Winter 11-1-2019

An Open-Source Integration Platform for Multiple Peripheral Modules with Kuka Robots

Mahyar Abdeetedal
Amazon

Mehrdad Kermani Ph.D., P.Eng.
Western University, mkermani@eng.uwo.ca

An Open-Source Integration Platform for Multiple Peripheral Modules with Kuka Robots

Mahyar Abdeetedal, Mehrdad R. Kermani

Abstract*—This paper presents an open-source software interface for the integration of a Kuka robot with peripheral tools and sensors, KUI: Kuka User Interface. KUI is developed based on Kuka Fast Research Interface (FRI) which enables soft real-time control of the robot. Simulink Desktop Real-Time^{TM}or any User Datagram Protocol (UDP) client can send real-time commands to Kuka robot via KUI. In KUI, third-party tools can be added and controlled synchronously with Kuka Light-Weight Robot (LWR). KUI can send the control commands via serial communication to the attached devices. KUI can generate Low-level commands using Data Acquisition (DAQ) boards. This feature enables rapid prototyping of new devices for the Kuka robot. Type II Reflexxes Motion Library is used to generate an online trajectory for Kuka LWR and the attached devices in different control modes. KUI is capable of interfacing a broad range of sensors such as strain gauges, compression load cells, pressure sensors/barometers, piezoresistive accelerometers, magnetoresistive sensors (compasses) using either a DAQ board or through the connection interface of amplified bridges. Sensors data, as well as all robot parameters such as joint variables, Jacobian matrix, mass matrix, etc. can be logged during the experiments using a separate stable thread. All these capabilities are readily available through a multithreaded Graphical User Interface (GUI). Three experimental case studies are presented to demonstrate the capabilities of the software in action. KUI is freely available as open source software under GPL license and can be downloaded from https://github.com/mahyaret/KUI

***Index Terms*—Industrial robots, Manipulator arm, Autonomous system, Robotic Grasp.**

I. Introduction

The Kuka Light-Weight Robot (LWR) is the outcome of a research collaboration between Kuka Roboter GmbH and the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR) [1]. This robot has unique features such as high payload ratio, active compliance, and torque sensor feedback which enable researchers to exploit new robot applications. Several research groups have investigated the development of a software interface for Kuka robots. The principal goal varies from taking away the tedious task of programming for communicating with the robot to extending the freedom of the researchers to control the robot. Kuka User Interface (KUI) intends to provide a simple user interface to the Kuka LWR and hides all communication and set-up issues behind the interface as well as essential functionalities for rapid prototyping new devices and sensors synchronously running with Kuka robot.

Mahyar Abdeetedal and Mehrdad R. Kermani are with Faculty of Electrical and Computer Engineering, Western University , London, Ontario, Canada `mabdeete@uwo.ca`, `mkerman2@uwo.ca`

Kuka Fast Research Interface (FRI) provides direct low-level real-time access to the Kuka Robot Controller (KRC) at rates of up to 1 kHz. Kuka FRI is a response to the growing robotic application development demand highlighted in a European Commission funded survey, BRICS [1]. Robotic applications mostly include implementing a haptic input device for augmented reality, attaching a new hand, visual servoing, etc. Several projects considered solving various issues with Kuka Robots communications independently. OpenKC is a control software for Kuka LWR which is restricted to the use of Kuka.RobotSensorInterface package [2]. A reverse engineering of Kuka Robot Language (KRL) is implemented to enable programming of industrial robots on top of the general purpose language which has safety limitations [3]. A collection of MATLAB functions for motion control of Kuka industrial robots is introduced as Kuka Control Toolbox (KCT) [4]. KCT is tailored to the underlying controller and requires the use of the Kuka.Ethernet KRL XML package. Kuka Sunrise.Connectivity is developed for Kuka LWR provides a collection of interfaces for influencing robot motion at various process control levels, but it is not compatible with Kuka LWR IV. JOpenShowVar is a Java open-source cross-platform communication interface to Kuka industrial robots; however, it is limited to soft real-time applications [5]. Robot Operating System (ROS) is a collection of software frameworks for robot software development [6]. Similar to ROS, our software is designed to be as distributed and modular as possible, so that users can use as many or as few components of it as they desire. However, unlike ROS, our software is designed and developed for the Microsoft Windows environment which makes it more accessible for a broader audience.

Our proposed interface, KUI, is based on Kuka FRI which can control Kuka robot and all attached devices synchronously in real-time. The software runs on a remote PC node running Microsoft Windows. Microsoft Windows is not a real-time operating system. Therefore, real-time and low latency responses can hardly be guaranteed except in cases which require slower response time. The remote PC is connected to the KRC (Kuka Robot Controller) via an Ethernet connection. A virtual UDP server is implemented which enables the program to accept real-time commands via the UDP connection. This feature allows Simulink Desktop Real-Time^{TM}, for instance, to send commands through the UDP virtual connection. In addition, KUI allows integrating third-party tools such as sensors and actuators. Sensors can be attached to the robot, and their data be logged. Tools with a different number of actuators can be added to the robot. The geometry of the tool can be defined, and its pose (position and orientation), as well as its actuators, can be controlled synchronously with Kuka LWR. Reaching to a target pose is achieved using real-time trajectory generation. Type II Reflexxes Motion Library [7] is used to generate the online trajectory for the different control modes of Kuka LWR and the attached tools. Additionally, the software provides necessary features for rapid prototyping of generic components such as National Instruments Data Acquisition (DAQ) [8] control panel, ATI Force/Torque Sensor Controller [9] communication system, and Phidgets sensors [10] USB connection system without the need for a massive programming tweaks. In this paper, three case studies are presented to demonstrate the capabilities of the software. These case studies focus on the integration of third-party tools, sensors with Kuka robot.

The structure of this paper is as follows: Section II presents the KUI development. Section III introduces the first case study on KUI ability in conducting fast prototyping. Section IV provides details of the second case study on synchronization and tool integration capabilities of the software. Section V covers the third case study of data logging in the Cartesian impedance control modes. Finally, Section VI concludes the paper and suggests future work.

II. Kuka User Interface

This section provides the details of Kuka User Interface (KUI) development scheme as well as the motivation behind its development. The system is designed to be real-time. The real-time design challenges every aspect of the development especially ensuring its robustness. The system is divided into multiple subsystems running in multiple threads to ensure reliability. The multithreaded structure enables including higher level programming features such as Graphical User Interface (GUI). Moreover, all other devices can be controlled in real-time alongside the Kuka robot. A comprehensive documentation that describes various functions and methods in KUI is available at https://github.com/mahyaret/KUI/wiki. Such functionality of KUI can overlap with other robotic platforms such as the Robotic Operating System (ROS).

