A YARP-Driven Generic Cartesian Control Architecture for Hardware-Agnostic Teleoperation in ROS 2 Ecosystems

Bartek Łukawski*, Edwin Daniel Oña, Juan G. Victores, Carlos Balaguer, Alberto Jardón
Departamento de Ingeniería de Sistemas y Automática, RoboticsLab
Universidad Carlos III de Madrid, ROR: 03ths8210
Avenida de la Universidad, 30 (edificio Sabatini), 28911, Leganés (Madrid), España
{blukawsk*, eona, jcgvicto, balaguer, ajardon}@ing.uc3m.es

***Abstract*—We present a Cartesian control architecture for teleoperation that abstracts hardware details by leveraging the open-source middleware framework YARP and interoperating with the ROS 2 ecosystem and its community tools and packages. The goal is to provide a flexible and modular network of distributed reusable components that can act as building blocks of a more complex and complete teleoperation application. In this way, users of our framework can extend these utilities to match their own applications, robots and needs. This work has been validated on the TEO humanoid, the ABB CRB 15000-5 "GoFa" collaborative robot, and the ASIBOT and AMOR assistive manipulators, as well as their corresponding virtual models for the Gazebo and OpenRAVE simulators. A variety of low-cost devices have also been explored: 3Dconnexion's SpaceMouse, Nintendo's Wiimote controller, and the Leap Motion hand tracker. All source code is publicly available on GitHub.**

***Index Terms*—software architecture, manipulator arm, collaborative robotics, teleoperation, middleware, ROS 2, YARP**

I. Introduction

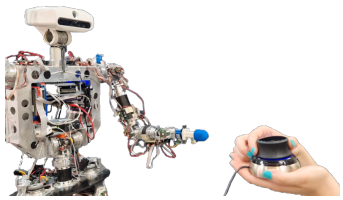
Teleoperation plays a significant role in the control of arm-like kinematic chains: besides the programmatic or kinesthetic teaching of target points, operators are usually presented with the option of commanding the robot through an external device, e.g., a joystick, preferably in the Cartesian space. Incidentally, teleoperation is also a means to showcase the physical and computational capabilities of the robot platform, necessitating a kinematic solver to perform the underlying calculations, and proper actuators and sensors to apply motion on each axis with the required precision. Most vendors provide their own proprietary software and hardware to achieve this. Therefore, their control schemes and tooling are usually bound to a manufacturer or even a specific model.

The goal of this work is to build an open-source, generic architecture that can be easily extended to arbitrary hardware platforms, providing developers the necessary building blocks to reuse or extend them according to individual requirements. We also build upon our existing codebase for teleoperation tasks across the available platforms, as detailed next. Through the integration of varied software components, we seek to develop practical demonstrators and tools that could be used, for instance, in academia, focusing on practical robotics sessions with undergraduate and graduate students.

II. Background

This work expands on previous developments by the RoboticsLab team, as shown in Figure 1. A generic teleoperation framework was first proposed and implemented leveraging the YARP middleware: it was demonstrated that a range of robot architectures can be teleoperated through a collection of low-cost devices and open-source software [1]. This was later extended to the ROS 2 ecosystem, intercommunicating our codebase with new ROS 2-based applications. A teleoperation application was built entirely with ROS 2 and tested on the TEO (1a) and AMOR (1c) robots, while a React-based web interface was designed for enabling easy joint and Cartesian control of TEO via desktop and handheld devices (1b) [2]. In a recent work, a similar interface was implemented on the ABB GoFa robot, leveraging our EGM ROS 2 package [3].

Other works explored haptic feedback (1d) [4], included low-cost webcams to implement visual servoing in a shared control scenario [5], or adopted virtual reality headsets in telepresence contexts for bimanual mobile base robots [6].



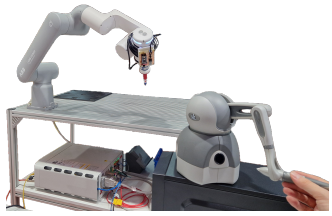
(a) Teleoperation of TEO with a 3D mouse [2].



