A Scalable and Unified Multi-Control Framework for KUKA LBR iiwa Collaborative Robots

Serrano-Munoz, Antonio; Elguea-Aguinaco, Inigo; Chrysostomou, Dimitrios; Bogh, Simon; Arana-Arexolaleiba, Nestor

Published in:
2023 IEEE/SICE International Symposium on System Integration, SII 2023

DOI (link to publication from Publisher):
10.1109/SII55687.2023.10039308

Publication date:
2023

Document Version
Accepted author manuscript, peer reviewed version

A Scalable and Unified Multi-Control Framework for KUKA LBR iiwa Collaborative Robots

Antonio Serrano-Muñoz[1], Íñigo Elguea-Aguinaco[2], Dimitris Chrysostomou[3], Simon Bøgh[4] and Nestor Arana-Arexolaleiba[5]

***Abstract*— The trend towards industrialization and digitalization has led more and more companies to deploy robots in their manufacturing facilities. In the field of collaborative robotics, the KUKA LBR iiwa is one of the benchmark robots. To communicate these robots with different components and generate an interoperability infrastructure, the software libraries provided by Robot Operating System are now widely widespread. However, the latency that such communication between devices often generates, diminishes the potential of machine learning control techniques, such as reinforcement learning, when the robot must react swiftly in an unstructured environment. This paper presents a scalable and unified control system that supports both Robot Operating System and direct control and outperforms current control frameworks in terms of exploiting the functionalities of the KUKA LBR iiwa. The framework's documentation can be found at https://libiiwa.readthedocs.io and its source code is available on GitHub at https://github.com/Toni-SM/libiiwa.**

I. Introduction

The implementation of Industry 4.0 brought about a paradigm shift in many areas, leading many companies to invest in the automation and digitization of their manufacturing processes. Now, some time after the introduction of this production model, far from stagnating, its overall market value is expected to increase in the next years [1]. Collaborative robotics will be one of the markets expected to register the highest growth. This growth will be driven by multiple factors, such as high-precision work and the increasing adoption of automation in the end-use industry, reaching $1.71 billion by the end of the current year [2].

While automated processes are often complex to reconfigure, collaborative robots can easily be reconfigured and retrained for new tasks in production [3]. Notwithstanding, this reconfiguration requires intuitive robot control interfaces for simple programming and straighforward reachability.

This study was partially financed by European Union's SMART EUREKA programme under grant agreement S0218-chARmER, Innovation Fund Denmark (Grant no. 9118-00001B), and by H2020-WIDESPREAD project no. 857061 "Networking for Research and Development of Human Interactive and Sensitive Robotics Taking Advantage of Additive Manufacturing – R2P2"

[1]Antonio Serrano-Muñoz is with the Faculty of Engineering, Mondragon Unibertsitatea, Arrasate, Spain aserrano@mondragon.edu

[2]Íñigo Elguea-Aguinaco is with the Faculty of Engineering, Mondragon Unibertsitatea, Arrasate, Spain ielguea@aldakin.com, inigo.elguea@alumni.mondragon.edu

[3]Dimitris Chrysostomou is with the Faculty of Engineering and Science, Aalborg University, Aalborg, Denmark dimi@mp.aau.dk

[4]Simon Bøgh is with the Faculty of Engineering and Science, Aalborg University, Aalborg, Denmark sb@mp.aau.dk

[5]Nestor Arana-Arexolaleiba is with the Faculty of Engineering, Mondragon Unibertsitatea, Arrasate, Spain narana@mondragon.edu

Fig. 1: An illustrative scenario of the components involved in the framework.

