JOpenShowVar: an Open-Source Cross-Platform Communication Interface to Kuka Robots*

F. Sanfilippo, L. I. Hatledal and H. Zhang
Dept. of Maritime Technology and Operations, Aalesund University College
Postboks 1517, 6025 Aalesund, Norway
{fisa, laht, hozh}@hials.no

M. Fago
IMTS S.r.L. Company
Taranto, Italy
massimiliano.fago@gmail.com

K. Y. Pettersen
Dept. of Engineering Cybernetics
Norwegian University of Science and Technology
7491 Trondheim, Norway
kristin.y.pettersen@itk.ntnu.no

Abstract—This paper introduces JOpenShowVar, a Java open-source cross-platform communication interface to Kuka robots that allows for reading and writing variables and data structures of the controlled manipulators. This interface, which is compatible with all Kuka robots that use KR C4 and previous versions, runs as a client on a remote computer connected with the Kuka controller via TCP/IP. JOpenShowVar opens up to a variety of possible applications making it possible to use different input devices, sensors and to develop alternative control methods.

To show the potential of the proposed interface, two case studies are presented. In the first one, JOpenShowVar is used to control a Kuka KR 6 R900 SIXX (KR AGILUS) robot with an Android mobile device. In the second case study, the same manipulator is controlled with a Leap Motion Controller that supports hand and finger motions as input without requiring contact or touching. Related simulations are carried out to validate efficiency and flexibility of the proposed communication interface.

Index Terms—Robot interface, Manipulator, Control system.

I. INTRODUCTION

As far as robotics is concerned, very few industrial manipulators with an open control interface have been released. Restricting the focus to Kuka robots [1], the standard programming language is the Kuka Robot Language (KRL) [2]. This language is text based and offers the possibility of declaring data types, specifying simple motions, and interacting with tools and sensors via I/O operations. A KRL program can only run on the KUKA Robot Controller (KRC), where it is executed according to real-time constraints. While the KRL offers an easy to use interface for industrial applications, it is very limited when it comes to research purposes [3], [4]. In particular, the KRL is tailored to the underlying controller and, as a consequence, it only offers a fixed, controller-specific set of instructions [5]. The KRL does not support advanced mathematical tools such as matrix operations, optimisation or filtering methods, thus making it very difficult to implement novel control approaches. There is no mechanism for including third party libraries. Due to this design, it is very difficult to extend the KRL with new instructions and functionalities. Moreover, no external input devices can be directly used. The standard workaround for partially expanding the robot’s capabilities consists of using supplementary software packages provided by Kuka. Some examples of such packages are the KUKA.RobotSensorInterface [1], which makes it possible to influence the manipulator motion or program execution via sensor data, or the KUKA.Ethernet KRL XML [1], a module that allows the robot controller to be connected with up to nine external systems (e.g. sensors). However, these supplementary software packages have several drawbacks such as limited I/O, a narrow set of functions and often require major capital investments.

To overcome these problems, this paper presents JOpenShowVar, a Java open-source cross-platform communication interface that makes it possible to read and write all of the controlled manipulator variables, allowing researchers to use different input devices, sensors and to develop alternative control methods. This interface is compatible with all Kuka robots that use KR C4 or previous versions. JOpenShowVar works as a middleware between the user program and the KRL. JOpenShowVar is an open-source project and it is available on the Internet at https://github.com/aauc-mechlab/jopenshowvar, along with several detailed class diagrams, documentation and demo videos.

The paper is organised as follows. A review of the related research work is given in Section II. In Section III, we focus on the description of the JOpenShowVar architecture, analysing the communication protocol and possible control approaches. In Section IV two case studies are presented. In the first case study, JOpenShowVar is used to control a Kuka KR 6 R900 SIXX (KR AGILUS) robot with an Android [6] mobile application. In the second case study, the same manipulator is controlled with a Leap Motion Controller [7] that supports hand and finger motions as input without requiring contact or touching. Concerning the latter case study, related simulations and results are shown in Section V. In Section VI, conclusions and future works are outlined.

