2025 IEEE/SICE International Symposium on System Integration (SII)
January 21-24, 2025, Munich, Germany

ur_rtde: An Interface for Controlling Universal Robots (UR) using the Real-Time Data Exchange (RTDE)

Anders Prier Lindvig, Iñigo Iturrate, Uwe Kindler and Christoffer Sloth

***Abstract*— In this paper we introduce an open source cross-platform C++ interface for controlling Universal Robot (UR) manipulators. This interface is capable of controlling all UR robots that facilitates a RTDE (Real-time Data Exchange), which is the communication protocol used by this interface. Previous interfaces did not leverage the RTDE and it was one of the motivating factors for writing this interface. *ur_rtde* can be used both from C++ and Python with bindings. A Python package has been released, to make it easy to install and use on Windows, Linux and MacOS. We show that the proposed interface outperforms other interfaces in terms of real-time characteristics.**

I. INTRODUCTION

This paper introduces *ur_rtde*, an open source[1] and performance-oriented cross-platform library for real-time control of Universal Robots (UR) robots via TCP/IP.

The advent of collaborative robots has not only opened up for easy integration of robot automation within the industry, but has also lowered the barrier of entry for researchers. Indeed, apart from the financial aspects, the safety features embedded in collaborative robots make them much easier to deploy in a laboratory environment, with minimal need for risk assessment.

It is arguably the prevalence of software interfaces on these cobots that truly enables research, as it allows researchers to easily develop and test robot controllers on a real robot platform. Indeed, manufacturers like Franka Emika and KUKA, among others, offer the Franka Control Interface (FCI) [1] and Fast Research Interface (FRI) [2], which allow real-time control of robots, and potentially even joint torques at up to 1000 Hz.

In the case of Universal Robots, the corresponding interface is the Real-Time Data Exchange (RTDE) interface [3], which was not used by previous UR interfaces such as the ur_modern_driver [4]. RTDE allows data synchronization with UR robots through a TCP/IP connection. While UR A/S provides a Python client that uses the interface[2], it includes an example that is limited to only move the robot to joint positions using the RTDE. For an implementation focused more on robot control, one must instead turn to the official drivers for ROS [5] and ROS2 [6], which, while providing the advantages of the large ROS ecosystem, also comes with disadvantages of performance overhead and high complexity.

Uwe Kindler is with CETONI GmbH uwe.kindler@cetoni.de and the rest of the authors are with the Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Denmark {anpl, inju, chsl}@mmmi.sdu.dk

[1]https://gitlab.com/sdurobotics/ur_rtde

[2]https://github.com/UniversalRobots/RTDE_Python_Client_Library

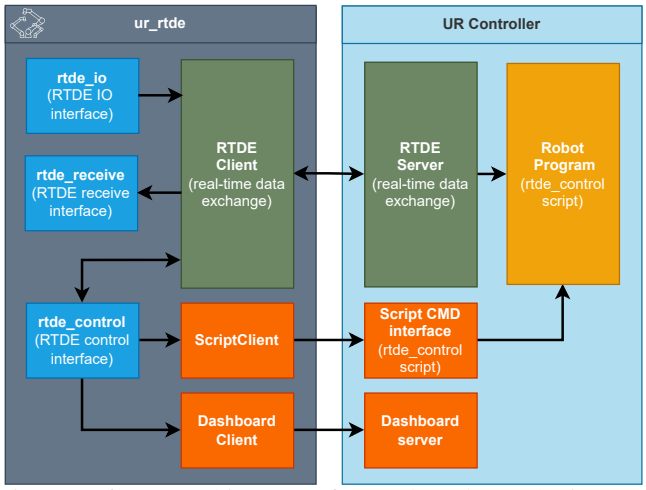


Fig. 1: Software architecture of *ur_rtde* - Shows the interaction between the physical UR controller and *ur_rtde* using RTDE (green) as well as the script and dashboard client (orange).