Currently, Robot Operating System (ROS) [4] is the de facto standard for software interoperability in robotics. It is a set of software libraries that provides a communications infrastructure to connect components, enabling consistent data exchange among different applications through a common channel. Consequently, this communication infrastructure facilitates the development of distributed computing systems, covering the need for communication among the multiple processes involved in a robotic system. It has been, for many years, widely used for robot programming in academic and research environments. However, for application-oriented industrial environments, the ROS-Industrial [5] initiative or the migration to ROS2 [6] is becoming highly relevant. Unlike ROS, ROS2 bases its communication on the Data Distribution Service (DDS) standard, offering advantages such as a completely decentralized and distributed system, support for different implementations or security mechanisms of the DDS-Security Specification. These features make ROS2 a communication system increasingly sought after by companies seeking to use ROS for advanced robotics projects and artificial intelligence.

However, in the field of artificial intelligence, specifically in machine learning, there are techniques that demand direct control. This is the case, for instance, of reinforcement learning (RL), where an agent learns a control policy based on its interaction with the environment. This control tech-

nique is currently employed in unstructured environments where, occasionally, the response latency of the robot must be low. Due to the communication structure proposed by ROS and ROS2, its application in reinforcement learning, for example, may be limited by higher response times. It is therefore advisable to rely on direct control that leads to lower response times in industrial applications where the robot must act quickly [7] [8].

In this paper, a new scalable cross-platform multi-control framework for KUKA LBR iiwa collaborative robots is described. The interface not only unifies and enables control and communication through ROS and ROS2, but allows direct control for those applications where minimum control frequency is required through a scalable, simple and well-documented Application Programming Interface (API) as sketched in Fig. 1.

A. KUKA LBR iiwa Collaborative Robot

The KUKA LBR iiwa[1] is a seven-degree-of-freedom serial manipulator with force/torque sensors along each axis. This sensory distribution allows both position and impedance control, achieving compliant behavior in force-sensitive tasks. The robot joints can be programmed individually or through the Cartesian system, enabling both point-to-point motion and linear movements to a target position. The KUKA LBR iiwa can be programmed via multiple software/hardware interfaces. These include Standard and Servo, both software interfaces, and Fast Response Interface (FRI), a hardware interface. The main difference between these control interfaces is their motion update intervals.

For programming, the robot has its own control software, KUKA Sunrise.OS, which runs JAVA code on the KUKA Sunrise Cabinet control hardware. All sensory data or information related to the current task is only available locally in the robot's control system or on the KUKA Smartpad.

B. Original contribution

To the authors' knowledge, the proposed implementation is the first framework that brings together the most relevant control workflows within the robotics community for KUKA LBR iiwa robots, being able to perform direct control both from the Sunrise cabinet itself and from an external computer. The API and communication protocol are directly and easily scalable to different programming languages, which accelerates the deployment of diverse and advanced conventional and collaborative robotic applications. It also allows to configure and leverage the extensive capabilities and features available in the manipulator. Lastly, the framework is supported by a user-friendly and detailed documentation that provides end-to-end guidance, accessible to a wide variety of people from different backgrounds.

II. Related Works

At present, there are not many specific frameworks for the KUKA LBR iiwa. Moreover, due to the heterogeneous scope of their use, the existing ones have been built according to their own domain.

Among the most significant are the implementations of *iiwa_stack* [9] [10] and *KUKA-IIWA-API* [11] , both developed for ROS. The former only provides a smartservo control and a limited implementation of the FollowJointTrajectory ROS control action service [12] communication protocol. This skips sending feedback on the incremental progress of a defined target goal. The latter only allows standard, discontinuous and blocking motion execution. In addition, both frameworks provide a subset of features within the availability offered by the KUKA LBR iiwa.

Regarding ROS2-based frameworks, *iiwa_ros2* [13] stands out. It is a standalone version similar to *iiwa_ros* [14] for ROS. Both provide a stack of files and configurations over the FRI protocol for position, velocity and torque control. However, for their use, the installation of a specific KUKA hardware interface is required.

For direct control of the robot from an API, the KUKA Sunrise Toolbox for Matlab [15] allows the KUKA LBR iiwa to be controlled via an external computer using Matlab. On top of this implementation, the IiwaPy and IiwaPy3 packages [16], from the same authors, offer direct control through Python programming. Yet, their functionalities are limited to only one type of motion and lack basic intuitive documentation.

