Software Architecture and Design of the Kontur-2 Mission

Martin Stelzer; Peter Birkenkampf; Bernhard Brunner; Bernhard-Michael Steinmetz; Jörg Vogel; Stefan Kühne

Software Architecture and Design of the Kontur-2 Mission

Martin Stelzer
German Aerospace Center (DLR)
Robotics and Mechatronics Center
Weßling, Germany
+49 8153 28 2153
Martin.Stelzer@dlr.de

Peter Birkenkampf
German Aerospace Center (DLR)
Robotics and Mechatronics Center
Wessling, Germany
+49 8153 28 1879
Peter.Birkenkampf@dlr.de

Bernhard Brunner
German Aerospace Center (DLR)
Robotics and Mechatronics Center
Wessling, Germany
+49 8153 28 1791
Bernhard.Brunner@dlr.de

Bernhard-Michael Steinmetz
German Aerospace Center (DLR)
Robotics and Mechatronics Center
Wessling, Germany
+49 8153 28 2488
Michael.Steinmetz@dlr.de

Jörg Vogel
German Aerospace Center (DLR)
Robotics and Mechatronics Center
Wessling, Germany
+49 8153 28 1049
Jörg.Vogel@dlr.de

Stefan Kühne
German Aerospace Center (DLR)
Robotics and Mechatronics Center
Wessling, Germany
+49 8153 28 2400
Stefan.Kuehne@dlr.de

Abstract—**This paper describes the software architecture and design of the space segment, communication and ground segment software of the Kontur-2 project, which contributed to the realization of telepresent planetary exploration. The main research objectives in Kontur-2 were the development and in-flight verification of a space qualified two degree of freedom (DoF) force-feedback joystick (RJo) inside the Zvezda Service Module of the International Space Station (ISS), the implementation of telepresence technologies and the study of human performance when controlling a force feedback joystick in microgravity. The project was conducted from 2012 to 2015 by a consortium consisting of the German Aerospace Center (DLR), the Russian Federal Space Agency (ROSCOSMOS), The Russian State Scientific Center for Robotics and Technical Cybernetics (RTC), S. P. Korolev Rocket and Space Corporation Energia (RSC "Energia") and the Yuri A. Gagarin State Scientific Research-and-Testing Cosmonaut Training Center (GCTC). The DLR conducted two sets of experiments in which a cosmonaut on ISS used RJo to perform different tasks with robots located on-ground. The first set was conducted with a two DoF robot equipped with a camera system, a task board and torque sensors that allowed the cosmonaut to perceive forces caused by contacts with the environment. For the second set of experiments we used a humanoid robot to perform a tele-handshake and a cooperative task between the cosmonaut on ISS and colleagues at RTC in St. Petersburg.**

To realize these experiments, the consortium developed on-board and on-ground software which will be described in this paper. The space segment software consists of the control software for RJo and user interfaces on a laptop to guide the cosmonaut efficiently through the experiments. We designed a state machine for these user interfaces to capture state changes during the experiment execution. This way we provided only relevant contextual information to the cosmonaut. On RJo, we deployed a component framework combining a data-centric architecture with a CCSDS Space Packet interface. Additionally, we designed the communication software for supporting a direct multi-channel connection between ground control and ISS using our own S-band radio equipment. During contact to ISS, the ground operators used the ground segment software at DLR for experiment support, supervision, maintenance and data logging. The visual feedback from the camera system required by the cosmonaut to perform the experiments was provided by a low-latency video stream through a communication channel with very restricted bandwidth.

During 23 experiment sessions in 2015, the Kontur-2 software formed the basis of the successful completion of the experiments. Their results contributed to the fields of telepresence technologies and human factors.

978-1-5090-1613-6/17/31.00 ©2017 IEEE

TABLE OF CONTENTS

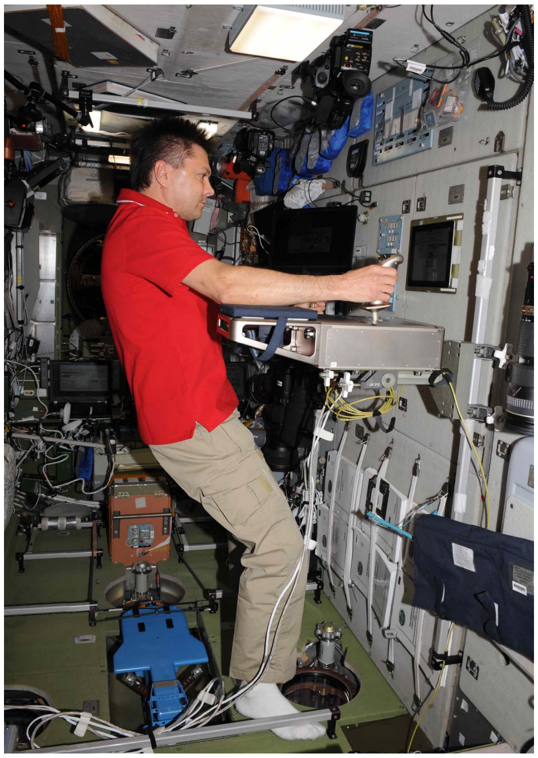


Figure 1. **Cosmonaut O. Kononenko with RJo on-board the ISS (Source: ROSCOSMOS)**

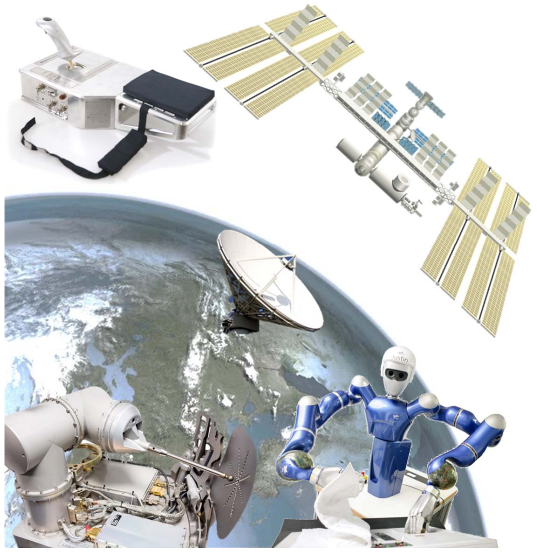


Figure 2. **KONTUR-2 scenario as supported by DLR**

1. Introduction

For decades humans have been exploring our solar system not only to satisfy their curiosity, but also to gather knowledge about where we live now and where we may live in future. To coordinate these efforts in space exploration between space agencies around the world, the International Space Exploration Coordination Group (ISECG) has set up the Global Exploration Roadmap (GER) in 2011 and updated it in 2013 [1]. The GER recognizes that human robot partnership has potential to increase benefits in the exploration of moon or mars. It also mentions the concept of telepresence [2]. In this concept an astronaut situated e.g. in a spacecraft orbiting a planet or moon teleoperates a robot on the surface with low latency. Tasks that require fast decision making or high dexterity could benefit from this concept in terms of increased scientific return or an increased number of achievable scientific objectives. However, two main challenges arise during teleoperation of a robot from an orbiting spacecraft. First, despite the time delay a stable and realistic haptic feedback is required [3]. Second, the operators‘ control precision degrades in microgravity [4].

The GER also encourages the space agencies to utilize the International Space Station (ISS) for research activities and technology demonstrations. In alignment with this roadmap a consortium consisting of the German Aerospace Center (DLR), the Russian Federal Space Agency (ROSCOSMOS), The Russian State Scientific Center for Robotics and Technical Cybernetics (RTC), S. P. Korolev Rocket and Space Corporation Energia (RSC ”Energia”) and the Yuri A. Gagarin State Scientific Research-and-Testing Cosmonaut Training Center (GCTC) conducted the Kontur-2 project from 2012 to 2015. DLR’s main research objectives in the Kontur-2 project were the development and in-flight verification of a space qualified two degree of freedom (DoF) force-feedback joystick (RJo) inside the Zvezda Service Module of the ISS, the implementation of telepresence technologies that can cope with time delay and the study of human performance when controlling a force feedback joystick in microgravity [5]. For this, DLR and RTC implemented two sets of experiments in which a cosmonaut on ISS (see Figure 1) used RJo to perform different tasks with remotely controlled robots located on-ground at DLR (cf. Figure 2).

The first set of experiments was conducted with a two DoF robot equipped with a camera system, a task board and torque sensors that allowed the cosmonaut to perceive forces caused by contacts with the environment.
For the second set of experiments we used DLR’s humanoid robot to perform interactive tasks between the cosmonaut on ISS and a coworker on-ground: A tele-handshake (DLR) and a cooperative object manipulation (DLR and RTC).

RTC in St. Petersburg performed similar experiments with their own robots on-ground: A stationary, kinematical redundant snake-like robot surrounded by its own workspace and a small wheeled mobile robot which was used for remotely controlled exploration of a different workspace [6]. However, as this paper focuses on DLR’s contribution to Kontur-2, a more detailed explanation of RTC’s scientific goals, their own experiments and their software architecture is beyond the scope of this paper.

In general the Kontur-2 setup relies on equipment of its precursor missions ROKVISS (Robot Component Verification on ISS) [7] and Kontur [8], such as the whole radio-based communication infrastructure and the small two DoF robot. The main goals of the ROKVISS experiment were the verification of light-weight robotic components under realistic mission conditions in free space, as well as the verification of direct telemanipulation of this robot using a force feedback joystick as haptic input device on-ground. In the Kontur project the teleoperation of the robot on ISS was conducted from St. Petersburg via public Internet. In simple terms, the ROKVISS and Kontur setups were mirrored in Kontur-2. Of course the reuse had a great potential of saving costs. However, it also imposed challenging boundary conditions that will be discussed later.

The architecture and design of the on-board and on-ground software developed to realize the Kontur-2 experiments will be described in this paper. To the knowledge of the authors, this is this first time the software architecture of such a telerobotic system is described in detail. The outline is as follows: Work related to this paper is sketched in Section 2. Section 3 details the requirements for the Kontur-2 software. After that, a brief overview of the mission setup is given in Section 4. The static architecture of the software is detailed in Section 5 for the space segment, in Section 6 for the communication part and in Section 7 for the ground segment. Section 8 gives an introduction to the mission operations of Kontur-2 to emphasize specific aspects of the software’s dynamic architecture. Finally, Section 9 concludes the paper with a brief description of the mission results and an outlook to upcoming missions related to Kontur-2.

