Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Comparison of KVP and RSI for Controlling KUKA Robots Over ROS ⋆

M. H. Arbo * **I. Eriksen** * **F. Sanfilippo** ** **J. T. Gravdahl** *

* *Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), 7491 Trondheim, Norway*
** *Department of Engineering Sciences, University of Agder (UiA), 4879 Grimstad, Norway*

Abstract: In this work, an open-source ROS interface based on KUKAVARPROXY for control of KUKA robots is compared to the commercial closed-source Robot Sensor Interface available from KUKA. This comparison looks at the difference in how these two approaches communicate with the KUKA robot controller, the response time and tracking delay one can expect with the different interfaces, and the difference in use cases for the two interfaces. The investigations showed that the KR16 with KRC2 has a 50 ms response time, and RSI has a 120 ms tracking delay, with negligible delay caused by the ROS communication stack. The results highlight that the commercial inferface is more reliable for feedback control tasks, but the proposed interface gives read and write access to variables on the controller during execution, and can be used for simple motion and tooling control.

Keywords: Robotics Technology, Motion Control Systems, Robots Manipulators

1. INTRODUCTION

1.1 Motivation and Outline

ROS is an open-source middleware for writing robot software, it provides a message-passing structure for interprocess communication. Packages written in ROS are transferrable from one robot setup to another, and the large international userbase provides packages ranging from indoor navigation, to robot simulation, to reference frame calculation. ROS-Industrial (ROS-I) is an open-source project aimed at extending ROS to new manufacturing applications. ROS-I provides a standardization of package structures for writing packages aimed at industrial robotics. Chitta et al. (2017) provides the `ros_control` framework for designing interfaces for controlling robot hardware from ROS. This has resulted in interfaces for different industrial robots including vendors like ABB, Adept, Fanuc, Motoman, and Universal Robots. Extensive research work has also gone into creating ROS drivers for KUKA robots using the RSI interface.

Controlling industrial robots from an external computer using ROS differs greatly from the classical proprietary robot programming methods provided by industrial robot vendors. As such, the commercial interfaces available for use with ROS often rely on the user writing custom programs on the robot controller to run for each new application. In this article an interface for communicating directly to global variables on the robot controller is used to explore an alternative communication vector for KUKA robot cells, this is done using the KUKAVARPROXY (KVP) server and the *BoostCrossCom* C++ interface for connecting to the server. This approach requires minimal knowledge of the *KUKA Robot Language* (KRL), the robot programming language for KUKA robot cells.

The main contribution of this work is a free and open-source KVP-based ROS package for controlling KUKA industrial robots. The work is a continuation of the work of Eriksen (2017b) and Sanfilippo et al. (2015a), and compares the KVP-based control method with the RSI-based control method for use with ROS. The results in this article also verify the response time and tracking delay found by Lind et al. (2010).

The article is split into four sections. The first section gives the motivation and related research. Section 2 describes the robotics lab as well as the *KUKA Robot Controller 2* (KRC2) and the two interfaces used. Section 3 describes the experiments comparing response time, tracking delay, feedback control with velocity-resolved closed-loop inverse kinematics, and an example to show the benefit of having access to the global variables in the controller. Section 4 gives a qualitative comparison of the interfaces and discusses the results, the final section is the conclusion.

1.2 Related Research

KUKA offers three interfaces for control and communication with a robot using an external computer: *Robot Sensor Interface* (RSI), *Ethernet KRL Interface* (EKI), and *Fast Research Interface* (FRI). RSI is used in this article and will be further discussed in the following sections. EKI is an interface intended for TCP/IP data communication between the computer and an external computer. As this data may include motion commands, it allows for motion

⋆ The work reported in this paper was based on activities within centre for research based innovation SFI Manufacturing in Norway, and is partially funded by the Research Council of Norway under contract number 237900.

