Enabling Robotic Adaptive Behaviour Capabilities for New Industry 4.0 Automated Quality Inspection Paradigms

Carmelo Mineo, Momchil Vasilev, Bruce Cowan, Charles N. MacLeod, S. Gareth Pierce

Department of Electronic and Electrical Engineering, University of Strathclyde
Glasgow, Scotland, G1 1XW, United Kingdom
Telephone: +44 (0) 141 548 2350
carmelo.mineo@strath.ac.uk

Cuebong Wong, Erfu Yang

Design Manufacture & Engineering Management, University of Strathclyde
Glasgow, Scotland, G1 1XJ, United Kingdom

Ramon Fuentes and Elizabeth J. Cross

Dynamics Research Group, University of Sheffield
Sheffield, England, S1 4ET, United Kingdom

Abstract

The seamless integration of industrial robotic arms with server computers, sensors and actuators can revolutionize the way automated Non-Destructive Testing (NDT) is performed and conceived. Achieving effective integration and the full potential of robotic systems presents significant challenges, since robots, sensors and end-effector tools are often not necessarily designed to be put together and form a holistic system. This paper presents recent breakthroughs, opening up new scenarios for the inspection of product quality in advanced manufacturing. Many years of research have brought to software platforms the ability to integrate external data acquisition instrumentation with industrial robots for improving the inspection speed, accuracy and repeatability of NDT. Robotic manipulators have typically been operated by predefined tool-paths generated through off-line path-planning software applications. Recent developments pave the way to data-driven autonomous robotic inspections, enabling real-time path planning and adaptive control. This paper presents a toolbox with highly efficient algorithms and software functions, developed to be used through high-level programming languages (e.g. MATLAB, LabVIEW, Python) and/or integrated with low-level languages (e.g. C#, C++) applications. The use of the toolbox can speed-up the development and the robust integration of new robotic NDT systems with real-time adaptive capabilities and is compatible with all 6-DOF KUKA robots, which are equipped with Robot Sensor Interface (RSI) software add-on. The paper describes the architecture of the toolbox and shows two application examples, where performance results are provided. The concepts described in the paper are aligned with the emerging Industry 4.0 paradigms and have wider applicability beyond NDT.

Key words: Automated and robotic NDT, Autonomous inspection, Industry 4.0.

1. Introduction

Quality inspection of critically important parts is always required in manufacturing (e.g. in the aerospace industry). Manual inspection requires highly trained workers and is time-consuming. Therefore, it is often a production bottleneck. Automating the inspection with robots has become an industrial priority to speed up inspection in the production chain [1]. However, robots do not come without its fair share of challenges [2]. All major robot suppliers offer support for the installation of new robots, through the provision of detailed reference manuals. However, a robot arm is only one component of robotic systems targeted to NDT within manufacturing processes. Such systems comprise sensors, end-effectors, additional hardware (e.g. laser cutting, welding, coating equipment, etc.), data acquisition systems and software. The system integration phase is often a challenge, which could slow down the advent and the growth of robotic sensing solutions. Furthermore, there is a growing gap in the skillset of workers in the manufacturing industry for efficiently operating the robotised NDT systems. The current trends of Industry 4.0 comprise the introduction of cyber-physical systems and the implementation of collaborative robots into the manufacturing processes [3, 4]. New integration approaches will play an important role to enable adaptive robotic behaviours and allow robots to work in dynamic and unstructured situations. The current robot controllers often allow the internal implementation and customization of algorithms to interface the robot manipulator with external sensors. However, they do not support advanced mathematical tools (such as matrix operations, optimization, and filtering tasks). It is also hard to integrate them with external hardware and software modules. A possible way to overcome these drawbacks is to build a software abstraction layer upon the proprietary robot programming languages. Moving towards this direction and focusing on KUKA hardware, several toolboxes have been presented in the past few decades for the modelling and control of robot systems [5-9]. However, such software toolboxes are only compatible with robots using controllers of second generation (KRC2) and third generation (KRC3), which are now outdated. Unfortunately, a robust and efficient software interfacing toolbox does not exist for KUKA robots based on the fourth generation of robot controllers (KRC4). Moreover, whilst some of the existing toolboxes could be adapted to support KRC4 robots, such toolboxes can solely be used within the MATLAB environment. That does not offer the optimal level of flexibility to integrators and researchers.

This work presents a cross-platform software toolbox, designed to facilitate the integration of KUKA robotic arms with sensors, actuators and software modules using an external server computer. The platform, named *Interfacing Toolbox for Robots Arms* (ITRA), contains fundamental functionalities for robust connectivity, real-time control and auxiliary functions to set or get key functional variables.