2. Related work

Besides the Kontur-2 consortium the European Space Agency (ESA) is another stakeholder for telerobotics on space. The aim of their *Multi-purpose End-To-End Robotic Operations Network* (METERON) is the control of robotic systems from a planetary orbiter with different human-robot interfaces [9]. Within the frame of METERON, the first haptic master device with one DoF was launched to ISS in 2014 during the Haptics-1 mission [10]. Three astronauts performed force-feedback and human perceptual motor performance tests with this device to study the influences of microgravity on psychomotor performance metrics. The same haptic input device was used in Haptics-2 to teleoperate a one DoF slave device on-ground [11]. The communication was performed via

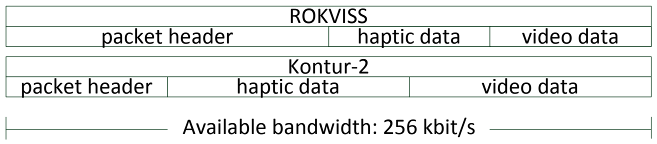


Figure 3. Bandwidth allocation in ROKVISS and Kontur-2

geostationary relay satellites resulting in a round trip time of about 0.8 seconds. In INTERACT the slave device on-ground was replaced by the Interact Centaur robot to perform a sequence of remotely operated tasks [12]. The Supvis-Justin Experiment is currently slated for 2017 and will use high level commands entered via a user interface on a tablet PC instead of haptic feedback to control a service robot on-ground [13]. The aim of this experiment is to investigate the viability of supervised autonomy in telerobotic tasks.

3. SOFTWARE REQUIREMENTS

This section focuses on the software requirements of the Kontur-2 mission since requirements regarding the electrical and mechanical parts of RJo have already been introduced by Riecke et al. [5].

To conduct the experiments according to the scientific program we had to implement two different kinds of setups (with two different robots), which will be described in detail in Section 4. Furthermore, a video stream to ISS was required to give the cosmonaut an adequate overview of the task board. During the mission we even had to supplement this video stream by a bidirectional audio and video connection to enable interactive sessions with the cosmonaut. Besides that, a LED board had to be built and controlled according to the task sequence. Finally, several user interfaces were required that are able to guide the cosmonaut through the experiments.

In practice, the provision of bandwidth for data transmission is strongly limited not only by technical reasons but also by the requirement to reuse already installed communication resources whenever possible. It was already mentioned in the introduction that the Kontur-2 project relies on the communication infrastructure of ROKVISS [7]. Therefore, the data transfer concept of the project Kontur-2 was designed for exclusive reuse of this already existing Russian / German owned serial S-Band link between the Russian Zvezda Service Module at ISS and DLR's ground control at Oberpfaffenhofen, Germany (via DLR's tracking station at Weilheim, Germany). This link supports some substantial features:

• It has a strong asymmetric data transmission characteristic (256 kbit/s uplink rate to ISS and 4 Mbit/s downlink rate from ISS). The reason for this asymmetric bandwidth allocation was that in ROKVISS a stereo video had to be transmitted from the space segment to the ground segment only. This allocation could not be changed without deploying new communication hardware on ISS.
• In ROKVISS the data packages used a standard protocol defined by the *Consultative Committee for Space Data Systems* (CCSDS) featuring a bandwidth for application data of 128 kbit/s in the uplink and 3.8 Mbit/s in the downlink. As per definition the haptic data for telepresence would always consume 50% of the uplink bandwidth for the application data, we did not have confidence in the feasibility of transferring a video in sufficient quality with the remaining 64 kbit/s. During the project Kontur-2 we were able to reduce the CCSDS protocol overhead in the uplink which lead to a bandwidth of 192 kbit/s for application data that is equivalent to a 50% increase of application data bandwidth in the uplink (cf. Figure 3). Nevertheless, it was still challenging to implement a smooth video stream with only 96 kbit/s available. Section 7 will illustrate our solution.
• The internal bidirectional transfer rate is 1.0 kHz resulting in a data package size of 32 bytes including protocol information and application data.
• An internal three-level concept supports a prioritized data transfer over the serial radio link.
• Multiple virtual data channels are available for quasi simultaneous data transfer via clocked time-multiplex of the underlying serial radio link.

In principle, radio contact is possible for four consecutive ISS orbits per day, but each contact is affected - to a greater or lesser extent - by different overlapping of the transmitter lobe, shadowing effects or specular / diffuse reflections of the radio signal at protruding ISS components. An unfavorable flight attitude of ISS intensifies or alleviates those problems. Typically such effects deteriorate the quality of the received signal and can cause an undesirable temporary loss of signal (LOS) in the worst case at any time. For that reason all communication links had to be equipped with a fast but unbuffered auto-resume capability.
Moreover, two interface computers called *On-Ground Computer* (OGC) and *On-Board Computer* (OBC) were an essential part of the ROKVISS infrastructure. They were utilized to encode and decode the application data for transmission via the radio link. Their further use for the Kontur-2 infrastructure was mandatory. But the software running on these computers had to be adjusted for the Kontur-2 mission for two reasons:

• providing external interfaces to the three internal priority levels of the serial communication channel, i.e. real-time data for telepresence, telemetry/telecommand data, and low priority data for video streaming and maintenance.
• increasing the internal bandwidth for application data by using a minimized CCSDS protocol overhead.

The inevitable remote exchange of essential communication software on a multiple CPU hardware already in space is really an extremely mission critical process which must not cause any dead-lock by incompatible software versions even in case of update faults. This requirement has an dramatic impact on the design of the new software and the upload procedure.

An additional requirement regarding fault tolerance was to have more than one storage for RJo's software to tolerate hardware faults and allow for a fail-safe software update mechanism.

Finally, due to the scientific character of the Kontur-2 experiment, it was a mandatory requirement to record all the available experimental data.

As a matter of fact such practical limitations are quite challenging for proving new concepts (e.g. for control) under realistic constraints. Nevertheless, the resulting experimental setup relies on proven technology as it could be used for real applications later on.

4. SYSTEM SETUP

Before we explain how these requirements were implemented, this section gives a more detailed overview of the Kontur-2 system setup.

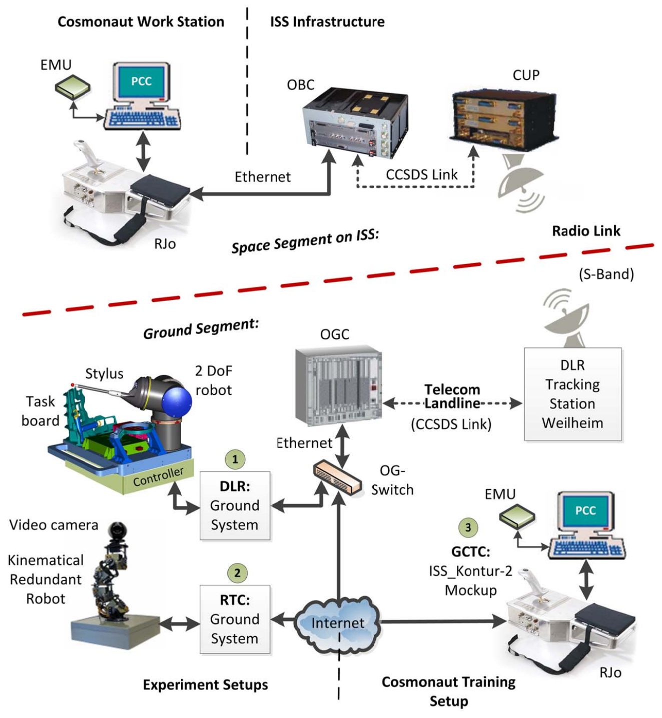


Figure 4. Kontur-2 system setup

Space segment overview

The upper part in Figure 4 symbolizes the hardware on ISS comprising the following elements:

- External Memory Unit (EMU)
- Portable Control Computer (PCC)
- Force-feedback joystick (RJo)
- On-Board Computer (OBC)
- Communication Unit for Payloads (CUP)

These elements can be grouped into two primary systems, i.e. the cosmonaut's workstation and the communication gateway. The workstation consists of the PCC for user I/O (including video and audio), the EMU which contains the operating system and user interface software, and RJo. DLR designed and built RJo as a haptic input device for the cosmonaut. It contains an ARM Cortex A8 micro controller with a microSD card interface, an on-board NAND-Flash, two EEPROMs and two motor modules with EtherCAT interfaces. The exclusive communication gateway on-board is a reused part of the ISS infrastructure and consists of the OBC for data handling and the CUP for real-time radio transmission. It allows an exclusive real-time data exchange with its counterpart on-ground in basically mirrored configuration.

Ground segment overview

The lower part in Figure 4 shows the mirrored configuration of the communication gateway, which mostly consists of a tracking station, the *On-Ground Computer* (OGC), and the various setups that have been established for Kontur-2 on-ground, i.e. the experiment setups of DLR and RTC, as well as the cosmonaut training setup at GCTC.

Experiment setup of DLR—The setup focuses on standardized experiments in order to investigate human hand-eye coordination, skillfulness and force-feedback in mircogravity. For that a small fix mounted two DoF robot (ROKVISS robot) is used with its specific task board which offers an environment for experiments with and without mechanical contact. A stylus on top of the robot's last link serves as pointing device for experiments without environment contact (light spot aiming, moving light spot pursuit tracking) or as touch device for experiments with environment contact (contour following, line tracing, spring pulling, free telepresence) [4]. The robot itself is equipped with a stereo camera system and torque sensors that allow the cosmonaut to sense the relative motion and the contact forces between stylus and environment. An additional fixed camera aligned with the task board allows a global view for aiming and pursuit tracking tasks. Throughout the paper, we will refer to this setup as the *task board setup*. Within this setup, the cosmonaut had to perform free motion tasks without physical interaction in order to investigate positional accuracy, as well as contact tasks in order to explore force regulation. In principle, the sequence of a task was as follows: An operator on-ground selected the desired task to perform, as soon as both uplink and downlink to the space station had been established. Upon this, the task and its description was displayed at the cosmonaut's user interface, the robot moved to the corresponding position of the task board and an appropriate camera was selected (robot or static camera) to provide the video stream. Next, the cosmonaut had to move the joystick handle to a predefined, task-specific starting position to create reproducible conditions over all the sessions of this task. Once this position was reached and held for a few seconds the cosmonaut had to press the deadman button. This enabled the telepresent control of the robot, so that the cosmonaut could conduct the task. The procedure was repeated until the signal to the space station was lost.

In extension to the previous experiment scenario DLR's humanoid robot *SpaceJustin*, which is a modified version of DLR's humanoid two-arm system Justin [14], was remotely controlled by the cosmonaut in order to interact with persons or objects on-ground in place of himself. In order to enhance the immersion, those activities were supported by a bidirectional live audio and video transmission. For those tasks the right arm of SpaceJustin was linked to the RJo on ISS and the left arm to an RJo at RTC in St. Petersburg. This setup will be referred to as *SpaceJustin setup* in the remainder. During both tasks the cosmonaut used SpaceJustin as a robotic avatar for shaking hands with an operator on-ground and for grasping and moving of a ball in cooperation with colleagues at RTC.

Experiment setup of RTC—In this setup RTC uses its own kinematical redundant snake-like robot irremovable placed inside a structured workspace. It is also used to study human skillfulness and sensitivity to force-feedback. Alternatively a small vehicle is available for a remotely controlled exploration of its own workspace.

Cosmonaut training setup at GCTC—The third application on-ground labeled with GCTC in Figure 4 emulates the Kontur-2 workstation of a cosmonaut on ISS and allows gaining experience with all control elements in preparation of the real mission. This setup is used for the cosmonaut training which, therefore, can be performed completely without the space segment.