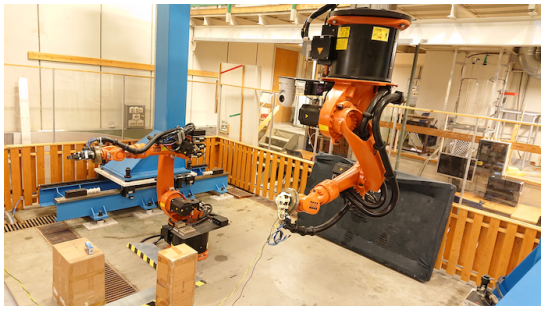

Fig. 1. Thrivaldi is a floor-mounted and a gantry-mounted KR16, currently with a pneumatic SCHUNK gripper (RH9010), and an IMU attached, respectively.

control. EKI is generally less expensive than RSI, but not as viable for feedback control. EKI requires the KRL program running on the KRC to specify the connection to establish, and variables that will be transmitted. FRI is a real-time interface supporting control modes from joint impedance control to joint position control, and may be used to read and write to variables specified in the KRL program, but it is only available for the KUKA lightweight manipulator series.

KVP was developed by IMTS s.r.l. as a freely distributed server that runs on the Windows portion of the KUKA Robot Controller. The open-source communication library *JOpenShowVar*, was created by Sanfilippo et al. (2015a) as a Java-based middleware for communication and control of the robot. It was created as an open-source alternative to current KUKA control packages. *JOpenShowVar* has been used in research on active heave compensation for offshore crane operations (Sanfilippo et al., 2015b), robotic welding of tubes (Bredvold, 2015), sensorless admittance control in human-robot interaction (Yao et al., 2018), and for robot-assisted 3D vibrometer measurements (Venugopal, 2018). The open-source nature of *JOpenShowVar* allowed for the development of *BoostCrossCom*, a minimal C++ library for communicating with the KVP server. *BoostCrossCom* was developed by Njåstad (2015) and further explored in Njåstad and Egeland (2016).

2. ROBOT SYSTEM AND SETUP

2.1 Thrivaldi

The lab, see Fig. 1, is situated at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU) and consists of two 6 degrees-of-freedom KUKA KR16 robots, where one is attached to a GÜDEL gantry crane giving it 9 degrees-of-freedom. The lab was named after Thrivaldi, a 9 headed giant from Norse mythology. Each robot is controlled with a KRC2 cabinet. The gantry crane is connected to the KRC2 and set up as synchronous external axes controlled directly by the cabinet. Both robots have a SCHUNK FTC-50-80 force/torque sensor attached, and a pneumatic tool changer. The project's GitHub repositories (Eriksen, 2017a) describe the lab setup and includes ROS drivers for controlling synchronous external axes with an RSI interface as well as the KVP packages.

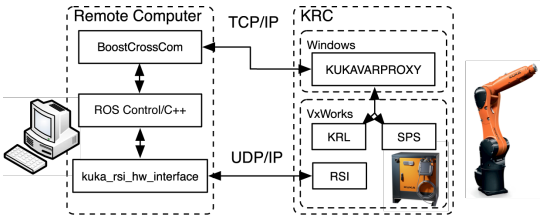

Fig. 2. RSI communicates directly with VxWorks, and KVP goes through a server on Windows to KRL and SPS. Both paths can be used simultaneously.

2.2 Software and Hardware

The KRC encompasses the power supply, servo controllers, control computer, I/O, etc. The control computer in the KRC2 is a standard x86 computer with a single core Intel Celeron CPU. The KRC2 uses VxWin, a KUKA specific real-time operating system based on VxWorks that runs both VxWorks and Windows XP. The KRC2 has an interpolation cycle (IPOC) of 12 ms. Within each IPOC, the KRL interpreter runs, I/O devices are updated, the software programmable logic controller (*SPS*) runs, and Windows tasks are executed. The KUKA Robot Language is KUKA's Pascal-based proprietary programming language used for executing robot motions. The KRC prioritizes execution of any running KRL program over handling Windows requests.

