INTUITIVE "HUMAN-ON-THE-LOOP" INTERFACE FOR TELE-OPERATING REMOTE MOBILE MANIPULATOR ROBOTS

*Robert Valner[1,2], Veiko Vunder[2], Andy Zelenak[2], Mitch Pryor[2], Alvo Aabloo[1], Karl Kruusamäe[1]

[1]*Intelligent Materials and Systems Lab, University of Tartu, Nooruse 1, Tartu, Estonia*
[2]*Nuclear and Applied Robotics Group, The University of Texas at Austin, TX, USA*

ABSTRACT

Teleoperated robots offer safe means for working in environments that are life-threatening or otherwise inaccessible for humans. Due to the severe conditions in space, technologically advanced robots are frequently adopted for unmanned planetary exploration and carrying out maintenance tasks on manned missions. As the robots grow technically more complicated and capable, the control interfaces must, in turn, become simpler. We have developed TeMoto, a speech and hand gesture based supervisory teleoperation software for hardware agnostic mobile manipulation. TeMoto addresses several chronic technical issues associated with most teleoperation systems including the operator's SA, communication delays, and interface intuitiveness. This paper presents the functional description of TeMoto's resource management tool – Resource Registrar (RR) – by exemplifying it with a demonstration where a series of sensor failures are induced on a remote mobile manipulator robot. Despite the sensor failures, the software remains operational and a visual representation of the surroundings is sustained for the operator.

1 INTRODUCTION

Teleoperated robots offer safe means for carrying out work in environments that are life-threatening and/or otherwise inaccessible for humans [1], [2]. In addition to space, such robots can be used for underground mining, radioactive decommissioning, search and rescue, fighting wildfires, etc. The common denominator for most telerobotics applications is the unpredictability of the remote location the robot is expected to operate in.

Due to the severe conditions in space, robots are frequently adopted for unmanned planetary exploration and carrying out maintenance tasks on manned missions. There are already some remarkable space robots, such as NASA's Roboaut-2 and Valkyrie as well as many planetary rovers [2]. The state-of-the-art robots are yet to reach full autonomy and therefore, the need for human operator persists [3]. However, with the development of assistive software algorithms, it is possible to transition some aspects of robotic tasks from humans to the controller – a type of control regarded as shared autonomy. For instance, an algorithm can take care of the self-collisions of a robot manipulator and plan a collision-free path from one end-effector pose to another. Some advanced teleoperation interfaces incorporate predictive simulations of the future states of the robot and thus enable a more efficient use of the robot and operator's time [4].

As the robots grow technically more complicated and capable, the control interfaces must, in turn, become simpler [5]. One can easily see the analogy from the history of personal computers: the first computers could only be wielded by trained experts whereas modern computers – that are by far more complicated and computationally powerful – can be used by small children through highly intuitive User Interfaces (UI). An intuitive UI commonly utilizes the natural elements of human communication (e.g., speech, gestures, mechanoreception) and are, thus, frequently applied in teleoperation applications. Yet task and UI specific requirements often creep into the underlying architecture of the teleoperation software, which vastly reduces the software modularity and reuse.

Even the best UI design can only provide what the back end of the system supports. If the robot is capable of partially autonomous (sub)task execution, the role of the UI is to deliver access and ensure usability of these features for the operator. It is reasonable to claim that the quality and availability of autonomous capabilities may change depending on the nature, criticality, and environment of the task. Preferably, the teleoperation interface is able to adjust to the situation while sustaining familiar user-experience for the operator. Also, from the developer's perspective, the system should be scalable and its capabilities extendable in well-structured and unified way.

Given the above considerations, we have designed a ROS-based telerobotics software development framework – TeMoto [6] – that inherently supports the main modalities of human communication and advocates a modular design of software. TeMoto provides the functionality to ground instructions in Natural Language (NL) to executable actions, i.e., modular pieces of code that allow:

- integrating any available hardware and software resources, e.g., sensors, manipulators, object recognition algorithms;
- executing any user specific code with no restrictions to the use of third-party software libraries.

