1. Introduction

This paper describes the on-board software developed under the Micro Robots for Scientific Applications 2 (MiRo2) activity commissioned by the European Space Agency (ESA). The activity designed and developed a Robotic Sampling System - a prototype of a tracked rover with a drilling system for deep (up to 2 meters) sampling on a planetary surface. Figure 1 illustrates the Mars rover and lander on Mars.

The system consists of the Mars lander and the rover. The rover carries the Drilling and Sampling Subsystem (DSS), which is able to penetrate and sample the Martian ground up to two meters deep. The rover and DSS are shown in Figure 2.
The DSS can be lifted and tilted. The drill-string is assembled from a drill bit and sections of drill-rod. Drill-rod sections are connected one by one as the drill is sunk, and disconnected as the drill is lifted. Drill bits and drill-rod sections are stored in carousels within the DSS. A tether cable connects the rover to the lander, which controls the rover and has a stereo video camera and image processing capabilities to map the terrain and plan rover movements. Both the lander and the rover have their own computers. In the prototype, the lander is simulated by a PC with an operator interface.

2. Goals and constraints

The focus of the MiRo2 activity was to develop a prototype and test the feasibility of the deep drilling rover concept. It was important to limit the cost of the prototype while ensuring that the prototype can be upgraded to flight quality.

Much of the on-board software development and testing had to be done before the actual rover computer and mechanics were available. The functions of the rover control program are quite complex:

- it obeys high-level commands from the lander to move the rover and run the DSS,
- it controls the rover and DSS for automatic drilling, mobility, and sample delivery, using several motors and mechanisms concurrently,
- it sends housekeeping, monitoring and reporting telemetry to the lander.

Moreover, we anticipated some difficulty in observing and testing the software in the actual rover, since the input/output interface is limited to the rover-lander tether, which carries a single bidirectional serial connection in addition to the rover electrical power.

2.1 Component selection

To keep the cost low, we chose commercial or free parts and tools, while keeping in mind eventual upgrading to flight quality.

We coded in the Ada language, to make the code portable to different processors and operating systems and also upgradeable to flight software. Low-cost or free Ada compilers are available. We used the GNU Ada compiler, Gnat.

We use a powerful, multi-user operating system rather than a small real-time kernel. This makes it easier to install, observe and test the control software. We chose the Linux
operating system, which is not only cheap but also makes it easy to write new device drivers.

For the rover computer, we chose a common PC104 stack, which includes a PC board with the Intel 486DX processor, an I/O board and a PWM board. This computer is powerful enough to run Linux, the rover control program, and several other programs that may be required for testing purposes.

2.2 Synergies

These software and hardware choices have interesting synergies:

- Since the Ada code is portable, we can equip the prototype with a powerful processor without impact on the software.
- The powerful processor lets us use a powerful operating system. Since concurrency is a standard feature of Ada, the choice of operating system has little impact on the Ada code.
- The powerful operating system allows far easier observation and testing of the embedded system than most real-time kernels would provide.

The rover computer normally boots a small Linux system from a Disk-On-Chip (DoC) solid-state device, which simulates a small hard disk. It also loads the rover software from the DoC. For development and testing purposes, we add a hard-disk board and connect a keyboard and a monitor, giving a full-featured Linux environment. The DoC is still visible as a second disk drive, making it easy to configure the DoC software and install new versions of the rover-control software.

Whether booted from the DoC or from a disk, the rover runs the PPP program to transmit IP traffic over the tether. Thus, the rover computer is connected to our LAN, and we can “telnet” into the rover to check its status, run test programs and so on. The TC and TM packets between the rover control program and the “lander” (operator’s PC) are transmitted over the tether as UDP packets, so the tether can be used at the same time for this normal TC/TM and for test information.

3. Rover Hardware

The mechanical and electrical structure of the rover is illustrated in the following Figure 3.
The rover control program controls all the mechanisms shown in the figure.

4. Software

The rover and DSS are controlled from a Control Unit PC that simulates the lander. The Control Unit provides a graphical user interface (GUI) for control and monitoring. The GUI and the rover control program allow manual control of all rover and DSS operations. A Testing Interface and a Rover Simulation program can be used to test the software without the rover hardware. Figure 4 illustrates the overall system.

4.1 Control Unit Software

The GUI is written in Java. It sends user commands to the Control Unit TC/TM program through a UDP socket. The Control Unit TC/TM program is written in Ada and reuses TC/TM code from the rover. It formats the command as a TC and sends it to the rover through the tether, again using UDP.

TM from the rover is received over the tether and passed through the Control Unit TC/TM software to the GUI for display.

4.2 Rover Software
Adequate autonomy of the rover-lander system is an essential task for future Mars missions. In-situ image processing for rover control must be included in the mission, since the time delay of one-way telecommunication between Earth and Mars varies from 3 to 22 minutes, depending on the relative positions of the planets. In the worst case, the ground receives the reply 45 minutes after sending a message. Navigating the rover from Earth would be very difficult and time-consuming, and therefore the lander must control the navigation autonomously.

The rover-lander tether is 40 meters in length, and the TC/TM bandwidth is limited (the prototype uses RS232 signals). Therefore, the rover must perform the lower level tasks autonomously, including the actual drilling and sampling.