(b) ROS 2 and React-based web interface for TEO [2].



(c) Teleoperation of AMOR with a 3D mouse [2].



(d) Teleoperation of ABB GoFa with haptic feedback [4].

Fig. 1: Relevant previous works by RoboticsLab UC3M.

Remote robot control and efficient teleoperation frameworks have been tackled by many research teams over the years. An example can be found in the work by Settimi et al., introducing a modular interface for humanoid robot control in urban search and rescue (USAR) scenarios [7]. This project combined both YARP and ROS middleware frameworks: YARP was intended for communicating low-level hardware and control modules, whereas ROS was used to retrieve sensor data and display robot status via RViz. The controller itself relied on motion description languages (MDL) to express basic intent via motion primitives and behaviors in three levels, ranging from human-like actions, through Cartesian space commands, to joint space motion: traded, shared and direct control. The operator is presented with a 3D reconstruction of the environment in which virtual objects can be interacted with.

Yang et al. proposed another YARP-based framework focusing on body motion tracking via the Kinect camera to command the dual-arm Baxter robot [8]. An alternative teleoperation scenario was also explored by the authors to control the iCub humanoid with a haptic device. The retargeting problem was addressed in a twofold manner: either via a vector method (mapping the angles of human and robot joints), or using inverse kinematics (obtaining joint angles from the end-effector pose). MATLAB and the Robotics Toolbox were used alongside iCub's YARP-based simulator, taking advantage of the YARP-MATLAB integration.

An example of a complex platform for which teleoperation was accomplished is the Centauro robot described by Klamt et al. [9]. It features a centaur-like body with four articulated wheeled legs, and an upper body with arms and dexterous hands. An operator interface was developed to remotely control robot parts through a telepresence suit, accounting for both locomotion and manipulation. A specialized control framework was developed, including YARP support for low-level dynamics, and ROS for general communication via topics.

In terms of generic and reusable teleoperation frameworks, the ROS End-Effector project by Torielli et al. tackled a problem similar to the one addressed in this work [10]. Their ROS package provided an open-source system capable of accommodating a wide range of commercial robotic end-effectors, with different grasping capabilities. Through high-level grasping primitives, the user is provided with hardware abstraction layers, programming interfaces, simulation tools and user interfaces to efficiently command the chosen tool. Although published as a ROS package and later migrated from ROS 1 to ROS 2, the project seems to have stagnated.

Our work targets the necessities of integrators and application developers by providing interfaces and tools for extending our software control blocks or building new ones. YARP is leveraged as a robotics framework that stands slightly closer to the actual hardware than ROS, whereas this latter middleware is currently the de facto standard across the robotics community. We provide the necessary bridges to allow for efficient interoperation. While past works relied on interface definition language (IDL) techniques [11], ROS 2 enables more efficient interfacing through its client library (RCL).

III. Materials and Tools

A. Robot Platforms

The versatility and flexibility of the teleoperation framework is demonstrated through the variety of robot platforms it supports. The experiments were conducted on a collection of manipulators available to the RoboticsLab UC3M team, as depicted in Figure 2. The full-size humanoid robot TEO (2a) built by RoboticsLab UC3M [12] serves as the basis for single and bimanual teleoperation. It features 28 degrees of freedom (DoF) distributed along its legs (6 each), arms (6 each), torso and neck (2 each). On the other hand, the commercial ABB CRB 15000-5 "GoFa" is a 6 DoF collaborative industrial manipulator arm with intrinsic safety capabilities and equipped with custom tools developed by our team (2b). The former platform implements low-level YARP controllers, whereas the latter features our novel ROS 2 motion control and communications package.

Two assistive manipulator arms were introduced in the experiments: ASIBOT, a 5-DoF symmetric climbing and portable robot built by RoboticsLab (2c), and AMOR, a commercial 7-DoF assistive arm (2d). Both platforms implement YARP-based controllers and have been deemed suitable for teleoperation tasks, specifically in the field of assistive robotics.



