IP COMMUNICATION AND DISTRIBUTED AGENTS FOR UNMANNED AUTONOMOUS VEHICLES

Vincenzo Liberatore[1] , Wyatt S. Newman[1], and Kul Bhasin[2]
[1] Electrical Engineering and Computer Science Department, Case Western Reserve University, Cleveland, OH 44106-7071, {vxl11,wsn}@po.cwru.edu.
[2] NASA Glenn Research Center, 21000 Brookpark Rd., Cleveland, OH 44135.

Abstract

Unmanned Autonomous Vehicles (UAVs) extend the reach of human activities and exploration, and are currently of utmost interest to NASA Mars missions and Earth Science enterprises. At present, fully autonomous machines are poorly equipped to survive in realistically complex and uncertain environments. However, incremental progress and useful intermediate results can be achieved through collaboration between local autonomy and remote supervision. In this paper, we submit that these objectives can be accomplished through the use of IP (Internet Protocol) coupled with

- An appropriate design of the local controller,
- Real- and non-real-time middleware, and
- Agent-based software.

This paper describes our experience in the application of these methodologies for the remote supervision of intelligent machines.

I Introduction

Unmanned Autonomous Vehicles extend the reach of human activities and exploration, and are currently of utmost interest to NASA Mars missions and Earth Science enterprises. For example, UAVs will aid human astronauts in the exploration of Mars and are critical for NASA research in the Earth-sciences field. UAVs critically depend on communication to

1. Report their observations to human experts or to data repositories,
2. Obtain high-level direction concerning their tasks and objectives, and
3. Coordinate among multiple units.

Communication is currently implemented with proprietary, application-specific, and inflexible protocols that use point-to-point, fixed-bandwidth channels. The challenge is to enable UAV communication that supports scalable, evolvable, and intelligent protocols.

In this paper, we submit that these objectives can be accomplished through the use of IP (Internet Protocol) coupled with real- and non-real-time middleware and agent-based software. IP has now become a standard protocol for data networks due to its flexibility and generality.

UAV communication will benefit from IP in its cross-mission design, in the presence of a pervasive IP infrastructure, and from the reliability and the security provisions that are built in the protocol. Additionally, IP enables UAV software to adopt widely tested middleware and agent platforms.

Each UAV currently participates in two networks:

a. An external network that enables the coordination and communication of the UAV with other entities, such as peer UAVs or ground stations, and

b. An internal network that connects on-board sensors, actuators and other devices.

IP will enable the logical and programmatic unification of the external and internal network[1].

In previous work[2,3] (Figure 1), we have shown that this perspective enables remote supervision of manipulation tasks through stationary units, and achieves

1. The ability to interface with remote control components or human supervisors in the presence of time delays, both with and without Quality of-Service provisioning in the network infrastructure,
2. The ability to coordinate multiple robots and to aggregate robot teams into controllable units,
3. Control evolvability in terms of rapid reprogrammability (addition of new functionality after hardware deployment), dynamic reconfiguration (creation of new collections of vehicles into coordinated, task-oriented teams), and extensibility (growth

through modular incorporation of additional assets),

4. Survivability and fault-tolerance (automatic reallocation of communications software in response to component failures).

In this paper, we describe the challenges to bring this infrastructure into the UAV environment as well as our current accomplishments in this area. The hurdles posed by UAV ultimately derive from their mobile operations in unstructured environments with often unreliable wireless communication links. A fundamental issue is the design of the local control software to achieve a globally desirable behavior. Another critical issue is to exploit software agent mobility to achieve properties 1-4.

The paper is organized as follow. Section II reviews the relevance of UAV technology and communication for NASA. Section III describes the coordination of local and remote control. Section IV discusses the interface and protocol between a unit and a networked environment. Section V describes a scenario where an intelligent machine would autonomously start communication. Section VI outlines current middleware concept and how these can be used to achieve the objectives 1-4 above. Finally, Section VII concludes the paper.

II Relevance to NASA

The significance for NASA ultimately derives from the objective of extending space IP networks to support UAV communication in the following contexts.

NASA's Earth Science Enterprise incorporates long flight duration, autonomous, Uninhabited Aerial Vehicles (UAV's), such as the Altus and Proteus, for in situ and remote sensing measurement of stratospheric and tropospheric climate conditions. Likewise, the Space Science Enterprise is studying the use of UAV's such as the Aerial Regional-scale Environmental Survey (ARES) of Mars for exploring the Mars surface at close range with imaging instruments. UAV's are ideal platforms for in situ atmospheric measurements and for close range surface imaging over the entire optical spectrum.