Communication overview

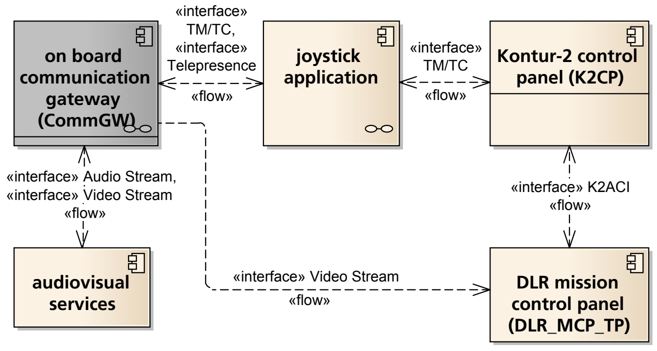
The overall data exchange inside both the space and the ground segment is based on Ethernet with one exception: The communication gateways between space segment and ground segment use a different protocol defined by the CCSDS. However, the CCSDS usage is limited to a very low level subset of the standard in order to reduce the protocol overhead and its service complexity to an absolute minimum. Ethernet packages from the ground and/or the space segment tunnel the radio link via CCSDS data containers as needed. Specific protocol requirements concerning integrity and reliability of the data transfer have to be considered at the Ethernet level.

An intentional add-on of this communication link is the spatial separation on-ground between DLR's radio tracking station in Weilheim and the OGC at DLR in Oberpfaffenhofen. This spatial distance is bridged by an additional tethered landline which in practice allows a wider range of applications for remote control of machinery or robot systems (e.g. in underground) than a pure radio link approach. In the context of Kontur-2, this dedicated landline acts as a simple extension of the "pure radio link" via cable for the transmitted data frames. This direct point to point connection ensures that as little artificial signal delay as possible will be added to the data transmission. The generation, processing and coding of data frames for radio transmission is the exclusive task of OGC and OBC. From a networking point of view both space and ground segment are pooled together to a common local area network ignoring all their spatial distances. For that reason the encapsulated radio link between space and ground segment can be easily replaced by a cable or by a fully functional radio link simulation.
Due to the low ISS orbit, any radio contact is limited to a maximum of 10.0 minutes whereof maximal 8.0 minutes can be used for the experiment supposing best conditions. The difference of approximately 2.0 minutes is owed to an 'Acquisition of Signal' (AOS) phase whose expenditure of time depends on the quality of the predicted ISS overpass and the used presettings of the radio antenna on-ground derived from the ISS path prediction. This AOS procedure requires at least one minute in order to conduct a frequency adjustment for compensating Doppler effects under manual control and an auto-negotiation between OGC and OBC in order to initiate the monitored cyclic data exchange. At the expected end of radio contact (referred to as 'Loss of Signal' - LOS) the radio signal simply fades away. This phase is affected by increasing drop outs of the signal and takes also round about one minute.

In general all applications on-ground are connected with the space segment via a common communication gateway. But while the ground system at DLR is directly connected to the communication gateway due to its close vicinity to the radio transmitter, the connection to the ground system at RTC requires in addition the public Internet in order to bridge the enormous spatial distance between Oberpfaffenhofen in Germany, and St. Petersburg in Russia. The Kontur-2 space segment mockup at GCTC is connected to the ground systems of RTC and DLR via public Internet, too. However, the major drawback of using a public Internet connection or something equivalent is the artificial but inevitable reduction of determinism, the disproportional magnification and uncontrollable fluctuation of the signal propagation time by which the total system performance is directly affected. It is an acceptable choice for rapid prototyping but does not meet long-term demands of high fidelity control applications. The evident objective of a data distribution concept has to be the elimination or minimization of any artificial source of degradation within a control system. But such an intensive customization is beyond the capabilities of pure public Internet.

Summary

In conclusion, the Kontur-2 setup is a distributed telerobotic system with space and ground segment interconnected by a radio link that was prone to communication blackouts and that featured a maximum of ten minutes of visibility.



on board communication gateway (CommGW)
«interface» TM/TC, «interface» Telepresence «flow»
joystick application
«interface» TM/TC «flow»
Kontur-2 control panel (K2CP)
«interface» Audio Stream, «interface» Video Stream «flow»
«interface» K2ACI «flow»
audiovisual services
«interface» Video Stream «flow»
DLR mission control panel (DLR_MCP_TP)

Figure 5. Space segment software overview

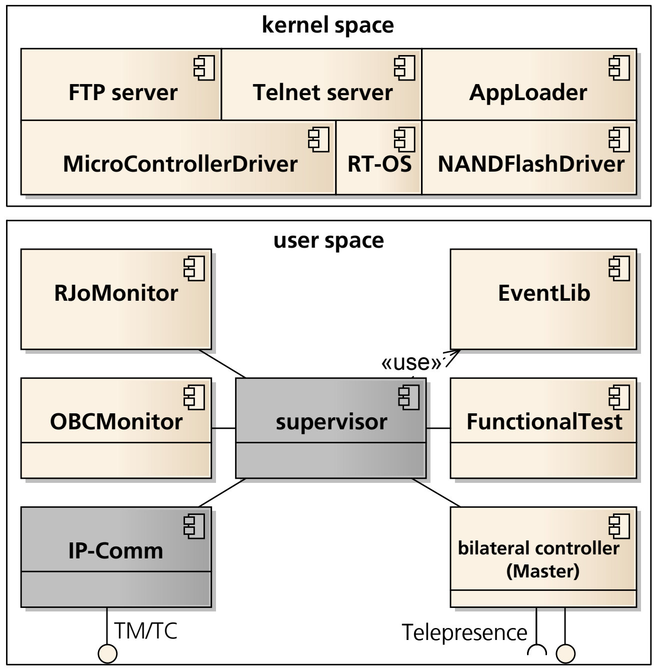


Figure 6. Software components of the joystick application

5. Space segment software

The static view of the components that are part of the space segment software is depicted in Figure 5. The *on-board communication gateway* (CommGW) is used for the communication with the ground segment. As it is part of the communication software it is shown in the figure only to get a better understanding of the overall software structure. Details about it will be given in Section 6. The *joystick application* is a generic term for all the software components that are deployed on RJo to implement its functionality. These components can be monitored and controlled via their telemetry/telecommand (TM/TC) interface either from ground or from the *Kontur-2 control panel* (K2CP). This panel provides an interface for the cosmonaut to RJo in order to trigger basic actions such as calibration, functional testing or starting the actual experiment. The *DLR mission control panel* (DLR_MCP_TP) implements an user interface to guide the cosmonaut through DLR's scientific experiments. It is attached to the K2CP via the proprietary *Kontur-2 application control interface* (K2ACI) for communication with RJo and the ground segment. Additionally, it displays the video stream received from ground to show the task environment. For the

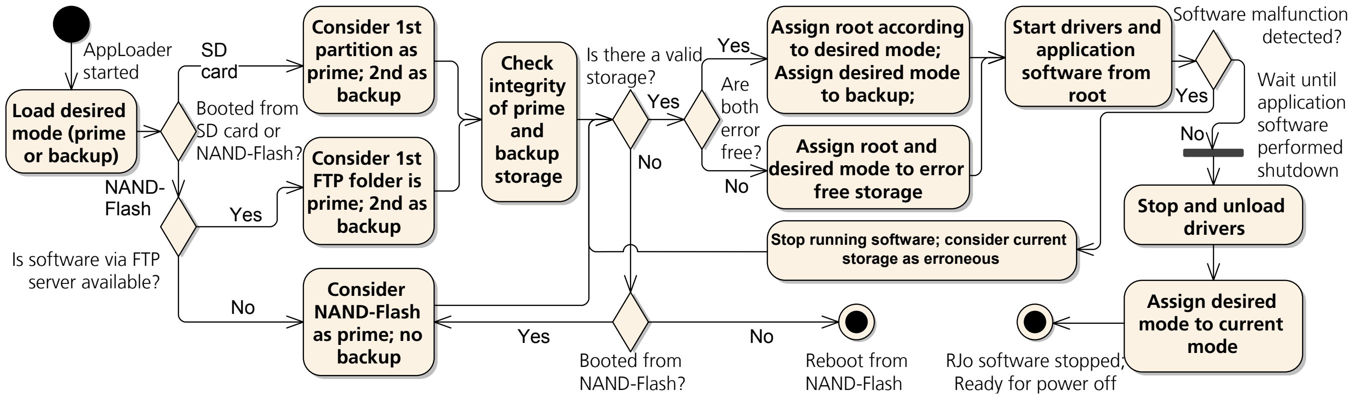

AppLoader started
Load desired mode (prime or backup)
Booted from SD card or NAND-Flash?
SD card
NAND-Flash
Consider 1st partition as prime; 2nd as backup
Is software via FTP server available?
Yes
Consider 1st FTP folder is prime; 2nd as backup
No
Consider NAND-Flash as prime; no backup
Check integrity of prime and backup storage
Is there a valid storage?
Yes
No
Are both error free?
Yes
Assign root according to desired mode; Assign desired mode to backup;
No
Assign root and desired mode to error free storage
Start drivers and application software from root
Software malfunction detected?
Yes
Stop running software; consider current storage as erroneous
No
Wait until application software performed shutdown
Stop and unload drivers
Assign desired mode to current mode
RJo software stopped; Ready for power off
Booted from NAND-Flash?
Yes
No
Reboot from NAND-Flash

Figure 7. **Activities of the AppLoader**

experiments in the SpaceJustin setup we used the *audiovisual (AV) services* to have bidirectional audio and video streams for an improved interaction with the cosmonaut compared to the chat interface provided by K2CP. In the following, these components mentioned before will be further detailed.

Joystick application

Figure 6 shows the components that belong to the joystick application. We chose a real-time operating system (OS) with priority based scheduling to enable a deterministic behavior of the control algorithms. This OS segregates its virtual memory in a kernel space for system services and drivers, and a user space for the application software. Among the system services the File Transfer Protocol (FTP) and the Telnet server can be used to upload software updates. Moreover, drivers for the micro controller board, its attached on-board flash and a fail-safe application loader (AppLoader) are executed in kernel space.

Bootloader—As RJo is exposed to radiation on ISS there is a small chance of hardware malfunction or memory corruption. But as there has been no requirement to deal with this thread, the hardware was designed in a non-redundant way for cost reasons. Nevertheless, we tried to implement different stages of software redundancy to maintain the functionality of RJo in case of malfunctions. When RJo is switched on, the 1st stage bootloader of RJo's ARM CPU is trying to fetch the 2nd stage bootloader from one of the configured booting devices. On RJo we are using microSD card and on-board flash as booting devices. The microSD card is searched first. We enhanced the 2nd stage bootloader so that it is able to verify the integrity of the OS kernel using the cyclic redundancy check (CRC) and stores the checksums in EEPROM. Hence, if the bootloader on the microSD card detects a checksum mismatch it will reboot the system from the on-board flash. This way, RJo is able to boot the OS even in case of loss or corruption of one of its booting devices. The only remaining problem is the case of a corrupted 1st or 2nd stage bootloader. In this case corrupted code would probably be executed and that in turn results in undefined behavior. However, we were not able to implement any countermeasures on this level without additional hardware so we accepted the risk of loosing the system in this unlikely case.