(a) TEO humanoid robot (b) ABB CRB 15000-5



(c) ASIBOT assistive arm (d) AMOR assistive arm

Fig. 2: Robot platforms.

Simulation has been used extensively to properly test the algorithms and controllers prior to their adoption on the real hardware. Figure 3 depicts the virtual model counterparts of the previous robot platforms, corresponding to the Gazebo, OpenRAVE and ABB RobotStudio simulators. The underlying software architecture backed by YARP enables testing the same controllers regardless of the reality (real or simulated robot), thus facilitating their development and integration.

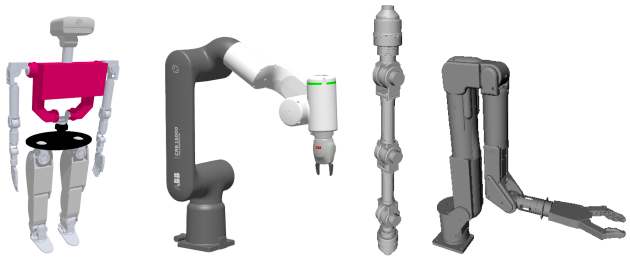


Fig. 3: Simulated counterparts of the robot platforms.

B. Teleoperation Devices

A selection of low-cost devices with available open-source toolkits for developers has been integrated in the teleoperation scenarios following a previous study, as depicted in Figure 4 [1]. Their capabilities determine the mode of operation imposed by our framework. For instance, the SpaceMouse 6-DoF 3D mouse by 3Dconnexion (4a) is suitable for velocity-based control: any displacement of the joystick is translated into a velocity of the tool center point (TCP) of the robot to be controlled, with three axes devoted for translation and the other three for rotation. This allows for precise control that depends on the operator for visual feedback and adjustments of the desired end effector pose. On the other hand, the Wiimote controller (4b) only offers three rotation axes based on its inertial sensor, therefore two buttons are programmed to enable forward or backwards motion depending on their state. Finally, the Leap Motion (4c) infrared sensor and hand tracking device is devoted to position-based control. The position and orientation of the palm is scaled if necessary and mapped to the pose of the TCP.

The 3D Systems Phantom Touch device (4d) is considered for future enhancements involving haptic feedback.



(a) 3Dconnexion SpaceMouse (b) Nintendo Wiimote

(c) Leap Motion (d) 3D Systems Phantom Touch

Fig. 4: Teleoperation devices.

IV. Proposed Architecture

The proposed software architecture is depicted in Figure 5. It relies on the YARP [13] and ROS 2 [14] middleware frameworks as the main orchestrators of the teleoperation task. Earlier works leveraged YARP's capabilities to introduce a distributed architecture of modular components, thus avoiding a monolithic control scheme that is hard to maintain and extend [1]. For instance, low-level joint and Cartesian control was built with reusability at its core, ultimately enabling the utilization of our architecture on multiple robots, realities (real hardware or simulated) or control devices. A thin software layer named "network communication server" (NWS) interfaces with the underlying controller implementations to expose their programming interfaces across the middleware network, adopting the remote procedure call (RPC) paradigm and streaming communication for state inspection and motion commands. Thanks to this, the middleware can be abstracted out, as demonstrated in a previous work that enabled the compatibility of our architecture with the ROS 2 ecosystem [2].

On the client side, a software block named "network wrapper client" (NWC) interfaces through the corresponding communication mechanisms with the NWS. In YARP ecosystems, the NWC-NWS pair communicates through YARP ports. In ROS 2, they leverage its core functionalities: topic publishers and subscribers, services and clients, action servers, and parameter servers. High-level applications either instantiate a NWC or interface directly through these communication channels, although the former option is preferable for the ease of application development.