To understand the KVP-based interfaces, we only require the system variables `ADVANCE`, `OV_PRO`, and the motion command `PTP`. `ADVANCE` defines how many commands ahead the motion planner should look when performing path smoothing. `OV_PRO` is the override speed percentage, the percentage of maximum permissible speed when executing a motion command. Note that the override speed is not necessarily the speed that will be used, but the speed limit in the motion planner. `PTP` is a point-to-point motion command to either a pose defined in Cartesian space or joint space. The `PTP` command executes a trapezoidal motion between the current pose to the desired pose. If the option `C_PTP` is supplied to the `PTP` command, and `ADVANCE>0`, the motion planner will start to move towards the next desired pose as soon as the robot is sufficiently close to the current desired pose. This smooths the motion, making it more efficient and faster, at the cost of positional accuracy. KRL supports defining workspaces that can constrain motion or I/O commands to only be executed in specific areas in the robot's workspace.

2.3 The Two Interfaces

The two different interfaces have two different access vectors to the robot as illustrated in Fig.2.

KUKA's RSI is intended for sensor-assisted motion and data exchange, the idea is to use external sensors to correct the position of the robot independent of any running KRL commands.

In the `kuka_experimental` package, RSI has been used to create an interface to control KUKA robots via ROS. The ROS interface uses joint position commands to control the robot. The RSI position corrections are intended for minor joint or Cartesian position corrections and work at

a lower level than the KRL interpreter. This means that it has a separate configuration for the maximum position corrections (effectively the override speed), and neither adheres to workspace limitations, nor perform trapezoidal motion between current and desired position. For ROS independent response time testing in Sec. 3.1, a bare-bones C++ interface was created from the XML header files in `kuka_experimental`.

KUKAVARPROXY (KVP) is a multi-client server that runs in Windows on the KRC and gives TCP/IP access to external computers. KVP communicates with the KRC using the *CrossCom* library, and can read and write to global variables. To move the robot we have a KRL program running on the KRC2 with a loop that executes a motion command with a KVP writable joint axis variable. As the KVP server runs in the lower-priority Windows OS, a stochastic communication delay may occur when the VxWorks tasks are prioritized.

The project repositories provides `kuka_kvp_hw_interface`, a ROS hardware interface that uses *BoostCrossCom* to communicate with the KVP server. The package has a node for reading joint states independently of any running KRL program, a joint position controller using the simple KRL program, and ROS services to read and write to global variables on the KRC.

3. EXPERIMENTAL RESULTS

3.1 Response Time

To test the response time independently of the interface used, an Arduino Micro with an MPU-6050 IMU is attached to the end-effector of the gantry-mounted robot. The time from a 30° movement on joint A5 is commanded from the external computer until the IMU senses it is used as the response time. After each motion, the robot is given 10 s to settle such that any vibrations caused by the motions does not affect the subsequent measurements. This means that the robot controller must also overcome the static friction in the joint. The test was performed 5000 times for each interface. The tests were run without using the ROS stack, only TCP/IP for the KVP interface and UDP for the RSI interface. The Arduino was connected over SPI to USB. Timing was performed using the Boost `cpu_timer` class in C++, and the tests were performed on an Intel Xeon CPU E5-1650 running Ubuntu 16.04.

Fig. 3 shows the results as a histogram where each bin is the length of one IPOC (12 ms). RSI consistently uses 4-5 IPOCs to overcome the static friction in the joint, and KVP uses longer. Statistics of the results are given in Tab. 1. KVP had tests in the hundreds of milliseconds range as the KVP server is a Windows task with lower priority than VxWorks tasks.

The KVP-based interface has 2-3 IPOCs longer response time than RSI. In simple write/read experiments over KVP to arbitrary variables, the same delay can be observed and is expected to be a limitation stemming from accessing global variables in the KRL interpreter by a server running on the Windows part of the KRC.

