Distributing Numerical Control for Reconfigurable Manufacturing Systems

Vuk Lesi, *Student Member, IEEE,* Zivana Jakovljevic, *Member, IEEE,* and Miroslav Pajic, *Member, IEEE*

***Abstract*—Ever-increasing demands for highly-efficient customized manufacturing are driving the development of Industry 4.0. Reconfigurable Manufacturing Systems (RMS) based on modular, convertible, and interoperable equipment present a key enabler of the 4th industrial revolution. Besides suitable mechanical design, control of these smart manufacturing resources should facilitate reconfigurability, unlike existing Numerical Control Kernels (NCK) which hinder rapid reconfiguration due to the complexity of their monolithic centralized controller.**

On the other hand, reconfigurability is naturally promoted by the distributed control paradigm; therefore, in this paper we investigate design challenges in distributing the conventional centralized NCK designs used for control of Computerized Numerical Control (CNC) systems. We introduce an architecture where each axis module is augmented with a networked Low-Level Controller (LLC) that performs local control and exposes a network interface for communication with other LLCs towards executing the part program. These smart manufacturing resources communicate with a cloud- or edge-based High-Level Controller (HLC) that provides the part program over the network and schedules manufacturing tasks. We investigate real-time and network bandwidth requirements of different mappings of the NCK layers to the LLCs and the HLC, providing design-time tradeoffs for implementing distributed CNC control. We demonstrate feasibility of our approach using industry-grade single-axis robots and low-cost microcontrollers, and show minimal accuracy impairment is introduced compared to the centralized setup based on ISO 230 and ISO 10791-7 standards.

***Index Terms*—Reconfigurable Manufacturing Systems, Industrial Cyber-Physical Systems, Computerized Numerical Control, distributed motion control**

I. Introduction

DEVELOPMENT and implementation of Reconfigurable Manufacturing Systems (RMS) represent one of the key enablers of high product variety manufacturing within Industry 4.0 [1], [2]. To responsively address fluctuating market demands, RMS enable rapid and cost effective changes in production line structure, capacity and functionality [3]. They are characterized by scalability, convertibility, diagnosability, and customization that are achieved through *ad hoc* reconfiguration [4], [5]. Furthermore, quick, easy, and cost effective RMS reconfiguration requires modular manufacturing equipment with interfaces for rapid integration.

Major components of RMS represent reconfigurable machines, including reconfigurable machine tools (RMT), usually characterized by a modular mechanical design [6]. Mechanical elements of modularly designed machine tools can be readily configured using computer-aided design systems, while cabling can be supplied using quick connection elements [7]. On the other hand, control system reconfiguration represents a bottleneck that requires special attention not only with respect to wiring and connection, but also related to the control software design. Currently, during reconfiguration of an RMS, experts adapt and configure automation software and communication protocols, thus increasing system downtime [7], [5].

One of the key enablers of quick reconfiguration is modular control designed for fast (if possible automatic) setup, interconnection and start-up [8]. Rapid control system integration can be achieved through distribution of control functions traditionally performed at a central monolithic controller to a *master node* and *slave nodes* (integral to mechanical modules) that communicate over field bus, or even a wireless network, as illustrated in Fig. 1. In networked CNC (Computerized Numerical Control) systems, transmission latency reduction and clock synchronization of master node (Numerical Control—NC controller) and slave nodes (i.e., servo drives, I/O modules, PLCs) have attracted significant attention. To reduce transmission latency in CNC systems based on real-time Ethernet, in [9] a *cut-through* routing mechanism is proposed in which slave nodes immediately start sending information to the next node, even before they receive the whole frame and before cyclic redundancy check is performed. A clock synchronization scheme based on frequency composition and peer-to-peer transparent clock mechanism is proposed in [10], while [11] proposes a three-ply CNC system reconfiguration based on (i) hardware reconfiguration through loading axis control files on FPGA, (ii) module reconfiguration using Profibus-DP, and (iii) manual software reconfiguration carried out by the operator using the local human-machine interface.

In [12], an embedded CNC system architecture based on PLCopen handling real-time tasks and cloud-based service to improve capacity for non-real-time computations is introduced. Another open architecture design from [13] consists of a host PC that performs generic software functions related to NC code interpretation, validation, error compensation and interpolation, while distributed microcontroller-based slave nodes implement *only* position control and sensor monitoring. While different MCU platforms were evaluated, the details of low-level control were not reported, and the reported accuracy and repeatability of the motion control is $0.2 - 1.0\ mm$. In [14], the Linux Real-Time Application Interface is utilized as the NCK runtime environment on a PC-based system featuring high-bandwidth, real-time Ethernet Powerlink connection

V. Lesi and M. Pajic are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, 27708 USA e-mail: vuk.lesi@duke.edu, miroslav.pajic@duke.edu.

Z. Jakovljevic is with the Faculty of Mechanical Engineering, University of Belgrade, 11000 Belgrade, Serbia e-mail: zjakovljevic@mas.bg.ac.rs

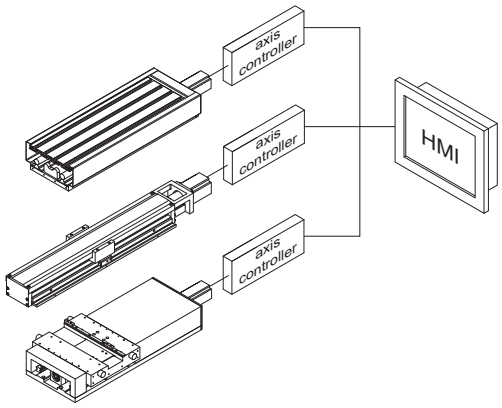

Fig. 1. High-level view of the proposed distributed CNC setup.

to axis servo drives.

Introduction of IEC 61499 is also seen as a major enabler of intelligent automation [15]. To facilitate reconfigurability of NC systems, in particular for slave node parameter self-reconfiguration, machine control-specific modeling language compliant with IEC 61499 is presented in [16], along with a proof of concept NCK. Furthermore, in [17], a multi-agent-based control system is proposed.