Contrary to the above implementations, we focus on a lightweight, cross-platform, and multi-language implementation that allows rapid prototyping, as well as low-overhead high-performance real-time control applications. The main features of *ur_rtde* are:

- Real-time control.
- Ease of installation and use, following the official URScript API as closely as possible.
- Multi-language support for C++ (native), Python (through bindings), and MATLAB (through Python interface).

In the remainder of this paper, we will present the library architecture in Section II, introduce multiple features in Section III, provide a performance evaluation and comparison to other available interfaces for UR robots in section IV, discuss some uses cases of the library in section V, and conclude in Section VI.

II. ARCHITECTURE

The *ur_rtde* library consists of three independent interfaces: rtde_receive, rtde_control, and rtde_io, marked in blue in Fig. 1. The motivation for splitting up the interface is to make it modular, so the user can choose to include only what is needed. This also provides more flexibility when using multiple functionalities of the robot at the same time, e.g., setting the IOs and operating the speed slider of the robot,

while commanding it to move, as this functionality can be parallelized.

The three interfaces are programmed in C++ following the C++11 standard [7], which was chosen for its backwards compatibility. The library is designed to be cross-platform, by using network sockets that work on Windows, Linux and macOS and by performing byte order conversion in a platform-independent way.

All three interfaces communicate with the UR Controller using the RTDE protocol, for which it is necessary to configure communication recipes that describe which data is sent to and received from the UR controller. To make *ur_rtde* more user-friendly, these communication recipes are automatically configured by each of the interfaces.

Currently, Universal Robots A/S only allows the execution of motion or control commands on the robot manipulator through script (URScript) commands. The implication of this is that ur_rtde will never achieve a low-level control of the robot, but instead provide a reference through script commands for the low-level control running on the controller. The *ur_rtde* library implements a rtde_control script that translates commands from the RTDE Control Interface client into executable robot commands. This script is typically sent by the rtde_control interface to the controller through the implemented ScriptClient, which communicates with the UR script command interface on the robot on port 30003. The script is only transmitted once, during the initialization of the rtde_control interface.

The rtde_control and rtde_receive interfaces communicate with the RTDE continuously using background threads, which will automatically inherit any real-time priority configured with the constructor of the given interface.

Furthermore, a function for setting the application real-time priority is made available in the rtde_utility header *setRealtimePriority(priority)*, which takes an integer value in the range (0-99) specifying the real-time priority that will be assigned to the calling thread. Here, the number 99 represents the maximum priority that can be assigned to a thread. By default, the scheduling policy called SCHED_FIFO is used, where the task scheduled will run until it finishes or a higher prioritized task preempts it, see [8]. On Windows, the range is different (16-31) with 31 being the maximum time critical real-time priority.

ur_rtde also implements a dashboard client that can be used to control the teach pendant of the robot by communicating with the dashboard server of the UR controller, for which Universal Robots has defined an API, see [9]. The dashboard client is used by the rtde_control interface, but is also useful as a standalone interface.

III. FEATURES

When designing *ur_rtde*, a strong emphasis was placed on designing a library that is easy to understand, use and install, supports multiple program languages, and allows for real-time control of the robot manipulator. An overview of features compared to other libraries for control of UR robots is provided in Table I. *ur_rtde* is compared to urx which is a python interface for UR robots, and urcl which a C++ interface developed by Universal Robots A/S and FZI. The following sections provide some examples of these features.

	urx	urcl	*ur_rtde*
UR script commands[a]	•		•
PyPI package	•		•
Servo commands[b]		•	•
Real-time capability		•	•
Cross-platform			•
Multi-language			•

ours

a) API has UR script commands (eg. moveJ, moveL).
b) Online real-time control of joint positions.

TABLE I: Overview of features provided by different interfaces for Universal Robots manipulators.