As these vehicle come into operational use for Earth science and weather monitoring, it will be highly desirable to have their data reported directly to subscribers on the Internet so that they will use that data to initiate immediate responses to changing conditions, such as taking cover when a tornado approaches. Furthermore, the implementation of networking technologies on-board UAV's and the Mobile IP technologies that come with networking will handle hand-off between ground stations and make it possible for operations teams to stay in touch and in control as vehicles fly on cross-country sorties. Inter-aircraft networking could also be implemented to form in-sky relay networks to pass data along for direct to user downlinks almost anywhere UAV's may be in sight of one another, thus enabling over-the-hill warning capability.

III Local and Remote Control Coordination

At present, fully autonomous UAV's are poorly equipped to survive in realistically complex and uncertain environments. Endowing a machine with sufficient intelligence to behave competently in an unstructured world will remain a research challenge for the foreseeable future. However, incremental progress and useful intermediate results can be achieved through a collaboration between local autonomy and remote supervision.

In space as well as defense applications, the remote control is impeded by significant communications time delays and by potentially unreliable connections. Communications limitations thus impose a requirement on the remote machine to behave competently independently in the absence of supervisory guidance. At least for limited periods, the autonomous robot must guard its own health and, ideally, continue to function productively.

At the lowest levels of control, actuator efforts are adjusted in response to sensory feedback at rates typically on the order of 1kHz. Such servo-level controls must occur with little jitter and with high reliability. These requirements mandate that at least this lowest level of control be performed locally. The next higher level of control abstraction concerns the fine-grained time sequence state setpoints. Such updates in robot controllers typically occur at roughly 100Hz rates. This level of control could conceivably be executed through wireless communications and via IP communications, but the QoS requirements are relatively high to achieve smooth, coordinated motions. In our implementations, we have determined that setpoint updates (e.g., trajectory generation) are also best performed by a controller intimately connected to the machine.

It is at the higher levels of control abstraction where the options for distribution of controls become more varied. It is desirable for the control architecture to accommodate varying degrees of autonomy on demand. In some situations, it may be feasible and desirable for a human operator to directly control a remote vehicle or robot by real-time teleoperation. However, if there is a degradation of communications, or if the human supervisor is overloaded, it would be optimal for the

robot to assume higher levels of control responsibility. To address this range of needs dynamically, we have proposed a control architecture composed of software agents organized as "virtual robots" (VR's).

Our hierarchy of virtual robots consists of interfaces with increases levels of abstraction, ranging from low-level teleoperation to high-level task-oriented command interfaces. The lowest levels of the hierarchy are comprised of real-time controllers that are local to each autonomous robot. At this lowest level, the virtual robot abstraction becomes blurred with the physical robot and its local controller. At higher levels of abstraction, each virtual robot is a software agent with a generic design. Each virtual robot accepts commands at some level of abstraction, decomposes its incoming commands into finer-grained subtasks and carries out its subtasks through a sequence of commands of lower-level abstraction issued to lower-level virtual robots. The hierarchy is the graphical equivalent of a tree or an open chain. The human operation interfaces with the control architecture at the highest level of abstraction enabled at any instant, and the physical-level real-time controllers interact with the hierarchy at the lowest level.

IV Low-Level Interfacing: The Robot Proxy

At the lowest level of our hierarchy, virtual-robot software agents interact with a "robot proxy". The robot proxy constitutes the bridge between the generic interface among VR software agents and the physical interface specific to a particular machine's real-time controller.

Through experimentation, we have arrived at methods and observations regarding this lowest-level interface. To accommodate unreliable communications, we have imposed self-preservation requirements on the lowest level controller. This is accomplished through construction of a "command server" and "reflex" controls. The command server accepts input from an interface that is indistinguishable from a generic VR, and it pushes received commands onto a buffer to be executed by the real-time controller. The recognized commands are defined to be executable smoothly with low latency when the communication rates are high, and at the same time the command set is designed for graceful performance degradation as communications deteriorate. In our implementations, commands received at this level are executed at full fidelity when the update rates are approximately 10Hz or better. Example commands at this level are:

- `GoTo(x)`: accept a fine-grain perturbation of a state command
- `MoveTo(x)`: accept a higher-level motion command with implicit request for the local controller to execute a coordinated, dynamically-graceful transition from the current state to the new commanded state
- `MoveUntil(motion,condition)`: command the low-level controller to initiate a persistent motion with specified direction and speed until some terminating condition is detected
- `GetInfo(type)`: request information from the low-level robot controller, which may include state variables, sensor values, or command-server status.