2. Interfacing toolbox

ITRA is a C++ based dynamic link library (DLL) of functions. It runs on a remote computer connected with KRC4 robots through a User Datagram Protocol (UDP/IP) socket. All embedded functions can be used through high-level programming language platforms (e.g. MATLAB, Simulink and LabVIEW) or implemented within a low-level language (e.g. C#, C++), providing the opportunity to speed-up flexible and robust integration of robotic systems.

2.1 Architecture

Figure 1 shows the architecture of the KRC4 controllers and of the ITRA toolbox. The graphic user interface (GUI) allows the user to write and execute robot programs, through defining robot bases, tool parameters and by jogging the robot arm. This GUI runs within an embedded version of Windows XP®. Hidden from the user is a separate operating system called VxWorks®. This is a real-time operating system, which is designed for embedded applications and is developed by Wind River Systems [10]. The VxWorks system controls all robot drives and is used in this platform because of its multi-tasking capabilities, real-time performance and reliability.

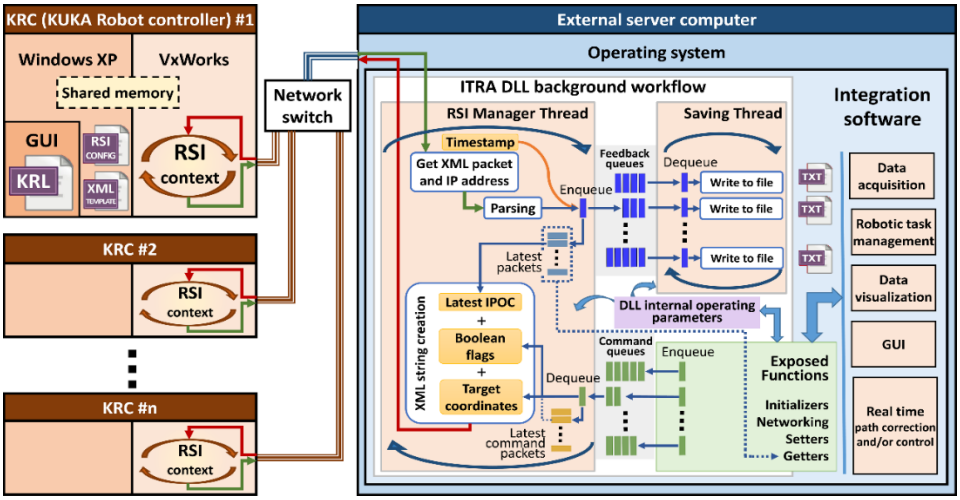

Figure 1. Architecture of the KRC4 controller and of the ITRA toolbox.

ITRA is compatible with all KRC4 robots equipped with a KUKA software add-on known as Robot Sensor Interface (RSI) [11]. RSI runs under the VxWorks operating system in a real-time manner. It was purposely developed by KUKA to enable the communication between the robot controller and an external system (e.g. a sensor system or a server computer). Cyclical data transmission from the robot controller to the external system (and vice-versa) takes place in parallel to the execution of the KUKA Robot Language (KRL) program. Using RSI makes it possible to influence the robot motion or the execution of the KRL program by processing external data. The robot controller communicates with the external system via the Ethernet UDP/IP protocol. No fixed data frame is specified. The user must configure the template of the structure and the content of the data packets in an XML file, stored in the robot controller. Typical data packets sent as ASCII packets by RSI to the external system can include feedback Cartesian or axial coordinates, status of digital I/O signals and real-time operating parameters (e.g. drives currents and torques). Typical data packets received from the external system can include a number of Boolean, integer or double precision variables.

The data packet received from the external system is processed within each machine cycle according to a data processing algorithm defined in the RSI configuration. That is generated through an object-based programming software application known as "*RSI-Visual*", using a library of RSI objects. Each RSI object performs a specific function with its signal inputs and makes the result available at its signal outputs. The linking of the signal inputs and outputs from multiple RSI objects creates a signal flow, which is called "*RSI context*". In the KRL program, the RSI context can be loaded and the signal processing parallel to program execution can be activated and deactivated. The signal processing is performed at the RSI cycle rate (every 12ms or 4ms).