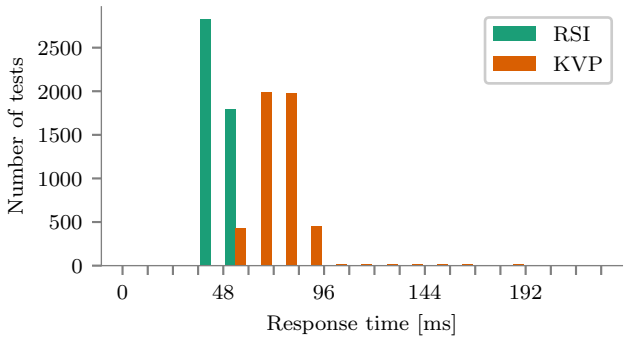

Fig. 3. Response time for the two different interfaces. Each bin of the histogram is the length of one IPOC (12 ms). Note that RSI is only in the 4th or 5th IPOC whereas KVP has longer delay.

Table 1. Response Time for 5000 Tests (in ms)

Library	Min	Max	Mean	Median	Std.Dev.
KVP	49.98	268.72	76.15	77.62	18.59
RSI	31.10	55.82	46.86	46.87	3.88

3.2 Tracking Delay

The tracking delay when using ROS is explored by creating a simple node publishing a sinusoidal signal on joint A3. The desired joint angle is

$$q_{A3,\mathrm{des}}(t) = A\sin(\omega t) \tag{1}$$

where $A = 0.2$ rad, and $\omega = 0.2$ rad/s. The tracking delay is the time between a desired joint command is sent until it is achieved, as reported by ROS.

As noted by Sanfilippo et al. (2015a), the KRL program used for the KVP-based interface can be designed to suite the needs of the particular application. To generalize for a wide variety of use cases, the chosen KRL program is a simple loop using the `PTP` command. There are three design parameters to the `PTP` command: `OV_PRO` that governs the override speed, `ADVANCE` that governs the look-ahead motion planner, and `C_PTP` that activates path smoothing.

In Fig. 4, the RSI-based interface was used. There is a 120 ms tracking delay from a command is given until the robot achieves the same joint angle. These results are similar to that of Lind et al. (2010) where tracking delay was tested with RSI without ROS. We therefore assume that any delay caused by the ROS communication stack and the reported timestamps of the rosbag to be negligible with respect to the tracking delay.

In Fig. 5, the KVP-based interface was used with 100% override speed, an `ADVANCE` of 1, and no `C_PTP`. The robot moves in a stop-and-go motion. As the override speed is very high, the robot performs a trapezoidal motion to the desired pose and then stops when the pose is reached. After this the robot requires some time to start a new motion, resulting in a stop-and-go motion.

In Fig. 6, `C_PTP` is activated and the override speed is set to 30% . This is approximately the speed at which the robot is required to move by the commanded signal, and

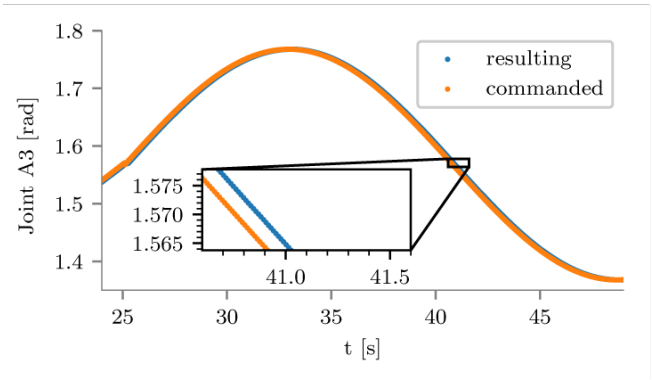

Fig. 4. Joint A3 moving with a sinusoidal motion using RSI.

