CEA Comité Español de Automática

XLVI JORNADAS AUTOMÁTICA CARTAGENA 2025

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

YARP Cartesian controller layers over ROS 2 for teleoperation and web applications

Bartek Łukawski*, Mercedes Rebollo, Ángel Gilabert, Juan G. Victores, Carlos Balaguer, Alberto Jardón

RoboticsLab, Department of Systems Engineering and Automation, Universidad Carlos III de Madrid, Avda. Universidad, 30, 28911, Leganés, Spain.

To cite this article: Bartek Łukawski, Mercedes Rebollo, Ángel Gilabert, Juan G. Victores, Carlos Balaguer, Alberto Jardón. 2025. YARP Cartesian controller layers over ROS 2 for teleoperation and web applications. Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12252

Resumen

Nuestros trabajos previos introdujeron una arquitectura de teleoperación para brazos robóticos orquestada por el entorno de trabajo e intermediario YARP. Un esquema distribuido de componentes software facilitó la adopción de nuevos periféricos para teleoperación, explorando múltiples modos de control. En este trabajo, proponemos un nexo entre los elementos de YARP ya disponibles y el ecosistema de paquetes y herramientas de Robot Operating System (ROS). Nuestro controlador cartesiano y sus interfaces C++ han servido como base para la nueva implementación, exponiendo comandos del robot y su configuración 3D en la red de ROS 2. Esto ha sido materializado en forma de una aplicación de teleoperación para el ratón 3D SpaceMouse de bajo coste, y una aplicación web construida con la librería React en JavaScript. Ambos componentes han sido probados y validados sobre robots reales y en simulación: sobre la plataforma robótica humanoide TEO y el brazo asistencial AMOR.

Palabras clave: robots humanoides, manipuladores robóticos, teleoperación, interfaces humano-robot, interfaces web.

Abstract

Our previous works elaborated on a teleoperation architecture for robot arm manipulators orchestrated by the YARP robotics framework and middleware. A distributed layout of software components eased the adoption of new peripherals for teleoperation, and different control modes were explored. In this work, we propose a bridge between existing YARP elements and the Robot Operating System (ROS) ecosystem of packages and tools. Our custom Cartesian controller and C++ interfaces underpinned the new implementation, exposing robot commands and 3D configuration over the ROS 2 network. It took the form of a teleoperation app using the low-cost SpaceMouse 3D joystick, and a web app built with the React JavaScript library. Both have been tested and validated in real robots and simulation: on the TEO humanoid robot platform and the AMOR assistive robotic arm.

Keywords: humanoid robots, robot manipulators, teleoperation, human-robot interfaces, web interfaces.

1. Introduction

Teleoperation in robotics, or telerobotics, has been defined as an "extension of human sensing and manipulating capability by coupling through communication means to artificial sensors and actuators" (Sheridan, 1989). It can be also perceived as a basic mode of operation. As a matter of fact, connecting an external peripheral (such as a joystick) with a robot control system poses a direct and intuitive way to validate the robot's motion capabilities and perform basic tasks such as pick-and-place demonstrations.

In the context of this work, a controller architecture has been developed in recent years to provide a teleoperation framework with a generic approach: compatibility is sought across multiple robot platforms (real or simulated), using varied peripherals. It has been implemented using YARP, a robotics framework comparable to the more widely adopted ROS 2. This work migrates the existing YARP architecture to ROS 2 to take advantage of the codebase and community-developed tools of the latter. An additional tool, a web-based command and diagnostics interface, is also presented here to showcase another use-case resulting from said migration.

*Corresponding author: blukawsk@ing.uc3m.es

This document is structured as follows. Section 2 provides a background on the main related technical topics. Section 3 describes the hardware and software tools, expanding on the implementation of the latter in Section 4. The applications are validated on different robot platforms through experiments covered in Section 5, and conclusions are drawn in Section 6.