For telerobotics in space, power consumption and fault tolerance are mission critical factors. Complicated remote systems, such as planetary rovers, often contain a variety of sensors that complement each other's functionality. A mobile robot can have sensors such as 2D and 3D LIDARs for obstacle detection and mapping, cameras for visual confirmation, and other sensors that help interpret the surrounding environment. In order to achieve operational reliability, the more critical systems (including sensors) are duplicated [7] but there may also be some redundancy from the overlapping capabilities of different sensors. For instance, a 3D pointcloud from a LIDAR can still give the operator some understanding of rover's immediate surroundings after all camera systems have failed.

This paper presents the main functionalities of TeMoto's resource management tool – Resource Registrar (RR) – from the perspective of mobile manipulator robot platform. A brief overview is given about the architecture of TeMoto and its NL grounding methodology. More specifically, the RR provides functionality to:

- maintain a registry of active resources, their clients and dependencies of sub-resources,
- dynamically start and stop resources,
- notify the clients in case of resource status updates (e.g., resource failure).

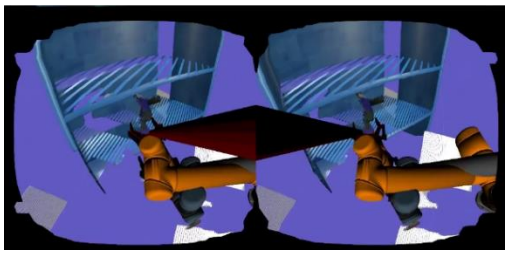
All in all, the RR helps to reduce energy consumption, efficiently utilize the available processing power, and support dynamic modifications to the hardware and UI layout of the teleoperation system. The next section of this paper gives an overview of related efforts, followed by a description TeMoto's RR. We demonstrate the practicality of the RR on a remote inspection sensor failure scenario.

2 PREVIOUS WORK

The original objective of TeMoto was to make controlling advanced robotic platforms intuitive for non-roboticist workers during inspection and manipulation in potentially hazardous environments, e.g., nuclear sites. TeMoto addresses several chronic technical issues associated with most teleoperation systems including the operator's Situational Awareness (SA), communication delays, and interface intuitiveness. To improve SA and reduce operator's mental modelling, we fuse relevant sensory data from the robot as well as the human into a single mixed-reality (MR) scene (Figure 1). The 3D MR scene can be displayed on a 2D computer screen (Figure 1a) as well as using a virtual reality headset (Figure 1b). The operator is able to change his or her point-of-view on the MR scene and trigger robot movements that are best suited for a given task. The operator controls the robot and the interface by means of intuitive hand gestures and verbal instructions. TeMoto is designed to be hardware-agnostic and has already been demonstrated on several robotic platforms (e.g., Clearpath Husky with two UR5 manipulators, KUKA youBot, and Yaskawa Motoman SIA5) to achieve large scale navigation (dozens of meters) and small (sub-millimeter) scale manipulation.

The first generation of TeMoto used Leap Motion Controller to track operator's hands as s/he was seizing gestural control of any of the robot's end-effectors (Figure 1a) [6], [8]. Additionally, a physical dial knob helped the operator to scale real-life hand movements to the most suitable range on the robot, thus enabling comfortable working ergonomics despite the requirements of the task. The same operator interface and modality could be used to navigate a robot across a warehouse (dozens of meters) and thread a needle with an eye width less than 1 mm [6], [8].

a)



b)

Figure 1: a) Operator interface of TeMoto using Leap Motion Controller to track hand poses, turn knob for scaling, and MR scene visualization on 2D screen. b) Visualization of TeMoto MR scene for virtual reality headset.