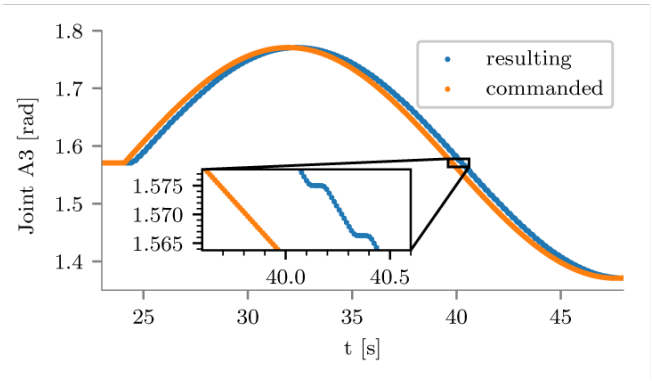

Fig. 5. Joint A3 moving with a sinusoidal motion using KVP at 100% override speed without `C_PTP`.

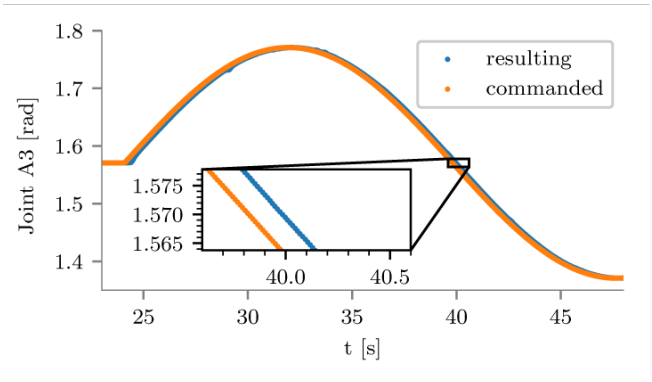

Fig. 6. Joint A3 moving with a sinusoidal motion using KVP at 30% override speed with `C_PTP`.

we see that we can achieve a behavior that is close to RSI, exhibiting approximately 150 ms delay.

3.3 Closed-Loop Inverse Kinematics Example

In this example the end-effector is to follow a 3D Lissajous trajectory defined by

$$\boldsymbol{p}_{des}(t) = 0.3 \begin{bmatrix} \sin(n_x \omega t) - 1.4 \\ \sin(n_y \omega t) + 0.7 \\ \sin(n_z \omega t) + 1.0 \end{bmatrix} \tag{2}$$

where $\omega = 0.02$, $n_x = 5$, $n_y = 2$, and $n_z = 3$. The trajectory w.r.t. the robot in its initial position is visualized in Fig. 7. The error between the current position and the desired position is described by

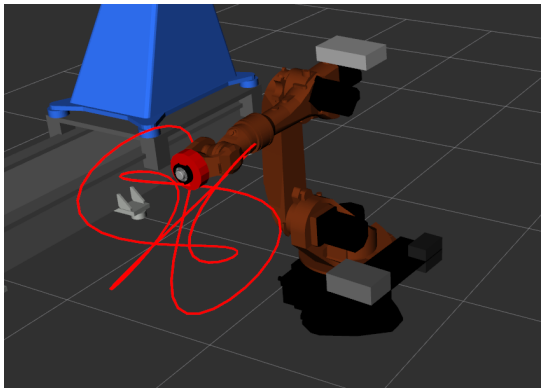

Fig. 7. Visualization of the 3D Lissajous trajectory the robot is to follow.

$$\boldsymbol{e}(t, q) = \boldsymbol{p}(q) - \boldsymbol{p}_{des}(t) \tag{3}$$

which we desire to converge to zero.