When the RSI context is activated, external data are processed by RSI and forwarded to a portion of the KRC memory that can be accessed by the KRL program. Appended to the end of every packet sent by RSI is a number identified as the Interpolation Cycle Counter (IPOC), which indicates the current timestamp of the data packet. RSI expects the external system to extract this timestamp and append it to the return packet, which must be received by the RSI context within the same cycle. If RSI does not receive the IPOC number back within the cycle duration, the packet is deemed late [11].
ITRA is a C++ language DLL, designed to get feedback parameters from one or more robots simultaneously, to monitor the status of the running KRL robot programs and trigger the progress of the robotic tasks from a server computer. The C++ language was chosen to develop the DLL, since it is particularly suitable to develop highly robust communication and data processing algorithms that run in a reliable real-time manner.
The ITRA DLL and its detailed reference manual can be downloaded through the permanent link given in appendix and additional details have been reported by the authors in [12]. A general description of the ITRA functions is given below. The reader can refer to the schematic representation given in Figure 1. Once ITRA is loaded into a hosting programming environment (e.g. LabView or MATLAB), the DLL constructor initializes fundamental variables to support UDP/IP connection with the robots. These are private variables that cannot be accessed by the hosting application. However, a certain level of control of the DLL internal operating parameters is available through some of the public DLL functions (described below), which allow specifying the number of robots to manage, their IP addresses and the directory that the DLL uses to store data. Only one socket is prepared by the DLL constructor, to communicate with all robots. The connection socket is open through the "*openConn*" function (see below). At this stage, the DLL does not manage any data packets received from the robots. Since each RSI XML packet must get a reply packet from the external system, the DLL needs to run a background thread that receives the RSI packets, parses the data, extracts the packet IPOC numbers and mirrors them to the robots. Such thread is critically important to maintain a robust communication with the robots. It is hereafter referred to as *RSI-Manager Thread* (RMT). RMT cyclically checks if data are available on the UDP socket. As soon as an XML packet is in the socket, the RMT takes a high-resolution clock timestamp and downloads the packet from the socket, decoding the IP address of the KRC that sent it. Then, the XML packet is parsed to extract the Cartesian and axial coordinates, the status of the digital outputs and the packet IPOC number.
It may be necessary to store the parsed positional feedback. Since writing data to files can cause disrupting delays in the RMT, the ITRA DLL uses a secondary auxiliary thread, hereafter referred as *Saving Thread* (ST). The transfer of the parsed data packets takes place through FIFO (first-in first-out) queues. These are container adaptors specifically designed to operate in a FIFO context, where elements are inserted into one end of the container and extracted from the other end [13]. Each data packet is enqueued jointly with the timestamp taken at the time of reception. The ST continuously looks for new packets in the queues and saves them into files, emptying the containers. Since these queues are used to hold robot feedback data, they are referred as "feedback queues" in Figure 1. Besides sending each received data packet and its timestamp to a queue, a copy of the timestamped data is temporarily stored into a structured array containing the latest packets received from each robot controller. The *setRobFeedbackOutput* function (see below) allows enabling/disabling the logging of the positional feedback for each robot,

specifying the data format to be sent to file. The ST creates a separate text file (.txt) for each connected robot, appending the feedback positional packets to the end of the files. The hosting application can use the public functions of the ITRA DLL. These functions support the development of simple and complex integration software platforms, comprising modules like data acquisition, multiple robot task synchronization, interfacing with sensors, data visualization, robot path control and graphical user interfaces. ITRA contains 25 public functions, which can be divided in four groups (see Table 1).

Table 1. List of ITRA functions divided into groups

	Function names	Description	Run time [µs]
Initializers	setNumRob	Set number of robots to manage	24.58
	setRobIP	Set IP address of robot(s)	7.81
	setRobConnType	Set connection type (receive or receive/send)	6.03
	setOutputDir	Set directory for saving feedback file	16.30
	setRobFeedbackOutput	Set format of positional feedback to store	5.74
Networking	openConn	Open connection socket	91.03
	isDataAvailable	Check if data are available in the socket	9.17
	startRSIManager	Start RSI Manager Thread (RMT)	1185.81
	terminateRSIManager	Terminate RMT	8.75
	closeConn	Close connection socket	49.48
Getters	isRSIRunning	Check if RSI is running on a specific robot	26.55
	isRobotTaskActive	Check if the robot task is active	11.99
	isRobStill	Check if the robot is still	12.04
	isRobMoveRequired	Check if a robot move is required	55.85
	isDataAcquRequired	Check if data acquisition is required	9.02
	getCurrPos	Get current robot position	45.79
	getTimestamp	Get current time	8.25
Setters	allowRobotStart	Allow robot to start its task	26722.90
	allowRobMove	Allow robot to move	10333.85
	allowRobotFinish	Allow robot to finish its task	23976.75
	requRealTimeEnd	Request termination of real-time control	98249.93
	requRobTaskEnd	Request termination of current robot task	11981.29
	setCartPos	Set target position in Cartesian space	24.80
	setAxialPos	Set target position in joint space	24.81
	setToolPathFromFile	Set external control tool-path from file	7414.96

2.2 Initializers

The functions referred as "*Initializers*" are designed to set internal fundamental operating parameters of the DLL (e.g. number of robots, IP addresses, type of connection and output directory).