A. Soft Real-Time Design

In a real-time application, there are deadlines to be met for each task. Some tasks are activated only when certain events occur. For instance, a system reconfiguration task may be activated only when a fault is detected. Such events may or may not be time-critical, but it must be serviced as soon as possible without risking the deadlines of the other tasks. Deadlines of real-time tasks can be categorized as hard, firm, or soft [11]. A deadline is said to be hard if the outcomes of not meeting it can be catastrophic. Periodic tasks usually have deadlines which belong to this category. A deadline is said to be firm if the results produced by the corresponding task cease to be useful as soon as the deadline expires, but the consequences of not meeting the deadline are not very severe. A deadline which is neither hard nor firm is said to be soft. The utility of results produced by a task with a soft deadline decreases over time after the deadline expires.

Microsoft Windows is not a real-time operating system. Therefore, hard and/or firm deadlines can hardly be met except in cases which require slower response time. KUI bandwidth is limited by the slowest component. For instance, if a force/torque sensor can only run as high as 30Hz, that is going to be the overall system bandwidth. 30Hz bandwidth means that KUI has to be able to handle all input/output signals every 40ms. Because of multithreaded and efficient design of KUI and today's typical computing power of a PC node, the required response can be achieved in real-time. Although, one cannot guarantee hard real-time due to the nature of Microsoft Windows design, KUI can achieve soft real-time performance.

B. ROS vs. KUI

Robot Operating System (ROS) is a robotics middleware. ROS provides services such as hardware abstraction, low-level device control, implementation of commonly used functionality, etc. Despite the importance of low latency in robot control, ROS, itself, is not a real-time OS (RTOS), though it is possible to integrate ROS with real-time code. ROS uses the Operating Systems processes management system, user interface, file system and programming utilities. The most used operating system is Linux, followed by Mac OS X [12].

There are some commonalities in ROS and Kuka User Interface (KUI), however, the main focus of KUI is the ability to control the KUKA Robots in real-time. Although the Robotic Operating System (ROS 2) allows customization for real-time use-cases, it is not fully supported yet. It is worth mentioning that there are separate projects trying to integrate ROS with OROCOS to achieve real-time controllers [13].

Another need for developing KUI is that almost none of the subsystems used in the proposed robotic test-bed is available in ROS. Furthermore, since Microsoft Windows is more common in labs, KUI is developed to be compatible with the Microsoft Windows platform.

That being said, the proposed program is not a ROS competitor and it can be integrated into ROS. ROS can provide packages that implement a bridge to expose access to the motion control and other capabilities of KUI which allows a ROS application to control Kuka robots just as it would a PR2, a Fetch or any other ROS-enabled robot.

C. Communication

KUI provides a simple user interface to the Kuka LWR and hides all communication and set-up complexities. KUI allows access to different controller interfaces of the Kuka system. The KUI runs on a remote PC node connected to the KRC (Kuka Robot Controller) via an Ethernet connection. KUI uses Kuka FRI to have direct low-level real-time access to the Kuka Robot Controller (KRC) at rates of up to 1 kHz. Moreover, all features, such as teaching, motion script functions, Fieldbus I/O, and safety are included. KUI is based on the Kuka Robot Controller version 2 (KR C2), and it does not require installation. KUI allows accessing to different controller interfaces of the Kuka system, including joint position control, Cartesian impedance control, and joint impedance control.

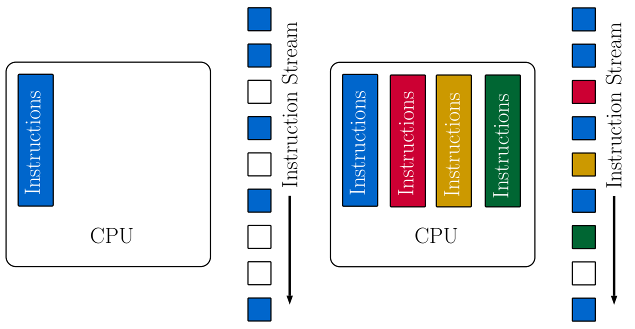


Fig. 1. A thread can consist of subsequent instructions depending on previous results, and running another thread concurrently prevents the computing resources from becoming idle, (a) a conventional processor, (b) multithread processor.

Users can set-up customized control architectures and application-specific controllers for the lightweight arm which is often desired in research projects. UDP packages containing a complete set of robot control and status data (e.g., joint positions, joint torques, drive FRIDriveTemperatures, etc.) are periodically elicited from the KRC unit to the remote host. The remote PC has to instantaneously send a reply message containing input data for the applied controllers (e.g., joint position set-points, joint stiffness set-points, etc.) after the reception of each package.

D. Multithreading

Figure 2 shows the primary architecture of KUI based on different threads. The main program includes Graphical User Interface (GUI), UDP virtual server, logging system, trajectory generator, various device interfaces (NI DAQ, ATI Force/Torque sensor, and PhidgetsBridge), and FRI connection C++ library. Each of these parts run in a separate thread independent from others. For instance, logging system does not stop logging in case of PhidgetsBridge malfunctioning.

Multithreading enables a central processing unit (CPU) or a single core in a multi-core processor to execute multiple processes or threads simultaneously. This capability is supported by most of modern operating systems such as Microsoft Windows and CPUs (see Fig. 1). It should be noted that this approach differs from multiprocessing, as with multithreading the processes and threads share the resources of a single or multiple cores. A significant advantage is that If a thread idles, the other threads can continue taking advantage of the unused computing resources, which leads to faster execution and more stability [14].

POSIX Threads, usually referred to as Pthreads is available on many Unix-like POSIX-conformant operating systems such as FreeBSD, NetBSD, OpenBSD, Linux, Mac OS X and Solaris. However, since Microsoft Windows does not support the Pthreads standard natively, we use Microsoft System.Threading namespace. System.Threading namespace is used for running each essential part of the program stably and separately. This namespace provides classes and interfaces that enable multithreaded programming. It also offers classes

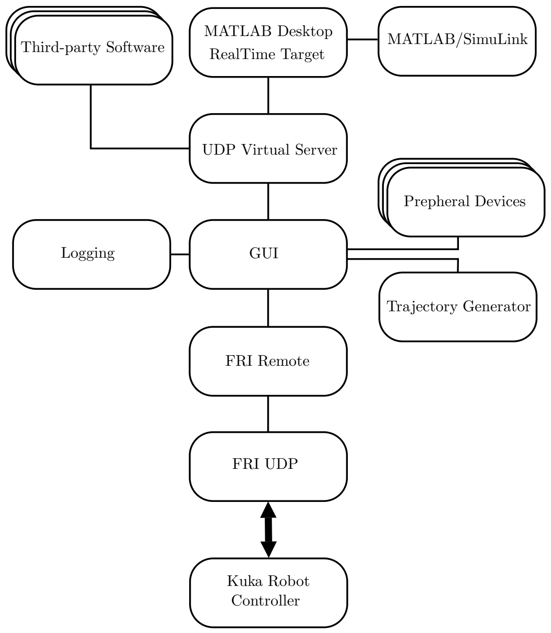


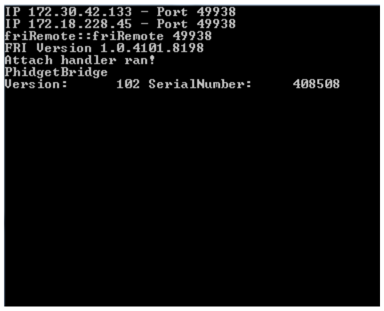
Fig. 2. The architecture of KUI. All features run stably in separate threads.

for synchronizing thread activities and access to data (Mutex, Monitor, Interlocked, AutoResetEvent, and so on). A thread being the execution path of a program, defines a unique flow of control. If an application involves complicated and time-consuming operations, it is often beneficial to set different execution paths or threads, with each thread performing a particular process. Threads are lightweight processes which can be used in the implementation of concurrent programming by modern operating systems. Moreover, the use of threads saves wastage of CPU cycle and increases the efficiency of an application.