All mentioned works consider the architecture where only the last stage of the NCK (i.e., position control) is distributed and all remaining functionality is executed centrally on the master node. This allocation of tasks to controllers local to axis modules does not lead to the desired control system reconfigurability, since significant user intervention is necessary for any RMS configuration change (i.e., low-level parameters need to be propagated throughout the entire NCK [7], [5]). A truly distributed NC requires higher autonomy of slave nodes contributing not only to the reduction of real-time requirements for networking during system execution, but also to lower bandwidth requirements and reduction of the necessary steps during control system reconfiguration. In [18], a simulation-based investigation of a distributed CNC architecture was presented, where all NCK stages would execute on every axis controller; i.e., controllers local to axis modules feature a significantly higher level of autonomy and the master node is essentially not needed. However, in addition to not showing feasibility with a physical implementation, extreme levels of redundant computation are introduced, as all axes individually parse and fully execute the same (complete) part specification (i.e., NC part program).

Consequently, in this work, we focus on design challenges for fully distributed numerical control as a key enabler of RMS. Specifically, we explore design trade-offs, limitations, and architectural constraints arising from different points of distribution of the NCK; namely, different allocations of NCK stages (also referred to as *tasks*) to the *low-level controllers* (LLCs) controlling their respective manufacturing resources (i.e., axes) to which they are integral, as well as a cloud- or edge-based *high-level controller* (HLC).[1] Essentially, by mapping lower stages (i.e., closer to position control) of the NCK to tasks on the embedded LLCs, the level of autonomy of axis modules is increased; the LLCs advertise capabilities of their respective modules (e.g., axis travel length, load handling specifications, maximum acceleration) over the network after joining the system, effectively providing the required modularity and reconfigurability, not only in the mechanical design, but also from the control software perspective. On the other hand, mapping upper stages of the NCK to the HLC decouples compute-intensive workloads (e.g., dealing with CAD/CAM designs and manufacturing process scheduling) from the LLCs allowing us to effectively decrease their cost.

A specific point of control distribution to LLCs/HLC imposes specific bandwidth and real-time constraints on the underlying network, as well as computation requirements for the HLC and LLC platforms; therefore, we analyze the effects of arbitrary NCK task distribution among the HLC and the LLCs. In this work, we focus on two standard NCK architectures (i.e., where acceleration/deceleration control is performed before or after interpolation) [19].

Furthermore, we develop a fully distributed CNC setup using industry-grade single-axis robots and low-cost ARM Cortex-M4F-based MCUs, and evaluate accuracy of distributed position control (relative to the centralized architecture) on a series of positioning and machining accuracy tests (i.e., ISO 230-2 and ISO 10791-7). We show that accuracy is negligibly degraded while reconfigurability is achieved when the NCK is distributed, provided that sufficient level of synchronization between LLCs is maintained.

This paper is organized as follows. In Sec. II, we give an overview of the state-of-the-art architecture highlighting its weaknesses in supporting reconfigurable manufacturing, before introducing our distributed architecture. We then define architecture-specific requirements for supporting the distributed NCK, while varying the point of control distribution in Sec. III. Sec. IV presents a proof-of-concept implementation of the distributed architecture while Sec. V presents results of standard tests utilized to evaluate our implementation. Sec. VI concludes the paper.

II. State-Of-the-Art and the Proposed CNC Architecture

A typical CNC system can be divided into three main components: (i) Numerical Control Kernel (NCK) which is responsible of controlling the tool positioning process, (ii) Programmable Logic Controller (PLC) which controls all other aspects of the process not directly involving controlled motion (e.g., cooling, spindle/end effector control, tool change), and (iii) Human-Machine Interface (HMI) through which the operator can enter the NC program, monitor the process state, or alter the machine configuration. In this work, we focus on the NCK, since its centralized architecture is the main feature hindering efficient reconfiguration of CNC-based systems.

NCKs can typically be divided into two classes based on the order in which trajectory interpolation and acceleration control are performed. Functional diagrams of Acceleration/Deceleration Control After Interpolation (ADCAI)

[1] In this work, we abandon the *master-slave* terminology as it is not suggestive of a truly distributed setup.

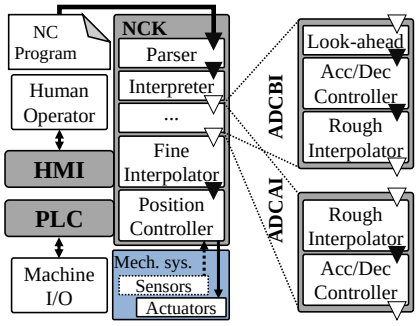

Fig. 2. Two standard numerical control kernel implementations: ADCAI—Acceleration/Deceleration Control After Interpolation (left) and ADCBI—Acceleration/Deceleration Control Before Interpolation (right). Notice the inverse order of rough interpolation and acc/dec control.

and Acceleration/Deceleration Control Before Interpolation (ADCBI) NCKs are shown in Fig. 2. In general, the NC program is parsed by the **parser** to extract information on the commanded linear and circular segments of the desired workpiece geometry (their end points, commanded feedrate, arc radius, etc.). This operation is a one-time process that is not critical to the operation of the NCK and will thus not be considered. Parsed commands are interpreted by the **interpreter** that distinguishes commands by command code, and extracts relevant commanded trajectory information.

In the case of ADCAI, this information is passed on to the **rough interpolator** (IPO) that constructs the tool trajectory by interpolating the tool path. This path is encoded as a vector of incremental reference positions that each axis follows during every sampling period. **Acceleration/deceleration (ACC/DEC) controller** modulates these values to enforce gradual velocity change (e.g., trapezoid- or S-shaped profile) during acceleration/deceleration of every axis in the system. In ADCAI, ACC/DEC control is performed on individual axis' trajectories independently. The result is a set of points interpolated with a predefined IPO period (typ. $\sim 10\ ms$), that conform to prespecified velocity profiles. Since position control loops are typically closed at a higher rate, the necessary sampling rate conversion is performed by linear or moving average **fine interpolator**. Finally, the output of the fine interpolator are incremental reference position samples for every axis to be forwarded to their respective **position controllers**.

The main difference between ADCBI and ADCAI is the order in which rough IPO and ACC/DEC control are executed. In ADCBI, ACC/DEC control is not performed over the interpolated points, but on the commanded linear or circular segments; i.e., limitations of both axes in a given IPO plane are taken into consideration. After ACC/DEC control, **rough** and **fine interpolator** are executed, for which same techniques as in ADCAI are used. As a result, trajectory tracking accuracy is improved in ADCBI for circular trajectories.[2] Additionally, ADCBI NCK features the **look-ahead** stage, where interpreted commands are inspected up to multiple hundreds of instructions ahead to reduce unnecessary slow-downs at end points of consecutive segments. As we show in Sec. III, this has implications for the proposed distributed architecture.