*This work is partly supported by the Research Council of Norway through the Centres of Excellence funding scheme, project number 223254 and the Innovation Programme for Maritime Activities and Offshore Operations, project number 217768.
II. RELATED RESEARCH WORK

Several research groups have investigated the possibility of creating a software interface to the Kuka industrial robots. In [3], OpenKC, an open-source real-time control software for the Kuka lightweight robot was presented. This software allows the external triggering and control of all the robot features by using a simple set of routines that can easily be integrated in existing software. This enables developers of robot applications to find solutions for a variety of different software scenarios. However, this software interface is restricted to a specific model of Kaka robots, the Kuka lightweight manipulator, and requires the use of the KUKA.RobotSensorInterface package. In [8], the Kuka Control Toolbox (KCT), a collection of MATLAB functions for motion control of Kaka robots was introduced to offer an intuitive and high-level programming interface for the user. This toolbox is compatible with all six degrees of freedom (DOFs) small and low-payload Kuka robots. In detail, a multi-thread server runs on the KRC and communicates via KUKA.Ethernet KRL XML with a client that manages the information exchange with the manipulator. This communication scheme guarantees high transmission rates, thus enabling real-time control applications. Nonetheless, as in the previous work, this approach is still tailored to the underlying controller and requires the use of the KUKA.Ethernet KRL XML package.

Other researchers have tried a different approach aimed at the disclosure of internal control architecture of the Kuka industrial manipulators. In [5], for instance, the reverse engineering of KRL was investigated and a set of Java-based Robotics APIs were presented for programming industrial robots on top of a general-purpose language. The Robotics APIs implement robot commands like motions and access to I/O calls. However, the Robotics APIs set presents some safety limitations because it is the result of a reverse engineering approach and it does not include a way of specifying complex triggers like it is possible in KRL.

Recently, Kuka has shown more interest in the research and education market. In particular, the KUKA youBot, a mobile manipulator platform with open interfaces that include several open-source software modules has been released [9]. However, even though, the KUKA youBot robot is the only manipulator from Kuka with such an open interface, it has several constraints concerning the limited payload and dimensions that make it more suitable for educational use rather than for industrial applications.

To the best of our knowledge, a cross-platform communication interface that works with all Kuka robots has not been released yet.

III. JOpenShowVar ARCHITECTURE

In this section, the authors refer to several specific functions, variables and configurations related to the KRL and the JOpenShowVar is a client written in Java, thus making

![Diagram](image_url)

Fig. 1: The proposed architecture for JOpenShowVar: a client-server model is adopted.
TABLE I: Reading variables

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>message ID</td>
</tr>
<tr>
<td>09</td>
<td>length of the next segment</td>
</tr>
<tr>
<td>01</td>
<td>type of desired function</td>
</tr>
<tr>
<td>$OV$</td>
<td>variable to be read</td>
</tr>
</tbody>
</table>

TABLE II: Writing variables

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>message ID</td>
</tr>
<tr>
<td>06</td>
<td>length of the next segment</td>
</tr>
<tr>
<td>01</td>
<td>type of desired function</td>
</tr>
<tr>
<td>$OV$</td>
<td>variable to be written</td>
</tr>
<tr>
<td>50</td>
<td>value to be written</td>
</tr>
</tbody>
</table>

cross-platform support possible. This client essentially provides one method, sendRequest, that allows for both reading and writing variables. The return type of the sendRequest method is a Callback instance containing the updated value.

A. Communication protocol

The communication protocol relies on the TCP/IP protocol. In particular, on top of the TCP/IP layer, specially formatted text strings are used for message exchanges. KUKAVARPROXY actively listens on TCP port 7000. Once the connection is established, the server is ready to receive any reading or writing request from the client.

In order to access a variable, the client must specify two parameters in the message: the desired type of function and the variable name. To read a specific variable, the type of function must be identified by the character “0”. For instance, if the variable to be read is the system variable $OV$PRO, which is used to override the speed of the robot, the message that the client has to send to the server will have the format shown in Table I. In detail, the first two characters of this string specify the message identifier (ID) with a progressive integer number between 00 and 99. The answer from the server will contain the same ID so that it is always possible to associate the corresponding answer to each request even if the feedback from the server is delayed. The two successive reading message characters specify the length of the next segment in hexadecimal units. In this specific case, 09 accounts for one character specifying the function type, two characters indicating the length of the next segment and seven characters for the variable length. The fifth character 0 in the message represents the type of the desired function - reading, in this case. Subsequently, there are two more characters indicating the variable length (in hexadecimal units) and finally the variable itself is contained in the last section of the message.

To write a specific variable, three parameters must be specified: the type of function, the name of the desired variable and the value to be assigned. The writing function is specified by the character “1”. For instance, if the variable to be written is the system variable $OV$PRO with a value of 50 (50% override speed), the message that the client has to send to the server will have the format shown in Table II.