E. Graphical User Interface

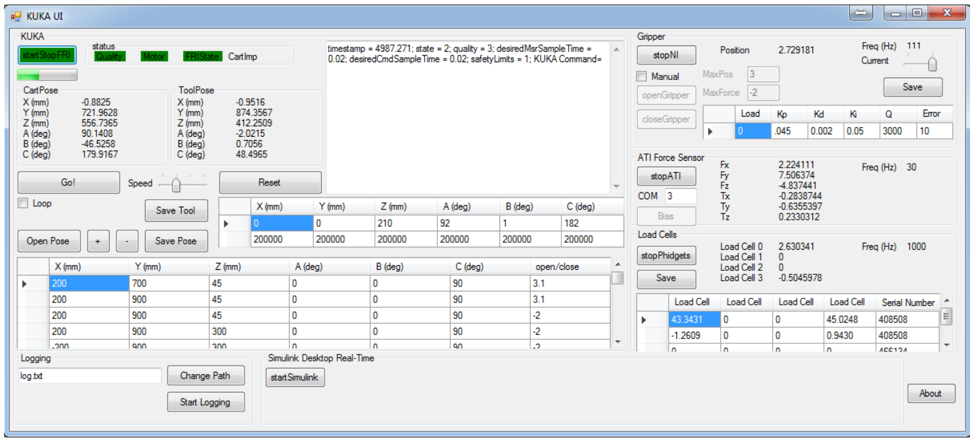
KUI is developed in .NET Framework environment. This environment provides a managed runtime for applications as well as an extensive set of libraries, known as the .NET Framework class library for developers. The .NET Framework handles memory and security which results in more robust applications. Since most of the time it is easier to work with a GUI compared to entering a command into terminal, we consider developing a user-friendly GUI which can receive necessary commands (see Fig. 3). Additionally, for more comfortable debugging, a command terminal is also included in the program.

Through the GUI, the user can start and stop FRI UDP connection to KRC. The status of Kuka FRI packet send/receive is shown by a progress bar. The connection's parameters such as quality, FRI state, control mode are updated in real time on the GUI. In addition, all sent commands and received data are shown in a separate text box. Since the Cartesian pose of the end-effector of a manipulator is essential, both position and orientation of Kuka's robot end-effector and the attached tool is getting updated and displayed in the GUI.



```
IP 172.30.42.133 - Port 49938
IP 172.18.228.45 - Port 49938
friRemote::friRemote 49938
FRI Version 1.0.4101.8198
Attach handler ran!
PhidgetBridge
Version:      102 SerialNumber:      408508
```

(a)



timestamp = 4987.271; state = 2; quality = 3; desiredMsrSampleTime = 0.02; desiredCmdSampleTime = 0.02; safetyLimits = 1; KUKA Command=

CartPose	
X (mm)	-0.8825
Y (mm)	721.9628
Z (mm)	556.7365
A (deg)	90.1408
B (deg)	-46.5258
C (deg)	179.9167

ToolPose	
X (mm)	-0.9516
Y (mm)	874.3567
Z (mm)	412.2509
A (deg)	-2.0215
B (deg)	0.7056
C (deg)	48.4965

X (mm)	Y (mm)	Z (mm)	A (deg)	B (deg)	C (deg)
0	0	210	92	1	182
200000	200000	200000	200000	200000	200000

X (mm)	Y (mm)	Z (mm)	A (deg)	B (deg)	C (deg)	open/close
200	700	45	0	0	90	3.1
200	900	45	0	0	90	3.1
200	900	45	0	0	90	-2
200	900	300	0	0	90	-2
-200	900	300	0	0	90	-2

log.txt

Position 2.729181
Freq (Hz) 111
MaxPos 3
MaxForce -2

Load	Kp	Kd	Ki	Q	Error
0	.045	0.002	0.05	3000	10

COM 3

Fx	2.224111	Freq (Hz) 30
Fy	7.506374	
Fz	-4.837441	
Tx	-0.2838744	
Ty	-0.6355397	
Tz	0.2330312	

Load Cell 0	2.630341	Freq (Hz) 1000
Load Cell 1	0	
Load Cell 2	0	
Load Cell 3	-0.5045978	

Load Cell	Load Cell	Load Cell	Load Cell	Serial Number
43.3431	0	0	45.0248	408508
-1.2609	0	0	0.9430	408508
0	0	0	0	466124

(b)

Fig. 3. Screenshot of the software. (a) command terminal of the software which can be used for debugging purposes. (b) The software GUI can be used for monitoring the robot states, and manually set variables and commands.

The GUI displays position and orientation of Kuka robot and can receive commands for sending to Kuka robot. We include essential command capabilities within the GUI to serve for manipulation tasks such as moving the robot in the Cartesian mode from one point to another. Positioning and orienting a tool is essential for manipulation tasks. Moving a tool requires the geometry of the tool to be known by the robot. A tool geometry can be added and be used for trajectory generation in the GUI. In other words, the trajectory generator of the program can use the Cartesian pose of the tool instead of the robot end-effector to move from one point to another. Several target poses can be defined for the robot to follow in the Cartesian space. The speed of the robot for moving from one point to the other can be set. The user can pause or reset the poses query when needed. All tool definitions and target poses can be saved for the next run.

The GUI displays signals from National Instruments Data Acquisition (NI DAQ) and can receive commands from the user to apply to NI DAQ. NI DAQ can be used for rapid prototyping applications. For instance, we can control a DC motor driven gripper using a PID controller. In this example, position and contact force are read as the feedback for the PID. Since these signals suffer from noise, Kalman filter is implemented to enhance the control performance. The DC motor can be controlled both in manual mode and automatic mode. In manual mode, there are separate buttons for rotating the DC motor in different directions, and the amount of force and position can be set using text boxes. In automatic mode, the controller reads the position/force setpoints and commands from the target pose query. At any given moment that Kuka is in a specific pose, it is possible for the DC motor to be in a particular position or to insert specific contact force. The amount of current applied to the motor can also be set in the GUI.

