceedings of the IEEE Conference on Robotics and Automation (ICRA), Taipei, Taiwan, 2003.

[6] Dupuis, E. and Gillett, R., "Remote Operation with Supervised Autonomy", 7th ESA Workshop on Avanced Space Technologies in Robotics and Automation (ASTRA 2002), Noordwijk, The Netherlands, November 2002.

[7] Dupuis, E. Gillett, R., Boulanger, P., Edwards, E. and Lipsett, M., "Interactive, Intelligent Remote Operations: Application to Space Robotics", SPIE Telemanipulator and Telepresence Technologies VI, Boston, Septembre 1999.

[8] Dupuis, E., Gillett, R., L'Archevêque, R. and Richmond, J., "Ground Control of International Space Station Robots", Journal of Machine Intelligence and Robotic Control, Special Issue on Space Robotics, Vol. 3, No. 3, pp. 91-98, September 2001.

[9] Edwards, E., Lamorie, J., Younghusband, D., Brunet, C. and Hartman, L., "Independent SCPS-TP development for fault-tolerant, end-to-end communication architectures", in Proceedings of Datat Systems in Aerospace – DASIA 2002, July 2002, Dublin, Ireland.

[10] Fontaine, B., Steinicke, L., Visentin, G.," A reusable and flexible control station for preparing and executing robotics missions in space", International Symposium on Space Mission Operations & Ground Data Systems - 'SpaceOps 96', Munich, Germany; September 1996.

[11] Gemme, S., Nsasi Bakambu, J. and Rekleitis, I., “3D Reconstruction of Environments for Planetary Exploration”, Proc. of 2nd Canadian Conference on Computer and Robot Vision – CRV 2005, Victoria, BC, Canada, May 2005.

[12] Gerkey, B.P., Vaughan, R.T. and Howard, A., "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". In Proceedings of the International Conference on Advanced Robotics (ICAR 2003), pages 317-323, Coimbra, Portugal, June 30 - July 3, 2003.

[13] Gottselig, G., Orbital Express advanced technology demonstration (ATD), January 2003.

[14] Hayati, S.A. Udomkesmalee, G. Caffrey, R.T., “Initiating the 2002 Mars Science Laboratory (MSL) focused technology program”, IEEE Aerospace Conference, Golden, CO, USA, March 2004.

[15] Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Das , H., Aghazarian, H., Ganino, A.J., Garrett, M., Joshi, S.S., Schenker, P.S., “CAMPOUT: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration”, IEEE Transactions on Systems, Man and Cybernetics, Part A, , Volume: 33 , Issue: 5 , Sept. 2003.

[16] Konolige, K. and Myers, K. The Saphira architecture for autonomous mobile robots. In Artificial Intelligent and Mobile Robots, chapter 9,
pages 211--242. AAAI Press, 1998.

[17] Laliberté, T., Birglen, L., and Gosselin, C.M., *Underactuation in robotic grasping hands*, Japanese Journal of Machine Intelligence and Robotic Control, Special Issue on Underactuated Robots, Vol. 4, No. 3, pp. 77--87. 2002

[18] L'Archevêque, R. and Dupuis, E., "Autonomous Robotics and Ground Operations", Proc. of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space – iSAIRAS 2003, Nara, Japan, May 2003.

[19] Maxwell, S., Cooper, B., Hartman, F., Wright, J. and Yen, J., “The Design and Architecture of the Rover Sequencing and Visualization Program (RSVP)”, 8th International Conference on Space Oeprations, Montréal, Canada, 2004.

[20] Montemerlo, M., Roy, N., and Thrun, S., ”Perspectives on Standardization in Mobile Robot Programming : The Carnegie Mellon Navigation (CARMEN) Toolkit". Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003). Las Vegas, October, 2003.

[21] Nesnas, I.A., Wright, A., Bajracharya, M., Simmons, R., and Estlin, T., "CLARAty and Challenges of Developing Interoperable Robotic Software," invited to *International Conference on Intelligent Robots and Systems (IROS),* Nevada, October 2003.

[22] Oda, M., “Space Robot Experiments on NASDA's ETS-VII Satellite - Preliminary Overview of the Experiment Results”, Proc. 1999 IEEE Conf. on Robotics and Automation (ICRA 99), Detroit, USA, pp.1390-1395, 1999.

[23] Parker, L.E., ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation, IEEE Transactions on Robotics and Automation, 14 (2), 1998.

[24] Pedersen, L., Sargent, R., Bualat, M., Kunz, C., Lee S., , and Wright, A., "Single-Cycle Instrument Deployment for Mars Rovers", *Proc. of 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Nara, Japan, May 2003.

[25] Popov, A., "Mission Planning on the International Space Station Program. Concepts and Systems", IEEE Aerospace Conference, Big Sky, MT, USA, March 2003.

[26] Potter, S.D., "Orbital Express: Leading the way to a new space architecture", *2002 Space Core Tech Conf.*, Colorado Spring, November 2002.

[27] Smith, P.H., "The Phoenix Scout Mission", American Geophysical Union Fall Meeting Abstracts, December 2003.

[28] Sommer, B., On-Orbit Servicing of Satellites (OOS) as a major application field – The TECSAS mission, 54th International Astronautical Congress of the International Astronautical Federation (IAF); Bremen, Germany; Sep. 29 - Oct. 3, 2003.

[29] Van Winnendael, M., Gardini, B., Roumeas, R., and Vago, J., "The ExoMars Mission of ESA's Aurora Programme", Earth & Space 2004 - Ninth Biennial Conference of the Aerospace Division, Houston, Texas, USA, March 2004.

[30] Vaughan, R.T., Gerkey, B.P., and Howard, A. "On Device Abstractions For Portable, Resuable Robot Code".. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), pages 2121-2427, Las Vegas, Nevada, October 27 - 31, 2003.

[31] Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R. and Das, H., "The CLARAty Architecture for Robotic Autonomy", Proceedings of the 2001 IEEE Aerospace Conference, Big Sky, Montana, March 10-17, 2001

[32] Whelan, D.A., Adler, E.A., Wilson, S.B. and Roesler, G.M., "DARPA Orbital Express program: effecting a revolution in space-based systems", in Proc. SPIE Vol. 4136, p. 48-56, Small Payloads in Space, p. 48-56, Small Payloads in Space, November, 2000.