A. Ease of Use

Accomplishing simple tasks with *ur_rtde* takes only a few lines of code. Minimal working examples for the RTDE Control, IO and Receive interfaces can be seen in Listings 1 - 3. Furthermore, the API has been designed to closely follow the syntax and parametrization of URScript, such that new users of the library can consult the official URScript documentation to find in-depth examples and explanations, and those experienced with UR programming are already familiarized with the syntax.

```
RTDEIOInterface rtde_io(ip="localhost");
rtde_io.setStandardDigitalOut(7, true);
```

Listing 1: Minimal working example in C++, where RTDE IO is used to set digital output #7 high.

```
from rtde_control import RTDEControlInterface
rtde_c = RTDEControlInterface(ip="localhost")
joint_pos = [0, 0.5, -0.5, 0.5, -0.5, 0.5]
rtde_c.moveJ(joint_pos, vel=1.05, acc=1.4)
```

Listing 2: Minimal working example in Python, where RTDE Control is used to perform a movement in joint space.

This ease of use makes *ur_rtde* well-suited to research applications, where often existing code that has been developed in simulation must be tested on a real robot environment. In this case, interfacing with the real robot is reduced to calling a few RTDE Control commands. Listing 4 shows such an example in a robot learning from demonstration application, where a dynamic movement primitive [10] is trained from a recorded kinesthetic demonstration, and subsequently executed on a UR robot manipulator using RTDE Control.

```
import py.rtde_receive.RTDEReceiveInterface
rtde_r = RTDEReceiveInterface(ip="localhost");
actual_q = rtde_r.getActualQ();
% Convert to MATLAB array of double
actual_q_array = cellfun(@double,
↪  cell(actual_q));
```

Listing 3: Minimal working example of MATLAB support, by taking advantage of MATLAB's Python compatibility interface.

```
from dmp import dmp_joint
import pandas as pd
from rtde_control import RTDEControlInterface

rtde_c = RTDEControlInterface(ip="localhost")

# Parameters
dt = 1.0/500  # 2ms
lookahead_time = 0.1
gain = 300

# Read Demonstration
demo = pd.read_csv("demonstration.csv")
q =
↪  demo[['actual_q_0','actual_q_1','actual_q_2',
↪  'actual_q_3','actual_q_4','actual_q_5']]
time =
↪  demo['timestamp']-demo['timestamp'].iloc[0]
tau = time.iloc[-1]

# Train joint-space DMP
dmp_q = dmp_joint.JointDMP(n_bfs=100)
dmp_q.train(q.to_numpy(), time.to_numpy(), tau)

# Execute DMP
dmp_q.reset()
xi = dmp_q.cs.step(dt, tau)
q, dq, ddq = dmp_q.step(xi, dt, tau)
rtde_c.moveJ(q)
while xi > 1e-20:
    t_start = rtde_c.initPeriod()
    xi = dmp_q.cs.step(dt, tau)
    q, dq, ddq = dmp_q.step(xi, dt, tau)
    rtde_c.servoJ(q, 0.0, 0.0, dt,
    ↪  lookahead_time, gain)
    rtde_c.waitPeriod(t_start)
rtde_c.servoStop()
```

Listing 4: Example showing how to easily integrate existing code that can train and evaluate a Dynamic Movement Primitive with *ur_rtde* in order to execute the primitive on a real UR robot.

B. Support for Multiple Programming Languages

As *ur_rtde* is natively written in C++, it can naturally be used in this language and this is recommended if performance is the utmost priority, as is the case in real-time control applications. A minimal working example in C++ can be seen in Listing 1.

```
import roboticstoolbox as rtb
import numpy as np
import spatialmath as sm
from rtde_control import RTDEControlInterface

rtde_c = RTDEControlInterface("localhost")

# Parameters
dt = 1.0/500  # 2ms
lookahead_time = 0.1
gain = 300

# Cartesian poses
T0 = sm.SE3(0.36177, 0.10525, 0.64767) @
↪  sm.SE3.Ry(np.pi)
T1 = sm.SE3(0.36177, -0.22975, 0.3162) @
↪  sm.SE3.Ry(np.pi/2)

# Cartesian trapezoidal velocity trajectory
t = np.arange(0, 2, dt)
T_trap = rtb.tools.trajectory.ctraj(T0, T1, t)
v0, theta0 = sm.base.tr2angvec(T0.R)
rtde_c.moveL(T0.t.tolist() +
↪  (v0*theta0).tolist())
for T_i in T_trap:
    t_start = rtde_c.initPeriod()
    v_i, theta_i = sm.base.tr2angvec(T_i.R)
    p_i = T_i.t.tolist() + (v_i *
    ↪  theta_i).tolist()
    rtde_c.servoL(p_i, 0.0, 0.0, dt,
    ↪  lookahead_time, gain)
    rtde_c.waitPeriod(t_start)
rtde_c.servoStop()
```

Listing 5: Example of simple integration between the third-party *Spatial Maths for Python* and *Robotics Toolbox for Python* libraries and RTDE Control.