The most noteworthy software aspect of a CNC systems' reconfiguration, regardless of the deployed NCK type, is the required propagation of the mechanical system characteristics (e.g., axis travel length, limit/home switch configuration, maximum allowed acceleration/deceleration) upward to all stages of the NCK. If a simple change of pose of axes is necessary, deep knowledge of the NCK structure is required to ensure correct system behavior upon reconfiguration. This process is not automatic in state-of-the-art systems, and implies non-negligible amounts of down-time. Axis-level reconfigurations also require significant efforts in terms of control hardware reconfiguration. Corresponding I/O, actuation and feedback device interfaces must be available on the centralized controller; furthermore, a large number of conductors must be wired to the centralized controller—in the case of our experimental setup described in Sec. IV, introducing a single axis module into the system requires wiring 4 conductors for the motor, 12 conductors for limit switches and 12 conductors for the position sensing system. Each sensing component features different connections (and communication protocols), effectively hindering inter-operability.

A. Distributed Numerical Control Kernel

We propose a true distributed CNC architecture, in which control functionalities guaranteeing overall system operation and performance are distributed among LLCs that close position loops with locally connected sensors and actuators, and execute parts of the NCK. The control distribution is based on the limitations of the respective mechanical subsystems that LLCs are controlling, with each LLC presenting itself to the network as a manufacturing resource with specific capabilities (e.g., I/O capabilities, maximum travel length, acceleration/deceleration and load limits).[3] LLCs can be extremely inexpensive, given that they can be based on standard low-cost microcontrollers. By utilizing communication capabilities LLCs receive commands and the machining program from a remote process planner (HLC) that can be implemented as a cloud- or edge-based service (depending on the timing requirements discussed in Sec. III-A), where optimized manufacturing process scheduling can be performed. Additionally, to ensure performance of the distributed CNC system, LLCs synchronize their execution over the network (e.g., via the IEEE 1588 Precision Time Protocol).

From the hardware perspective, such an architecture alleviates the physical part of the reconfiguration since an axis' interface reduces solely to a power connection and a standard network connection (if wired networking is employed). However, it also requires thorough understanding of the interfaces between NCK stages, and their real-time and bandwidth requirements. Therefore, in the remaining of the paper we also address the following two challenges.

Challenge 1: Given a specific NCK task distribution among the HLC and LLCs (i.e., point of distribution), what are

[2]Detailed comparison of these two architectures in terms of achievable accuracy can be found in Chapter 4 of [19].

[3]While our proposed architecture easily extends to other actuator/end effector modules, in this work we focus on axes as their control is most involved.

the real-time and bandwidth requirements that the employed network has to satisfy in order to ensure correctness and desired performance of the distributed CNC system?

Challenge 2: Given the functional structure of a standard NCK, does there exist an optimal distribution of NCK tasks among the HLC and LLCs from the standpoint of system design complexity, cost, and performance (i.e., accuracy)?

Consequently, we investigate design choices and pinpoint challenges that arise depending on which part of the NCK is mapped to the controller local to the axis module (LLC), and which part is mapped to the remote planner (HLC).

III. Distributing the Numerical Control Kernel

In this section we begin by giving a detailed specification of data structures bridging inputs/outputs of the NCK stages, and the temporal parameters and bandwidth requirements of their exchange for two standard NCK variants (e.g., as described in [19]). Then, we discuss implications of different NCK task mappings onto HLC and LLCs.

A. Numerical Control Kernel Data Flow

Fig. 3 and 4 show the relevant data structures exchanged by different NCK stages in the cases of ADCAI and ADCBI NCKs respectively. NCK stages are typically coupled with ring buffers or shared memory. We first analyze bandwidth requirements and temporal parameters of the information flow through each of the buffers; these features will directly map into real-time networking requirements depending on the choice of the point of distribution. We will denote NCK stages as **stage**, relevant variables as *variable*, and the corresponding structure types transferred through buffers as `buf_type`. It is important to note that we will consider only single-distribution-point configurations, i.e., configurations where specific upper layers of the NCK are mapped on the HLC, and the corresponding lower layers mapped on the LLC limiting communication overhead to a single HLC–LLC link.[4]

1) Data Flow in the ADCAI NCK: As shown in Fig. 3, the NC program is interpreted by the **interpreter** and every trajectory block is encoded with an `INTbuf_type` structure that contains the *block number* and *G-code type*, *start position* and *end position* in case of linear blocks, as well as *arc center point* and *arc radius* in case of circular blocks. Additionally, desired *feedrate* is specified, as well as *path control mode* indicating whether complete tool stop is desired at the end of the current block (i.e., *exact stop* mode) or if the ACC/DEC controller is allowed to maximize the speed at the joint with the following block (i.e., *continuous* mode) with regards to the maximum allowed accelerations. Corresponding variable types are specified within `INTbuf_type` in Fig. 3. Memory size required to encode `INTbuf_type` (and all structures analyzed in the sequel) depends on the number of axes the machining program utilizes; we denote the number of axes as N. In the case of `INTbuf_type`, $10{+}12N\ Bytes$ of memory is sufficient and one instance of the structure is needed for every linear or circular block of the commanded trajectory.

The **rough interpolator** processes the interpreted commands and interpolates all trajectory points in between the commanded start and end points, based on the chosen interpolation method, on a block-by-block basis. As a result, `roughIPObuf_type` contains *incremental reference positions* per every rough interpolation period which effectively define a rectangular velocity profile. Additionally, *path control mode* is passed on to the ACC/DEC controller. $1{+}4N\ Bytes$ is sufficient to encode `roughIPObuf_type` and one instance of the structure is required per every rough interpolation period (on the order of $10ms$ [19]). The **ACC/DEC controller** utilizes prespecified mechanical limitations, desired acceleration and deceleration times, and the *path control mode* variable to modulate the input velocity profile; i.e., a rectangular velocity profile is transformed into the desired velocity profile. Such adjusted *incremental reference positions* per every rough interpolation period are passed onto the **fine interpolator** through `ACCDECbuf_type` which requires $4N\ Bytes$ of memory to be encoded. Finally, the **fine interpolator** acts as a rate transition block by utilizing linear or moving average interpolation to adjust produced incremental reference positions for each axis in the periodicity of position control (on the order of $1\ ms$ [19]). These samples are transferred to **position control** through `fineIPObuf_type` structure which requires the same amount of memory as the `ACCDECbuf_type` structure.