AppLoader—Once the OS is loaded by the procedure mentioned earlier, it immediately runs the *AppLoader*. The idea behind this loader is to prolong the validation concept of the 2nd stage bootloader to the phase of starting drivers and application software. Altogether, we provide three different storage locations for the software: A microSD card is our main memory we planned to work with. It has two partitions where each contains a full copy of all software artifacts such as binaries, libraries and configuration files. In case the hardware of the microSD card gets corrupted we still want to have the possibility for software updates. For this reason, an FTP server is running on PCC which provides two more copies of those files. Finally, the on-board flash has a single read-only copy as last resort.

The AppLoader implements the concept shown in Figure 7 to integrate all of these storage locations. Upon start, the AppLoader reads the desired mode from the EEPROM. This mode defines which software copy to choose if there are more than one available at the current storage location. The modes are named *prime* and *backup*. The motivation behind these two modes is to have a working software copy even when a software update fails. Thus, we perform such updates on the prime location first. If the update is successful, this software is used the next time RJo is switched on and the backup location can be updated some time later. If not, the backup location is used for the next experiment, and the operators on-ground are be able to fix or undo the software update.

The storage location to be used for starting the software is selected based on the booting device that has been used initially to load the OS kernel. In case of microSD card, the first partition is considered as the prime storage and the second partition is regarded as the backup storage. If the OS was loaded from on-board flash, the AppLoader will check if the software is available via FTP. If so, one FTP folder is considered as prime, the other folder as backup. If not, the software located in on-board flash is treated as prime without a backup being present.

Once the AppLoader has determined the booting device, all software copies on it are verified by means of the Simple File Verification (SFV). For this purpose a file contains CRC checksums for all relevant files. The AppLoader verifies all of these checksums and marks the storage location as invalid if a mismatch is found. This procedure is skipped for the on-board flash, because the artifacts have been linked to the kernel image for this type of storage. Hence, the 2nd stage bootloader already performed the verification of kernel and software during the boot process.

After the verification the AppLoader calculates the number of valid storage locations. If all are marked invalid and the system has started from microSD card, it will be rebooted from on-board flash. In case both FTP folders are invalid, the read-only copy that resides in the on-board flash will be used. As next step the AppLoader selects the actual storage location (denoted as *root* in the figure) the software will be started from. If both the prime and backup location are available,

the location is selected according to the desired mode loaded at the beginning and the remaining location is set to the new desired mode in EEPROM. Hence, in case a system reboot is triggered for some reason, e.g. a software failure, before the software has been stopped by user's command, the alternative storage location is used after the reboot hopefully avoiding the same situation. If there is only one storage available, this one is selected as the storage location to use and as the new desired mode.
Next, drivers and application software are started. If the AppLoader is able to detect a software malfunction during software start up, it will try to shut down the software, mark the used software copy as erroneous and restart the selection of a storage location. Otherwise, it is waiting for a graceful shut down of the application software from user side to subsequently stop the drivers. Finally, the new desired mode in EEPROM is reset to the mode currently in use.

NANDFlashDriver—The NANDFlashDriver implements a background task that cyclically repeats the verification of the on-board flash and provides the results to the application software for monitoring purposes.

MicroControllerDriver—Access to the hardware peripherals such as the motor controllers, LEDs, buttons and temperature sensors is provided by the MicroControllerDriver. Furthermore, it is also responsible for the limitation of the force at the joystick handle, supervises the temperature sensors and implements a deadman button functionality. Lastly, the MicroControllerDriver facilitates an algorithm to calibrate the handle's position after power-on, a position controller to move the handle autonomously and an alive check for its routines.

FTP server—The FTP server provides access to both partitions of RJo's microSD card so we are able to upload new versions of the software. Additionally, log files created by RJo can be retrieved via this service.

Telnet server—Usually, the software is uploaded only to the prime partition because of the limited bandwidth in the uplink. If the updated software is working without problems, we establish a Telnet session to copy the new content from the prime partition to the backup partition so both partitions are in sync again.

User space software—We use the component framework HIROSCO (HIgh-level RObotic Spacecraft COntroller) [15] to implement the application software running in user space. This framework has a service-oriented architecture, which means it provides services specified in the Packet Utilization Standard (PUS) of the European Cooperation for Space Standardization (ECSS) for the interaction of components implemented with it. In order not to burden the component developer with details of the PUS, HIROSCO promotes a data-centric approach. Data of a component just needs to be registered in a so-called dictionary. Using an XML-file the framework can be configured to provide this data to other components or to ground via the PUS interface, e.g. for housekeeping. Besides the framework, HIROSCO provides a component named *supervisor* that manages the components attached to it and their interaction. For example, it is responsible to start and stop them, to monitor their real-time behavior and to react on events. Mission specific events are handled by a library dynamically loaded at startup (denoted as *EventLib* in Figure 6). These events were required mainly to implement a temperature control system so that RJo does not overheat and to shutdown the force-feedback in case of internal anomalies. Finally, the HIROSCO framework comes with a component that runs a TCP/IP server to which external clients such as user interfaces can connect for monitoring and control.

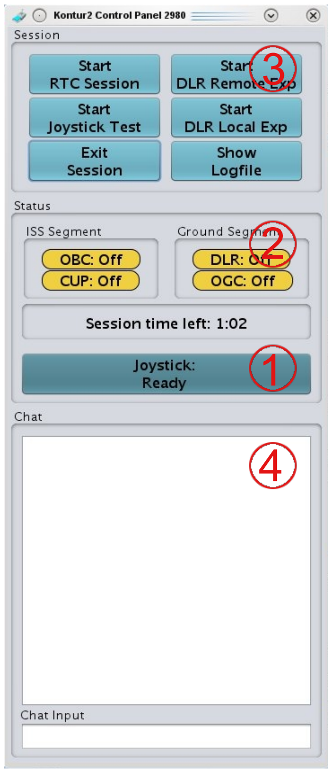


Figure 8. K2CP cosmonaut user interface

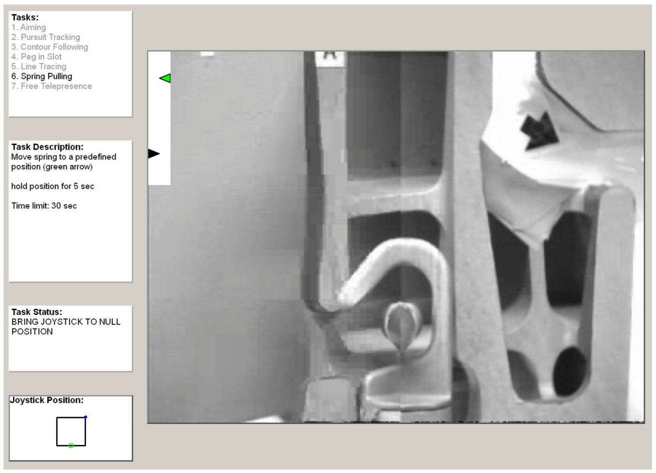
The application software itself consists of several components that are described next. The *RJoMonitor* collects the monitoring information provided by the drivers mentioned before and generates events in case of off-nominal situations. Additionally, it provides this data as housekeeping parameters for monitoring on-ground and on-board. The *OBCMonitor* fetches status information from the CommGW to provide them as housekeeping data to the K2CP. This proxy service was introduced to avoid an additional communication relationship between CommGW and K2CP and instead favor the already existing ones. We used the *FunctionalTest* for qualification and acceptance tests, as well as for the commissioning phase of RJo on ISS. Therefore, we implemented a friction test, a workspace test, a virtual spring to test the force feedback, and tests for the buttons and LEDs of RJo. Finally, the *bilateral controller* realizes the master side of the telepresence control system. For details about the design and performance of this controller, please refer to the description by Artigas et al. [3].

Kontur-2 control panel

The Kontur-2 control panel is a program running on PCC. The K2CP provides basic experiment control functionality on-board for the following three tasks:

(1) Provide a cosmonaut user interface—A small user interface informs the cosmonaut about the current joystick and experiment state (see encircled annotations in Figure 8).

- Joystick state display and related commands ①
- Space, ground and communication system states ②
- Experiment state display and start/stop functionality ③
- A chat window to communicate with ground control ④



Tasks:
1. Aiming
2. Pursuit Tracking
3. Contour Following
4. Peg in Slot
5. Line Tracing
6. Spring Pulling
7. Free Telepresence

Task Description:
Move spring to a predefined position (green arrow)

hold position for 5 sec

Time limit: 30 sec

Task Status:
BRING JOYSTICK TO NULL POSITION

Figure 9. **Mission control panel**

(2) Command and display joystick state—K2CP continuously tries to connect to the on-board joystick and displays its current state. Joystick state is categorized into a set of states, which are: Normal States (init, standby, calibrated, halted), Error States (calibration error, recoverable error, fatal error) and Application States (German or Russian experiment, joystick test application). A very restricted set of commands is available to allow the cosmonaut to drive the joystick along these states.

(3) Operate as a communication relay to ground segment—Under the hood K2CP acts as the central communication and control relay between joystick, experiment applications, ground communications and ground mission control. Therefore, it manages slow rate TM/TC data from and to the joystick and ground link. It is also responsible to distribute high rate position/torque/button data from the joystick on-board, to synchronize the joystick application module and their related application user interface and to provide communication means for them. Finally, it tunnels the protocol used for experiment data (K2ACI) through the PUS based communication system.

Mission control panel for telepresence

The mission control panel is equipped with three UDP-interfaces to

- receive the video stream from the telepresence setup
- obtain position data from the joystick
- establish a communication channel with ground control

We designed the panel (see Figure 9) to present only the minimal amount of information necessary to control the experiments. Therefore, we used stacked widgets to minimize the number of items on the panel. The display inside each widget is controlled by a state machine so there is no need to provide graphical control elements such as a tool bar. The four widgets on the left part of the mission control panel display

- a list of all available tasks (highlighting the current task)
- a short description of the current task
- the current status of the task or specific instructions
- the optimal joystick start position together with the current joystick position

The right part of the mission control panel shows the video stream. Depending on the task it is possible to superimpose a small widget to provide additional graphical information about the state of the task.

The central component of the mission control panel is a state machine where all possible states and state changes of the experiments are predefined. State changes can be triggered either by the cosmonaut pressing a joystick button or by signals from ground control. Information about state changes inside the mission control panel are immediately sent to ground control. Each state change triggers a consistent display of textual and graphical information inside the widgets of the panel.

Audiovisual (AV) services

For the task board setup experiments, only a video stream of the robot or static camera had to be displayed to the cosmonaut. This was done by the DLR_MCP_TP. For the SpaceJustin experiments, the space segment software was extended by the AV services to provide bidirectional audio and video transmission. This module consists of three components using the GStreamer multimedia framework [16] *VideoOut*, *AudioOut*, and *AudioIn*.

