2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

MoRC—A Modular Robot Controller

Carsten Oldemeyer[1], Matthias Hellerer[1], Matthias Reiner[1],
Bernhard Thiele[1], Patrick Weber[1] and Tobias Bellmann[1]

Abstract— **MoRC is a high-performance modular robot controller based on the Functional Mock-up Interface (FMI) standard. The goal is to control any (industrial) robot with electrical drives using a customizable vendor-agnostic control cabinet and an innovative, self-developed software architecture based on exchangeable multi-rate real-time control components with standardized interfaces. On the hardware side, the use of EtherCAT (Ethernet for Control Automation Technology) allows connecting a freely selectable number of COTS (commercial off-the-shelf) electrical drives and sensors. On the software side, this is matched with exchangeable control software modules based on the FMI standard. Those can be interconnected for forming user-defined multi-rate control structures which can be executed as synchronized real-time threads on a central Linux-based multi-core computing unit. That unlocks additional computational potential for advanced high-frequency control algorithms. Control structures can be switched at runtime to handle highly diverse control tasks. This paper presents the architectural concepts as well as first experiments on an industrial robot testbed.**

I. INTRODUCTION

While there exists a plethora of (open-source) software packages for robotics, often driven by academic research groups interested in autonomous robots, anthropomorphic robots, and artificial intelligence (AI), the market for *industrial robotics* is characterized by proprietary solutions. As a result, the achievable control functionality and available interfaces in general depend on the specific manufacturer. In particular, low-level access to the robot's software, essential for research and development of new control and HMI concepts, is generally not possible. Safety challenges, arising from dynamic experiments with heavy industrial robots, further hinder the availability of open controller possibilities for such systems.

The idea behind our Modular Robot Controller (MoRC) is creating a vendor-agnostic, highly flexible, robot controller for industrial robots. In order to achieve that goal two main areas are addressed and presented in this paper: In terms of hardware, a "Drop-in replacement" for proprietary control cabinets has been designed and implemented, housing commercial off-the-shelf (COTS) power electronics components, safety systems and a (for a robot control cabinet) potent computer. This computer runs the new software framework responsible for the real-time control of the robot connected to the cabinet.

[1]The authors are with the Institute of System Dynamics and Control, German Aerospace Center (DLR), 82234 Oberpfaffenhofen, Germany. Contact: tobias.bellmann@dlr.de

Another new concept in this paper is the software frameworks ability to exchange complete control architectures at runtime to accommodate different control concepts with heavily varying structures, sample rates and interfaces (e.g. classical PTP movements vs. adaptive force controlled movements). By default, an exchangeable dynamic robot model is integrated in the MoRC software for system definition and seamless switching between simulated and real robot movements. This enables the user to safely test new controllers or software components in a virtual commissioning process.

ROS is a popular open-source software framework for robot applications[1], [2]. While the initial ROS had no direct support for real-time applications, the ROS2 revision removed that limitation. Nevertheless, the main focus of ROS is not high-frequency real-time control, but rather being a middleware suite for more high-level functionality. Another well-known, general-purpose robotic software package with determined focus on real-time capability is OROCOS [3], [4]. Most projects using OROCOS rely on vendor-provided interfaces which usually include frequency and access limitations. While OROCOS uses its own component definition, MoRC leverages the widely used Functional Mock-up Interface (FMI) standard [5] for that purpose.

FMI is an open standard that defines a container and an interface to exchange simulation models. The model container (Functional Mock-up Unit, FMU) implementing FMI is distributed as one ZIP archive file containing all relevant files. FMUs may have inputs, outputs and parameters, therefore typical "control blocks" can be wrapped as FMU. Today, more than 170 tools can export or import FMUs [6]. Leveraging FMUs within MoRC opens up an interesting pool of tools which can be used for developing the control algorithms.

The necessity for multi-rate control systems arises because various sensors and control effectors need to be sampled at different rates, or for performance enhancement on finite computational resources (see for example [7], [8]). Forget et al. [9] proposed a language for programming multi-rate control systems which extends formal approaches known from clocked synchronous languages [10]. Similar to this approach, the MoRC software supports the execution of concurrent, synchronized software processes, based on clocked synchronization of communicating multi-threaded FMUs.