2) Data Flow in the ADCBI NCK: In ADCBI NCK, ACC/DEC control is performed before interpolation while the (optional) **look-ahead** module receives trajectory information from the **interpreter**. Therefore, the structure `INTbuf_type` remains unchanged. The output of the **look-ahead** stage—`LAbuf_type`, in addition to *G-code type*, *start position* and *end position*, *arc center point* and *arc radius* contains optimized *look-ahead calculated start velocity* and *end velocity*. The size of the `LAbuf_type` is $(9 + 12N)\ Bytes$, and one instance of the structure is needed per linear or circular block of the commanded trajectory.

Followed by **look-ahead**, the **ACC/DEC controller** computes *acceleration time*, *constant velocity time*, and *deceleration time* for the current block. Thus, `ACCDECbuf_type` contains these and *G-code type*, *start position* and *end position* in the case of linear blocks, as well as *arc center point* and *arc radius* in case of circular blocks and the commanded *feedrate* during constant velocity period. `ACCDECbuf_type` structure requires $(19 + 12N)\ Bytes$ for each trajectory block. The **rough** and **fine interpolators** perform the same tasks as in ADCAI NCK, and thus the corresponding `roughIPObuf_type` and `fineIPObuf_type` are the same as in the case of ADCAI NCK, with the exception of *path control mode* in `roughIPObuf_type`.

Tables I and II summarize the temporal properties and bandwidth requirements of the communication between stages of the NCK—in essence they address Challenge 1. The required bandwidth is computed on a per-axis basis by adding the memory footprint of all variables for a given data structure (i.e., one of each of the scalar variables, and one element of each of the vector variables) and multiplying it by the corresponding

[4]Arbitrary task mappings that require multiple communication links between stages of the NCK mapped on the HLC and LLCs feature prohibitively high overhead and do not promote reconfigurability.

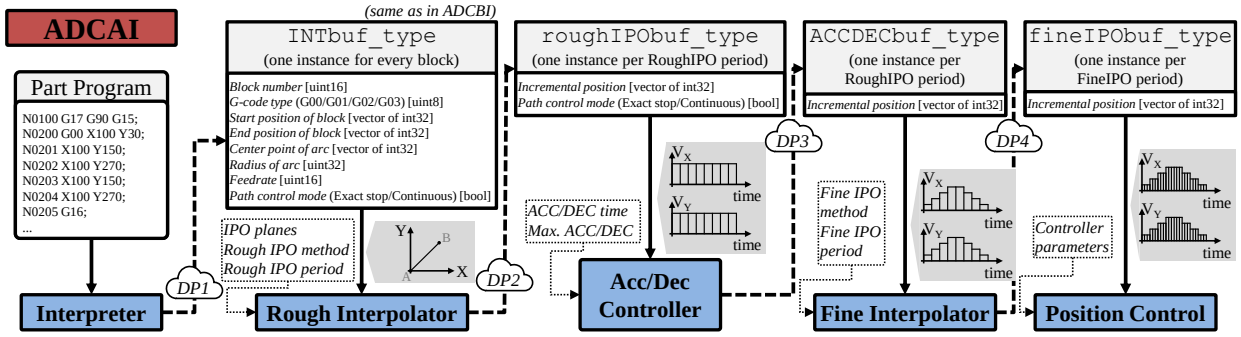

Fig. 3. Data flow in a typical ADCAI NCK. NCK stages are denoted as **stage**, variables as *variable*, and the corresponding structure types as `buf_type`. Possible distribution points (DP) are denoted with clouds (e.g., $DP1$).

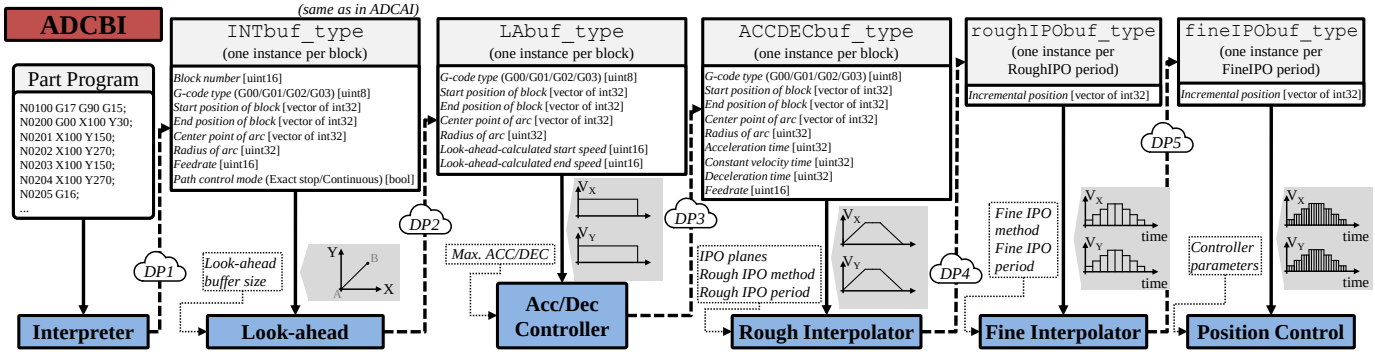

Fig. 4. Data flow in a typical ADCBI NCK. NCK stages are denoted as **stage**, variables as *variable*, and the corresponding structure types as `buf_type`. Possible distribution points (DP) are denoted with clouds (e.g., $DP1$).

rate.[5] Additionally, notice that the output structures of the *interpreter* in ADCAI, and outputs of the *interpreter*, *look-ahead*, and *ACC/DEC controller* in ADCBI are instantiated per trajectory block; thus all instances of this structure could be transmitted once over the network before program execution, or gradually throughout execution, which does not impose real-time requirements.

Remark 1 (Computational capability requirements): In terms of processing power required by the NCK software modules, it is worth noting that while generally lower NCK layers impose more stringent real-time requirements (as can be seen in the presented analysis), upper layers could benefit more from higher computational capabilities. For instance, part code interpretation is a one-time task that, if executed faster, enables faster startup of the manufacturing process. On the other hand, interpolation imposes strict real-time requirements, and the complexity of the employed interpolation method directly determines the required compute power. For instance, linear interpolation requires significantly less resources than spline-based interpolation. However, with the increasing capabilities of low-cost microcontroller and crossover platforms, even mapping these tasks to LLCs is not intractable.