To do this we apply CASCLIK (Arbo et al., 2019), a CasADi-based closed-loop inverse kinematics Python framework. The framework allows for formulating controllers for multiple constraint-based tasks and solves them either using the Moore-Penrose pseudoinverse, or as constraints to an optimization problem. In this example we apply the quadratic programming controller, and CASCLIK formulates the constraint

$$\boldsymbol{J}\dot{\boldsymbol{q}}_d = -K\boldsymbol{e} - \frac{\partial \boldsymbol{e}}{\partial t} \tag{4}$$

and the cost

$$c = \dot{\boldsymbol{q}}_d^T \dot{\boldsymbol{q}}_d \tag{5}$$

where $\boldsymbol{J}$ is the task Jacobian describing the partial derivative of the end-effector position, $\boldsymbol{p}$, with respect to the joint variable, $\boldsymbol{q}$. The task Jacobian is automatically generated from the KR16 URDF, $\dot{\boldsymbol{q}}_d$ is the desired joint velocity that will be applied, $K = 50$, and the cost c ensures that the resulting joint velocity remains bounded. The desired joint velocity is integrated and position commands are sent to the robot.

In Fig. 8 we see the Euclidean norm of the error for four different experiments: using the RSI interface, using KVP with 100% override speed and being lucky with the computer and the KRC2 synchronizing, using KVP with 100% override speed but not achieving synchronization, and using KVP with automatic tuning of the override speed. If synchronization is achieved the KVP with 100% override speed is close to the tracking error of the RSI interface, but stochastic delays in communication causes errors that makes us intermittently lose this synchronization. With automatic tuning of the override speed according to the desired joint velocity we can reduce the occurrences of the stop-and-go motion.

The cyclic tracking error is a result of not having an integrating effect on the controller, the tracking delay, and the linearization assumption inherent in (4). Tuning the gain K can decrease this error.

3.4 Benefits of Accessing Global Variables

In this example the robot moves a stapler from one cardboard box to another using either RSI for motion, or KVP. The robot moves between four points defined in

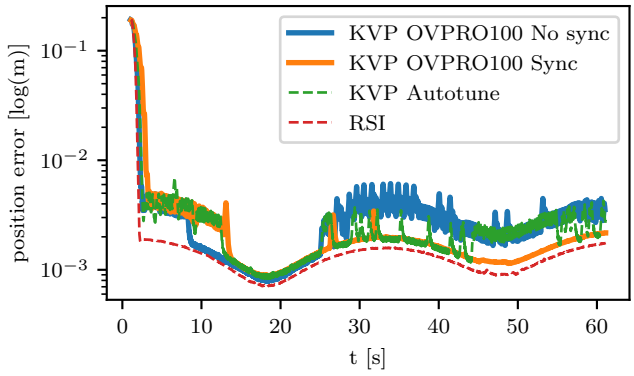

Fig. 8. Tracking error of the closed-loop inverse kinematics example.

Fig. 9. KR16 ready to transfer the stapler.

joint space, stopping momentarily above the cardboard boxes to release and grasp the stapler. With RSI, the trapezoidal motion profile in joint space between any two points is created by the external computer and commanded as small angle corrections. With KVP, only the end position is commanded and the system relies on the internal trapezoidal motion planner. The gripper is controlled using `kvp_variable_interface` and writing to the relevant global variable. SPS and KUKA workspaces are used to ensure that the gripper can only be activated when the end-effector is in designated workspaces above the boxes and is in effect regardless of which KRL program is running.

In Fig. 10, the Cartesian motion of the end-effector with respect to time is given for when the task is performed with the KVP-based interface. In Fig. 11, the Cartesian motion of the end-effector with respect to time is given for when the task is performed with the RSI-based interface. In this case the motion is commanded using RSI and the gripper is controlled using ROS services from the `kvp_variable_interface_node`. Note that with only the

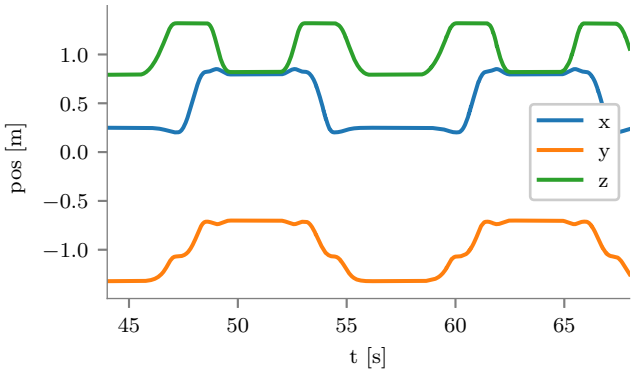

Fig. 10. Cartesian coordinates of the end-effector when transferring stapler using KVP.