2. Background

Teleoperation tasks in robotics require the intervention of a human operator; although certain degree of autonomy can be achieved, these systems are never fully autonomous. The person relies on visual feedback, either direct (via eye contact) or through cameras, to remotely guide the robot in a supervised manner. Supervision can be partial in case the robot system aids the user in completing the task, through negotiated or shared control strategies. Since the first master-slave manipulators in the 1940s, teleoperated robots have succeeded in several fields which, for instance, require precise motions e.g. in surgical contexts (telesurgery), or entail any hazard to humans like the handling of nuclear materials, in addition to space and underwater exploration. Recent years have seen the introduction of virtual and mixed reality, in which simulated robots require some sort of human control. In parallel, telepresence applications help operators project themselves through the robot, thus feeling like being present at the teleoperation site where the robot is located (Lichiardopol, 2007).

Teleoperation has been adopted in multiple fields of robotics. Some studies categorized this area in the topic of mobile robotic platforms in the following manner (Moniruzzaman et al., 2022): direct teleoperation of unmanned ground (UGV) and aerial (UAV) vehicles and remotely operated vehicles (ROV), relying on visual feedback transmitted from the mobile robot; supervisory teleoperation in which control duty is shared between the operator and the system; and multimodal teleoperation with the aid of multiple sensor interfaces to increase situation awareness. In contrast, humanoid teleoperation poses an additional challenge (Darvish et al., 2023): it needs to account for human kinematics, dynamics and physiology in order to adapt the robot's higher versatility to the available motion control and feedback devices, e.g. depth and Light Detection and Ranging (LIDAR) sensors, haptic and tactile sensors, electroencelography (EEG) sensors, etc.

Other studies delved into bilateral and multilateral control systems to analyze teleoperation scenarios beyond the communication between the operator and the robot (Shahbazi et al., 2018): multiple operators and/or multiple robots. In an assistive context, a shared control system was developed combining an eye-in-hand visual servoing controller with proximity sensors (Oña et al., 2020).

Some works explored the ROS framework and its ecosystem. For instance, the SpaceMouse device used in this work was previously adopted in the TIAGo service mobile base (Calzada et al., 2024). Regarding user interfaces, a link was established between ROS robot description files and interface design in teleoperation contexts (Mortimer et al., 2017). Finally, a study can be cited in which three teleoperation modalities have been implemented on the JAKA Minicobo arm and compared: graphical user interfaces, hand gestures and a virtual reality controller (Chen et al., 2023).

3. Robot Platforms and Tools

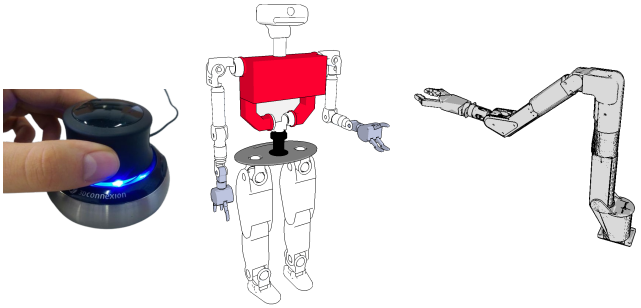
3.1. Robots

The proposed teleoperation application has been tested using the SpaceMouse by 3Dconnexion, a low cost peripheral aimed at computer-assisted design (CAD), depicted in Figure 1(a). This joystick allows six degrees-of-freedom (DoF) distributed as three axes of translation and three axes of rotation, thus mapping to the exact number of axes that would allow to translate and rotate the tool center point (TCP) of a standard robotic manipulator arm.

All experiments have been done in simulation first, using the OpenRAVE and Gazebo (classic and modern) open source simulators. Two robot models have been considered: TEO and AMOR, shown in Figures 1(b) and 1(c), respectively.

TEO is a full-sized humanoid robot platform developed in Universidad Carlos III de Madrid (Pérez Martínez et al., 2010). Comprised of 28 DoF (six per limb, two on the torso, two on the neck) and modular grippers, along with sensing technologies such as force-torque and inertial sensors, color and depth (RGBD) cameras, and a microphone and speaker, it allowed to explore a variety of applications: single and dual manipulation, sensor fusion, stable gait, or sign language, among others. In this work, the teleoperation control scheme can be applied on any limb, although more specifically both of its 6 DoF arms are targeted.