2.3 Networking

The networking functions allow opening the UDP connection, checking if data are available in the socket, starting the RMT to manage the connection with the robots, terminating the background service threads when they are no longer required and closing the connection. The saving thread is automatically launched and terminated together with the RSI-manager thread.

2.4 Getters

The "*Getters*" are functions to retrieve data required by the hosting application. They query the structured array containing the latest packets received from the robot controllers. The function to get the current robot position accesses the requested element of the array and retrieves the parsed Cartesian and axial coordinates, returning them to

the hosting application as an array of double precision values. These can be used to monitor the robot position remotely from the server computer or to encode sensor data in a real-time fashion. Other getters return a Boolean value (TRUE or FALSE); these ITRA functions operate on the status of the four digital outputs inserted by RSI into the XML packets. The function that gets the current clock time (the current timestamp) is the only function that does not query the array with the latest packets. It retrieves the current value of the internal DLL performance counter, which runs in a high-resolution clock. The DLL performance counter uses the same clock used to timestamp the received packets sent to the feedback queue and (optionally) stored into files. Getting access to the same clock used to add a timestamp to each feedback positional packet is critical in many applications, for example when it is necessary to encode sensor data through interpolated robot positions.

2.5 Setters

The "*Setters*" are functions to influence the execution of predefined KRL programs and/or to control the robot tool-path. When called by the hosting applications, these functions generate command data packets addressed to one of the connected robots. The index of the target robot is given to the setters as an input. The generated command packets are sent to reserved FIFO queues, separated from the feedback queues. Such containers are referred as "*command queues*" (see Figure 1); they are also initialized by the ITRA constructor as soon as the DLL is loaded into the hosting application. The command packets are dequeued by the RSI-Manager Thread. After parsing the RSI packet received from the i-th robot controller, the RMT looks for command packets available in the i-th command queue. If the queue is not empty, the packet at the front of the queue is dequeued and its content is concatenated into a string, according to the XML format expected by the RSI context. Through some of the setters, the hosting application can trigger a robot to start its task, continue the task (e.g. after a phase during which the robot must be still) or allow the robot to terminate the task and return to the home position. Such type of control is achieved through acting on four Boolean variables. Moreover, ITRA has functions to set command coordinates in Cartesian-space and in the robot joint-space. External robot control is achieved by transmitting the command coordinates and the preferred robot speed and acceleration through eight double precision variables.

3. Automated and autonomous robotic inspections

The architecture of the introduced interfacing DLL supports the integration of a wide range of applications, especially in the field of "*Robotically Enabled Sensing*". ITRA has been used to integrate robotic NDT inspection systems and enables the possibility to investigate new inspection approaches. This section presents two application examples to demonstrate the use of ITRA. The first example regards a system with three robotic manipulators, used to perform automated photogrammetric and ultrasonic inspection of large high-value manufacturing parts. In this application, the robots follow predefined tool-paths, programmed in KRL through commercial off-line path-planning software. ITRA is used to control the execution of the robot KRL programs, synchronize the data acquisition with the robotic movement, timestamp the data packets and acquire robot positional feedback. However, automated inspections can lead to the gathering of huge data volumes, which can create a bottleneck in data analysis. The second application

introduces a novel inspection approach, based on an autonomous scheme and the use of ITRA, which enables external control of robotic arms.

3.1 Automated inspection through predefined robot tool-paths

ITRA has been used to integrate a robotic inspection prototype system, schematically described in Figure 2 [14]. The robotic hardware of the system comprises three KUKA KR90 R3100 extra HA manipulators. The integrated system is capable of performing volumetric ultrasonic inspection of the part, through an ultrasonic probe manipulated by Robot #1. The ultrasonic instrumentation is linked to the server computer via a PCI Express bus. The camera and the projector are both connected via USB links.