VideoOut is responsible for capturing ISS on-board video of the experiment and sending it to earth via a restricted communication channel. For this, we use an already existing Sony Z7E camera on-board of the ISS which can be connected to the IEEE 1394 (4-pin) socket of the PCC. The video (640x480px @ 30fps) is captured using DirectShow and compressed using the H264 encoder [17] with low-latency configuration and a maximum bitrate of 512 kbit/s. The resulting data is streamed to earth using RTP protocol via UDP.

AudioOut realizes the audio downlink from ISS to earth. Due to the low quality of the on-board microphone of the PCC, we use the microphone of a headset which can be connected to the phone-connectors of the PCC. The audio signal is captured using DirectShow [18] and compressed using the Opus encoder in 'voice' configuration [19] and a maximum bitrate of 64 kbit/s. Like the downlink video, the data is streamed using RTP and UDP.

AudioIn receives the audio stream of the experiment partner on earth and provides it to the cosmonaut. Therefore, the RTP-UDP data stream is received, decoded (Opus) and finally played back by the cosmonauts headset using DirectSound [20].

6. COMMUNICATION SOFTWARE

From the application point of view communication between ISS and ground control is a mandatory service, but should be as transparent as possible. Although its data transfer characteristics influence the results of the experiments directly, it is independent from the application and non-controllable by the application itself. The effectively used communication method can be easily replaced due to its black-box character, using standard socket interfaces for any external access. This decouples the system development process from mutual dependencies between application and communication. The communication module can be implemented and tested without its final application environment for reaching maximum continuous throughput as close as possible to the limiting constraints caused by physics and equipment.

Customization of serial data link for control applications

A serial line is the most practical way to realize a bidirectional straight connection of at least two different components (a control unit and an execution unit) over a substantial distance with minimal use of hardware. But this approach makes it difficult to handle data of different quality especially under real-time constraints. It is obvious that an application specific data transfer structure is necessary in order to obtain a quasi-parallel data transfer behavior over a common serial byte stream with sufficient reliability.

In case of Kontur-2, a multi-channel concept was selected in order to meet the different needs of different data streams:

- Realtime channel for cyclic transfer of commands to control of the remote device.
- Realtime channel for cyclic transfer of telemetry from the remote device.
- Live video stream with application specific resolution (if necessary even bidirectional).
- Live audio or chat channel for direct information exchange with the cosmonaut.
- Request channel for both cyclic and acyclic intervention on the remote system (activity trigger, manual commands like parameter update, etc).
- Return channel for both cyclic and acyclic replies from the remote system (telemetry, acknowledgement).
- An autonomous closed loop channel for automatic services (e.g. file transfer).

It is mandatory for the intended data distribution to ensure the following major aspects:

- Specific properties of the used communication concept must not dominate the overall system behavior (e.g. huge artificial signal delay, transfer rate or extensive jitter).
- The transfer of cyclic data has to be carried out uniformly and continuously within a predefined response time. Its transmission is performed fast with just low level error detection but no correction. The quality of transmission depends directly on the hardware reliability, atmospheric and astronomic impacts. The continuity of the data flow must not be distorted by any kind of data buffering or transmission bursts, but an occasional loss of cyclic generated data packages is acceptable.

This statement relies on a simple rule of thumb well known from basic control theory: In principle the risk for a drastic system impact caused by losing isolated data packages randomly out of a cyclic compiled data package stream is indirectly proportional to the product of the dominating system time constant and the applied data package transfer frequency.

- The transfer of acyclic data is essentially the opposite of handling cyclic data. Not the timing of transmission is in the main focus but rather its integrity and reliability. Nevertheless an acyclic data transfer must not violate an application specific maximum time slot. In practice more than one individual acyclic data channel will be opened at the same time. In general the loss of acyclic data packages is a severe problem but it can be minimized by using a safe data protocol for the data transmission. If required, acyclic data channels in up- and downlink can be linked together forming a closed loop in order to process autonomous services between ground and space segment.
- The separate transmission of different types of data with an internal interrelationship should be performed quasi-simultaneously or at least with an imperceptible difference in time in order to avoid divergent sensory impressions on the operator side (e.g. real-time motion data and video camera signal).
- The ongoing data transfer has to be monitored permanently and should be resumed automatically in minimum time after a breakdown. The transfer of telemetry is pressed ahead whenever possible even in case of partial jamming or malfunction of the link.
- In case of partial failure of the communication link (no up- or downlink) the system always tries to transfer a maximum of information. This means that even commanding can be exceptionally performed without acknowledgement. In this situation the recipient is responsible for making the most of the obtained data.

Kontur-2 data link design

The main feature of the data link design is the mapping from three supported virtual data links with different transmission characteristics to the internal uniform serial data stream of the radio link. Figure 10 visualizes the methodology for data transmission from ground control to ISS and vice versa.

The bidirectional link structure is a symmetrical design which does not differentiate between up- and downlink. Just the parametrization is different. The access of applications to the data exchange mechanism is exclusively coordinated by OBC on ISS and OGC at ground control. Both devices work as data gateway to the encapsulated internal radio link between the tracking station (TS) on-ground and the transceiver (CUP) on ISS. Due to the bidirectional nature of the transmission both gateways support a sender block for data forwarding and a receiver block for data reception which work in parallel. Each gateway relies on a two-CPU architecture:

- Main-CPU, which is used as application interface and
- Com-CPU, which keeps on running the clocked serial bit stream over the radio link.

Local data exchange at each gateway is managed by a full duplex backplane connection. This technique supports both a low level memory-mapped message queueing for fast data exchange and a more sophisticated backplane driver for slower IP based networking.

Main-CPU— It acts as an event-driven prioritized multi-port application interface, which is directly addressable by IP networking from outside. Its graded output data are buffered for a cyclic but not synchronized collection and processing by the Com-CPU, whose principal task is to maintain a predefined steady bit-stream towards the radio transmitter. Due to the concurrent bidirectional nature of transmission, the Main-CPU supports a sender- and a receiver-block without any mutual dependencies. Within the sender-block, the contents of incoming IP-based data channels will be classified into different priority levels by the Main-CPU in order to guarantee a data type dependent forwarding within the transmission block by the Com-CPU. User data is handed over to the Com-CPU via hierarchical priority queues what allows additionally a compensation for temporary data-jams within the transmission-block. Normally these queues are working as FIFO buffers except for the real-time link which always uses the most recent data packages only. At the receiver-block, incoming data at the dispatch queues of the Main-CPU triggers the event-driven pre-emptive routing of that prioritized data towards the corresponding outgoing IP data channel.

With minimal modifications it is possible to connect on-ground application data channels with their on-board counterparts directly bypassing the original OBC / OGC connection, because all application interfaces are consequently designed for IP-based networking.

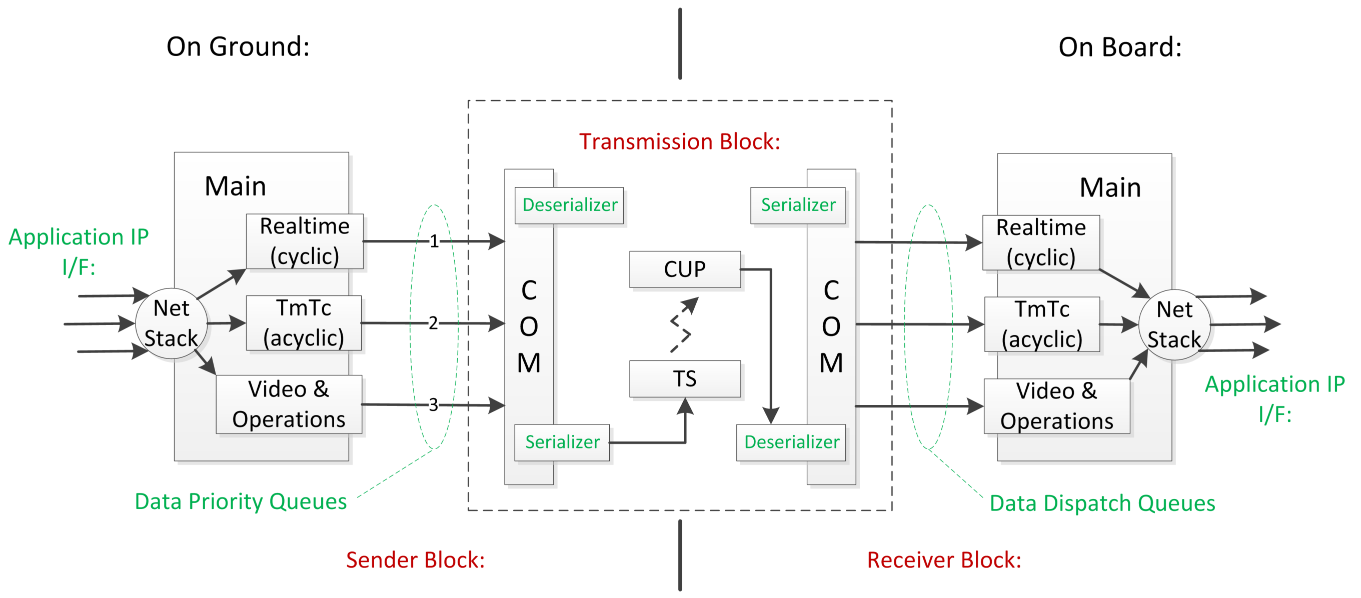


Figure 10. **Block diagram for uplink direction of the bidirectional link design**

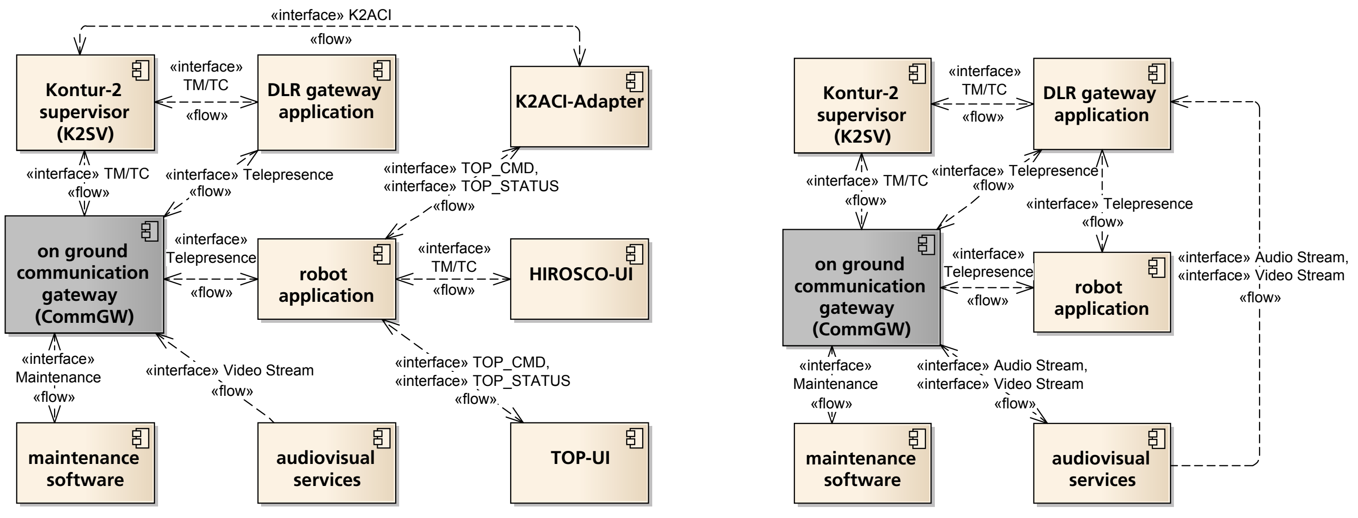
For safety reasons a permanent supervision service on the Main-CPU both monitors and tests autonomously the state of all application links and the connection to the remote Com-CPU for correct operation. In case of idleness or breakdown existing links will be immediately reset or replaced by new connections for the same purpose.