Instability and inefficiency due to time delays as well as any potential errors due to unwilling gestures were all mitigated by utilizing a supervisory ("human-on-the-loop") controller. For instance, when basic path planning capabilities were integrated to TeMoto, the operator could simply point to a place within the MR scene (Figure 1) and say, 'Robot please go' to trigger motion planning algorithm and the autonomous

movement of robot. Another advantage of the supervisory controller is that by delegating any task the robot is able to complete on its own, we reduce the workload of the operator and free up time that can potentially be used to control a fleet of robots.

3 RESOURCE REGISTRAR (RR)

3.1 Overview

Since the inception, TeMoto has grown into a ROS-based framework and toolbox that facilitates the creation of supervisory ("human-on-the-loop") teleoperation systems [9]. TeMoto can be divided into three hierarchical abstraction layers (Figure 2):

- **Supervisory layer** that handles the interaction with the operator and directs the rest of the system.
- **Management layer** is responsible for acquiring, managing and maintaining knowledge about resources via respective manager subsystems.
- **Resource Access** layer allows the management layer to access resources that are external to TeMoto, e.g., sensors and actuators (in the format of a ROS node).

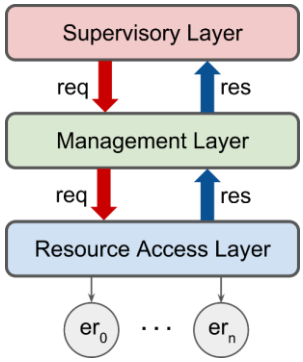


Figure 2: Architecture of TeMoto. doi.org/10.5281/zenodo.1226854

The architecture of TeMoto centers around the concept of a *resource*, i.e., something which *is provided upon a request*. Many things can be resources: sensors and algorithms are resources for acquiring and manipulating data; information about physical objects in the surrounding environment or a planet's weather can be regarded as resources for decision-making.

The given definition for a resource provides a lot of flexibility (e.g., resources that combine sub-resources) but in order to facilitate this flexibility efficiently, the RR was developed to solve the following issues:

- How to share a resource without allocating the same resource multiple times (multiple clients)?
- How to make sure that no clients are using the resource when deallocating it?
- How to notify clients if the status of a resource changes (e.g., the resource has failed)?
- How to notify the involved clients upon receiving a resource status update message from a sub-resource?

In TeMoto, every subsystem keeps track of its own use of resources, i.e., has its own instance of RR which contains a registry of all active inbound and outbound resource queries. Such approach allows the system to scale easily, i.e., a complex hierarchy of resources can be created without encountering the aforementioned issues. Hence TeMoto's RR provides the core functionalities for designing a fault-tolerant and resource-efficient robotic systems.

The next subsection describes the process of requesting a resource, followed by details about implementation of RR in subsection 3.3.

3.2 Actions and Resource Requests

The supervisory layer is responsible for parsing the NL instructions into executable actions, the other layers are responsible for providing resources that were requested by the actions.

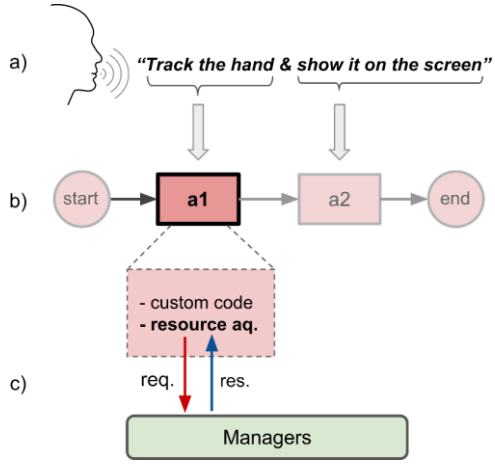


Figure 3: NL grounding in TeMoto. a) The operator gives an instruction in NL. b) The instruction is mapped to an action tree. c) Actions execute the developer-defined code and request resources from the Management Layer. doi.org/10.5281/zenodo.1227081

Figure 3 illustrates the process of grounding NL instructions into actions. The Supervisory Layer parses the NL utterance, e.g., "track the hand and show it on the screen" (Figure 3a), and maps every recognized instruction to an action and thus forming an action tree. The action tree is then executed, starting from the root node of the tree (Figure 3b). Note that the tree could also contain parallel branches. The actions access system resources through the manager interfaces (Figure 3c). Once an action

finishes, it can stay in the memory (asynchronous action), thus maintaining the allocated resources, or it can destruct right after execution (synchronous action), thus automatically releasing allocated resources (feature of the RR).