The *iiwapy* [17], an independent implementation but sharing the same name as the one described above, enables to control the robot from a Python API. However this module does not allow the direct control of the robot, since its implementation is built on top of the *iiwa_stack*, using the ROS Python API to communicate with the latter.

This paper, therefore, presents a new scalable and unified cross-platform framework that not only supports ROS and ROS2 control, but also direct control and exploits a wide variety of features available in the KUKA LBR iiwa, namely control interface and mode, and motion and execution types, outperforming the state-of-the-art frameworks in terms of robot functionality.

III. Scalable and Unified Multi-Control Framework

This section describes the proposed framework in more detail, with emphasis on its architecture and integrated control workflows.

A. Framework Architecture

The Fig. 2 shows the architecture of the implemented framework. It is divided into three main blocks (where each block represents a physical device) as described below, from the bottom to the top block.

1) KUKA LBR iiwa: The sensitive lightweight robotic manipulator, which is in contact with and modifies the environment.

[1] www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

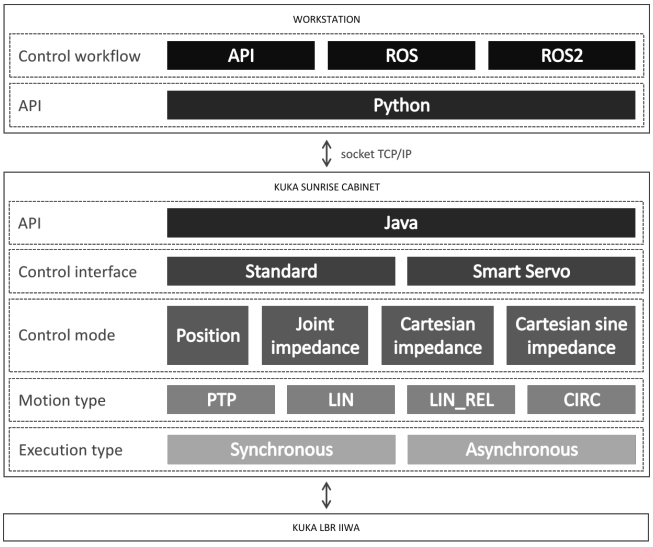

Fig. 2: Architecture of the proposed framework divided into nested blocks according to the physical devices involved and the features and capabilities of the robot, as well as the APIs and control workflows provided.

2) KUKA Sunrise Cabinet: The robotic manipulator's controller workstation which has specific software, hardware and interfaces for controlling the latter. KUKA Sunrise.OS is the operating system of the cabinet which offers toolboxes and libraries programmed in JAVA to read and modify the robot state.

One of the components implemented in this work (*libiiwa* - JAVA API) defines an interface that uses the KUKA Sunrise.OS libraries to access most of the specific features and capabilities of this manipulator in a simple and clear way. The API allows communication and control from external stations via TCP/IP protocol. In addition, it can be used to quickly develop applications executed from the cabinet itself. The following features and capabilities of the robot can be configured and used through the *libiiwa* - JAVA API:

- Execution type: Defines how the movements of the robot will be executed by the internal real-time controller. The synchronous type executes a motion command after the current motion has been completed, blocking the program flow. The asynchronous type executes a motion command without interrupting the program.
- Motion type: Defines the types of movements to be performed by the robot to go from one pose in space to another. The types supported by the implemented API are: point to point (PTP), linear (LIN), linear relative to its current position (LIN_REL) and circular (CIRC).
- Control mode: Defines which controller can be used to operate the robot. The API allows to operate in position and impedance control mode, the latter in its different variants for Cartesian and joint control space.
- Control interface: Defines which interface will be used to operate the robot controllers. The API allows to select between the two software interfaces: standard and servo (or servoing). The latter is a real-time soft interface and requires the presence of some KUKA specific libraries in the cabinet.

3) Workstation: An external computer that functions as an access interface to the operator, to other external control programs such as learned or designed control policies, or to ROS/ROS2 environments, for example.