Com-CPU—It conducts the necessary subsequent processing of incoming data streams from both Main-CPU and the radio link. It encodes or decodes the application data in accordance with the CCSDS protocol regulations and manages their further dispatch.
Similar to Main-CPU, the Com-CPU supports also a sender- and a receiver-block without any mutual dependencies for concurrent bidirectional data transmission. Within the sender-block, all data available from several Main-CPU priority queues is packed up sequentially - starting with the highest priority - in a common transport frame up to reaching its capacity limit. This data packing is triggered by cyclic clocked polling of the Com-CPU only and does not wait for additional synchronization-events from Main-CPU. It strictly pays attention to transmission specific data properties (e.g. priority, repetition rate etc.) and concurrently assures that the outgoing bit stream is compliant with the maximum admissible radio link bit rate. The transmission of not available or not intended or capacity exceeding data for the current transport frame is postponed to the next transfer cycle. This management of dynamic transport frames is mandatory for obtaining a maximally uniform and continuous user data flow over the radio link without violating the supported maximum radio link bit rate. The drawback of the dynamic transport frame management is the reduction of the user data rate due to the inevitable usage of additional transfer protocol layers defined by CCSDS and a slight jitter due to the unsynchronized data exchange between Main- and Com-CPU.
Once a transport frame is created and ready for dispatch, it is finally serialized for transmission over the radio link.
At the receiver-block, both CCSDS protocol information and user data are extracted out of the received serial bit stream. After classification in the transmission-block, user data is mapped to their corresponding Main-CPU dispatch queues in accordance with their priority.
Because the radio transmission is a major source of bit errors, this transfer level is monitored by CRC in order to filter out destroyed data frames as soon as possible. Securing data integrity and reliability over the complete OBC / OGC data link is a higher-level feature exclusively under control of application specific data protocols.

A critical phase of operation is the communication startup at the beginning of a radio contact (the so-called auto-negotiation between OGC and OBC). First it is necessary to verify the correct local interaction between Main- and Com-CPU on both sides (OGC and OBC) independently. In a second step each side of the radio link waits for the reception of data frames which are interpreted as alive-indicator from the remote side. Finally the link master – in this case located on-ground – checks and configures the link slave state of the opposite side. Error cases are detected by reception timeouts or behavioral anomalies during both the bilateral master-slave startup sequence and the operational data transfer later on. The final approval of the total data link for usage is bound to the correct mutual reception of the first application data.
In case of error, both master and slave try to recover the affected data connection autonomously. If this does not work the current software version is marked as defective and will be replaced with the last executable software version from top of the backup stack. This stack contains a version history with several entries. It ends up in an elementary version of the communication software which allows at least remote service access to the link slave on ISS from master on-ground. It is not possible to compensate a hardware breakdown by software as the data link hardware does not include any kind of redundancy.

7. Ground Segment Software

The ground segment software for both experiment setups is depicted in Figure 11. One can easily see that many software components are identical in both setups. Despite of the same name, the *robot applications* are different for each setup and so are the robots they control. The *on-ground communication gateway* was already detailed in Section 6. The *Kontur-2 supervisor* (K2SV) allows the operators on-ground to monitor the housekeeping data provided by the



(a) Task board setup

(b) SpaceJustin setup

Figure 11. Ground segment software overview

space segment software. Moreover, we use the *maintenance software* to perform software updates in the space segment and to supervise the on-ground software execution. To communicate with our colleagues from RTC in St. Petersburg via Internet, we established a *gateway application* that provides the corresponding network address translation (NAT) rules for this purpose. For RTC's task board setup, the data is directly routed between the space segment and St. Petersburg. For the SpaceJustin setup, we route the data between RTC and the robot application to enable them to control the left arm. The *audiovisual (AV) services* are responsible for audio and video streaming to and from the space segment. While in the task board setup only a video stream is uplinked to the DLR_MCP_TP we established bidirectional audio and video streams to the space segment and to St. Petersburg in the SpaceJustin setup. In the task board setup we have additional software components that control the task execution. The user interfaces display the task execution and robot status to the operator. He can also trigger the initialization of a new task with it. The actual start of the task is initiated by the cosmonaut. Therefore, the robot application is connected to the K2SV via K2ACI through an adapter for protocol conversion. The K2SV in turn is routing the data via the CommGWs and K2CP to the mission control panel. In the remaining section these components will be further discussed.

Robot application for the task board setup

The robot application software for our experiments with the task board setup (cf. Figure 12) uses the same OS as RJo and, therefore, also has the same partitioning into kernel and user space. No sophisticated application loader is present as the robot is located in our lab so there is no risk of memory corruption by radiation. Hence, only a Sercos master runs in kernel space to provide an interface to the Sercos field bus of the ROKVISS robot.

Similar to the joystick application of the space segment software we use the HIROSCO component framework to build up the actual application in user space. Again, supervisor and the IP-Comm component for TCP/IP connections are taken from the framework. Besides that, the *robot controller* is responsible for the control of the ROKVISS robot at a sampling rate of 500 Hz either in joint position, in joint torque or in joint impedance control mode. The *bilateral controller*

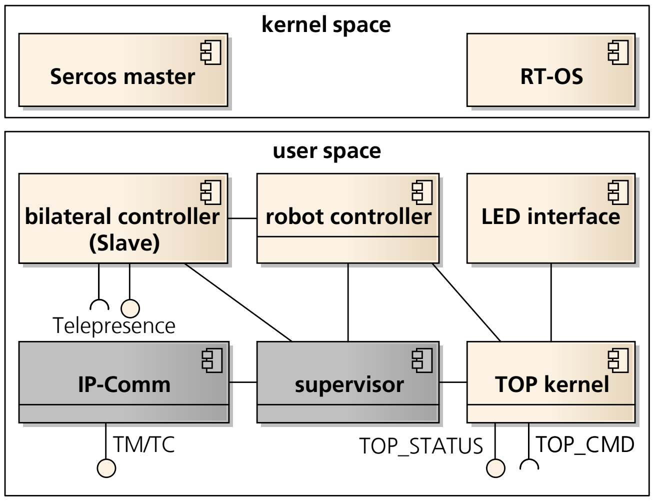


Figure 12. Software components of the robot application

implements the slave side of the telerobotic control system and, therefore, acts as the counter part to the bilateral controller as part of the space segment software. It uses the robot controller in joint torque mode to perform joint movements. For more details, please refer to the explanation of Artigas et al. [3]. Before a cosmonaut can start the execution of the desired tasks the robot has to move to a specific starting position, e.g. in the middle of the LED screen. This movement is accomplished by a high-level command sent from the mission control panel (cf. Figure 6) once the cosmonaut confirms readiness. This command is processed by the *task-oriented programming* (TOP) component [21]. This component is able to move the robot to arbitrary positions within its workspace by means of a variety of interpolators that access the joint position interface of the robot controller. This way, the robot is commanded to the predefined starting position for the currently selected task. Additionally, the TOP component uses the LED interface to draw the LED patterns required for the task. This LED interface is deployed on another machine as the remaining application because this machine has the required hardware interface to the LED panel. Thus the LED interface is not realized with HIROSCO, but attached via UDP messages.

Robot application for the SpaceJustin setup

SpaceJustin is a very sophisticated humanoid robot and the description of the complete robot application is beyond the scope of this paper. The interested reader can find a good introduction into this topic by referring to the publication of Borst et al. [14]. From the Kontur-2 point of view, we managed to integrate the same bilateral controller as for the task board setup. Hence, we do not have to modify the space segment when switching between the two setups.

Task-oriented programming and execution

Having in mind that for each experiment session a time slot of not more than eight minutes is available, the efficiency of the experiment handling had a great impact on the design of the commanding scheme, both on-board and on-ground. First, the cosmonaut on-board must have the tools to perform the desired operations without a deep knowledge about the internal structure of the robot system on-ground. Second, the robot operator on-ground must be able to control the robot system by an easy-to-use graphical user interface. The backbone of such a high-level commanding scheme is a hierarchical command description based on the TOP approach [21].

Looking at the default experiment sequence (cf. Section 8), the operator on-ground only has to select and activate the desired task. In the following step the cosmonaut will confirm this selection, calibrate the joystick and start the robot task. This robot task performed on-ground will autonomously guide the cosmonaut step by step through the experiment. To achieve this ”autonomous guidance”, the robot task is described by a sequence of elemental operations (Elemop), whose respective end conditions are continuously checked by the task execution software on-ground. Namely, two subtasks (operations) are necessary to perform an experiment run: *start<Task>*, activated and executed on-ground, and *exec<Task>*, activated on-board and performed also on the ground robot equipment. The on-ground software for the task board application uses an XML schema to describe these subtasks in a hierarchical way. To give an idea of this concept, we simply describe the start and exec subtask for the aiming task, i.e. the task, in which the cosmonaut has to move the robot to a desired pose, highlighted by an illuminated LED. The *startAiming* task prepares the robot environment with the right controller, moves the end-effector into the start position, selects the correct camera view, switches on the first LED and starts the data logging mechanism:

```
<Operation name=”startAiming”>
  <Elemop ref=”powerOff ”/>
  <Elemop ref=”positionControl ”/>
  <Elemop ref=”powerOn ”/>
  <Elemop ref=”selectCamTcp ”/>
  <Elemop ref=”highlightLed0 ”/>
  <Elemop ref=”homePosition ”/>
  <Elemop ref=”startLogging ”/>
</Operation>
```

After execution of this subtask, the on-ground system will send the task name to the on-board system to tell the cosmonaut what the next experiment is. Simultaneously, it is ready for the activation of the telepresence performance task, triggered by the cosmonaut’s command, e.g. *execAiming*. For that the robot controller will be switched from the position control mode into the telepresence control mode. In the following all the desired steps (highlight the desired LED and check the cosmonaut’s motion commands for achieving this pose) will be performed.

```
<Operation name=”execAiming”>
  <Elemop ref=”powerOff ”/>
  <Elemop ref=”telePresenceControl ”/>
  <Elemop ref=”powerOn ”/>
  <Elemop ref=”highlightLed7 ”/>
  <Elemop ref=”checkPos7 ”/>
  <Elemop ref=”highlightLed0 ”/>
  <Elemop ref=”checkPos0 ”/>
  <Elemop ref=”highlightLed5 ”/>
  <Elemop ref=”checkPos5 ”/>
  <Elemop ref=”highlightLed0 ”/>
  <Elemop ref=”checkPos0 ”/>
  <Elemop ref=”stopLogging ”/>
  <Elemop ref=”powerOff ”/>
</Operation>
```

The execution of such subtasks is supervised by the *TOP kernel* (cf. Figure 12), i.e. a state machine, decoupled from the robot controller of the robot application. The state machine is separated from the robot controller to avoid a blocking behavior. It is sufficient, due to the asynchronous command mode (incoming *exec* command from space), that this state machine is running at 10 Hz.