The proposed architecture provides flexibility and compatibility with existing application layers. For instance, to communicate with a YARP-based robot, an application that was devised for ROS 2 ecosystems can interoperate through the dedicated NWS. In this work, we present the opposite approach: a high-level YARP-based application can communicate with its ROS 2 NWC component with a ROS 2-based controller server. To illustrate this, we also developed a ROS 2 package based on the ABB Externally Guided Motion (EGM) feature, which externalizes robot control and state monitoring with high-frequency (250 Hz) control rates [15].

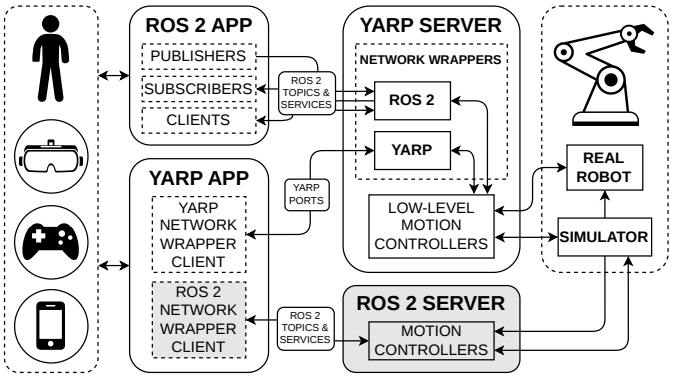


Fig. 5: Proposed software architecture. New components introduced in this work have been highlighted.

V. IMPLEMENTATION

A. Cartesian Control Interface

The Cartesian control interface "ICartesianControl", exposed by the relevant NWS software blocks, has been iterated upon, rendering as shown in the class diagram of Figure 6, encoded using the Unified Modeling Language (UML).

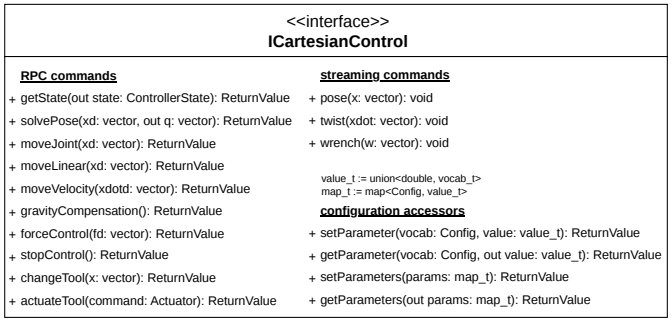


Fig. 6: Unified Modeling Language (UML) class diagram.

The pure virtual methods of this abstract C++ class (or interface) can be split into three groups: RPC commands, streaming commands, and configuration accessors. RPC commands are meant for bidirectional, one-to-one communication based on a request and a response, or a command and the subsequent acknowledge. For this reason, these are not suitable for high-frequency communication, but allow querying the robot state and issue point-to-point motion commands, for instance.

- getState: query the internal controller state (pose, mode, last timestamp, duration and progress of the current trajectory, completion flag).
- solvePose: solve inverse kinematics for the provided pose.
- moveJoint: move to the desired pose without applying geometric constraints (isochronous joint trajectories).
- moveLinear: move to the desired pose following a straight line and applying a velocity profile.
- moveVelocity: apply the desired twist.
- gravityCompensation: enable gravity compensation.
- forceControl: exert the desired Cartesian force on the environment using the robot's TCP.
- stopControl: cancel the current motion.
- changeTool: replace the last transformation of the kinematic chain, i.e., post-multiply it with a fixed homogeneous transformation matrix.
- actuateTool: enable or disable tool actuation, if supported.

Concrete instantiations of this interface can communicate with joint controllers via dedicated NWC/NWS pairs, and rely on additional software modules to solve kinematic equations.

The interface also features high-frequency, one-way streaming commands that don't expect any response, depending on the physical magnitude to be commanded: pose, twist and wrench. Lastly, a pair of setter and getter methods allow to access controller configuration internals (individually or in batches) such as gain, trajectory duration, reference speed and acceleration of the velocity profiles, period of the control cycle, and the reference frame to interpret commands in.