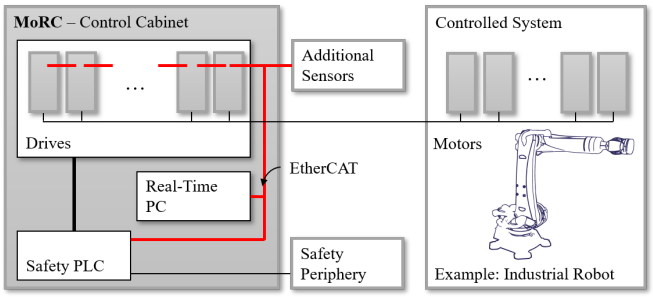


Fig. 1: MoRC hardware overview

II. HARDWARE

A. Conceptual Overview

Fig. 1 gives a conceptual overview of the system. The left side depicts the hardware manifestation of the robot controller – a control cabinet built from COTS components. In particular, it consists of a number of electrical drives (power electronics) and a safety programmable logic controller (PLC) for achieving conformance with safety regulation for industrial robots (e.g., relevant standards [11]). EtherCAT (Ethernet for Control Automation Technology) [12] is used as real-time communication backbone. It connects the drives, PLC and sensors with a central computing unit (PC with real-time Linux) and is open for adding additional components, e.g., sensor devices. The right side shows the controlled system. The robot controller is vendor-agnostic, it is adaptable to any industrial robot with reasonable effort. Actually, it is not limited to industrial robots, but may also be a reasonable choice for controlling other motion device systems.

B. Control Cabinet

A rendered image of our control cabinet prototype is shown in Fig. 2. The control cabinet contains six Bosch IndraDrives, each connected as an independent EtherCAT slave. The used designs are compact single-axis converters. By adding more converters, additional motors can be controlled. Other components which are connected as EtherCAT slaves are the Beckhoff IO terminals with coupler, providing the possibility of integrating additional sensors. An additional safety communication module from Pilz with safety rated IO terminals is installed for emergency switches and other external safety devices. The connectors for the in- and outgoing lines for power and data are subsumed as external connectors in the rendering at the bottom of the cabinet. The mains power supply connection is switched by the ON/OFF-switch and the drives' power can be cut by individual circuit breakers.

Due to the flexibility of EtherCAT it is simple to add slave devices, e.g., for attaching and integrating additional sensors and robot accessories. Bus interfaces on the underside of the cabinet can expand the existing system as desired with additional components. A Linux-PC is used as central real-time computing unit. The PC uses two Ethernet interfaces, one is reserved for real-time EtherCAT communication, the other is connected to a standard Ethernet network for communicating with the operator PC which is used for programming and displaying visualization and plotted data.

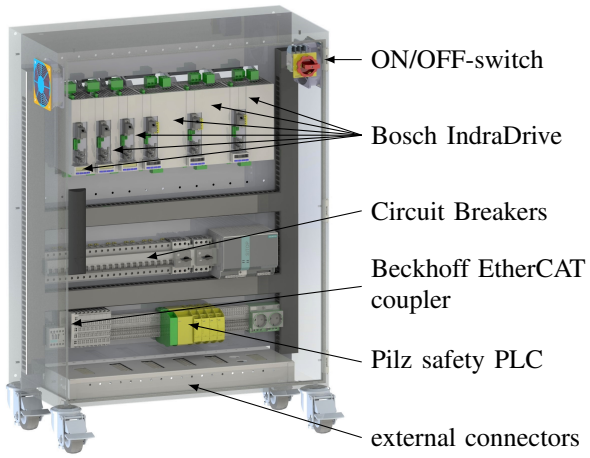


Fig. 2: Rendering of MoRC control cabinet with component designations, Real-Time PC not shown

TABLE I: MoRC Control Cabinet Technical Data

Name	*Value*
Dimensions	600 mm × 1100 mm × 450 mm
Mass	125 kg
Supply voltage	200 – 500 V
Supply current	32 – 64 A
Drive 1-3	14 kW
Drive 4-6	4.8 kW

C. Electrical Drives

The motor drives are capable of operating a wide variety of motor types. Commutation is specified and monitored by the drive. Built-in multi-encoder interfaces or additional plug-in cards can read almost all types of feedback from the robot. This makes it possible to operate common commercially available motors on the market. In addition, included safety features monitor motor data or mechanical components. Table I lists the operating range of the currently used drives.