It is, however, also possible to use the library directly in Python. An example of this can be seen in Listing 2. The availability of Python bindings also makes it very easy to integrate *ur_rtde* with other packages and libraries. For instance, the *Robotics Toolbox for Python*[3] [11] and *Spatial Math for Python*[4] libraries can be used for coordinate transformations and trajectory generation, and RTDE Control easily allows for sending of the interpolated poses to a UR manipulator using *servo* commands, as shown in Listing 5.

An additional advantage of the Python bindings is that, as MATLAB supports interfacing with Python code directly, it is possible to integrate *ur_rtde* with MATLAB code and thereby also with Simulink. As MATLAB and Simulink are

[3]https://github.com/petercorke/robotics-toolbox-python

[4]https://github.com/bdaiinstitute/spatialmath-python

widely popular platforms and have strong support for control systems design, this opens up for many possibilities when it comes to studying the effects of such control systems on a real robot manipulator. A simple example showing the usage of RTDE Receive through MATLAB's Python integration can be seen in Listing 3.

C. Real-time Control

A major focus when designing the library was to provide performance-focused interfaces that allow for real-time control of UR manipulators. As will be shown in section IV; this is in contrast to other alternatives, which suffer from higher latency that deteriorates the performance of the controller.

Listing 6 shows an example of how to perform real-time control of the robot in order to follow a circular trajectory, while adapting to changes in speed scaling.

```cpp
#include <ur_rtde/rtde_control_interface.h>
#include <ur_rtde/rtde_receive_interface.h>

double time_counter = 0.0;
std::string robot_ip = "localhost";
double freq = 500.0; // Hz
double dt = 1.0 / freq; // 2ms
uint16_t flags =
  ↪ RTDEControlInterface::FLAG_VERBOSE |
  ↪ RTDEControlInterface::FLAG_UPLOAD_SCRIPT;
int ur_cap_port = 50002;
int rt_receive_priority = 90;
int rt_control_priority = 85;

RTDEReceiveInterface rtde_receive(robot_ip,
freq, {}, false, false, rt_receive_priority);
RTDEControlInterface rtde_control(robot_ip,
freq, flags, ur_cap_port, rt_control_priority);
// Set application realtime priority
RTDEUtility::setRealtimePriority(80);

while (time_counter > 3)
{
  auto t_start = rtde_control.initPeriod();
  double speed_scaling_combined =
  ↪ rtde_receive.getSpeedScalingCombined();
  std::vector<double> servo_target =
  ↪ getCircleTarget(init_pose, time_counter);
  rtde_control.servoL(servo_target, vel, acc,
  ↪ dt, lookahead_time, gain);
  rtde_control.waitPeriod(t_start);
  time_counter += dt * speed_scaling_combined;
}
```

Listing 6: Example of RTDE Control used for real-time tracking of a circular target trajectory in the XY-plane. By reading the value of the speed scale, the motion is adjusted in real time to adapt to changes in the speed slider performed by either the user or the safety system.

IV. PERFORMANCE EVALUATION

To benchmark the real-time performance of the library when it comes to control of the robot manipulators, a test with a continuous circular motion was conducted on a UR10e robot. The robot was commanded to a target position that follows a circle based on Eq. (1) by using the C++ code shown in Listing 6.

$$\begin{aligned} x &= x_{init} + r \cdot \cos(2\pi f t) \\ y &= y_{init} + r \cdot \sin(2\pi f t) \\ z &= z_{init} \end{aligned} \tag{1}$$

In Eq. (1), r represents the radius of the circle and f is the frequency of circular motion. The robot followed the circle in a loop for 3 seconds with and without real-time priority. Meanwhile, the test PC (see Table II) was subjected to stress using the Unix command line tool *stress*, which imposes CPU, memory, I/O, or disk stress on the operating system. The robot data was recorded while the test was conducted.