B. ROS 2 Command Mappings

In order to expose "ICartesianControl" over the ROS 2 network, mappings between these methods and ROS 2 communication utilities needed to be established. Table I summarizes the equivalent messages or service for each command. Standard ROS types are used when possible, otherwise custom types are introduced.

TABLE I: *ICartesianControl* methods mapped to ROS 2 message types. *rl_msgs* is short for *rl_cartesian_control_msgs*.

interface method	ROS 2 message or service	YARP usage
getState	geometry_msgs/msg/PoseStamped	RPC service: request + response or command + acknowledge
solvePose	rl_msgs/srv/SolvePose	
moveJoint	*rl_msgs/action/PoseTrajectory*	
moveLinear		
moveVelocity	rl_msgs/srv/MoveVelocity	
gravityCompensation	std_srvs/srv/Trigger	
forceControl	rl_msgs/srv/ForceControl	
stopControl	std_srvs/srv/Trigger	
changeTool	rl_msgs/srv/ChangeTool	
actuateTool	rl_msgs/srv/ActuateTool	
pose	geometry_msgs/msg/Pose	streaming commands (one-way)
twist	geometry_msgs/msg/Twist	
wrench	geometry_msgs/msg/Wrench	

There is an almost exact mapping between YARP RPC commands and ROS 2 services, and YARP streaming commands and ROS 2 topic subscribers. The former take advantage of services as they better suit the low-frequency request-response paradigm, whereas the latter rely on the 1-to-N high-frequency nature of topics. As a sole exception, the "moveJoint" and "moveLinear" commands have been mapped to a ROS 2 action server in order to provide feedback during the execution of the trajectory, and to signal the moment it has been completed.

In addition to the aforementioned methods, YARP configuration accessors have been exposed through the ROS 2 parameter server.

C. Teleoperation Control Scheme Overview

The proposed architecture has been materialized in a generic teleoperation application that aims to adhere to the software and hardware setup at hand: any robot and reality, any device, any middleware. The constraints imposed on the version presented in this work are encompassed in the materials described earlier and the two adopted middleware frameworks: YARP and ROS 2. However, we provide the guidelines to extend this scheme to other ecosystems, taking advantage of the flexibility and modularity of this approach.

Figure 7 describes the component diagram of the proposed application, which is composed of the following main blocks:

1) A high-level native ROS 2 application that communicates with the YARP-based Cartesian controller through a thin ROS 2 NWS layer and standard ROS 2 channels (topics, services, parameters). This application adopts the solutions that best fit the ROS 2 ecosystem, such as community-developed packages for data acquisition from teleoperation devices like the SpaceMouse.