D. Safety

Independently from the MoRC software a safety PLC monitors the connected robot cell safety devices including door and enabling switches. This system is variably programmable and can be individually combined with other systems on a modular basis. Communication takes place in two different ways. Either via EtherCAT or via safely connected relays. Individual safety components in the modules can thus exchange data or commands in parallel. Functions of the electrical drives like Safe-Torque-Off, Safe-Brake-Control and Emergency-Stop are triggered in emergency operation. In addition, a brake relay is integrated, which safely closes all brakes. This is usually necessary if the connected robot has to be stopped in the event of a fault.

III. SOFTWARE

The purpose of the MoRC Software is to configure and orchestrate the operations of all connected hardware components. It provides tools for programming the robot motion, integrating different kinds of controllers and supporting features such as real-time monitoring and visualization, logging and data recording.

The fundamental software structure of MoRC comprises four core modules: The *EtherCAT* module encapsulates communication with the connected devices. The systems behavior, i.e. how sensors and actuators are connected, is defined by a task-specific collection of real-time components, in a base module called *Assembly*[1]. The components of the controller are configured and connected in the *Assembly* module by Lua scripts. The *Engine* module contains both the *EtherCAT* and the *Assembly* module, providing a real-time execution context as well as facilitate the online change of the currently running Assembly. Parallel to the *Engine* module the *Program Interpreter* executes user-defined high-level robot programs.

An example instance of the MoRC software structure is depicted in Fig. 3.

A. EtherCAT module

The *EtherCAT* module has one instance of the *Bus* class and one *Device* class instance for each connected device. Basic EtherCAT functionality including drivers is provided by the IgH EtherCAT Master [13]. Every device type needs a dedicated *Device* class implementation of its behavior. This class has to be implemented only once per generic device type. The universal class for drives e.g. encapsulates handling of the device state machine in accordance to the CiA 402 protocol [14], which is accessed via CANoverEtherCAT, eliminating the need for other parts of the MoRC software to know and consider these details.

Another component of the *EtherCAT* module is the *Safety PLC* class, responsible for interfacing with corresponding hardware devices. A setup-specific program runs on the safety PLC, monitoring conditions such as emergency switch status, robot cell door closures, MoRC software pulse signals, and motor safety. EtherCAT facilitates data exchange between the MoRC software (*Safety PLC* class) and the safety PLC, with the PLC program remaining unchanged after commissioning. This information can be leveraged to provide users with insights into why the safety system may be inhibiting operation.

B. Assembly module

Fig. 3, depicts an assembly with typical components for industrial robot control. It includes a trajectory generator for shaping robot paths and a feed-forward controller, incorporating robot and payload models, to compute target values for a feedback control component. This component manages motor torques and integrates sensor feedback, aligning with