The following subsection analyzes feasibility and practicality of different NCK task mappings.

[5]Notice that this calculation does not include protocol-incurred overhead and is thus the required application (i.e., payload) data rate.

TABLE I
TEMPORAL PROPERTIES AND BANDWIDTH REQUIREMENTS OF COMMUNICATION BETWEEN STAGES OF THE ADCAI NCK.

Data structure	Size [Bytes]	Period [ms]	Bandwidth per axis [kbps]
`INTbuf_type`	$10 + 12N$	—	—
`roughIPObuf_type`	$1 + 4N$	10	4
`ACCDECbuf_type`	$4N$	10	3.2
`fineIPObuf_type`	$4N$	1	32

B. Optimal HLC-LLC Task Mapping

1) ADCAI NCK Task Mapping: If the ADCAI NCK is distributed at distribution point 1 ($DP1$) in Fig. 3, and the part program supplied to LLCs through the network, fully parallel execution of the entire NCK is obtained. Each LLC executes the part program through its local NCK with regard to its controlled axis. Introduction of a new axis in the system, or change of axis pose reduces to the agreement between LLCs on which controller is in charge of which axis in the system, which promotes reconfigurability. Additionally, this configuration introduces no real-time constraints on the network, as the part program execution can be deferred until the entire program is sent over the network, and received by all LLCs. However, this increases hardware complexity of LLCs, required memory size (i.e., to store the entire part program and inputs/outputs of all stages), and consequently their cost. Moreover, the replicated execution of all layers of the NCK results in excess redundant computation on LLCs.

Distribution points $DP2$ and $DP3$ are similar in terms of

TABLE II
TEMPORAL PROPERTIES AND BANDWIDTH REQUIREMENTS OF COMMUNICATION BETWEEN STAGES OF THE ADCBI NCK.

Data structure	Size [Bytes]	Period [ms]	Bandwidth per axis [kbps]
`INTbuf_type`	$10 + 12N$	—	—
`LAbuf_type`	$9 + 12N$	—	—
`ACCDECbuf_type`	$19 + 12N$	—	—
`roughIPObuf_type`	$4N$	10	3.2
`fineIPObuf_type`	$4N$	1	32

bandwidth and real-time requirements (as shown in Table I), but distinct in terms of support for reconfigurability. ACC/DEC control is highly dependent on the properties of the axis' mechanical components. Thus, it is beneficial to map the ACC/DEC control task to the LLC as it locally has access to the mechanical resources and their limitations. In this case (i.e., distribution at $DP2$), each axis advertises itself to the network as a manufacturing resource with specific capabilities (e.g., travel length, max. speed). The mapping of the axes in the machining program to the physical axes can be performed automatically by the HLC (based on axes' configuration in the workspace) or set manually by the operator.

In the case of $DP4$, distribution practically affects position control only. All stages of the NCK execute on the HLC, while position controllers are networked. Due to high bandwidth requirements, and hard real-time networking requirements at this point, special purpose network technologies and communication protocols guaranteeing timely packet delivery are required. As previously discussed this principle is applied in state-of-the-art commercial systems, where position control is embedded in smart motor drives, that are interfaced through a field bus. This architecture *does not* ease the reconfiguration problem, since any changes in the position control layer, or the servo and mechanical subsystems, require manual propagation of machine configuration to upper layers, i.e., distributed LLCs feature an extremely low level of autonomy.

In conclusion, distributing the ADCAI NCK at the output of the rough interpolator (i.e., at $DP2$) is most beneficial in terms of reconfigurability while incurring moderate real-time and bandwidth requirements.

2) ADCBI NCK Task Mapping: If all tasks are replicated on all LLCs (i.e., by introducing the network at $DP1$ in Fig. 4) or if only position control is distributed (i.e., $DP5$ in Fig. 4), exact arguments extend from ADCAI NCK analysis from the previous subsection.

In terms of distribution points $DP2$–$DP4$, slight differences exist compared to ADCAI NCK due to the inverse ordering of interpolation and ACC/DEC control. Optimally (in terms of network requirements and reconfigurability), tasks of part code **interpretation**, **look-ahead** and **ACC/DEC control** are performed jointly for all axes and can thus be mapped to the HLC. On the other hand, **rough** and **fine interpolation** are mapped on to the respective LLCs. *This configuration (i.e., introduction of network at $DP4$ in Fig. 4) promotes reconfigurability as again axes can present themselves to the HLC as smart manufacturing resources with a set of capabilities and limitations (e.g., axis travel length, load handling specifications, maximum acceleration).*

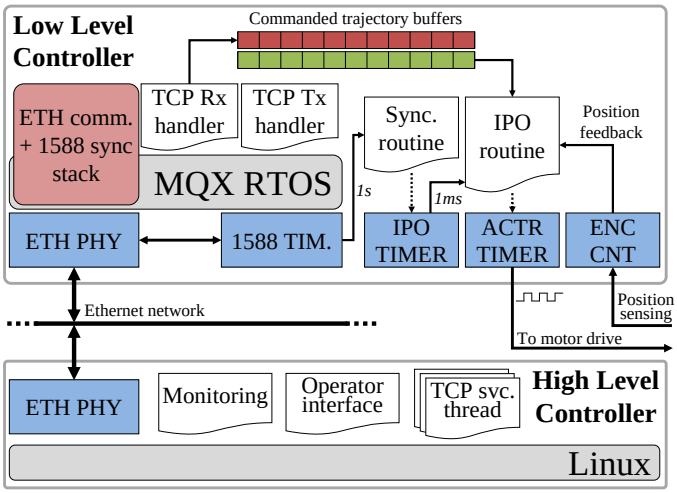

Fig. 5. LLC/HLC software/firmware architecture and hardware dependencies.

It is worth noting that the two preferred configurations (i.e., distribution at $DP2$ in ADCAI and $DP4$ in ADCBI) support custom implementations of interpolation and acceleration/deceleration control on LLCs. This promotes interoperability; e.g., in specific configurations such as 2.5D machining, Z-axis motion interpolation is decoupled from X and Y axes that may implement more complex methods. In general, the optimal balance between network overhead and reconfigurability benefits is obtained when the system is distributed at the point in the NCK when control over axes becomes decoupled, i.e., NCK tasks common for all axes are executed on the HLC, which enables LLCs to independently perform non-redundant computations for their respective axes. This analysis effectively addresses Challenge 2.