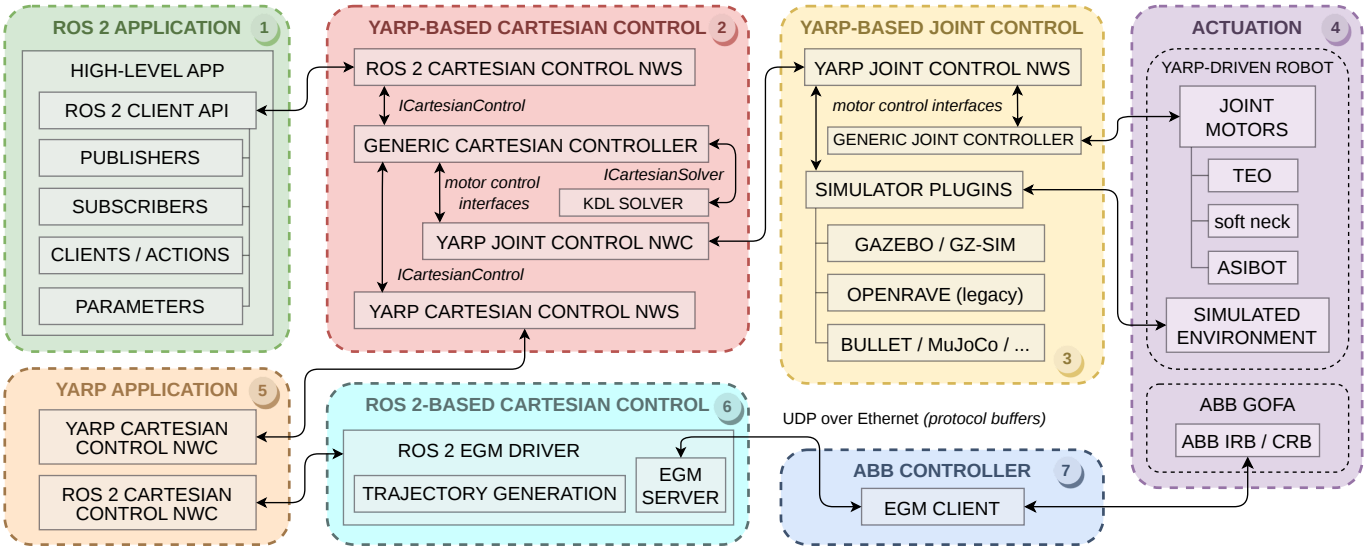


Fig. 7: Component diagram of the teleoperation application on YARP and ROS 2-based robot platforms.

2) The YARP-based Cartesian control block, which exposes the underlying kinematic solvers and controller implementations through the dedicated YARP and ROS 2 NWS layers. In turn, it interfaces with the low-level joint controllers through their relevant NWC layers.
3) The YARP-based joint control block, particularized for the specific robot hardware to interface with, including plugins for simulation.
4) An actuation layer composed of the actual robot hardware and simulators. In this work, ROS 2 based robots, i.e. ABB manipulators, are introduced.
5) A high-level native YARP application, which may implement a multiplexer to conditionally communicate with either a YARP- or ROS 2 based Cartesian controller, through the YARP NWC or dedicated ROS 2 channels, respectively. We have devised the latter as an addition in this work to interface with the EGM-based ROS 2 controller for ABB robots.
6) The ROS 2-based Cartesian controller that understands the low-level details of the robot. Our EGM-based driver is described in the next section. Actual communication with the hardware is performed by the proprietary EGM client-server pair. Messages are encoded using Google's protocol buffers (protobuf) and sent via UDP packets.
7) The proprietary ABB controller that forwards motion commands and state feedback to and from the hardware.

The diagram illustrates the server-client paradigm implemented with NWS-NWC components. Each pair abstracts out the protocols and channels understood by YARP and ROS 2, thus hiding these details from the programmer and routing the program flow through programming interfaces. The ROS 2 NWC, besides pairing with its NWS counterpart, can also connect to native ROS 2 channels in compatible Cartesian servers that do not implement the YARP-oriented NWS-NWC paradigm, such as our EGM-based package.

D. ROS 2 Package for ABB EGM

To integrate ABB robots, specifically OmniCore-compatible collaborative CRB "GoFa" models, in our control scheme, we developed a ROS 2 package that interfaces with the proprietary robot controller through EGM using an open-source dependency[1]. It implements the Cartesian control commands described earlier, thus allowing it to be integrated in the teleoperation architecture, boosting component reuse.

The main node implements all three EGM modes: position stream (for publishing robot state), position guidance (for following a trajectory whose setpoints originate from an external device), and path correction (for performing fine adjustments along the path coordinates based on external information such as a sensor). Position stream is always active, and it publishes live joint configuration and TCP pose. Position guidance and path correction must be preconfigured in the RAPID template program executing on the robot controller. The former accepts two command domains depending on the coordinate to be attained and must also be configured via RAPID: "joint" and "pose". However, an internal solver has been implemented to account for the necessary transformations between joint and Cartesian space; therefore, as long as the information about the robot's kinematic chain is provided during node initialization, the "pose" command subset is enhanced with "joint" commands.