[1]not to be confused with the low-level programming language

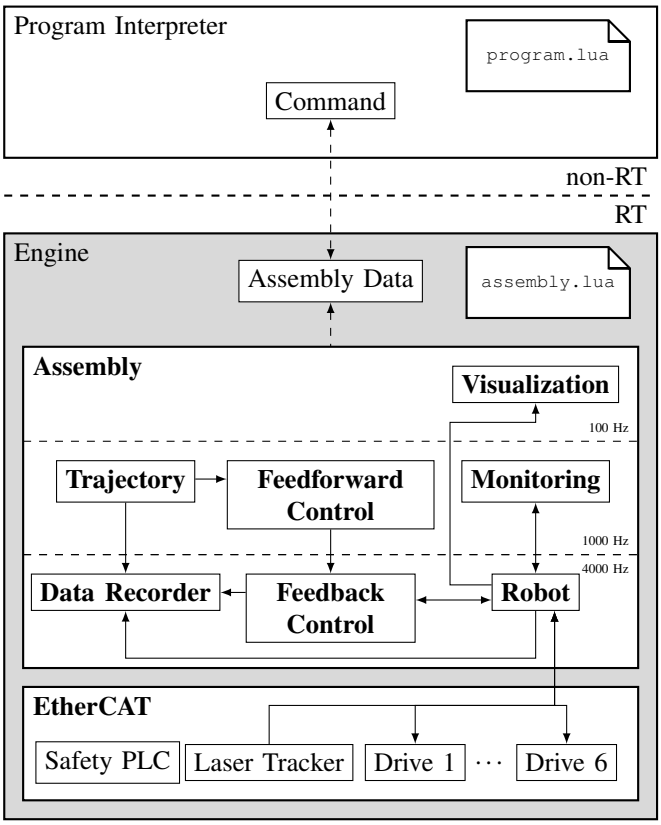


Fig. 3: Software architecture sketch showing an example instance with a typical control structure used in industrial robots, components printed with bold letters are updated by their own thread. Arrows indicate signal flow, dashed arrows indicate acyclic data transfer

the concept of "Behavioral Control" proposed by Siciliano et al. in [15]. In contrast to the approach in [16], a MoRC *Assembly*'s structure remains flexible until instantiation, with just its components initially existing in binary form, representing a slightly higher-level abstraction.

Assemblies are described by Lua scripts and can be exchanged during runtime between the commands of the robot program. This allows for highly flexible control structures and eliminates the need to recompile the MoRC software. This modular approach supports experiments with single components without compromising safety.

Swapping assemblies at runtime enables specialized robot capabilities, by optimizing control structures as separate assemblies for each functionality. One example is the switch between position and force control scenarios: initially, a high-precision position-based controller can be employed for object approach, transitioning to a force control structure for object handling.

As interface definition between components MoRC employs the FMI Standard. FMUs in this model feature named inputs, outputs, and parameters, supporting data types like boolean, integer, real, or string. They undergo a defined life cycle, including creation, initialization, periodic updates, and eventual destruction. During the update phase, components

compute fresh output values based on inputs, adhering to a predetermined time grid configured during component creation.

In the exemplary assembly shown in Fig. 3, all components except for Data Recorder and Robot are currently FMU components implemented using the FMI4CPP library [17]. FMUs are used to integrate mechanical models and controllers which are created in languages more specialized for the task than C++ like Modelica or Matlab Simulink. They are created mostly from DLR Modelica libraries (e.g. [18][19]) serving as a know-how backbone for robot control and path-planning. Such model libraries condensate knowledge and ease the fast development of new components. However, it is also possible to write FMUs directly in other languages such as C/C++.

The *Robot* component demonstrates the practical use of FMUs as carriers of model knowledge. This component is defined with a Lua script called *Robot Definition Script* and initialized during startup with a specific FMU, describing the robot cell. Contrary to classic approaches, this FMU not only contains the machine data of the robot, but provides complete semantics of the robot cell. This encompasses information as sensor and actuator placements as well as a model of the plant, tool and workpiece. This makes it possible to switch between either a simulation of the robot program or the actual control of the real robot, allowing a risk-free virtual commissioning of new robotic applications.

This makes the *Robot* component the translator between real world and model world as indicated by this component being the only one with connections to the *EtherCAT* module in Fig. 3.

A *Monitoring* FMU oversees the completion of the current command and detects unsafe states of the robot, prompting termination of the current assembly.

One of the components is a *Data Recorder*, collecting selected system outputs and stores them in an HDF5 file with relevant metadata. These measurements form the foundational dataset for system identification and controller optimization.

The pre-compiled components comprising an assembly can be selected by the user with a Lua script. Utilizing the sol2 [20] Lua interface, components are instantiated, configured, and linked accordingly.

Each component is assigned an appropriate sampling time, typically guided by the frequencies in Fig. 3. Assembly parameters rely on data stored in the *Engine*'s *Assembly Data* database, including information like initial motor angles and target positions for specific maneuvers by higher-level modules. The content and structure of the database is assembly-specific.

After the current robot program command and its corresponding assembly finishes, components can send relevant data to the *Assembly Data* database.