HP EliteDesk 800 G2:	
CPU	Intel® Core i5 @ 3.2GHz
RAM	16 GB
OS	Ubuntu 22.04 (PREEMPT-RT)
Network	Intel I219LM Gigabit NIC

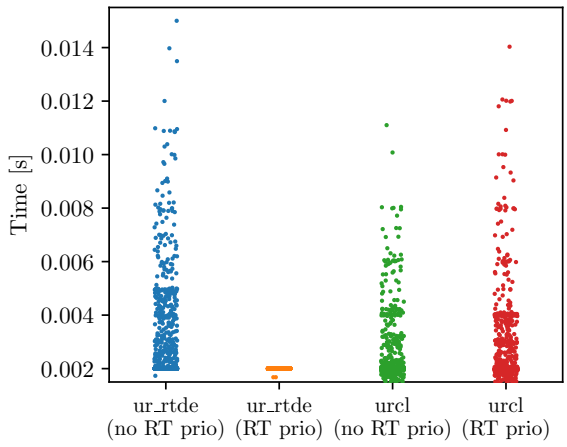
TABLE II: Test PC specifications

The test was first conducted with the *ur_rtde* library. The xy-position of the robot during the test can be seen in Fig. 2c. It is clear that without real-time priority (blue) the robot cannot maintain the xy-position, while with the real-time priority is stays on the circle (orange).

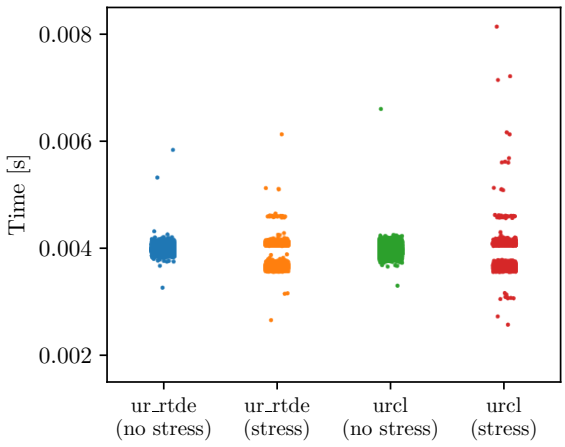
The cycle-time of the control loop for *ur_rtde* can be seen on Figure 2a, which clearly shows that with no real-time priority (blue) the 2 ms time slice is violated multiple times: 296 samples are above 2.5 ms, while with the real-time priority (orange) it stays around the 2 ms and 0 samples are above the chosen cut-off of 2.5 ms.

The same test was performed against the urcl library (Universal Robots Client Library) [12]. The xy-position of the robot during the test can be seen in Figure 2d, it is clear that even with RT priority the robot cannot follow the target position and looking at the cycle-time in Figure 2a, 160 samples are above 2.5 ms without real-time priority (green), while with real-time priority (red) 279 samples are above.

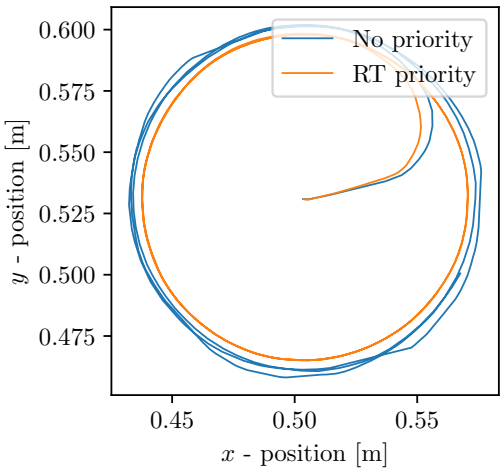
Finally, the Round-trip time (RTT) of *ur_rtde* and urcl was measured by setting the speed slider of the robot to a fixed value and then waiting for that change to be read by the interface. The RTT was measured with and without stress on the test PC. The expected RTT is: $RTT = 2 \cdot$Propagation delay, where the propagation delay corresponds to the cycle-time of 2 ms, so the expected RTT is 4 ms. Figure 2b shows the RTT with and without stress for *ur_rtde* and urcl. For *ur_rtde* without stress (blue), 2 samples were above the chosen cut-off of 4.5 ms and with stress (orange) 21 samples were above 4.5 ms. For urcl, without stress (green), 1 sample was above the chosen cut-off of 4.5 ms and with stress (red) 35 samples were above 4.5 ms.