B. Control approach

JOpenShowVar opens up to a variety of possible applications making it possible to use different input devices and to develop alternative control methods. In particular, the proposed interface provides the possibility of implementing either a position or a velocity control approach. The user experience is substantially different in each case. When using the position control mode, the operator simply controls the position of the robot’s end-effector with constant velocity; when operating in velocity control mode, the operator also sets the velocity of the robot tool. In the first case, when the operator releases the input device, the end-effector moves back to its starting point, while in the second scenario, the arm just stops moving but it keeps the last given position.

To control the robot motion according to the desired operational scenario, JOpenShowVar allows researchers to use the standard kinematics provided with the KRC. However, it is also possible to implement alternative control algorithms according to current needs as shown in Fig. 2-a and in Fig. 2-b respectively. It should be noted that KRL does not provide a native way to obtain velocity control. When using the KRC kinematics, this limitation can be overcome by expressing the target position as:

\[ x_t = x_d, \]

if operating in position control mode, or by:

\[ x_t = x_d + x_d \Delta t, \]

if operating in velocity control mode, where \( \Delta t \) is the time interval between two successive iterations. Alternatively, when a custom control algorithm is needed, the target joint configuration is given by:

\[ \theta_t = \theta_d, \]

if operating in position control mode, or by:

\[ \theta_t = \theta_d + \theta_d \Delta t, \]

if operating in velocity control mode.

When the operator manoeuvres the manipulator, a vector signal with no semantic, \( s \), is sent from the input device to the user program. Here, according to the operational scenario, the vector signal is interpreted as the target position \( x_t \). If the intent is to use the standard kinematics provided with the KRC, the user program simply works as a driver for the input device and uses the sendRequest method of JOpenShowVar to forward \( x_t \) to a KRL program where the standard KRC kinematics is used to calculate the joint angles \( \theta_d \). Alternatively, a custom control algorithm can be implemented within the user program to calculate the joint values for the robot according to \( x_t \). Essentially, the custom...
The control method has to implement classic inverse kinematic functions that can be generalised as follows:

$$\theta_d = f_p^{-1}(x_d),$$  \hspace{1cm} (5)$$

concerning position control, and

$$\dot{\theta}_d = f_v^{-1}(\theta_a, \dot{x}_d),$$  \hspace{1cm} (6)$$

for velocity control, where $\theta_a$ is the actual joint angles vector. These values are then forwarded to a KRL program where the standard KRC functions are used to actuate the robot.

Note that the possibility of implementing certain control features does not influence the design for the presented interface. Instead, JOpenShowVar extends the functionalities of the KRL language.

IV. CASE STUDIES

A. Case study 1: controlling the Kuka KR 6 R900 SIXX manipulator with an Android mobile device

To show the potential of the presented interface in controlling a Kuka robot from an alternative input device, as a first case study, JOpenShowVar is used to control a Kuka KR 6 R900 SIXX manipulator with an Android mobile device. The Kuka KR 6 R900 SIXX, shown in Fig. 3-a, is a 6 DOFs robotic arm with a slim design and a small footprint.

According to the operational scenario, an Android mobile application whose Graphic User Interface (GUI) is shown in Fig. 3-b, is used to set the target position $x_t$. By using the sendRequest method of JOpenShowVar this vector is forwarded to the KUKA VARPROXY and stored as a global value in a data structure. Finally, a KRL actuator program iteratively retrieves the new global data and uses the KRC kinematics to actuate the robot. The code of the KRL actuator program is shown in Algorithm 1. For Kuka robots, the idle time between motions can be shortened by executing the

```
DEF ACTUATOR()
INI
PTP HOME Vel = 100 % DEFAULT
$ADVANCE=1
LOOP
PTP_REL MYPOS C_PTP
ENDLOOP
PTP HOME Vel = 100 % DEFAULT
END
```

Fig. 2: (a) the user program utilises JOpenShowVar to set the desired end-effector position and then the robot joints are calculated by the KRC using the standard kinematic model, (b) a custom control algorithm can be implemented by the user to calculate the joint values for the robot and then send these angles to the KRC to be actuated.

Fig. 3: (a) The Kuka KR 6 R900 SIXX manipulator, (b) the GUI of the Android mobile application used to control the arm.

Algorithm 1: KRL actuator program for the case study 1 time-consuming arithmetic and logic instructions between motion commands while the robot is moving, i.e. processing them during the advance run (the instructions “run” in advance of the motion). Using the system variable $ADVANCE$, it is possible to define the maximum number of motion blocks the advance run may process ahead of the main run (the motion block currently being executed). Since the main loop of the Server program consists of only one instruction, the system variable $ADVANCE$ is initially set to 1 in order to avoid the unwanted execution of the same line of code. Inside the main loop, a relative movement is iteratively executed to the global variable MYPOS, which is the one that stores the target position. The key word C_PTP is used to approximate
the movement. The approximate positioning instruction is executed in a time-optimised manner in the sense that there is always at least one axis moving with the programmed acceleration or velocity limits. The system simultaneously ensures that the permissible gear and motor torques for each axis are not exceeded. Furthermore, the higher motion profile, set by default, ensures motion that is optimised in terms of velocity and acceleration.