C. Engine module

As shown in Fig. 3, the *Engine* module ensures the correct execution of both the *Assembly* and *EtherCAT* modules. Its main function is to coordinate the real-time execution as well as starting and stopping assemblies.

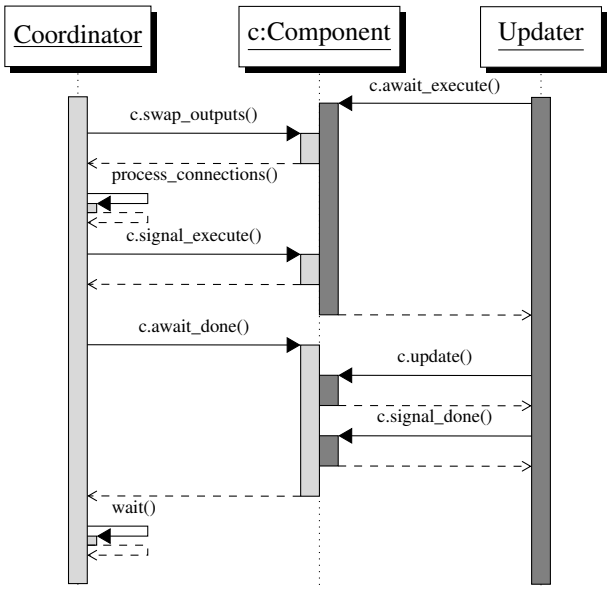


Fig. 4: Sequence diagram of MoRC component execution model showing how the two threads *Coordinator* and *Updater* use the components functions to achieve a synchronized update

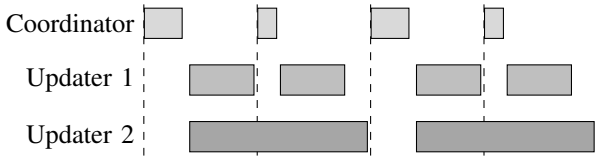


Fig. 5: Time grid of thread synchronization

To fully utilize the processing power of modern CPUs, MoRC adopts a parallel, multi-rate approach to component execution. Each component operates within its dedicated *Updater* thread, all of which are coordinated by a central *Coordinator* thread.

In the MoRC system, the *Coordinator* thread within the *Engine* is the sole thread directly connected to the system clock. All other threads achieve synchronization by logically aligning with this *Coordinator* thread.

The task of the *Coordinator* is to collect all data and synchronize the *Updater* threads. It collects all output data of currently finishing *Updater* threads and sends it to connected components. Depending on the sample time of the component, it signals the execution of the waiting *Updater*. The Updater signals completion of the update to the Coordinator. When all components relevant to the current time step are done, the Coordinator waits until it is time to start the next cycle.

For a visual representation of the interaction between *Coordinator* and *Updater* threads for a single component, please refer to Fig. 4.

Because the *Coordinator* thread has the same sample time of the fastest component, i.e. the *EtherCAT* module, and

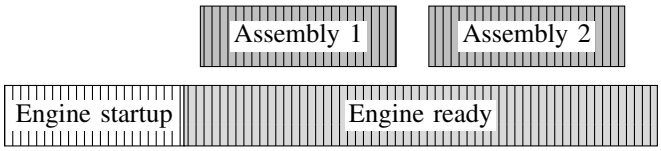


Fig. 6: Illustration of MoRC program sequence with time progressing from left to right

needs to process the data transfer for all components, their sample times have to be multiples corresponding to the bus frequency.

To allow the assembly to be exchanged the *Coordinator* and *Updater* mechanism is two-tiered with the *Assembly* being a component itself. The Updater thread of the *Assembly* is in turn the *Coordinator* for all components in the Assembly having their own *Updater* threads. This hides the variability in the number of components to update between Assemblies from the *Engine Coordinator*.

As the startup duration of assemblies can vary, the *Engine Coordinator* suspends the update process for the current assembly. In the meantime it continues to update the *EtherCAT* module. This approach removes the necessity for a field bus restart upon the commencement of each new assembly.