The GUI displays force and torque measurements of an ATI Force/Torque sensor controller and depending on the method of force/torque measurement the controller can be biased. Biasing process an be done using the ATI Industrial Automation Stand-Alone F/T System. For the sake of user convenience, the biasing capability is added to the GUI. Furthermore, the serial port for the controller connection is also read and can be changed from the GUI.

The GUI displays signals from PhidgetsBridge which is mostly used for connecting load cell or any devices with a low voltage output signal in which amplification is needed. The coefficient of linearization for each of the amplified signals can be set and saved using the GUI. Furthermore, all settings related to the logging mechanism (such as default location for logging file) is available in the GUI.

UDP server can be activated using GUI. This part enables the UDP server and can receive commands from UDP clients. The connection status is also displayed. In this way, popular research software such as MATLAB/Simulink Desktop Real-Time$^{\mathrm{TM}}$can connect to the server and sends command in different control modes.

F. National Instruments Data Acquisition Device

National Instruments M Series multifunction data acquisition (DAQ) modules for USB are popular in research laboratories. In general, USB DAQ devices are ideal for test, control, and design applications including portable data logging, field monitoring. We integrated the programming libraries of NI DAQ in our program to exploit the power of rapid prototyping and controlling synchronously with Kuka robot.

To employ the NI DAQ capability in implementing a controller, a separate thread which is continuously running is defined. This thread is running at the rate of up to 100 Hz (30 Hz is high enough for most control approaches [15]). Figure 4 shows the architecture of this thread. Signals from NI DAQ Analog-to-Digital Converter (ADC) are fed into the Controller, then in Controller block signals are filtered using Kalman filter, and the control signal is sent to NI DAQ Digital-to-Analog Converter (DAC). All received/sent signals are checked before processing in error handling block which tests the sanity of them.

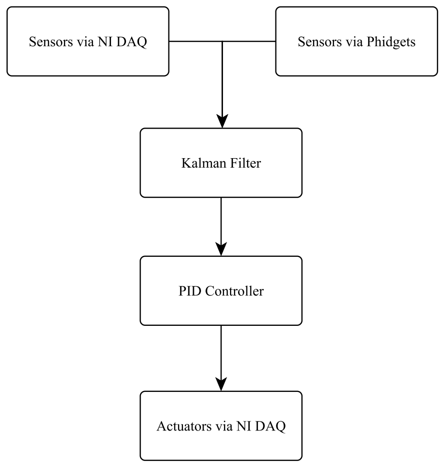


Fig. 4. NI DAQ and Phidgets architecture in which DAQ device is used as both a controller and data acquisition device.

A position control is implemented using NI DAQ to demonstrate the usability of such device in practical applications. Implementation of a PID position control is done using position signal as the feedback signal and current as the control input. CRS Robotics gripper is considered as the test device. The position of the gripper jaw is provided by a potentiometer, which is read by built-in NI DAQ ADC. This signal is filtered by a Kalman filter and fed back to the controller. The controller continuously calculates the error value as the difference between the desired setpoint and the measured process variable and applies a correction based on proportional, integral, and derivative terms using a current signal by NI DAQ DAC. This simple example can be expanded to more sophisticated controllers using the developed structure and ready-to-use tools within the program.

G. Phidgets

Phidgets are low-cost electronic components and sensors that can be controlled easily. Phidgets are mostly suitable for small projects. We consider integrating the PhidgetBridge which allows connections of up to 4 un-amplified Wheatstone bridges, such as strain gauges, compression load cells, pressure sensors/barometers, piezoresistive accelerometers, magnetoresistive sensors (compasses). Moreover, the data rate and gain values can be configured in the software.

Phidgets can be seen as a separate easy-to-use signal acquisition which comes handy in many applications in which signal amplification is needed. It can be run stably at 500 Hz (30 Hz is high enough for most control approaches [15]). Two points linear calibration is required to be done for each input. Signals are calibrated using the resulted parameters and written into the memory.

As an example of inter-thread communication, a PID controller enhanced by a Kalman filter is implemented for the CRS Robotics gripper. In this controller, a setpoint for contact force is considered. This force is measured using load cells which are connected via PhidgetBridge. NI DAQ ADC reads the position of the gripper's jaw. These signals are fed back to the PID controller (see Fig. 4). The current control signal is set using NI DAQ DAC. This example shows the smooth cooperation of different threads of NI DAQ and PhidgetsBridge together.

H. ATI Industrial Force/Torque Sensor

ATI INDUSTRIAL AUTOMATION is the developer of robotic accessories and robot arm tooling, including Multi-Axis Force/Torque Sensing Systems which are widely used by research laboratories. Most of ATI Force/Torques sensors connect to a computer via ATI control box. The ATI control box has an RS232 interface which can be accessed by third-party programs. In telecommunications, RS232 is a standard for serial communication transmission of data which is used for connections to industrial peripheral devices. We enabled KUI to send/receive packets using RS232 protocols for reading from ATI Force/Torque sensors. This capability can be used for connecting to any other RS232 device.

To communicate with ATI F/T Sensor Controller System, three types of data through the RS232 serial port are available: raw strain gauge data in hexadecimal format, raw strain gauge data in decimal integer format, and resolved force/torque data in decimal integer unit format. Data is available in either ASCII or binary format output modes. The length (in bytes) of an output record depends upon the type of data and the output mode. Binary output has the benefit of faster output due to the smaller number of bytes needed to carry information, but cannot be read without further computation. ASCII mode is slow, however, has the benefit of providing data in readable characters. ASCII mode is implemented in the program with a maximum baud rate of 115200 symbols per second which provides a frequency of 30 Hz. This refresh rate is the slowest refresh rate in KUI device support list. KUI limits the whole system control cycle to 30 Hz to achieve hardware synchronization. This frequency rate is sufficient for most Robotic manipulation purposes [15], [16], [17], [18], [19]. For communication with higher speed, NI DAQ can be used to directly collect data from the ATI F/T Sensor Controller System. Figure 5 shows the thread that is responsible for communicating with ATI Force/Torque Sensor Controller System. Data process includes filtering and unit conversion. The processed data is then stored in a shared memory location with logging thread.

It is notable that the serial communication is implemented in a modular way to be readily used for expanding the program to communicate to other RS232-enabled devices. There are functions independently defined for RS232 communication which can be called from any part of the program.

I. Simulink Desktop Real-Time^{TM}

There is no doubt about the power of MATLAB/Simulink in numerical computation and static analysis, making this software to be widely used across the board. Simulink Desktop Real-Time provides a real-time kernel for executing Simulink models on a Windows or Mac laptop or desktop. It includes library blocks that connect to a range of I/O devices. You can create and tune a real-time system for rapid prototyping or

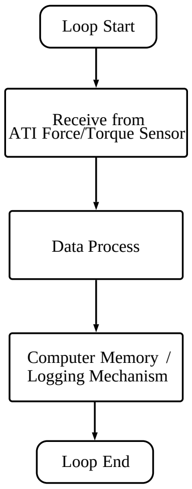

Loop Start
Receive from ATI Force/Torque Sensor
Data Process
Computer Memory / Logging Mechanism
Loop End

Fig. 5. ATI Force/Torque controller software architecture.