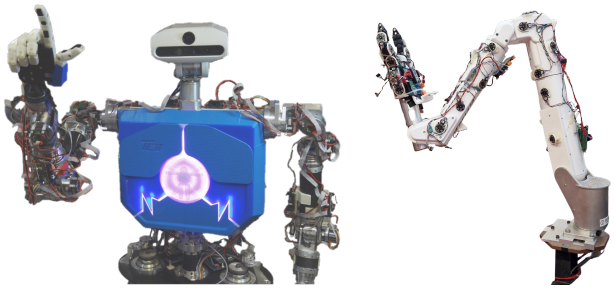
Besides, the commercial AMOR 7 DoF assistive robotic arm was also available for testing and therefore has been used to validate the application on a slightly different arm topology.



(a) SpaceMouse (b) TEO humanoid (c) AMOR arm

Figure 1: 3D mouse peripheral and OpenRAVE robot models.

Figure 2 illustrates the real robots on which the application was launched after an initial validation was completed on their simulated counterparts.



(a) TEO humanoid (b) AMOR assistive robot

Figure 2: Real robot platforms.

3.2. YARP and ROS 2 Frameworks and Middlewares

On the software side, YARP (Yet Another Robot Platform) is used extensively in the low-level implementation and high-level connectivity layers. This framework empowers the development of single-responsibility components that fit in a distributed architecture, rendering it convenient for building robotic applications (Metta et al., 2006). As it is written in C++, developers can implement their own building blocks for motor control, sensing (e.g. through force-torque, inertial, and camera sensors), audio broadcast, kinematic operations, etc., using a wide collection of C++ interfaces or introducing their own. These blocks are consolidated as dynamically loaded libraries (plugins), also called "devices" in the YARP jargon. It is standard to encapsulate the low-level code, having direct access to the hardware in one device, then expose its interface over the YARP network via pairs of devices denoted as "network wrapper server" (NWS) and "network wrapper client" (NWC), respectively. The middleware capabilities of YARP are leveraged to handle the communications between components across a local network through a range of inter-process communications (IPC) techniques involving system ports.

On the other hand, ROS (and its modern iteration ROS 2) provides its own set of framework and middleware capabilities (Quigley et al., 2009). Most importantly, messages (for one-way communication) and services (used in two-way request-response scenarios) are described in their own definition files and compiled into C++ code. Then, communications channels known as "topics" allow to either send those messages (via topic publishers) or listen to them (via topic subscribers), thus connecting remote processes or "nodes" over the local network. Moreover, additional tools such as a parameter server allow to extend and modify the configuration of the robot both on initialization and on runtime.

ROS 2 has erected as the de-facto standard in modern robotics, hence compatibility is sought between the existing YARP codebase and the ROS 2 ecosystem. The goal of this project is to re-use the existing YARP implementation, and to introduce a thin layer of communications over the ROS 2 network to expose those underlying interfaces.

4. Proposed Architecture

Figure 3 illustrates the proposed architecture, iterating over a previous solely YARP-oriented work (Łukawski et al., 2023). The user initiates the teleoperation task by two means: either acting upon a joystick-like peripheral (e.g. the SpaceMouse), or via a web application (for PC, tablet or phone).

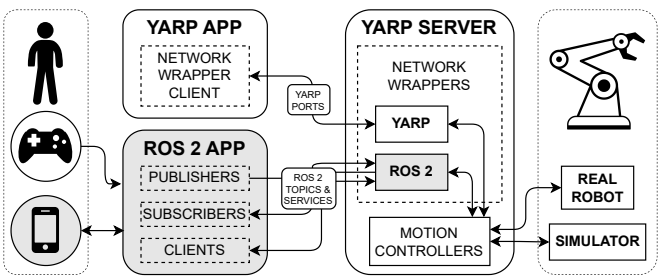


Figure 3: Proposed architecture. New components are highlighted.

The robot platforms, either real or simulated, lay on the other end of this control scheme. A motion controller manages robot state and motor commands, typically composed of a joint controller and a Cartesian controller on top of it, and implemented as one or several YARP devices. This low-level implementation is "wrapped" or called by the thin communications layer that exposes its interface over the network, the NWS device. Still in YARP jargon, it is possible to "attach" the implementation device to one or many NWS devices, such as the YARP and ROS 2 ones, to communicate with the motion controller from the outside through their respective networks.