Kontur-2 supervisor (K2SV)

The Kontur-2 supervisor (K2SV) links the space communication system to all other ground systems. At the same time it provides a detailed state view of all connected systems to the supervisor and experimenter on-ground. The following components report their state to K2SV:

- Space segment: RJo and K2CP
- Communication system: OBC, OGC and CUP
- Ground segment: German and Russian experiment

To serve this task K2SV relays and converts packets within a variety of data links and protocols such as the space link for TM/TC (PUS protocol), the experiment data link (K2ACI and RTC command protocol) and the routing of chat messages (K2ACI protocol).

Audiovisual (AV) services

The AV services provide different modules for bidirectional audio and video streams between earth and ISS. For the task board setup, only the video stream of the ROKVISS robot or of the static camera is sent to the DLR_MCP_TP. The modules for real bidirectional communication have been added for the SpaceJustin setup.

Uplink video stream—The ROKVISS robot is equipped with a pair of cameras for stereo vision. These were used during the original ROKVISS experiments. Due to the lack of 3D video playback devices on-board the ISS and the limited bandwidth of the uplink channel, it was not feasible to use 3D video in Kontur-2. Only using the video of the left or right camera had the drawback of a lot of occlusion due to the stylus of the ROKVISS robot. This occlusion would have made it difficult to achieve good results in the pointing tasks of the experiment. As the task board, contour and robot stylus had the same distance to the cameras of the robot, we finally decided to merge the stereo images of the video using the left part of the left camera and the right part of the right camera. The width of the image parts was chosen so the tip of the stylus was completely visible in the resulting image (cf. Figure 9). For the Kontur-2 mission, the ROKVISS robot was upgraded with an external camera showing the task board. The video output of all cameras was captured and recorded on-ground for experiment evaluation.
During an experiment, the robot application selects which

camera to use. For this, a TOP command is sent to a *selector* process which forwarded the corresponding video stream to a shared memory for further processing. By this separation of the video capture and the sending to the communication partners, we realized a system which is very robust to failures of individual components. It also allows us to easily add new communication endpoints or implement new processing steps. The bandwidth of application data in the uplink channel to the ISS has been limited to 96 kbit/s for video streaming. Due to this restriction, we had to highly compress the video stream while keeping a low latency to allow telerobotics experiments. We use the H264 encoder in low-latency configuration with a maximum bitrate of 90 kbit/s (85 kbit/s for the SpaceJustin setup) to compress the incoming video stream. The resulting data stream is sent to the ISS via UDP using the RTP protocol. We also display the video at the ground control to allow the system operators to follow the experiment and observe the performance of the video stream.
For the SpaceJustin setup, we extended the *selector* process to also handle the video of the SpaceJustin head camera. In addition, we extended our software to forward the resulting video stream to our colleagues at RTC using the same processing pipeline we used for the ISS video stream but with an increased bitrate.

Downlink video stream—The sending processes of the downlink video stream were described in Section 5. The resulting video stream is received on-ground and forwarded to a shared memory for further processing and recording. From there, the video is forwarded to the experiment room to allow visual communication of the experimenter and the cosmonaut. In addition, the video is forwarded and displayed at the ground control to allow the system operators to keep track of the experiment. Using the same processing pipeline as for the uplink video, we also forward the downlink video stream to our colleagues at RTC.
To allow easy and intuitive communication between the partners during the experiment, we implemented an additional video stream from RTC to DLR. The processing is similar to the ISS downlink video stream described earlier.

Uplink audio stream—For the SpaceJustin setup, a voice loop between the cosmonaut and the experimenter on earth was required. The implementation of the audio uplink has been challenging because it shared the limited 96 kbit/s uplink channel with the video stream. We solved the problem using the Opus codec which provides good sound quality at short latency and low bitrates. We use the encoder in *speech* mode with a maximum bitrate of 8 kbit/s. The processing pipeline of the audio stream is similar to the processing of the video data: The signal is captured and stored in a shared memory. From there it is distributed to the ISS (via RTP UDP), RTC, and the ground control.

Downlink and RTC audio streams—Section 5 described the sending processes of the downlink audio stream. Similar to the video streams, the downlink and RTC audio streams are received on-ground and stored in a shared memory each. They are then forwarded to the experiment room and the ground control. The downlink audio stream is also forwarded to RTC.

Using the AV services, we realized a system fully capable of bidirectional communication between the cosmonaut on-board the ISS and the experimenter on earth.

Maintenance software

Although the operation and interaction of the ground segment and the space segment is dominated by automatic processes, their work flow is additionally monitored by an operator. For this purpose the ground segment (via OGC) and the space segment (via OBC) permit extensive remote access in parallel to the ongoing experiment support. Under nominal conditions the operator limits himself to acquire and archive status data and event statistics. He prepares the ground segment for an upcoming radio contact, monitors the correct operation of all necessary processes and services, checks the software versions in use and their compatibility, supervises the available link bandwidth, and informs the experimenter if the experiment environment degrades or indicates a malfunction. Special attention is emphasized on the ”Acquisition of Signal” (AOS) phase at the beginning of the radio contact, possible degradation of the radio signal due to shadowing effects during the experiment execution and the notification of the upcoming end of the radio contact (LOS). In addition, the operator is responsible for performing software updates in both the ground and the space segment. Opposite to the already mentioned activities, an update process cannot be conducted concurrently to an experiment execution and requires an always unambiguous and fail-safe procedure.
In order to handle all these responsibilities efficiently, the operator uses a set of different software tools:

- An interactive script-based command-line tool, which supports arbitrary line access for spontaneous modifications of predefined command sequences. (For safety reason each command has to be confirmed by the operator separately).
- A scalable process control and monitoring tool, which allows a centralized access to a distributed system architecture.
- Standard IP services for remote access (Telnet), file transfer (FTP), and an interactive real-time operating system shell, which allows an open manual remote access to components of the space and ground segment in parallel to the ongoing experiment support as soon as the data exchange is online.
- Several discrete realtime displays for textual progress indicators, trend analysis, and milestone completion confirmations.
- A full data logging option, which stores all sent and received information.

Data logging and evaluation

As mentioned in Section 3, it was a mandatory requirement to record all the available experimental data. One of the main challenges was to save the data coming from a lot of various sources in a coherent way. In particular, it was necessary to associate the cyclic telepresence data transferred via the real-time channels (cf. Section 6) with the trigger commands and acknowledgments transferred in the acyclic request and return channels or generated by the robotic system on-ground during the task execution. To achieve a continuous and coherent logging of all these accruing data, we established one and only one recording instance as data sink of all sources on-ground.

The logging component is connected to all the components providing experiment data by means of our internal data communication system *links_and_nodes* (LN), which is a framework for easy creation and maintenance of distributed computing networks. The communication is based on a publisher-subscriber model, which provides a real-time, low-latency, fixed sized signal publishing. So each data generation/collection component on-ground has an LN publisher plugin connected with the logging process, which serves as the subscriber. The logging process will be started or stopped by the respective step within *start<Task>* or *exec<Task>*. We have chosen the Matlab data format as data format for the recording sets, to facilitate the evaluation and analysis with a common and widely used tool.

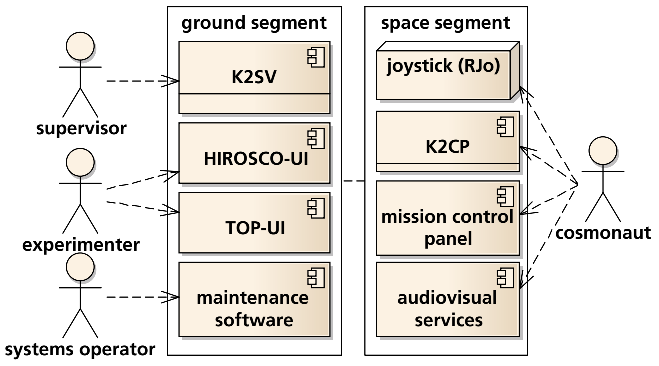


Figure 13. **Roles and responsibilities during a Kontur-2 experiment**

For more details on the evaluation, analysis and interpretation of the data sets acquired during the task board experiments refer to the explanations of Weber et al. [4].

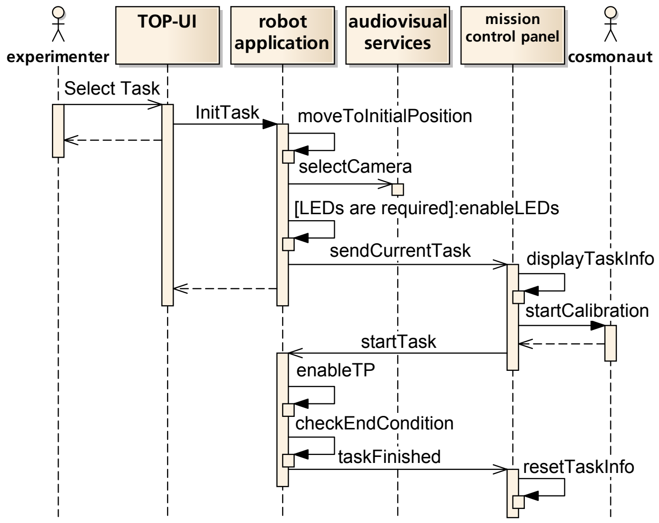


Figure 14. **Task execution sequence**

8. Mission operations

So far, Sections 5, 6 and 7 explained the static architecture of the Kontur-2 software. This section will focus on the dynamic architecture of the software and its interaction with the humans involved.

For mission operations we defined three roles for the mission operations team in the ground segment, namely the *supervisor*, the *experimenter* and the *systems operator*. They are shown in Figure 13. The supervisor continuously monitors the K2SV screen to search for anomalies in the housekeeping data of the joystick application and the communication gateways. Additionally, he is connected via phone to the ground station in Weilheim to hear about events such as acquisition and loss of signal to ISS. The experimenter has a detailed plan which tasks should be performed during the experiment session. He selects the tasks one after the other via the TOP-UI and monitors their progress of execution. Using the HIROSCO-UI he supervises the housekeeping data of the robot application and is able to reset error conditions of the robot. During an experiment session the systems operator checks the status of the ground software. During maintenance sessions e.g. for software upload he is responsible for actually deploying the software on the equipment in space using the maintenance software. In the ISS there is only the cosmonaut interacting with the space segment software and hardware.