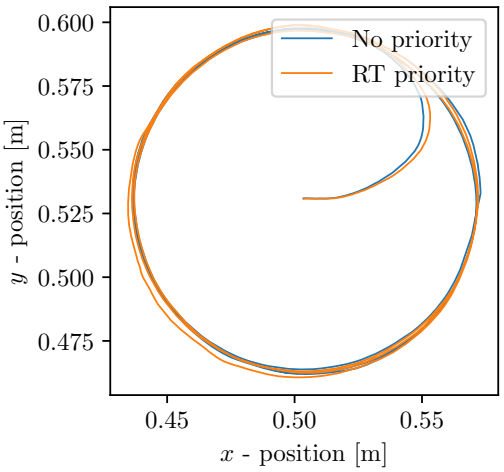
(a) Cycle-time - *ur_rtde* vs. urcl



(b) Round-trip time (*ur_rtde* vs. urcl)



(c) *ur_rtde* - Robot position XY (RT vs. No RT)



(d) urcl - Robot position XY (RT vs. No RT)

Fig. 2: Performance comparison between *ur_rtde* and urcl with and without realtime priority.

V. USERS

Since its release in March 2020, *ur_rtde* has been adopted by a wide variety of users in different use cases. This section will provide a few examples to illustrate how *ur_rtde* can contribute to different applications involving Universal Robots.

A. Industry: RiACT ApS

RiACT makes software for robots and has developed a skill-based robot control platform for simple and extensible robot applications. They state the following about *ur_rtde*:

> "*ur_rtde* allows us to control and monitor the arm's activity in Python, thus allowing us to integrate multiple UR arms in our ecosystem. This reduces the development time and enables us to focus more on delivering the solution that RiACT set out to provide for the industry."

B. Industry: CETONI GmbH

CETONI uses UR robots to automate laboratory processes and wanted an interface for them to be integrated into the CETONI Elements software. CETONI Elements is a Qt-based Windows software written in C++. The *ur_rtde* C++ library was the perfect solution for this. In contrast to the Universal Robots Client Library, the *ur_rtde* library can also be compiled and used under Windows. Thanks to the good and comprehensive documentation, integration was quick and easy to realise. The Python bindings were a major advantage. These make it possible to quickly try out and test functions and features in Python and then integrate them. Because the library is hosted as open source on GitLab, it was easy for us to add missing functions, such as jogging the robot in different reference frames.

The result is a complete integration of the UR robots in CETONI Elements, see Fig. 3. The robots can be controlled without using the teach pendant. All functions such as manual jogging, teaching or defining reference frames

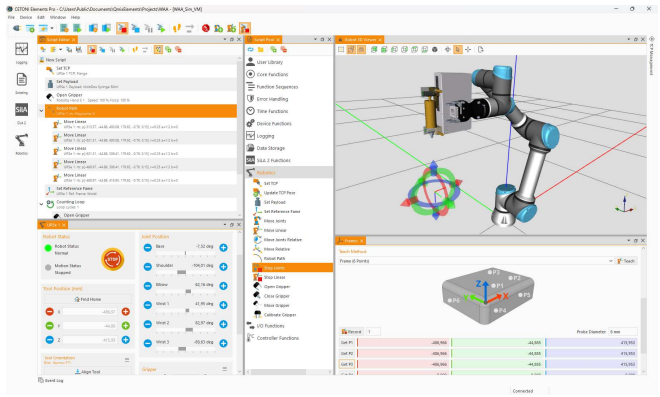


Fig. 3: CETONI Elements - Robotics plugin based on *ur_rtde*

are available. Integration into the CETONI Elements script system means that complex sequences can be conveniently programmed on the PC using drag & drop. All important robot parameters, such as forces, currents or positions, can be recorded in the software via the *ur_rtde* library and visualised in diagrams.