External access or control workflows rely on a component of the library (*libiiwa* - Python API) programmed in Python that communicates, via TCP/IP, with its pair mentioned above (*libiiwa* - JAVA API). The supported control workflows are described in the following subsection.

B. Control Workflows

At the time of writing, the following workflows are supported by the developed scalable and unified multi-control framework.

1) Direct Control: The robot can be manipulated directly through the available APIs. The JAVA API can be used only for the development of applications inside the cabinet. This API also extends a communication interface that allows external handling in any programming language that interprets and follows the defined protocol described in Fig. 3.

The API programmed in Python, a programming language popular among data scientists, artificial intelligence and machine learning specialists and roboticists, enables control from external workstations or devices connected to the cabinet.

2) ROS & ROS2: On top of the Python API, a node has been developed, for both ROS and ROS2, that allows the integration of the framework (and the manipulator) with a distributed ROS environment.

This node implements subscription topics for manipulating the robot in Cartesian and joint control space as well as publication topics for exporting sensor information such as the position, velocity and torque of the joints, the Cartesian pose of the end-effector and the force and torque measured

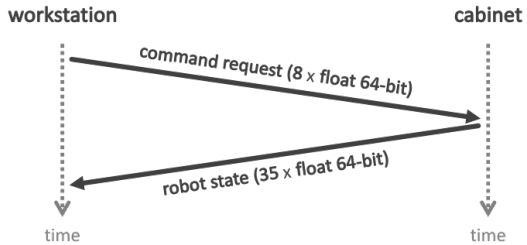

Fig. 3: Communication protocol. Communication is always initialized from the external workstation which sends a command request and receives the robot status from the cabinet. The request is formed by a command code (1) and its respective arguments (7). The robot status is formed by the command status (1), the last error code (1), the position (7), velocity (7) and torque (7) of the joints and the Cartesian position (3), orientation (3), force (7) and torque (3).

in that frame, for example. Also, it implements ROS services to configure and use the features and capabilities described above.

Furthermore, the node implements the ROS Control's FollowJointTrajectory action service used by MoveIt [18], a very popular motion planning and manipulation framework for kinematics, control and navigation, to access the robot's low level controllers. For both ROS and ROS2 versions of the node, the topic and service names, execution frequency and other parameters can be modified from the roslaunch files provided.

C. Control frequency and latency times

According to KUKA's reference for the LBR iiwa, the recommended destination setting interval should be greater than 100 milliseconds (10 Hz) for the standard interface and greater than 20 milliseconds (50 Hz) for the servo interface.

The Fig. 4 shows the mean and standard deviation of the round-trip latency time (including command execution) over 10000 samples for the standard and servo control interfaces.

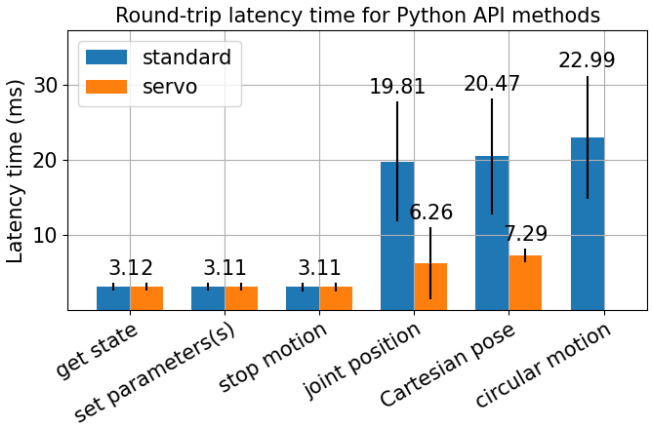

Fig. 4: Round-trip latency time (mean and standard deviation over 10000 samples) in milliseconds for the Python API methods. Methods for setting and configuring operation modes, parameters, constants and limits are grouped under the "set parameter(s)" label.