Figure 14 shows the sequence of interactions during the execution of a task which was already described in an abstract manner in Section 4. The experimenter uses the TOP user interface to select the task that should be performed by the cosmonaut. This interface brings the robot application to perform the initialization sequence for this task: The robot moves to the initial position for the task, a suitable camera is selected for the uplink stream by the AV services, LEDs are enabled if required for the task and, finally, the mission control panel is notified about the selected task. Once the panel receives this notification, it displays information about the task to the cosmonaut and asks him to put the joystick handle to a predefined position. Upon completion of this calibration the panel triggers the robot application to start the task. The robot application enables the robot and its telepresent control after reception of this request and monitors the end conditions of the task. During that time, the TOP user interface shows the progress of task execution (not depicted in the figure) to the experimenter. As soon as the robot application detects that the task is accomplished by the cosmonaut, it notifies the mission control panel. This panel in turn informs the cosmonaut that the task has been completed and asks him to wait for the next task.

9. Conclusions

During 23 experiment sessions in 2015, the Kontur-2 software formed the basis of the successful completion of our experiments. All software components described in this paper worked reliably so that every ISS orbit allocated for DLR could be used to conduct our experiments. Cosmonauts Oleg Kononenko and Sergey Volkov reported that the experiment tasks were easy to perform with the provided hard- and software. Regarding the requirements, we were able to successfully integrate the ROKVISS infrastructure into the Kontur-2 mission. Thanks to an elaborated video streaming implementation the bandwidth of 96 kbit/s for uplink user data was sufficient to provide a video stream the cosmonauts were able to work with. During the whole mission RJo's microSD card as main memory storage did not show any errors so the fail-safe application loader was not challenged on this regard. However, an update of RJo's software was performed where the concept of multiple software storage locations worked out as expected.

From the scientific point of view the Kontur-2 mission contributed to the fields of telepresence technologies and human factors. On the one hand we proved that stable and realistic haptic teleoperation between an operator in an orbiter and a robot on a planetary surface is possible using the time domain passivity approach [3]. On the other hand we found evidence that the performance losses caused by a degraded proprioception of the cosmonaut in micro-gravity can partially be compensated with adjusted mechanical properties of the input device (e.g. higher damping) [4].

In 2016 we will conduct further experiments using the Kontur-2 software and hardware infrastructure. During a local simulation study on-board the ISS, the spring stiffness, damping and virtual inertia of RJo will be varied systematically while the cosmonaut's control precision will be determined during basic movement tasks like tracking and aiming. As result of this study, we are able to describe the relationship between physical joystick parameters and the human performance in 1G and 0G conditions. Besides that, our colleagues from RTC will continue their experiments with their mobile robotic platform. Finally, there are already plans to control additional robotic systems from ISS such as DLR's lightweight rover unit [22] using a geostationary satellite for the communication to extend the experiment duration from a maximum of eight minutes to 45 minutes and more.

References

[1] R. Martinez, K. Goodliff, and R. Whitley, "ISECG global exploration roadmap: A stepwise approach to deep space exploration," in *AIAA SPACE 2013 Conference and Exposition*, vol. 5504, 2013.

[2] K. C. Laurini, B. Hufenbach, J.-C. Piedboeuf, D. H. Kuninaka, N. Sato, and D. J. Hill, "The ISECG global exploration roadmap strengthening exploration through increased human robotic partnership," in *64th International Astronautical Congress*, Beijing, China, 2013.

[3] J. Artigas, R. Balachandran, C. Riecke, M. Stelzer, B. Weber, J.-H. Ryu, and A. Albu-Schäffer, "Kontur-2 - force-feedback teleoperation from the international space station," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, 2016, pp. 1166–1173.

[4] B. Weber, S. Schätzle, C. Riecke, B. Brunner, S. Tarassenko, J. Artigas, R. Balachandran, and A. Albu-Schäffer, "Weight and weightlessness effects on sensorimotor performance during manual tracking," in *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications, Eurohaptics*. London: Springer, 2016, pp. 111–121.

[5] C. Riecke, J. Artigas, R. Balachandran, R. Bayer, A. Beyer, H. Buchner, T. Gumpert, R. Gruber, F. Hacker, K. Landzettel, G. Plank, S. Schätzle, H.-J. Sedlmayr, N. Seitz, B.-M. Steinmetz, M. Stelzer, J. Vogel, B. Weber, B. Willberg, and A. Albu-Schäffer, "Kontur-2 mission: the DLR force feedback joystick for space telemanipulation from the ISS," in *The International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, Beijing, 2016.

[6] J. Artigas, C. Riecke, B. Weber, M. Stelzer, R. Balachandran, S. Schätzle, R. Bayer, M. Steinmetz, J. Vogel, B. Brunner, A. Albu-Schaeffer, M. Guk, V. Zaborovskyi, A. Kondratiev, V. Muliukha, A. Silinenko, and O. Shmakov, "Force-feedback teleoperation of on-ground robots from the international space station in the frame of the "kontur-2" experiment," in *Proceedings of "International Extreme Robotics Conference", St.Petersburg, Russia*, 2016 (accepted for publication).

[7] G. Hirzinger, K. Landzettel, D. Reintsema, C. Preusche, A. Albu-Schäffer, B. Rebele, and M. Turk, "Rokviss - robotics component verification on ISS," in *Proc. of the 8th Int. Symposium on Artifical Intelligence, Robotics and Automation in Space, iSAIRAS*, Munich, Germany, 2005.

[8] K. Landzettel, V. Zaborovskyi, E.Babkin, M. Belyaev, A. Kondratiev, and A. Silinenko, ""kontur" experiment on russian segment of the iss," Scientific Readings in Memory of K.E. Tsiolkovsky, 2009. [Online]. Available: http://readings.gmik.ru/lecture/2009-EKSPERIMENT-KONTUR-NA-ROSSIYSKOM-SEGMENTE-MKS

[9] A. Schiele, "Meteron - validating orbit-to-ground telerobotics operations technologies." in *11th Symposium on Advanced Space Technologies for Robotics and Automation (ASTRA)*, 2011.

[10] A. Schiele, M. Aiple, T. Krueger, F. van der Hulst, S. Kimmer, J. Smisek, and E. den Exter, "Haptics-1: Preliminary results from the first stiffness jnd identification experiment in space," in *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications, Eurohaptics*. London: Springer, 2016, pp. 13–22.

[11] T. Krueger and A. Schiele, "Preparations for the haptics-2 space experiment on-board the international space station," in *13th Symposium on Advanced Space Technologies for Robotics and Automation (ASTRA)*, 2015.

[12] A. Schiele, "Towards the interact space experiment: Controlling an outdoor robot on earth's surface from space," in *13th Symposium on Advanced Space Technologies for Robotics and Automation (ASTRA)*, 2015.

[13] N. Y. Lii, D. Leidner, A. Schiele, P. Birkenkampf, B. Pleintinger, and R. Bayer, "Command robots from orbit with supervised autonomy: An introduction to the meteron supvis-justin experiment," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*. ACM, 2015, pp. 53–54.

[14] C. Borst, C. Ott, T. Wimböck, B. Brunner, F. Zacharias, B. Bäuml, U. Hillenbrand, S. Haddadin, A. Albu-Schäffer, and G. Hirzinger, "A humanoid upper body system for two-handed manipulation," in *Proceedings IEEE International Conference on Robotics and Automation*, Roma, Italy, 2007, pp. 2766–2767.

[15] M. Stelzer, B. Brunner, K. Landzettel, B.-M. Steinmetz, J. Vogel, and G. Hirzinger, "HIROSCO - a high-level robotic spacecraft controller," in *The 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Sapporo, Japan, August 2010. [Online]. Available: http://elib.dlr.de/67308/

[16] S. D. Burks and J. M. Doe, "Gstreamer as a framework for image processing applications in image fusion," in *Proc. SPIE Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications*, vol. 8064, 2011, pp. 80 640M–80 640M–7.

[17] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[18] Microsoft Developer Network, "Directshow documentation," https://msdn.microsoft.com/dd375454.aspx, accessed: 2016-10-15.

[19] K. Vos, K. V. Sørensen, S. S. Jensen, and J.-M. Valin, "Voice coding with opus," in *Audio Engineering Society Convention 135*, Oct 2013.

[20] Microsoft Developer Network, "Directsound documentation," https://msdn.microsoft.com/ee416960.aspx, accessed: 2016-10-15.

[21] K. Landzettel, B. Brunner, G. Hirzinger, R. Lampariello, G. Schreiber, and B.-M. Steinmetz, "A unified control and programming methodology for space robotics applications," in *International Symposium on Robotics*, Montreal, Canada, May 2000.

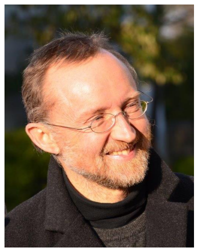
[22] A. Wedler, B. Rebele, J. Reill, M. Suppa,

H. Hirschmüller, C. Brand, M. Schuster, B. Vodermayer, H. Gmeiner, A. Maier *et al.*, "LRU-lightweight rover unit," in *Proc. of the 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, Noordwijk, The Netherlands, 2015.

BIOGRAPHY

Martin Stelzer *studied computer science at FH Ingolstadt and the University of Hagen and received his M. Sc. Degree in 2012. Since 2007 he has been working at the German Aerospace Center in the field of onboard software frameworks and was involved in the space projects ROKVISS and Kontur-2.*

Peter Birkenkampf *received his M.Sc. Degree in "Robotics, Cognition, Intelligence" from Technical University of Munich in 2013. He joined the German Aerospace Center Institute of Robotics and Mechatronics in 2011 and was involved in the space robotics projects Kontur-2 and Meteron Supvis Justin. His main interests lie in Human-Robot Interaction and Supervised Autonomy.*



Bernhard Brunner *graduated in Computer Science from the TU Munich in 1989 and joined the DLR Institute of Robotics and Mechatronics in the same year. He was involved in all of the Institute's space robotics projects, starting with ROTEX 1993, where he provided the predictive graphical environment. In the last decade, he focused his research work on a high-level task-oriented programming system, which was used in most of the DLR's space and service robotics applications (e.g. ETS-VII, ROKVISS, Robutler, Justin, Kontur-2). He has a long-term experience in software engineering, developing distributed applications, and in real-time programming.*

Bernhard-Michael Steinmetz *received his diploma in Electrical Engineering from Technical University of Munich (TUM, Germany) in 1989. He joined the DLR Institute of Robotics and Mechatronics in 1989 as a project and research engineer. Over the course of years he was responsible for system design and operation, hard real-time process control, robotics, communication (short and long distance), navigation and image processing in both national and international projects (e.g. ROTEX, ETS-VII, PSPE, ROKVISS, Kontur-1..2).*

Jörg Vogel *studied computer science at TU München. Since 1995 he is working at German Aerospace Center as a software developer, clean code campaigner and Java evangelist.*

Stefan Kühne *studied mathematics at the RWTH Aachen and received his diploma in 2005. Since 2014 he has been working at the German Aerospace Center as an external software developer. His main interests lie in GUI development, distributed computing and network programming.*