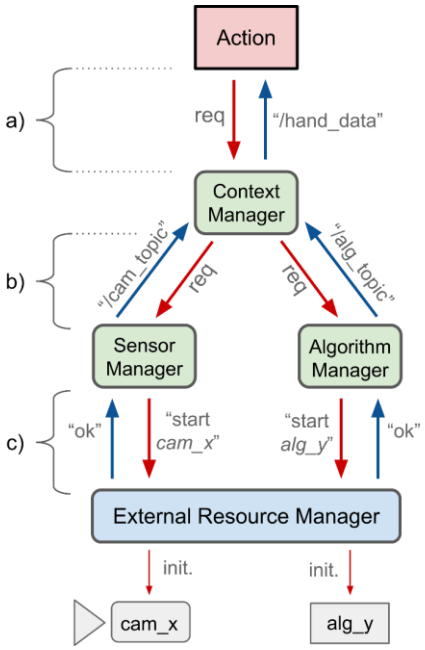


Figure 4: Example of resource abstraction. a) An action requires the position of operator's hand. b) The information about the hand is retrieved by combining two resources – a camera feed and an algorithm that uses the feed to detect the hand. c) The request to initiate a specific camera "cam_x" and specific algorithm "alg_y" is directed to the External Resource Manager.
doi.org/10.5281/zenodo.1227105

The term *abstraction layer* emphasizes the idea that a client of a given resource is oblivious to its origins. For example, if an action requires the position of operator's right hand (Figure 4a), it does not know that the resource (hand position) may have actually been acquired by combining two sub-resources (Figure 4b) – a camera and an algorithm that extracts information about hands from the camera feed. If any of the sub-resources fail, the action (and all mediating subsystems) must be notified about the event (*resource status update*) to make appropriate rearrangements.

Whereas Figure 4 illustrates only one specific example, in general all of the following could be true about a resource:

- The same resource could have been requested by different actions.
- The same action could request different resources.
- A resource could hierarchically depend on multiple layers of sub-resources.

Therefore, every component of TeMoto that provides and requests resources needs to maintain information about incoming and outgoing resource requests. RR was designed exactly for these purposes.

3.3 Implementation of Resource Registrar

The RR is the central component of every subsystem of TeMoto to provide and request resources. The RR is comprised of three main parts (Figure 5):

a) RR Servers process the incoming resource requests.
b) RR Clients mediate the outgoing resource requests.
c) Active Resource Registry keeps a record of all active in- and outbound resource queries.

a) Active Resource Registry

The Active Resource Registry (hereinafter referred to as registry) is needed for knowing the currently active/instantiated resources, e.g., a sensor which is in operational state. The resource is activated by the first request for the resource. Every resource entry in the registry contains information about the query (resource request and response) and information about the clients that are using this resource.

b) Resource Registrar Servers

A subsystem in TeMoto's Management Layer (Figure 2) could provide multiple resource types within the same subsystem. For example, the Context Manager subsystem provides information about the objects in the surrounding environment, and it also provides a service for tracking the objects – these are regarded as separate resource types.