Table II lists the communication channels and their message types that map to a selection of supported "ICartesianControl" methods in the controller's "pose" mode. Italics denote commands that require the kinematic chain to be described during initialization. In addition to these channels, the main node also supports configuration parameters through the ROS 2 parameter server. Most importantly, a smooth factor is provided to support an exponential moving average (EMA) low-pass filter.

[1]https://github.com/FLo-ABB/ABB-EGM-Python

TABLE II: ROS 2 commands exposed by the abb_egm_driver package. *rl_msgs* is short for *rl_cartesian_control_msgs*.

topic name	ROS 2 message or service	usage
/state/pose /state/joint /state/data	geometry_msgs/msg/PoseStamped sensor_msgs/msg/JointState std_msgs/msg/Float64MultiArray	state feedback (topic publishers)
/command/pose */command/joint* /command/data	geometry_msgs/msg/Pose std_msgs/msg/Float32MultiArray std_msgs/msg/Float64MultiArray	streaming commands (topic subscriptions)
/actuate_tool /stop_control */solve_pose*	rl_msgs/srv/ActuateTool std_srvs/srv/Trigger rl_msgs/srv/SolvePose	RPC commands (service calls)
/trajectory/joint /trajectory/pose	rl_msgs/action/JointTrajectory rl_msgs/action/PoseTrajectory	trajectory commands (action servers)

VI. Experiments and Results

The teleoperation application was first validated on the ABB RobotStudio simulator with the "GoFa" robot model, and subsequently tested on the real robot, as shown in Figure 8, using the SpaceMouse device and a sequence of velocity-controlled displacements. It has been observed that communication is stable and latency is low, with the 4 ms command period providing a swift response from the robot upon any displacements applied to the 3D mouse. Smoothness and speed are adjusted through configuration parameters to fit the requirements of the task.



Fig. 8: Sequence of a teleoperation task carried out on the ABB "GoFa" robot with a SpaceMouse device.

VII. Conclusions

This work introduces a generic teleoperation architecture based on YARP and ROS 2. It was devised to leverage the flexibility and modularity of these middleware frameworks to facilitate the design of a distributed network of reusable software components. Thanks to this, existing blocks (e.g., joint or Cartesian controllers, kinematic solvers, data acquisition, etc.) can fit a variety of robots, realities (real hardware or virtual models) and teleoperation devices, as demonstrated in this work. All code has been published on the GitHub platform.[2]

Expanding on a preliminary work [4], haptic devices are planned for future projects to combine the existing teleoperation commands with live force-torque feedback [16].

[2]Public account: https://github.com/roboticslab-uc3m; relevant repositories: "kinematics-dynamics" (kinematics and dynamics solvers and applications), "teleop-yarp-devices" (YARP software components devoted to teleoperation), "rl_cartesian_controllers" (ROS 2 packages for Cartesian control and teleoperation), "abb_egm_driver" (EGM-based ROS 2 driver).

Acknowledgments

This research has been financed by "iRoboCity2030-CM, Robótica Inteligente para Ciudades Sostenibles" (TEC-2024/TEC-62) and "FotoArt5.0-CM, Laboratorios inteligentes para la ciencia del futuro" (TEC-2024/TEC-308), both funded by "Programas de Actividades I+D en Tecnologías de la Comunidad de Madrid"; "ROBOASSET: Intelligent robotic systems for assessment and rehabilitation in upper limb therapies" (PID2020-113508RB-I00, financed by AEI/10.13039/501100011033); "iREHAB: AI-powered Robotic Personalized Rehabilitation" (DTS22/00105, financed by Instituto de Salud Carlos III (ISCIII)); and EU structural funds.