These values are much smaller than the recommended limits (over 30% and 50%, in the worst case, for standard mode and servo mode respectively), which ensures adequate control.

D. Documentation

One of the relevant points of the proposed implementation is its detailed and user-friendly documentation. The documentation is written using reStructuredText and hosted online by Read the Docs under the url https://libiiwa.readthedocs.io.

The documentation covers the installation and start-up steps of the library and details of the JAVA and Python APIs. For the ROS/ROS2 control workflow, released packages and details of the protocols and messages involved are provided. For all of them code snippets in the respective programming languages and command line call examples are included.

E. Baseline Comparison

The Table I shows a comparison of the proposed framework with respect to the most relevant existing implementations. The comparison is made on the basis of the LBR iiwa features and capabilities and the control workflows described above.

As shown in the table, our implementation covers most of the various features and capabilities of KUKA LBR iiwa robots. This makes it possible to develop robotics applications that span different levels, from basic to more complex and involving different control workflows. In addition, it enables the robot's capabilities to be exploited to new domains and tasks.

IV. Leveraging Multi-Control Framework

To prove the functionality of the proposed multi-control framework, two show cases, both for control through ROS and ROS2 via command-line interface, Python scripting and MoveIt, as well as a direct control to operate the robot using a RL control policy, are described below. Supplementary material (videos and code snippets) on the experimental results is available in the framework's documentation.

A. ROS and ROS2

For the command-line interface, the *rostopic* and *ros2 topic* tools, from ROS and ROS2 respectively, are used to operate the manipulator in Cartesian and joint space. For the configuration of the different features and capabilities of the robot, the tools *rosservice* and *ros2 service*, from ROS and ROS2 respectively, are employed.

Regarding control from Python scripting, the same commands are invoked using Python's *rospy* and *rclpy* libraries, for ROS and ROS2, respectively. For more information and code snippets and examples see the framework's documentation.

In the case of MoveIt, the implementation of the FollowJointTrajectory action service is tested through its RViz plugin that allows to create start and destination goal states for the robot interactively, test several motion planners and visualize the result.

B. Direct Control

The framework is also used to support 3D reaching experiments in which the KUKA LBR iiwa is commanded by a RL policy. This machine learning control technique allows a robot to autonomously discover optimal behavior through trial-and-error interactions with its environment. This interaction between the agent and its environment is formalized as a Markov Decision Process (MDP), which defines sequential decision making as a semi-random process and according to which the agent acts.

In the testing scenario, the robot is required to move from an initial to a target position. Subsequently, the formulation of the state, actions and reward for the RL agent training in simulation is defined.

TABLE I: Comparison of the framework developed in this work with other related frameworks/libraries.

Framework / library	Execution type		Motion type					Control mode				Control interface			Control workflow		
	Sync	Async	PTP	LIN	LIN_REL	CIRC	SPLINE	P	JI	CI	CSI	Standard	Servo	FRI	Direct	ROS	ROS2
libiiwa (our)	*	*	*	*	*	*		*	*	*	*	*	*		*	*	*
iiwa_stack		*	*	*		*	*	*	*	*	*		*			*	
KUKA-IIWA-API		*	*	*	*	*		*		*		*				*	
iiwa_ros														*		*	
iiwa_ros2														*			*
KUKA Sunrise Toolbox for Matlab	*	*	*	*	*	*		*		*		*	*		*		
iiwaPy/iiwaPy3	*	*	*	*	*	*		*		*		*	*		*		

- Control mode abbreviations: Position (P), Joint impedance (JI), Cartesian impedance (CI), Cartesian sine impedance (CSI).
- Shaded cells do not apply to the marked fields.

State Space: $s = [q, \dot{q}, p_t]$, where q and $\dot{q}$ are joint positions and their derivative, respectively, and p_t is the target position.

Action Space: $a = [\Delta q]$

Reward Function: $r = -\mathrm{d}(p_{ee}, p_t)$, where $\mathrm{d}(p_{ee}, p_t)$ is the Euclidean distance between the end-effector Cartesian position, p_{ee}, and the target Cartesian position, p_t.