Every resource type has its own dedicated callback function, similarly to the concept of native ROS services where incoming requests are handled inside a callback. The callback functions are registered in the RR and after registration, a dedicated RR Server is created (S_0 to S_m in Figure 5). The RR Server maintains information about unique (decided by a predefined metric) active resources inside the registry. Every time an RR Server receives a unique request, it:

1. directs the request to the callback,
2. retrieves a response from the callback,
3. adds a new resource entry about the query to the registry,
4. and returns the response to the client that initiated the query (incoming query in Figure 5).

If the request of the incoming query is not unique (i.e., it is already in the registry), then the response is retrieved from the registry without invoking the callback. The information about the new client is added to the corresponding resource entry.

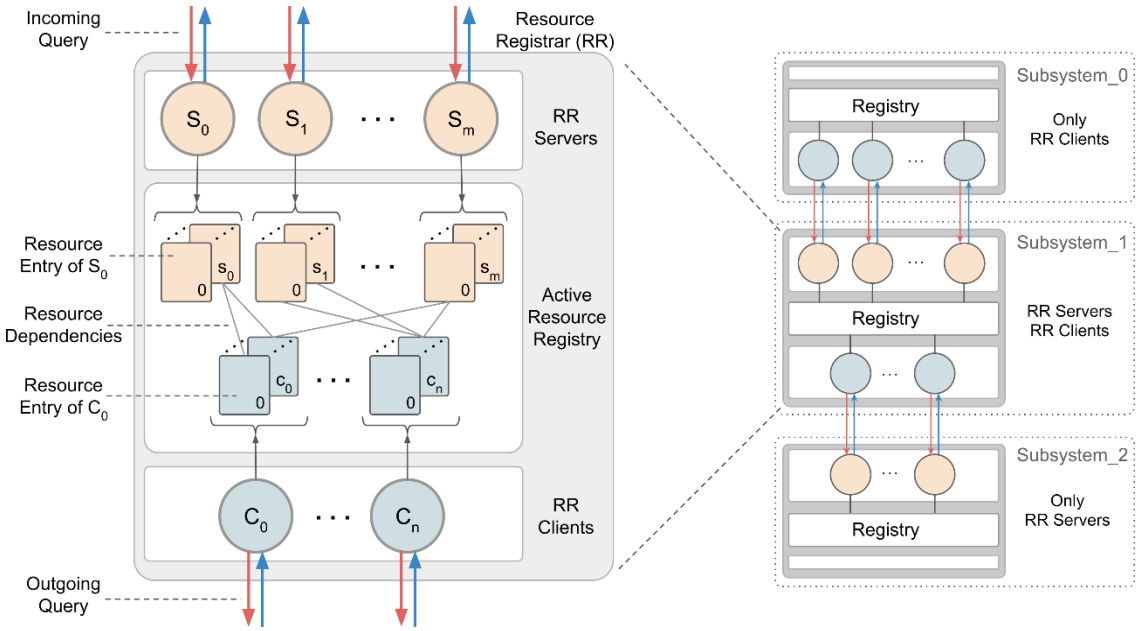


Figure 5: a) Structure of the Resource Registrar. b) Resource Registrar usage.
doi.org/10.5281/zenodo.1227113

c) Resource Registrar Clients

Analogously to RR Servers, every RR Client (C_0 to C_n in Figure 5) corresponds to a specific resource type. RR Clients are used for making resource queries to other RRs, which is again similar to the concept of native ROS clients where an instance of a client is used to manage the connection to a server. In order to mitigate the risk of circular dependencies, an RR does not support RR Client queries to the RR Servers within the same RR. Hence RR Clients are used only for requesting resources that are not managed by the same RR. Since every subsystem normally has one RR, the queries are targeted to the RRs of other subsystems (e.g., the flow shown in Figure 4).

The RR Client maintains information about unique active resource queries inside the registry. If the invoked query is not unique, then the response is retrieved from the registry without invoking the RR Client call procedure.