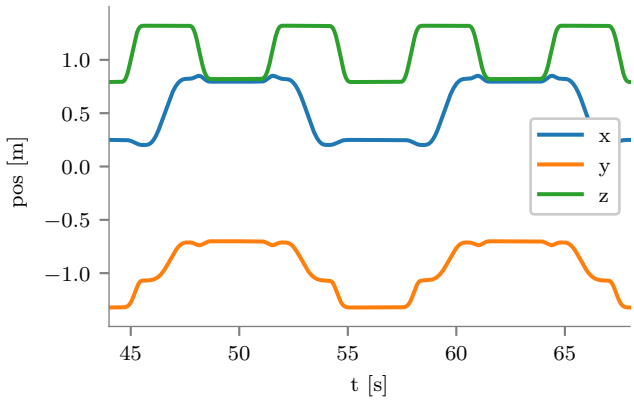

Fig. 11. Cartesian coordinates of the end-effector when transferring stapler using RSI.

KVP-based interface, the motion is slightly smoother and slower as the KRL program is running with an override speed of 30% and uses the internal trapezoidal motion planner of KUKA. The RSI-based interface uses a trapezoidal motion created in the external computer that does not exactly match the KVP based interface. The overall curvature of the two are similar, and the example demonstrates usage of both KVP and RSI at the same time for motion and tool control.

4. DISCUSSION

The qualitative comparison between the two interfaces is summarized in Tab. 2. RSI is a commercial real-time interface which demands care and consideration of the programmer, both in terms of safety handling and keeping the real-time communication requirements. The KVP-based interface is an open-source interface with natural limitations on how fast and accurate one can expect the control to be, both in terms of timing reliability and path accuracy when `C_PTP` is active as demonstrated in the experiments.

Lind et al. (2010) attributes the 120 ms tracking delay of the RSI interface to motion buffers in the system. Although KVP uses point-to-point motion in a different manner than RSI, it also exhibits a tracking delay of the same order of magnitude, suggesting that the hypothesis

Table 2. Comparison of the RSI and KVP based ROS interfaces

RSI	KVP
Commercial	Open-Source
Real-time	Stochastic delay
Small angle corrections	Uses KRL motion planner
Ignores workspaces	Uses workspaces
Specific KRL for new tools	Read and write global variables

is correct. This has not been examined on the KRC4, but the tracking delay is expected to be lower.

Both interfaces provide joint position control. The stochastic delay in the KVP-based interface may introduce errors in the tracking accuracy when the external computer performs finite differences to approximate joint-velocity control.

The different behavior of RSI- and KVP-based ROS interfaces suggests the possibility of different usage scenarios. The KVP-based interface is closer to programming in KRL. It provides the same layer of abstraction from motion profile planning that may stop novice users of the robot system, and it does not require timely responses to the external computer. This means that one can restart a program in ROS without having to restart the program on the KRC, requiring less interaction with the KRC. The differences are summarized in Table 2.

This layer of abstraction can be problematic for advanced users who want to perform sensor feedback control tasks. To them, RSI is more appropriate. However, the KVP-based interface can be used to control tooling and other SPS-based aspects of the system while the RSI interface is used for motion. The KVP-based interface also allows for logging and plotting the robot motion when executing KRL programs by utilizing the `kvp_joint_state_node`.