Client applications close the gap between the NWS and the peripherals handled by the user. On this layer, in contrast, frameworks are not being mixed to avoid unnecessary dependencies and hindering code maintenance. For YARP applications, the NWC device communicates with the corresponding server. For ROS 2 applications, the usual tools and facilities provided in this framework need to be adopted: topic publishers and subscribers, and service and parameter clients.

In this work, the base YARP controller interfaces are exposed over a newly developed ROS 2 NWS device, which communicates over this network with a pair of new teleoperation applications: one that accepts motion commands from the SpaceMouse joystick, and another that provides a graphical user interface (GUI) for robot control and state monitoring.

4.1. C++ Cartesian Control Interface

Figure 4 summarizes `ICartesianControl`, a custom YARP interface designed in the RoboticsLab software ecosystem for enabling motion commands and state inspection in the Cartesian space through an underlying kinematics solver. Its methods are organized as follows:

- Remote procedure call (RPC) low-frequency commands and services that return either an acknowledge value (success or failure) or the result of a kinematic computation (e.g. `inv` obtains the inverse kinematics).
- Streaming high-frequency commands that override the internal trajectory generation.
- Configuration accessors to retrieve and modify internal controller parameters such as the duration of point-to-point trajectories or the Proportional-Integral-Derivative (PID) gain.

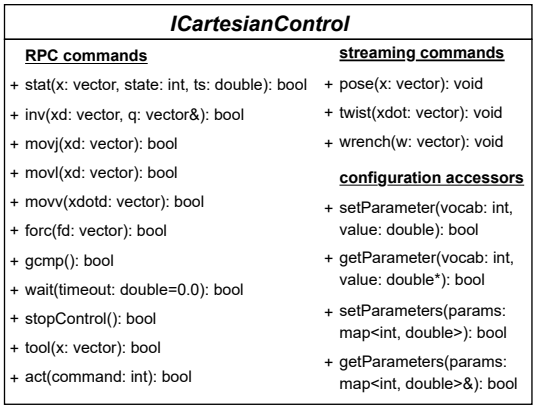


Figure 4: Unified Modeling Language (UML) class diagram of the Cartesian control YARP interface. Methods are grouped by their purpose and usage.

Most notably, the following RPC methods are highlighted due to its inclusion in the ROS 2 NWS:

- `stat`: forward kinematics and controller state.
- `movj`: isochronous motion to a pose target with no imposed restrictions on the resulting Cartesian space path.
- `movl`: linear motion to a pose target.
- `movv`: linear motion with a given velocity.
- `forc`: constant force to be applied by the TCP.
- `act`: open or close the gripper, if any.

An internal control loop is backed by a trajectory generator to ensure the path is accurately followed. Therefore, in this mode all targets are to be interpreted as setpoints and should be sent at lower frequencies. In contrast, streaming commands are issued in a high-frequency fashion, hence they are best suited for continuous teleoperation with joysticks:

- `pose`: instantaneous motion to a pose target.
- `twist`: instantaneous speed followed by the TCP.
- `wrench`: force and torque applied by the TCP.

The real TEO robot features joint motion control drives next to each motor. Before switching between `pose` and `twist` commands, for instance, the internal mode has to be set accordingly. This process is not instantaneous and therefore must occur before the next batch of streaming commands is processed. For this reason, the internal parameter manager (accessed through the `setParameter` and `getParameter` methods) is used to preset the current streaming command mode. In addition, this manager also allows to specify the frame all input parameters are referred to: either the robot's base frame (default), or the TCP frame.

4.2. ROS 2 Network Wrapper Server

In order to expose the methods of `ICartesianControl` over the ROS 2 network through the new NWS device, a mapping must be established first between the available interface methods described in the previous section, and the ROS 2 topics and services that will relay all incoming messages to their respective callbacks. Table 1 reflects those mappings.