C. Research: MOPS

The *ur_rtde* library has been adopted by MOPS, *A modular and open platform for surgical robotics research* [13], where it is integrated in a more complicated system that also uses ROS, the Robot Operating System, for communication and device drivers. Using *ur_rtde* together with a preemptive real-time kernel to implement the control loop of the Universal Robot manipulators used in the system allows MOPS to guarantee better real-time performance than using, e.g., ros_control or urcl, would. The MOPS system has been used, among other things, to implement autonomous control of surgical robots based on suturing primitives learned from demonstration [14].

VI. CONCLUSIONS

This paper has presented *ur_rtde* a library for interfacing with Universal Robots manipulators through the Real Time Data Exchange interface. The library focuses on ease of installation and use, multi-language support, and real-time low level control of the robots. It is implemented in C++, with included Python bindings and distributed through PyPi, and additionally supports MATLAB through its Python interface. The design is modular, with separate interfaces for control, data reading, and IO handling, which makes the library highly flexible. We have shown multiple working examples of different use cases both in research and industry. A comparison against urcl shows that *ur_rtde* outperforms it when it comes to real-time control use cases, particularly when a preemptive real-time kernel is used. The real-time performance of ur_rtde can be improved further by using clock of the received data to govern the control loop timing.

ACKNOWLEDGMENT

We would like to thank all people that have contributed to *ur_rtde* and RiACT ApS and CETONI GmbH for letting us share their thoughts on *ur_rtde* in this paper. Parts of the development and evaluation of *ur_rtde* were supported by the SDU I4.0 Lab.

REFERENCES

[1] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin, "The Franka Emika robot: A reference platform for robotics research and education," *IEEE Robotics & Automation Magazine*, vol. 29, no. 2, pp. 46–64, 2022.

[2] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the KUKA lightweight robot," in *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications How to Modify and Enhance Commercial Controllers (ICRA 2010)*, 2010, pp. 15–21.

[3] Universal Robots A/S, "Real-time data exchange (RTDE) guide," 2019. [Online]. Available: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/

[4] T. Andersen, *Optimizing the Universal Robots ROS driver.* Technical University of Denmark, Department of Electrical Engineering, 2015.

[5] Universal Robots A/S and FZI Forschungszentrum Informatik, "Universal robots ROS driver," 2024. [Online]. Available: https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

[6] ——, "Universal robots ROS2 driver," 2024. [Online]. Available: https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver

[7] International Organization for Standardization, "Information technology - programming languages: C++," Vernier, Geneva, Switzerland, 2011. [Online]. Available: https://www.iso.org/standard/50372.html

[8] M. Kerrisk, "sched(7) — linux manual page," 2023. [Online]. Available: https://man7.org/linux/man-pages/man7/sched.7.html

[9] Universal Robots A/S, "Dashboard server remote control interface - e-series," 2022. [Online]. Available: https://man7.org/linux/man-pages/man7/sched.7.html

[10] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[11] P. Corke and J. Haviland, "Not your grandmother's toolbox – the robotics toolbox reinvented for python," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 357–11 363.

[12] Universal Robots A/S, "Universal robots client library," 2020. [Online]. Available: https://github.com/UniversalRobots/Universal_Robots_Client_Library

[13] K. L. Schwaner, I. Iturrate, J. K. H. Andersen, C. R. Dam, P. T. Jensen, and T. R. Savarimuthu, "MOPS: A modular and open platform for surgical robotics research," in *2021 International Symposium on Medical Robotics (ISMR)*. IEEE, 2021, pp. 1–8.

[14] K. L. Schwaner, I. Iturrate, J. K. H. Andersen, P. T. Jensen, and T. R. Savarimuthu, "Autonomous bi-manual surgical suturing based on skills learned from demonstration," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 4017–4024.