In the following section, we describe the distributed ADCAI architecture employed on our proof-of-concept CNC system that we developed using low-cost LLCs and industrial-grade single-axis robots.

IV. PROOF OF CONCEPT SYSTEM ARCHITECTURE

To demonstrate feasibility of our approach and validate of our analysis, we developed a distributed CNC system based on HIWIN KK86 single-axis robots driven by TRINAMIC QSH5718-series stepper motors and Geckodrive G203V stepper motor drives. With the motor step size of $1.8°$, and 10-microstepping capability of the motor drive, the axes' Basic Length Unit (BLU) is $5\ \mu m$. Traditionally, as discussed in previous sections, rough and fine interpolation are separated due to restricted computational capabilities of legacy CNC controllers. However, modern platforms provide sufficient computational power at no added cost, enabling the interpolation (in the case of both ADCAI and ADCBI) and ACC/DEC control (in case of ADCAI) to be performed at the target rate of position control eliminating the need for fine interpolation.

We realized this single-rate version of ADCAI NCK by implementing part code interpretation on the HLC, while executing interpolation, ACC/DEC control and position control on LLCs. The HLC is based on an Nvidia Jetson TK1

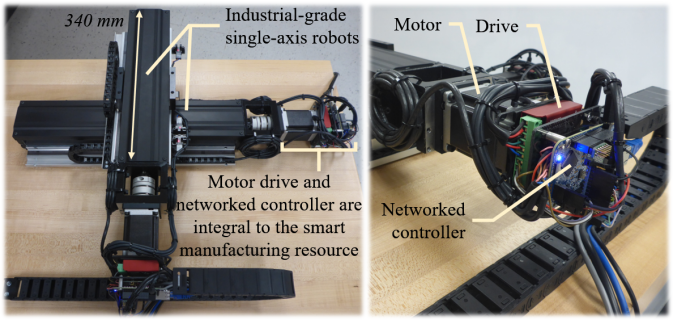

Fig. 6. Our physical reconfigurable CNC axes (left), and the motor-drive-controller module (right).

Linux-based embedded platform featuring an ARM Cortex-A15 CPU and a low-power Nvidia Kepler-based GPU. The HMI is integrated within the HLC; i.e., the GPU unit is used to accelerate visualization of the current tool position for the operator with near-real-time response.[6] The LLCs are embedded in the *axis-motor-stepper drive* module (shown in Fig. 6) and based on low-cost NXP FRDM-K64F Arm Cortex-M4F-based microcontrollers. LLCs are running MQX RTOS where all NCK- and communication-related software components are realized through threads of different priorities. The electrical interface of an axis module is reduced from a total of 28 conductors (for the motor, limit switches and the position sensor), to a simple 2-wire power connection and a standard network connection (discussed in the sequel), which greatly reduces installation/reconfiguration time.

High-level system overview of the HLC and LLC software architectures (including hardware dependencies) is shown in Fig. 5. HLC communicates with LLCs via standard IEEE 802.3 Ethernet protocol. Recall that since the NCK interpreter executes on the HLC (i.e., the system is distributed at $DP2$ in Fig. 3), the underlying communication network need not provide real-time guarantees. The HLC runs a Transmission Control Protocol (TCP) server and accepts TCP communication requests from LLCs.[7] A dedicated *TCP service thread* is created at runtime when a new manufacturing resource (i.e., axis) joins the system, and is used to communicate with the TCP client (*TCP Rx* and *TCP Tx handlers* in Fig. 5) running on the respective LLC. While the part program can be transmitted to the LLCs in its entirety before execution starts, this may impose impractical memory size constraints on LLCs for large part programs. Therefore, in our implementation the HLC transmits interpreted commands in bursts of 10 trajectory blocks while LLCs feature a double receiving buffer (*commanded trajectory buffers* in Fig. 5) to ensure starvation-free part program execution. Thus, in this implementation, part program size does not impose memory storage requirements on the LLCs, while allowing use of standard non-real-time communication protocols.

[6]Notice that if trajectory visualization is not of interest, the HLC need not be equipped with a graphics processing unit.

[7]We implemented the LLC-HLC communication through TCP rather than the User Datagram Protocol (UDP) to ensure reliable delivery during reconfiguration and part program execution as TCP packets are tracked and checked for errors, eliminating lost and/or corrupt transmissions.

The LLCs synchronize their operation through the standard IEEE 1588 Precision Time Protocol (PTP) using the methodology described in detail in. In essence, the protocol on this platform allows generation of synchronous 1 PPS (Pulse Per Second) events on LLCs that invoke the *synchronization routine*. The sync. routine ensures local time offsets are corrected, both in terms of timely processing of the commanded trajectory (i.e., strictly periodical execution of interpolation, ACC/DEC control, and position control), and in terms of synchronous generation of the stepping pulse train (i.e., pulse edge and frequency synchronization between LLCs). The *IPO timer* invokes the chain of interpolation, ACC/DEC control and position control with 1 ms period. As a result, a decision is made on whether the axis should be accelerated, decelerated, or current feedrate maintained, and this decision is propagated to the motor drive by adjusting the stepping pulse generator (i.e., actuation timer—*ACTR timer*). We employed closed-loop position control; position sensing is performed trough compact high-resolution magnetically-coded BALLUFF BML-S1F1 positioning system with 1 μm digital resolution and overall system accuracy of ± 10 μm. Incremental position pulses are accumulated by the encoder counter (*ENC CNT*) within the LLC.

The following section gives results of standardized accuracy tests that prove only minor performance degradation (i.e., accuracy impairment) is incurred when distributing the NCK, while reconfigurability is ensured.

V. Testing and Evaluation

We validate the 2D configuration of our distributed CNC implementation by running standardized tests defined by the International Standardization Organization (ISO). While these standards provide definitions of representative test conditions in general, our goal is to compare performance of centralized and distributed control architectures conditioned that the position control algorithms and the underlying mechanical system are identical. Influence of different systematic and random errors inherent from the underlying mechanical system can be isolated if both centralized and distributed control modes are tested using the same components; thus we implement a *synchronous* mode of operation in our system described in Sec. IV. In synchronous mode, IEEE 1588-based synchronization is bypassed and the Y-axis controller is synchronized to the X-axis controller by means of a simple wired connection; the 1 ms-period IPO timer on X-axis triggers execution of the local IPO routine and a change in the state of an output port, which triggers execution of the IPO routine on Y-axis. This ensures perfect synchronization between axis controllers and allows us to validate performance of the distributed setup. The following subsections give results of tests conducted under ISO 230-6 [20] and ISO 10791-7 [21] standards.