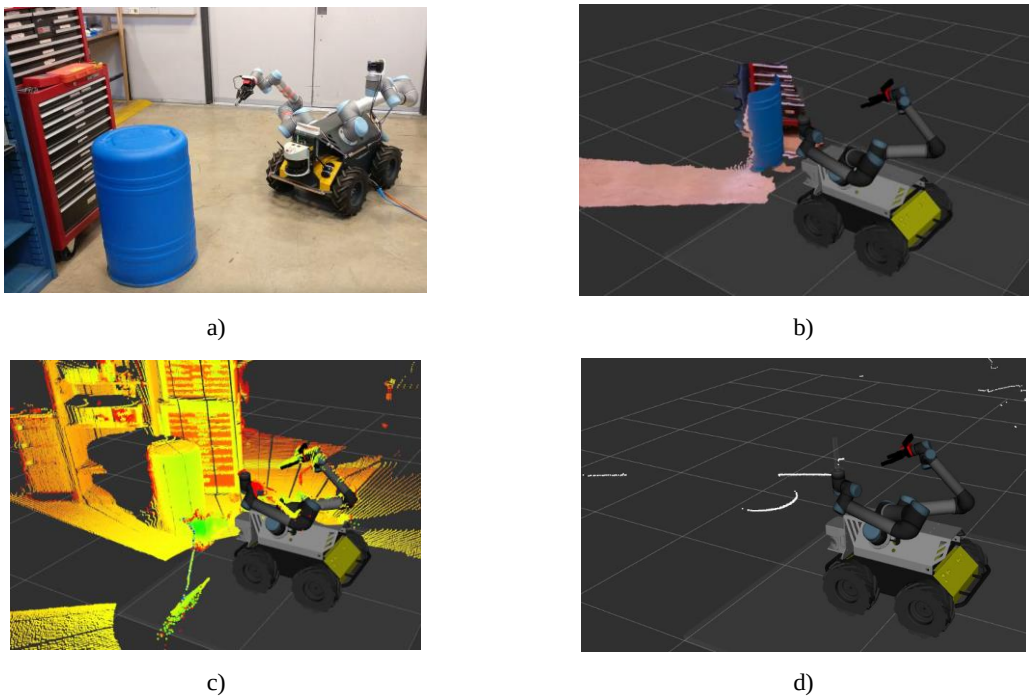
d) RR usage and resource dependencies

The RR contains both the RR Client and RR Server Application Programming Interfaces (API), which are utilized by TeMoto's subsystems based their functionality. For example, in Figure 5b, the depicted "Subsystem_0" only utilizes RR Clients because it is not providing any resources. Analogously, the "Subsystem_2" only hosts RR Servers. Most subsystems in the Management Layer (Figure 2) provide resources that depend on sub-resources ("Subsystem_1" in Figure 5b; Figure 4b, c), resulting in a resource dependency, illustrated in the Active Resource Registry in Figure 5.

The dependency is registered when the RR Client is invoked during RR Server callback procedure. Hence the dependencies are query-based and not fixed statically to a certain resource type. By knowing the resource dependencies, a deallocation of a top-level resource will automatically trigger the deallocation of dependent resources. It eliminates the issue of dangling resources. Additionally, the dependency allows forwarding resource status information to appropriate clients. For example, if the External Resource Manager in Figure 4 detects the failure of "cam_x", it will use the status info API of the RR to send a failure message to all related clients. Then the Sensor Manager will receive the status message and use the status info API to forward the message to all related clients. Finally, the action will receive the resource failure message from the Context Manager and decides on the next move (request a substitute resource, terminate, etc.).

4 DEMONSTRATION OF SENSOR REDUNDANCY

TeMoto's RR provides the core functionalities for the developer to design a fault-tolerant and resource-efficient robotic system. In order to evaluate the enabling capabilities of the RR, we devised a sample teleoperation scenario, where the critical part of the mission was to sustain continuous visual feedback about the surrounding environment in the presence of



a) b)

c) d)

Figure 6: A test scenario which demonstrates the core functionalities of RR. a) Image of the actual environment. b) Environment sensed via RGB-D camera. c) Environment sensed via arm-mounted spinning 2D LIDAR. d) Environment sensed via base-mounted 2D LIDAR.

cascading sensor failures. We demonstrate how resources can be dynamically and easily rearranged via the resource status propagation mechanism of the RR and TeMoto's Management Layer API.