Upon successful initialization of the assembly, its threads synchronize with those already in operation, as depicted in Fig. 6. The initiation and setup of assemblies are managed by the *Engine*, but the determination of which assembly to execute at any given time is determined by the *Program Interpreter*.

D. Program Interpreter module

To accommodate the integration of advanced algorithms such as sophisticated planners into assemblies while addressing their computational demands, MoRC offers a *Program Interpreter*. This interpreter operates asynchronously, without real-time constraints. This architectural distinction is visually represented in Fig. 3 by the dashed line connecting the *Program Interpreter* and the *Engine*, as well as the acyclic data exchange via *Assembly Data*. The *Program Interpreter* utilizes Lua scripts with added custom tailored robot commands (example in [18]), allowing for the flexible definition of new commands from accompanying Lua libraries.

The robot language uses the skill model with precondition check, execution and evaluation as described in [21] as a basis for command definitions. Each command is a Lua function with included hooks for precondition check and evaluation. Every robot command corresponds to a distinctive assembly, to be loaded upon the execution of the command. New commands can easily be added, to e.g. integrate new Assemblies, facilitating convenient testing of new robot behaviors along proven commands. Commands can be provided either from a file or via a command line interface, enabling direct user engagement in program execution, often referred to as 'Read-Eval-Print Loop' (REPL).

When commands, along with their corresponding assemblies, are executed sequentially, the resultant program sequence closely resembles the depiction in Fig. 6.

Fig. 7: Experiment setup with KUKA KR16 and laser tracker. Laser probe is attached to the 16kg load facing the laser tracker in the background.

The *Engine* startup is focused on preparing the hardware devices for operation. When the *Engine* is ready, the *Program Interpreter* begins with the first command starting the first assembly. After assembly 1 has finished, there is a short period without an assembly running until the next command is read and the next assembly is prepared and initialized. Once there are no more commands all modules are stopped and the MoRC program awaits new user input.

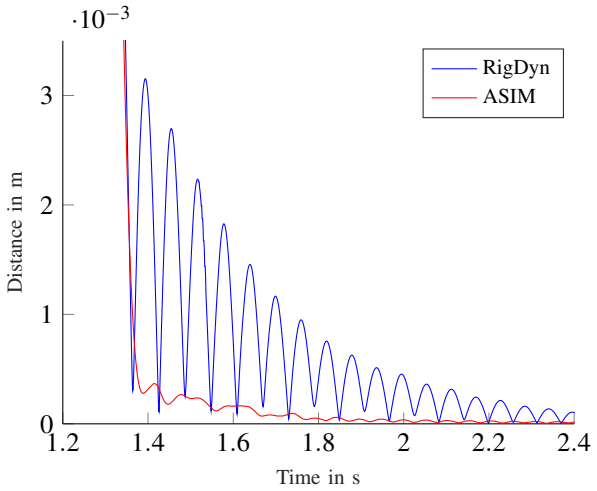
IV. HARDWARE EXPERIMENTS

After finishing the development of the MoRC software on a motor testbed, without the mechanical hazards of a moving robot arm, first experiments have been performed on a KUKA KR16.

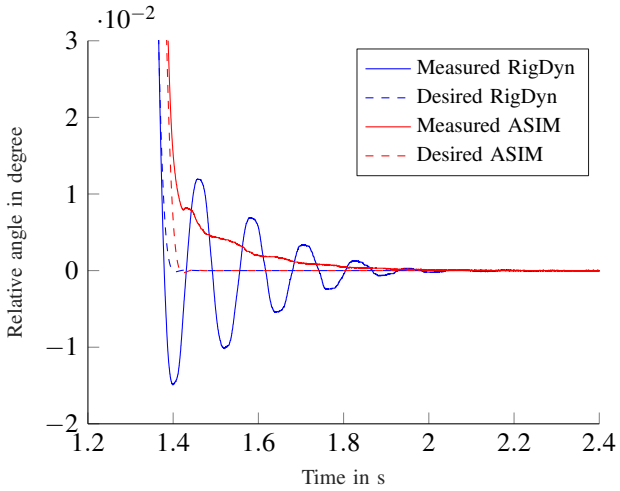
In this testbed, the robot is placed in a safety controlled cell, where robot movements and dynamic behavior can be analyzed and optimized. A laser tracker (Leica AT960) can be used to obtain exact positions of the robots TCP. The modular architecture of the MoRC software based on FMU modules allows for a very flexible control framework. The complexity of these modules can vary and depending on the computational requirements, the sample rate of the component can be adjusted. To demonstrate the capabilities of MoRC an experiment was setup as an example comparing two variants for a feed-forward (FF) module. For the comparison two different assemblies with FMU based controllers have been evaluated. The first assembly contains a simple rigid body (RigDyn) feed-forward component. The rigid body model of the robot contains the full rigid body dynamics, as well as motor inertia and friction and the load at the robot TCP.