B. Case study 2: controlling the Kuka KR 6 R900 SIXX manipulator with a Leap Motion Controller

As a second case study, JOpenShowVar is used to control the same robot (from the first case study) with a custom control algorithm. This is done to highlight the potential of the presented interface in developing alternative control methods that do not use the standard kinematic model provided by Kuka. Moreover, a Leap Motion Controller [7], shown in Fig. 4, is used as alternative input device to control the robot. The Leap Motion Controller is a small USB input device that supports hand and finger motions as input without requiring contact or touching. This controller is designed to be placed on a physical desktop, facing upwards. Using two monochromatic infra-red (IR) cameras and three IR light-emitting diodes (LEDs), the device observes a roughly hemispherical area, to a distance of about 1 meter. The LEDs generate a 3D pattern of dots of IR light and the cameras generate almost 300 frames per second of reflected data, which is then sent through a USB cable to the host computer, where it is analysed by the Leap Motion Controller software and can be retrieved using the Leap Motion APIs.

The user program runs on a remote computer and uses the Leap Motion APIs to retrieve the target position \( \mathbf{x}_t \) according to the operational scenario. By using the \texttt{sendRequest} method of \texttt{JOpenShowVar}, the actual joint angles \( \theta_a \) are received. This data is used as input for the custom control algorithm. In this specific case study, the classical kinematic functions and the Jacobian method [10] are used to implement (5) and (6). Then, by using the \texttt{sendRequest} method of \texttt{JOpenShowVar} the target joint configuration \( \theta_t \) is forwarded to the \texttt{KUKA VARPROXY} and stored as global value in a structure. Finally, a KRL actuator program iteratively retrieves the new global data and actuates the robot.

The code of the KRL actuator program is shown in Algorithm 2. It should be noted that the variable MYAXIS is initialised to default values inside the INI fold. The system variable \$ADVANCE is initially set to 1. Then the current joint values are assigned to a local structure variable named LOCAL. Inside the main loop, the desired joint angles are iteratively assigned to LOCAL, axis by axis. Finally a PTP movement with C_\( \text{PTP} \) approximation is executed.

V. Simulations and Experimental Results

Related simulations are carried out in order to test the proposed communication interface within the particular case study of the Leap Motion Controller. A real-time trajectory tracking analysis of the Cartesian paths for \( X, Y \) and \( Z \) coordinates is performed, measuring the difference between the desired and actual position of the robot’s end-effector. The results are shown in Fig. 5.

Moreover, to show the responsiveness of \texttt{JOpenShowVar}, a time-delay analysis is carried out for the same Cartesian paths as shown in Fig. 6. Even though there are few spikes with a larger time interval, an average access time of 4.27 ms is obtained in this case. The interface provided by

![Fig. 4: The Leap Motion Controller used to operate the Kuka KR 6 R900 SIXX manipulator.](image)

Algorithm 2: KRL actuator program for the case study 2

```plaintext
DEF EXT_MOVE_AXIS()
DECL AXIS LOCAL
INI
PTP HOME Vel = 100 % DEFAULT
$ADVANCE = 1
LOCAL.A1 = $AXIS_ACT.A1
...
LOCAL.A6 = $AXIS_ACT.A6
LOOP
  LOCAL.A1 = LOCAL.A1 + MYAXIS.A1
  ...
  LOCAL.A6 = LOCAL.A6 + MYAXIS.A6
ENDLOOP
PTP LOCAL C_\text{PTP}
END
```

**VI. CONCLUSION AND FUTURE WORK**

This paper highlights the features of JOpenShowVar as a cross-platform communication interface to Kuka robots. Even though JOpenShowVar only provides a soft real-time access to the manipulator to be controlled, this middleware package opens up to a variety of possible applications making it feasible to use different input devices, sensors and to develop alternative control methods.

In the future, different control algorithms such as the ones implemented in [11], [12] and [13] may be tested as alternatives to the standard KRC. Finally, some effort should be put in the standardisation process of JOpenShowVar to make it even more reliable for both the industrial and the academic practice. In the author’s opinion, the key to maximising the long-term, macroeconomic benefits for the robotics industry and for academic robotics research relies on the closely integrated development of open content, open standards, and open source.

**REFERENCES**