4.1 Description of the Setup

A dual-arm mobile robot (NRG VaultBot [10]) was used for the demonstration. The robot has 2 Universal Robots UR5 manipulators mounted on the mobile Clearpath Husky base. The robot is equipped with a SICK LMS511 2D LIDAR (mounted to the front of Husky base), a rotating Hokuyo UTM-30LX 2D LIDAR (mounted to one UR5 arm), and Intel RealSense D435 RGB-D camera. The arm-mounted 2D LIDAR was used to create 3D pointclouds by spinning the LIDAR and stitching the laserscans [11]. A TeMoto action was implemented to maintain the goal of the mission by rearranging the resources every time a resource failure status message was received.

The scenario started with the operator giving an NL instruction – "*Show the environment*" – to TeMoto. As a result, the action with the following two functions was invoked.

1. The first function (invoked during the initial normal operation) requested the Sensor Manager to start a RGB-D camera and requested the Output Manager to show the pointcloud produced by the RGB-D camera on the screen.
2. The second function was invoked every time a resource failure message was received. It executed the same procedure as in the first function, but with a different sensor.

The sensor allocation hierarchy was defined from the perspective of human perception. An overlay of a video feed to a pointcloud gives potentially the best perception of objects around the robot. But even if the video link fails, a regular 3D pointcloud is helpful for an operator and gives a better understanding of the surroundings than a minimalistic 2D laserscan. Therefore, the sensors were prioritized as follows:

1. RGB-D camera – yielding 3D pointcloud with RGB overlay (Figure 6b)
2. Arm-mounted 2D LIDAR – yielding regular 3D pointcloud (Figure 6c)
3. Base-mounted 2D LIDAR – yielding 2D laserscan (Figure 6d)

During the demonstration, resource failures were physically induced by unplugging the sensor cables from the robot. First the RGB-D camera was unplugged, followed by unplugging of the arm-mounted 2D LIDAR.

4.2 Results

Figure 6a shows the Vaultbot in the demonstration environment. Figure 6b depicts the operator's perspective on the Mixed Reality (MR) scene during initial normal operation. Next, a depth camera failure

causes the TeMoto action to preserve the visual feedback in the MR scene via arm-mounted LIDAR (Figure 6c). After a resource failure was induced to the arm-mounted LIDAR, the TeMoto action invoked the base-mounted LIDAR to provide 2D laserscan representation of the surroundings (Figure 6d). The switching between different modes of visual representation and acquisition of depicted sensor feeds occurs seamlessly for the operator and is attributed to the RR.

5 DISCUSSION

The RR is a tool which is embedded in all subsystems of TeMoto. It helps to use resources efficiently and allows keeping track of complex sub-resource relations via the Active Resource Registry. Also, based on the information kept in the registry, the RR provides the capability to back propagate resource status information, e.g., device failures.

The next step in RR development is to increase the reliability and robustness of the RR by developing a P2P (peer-to-peer) protocol for keeping a copy of the registry in neighboring RRs (RRs which queried or provided resources for the given RR instance). Since the registry contains information about active resources and their dependencies, it is crucial to recover its state after a subsystem failure. Furthermore, the distributed approach for backing up the registry is potentially more robust to failures compared to a centralized system.

6 CONCLUSIONS

We demonstrated an intuitive "human-on-the-loop" teleoperation framework TeMoto and its Resource Registrar on a mobile manipulation platform where sensor failures were externally induced. A system that was responsible for visualizing robot's immediate surroundings for the operator was able to fulfill its objective by leveraging from RR to seamlessly switch between several sensor feeds as failures occurred.