The second assembly contains an FMU which was designed as an advanced feed-forward component, also considering the nonlinearity of the powertrain with stiffness and friction, as well as the structural flexibility of the robot. The advanced feed-forward controller component is based on the approximate structural-elastic inverse model (ASIM) design described in more detail in [22].

Since this was one of the first experiments with MoRC controlling the KR16 testbed, we can show only some preliminary results without final tuning and robot calibration.



(a) Laser tracker data. Cartesian distance adjusted to the same final position.



(b) Motor angles of axis 1 adjusted to the same final position converted to link side values.

Fig. 8: Excerpts of experiment data showing end of a 5° Point-to-Point (PTP) movement of axes 1, 2 and 3. Movement starts at 1 s. Final joint angles are [0,−90°,90°,0,0,0] according to vendor convention. Maximum load of 16 kg attached.

The focus is on demonstrating initial operation. Separate assemblies are created for each FF variant, executed one after the other, allowing the experiment to be done within a single robot program without restarting the complete MoRC setup.

For the experiment, a simple PTP trajectory was designed using only kinematic constraints for the acceleration and velocity. This trajectory is then filtered using a low pass Bessel filter and the results are fed to both the rigid body FF component as well as the ASIM component in both assemblies. Both FF components compute the resulting desired motor torques, positions and velocities. These are then used by the feed-back control component as desired input. For this first experiment a simple P-PI feedback control module FMU was used for both.

Using a high end laser measurement system the difference in the trajectory tracking capabilities between the two FF approaches is measured. The laser reflector mirror is mounted at the load of the KR16, near the TCP. Because the RigDyn does not consider the gear and structural elasticity, the demand values on the motor side are different compared to the ASIM. However both use the same low pass filtered desired link side trajectory. To improve comparability the motor sided and link sided measurements are adjusted to the same end position, removing differences because of gear and structural elastic deformation. For the ASIM FF Fig. 8a shows a significant decrease of the oscillations immediately following the robot's arrival at the target position with the maximum spike reduced by a factor of around six to under 0.5 mm. In Fig. 8b desired and measured motor angles of the first axis, which is strained most in the chosen position, are plotted. The ASIM FF allows the feed-back controller to settle faster than using the RigDyn FF by reducing excitation of the robot's elasticities. Thus the tracking error is improved on motor as well as link side.

V. OUTLOOK

The new robot controller presented in this paper opens up new possibilities in (industrial) robotics research. With the availability of a modular, safe and vendor-independent control environment for electrical driven systems, scientists and engineers are enabled to implement new low-level controller architectures, HMI concepts, Industry 4.0 integration and more. Often overseen aspects as safety and norm conformity are addressed in the controller hardware, so that even researchers new to the field can work on oftentimes large and dangerous robot systems. The possibility to change online between different multi-rate software architectures allows for very specialized and task-tailored control strategies during the same robot program run.

Because most software modules are provided as exchangeable FMUs, a multitude of developer tools can be used. The use of COTS hardware modules allows to easily integrate new sensors and tools based on EtherCAT. Since the base hardware is also build up from COTS parts with standardized interfaces, upgrades to increase e.g. computational or electric performance are effortlessly possible. The vendor independence also means that a large number of industrial robots are suitable for the use with MoRC, it even can be adopted and used for non-robotic test bench setups.