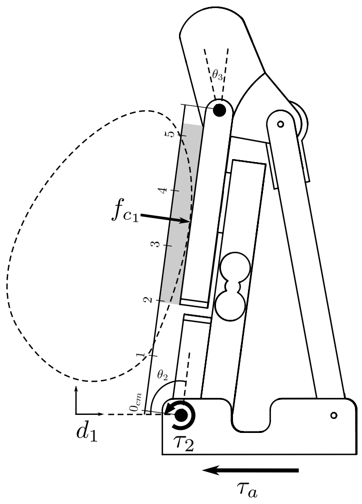


Fig. 6. Generalized actuation, τ_a, moves the finger toward the object. Contact force f_{c_1} causes finger closure which is reacted by spring and results torque τ_2. d_1 is the first joint variable, θ_2 is the second joint variable, and θ_3 is the third joint variable. The gray area (on the second plate) is the length that contact force is measurable (60% of the first phalanx length).

hardware-in-the-loop simulation with your computer. Therefore, the development of MATLAB connectivity is considered in the software. In fact, we consider the more general case of exploiting the FRI connectivity and develop UDP package server for interfacing any third-party software to send commands and read data to/from Kuka robot.

J. On-Line Trajectory Generation

To feed Kuka robot with different positions for following a path, the Reflexxes Motion Libraries is integrated in KUI. These libraries contain algorithms to deterministically compute motion trajectories instantaneously with worst-case computation times in the range of microseconds. Reflexxes Motion Libraries has an extensive application in robotics, CNC machining, and servo drive control systems. The On-Line Trajectory Generation algorithms of the Reflexxes Motion Libraries are capable of generating motion trajectories in different modes of synchronization behavior: non-synchronized, time-synchronized, and phase-synchronized. The trajectory generation of non-synchronized and time-synchronized modes are always possible; however, phase-synchronized trajectories require certain input values. Discussion on the detail of the algorithm is out of the scope of this paper and can be found in [20].

KUI is capable of trajectory generation for generating motions for both Kuka robot and the attached tool. Since time synchronous movement is critical in cooperation of the tool and the Kuka robot, all trajectory generations are constrained to be done in time-synchronized mode. In other words, all degrees of freedom in Kuka robot and the tool start moving and end moving in the target points at the same time. This critical feature enables smooth cooperation of the Kuka robot and the attached tool. This capability is demonstrated in the second case study in Section IV.

III. Case study: Fast Prototyping

In this study, the capability of KUI in controlling a third-party tool using NI DAQ was investigated. We wanted to show that KUI can control a virtually unknown device using low-level signal commands via NI DAQ. CRS Robotics is an open/close gripper which was attached to the manipulator's end-effector. However, due to its support discontinuation, a custom control system was needed. This process was only possible promptly using a fast prototyping device like NI DAQ. A PID controller enhanced with Kalman filter was developed in KUI to control the gripper and two underactuated fingers attached to the gripper. The fingers were designed and prototyped in the lab. The design detail of the finger is out of the scope of this paper. However, such details can be found in a separate paper [17]. Fig. 8 shows the setup for this experiment. The goal was to produce a synchronous motion in which the Kuka robot and the gripper manipulate a grasped object. Grasping an object and moving it required the cooperation of both the gripper and Kuka robot.

Since CRS Robotics gripper was used in our experiments, fingers were actuated using the prismatic joints. Force control was obtained based on the force feedback provided by the load cell connected to PhidgetsBridge. Since the strain gauge based force sensor noise is unavoidable, a PID controller enhanced with Kalman filter was used to regulate the contact forces. The force regulations for f_{c_1} shown in Fig. 6 at $3.5N$, $5.5N$, and $7N$, as well as the respective joint variables, are illustrated in Fig. 7. The velocity at impact was transferred into a high force overshoot. The overshoot in the force control was also due to static friction of the joints and the stress of the torsion spring. We believe that the fusion of the position data and force readings can better deal with impact force and contact detection. This conjecture however, requires further investigation.

To define this task in KUI, the gripper dimensions were defined in the software (see Table 1). Table 1 can be used as a reference for filling out the tool data grid in KUI. The tool was defined using six parameters, i.e., X, Y, and Z, the dimensions of the tools in millimeter, and A, B, and C, the

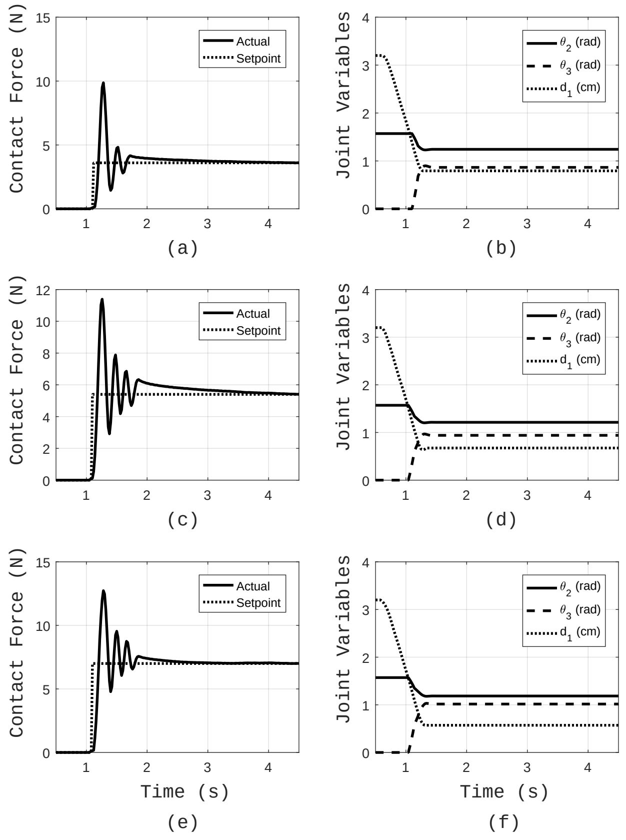


Fig. 7. Force control and joint variable. d_1 is the first joint variable, θ_2 is the second joint variable, and θ_3 is the third joint variable. (a), and (b) Force regulation for $3.5N$ and the joint variables. (c), and (d) Force regulation for $5.5N$ and the joint variables. (e), and (f) Force regulation for $7N$ and the joint variables.