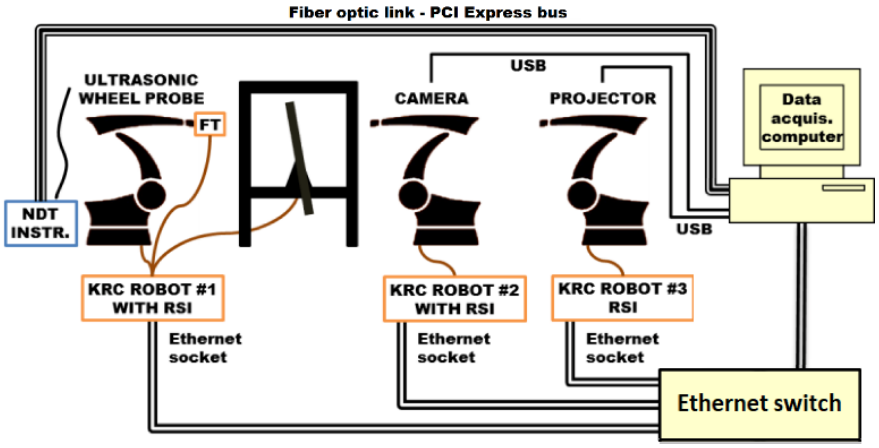

Figure 2. Representation of the robotic inspection system [14].

ITRA has allowed using a single server computer for managing all aspects of the system, controlling the execution of all robotic tasks simultaneously. Each robot KRL program can change the status of four KRC digital outputs during its execution. The values of these digital outputs are inserted by RSI into the packets sent to the external server computer. The system is based on robots following predefined tool-paths (no external path control is used). ITRA is simply used to track the execution of the KRL program, by tracking the status of the KRC digital outputs. On the other hand, ITRA can pause or resume the execution of the KRL programs by acting on the value of four Boolean flags, which are sent by the RMT to the RSI context. This allows synchronizing the ultrasonic and photogrammetric data acquisition with the robotic movements.

3.2 Autonomous inspection based on Bayesian optimisation and external control

Whilst the motivations for robot-based NDT are clear, and the relevant research is under-way [1, 15, 16], little or nothing has been done in the way of performing *autonomously* as opposed to automated inspections. In an automated inspection scheme (e.g. the application described above in Section 3.1), the robot path is programmed prior to the inspection, and the robot makes no decisions regarding what areas of a component it should prioritise. The collected data are reported back to a human, in order to assess the state of the component. However, increasing the use of automated inspections quickly leads to the gathering of large quantities of data, which makes it inefficient, perhaps even unfeasible for a human to parse the information contained in it. In an autonomous scheme, on the other hand, a robot would make its own decisions regarding the path it should take and should also continuously perform its own critical assessment of the component. This section discusses an algorithm developed by the authors [17] that enables such

autonomous inspection paradigms. Exploiting the robot's external control capabilities offered by ITRA has enabled the investigation of such an algorithm in realistic scenarios. In [17], autonomy is achieved by guiding the robot to collect data only in locations of either high uncertainty, or high probability of damage. The result is a robotic system that forms its own picture, *as data is collected*, of whether the component being scanned contains damage/defects, using the minimum possible number of observations to achieve this. The algorithm combines ideas from Structural Health Monitoring (SHM), robust outlier analysis, spatial statistics and optimisation (see Figure 3). First, damage-sensitive features are extracted from the raw NDT data (e.g. time-of-flight and attenuation factors in ultrasound signals). Then, *novelty indices* are computed for each observation. This provides a dissimilarity measure between any given observations against the rest.

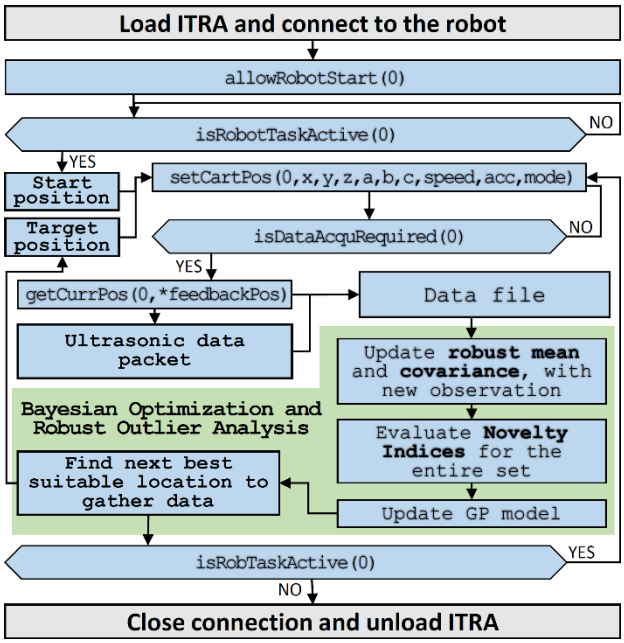

Figure 3. Flow diagram of the autonomous inspection system.