TeMoto is growing to become a framework that facilitates integration of existing and future capabilities of other teleoperation software packages (e.g., predictive mechanisms during delay [4]). It provides tools for developers and ensures usability through modular approach and hardware agnosticism.

The independence from hardware has been demonstrated by using TeMoto on a wide range of robots, e.g., Clearpath Husky with two UR5 manipulators, KUKA youBot, and Yaskawa Motoman SIA5. Additionally, the practicality of TeMoto's MR scene visualization has been demonstrated on regular computer displays as well as with immersive virtual reality headsets [11], [12]. The herein presented concepts of resources and RR render TeMoto truly universal because any resource (sensor or algorithm) can be used interchangeably as long as it serves the main function.

Acknowledgements

This research was in part supported by Los Alamos National Laboratory and Estonian Centre of Excellence in IT (EXCITE) funded by the European Regional Development Fund. R.V. and K.K. acknowledge Dora Pluss, receiving funding from the European Regional Development Fund. V.V. would like to thank the Research Scholarship Program of Baltic-American Freedom Foundation.

References

[1] T. B. Sheridan, "Telerobotics," *Automatica*, vol. 25, no. 4, pp. 487–507, 1989.

[2] T. Fong, J. Rochlis Zumbado, N. Currie, A. Mishkin, and D. L. Akin, "Space Telerobotics: Unique Challenges to Human–Robot Collaboration in Space," *Rev. Hum. Factors Ergon.*, vol. 9, no. 1, pp. 6–56, 2013.

[3] W. Carey, P. Schoonejans, B. Hufenbach, K. Neergard, F. Bosquillon de Frescheville, J. Grenouilleau, A. Schiele, "METERON: A mission concept proposal for preparation of human-robotic exploration," presented at the Global Space Exploration Conference, Washington D.C., 2012.

[4] R. Burridge and S. Gee, "Supervising a humanoid robot across time delay," in *Proceedings of i-SAIRAS 2016: The 13th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2016.

[5] H. A. Yanco, A. Norton, W. Ober, D. Shane, A. Skinner, and J. Vice, "Analysis of human-robot interaction at the DARPA robotics challenge trials," *J. Field Robot.*, vol. 32, no. 3, pp. 420–444, 2015.

[6] R. Valner, K. Kruusamäe, and M. Pryor, "TeMoto: Intuitive Multi-Range Telerobotic System with Natural Gestural and Verbal Instruction Interface," *Robotics*, vol. 7, no. 1, p. 9, 2018.

[7] G. Visentin, M. van Winnendael, and P. Putz, "Advanced mechatronics in ESA's space robotics developments," in *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2001, pp. 1261–1266.

[8] K. Kruusamäe and M. Pryor, "High-precision telerobot with human-centered variable perspective and scalable gestural interface," in *Proceedings - 2016 9th International Conference on Human System Interactions (HSI 2016)*, 2016, pp. 190–196.

[9] R. Valner, V. Vunder, M. W. Pryor, and A. Zelenak, "TeMoto 2.0: Source Agnostic Command-to-Task Architecture Enabling Increased Autonomy in Remote Systems," in *Proceeding of the 2018 Waste Management*

Symposium, Phoenix, AZ, 2018.

[10] A. Sharp, K. Kruusamäe, B. Ebersole, and M. Pryor, "Semiautonomous Dual-Arm Mobile Manipulator System with Intuitive Supervisory User Interfaces," in *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2017.

[11] V. Vunder, R. Valner, C. McMahon, K. Kruusamäe, and M. Pryor, "Improved Situational Awareness in ROS using Panospheric Vision and Virtual Reality," in *2018 11th International Conference on Human System Interactions (HSI)*, 2018, submitted.

[12] A. Zelenak, V. Vunder, and M. Pryor, "Data-Driven Design of an Efficient User Interface for Mobile Manipulation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, submitted.