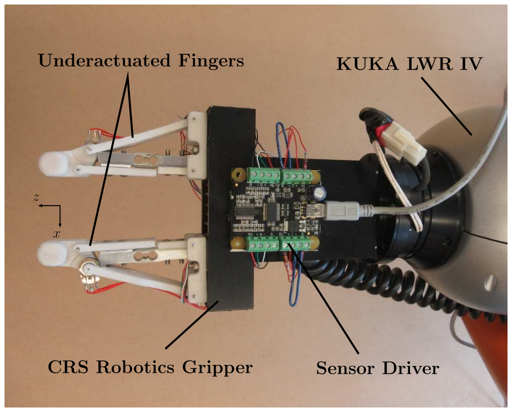


Fig. 8. The experimental setup.

X	Y	Z	A	B	C
0	0	210	91	1	181

Tab. 1. Tool geometrical specifications.

orientation of the tool in degree. A, B, and C are orientations about Z, Y, and X axes, respectively. These parameters were used to change the end-effector's pose to the gripper's pose for the Kuka robot which resulted in a new task space. The new task space was considered to be the end point of the gripper. Note that all matrix operations on transformation were done in a separate thread in KUI. This case study was implemented in the Kuka Cartesian Impedance mode, and the gripper took rotation commands. The tool's pose was computed in the software and fed into the trajectory generator.

Remark: The notation for orientation in Kuka controller is not in line with the ISO standard for orientation. Orientation A should be around the X axis and C around the Z axis. The standard, ISO 9787:2013, Manipulating industrial robots - Coordinate systems and motions, gives a clear definition of the orientations [21]. However, in this paper, for the sake of compatibility with Kuka controller this standard is not followed.

Figure 9 shows the synchronous motion of the manipulator and the attached tool in different grasp types. We aimed for rotating an egg (power grasp) and a coin (precision grasp) in task space. To have smooth orientation around the new end-effector (gripper), both motion commands for the Kuka robot and the gripper had to be synchronous. The sequential point commands are listed in Table 2. Table 2 can be used as a reference for filling out the command grid in KUI. Under "open/close" column we set the gripper parameters. Positive values show that the command is for opening and they represent the position of gripper's jaw. Negative values show that the command is for closing and they represent the contact force in Newton.

KUI started from the first pose. KUI generated actuation commands for both the gripper and the Kuka robot in real-time and followed each of them one after another. Note that the gripper and the Kuka robot had two separate control systems with even different connection protocols. The Kuka robot was controlled via FRI, and the gripper was controlled using fast prototyping method with NI DAQ. KUI was successfully able to apply commands to both of these devices and manipulate the object. Readers are referred to a video of the experiment at https://youtu.be/yruMRA9iLS8

IV. Case study: Synchronization

In this study, the capability of KUI in controlling a third-party tool using RS232 was investigated. We wanted to show that KUI can synchronously control multiple devices even via different protocols and mediums. A gripper was attached

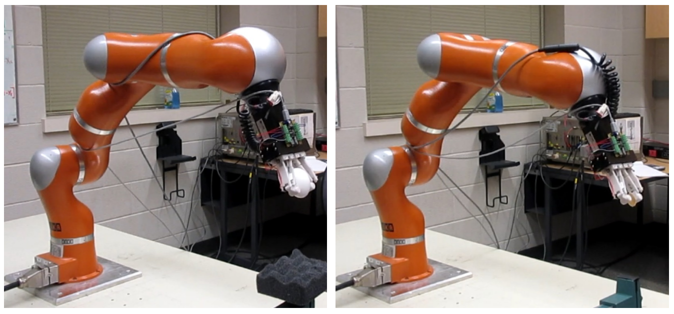


Fig. 9. Different type of grasping of an egg and a coin. (a) power grasping (egg). (b) precision grasping (coin).

X	Y	Z	A	B	C	open/close
0	810	225	0	0	90	3
0	810	225	0	0	90	-5
0	810	340	0	0	90	-5
0	810	340	0	0	90	-5
0	810	340	0	0	30	-5
0	810	340	0	0	90	-5
0	810	225	0	0	90	-5
0	810	340	0	0	90	3

Tab. 2. Trajectory sequence of case study 1

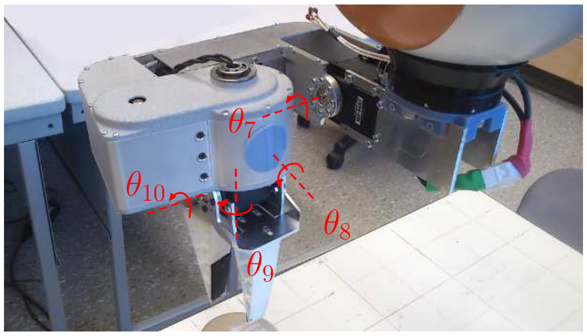


Fig. 10. A gripper with four revolute joints.

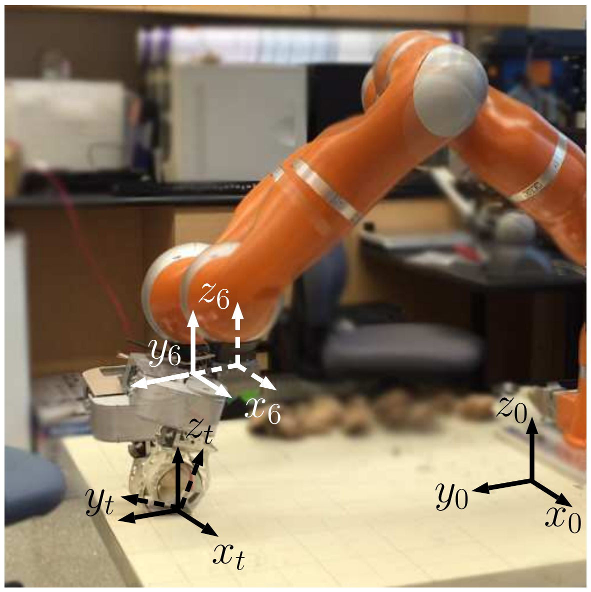


Fig. 11. The cooperation of the gripper and Kuka robot for generating orientation around the end-effector of the gripper.

which was designed and prototyped in the lab. The gripper added 4 Degrees Of Freedom (DOF) to the Kuka robot (see Fig. 10). All joints in the gripper were revolute as it can be seen in Fig. 10. The design detail of the gripper is out of the scope of this paper. This tool was controlled using STM ARM Development Kit, and the commands were sent using serial RS232. The control commands were sent synchronously with Kuka motion commands. Grasping an object and moving it required the cooperation of both the gripper and Kuka robot. The goal was to produce a synchronous motion in which the gripper oriented itself around the center of the grasped object.

To run this task, the gripper dimensions were defined in the software. These dimensions were used to change the end-effector's pose to the gripper's pose for the Kuka robot which resulted in a new task space. The new task space was the end point of the gripper. Note that all matrix operations were performed in a separate thread in the software. This case study was implemented in the Kuka Cartesian Impedance mode, and the gripper took rotation commands. The tool's pose was computed in the software and fed into the trajectory generator.