The decision of whether an observation belongs to a damaged or undamaged class can be made by checking whether the novelty index falls above or below a suitable alarm threshold. This still does not solve the problem of large data quantities; one may wish to make that decision without having to collect observations across an entire specimen. This can be time consuming and costly. Ideally, one would be able to *predict* the novelty indices at unobserved locations. The autonomous inspection algorithm of [17] takes a step further in this direction and applies the framework of Gaussian Processes (GPs), an advanced probabilistic nonlinear regression technique. GPs provide a predictive model that can estimate the novelty index at any given location, while quantifying uncertainty for the predictions. The quantification of uncertainty is a key part of the autonomous algorithm. As data arrives, the best potential locations for placing an observation are judged in terms of high-predicted novelty indices and high uncertainty. At every iteration of the algorithm, the best candidate location is the one that offers the most *information gain*. This is the principle behind Bayesian optimisation. The resulting robotic system implementing this algorithm will stop when a) damage has been found with high probability or b) there is a negligible probability of gaining new information from placing further observations. As a means of example, Figure 4 illustrates the application of this

autonomous algorithm to a small 130 x 130 x 25mm steel plate, containing four small sections of damage, introduced as flat-bottom holes drilled from the back wall of the plate.

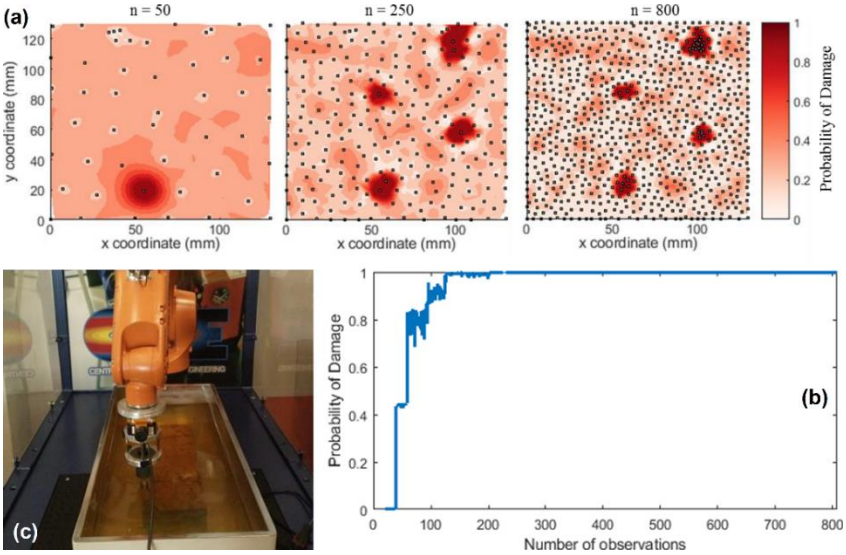

Figure 4. a) Two-dimensional spatial PoD field as observations (*n*) are gathered, b) robotic system used to test the autonomous inspection approach and c) PoD evolution with increasing *n*.

The proposed algorithm has been tested through the small-scale robotic system shown in Figure 4c. The system is based on a KR6 R900 robot and was controlled by a laptop computer, running ITRA and the described sequential algorithm within LabVIEW 2017. A schematic diagram illustrating the logical workflow is given in Figure 3. The application focused on ultrasound-based NDT; however, the presented ideas are applicable to other types of testing. Pulse-echo inspection of a small specimen was performed through a single-element ultrasonic probe. The robot was controlled through the ITRA functions, by the target positions generated with the sequential algorithm. Figure 4a shows an example of the resulting Probability of Damage (PoD). The four defects are clearly shown by the four areas of high PoD. The illustration shows the evolution of predictions as observations are collected (*n* is the number of observations). One of the features of this algorithm is that if there is damage present, it will be found with an optimally minimal number of observations. This is illustrated in Figure 4b, where the (maximum) PoD is plotted against the number of observations. It is possible to observe that in under 150 observations, the system has formed an opinion that there is close to 100% PoD. Any further observations will serve to explore other areas for more potential damage, and to place even more observations around the damaged areas to confirm that measurements nearby are also damaged and to define the region of damage in more detail. As a point of comparison, the robotic system carrying out the autonomous algorithm took approximately 3 minutes to arrive at the decision of PoD=100%, with 3.5MB of supporting raw data. It is important to highlight that this is not just data; it is a decision over the state of the component. In contrast, the same system running a standard raster scan takes over 15 minutes to collect over 1 GB of data, all of which needs further post-processing. Suitable algorithms to reduce the amount of time series data have also been developed by these authors, using compressive sensing technology [17].