A. ISO 230-6 Diagonal Displacement Tests

ISO 230 standard [20] consists of eleven parts; in this work we consider positioning accuracy tests specified by Part 6. According to this part of the standard, axes of the machine tool should be commanded a trajectory spanning all diagonals

TABLE III
EXCERPT OF ACCURACY METRICS DEFINED IN ISO 230-6 MEASURED ON OUR 2D SYSTEM IN DISTRIBUTED AND SYNCHRONOUS MODES.

Parameter	Sync. mode	Distr. mode	Impairment
Repeatability of positioning (left) $[\mu m]$	10.8443	13.0231	2.1788
Repeatability of positioning (right) $[\mu m]$	16.4316	15.2970	—
Accuracy of positioning (left) $[\mu m]$	11.3909	13.0231	1.6322
Accuracy of positioning (right) $[\mu m]$	16.4316	15.8333	—
Systematic positional deviation (left) $[\mu m]$	1.5557	1.9799	0.4242
Systematic positional deviation (right) $[\mu m]$	3.8183	4.8082	0.9899

of the workspace, such that positioning accuracy is evaluated at multiple target positions along the diagonals. In the case of our 2D setup, there exist two diagonals. According to the standard, the dwelling duration of the axes in the target positions should be long enough to permit stabilization of axes' positions and measurement acquisition. The set of target positions along the axes is determined by ISO 230 Part 2 as a minimum of five points that are to be approached by the respective axes from both directions five times. Notice that when tests are performed according to Part 6 of the standard, these target positions are chosen along diagonals (rather than individual axes). Exact position of each point is obtained by adding a random term to equidistant target positions; this ensures suppression of effects of periodical deviations on estimates of positioning accuracy. We conduct measurement acquisition by means of sampling the position sensors (encoders) used in the local position feedback loops. This procedure is alternative to using equipment such as a laser interferometer, and is permitted by Annex D of ISO 230 Part 4 [20]. These tests are referred to as *tests using feedback signal*.

Table III shows an excerpt of accuracy metrics for our system according to ISO 230, both for the synchronous and the distributed mode of operation, for one of the workspace diagonals. It is worth starting with the manufacturer's specification, advertised repeatability of positioning is $\pm 10\ \mu m$ for our specific single-axis robots [22]. Our measured repeatability (first two rows of Table III), approximately attest this (notice that the specifications are single-axis parameters, while diagonal repeatability may be worse). Furthermore, notice that, since acquisition of obtained positions is performed when the position has stabilized (while both axes are halted in the target position), performance of our setup in both synchronous and distributed modes of operation is practically identical. This is expected since the closed-loop position control will eventually stabilize the axis within a $\pm 1\ BLU = \pm 5\ \mu m$ range of the target position (as this is the granularity of displacement the feedback controller is operating with). Results show insignificant impairment when the NCK is distributed.

B. ISO 10791-7 Machining Accuracy Tests

ISO 10791 Part 7 [21] defines test pieces to be machined as a part of evaluation of geometric accuracy of a machining center. Fig. 7 shows the test piece drawing we used along with tolerances on geometric features. We perform tests under no-load conditions using the feedback signal. Since standard tolerances could not be obtained from an independent measuring instrument (as tests using feedback signal are conducted), we computed the *straightness* error by finding maximal deviations (in both perpendicular directions) from a straight line fitted to the first and the last sample corresponding to the respective feature. *Angularity* was determined by computing the distance between two lines sloped according to the feature angle that fully enclose the obtained trajectory. *Position* error was determined by computing the distance between the commanded and the obtained bore positions.

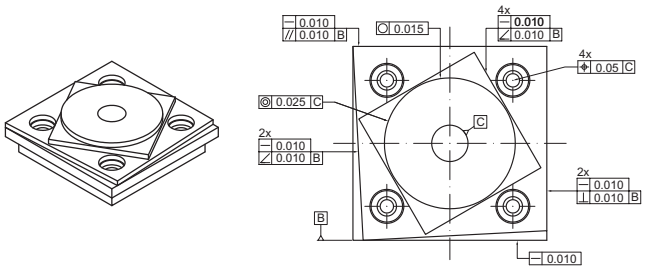

Fig. 7. ISO 10791-7 test piece; machining trajectory of the piece in the X-Y plane is used as a reference input during testing. Tolerances are captured as $(type, tolerance, datum)$, where $type$ specifies how the tolerance is measured, $tolerance$ is the allowed deviation, and the optional $datum$ specifies the reference features relative to which the respective tolerance is defined.

TABLE IV
EXCERPT OF TOLERANCES DEFINED IN ISO 10791-7 MEASURED ON OUR 2D SYSTEM IN DISTRIBUTED AND SYNCHRONOUS MODES.

Feature and tolerance		Synchronous mode	Distributed mode	Impairment
outer edges	B datum straightness $[\mu m]$	1.0326	1.0392	0.0066
	left vertical edge straightness $[\mu m]$	0.0000	1.3285	1.3285
diamond edges	bottom-left edge straightness $[\mu m]$	16.2776	24.0416	7.7640
	bottom-left edge angularity $[\mu m]$	15.5563	17.6777	2.1214
corner bores	bottom-left bore position $[\mu m]$	1.0000	2.2361	1.2361
	bottom-right bore position $[\mu m]$	4.0000	5.0000	1.0000

Table IV compares an excerpt of straightness, angularity and position tolerances defined in ISO 10791-7 for the synchronous and distributed modes of operation. Again, these tests indicate practically insignificant impairments when the NCK is distributed, given that the basic length unit of our axes is $1\ BLU = 5\ \mu m$, and the overall accuracy of position sensing is $\pm 10\ \mu m$ confirming feasibility of the plug & play CNC architecture.