Table 1: *ICartesianControl* methods mapped to ROS 2 message types.

method	ROS 2 message or service	YARP usage
`stat`	geometry_msgs/msg/PoseStamped	RPC service: request + response or command + acknowledge
`inv`	*custom*	
`movj`	geometry_msgs/msg/Pose	
`movl`	geometry_msgs/msg/Pose	
`movv`	geometry_msgs/msg/Twist	
`forc`	geometry_msgs/msg/Wrench	
`gcmp`	std_srvs/srv/Trigger	
`stop`	std_srvs/srv/Trigger	
`tool`	geometry_msgs/msg/Pose	
`act`	std_msgs/msg/Int32	
`pose`	geometry_msgs/msg/Pose	streaming commands (one-way)
`twist`	geometry_msgs/msg/Twist	
`wrench`	geometry_msgs/msg/Wrench	

Most methods map to ROS 2 one-way communication topics (implemented as subscription callbacks, i.e. the messages arrive at the NWS) despite being marked as two-way request-response or command-acknowledge RPC services on the YARP side. That is, the YARP usage assumes that the Cartesian controller returns a status flag about the success or failure of the command, whereas our NWS implementation doesn't enforce that returning message to be sent back to clients. This has been done on purpose since, in the ROS ecosystem, services are meant to convey more of a request-response interchange such as querying the state or doing calculations. Consequently, a custom service was implemented for the `inv` command to perform inverse kinematics: the input data is of type geometry_msgs/msg/Pose, producing an output of type std_msgs/msg/Float64MultiArray. In addition, a simple trigger-like service was added to `gcmp` (enable gravity compensation) and `stop` (stop controlling) commands.

Besides the topic subscription, the NWS also features a background thread publishing continuous robot state as obtained through the `stat` method. Lastly, the internal parameter management is exposed via the ROS 2 parameter server. Clients can use it to preset the streaming command and change the current reference frame.

4.3. SpaceMouse Controller Application

The proposed controller architecture and the previous components are materialized in the overall scheme presented in Figure 5. A collection of distributed processes, both in the YARP realm as devices and in the ROS 2 ecosystem as nodes, are combined and coordinated through their corresponding middlewares, ranging from low-level joint controllers to high-level application execution.

The ROS 2 NWS communicates with the Cartesian controller implementation, either *BasicCartesianControl* for generic robots (including TEO), or *AmorCartesianControl* for the real AMOR robot. In turn, the former manages an instance of the Cartesian solver (*KdlSolver*) that performs kinematic calculations, using the Kinematics and Dynamics Library (KDL), and the NWC counterpart of the joint controller the result of these calculations are forwarded to.

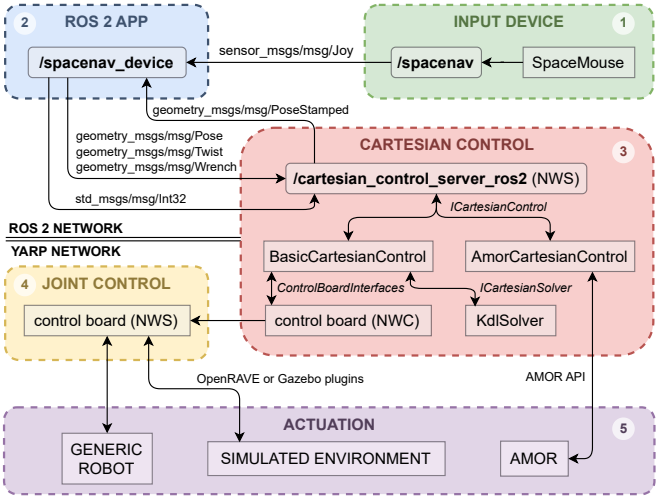


Figure 5: Teleoperation architecture for the SpaceMouse. ROS 2 nodes have been bolded out. Arrows indicate direction of data flow and message type.