4. Additional ITRA external control capabilities

The autonomous inspection system introduced in Section 3.2 gives an example of how the ITRA-based robot's external control can be used to reach target points and perform an incremental autonomous inspection. It is important to note that such application does not represent all real-time external control capabilities offered by ITRA.

Real-time robot motion control can be divided into two subproblems: (i) the specification of the control points of the geometric path (path planning), and (ii) the specification of the time evolution along this geometric path (trajectory planning). The software toolbox allows achieving external control of robotic arms through three different approaches, each offering specific performance. On the other hand, the path-planning subproblem is always dealt with by the computer hosting ITRA, where processing of machine vision data and/or other sensor data can take place to compute the robot target position. The trajectory-planning subproblem can be managed by different actors of the system. In the first approach (hereafter referred to as *KRL-based* approach), the trajectory planning takes place at the KRL module level within the robot controller. This approach has been used for the application in Section 3.2, to command every target coordinate to locations where local NDT data acquisition is required. The second approach has trajectory-planning performed within the external computer, soon after path-planning, and is referred as *Computer-based* approach. The third approach relies on a real-time trajectory-planning algorithm implemented into the RSI configuration. Therefore, trajectory planning is managed by the RSI context and the approach is referred to as *RSI-based*. The latter approach allows true real-time path control of KUKA robots based on KRC4 controllers. This approach permits applying fast online modifications of planned trajectory, to adapt to changes in the dynamic environment and react to unforeseen obstacles. Whereas the path-planning takes place in the server computer, the trajectory planning has been implemented through RSI configuration, employing the second-order trajectory generation algorithm presented in [18].

5. Benchmarking

The run-time of all ITRA functions was investigated by loading the DLL into Matlab 2018a (64bit version), running within a computer with Intel i7-7700HQ CPU and 16GB of RAM. The computer was linked to one KR6 R900 AGILUS robot running a KRL module that contained all required lines to enable the execution of the ITRA functions. Each function was executed 100 times, to record the mean run-time value. The right column of Table 1, in Section 2.1, reports the resulting values in microseconds (µs).

5.1 External control reaction times

The performance of the three external control approaches was also tested. Reaction time is the most important parameter in real-time control, since it measures the promptness of the system. Reaction time in humans is a measure of the quickness the organism responds to some sort of stimulus. The average reaction time for humans is 150ms to a haptic stimulus [24]. Achieving small reaction time is crucial for robots that need to have real-time adaptive behaviours to respond to dynamic changes and/or to interact with humans. The external control latency (or reaction time) is defined herein as the time interval between the instant a new target position becomes available on the external computer and is sent to the robot via *setToolPathFromFile*, *setCartPos* or *setAxialPos* and the instant the robot starts reacting to reach such commanded target. With ITRA running within

Matlab and saving robot feedback positions through the saving thread, the reaction time of each external control approach was measured 100 times through commanding the robot to move to a target from a static position. The timestamp of the first robot feedback positional packet, reporting a deviation greater or equal to 0.01mm from the original home position, was compared with the timestamp taken by *getTimestamp* just before sending the target position to the robot. The resulting reaction times are given in Table 2.

Table 2. Performance of External Control Approaches

External control approach	RSI cycle	Update rate	Reaction time [ms]
KRL-based	4 ms	Variable	113.44
Computer-based	12 ms	Variable	64.84
RSI-based	4 ms	250 Hz	30.03

The average robot reaction time given by the three approaches is always better than the human reaction time. The first approach is 23% better than human reaction. The second and the third approach are respectively 57% and 80% better. The update rate of the first and second approach is variable, since a new target position can be commanded only after the previous target is reached. The update rate of the RSI-based approach is equal to the running frequency of the RSI context, so a new target position can be set every 4ms with the robot expected to react within 30ms (±3ms).