Additionally, there are utility commands for engaging tools, end-effectors or other peripheral devices, and for aborting previous commands. It is our objective to deliberately keep the command set small, yet make the commands sufficiently capable that arbitrarily sophisticated behaviors can be composed from this small command set.

One of the requirements that makes it desirable to have a small command set is the need to implement self preservation. Although some levels of threat to an autonomous machine can only be recognized at higher levels of abstraction, other crises are best handled at the lowest level of control. For example, an impending collision may be recognized and averted by a low-level controller before higher levels become aware of the danger. We thus design into the low-level controller a reflexive behavior, which is responsible for quick reactions necessary for self preservation.

Our reflexive controller monitors state of the machine relative to the environment to detect short time-scale threats. At this level of abstraction, sensory processing is crude and only obvious and immediate hazards are detected and avoided. Such protection is necessary not only due to unexpected threats from the environment, but also due to inappropriate commands from higher levels. While it is the responsibility of higher levels of abstraction to generate goal-oriented commands, it is also necessary that the higher levels be insulated from the details of the low-level agents. Thus, a command may be issued that is dangerous or inappropriate for a given mechanism or vehicle. At the level of the robot proxy, the machine controller may not be aware of any higher-level goals, but it is better informed of the limitations particular to its hardware. Thus, commands that might result in damage to the machine would be refused by the command server. Additionally, erratic communications may result in reception of commands that were appropriate at some time in the past but which have become dangerous due to evolution of the

machine or its environment during transmission latency. Such commands must also be evaluated and potentially refused by the local reflex controller.

An example of the robot proxy with command server and reflex controller has been constructed at CWRU in the context of manipulation. The scene shown in Fig. 1 includes a collection of valves, switches and knobs designed for human manipulation. We have enabled a robot to manipulate these controls using our VR architecture implemented as a distributed, networked system. The robot proxy is responsible for accepting gross or fine motion commands. The reflex controller evaluates if the commands are reachable by the robot and refuses to accept commands that would result in damage. Additionally, the reflex controller monitors the dynamics of the robot relative to its environmental constraints and imposes velocity limits in regions where collisions are possible, overriding higher-level commands if necessary. In serving commands from higher levels, the local robot controller accepts GoTo and MoveTo commands as "soft attractor" inputs, which enables the robot to limit quasi-static contact forces during manipulation. With this low-level controller in place, the robot is tolerant of communications delays, drop-outs, and even garbled or grossly inappropriate commands. The robot in this context is incapable of hurting itself or its environment,

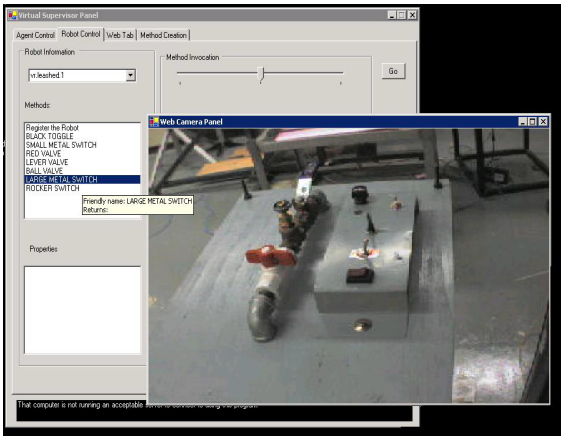

Figure 1. The Graphical User Interface of a Virtual Supervisor in the current prototype. The window on the right is a live video feed of the robot work environment. The window on the left allows an operator to give instructions to the robot by selecting a function and its parameters. Another tab ("Agent control") allows the operator to create and interconnect the agents.

regardless of disruptions or errors at higher levels in the control hierarchy.