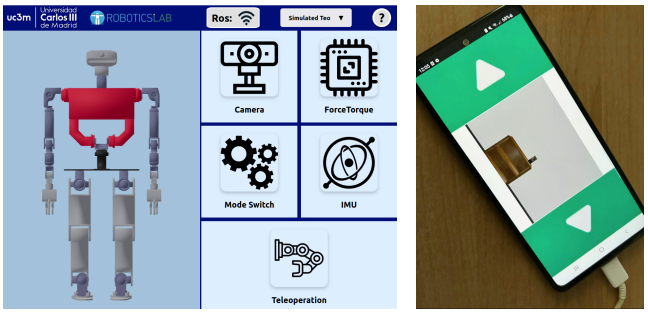
The main ROS 2 application is configured either at start or during runtime to operate in either `pose`, `twist` or `wrench` mode, communicating the user's choice to the parameter server of the NWS device. It subscribes to a topic the community-maintained SpaceMouse node publishes into, continuously transforms the incoming data accordingly using the ROS 2 transforms library (tf2), and forwards it as Cartesian commands to the NWS using the corresponding publisher topic and message type. Robot state is received once at start to fetch the initial configuration used by the `pose` command, in which case a virtual reference point is actually being applied the desired motion before converting it to a robot command. Since the SpaceMouse node streams at high frequency (circa 1 ms) and hence performance might be impacted, all processing is slowed down to 50 Hz by a timer callback in which the aforementioned process occurs. Button state is also forwarded to the NWS to invoke the `act` command of the underlying Cartesian controller, thus opening or closing the gripper.

4.4. Web Application

An additional application has been developed, targeting web browsers for any compatible devices: personal computers, laptops, tablets, smartphones. It is based on a React server application written in JavaScript, which serves HTML web pages with dynamic elements managed by client-side handlers, offering a GUI for users in order to perform teleoperated motion control and status monitoring. In the backend, the WebSockets technology is leveraged through the "rosbridge_server" ROS 2 package[1] to enable communication between the NWS nodes and the React application. In addition to the Cartesian NWS, a ROS 2 NWS device also wraps the YARP joint controller, thus exposing joint space commands and state on both YARP and ROS 2 networks.

Besides joint and Cartesian space commands and state, this web app was also devised to expose varied sensor data such as RGBD camera frames, force-torque and inertial sensor time-series plots, and audio frame playing and recording, to either stationary or handheld screen-based devices, using their integrated sensors if available. Since it runs on web browsers, maximum compatibility across device vendors is sought, also accounting for an adaptive design on smaller screens.

Figure 6 shows the web application's interface as displayed on the user's device.



(a) Main web application's window (b) Teleoperation mode

Figure 6: Graphical interfaces of the proposed web application.

Among the provided functionalities, a slider-based interface allows issuing motions in the joint or Cartesian spaces. The latter can be configured to follow either linear or unrestricted paths, mapping to the `movl` and `movj` commands, respectively. A 3D viewer is presented alongside the sliders, showing a dynamic representation of the robot through its virtual model designed in Blender. Whenever the user modifies a slider value, the desired robot pose is shown on said viewer prior to acknowledging and sending it to the NWS. In addition, a smartphone-only teleoperation mode was added to replicate the device's motion (as captured by the integrated gyroscopes) on the robot's TCP in a dynamic fashion.

Planned enhancements include a text-to-speech (TTS) interface for voice synthesis using TEO's speaker, and a demo application launcher for executing predefined actions.

5. Experiments

A set of experiments have been conducted on the available robot platforms (TEO and AMOR) and the simulators (OpenRAVE and Gazebo) to validate the proposed applications.

5.1. Teleoperation with TEO

Figure 7 depicts real TEO's left arm being issued `pose` commands. This control mode requires a fast and efficient inverse kinematics (IK) solver, which has been implemented in the *KdlSolver* device (wrapped by the Cartesian controller device) using screw theory algorithms (Łukawski et al., 2022). This mode was preferred due to the improved PID tuning in the drive's internal position control loops. It has been observed that motion is accurate and highly responsive, quickly adapting to the user's input on the SpaceMouse.