5. CONCLUSION

This article compares two different ROS interfaces for controlling KUKA robots using ROS: an RSI-based interface, and a new KVP-based interface introduced by the authors. The commercial interface is more reliable for feedback control tasks, and the open-source interface can approach similar performance, but unreliably. The article also recreates the timing results of Lind et al. (2010), showing an approximately 50 ms response time and 120 ms tracking delay when using RSI. The KVP-based interface was 2-3 IPOCs slower, a delay associated with the KVP-server running on the Windows part of the KRC2.

Without modifying any KRL program running on the robot, the KVP-based interface can read and write to any global variables on the KRC2. This can be beneficial in Industry 4.0, as demonstrated by Øvern (2018). One can create a fully open-source digital twin of a KUKA robot cell using tools in the ROS community such as Gazebo and RViz. The KVP-based interface can monitor and log the behavior of existing robot cells without interfering with any currently running KRL programs. However, the stochastic nature of the KVP server's access to the global variables limits the useability of its approach in real-time monitoring and control.

Although this article considers an aging KUKA robot platform, the results are important considerations for control theory applications on industrial robots in general. Both the response time and the tracking latency are aspects that can negatively impact the transition from academic results to industrial application. For rapid system integration, the less reliable but more open interface may reduce the programming effort. Documenting and describing the nature of different control interfaces is essential for using and improving upon them.

REFERENCES

Arbo, M.H., Grøtli, E.I., and Gravdahl, J.T. (2019). CASCLIK: CasADi-Based Closed-Loop Inverse Kinematics. URL `https://arxiv.org/abs/1901.06713`.

Bredvold, S.H. (2015). *Robotic Welding of Tubes with Correction from 3D Vision and Force Control*. Master's thesis, Norwegian University of Science and Technology.

Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Rodríguez Tsouroukdissian, A., Bohren, J., Coleman, D., Magyar, B., Raiola, G., Lüdtke, M., and Fernández Perdomo, E. (2017). ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*.

Eriksen, I. (2017a). Itk-thrivaldi github project. URL `https://github.com/itk-thrivaldi/`.

Eriksen, I. (2017b). *Setup and Interfacing of a KUKA Robotics Lab*. Master's thesis, Norwegian University of Science and Technology.

Lind, M., Schrimpf, J., and Ulleberg, T. (2010). Open Real-Time Robot Controller Framework. *2010 3rd CIRP Conference on Assembly Technology and Systems*, (June 2010), 13–18.

Njåstad, E.B. and Egeland, O. (2016). Automatic touch-up of welding paths using 3d vision. *IFAC-PapersOnLine*, 49(31), 73 – 78. 12th IFAC Workshop on Intelligent Manufacturing Systems IMS 2016.

Njåstad, E.B. (2015). *Robotic Welding with Correction from 3D camera*. Master's thesis, Norwegian University of Science and Technology.

Øvern, A. (2018). *Industry 4.0 - Digital Twins and OPC UA*. Master's thesis, Norwegian University of Science and Technology.

Sanfilippo, F., Hatledal, L.I., Zhang, H., Fago, M., and Pettersen, K.Y. (2015a). Controlling kuka industrial robots: Flexible communication interface jopenshowvar. *IEEE Robotics Automation Magazine*, 22(4), 96–109.

Sanfilippo, F., Hatledal, L.I., Zhang, H., Rekdalsbakken, W., and Pettersen, K.Y. (2015b). A wave simulator and active heave compensation framework for demanding offshore crane operations. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1588–1593.

Venugopal, S.K. (2018). *Robot assisted 3D vibrometer measurements with one vibrometer*. Master's thesis, Technische Universität Braunschweig.

Yao, B., Zhou, Z., Wang, L., Xu, W., Liu, Q., and Liu, A. (2018). Sensorless and adaptive admittance control of industrial robot in physical human-robot interaction. *Robotics and Computer-Integrated Manufacturing*, 51, 158 – 168.