Figure 11 shows the synchronous motion of the manipulator and the attached tool. We aimed for rotating the gripper around x_t in task space. The resulting orientation around the gripper end-effector was shaped based on rotating θ_7 and displacing Kuka end-effector in y_6 direction. To have smooth orientation around the new end-effector (gripper), both motion commands for Kuka and the gripper had to be synchronous.

Figure 12 shows such synchronizations in actuating end-effector about $5cm$ in y direction while rotating the attached gripper θ_7 about $45°$. It can be seen that the velocity and position of both θ_7 and y_6 reached at the same time to their target points which shows a synchronous behavior of the trajectory generator.

KUI successfully generated and applied a circular motion around an arbitrary center defined by the user. Note that the trajectory generated for all degrees of freedom of the gripper (4 DOF) and the Kuka (7 DOF) simultaneously. While the trajectory was generated for the *new* robot with 11 DOF, the commands were sent separately under different protocols. Kuka was controlled via FRI, and the gripper was controlled using serial RS232. Readers are referred to a video of the experiment at https://youtu.be/vEYGTlQS7Z0

V. Case study: Data Logging

In this study, the manipulation capability and acquiring data from various sensors is demonstrated. An important issue of manipulation in a cluttered environment is that the kinematic uncertainties will result in high internal forces and instability. One approach for addressing this issue is applying impedance control using force/torque and position sensors. Kuka LWR is among those robots that can be run in impedance control mode. By taking advantage of this capability, we are not only able to consider manipulations in an unstructured environment but also capable of applying a controlled amount of stress to objects. This capability is desirable in applications such as harvesting and metal forming in which applying stress and tolerating a foreseeable amount of internal force is required. These types of manipulations rely heavily on data collection. Data collection can be easily done using KUI. Table 3 shows an example of such data collection for four data sets. In this table, the number next to *time* is the time stamp of each data set, the number next to *iPose* is the index of the current command, the numbers next to *estExtTcpFT* is the force/torque estimation provided by Kuka controller, the number next to *NI* is the PID controller command for gripper position, the numbers next to *CalibratedPhidgets* are the amplified and calibrated readings from PhidgetsBridge, the numbers next to *AtiSensor* are force/torque readings using ATI F/T sensor, and the numbers next to *EndEffectorPose* are the pose for

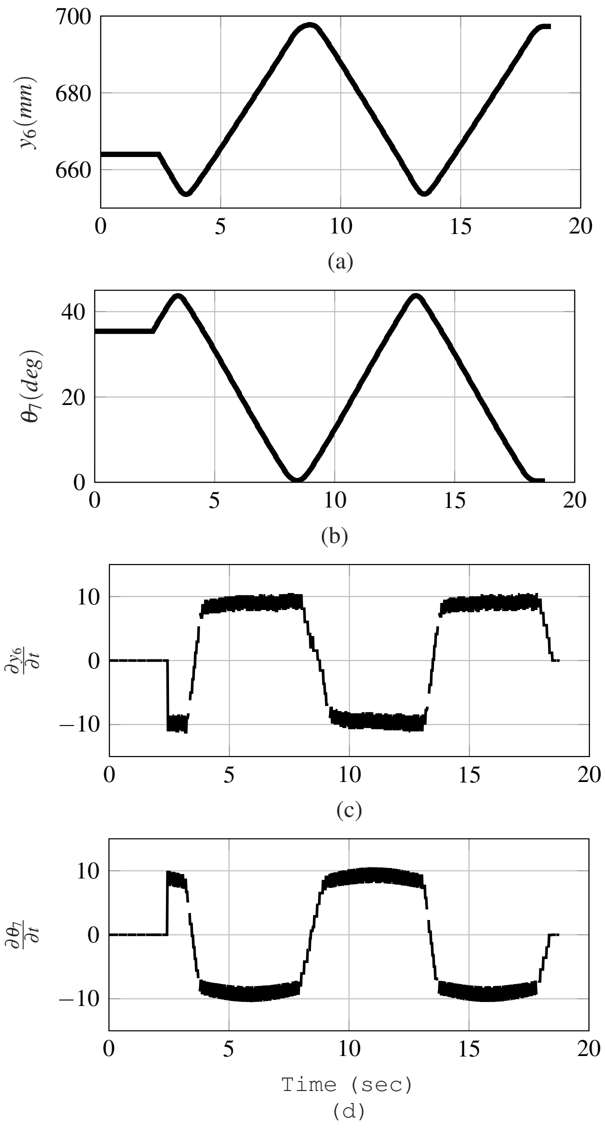


Fig. 12. Position and velocity synchronous outcome of trajectory generation. (a) Position of y_6. (b) Position of θ_7. (c) Velocity of y_6. (d) Velocity of θ_7.

the attached tool. In this study, a force controlled gripper is implemented. Stress analysis was done using sensors' readings during bending a steel rod. The sensor data was logged using KUI logging mechanism. The details of this study were presented in a separate paper [16].

To run this task, KUI read data from multiple sources. An ATI 6-axis Force/Torque sensor at the wrist was used for acquiring force and torque data of the wrist (see Fig. 13). PhidgetsBridge was used for reading contact forces using load cells in fingers. The fingertip that was in contact with the object is shown in Fig. 13. The contact region was a plate screwed to the load cell. The load cell was also screwed to the finger fixture which was actuated by the gripper. With such finger structure, contact force was measured by the load cell. Kuka LWR IV and CRS Robotics underactuated gripper were used for conducting the experiments. Each joint variable of both Kuka and the gripper was logged to be used for stress analysis.

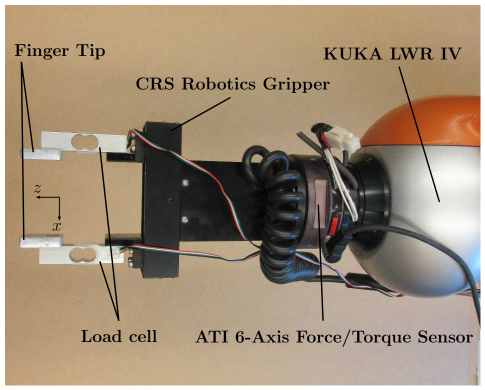


Fig. 13. The experimental setup.

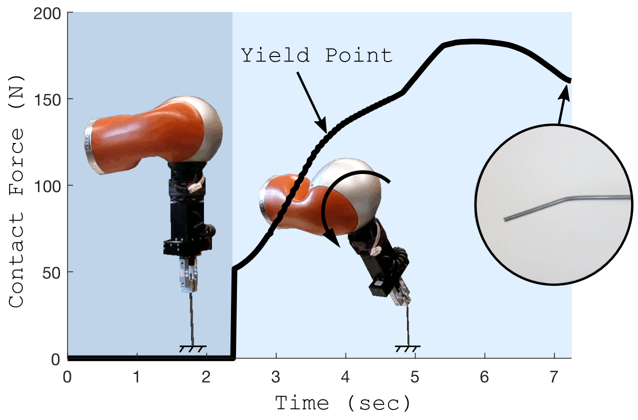


Fig. 14. Yielding plot test results. The steel beam is permanently distorted.