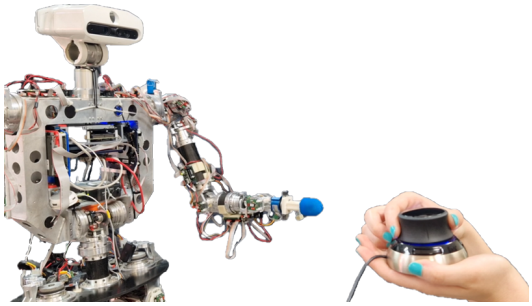


Figure 7: Teleoperation of TEO's left arm using the SpaceMouse device.

5.2. Teleoperation with AMOR

In AMOR, `twist` commands are preferred instead of `pose` due to the vendor-provided library performing better when calculating differential IK on the redundant 7-DoF arm. In addition, this robot was also shipped with a gripper that enables open/close motion to grasp objects. The side buttons of the SpaceMouse have been used for this purpose.

Figure 8 depicts three subsequent stages during a pick-and-place teleoperation task, i.e. the yellow cube was to be grabbed and released at the desired destination. A little prior practice was deemed necessary to guide the robot's TCP as desired due to the arm's kinematic redundancy.

[1] `https://github.com/RobotWebTools/rosbridge_suite`

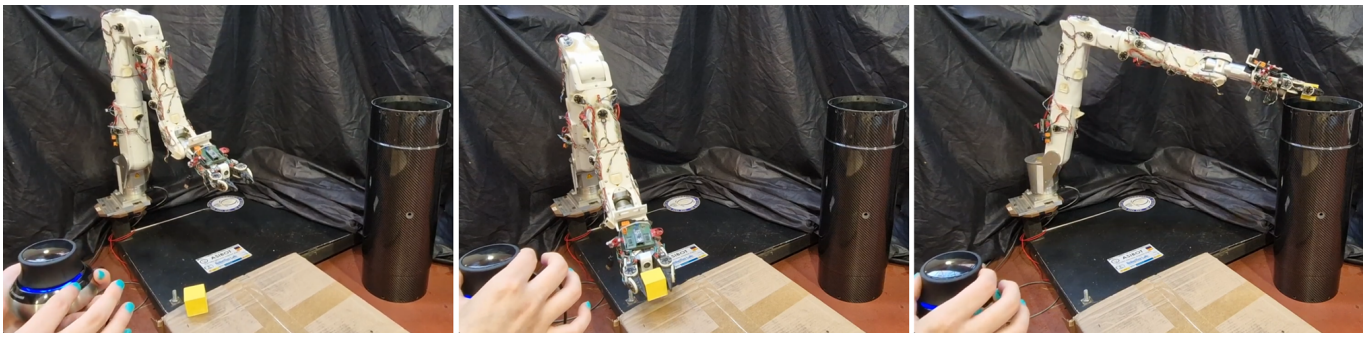
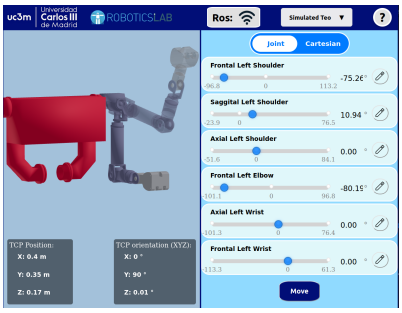


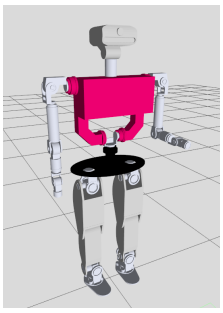
Figure 8: Pick-and-place teleoperation task with AMOR.

5.3. Teleoperation via Web Application

Figure 9(a) shows the slider interface in the joint command view. The user is allowed to preview the motion before sending the appropriate command to the robot. This application was tested on the simulated Gazebo robot as shown in Figure 9(b). Cartesian commands in both `movj` and `movl` modes have been also validated. The robot model embedded in the GUI accurately depicts the current state during operation.



(a) Slider interface for joint control



(b) TEO model on Gazebo

Figure 9: Joint control via sliders using the web app GUI.

6. Conclusions

This work further expands on existing teleoperation-related and Cartesian controller libraries and utilities developed in RoboticsLab. A first step into exposing current interfaces over the ROS 2 network and ecosystem was validated, paving the way for exploring new applications, extending the existing ones that currently rely on YARP, and adopting community-developed tools.