Future work includes updating robot models, adding features and comparing MoRC's control performance to the vendor cabinet. It is also planned to connect larger robot types, like the KUKA KR210 or KR500, supporting relevant research platforms at our institute and exploring the system's operational range. On platform level next steps will include the development of a prototype for a hand terminal and related programming HMI to improve the ergonomics of interacting with the robot. This will result in a more complete system, which might be attractive beyond basic research.

REFERENCES

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, Kobe, Japan, May 2009.

[2] 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. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[3] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 3, 2001, pp. 2523–2528 vol.3.

[4] P. Soetens and H. Bruyninckx, "Realtime hybrid task-based control for robots and machine tools," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, Apr. 2005, pp. 259–264.

[5] *Functional Mock-up Interface for Model Exchange and Co-Simulation*, FMI development group, Modelica Association Std., Rev. 2.0.4, Nov. 2022. [Online]. Available: https://www.fmi-standard.org/

[6] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the 8th International Modelica Conference*, ser. Linköping Electronic Conference Proceedings, C. Clauß, Ed. Linköping University Press, März 2011, pp. 105–114. [Online]. Available: https://elib.dlr.de/74668/

[7] D. P. Glasson, "Development and Applications of Multirate Digital Control," *Control Systems Magazine, IEEE*, vol. 3, no. 4, pp. 2–8, 1983.

[8] M. Tomizuka, "Multi-Rate Control for Motion Control Applications," in *The 8th IEEE International Workshop on Advanced Motion Control, 2004. AMC '04.*, 2004, pp. 21–29.

[9] J. Forget, F. Boniol, D. Lesens, and C. Pagetti, "A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 527–534.

[10] A. Benveniste, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The Synchronous Languages 12 Years Later," in *Proceedings of the IEEE*, vol. 91 (1), 2003, pp. 64–83.

[11] *Robots and robotic devices - Safety requirements for industrial robots - Part 1: Robots*, International Organization for Standardization Std. ISO 10 218-1:2011, July 2011.

[12] *DIN EN IEC 61158-1, Industrielle Kommunikationsnetze - Feldbusse. Teil 1, Überblick und Leitfaden zu den Normen der Reihen IEC 61158 und IEC 61784 (IEC 61158-1:2019) - Industrial communication networks - fieldbus specifications. Part 1, Overview and guidance for the IEC 61158 and IEC 61784 series (IEC 61158-1:2019)*. Frankfurt am Main: Beuth Verlag GmbH.

[13] F. Pose, "EtherCAT Master," Ingenieurgemeinschaft IgH. [Online]. Available: https://gitlab.com/etherlab.org/ethercat

[14] *Adjustable Speed Electrical Power Drive Systems: Generic interface and use of profiles for power drive systems - profile type 1 specification*, International Electrotechnical Commission Std. IEC 61 800-7-201, 2015.

[15] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Springer Berlin Heidelberg, 2008.

[16] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part II)," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 100–112, 2010.

[17] L. I. Hatledal, "FMI4cpp." [Online]. Available: https://github.com/NTNU-IHB/FMI4cpp

[18] T. Bellmann, A. Seefried, and B. Thiele, "The DLR Robots library - using replaceable packages to simulate various serial robots," in *Proceedings of Asian Modelica Conference 2020*, ser. Linköping Electronic Conference Proceedings, no. 174. Linköping University Press, 2020, pp. 153–161. [Online]. Available: https://elib.dlr.de/138327/

[19] S. Kümper, M. Hellerer, and T. Bellmann, "DLR Visualization 2 library - real-time graphical environments for virtual commissioning," in *Proceedings of 14th Modelica Conference 2021*, M. Sjölund, L. Buffoni, A. Pop, and L. Ochel, Eds. Modelica Association and Linköping University Electronic Press, September 2021, pp. 197–204. [Online]. Available: https://elib.dlr.de/144780/

[20] J. Meneide, "sol2." [Online]. Available: https://github.com/ThePhD/sol2

[21] S. Bøgh, O. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills," *Proceedings of the 43rd International Symposium on Robotics*, 2012.

[22] M. Reiner, "Modellierung und Steuerung von strukturelastischen Robotern," Ph.D. dissertation, Technische Universität München, Fakultät für Maschinenwesen, 2011.