The TC/TM system follows the PUS guidelines. In nominal operation, the lander uses high-level TC/TM to start and observe autonomous rover functions. There is also low-level TC/TM for direct control of all rover devices, i.e. it is possible to drive each motor independently.

In the rover control program, the TC/TM Communications module receives the TC. If the data packet header is valid, the command is forwarded to the TC Handler Task, which parses the TC and starts the related Control Tasks. The Control Tasks use the Device Interface to control the rover and DSS devices. The Device Interface consists of Linux device drivers and the normal “open”, “read” and “write” functions.

The Control Tasks have execution frequencies between 5 and 20 Hz. With the powerful processor on the prototype rover, the software can run at these frequencies under a normal Linux scheduler, and we did not use one of the “real-time” Linux systems.

The TC Handler Task and the Control Tasks send TM reports to the TC/TM Communications module, which creates the data packets and sends them over the tether to the Control Unit. The TM Handler Task periodically reads sensor data from the Device Interface and sends it as TM, for status monitoring.

### 4.4 Testing the software

Embedded software is often hard to test without its real hardware environment, because the software needs special software/hardware interfaces and also expects certain reactions from the hardware, without which the software will not run normally. The usual solution is to isolate the hardware/software interfaces in a software module, the low-level driver software. This module exists in two versions, a “real” version and a “test” version, where the test version simulates the hardware to provide near-normal reactions.

For the MiRo2 rover, this approach was simplified by the use of Linux: all rover devices are accessed as Linux devices, using the normal “read”, “write” and “ioctl” functions. The low-level drivers are therefore not linked into the rover control program; they are dynamically loaded into the running Linux system before the rover control program is started.

The low-level drivers are written in the C language, since the Linux driver-development support relies on a number of C header files. As in the normal solution, the low-level drivers exist in two versions, a “real” version that operates the interface boards over the PC104 bus, and a “test” version that does not depend on the presence of the interface boards or of the rover hardware.
The test-drivers are “two-faced” in the sense that each driver implements two Linux devices: a normal device and a test device. The rover control program uses the normal device to send a command to the rover hardware or to read status data from the hardware. However, instead of sending the command to the actual hardware, the test-driver stores the command in an internal command buffer. This buffer can be read via the test device. Analogously, when the rover control program reads data from the normal device, the test-driver actually provides this data from an internal data buffer, which can be written via the test device. Figure 5 may clarify this.

![Figure 5. A two-faced test driver.](image)

When the rover control program is tested without the real rover hardware, it runs concurrently with a rover simulation program. The rover simulation program reads the test devices to fetch the hardware commands (sent by the control program), updates the simulated rover state accordingly, and writes the new status data to the test devices (for the control program to read).

The rover simulation program is also written in Ada and can run on the same Linux machine as the control program, or on a different machine. If a different machine is used, a “stub” simulation program is run on the same machine as the control program. The “stub” accesses the test devices and uses UDP to transfer the data and commands, over the network, to the real simulation program running elsewhere.

### 5. Summary

Developing and testing the MiRo2 on-board rover control program was greatly helped by running a multi-user operating system (Linux) in the rover and using multiplexing communication protocols (IP, UDP, TCP) on the rover-lander tether. For this, the prototype rover had to have a powerful computer. By writing the control program in Ada, a language in which all the required functions can be written portably, we could equip the prototype rover with a powerful and cheap COTS computer without distorting the control program and without making it harder to upgrade to a flight version.

For testing the rover control program without the real hardware of the rover and the drilling and sampling system, we used “two-faced” Linux I/O drivers to interface the rover control program with a rover simulation program.

The MiRo2 software development was successful. The principal problems were the limited bandwidth and sensitivity to EM noise of the RS232 line in the tether. For some testing phases, we used the Enhanced Parallel Port on the rover computer as a higher-speed IP
channel, by means of the PLIP program. This, too, demonstrates the flexibility of our approach.

Acknowledgements

Many people and organisations contributed time, knowledge, skill, and support to this project, and we are pleased to acknowledge their contributions:

European Space Agency (ESA):
http://www.estec.esa.nl/
Gianfranco Visentin, ESTEC, project manager.
European Space Agency (ESA) funded and coordinated the activity "Micro Robots for Scientific Applications 2".

Space Systems Finland (SSF):
http://www.ssf.fi/
Sami Laitinen, Mika Zelikman, Jukka Huvinen.
SSF worked as a prime contractor in the MiRo2 project and developed the software.

Technical Research Centre of Finland (VTT):
http://www.vtt.fi/aut/
Tomi Ylikorpi, Petri Kaarmila.
VTT was responsible for the development of the Drilling and Sampling Subsystem.

Helsinki University of Technology (HUT):
http://www.automation.hut.fi/
Prof. Aarne Halme, Jussi Suomela, Jari Saarinen, Teemu Levomäki.
HUT Automation Laboratory developed the roving system and they were assisted by the Rover Company Ltd. (RCL), http://www.private.peterlink.ru/rcl/.

For additional information about the MiRo2 project:
http://www.ssf.fi/miro/