The target object was fixed from one end to the table. In this experiment, the gripper placed the fingers 5mm above the target point. Robot oriented the gripper in a planar motion around the target point (grasped point). In this way, the probable fracture was set to be within the 5mm region. The sequential point commanded to the robot are listed in Table 4. X, Y, and Z are in millimeter, and A, B, and C are the degrees of orientation. A, B, and C are orientations about Z, Y, and X axes, respectively. Under "open/close" column the position of jaw is set for opening (with positive value) or contact force of jaw closing (with the negative value).

Figure 14 depicts the measured reaction contact force while breaking the object. By continuously orienting the gripper around a predetermined yielding location, a more significant reaction contact force was sensed as shown in Fig. 14 for steel beam until the distortion became permanent and the resisting moment dropped. Stress analysis showed that the *normal stress* in this process for steel beam was $2.4770 \times 10^8 N/m^2$ which is larger than its *yielding strength*. Readers are referred to a video of the experiment at https://youtu.be/2uZ6xMaOPbs

VI. Conclusion and Future Works

This paper addressed the programming burden of Kuka robot controlling. A user-friendly graphical user interface, as well as a full range of features for rapid prototyping

time	6987.988	iPose	0	estExtTcpFT	-2.1212	1.1796	-0.4467	-0.1441	0.3363	0.6107	NI	5.8639
CalibratedPhidgets	-0.3206	0	0	-0.1102	AtiSensor	-0.0556	-0.0556	-0.0556	0.0014	-0.0042	-0.0014	
EndEffectorPose	0.0002	0.7953	0.3997	-0.0031	-0.0037	0.7865						
time	6988.008	iPose	0	estExtTcpFT	-2.1191	1.1687	-0.4554	-0.1432	0.3351	0.6074	NI	2.9998
CalibratedPhidgets	-0.3227	0	0	-0.1121	AtiSensor	-0.0556	-0.0556	-0.0556	0.0014	-0.0042	-0.0014	
EndEffectorPose	0.0002	0.7953	0.3997	-0.0031	-0.0037	0.7865						
time	6988.028	iPose	0	estExtTcpFT	-2.1104	1.1424	-0.4572	-0.1439	0.3346	0.6078	NI	3.0027
CalibratedPhidgets	-0.311	0	0	-0.1244	AtiSensor	0	-0.0556	0	0.0014	-0.0028	0	
EndEffectorPose	0.0002	0.7953	0.3997	-0.0031	-0.0037	0.7865						
time	6988.048	iPose	0	estExtTcpFT	-2.1145	1.1939	-0.4576	-0.1445	0.3352	0.6126	NI	2.5121
CalibratedPhidgets	-0.3185	0	0	-0.1121	AtiSensor	0	-0.0556	0	-0.0014	-0.0042	0	
EndEffectorPose	0.0002	0.7953	0.3997	-0.0031	-0.0037	0.7865						

Tab. 3. Logging sample for four data sets which were gathered using KUI.

X	Y	Z	A	B	C	open/close
0	620	300	0	0	0	3
0	680	300	0	0	0	3
0	680	300	0	0	0	-100
0	680	300	0	40	0	-100
0	620	300	0	40	0	3

Tab. 4. Trajectory sequence of case study 2

of new devices and sensors to be used alongside the Kuka robot, was introduced. Three case studies were covered to demonstrate the capabilities of the software. We showed that producing specific patterns of motion by the attached tool to the robot was possible since different devices received time-synchronized commands. An important future work for us, in addition to its constant refactoring, is the integration of other well-known robots such as the Phantom Haptic device for Tele-controlling the Kuka robot. Kuka User Interface (KUI) is freely available as open source software under GPL license and can be downloaded from https://github.com/mahyaret/KUI

References

[1] 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)*. Citeseer, 2010, pp. 15–21.

[2] M. Schoepfer, F. Schmidt, M. Pardowitz, and H. Ritter, "Open source real-time control software for the kuka light weight robot," in *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*. IEEE, 2010, pp. 444–449.

[3] H. Mühe, A. Angerer, A. Hoffmann, and W. Reif, "On reverse-engineering the kuka robot language," *arXiv preprint arXiv:1009.5004*, 2010.

[4] F. Chinello, S. Scheggi, F. Morbidi, and D. Prattichizzo, "Kuka control toolbox," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 69–79, 2011.

[5] F. Sanfilippo, L. I. Hatledal, H. Zhang, M. Fago, and K. Y. Pettersen, "Controlling kuka industrial robots: Flexible communication interface jopenshowvar," *IEEE Robotics & Automation Magazine*, vol. 22, no. 4, pp. 96–109, 2015.

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

[7] "The type ii reflexxes motion library," http://www.reflexxes.ws, accessed: 2017-02-23.

[8] "Ni m series data acquisition," http://www.ni.com/dataacquisition/mseries, accessed: 2017-02-23.

[9] "Ati industrial automation," http://www.ati-ia.com, accessed: 2017-02-23.

[10] "Phidgets," http://www.phidgets.com, accessed: 2017-02-23.

[11] K. G. Shin and P. Ramanathan, "Real-time computing: A new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, 1994.

[12] "Ros installation instructions," http://wiki.ros.org/melodic/Installation, accessed: 2018-08-14.

[13] "OROCOS/ROS Components for Light Weight Robots kernel description," https://rtt-lwr.readthedocs.io/en/latest/, accessed: 2018-08-14.

[14] M. Nemirovsky and D. M. Tullsen, "Multithreading architecture," *Synthesis Lectures on Computer Architecture*, vol. 8, no. 1, pp. 1–109, 2013.

[15] R. I. Zappulla and M. Romano, "A systematic approach to determining the minimum sampling rate for real-time spacecraft control," in *the 27th AAS/AIAA Spaceflight Mechanics Meeting*, 2017.

[16] M. Abdeetedal and M. R. Kermani, "Grasp evaluation method for applying static loads leading to beam failure," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 5999–6004.

[17] ——, "Development and grasp analysis of a sensorized underactuated finger," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 6331–6336.

[18] ——, "Grasp synthesis for purposeful fracturing of object," *Robotics and Autonomous Systems*, vol. 105, pp. 47–58, 2018.

[19] ——, "Grasp and stress analysis of an underactuated finger for proprioceptive tactile sensing," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 4, pp. 1619–1629, 2018.

[20] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.

[21] "Iso 9787:2013 - robots and robotic devices – coordinate systems and motion nomenclatures," https://www.iso.org/standard/59444.html, accessed: 2018-08-14.