The source code of the kinematic controllers and devices[2], the web application[3] and its reusable React components[4] have been published on GitHub.

Acknowledgments

The research leading to these results was financed by "iRoboCity2030-CM" (TEC-2024/TEC-62) of Comunidad de Madrid, Dirección General de Investigación e Innovación Tecnológica, 5696/2024; "iREHAB" (DTS22/00105) of Instituto de Salud Carlos III; and EU structural funds.

[2] `https://github.com/roboticslab-uc3m/kinematics-dynamics`
[3] `https://github.com/roboticslab-uc3m/teo-react-webapp`
[4] `https://github.com/roboticslab-uc3m/react-components`

References

Calzada, A., Łukawski, B., Victores, J. G., Balaguer, C., 2024. Teleoperation of the robot TIAGo with a 3D mouse controller. In: Simposio de Robótica, Bioingeniería y Visión por Computador. Comité Español de Automática (CEA), pp. 133–138.

Chen, J., Moemeni, A., Caleb-Solly, P., 2023. Comparing a graphical user interface, hand gestures and controller in virtual reality for robot teleoperation. In: ACM/IEEE Int. Conf. on Human-Robot Interaction. pp. 644–648. DOI: 10.1145/3568294.3580165

Darvish, K., Penco, L., Ramos, J., Cisneros, R., Pratt, J., Yoshida, E., Ivaldi, S., Pucci, D., 2023. Teleoperation of humanoid robots: A survey. IEEE Transactions on Robotics 39 (3), 1706–1727. DOI: 10.1109/TRO.2023.3236952

Lichiardopol, S., 2007. A survey on teleoperation. DCT rapporten 2007.155.

Łukawski, B., Montesino Valle, I., Victores, J. G., Jardón Huete, A., Balaguer, C., 2022. An inverse kinematics problem solver based on screw theory for manipulator arms. In: XLIII Jornadas de Automática. CEA, pp. 864–869. DOI: 10.17979/spudc.9788497498418.0864

Łukawski, B., Victores, J. G., Balaguer, C., 2023. A generic controller for teleoperation on robotic manipulators using low-cost devices. In: XLIV Jornadas de Automática. CEA, pp. 785–788. DOI: 10.17979/spudc.9788497498609.785

Metta, G., Fitzpatrick, P., Natale, L., 2006. YARP: yet another robot platform. International Journal of Advanced Robotic Systems 3 (1), 43–48. DOI: 10.5772/5761

Moniruzzaman, M., Rassau, A., Chai, D., Islam, S. M. S., 2022. Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey. Robotics and Autonomous Systems 150, 103973. DOI: 10.1016/j.robot.2021.103973

Mortimer, M., Horan, B., Seyedmahmoudian, M., 2017. Building a relationship between robot characteristics and teleoperation user interfaces. Sensors 17 (3), 587. DOI: 10.3390/s17030587

Oña, E. D., Łukawski, B., Jardón, A., Balaguer, C., 2020. A modular framework to facilitate the control of an assistive robotic arm using visual servoing and proximity sensing. In: IEEE Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC). pp. 28–33. DOI: 10.1109/ICARSC49921.2020.9096146

Pérez Martínez, C., Pierro, P., Martinez, S., Pabon, L., Arbulú, M., Balaguer, C., 2010. RH-2: an upgraded full-size humanoid platform. In: Mobile Robotics: Solutions and Challenges. World Scientific, pp. 471–478. DOI: 10.1142/9789814291279_0058

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al., 2009. ROS: an open-source Robot Operating System. In: ICRA workshop on open source software. Vol. 3. Kobe, p. 5.

Shahbazi, M., Atashzar, S. F., Patel, R. V., 2018. A systematic review of multilateral teleoperation systems. IEEE Transactions on Haptics 11 (3), 338–356. DOI: 10.1109/TOH.2018.2818134

Sheridan, T. B., 1989. Telerobotics. Automatica 25 (4), 487–507. DOI: 10.1016/0005-1098(89)90093-9