6. Conclusions

The paper has presented a new Interfacing Toolbox for Robots Arms (ITRA). The ITRA contains high-level functions for robust connectivity between multiple KRC4 KUKA robots and a server computer. The toolbox is designed to speed-up efficient integration of robotic systems. Crucially, the ITRA can be used to enable real-time adaptive robot behaviour, maximizing the robot promptness and respecting constraints (maximum accelerations and velocities). The paper has given application examples demonstrating how the toolbox can be used to integrate NDT robotic systems and can play an important role in the area of robotically enabled inspection. A novel paradigm for autonomous robotic inspection was also introduced and demonstrated in practice. The concepts described in the paper are aligned with the emerging Industry 4.0 and have wider applicability beyond NDT. The ITRA allows controlling robot arms with update rates up to 250Hz, achieving robot reaction times as short as 30ms. The benchmarking provided accurate measurement of the run-time of all ITRA functions.

Appendix

A1 - ITRA library and user manual package: https://doi.org/10.15129/bfa28b77-1cc0-4bee-88c9-03e75eda83fd

Acknowledgements

This work was funded by the UK Engineering and Physical Science Research Council (EPSRC), through the grant EP/N018427/1 - Autonomous Inspection in Manufacturing and Re-Manufacturing (AIMaReM project) and by Innovate UK (VIEWS project). The Advanced Forming Research Centre (University of Strathclyde, Glasgow) has provided additional support, through the Route to Impact funding scheme.

References

[1] C. Mineo, S. Pierce, B. Wright, I. Cooper, and P. Nicholson, "PAUT inspection of complex-shaped composite materials through six DOFs robotic manipulators," *Insight-Non-Destructive Testing and Condition Monitoring*, vol. 57, pp. 161-166, 2015.

[2] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, *et al.*, "The grand challenges of Science Robotics," *Science Robotics*, vol. 3, pp. 1-14, Jan. 2018.
[3] R. R. Murphy, T. Nomura, A. Billard, and J. L. Burke, "Human–robot interaction," *IEEE robotics & automation magazine*, vol. 17, pp. 85-89, Jun. 2010.
[4] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, pp. 239-242, 2014.
[5] P. I. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* vol. 73. Berlin: Springer, 2011.
[6] A. Breijs, B. Klaassens, and R. Babuška, "Automated design environment for serial industrial manipulators," *Industrial Robot: An International Journal*, vol. 32, pp. 32-34, 2005.
[7] G. L. Mariottini and D. Prattichizzo, "EGT for multiple view geometry and visual servoing: robotics vision with pinhole and panoramic cameras," *Robotics & Automation Magazine, IEEE*, vol. 12, pp. 26-39, 2005.
[8] M. Toz and S. Kucuk, "Dynamics simulation toolbox for industrial robot manipulators," *Computer Applications in Engineering Education*, vol. 18, pp. 319-330, 2010.
[9] F. Chinello, S. Scheggi, F. Morbidi, and D. Prattichizzo, "Kuka control toolbox," *Robotics & Automation Magazine, IEEE*, vol. 18, pp. 69-79, 2011.
[10] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application," in *Real-Time Conference, 2007 15th IEEE-NPSS*, 2007, pp. 1-5.
[11] KUKA, *KUKA.RobotSensorInterface 3.2 Documentation - Version: KST RSI 3.2 V1*, 2013.
[12] C. Mineo, C. Wong, M. Vasilev, B. Cowan, C. N. MacLeod, S. G. Pierce, *et al.*, "Interfacing Toolbox for Robotic Arms with Real-Time Adaptive Behavior Capabilities," 2019.
[13] B. Stroustrup, *The C++ programming language*: Pearson Education, 2013.
[14] C. Mineo, C. MacLeod, M. Morozov, S. G. Pierce, R. Summan, T. Rodden, *et al.*, "Flexible integration of robotics, ultrasonics and metrology for the inspection of aerospace components," in *AIP Conference Proceedings*, 2017, p. 020026.
[15] C. Mineo, S. G. Pierce, P. I. Nicholson, and I. Cooper, "Robotic path planning for non-destructive testing–A custom MATLAB toolbox approach," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 1-12, 2016.
[16] C. Mineo, C. MacLeod, M. Morozov, S. G. Pierce, T. Lardner, R. Summan, *et al.*, "Fast ultrasonic phased array inspection of complex geometries delivered through robotic manipulators and high speed data acquisition instrumentation," in *Ultrasonics Symposium (IUS), 2016 IEEE International*, 2016, pp. 1-4.
[17] R. Fuentes, C. Mineo, G. S. Pierce, K. Worden, and E. J. Cross, "A probabilistic compressive sensing framework with applications to ultrasound signal processing," *Mechanical Systems and Signal Processing*, 2018.
[18] R. Haschke, E. Weitnauer, and H. Ritter, "On-line planning of time-optimal, jerk-limited trajectories," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, 2008, pp. 3248-3253.