V. Conclusions

This paper presents a new framework for the control of KUKA LBR iiwa collaborative robots through different workflows. These workflows include both the standardized ROS and ROS2 software libraries for software interoperability in robotics, as well as direct control for those artificial intelligence applications that require low latency. The proposed interface outperforms current approaches in terms of functionality, accessibility to robot control and integration.

In order to demonstrate the functionality and operability of the proposed communication framework, experiments are carried out and the results are reported in the framework's documentation.

The authors expect that this work can support the robotics community in the future development of new applications of the KUKA LBR iiwa, thus leveraging all the functionalities that this robot offers. In addition, they expect that the current schema can be adapted or used as a reference for the implementation of control frameworks for other robotic manipulators.

References

[1] C. Cimini, G. Pezzotta, R. Pinto, and S. Cavalieri, "Industry 4.0 technologies impacts in the manufacturing and supply chain landscape: an overview," in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 2018, pp. 109–120.

[2] F. M. I. Global and C. P. Ltd., "Collaborative robots market is reaching a valuation of us$ 8.65 billion by 2029 - comprehensive research report by fmi," feb. 2022.

[3] C. Schou, "Easy reconfiguration of modular industrial collaborative robots," 2016.

[4] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[5] M. T. Hoske, "Ros industrial aims to open, unify advanced robotic programming," *Control Engineering*, vol. 60, no. 2, pp. 20–21, 2013.

[6] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[7] T. Zhang and H. Mo, "Reinforcement learning for robot research: A comprehensive review and open issues," *International Journal of Advanced Robotic Systems*, vol. 18, no. 3, p. 17298814211007305, 2021.

[8] M. Panzer and B. Bender, "Deep reinforcement learning in production systems: a systematic literature review," *International Journal of Production Research*, vol. 60, no. 13, pp. 4316–4341, 2022.

[9] C. Hennersperger, B. Fuerst, S. Virga, O. Zettinig, B. Frisch, T. Neff, and N. Navab, "Towards mri-based autonomous robotic us acquisitions: a first feasibility study," *IEEE transactions on medical imaging*, vol. 36, no. 2, pp. 538–548, 2017.

[10] S. Virga, O. Zettinig, M. Esposito, K. Pfister, B. Frisch, T. Neff, N. Navab, and C. Hennersperger, "Automatic force-compliant robotic ultrasound screening of abdominal aortic aneurysms," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 508–513.

[11] S. Mokaram, J. M. Aitken, U. Martinez-Hernandez, I. Eimontaite, D. Cameron, J. Rolph, I. Gwilt, O. McAree, and J. Law, "A ros-integrated api for the kuka lbr iiwa collaborative robot," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 859–15 864, 2017.

[12] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, *et al.*, "ros_control: A generic and simple control framework for ros," *The Journal of Open Source Software*, vol. 2, no. 20, pp. 456–456, 2017.

[13] ICube-Robotics, "Iiwa_ros2," 2022. [Online]. Available: https://github.com/ICube-Robotics/iiwa_ros2

[14] K. Chatzilygeroudis, B. Fichera, and A. Billard, "iiwa_ros: A ros stack for kuka's iiwa robots using the fast research interface," 2019. [Online]. Available: http://github.com/epfl-lasa/iiwa_ros

[15] M. Safeea and P. Neto, "Kuka sunrise toolbox: Interfacing collaborative robots with matlab," *IEEE Robotics Automation Magazine*, vol. 26, no. 1, pp. 91–96, March 2019.

[16] M. Safeea, "iiwapy2/iiwapy3: Python library used to control kuka iiwa robots, the 7r800 and the 14r820, from an external computer," 2020. [Online]. Available: https://github.com/Modi1987/iiwaPy3

[17] R. Edwards and R. Bush, "A python interface to the KUKA LBR IIWA R800/R820," 2020. [Online]. Available: https://github.com/rivelinrobotics/iiwapy

[18] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.