References

[1] B. Łukawski, J. G. Victores, and C. Balaguer, "A generic controller for teleoperation on robotic manipulators using low-cost devices," Jornadas de Automática, CEA, 44, 2023. DOI: 10.17979/spudc.9788497498609.785

[2] B. Łukawski, M. Rebollo, Á. Gilabert, J. G. Victores, C. Balaguer, and A. Jardón, "YARP Cartesian controller layers over ROS 2 for teleoperation and web applications," Jornadas de Automática, CEA, 46, 2025. DOI: 10.17979/ja-cea.2025.46.12252

[3] B. Łukawski, G. Hovhannisyan, L. M. Reques, I. Ureta, J. G. Victores, and A. Jardón, "Integrating YARP and ROS 2 for web-based and embedded robot GUIs," Jornadas de Automática, CEA, 47, 2026. DOI: 10.17979/ja-cea.2026.47.13858

[4] Á. Santos-García, B. Łukawski, D. Copaci, J. G. Victores, and C. Balaguer, "Bilateral haptic teleoperation bridging YARP and ROS 2 ecosystems," Simposios del Comité Español de Automática, CEA, 2(2), 2026. DOI: 10.64117/simposioscea.v2i2.216

[5] E. D. Oña, B. Łukawski, A. Jardón, and C. Balaguer, "A modular framework to facilitate the control of an assistive robotic arm using visual servoing and proximity sensing," Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC), IEEE, 2020. DOI: 10.1109/ICARSC49921.2020.9096146

[6] B. Łukawski, I. Montesino, E. D. Oña, J. G. Victores, C. Balaguer, and A. Jardón, "Towards the development of telepresence applications with TIAGo and TIAGo++ using a virtual reality headset," Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC), IEEE, 2025. DOI: 10.1109/ICARSC65809.2025.10970173

[7] A. Settimi et al., "A modular approach for remote operation of humanoid robots in search and rescue scenarios," Int. Workshop on Modelling and Simulation for Autonomous Systems (MESAS), Springer, 2014, pp. 192–205. DOI: 10.1007/978-3-319-13823-7_18

[8] C. Yang, H. Ma, and M. Fu, "Robot teleoperation technologies," Advanced Technologies in Modern Robotic Applications, Springer, 2016, pp. 187–229. DOI: 10.1007/978-981-10-0830-6_6

[9] T. Klamt et al., "Remote mobile manipulation with the centauro robot: Full-body telepresence and autonomous operator assistance," J. of Field Robotics, 37(5), 2020, pp. 889–919. DOI: 10.1002/rob.21895

[10] D. Torielli, L. Bertoni, F. Fusaro, N. Tsagarakis, and L. Muratore, "ROS end-effector: A hardware-agnostic software and control framework for robotic end-effectors," J. of Intelligent and Robotic Systems, 108(4), 2023, p. 70. DOI: 10.1007/s10846-023-01911-5

[11] M. Aragão, P. Moreno, and A. Bernardino, "Middleware interoperability for robotics: A ROS–YARP framework," Frontiers in Robotics and AI, 3(64), 2016, p. 64. DOI: 10.3389/frobt.2016.00064

[12] C. Pérez Martínez, P. Pierro, S. Martínez, L. Pabón, M. Arbulú, and C. Balaguer, "RH-2: an upgraded full-size humanoid platform," Mobile Robotics, World Scientific, 2009, pp. 471–478. DOI: 10.1142/9789814291279_0058

[13] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet Another Robot Platform," Int. Journal of Advanced Robotic Systems, 3(1), 8, 2006, pp. 43–48. DOI: 10.5772/5761

[14] M. Quigley et al., "ROS: an open-source Robot Operating System," ICRA workshop on open source software, 3, 2009. Corpus ID: 6324125

[15] B. Łukawski, E. D. Oña, A. Jardón, J. G. Victores, and C. Balaguer, "Development of educational applications with ABB GoFa collaborative robot using Externally Guided Motion," Int. Conf. on Control, Automation and Diagnosis (ICCAD), IEEE, 2025. DOI: 10.1109/ICCAD64771.2025.11099395

[16] B. Łukawski, A. Olano-Díaz, A. González-Villasante, E. D. Oña, J. G. Victores, and A. Jardón, "External force-torque sensor feedback on the ABB GoFa collaborative robot," Simposios del Comité Español de Automática, CEA, 2(2), 2026. DOI: 10.64117/simposioscea.v2i2.215