V Upstream Communications: Appeals for Assistance

In addition to commands flowing down from higher levels of abstraction to lower-level VR's, it is also valuable for low-level VR's to be able to appeal to higher levels for assistance. As an experimental illustration, we constructed a robotic system for performing the task of sorting laundry [3]. This application constituted a relatively high level of autonomy. However, realizing a system that is fully competent as an autonomous creature is beyond near-term technology. The robot was capable of bouts of adequate performance, but it periodically became confused and incapable of progressing. This example was illustrative of the technical challenges of autonomy more generally. A fully-competent autonomous machine is still a distant goal. However, partial competence can be useful in the near term if the artificial creature allowed human assistance. In our example, human assistance was provided on demand by the robot. In situations where the robot's interpretation of the environment was not adequately competent, the robot would contact a remote human supervisor via network communications. The robot presented its view of the environment to the remote advisor (using a web-cam) and requested instructions from the advisor. These instructions could be either relatively high level (e.g., try again, move to named location, point camera at indicated target) or at a low level (e.g., enabling remote control via teleoperation in Cartesian or joint space).

With the ability to initiate appeals to higher levels, a partially competent system is able to achieve greater productivity. It can proceed autonomously when it is capable of interpreting the environment and performing its tasks within a constrained range of parameters. In instances when the lower-level controls are confused by novel situations or variations beyond the range of local competence, appeals to higher levels may enable the machine to overcome its confusion and resume autonomous operation.

VI Agent and Middleware

Perhaps the most important feature of IP networks is its pervasiveness, both on Earth and in its planned space deployment. In turn, the commonness of IP has led to a substantial development effort in the areas of communication support, software engineering for distributed environments, and middleware. The availability of this infrastructure leads to substantially reduced development times, increased software reuse,

monetary saving, reliability on an amply deployed infrastructure, and availability of advanced features off-the-shelf.

In our work, we have identified the following two methodologies to achieve objectives such as survivability and evolvability. In the first place, the use of off-the-shelf *look-up services* (such as Jini) allows units to find each other in a distributed environment even with no prior knowledge of their respective existence. Available look-up services also have powerful provisions for fault-tolerance that are especially critical in an unstructured environment. *Mobile agents* are essential to achieve our target general objectives. Agents are software components that are autonomous, adaptable, reactive, knowledgeable, goal-oriented, flexible, learning, mobile, collaborative, and persistent[4]. For example, mobility can enable a high-level notion of survivability as demonstrated by the following example. Suppose that UAV *x* is about to move out of sight of other units so that *x* will not be able to communicate with any other system for a certain period of time. Immediately before *x* loses connectivity, some of the software components that execute on *x* can move to another UAV *y*, i.e., they will start executing on the computational facilities available on *y*. Software mobility is particularly relevant if *x* was functioning as a "platoon leader", i.e., a UAV that coordinates the movement of a swarm of other similar units. In a static non-mobile environment, the "platoon" would lose a central point of control, whereas mobility enables the other UAV *y* to assume the role of a new command center. Mobility is also critical to achieve rapid reprogrammability: we have demonstrated that a new behavior can be created at a remote location by instantiating a new mobile VR, which then finds its way to the unit that is meant to control. In conclusion, we believe that software mobility is an extremely powerful notion that has allowed us to rapidly design and deploy large control software systems and achieve evolvability and survivability in these systems.

VII Conclusions and Future Work

In this paper, we have described a framework to enable communication with otherwise intelligent and autonomous machines. Our approach leverages on the pervasiveness of IP networks and on the substantial middleware that is available on this platform. It also critically depends on the careful design of the coordination between local and remote units.

Important open issues include the limit to software agent mobility, which is imposed by the low bandwidth and unreliability of UAV links and from the UAV spatial mobility. Finally, the on-board UAV network deserves more work.

References

[1] V. Liberatore. Scheduling of Network Access for Feedback-based Embedded Systems. *Quality of Service over Next-Generation Internet, SPIE ITCom* 2002, 73-82.

[2] A. Al-Hammouri, A. Covitch, M. Kose, D. Rosas, W. S. Newman, and V. Liberatore. Compliant Control and Software Agents for Internet Robotics. *Eighth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003).*

[3] M. L. Ngai, V. Liberatore, and W. S. Newman. An Experiment in Remote Robotics. *2002 International Conference on Robotics and Automation (ICRA 2002).*

[4] M. L. Griss and G. Pour. "Accelerating Development with Agent Components". Computer, **34** (5), 37-43, May 2001.