C. Reconfigurability Benefits of the Proposed Architecture

Besides significantly simplifying the hardware aspect of reconfigurations (as described in Sec. II-A), the proposed architecture is proven feasible from the functional performance perspective, as shown above. On the other hand, from the

higher-level reconfigurability aspect, the increased autonomy of CNC system's building blocks controlled by LLCs decouples the centralized planning software running on the HLC from low-level subsystem-specific control. This allows the HLC to utilize modules (over the network) based on *what* they are capable of (e.g., travel length and load handling characteristics), without worrying about *how* control over a specific module should be performed (e.g., AC vs DC motor control).

With the developed architecture, in a typical reconfiguration scenario, such as adding a new degree of freedom, after being physically introduced into the system and powered on, the axis module (i.e., its LLC) communicates its capabilities to the HLC over the network. Depending on the type of module, the reconfiguration can be automatic, i.e., functional role of the module can be assigned automatically according to a CAD model for the specific manufacturing task. Alternatively, the operator can be prompted to only provide the desired configuration, i.e., assign functional role (e.g., X, Y, Z axis) to each of the modules. Beyond reducing the downtime based on control functionality distribution, this architecture does not fundamentally limit interoperability, as even modules coming from different series or manufacturers (potentially employing different proprietary control techniques) can be used if they can be interfaced over the network in the described manner and composed mechanically to obtain the desired workspace.

VI. Conclusion

In this work we have proposed a distributed motion control architecture suitable for reconfigurable machine tools. In this architecture, each axis module is augmented with an LLC controlling the axis module by utilizing locally connected sensors and actuators, and exposes a network interface through which it communicates with other LLCs and a HLC where process planning is performed. This architecture promotes reconfigurability as a configuration change such as introduction of a new module in the system, is performed by simply presenting the new module to the network of smart manufacturing resources. We have investigated design tradeoffs arising when different stages of the numerical control kernel are mapped onto the LLCs and the HLC, while considering two standard NCK implementations. Additionally, we have instantiated the distributed motion control architecture on industry-grade single-axis robots augmented with low-cost Arm Cortex-M4F-based controllers and have shown minimal accuracy impairment is introduced when the NCK is distributed.

References

[1] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster, *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry*. Forschungsunion, 2013.

[2] H. ElMaraghy, G. Schuh, W. ElMaraghy, F. Piller, P. Schönsleben, M. Tseng, and A. Bernard, "Product variety management," *CIRP Annals - Manufacturing Technology*, vol. 62, no. 2, pp. 629 – 652, 2013.

[3] Y. Koren, X. Gu, and W. Guo, "Reconfigurable manufacturing systems: Principles, design, and future trends," *Frontiers of Mechanical Engineering*, vol. 13, no. 2, pp. 121–136, 2018.

[4] Y. Koren and M. Shpitalni, "Design of reconfigurable manufacturing systems," *Journal of Manufacturing Systems*, vol. 29, no. 4, pp. 130 – 141, 2010.

[5] J. Otto, B. Vogel-Heuser, and O. Niggemann, "Automatic parameter estimation for reusable software components of modular and reconfigurable cyber-physical production systems in the domain of discrete manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 275–282, Jan 2018.

[6] M. Gadalla and D. Xue, "Recent advances in research on reconfigurable machine tools: a literature review," *International Journal of Production Research*, vol. 55, no. 5, pp. 1440–1454, 2017.

[7] G. Pritschow, K.-H. Wurst, C. Kircher, and M. Seyfarth, "Control of reconfigurable machine tools," in *Changeable and Reconfigurable Manufacturing Systems*, H. A. ElMaraghy, Ed. Springer-Verlag London, 2009, pp. 71–100.

[8] M. Abel and P. Klemm, "Flexible soa based platform for research on start-up procedures for reconfigurable production machines," *Lecture Notes in Mechanical Engineering*, vol. 7, pp. 489–501, 2013.

[9] Y.-Q. Wang and F.-C. Huang, "A complete real-time ethernet solution for numerical control systems," *International Journal of Advanced Manufacturing Technology*, vol. 74, no. 1-4, pp. 89–100, 2014.

[10] X. Xu, Z. Xiong, J. Wu, and X. Zhu, "High-precision time synchronization in real-time ethernet-based cnc systems," *Int. J. of Advanced Manufacturing Technology*, vol. 65, no. 5-8, pp. 1157–1170, 2013.

[11] T. Wang, L. Wang, and Q. Liu, "A three-ply reconfigurable cnc system based on fpga and field-bus," *International Journal of Advanced Manufacturing Technology*, vol. 57, no. 5-8, pp. 671–682, 2011.

[12] H. Wang, X. Tang, B. Song, and X. Wang, "A novel architecture of the embedded computer numerical control system based on plcopen standard," *Proc. of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 228, no. 4, pp. 595–605, 2014.

[13] A. Amra, J. Padayachee, and G. Bright, "An open architecture control system for reconfigurable numerically controlled machinery," in *Machine Vision and Mechatronics in Practice*, J. Billingsley and P. Brett, Eds. Springer-Verlag Berlin Heidelberg, 2015, pp. 309–322.

[14] K. Erwinski, M. Paprocki, L. M. Grzesiak, K. Karwowski, and A. Wawrzak, "Application of ethernet powerlink for communication in a linux rtai open cnc system," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 2, pp. 628–636, Feb 2013.

[15] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 768–781, Nov 2011.

[16] N. Zhou, D. Li, S. Li, S. Wang, and C. Liu, "Model-based development of knowledge-driven self-reconfigurable machine control systems," *IEEE Access*, vol. 5, pp. 19 909–19 919, 2017.

[17] A. Bruzzone and D. D'Addona, "New perspectives in manufacturing: An assessment for an advanced reconfigurable machining system," *Procedia CIRP*, vol. 67, pp. 552–557, 2018.

[18] V. Lesi, Z. Jakovljevic, and M. Pajic, "Towards Plug-n-Play Numerical Control for Reconfigurable Manufacturing Systems," in *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016, pp. 1–8.

[19] S.-H. Suh, S. K. Kang, D.-H. Chung, and I. Stroud, *Theory and Design of CNC Systems*. London, UK: Springer-Verlag London, 2008.

[20] International Standardization Organization, *ISO 230 Test code for machine tools*, Std., 2012.

[21] ——, *ISO 10791 Test conditions for machining centres – Part 7: Accuracy of finished test pieces*, Std., 1998.

[22] HIWIN Motion Control and System Technology, *Industrial Robot, Technical Information*, Std. K02TE10-1301, 2013.