Toward a Plug-and-Work Reconfigurable Cobot

Edoardo Romiti, Jörn Malzahn, Navvab Kashiri, Francesco Iacobelli, Marco Ruzzon, Arturo Laurenzi, Enrico Mingo Hoffman, Luca Muratore, Alessio Margan, Lorenzo Baccelliere, Stefano Cordasco, and Nikos Tsagarakis

Abstract—The ongoing trend from mass-produced to mass-customized products with batch sizes as small as a single unit has highlighted the need for highly adaptable robotic systems with lower downtime for maintenance. To address these demands, this article proposes the development of a novel reconfigurable collaborative robot (cobot), which has the potential to open up many new scenarios within the rapidly emerging flexible manufacturing environments. As the technological contribution, we present a complete hard- and software architecture for a quickly reconfigurable EtherCAT-based robot. This novel approach allows to automatically reconstruct the topology of different robot structures, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automaticallyreshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.

I. INTRODUCTION

COBOTS keep entering new application domains in which robotic automation was not conceivable until recently. Yet, many small- and medium-sized enterprises struggle to benefit from robotics. The main perceived barriers are the limited flexibility to serve manufacturing trends of shorter product life cycles, smaller lot sizes, and mass-customized products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customizable from a software perspective, the hardware customizability of existing robotic systems, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automatically reshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.

I. INTRODUCTION

COBOTS keep entering new application domains in which robotic automation was not conceivable until recently. Yet, many small- and medium-sized enterprises struggle to benefit from robotics. The main perceived barriers are the limited flexibility to serve manufacturing trends of shorter product life cycles, smaller lot sizes, and mass-customized products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customizable from a software perspective, the hardware customizability of existing robotic systems, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automatically reshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.

I. INTRODUCTION

COBOTS keep entering new application domains in which robotic automation was not conceivable until recently. Yet, many small- and medium-sized enterprises struggle to benefit from robotics. The main perceived barriers are the limited flexibility to serve manufacturing trends of shorter product life cycles, smaller lot sizes, and mass-customized products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customizable from a software perspective, the hardware customizability of existing robotic systems, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automatically reshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.

I. INTRODUCTION

COBOTS keep entering new application domains in which robotic automation was not conceivable until recently. Yet, many small- and medium-sized enterprises struggle to benefit from robotics. The main perceived barriers are the limited flexibility to serve manufacturing trends of shorter product life cycles, smaller lot sizes, and mass-customized products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customizable from a software perspective, the hardware customizability of existing robotic systems, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automatically reshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.

I. INTRODUCTION

COBOTS keep entering new application domains in which robotic automation was not conceived before. Yet, many small- and medium-sized enterprises struggle to benefit from robotics. The main perceived barriers are the limited flexibility to serve manufacturing trends of shorter product life cycles, smaller lot sizes, and mass-customized products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customizable from a software perspective, the hardware customizability of existing robotic systems, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automatically reshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.

I. INTRODUCTION

COBOTS keep entering new application domains in which robotic automation was not conceived before. Yet, many small- and medium-sized enterprises struggle to benefit from robotics. The main perceived barriers are the limited flexibility to serve manufacturing trends of shorter product life cycles, smaller lot sizes, and mass-customized products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customizable from a software perspective, the hardware customizability of existing robotic systems, composed of a set of body modules, each of which represents an EtherCAT slave. As the theoretical contribution, we propose a method to obtain in an automatic way the kinematic and dynamic model of the robot and store it in universal robot description format (URDF) as soon as the physical robot is assembled or reconfigured. The method also automatically reshapes a generic optimization-based controller to be instantly used after reconfiguration. While this article focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of the following: how the topology of the robot is reconstructed and the URDF model is generated, and a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurability of the system from a 4-degrees-of-freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.

Index Terms—Automation, flexible manufacturing systems, manipulator dynamics, robot kinematics, robot programming.
and merged to quickly increase throughput for simple tasks with few required DOF and reduce energy consumption. For maintenance, individual modules can be selectively and swiftly swapped. Swapped modules can be comfortably returned to the manufacturer for maintenance, whereas a conventional robot would have to be substituted entirely or maintained on-site resulting in longer downtimes.

The first prototypes of modular manipulators start to appear between the late 80s and early 90s, with some pioneering work in [2]–[4]. Later in the 90s, Chen proposed a generic approach for the automatic model generation of reconfigurable robot systems [5] and later exploited it to realize a reconfigurable robotic workcell [6]. From the early 2000s, other applications such as search and rescue robots [7], and space applications [8] have been explored along with self-reconfigurable robots [9], [10], intrarobot reconfiguration [11], or approaches for reconfigurable redundant manipulators [12]. The topic was also investigated more recently by works focusing on spring-assisted modular robots [13], autonomous exploration [14], self-reconfiguration planning [15], or optimization-based design [16], [17]. In the industrial world, a modular arm was developed by Schunk [18] and employed in [19] and [20]. A prototype modular robot was also proposed by Traclabs in [21].

To our observation, works concentrating on the electromechanical module design and modeling of reconfigurable robots have been dominating the literature until recently. Only few—mostly recent—works address how to enable a quickly reconfigurable robot to be also quickly programmable and operational by the user immediately after its reconfiguration. From our perspective, the flexibility benefits offered by reconfigurable robots are tangible only if the time to physically assemble the robot and the time to program it for the desired task are on an equally short time scale.

This directly leads to research questions and challenges concerning the fundamentally necessary capabilities for automated module discovery [22], [23], autogeneration and verification of complete and accurate kinematic and dynamic models, as well as automatic controller generation and tuning [24], [25].

In [26], an axiomatic design is applied to define a general architecture of reconfigurable robot systems that, with respect to previous approaches, allows to deal with more general types of modules and connections. In this regard, the authors introduced an object incidence matrix (OIM), which contains module-specific kinematic/dynamic parameters, extending the assembly incidence matrix (AIM) introduced in [27] to describe the robot topology. Nonetheless, neither of these works has shown how to automatically extract this inherent information from the communication network. Recently a similar work was presented in [28], where the kinematic structure is automatically detected, although no indication of the time scale of the reconfiguration process was given and only kinematic model of the robot was reconstructed, not allowing to implement dynamics-based controller as the one presented in this article.

The key contributions of this article are as follows:

1) A method for the automatic robot model detection and generation, by exploiting four-port EtherCAT network configurations.
2) The software architecture to realize a controller that, independently of the robot configuration, implements a torque-controlled compliant behavior, with no need for user input.

While the model generation method is applicable to any reconfigurable robot, this article focuses on user-reconfigurable cobots of size, payload, and power ratings, equivalent to the commercially available conventional cobots requiring torque-controlled functionality as a crucial requisite. The technological contribution of this article is the introduction of a novel hardware and software architecture for user-reconfigurable cobots, which extends the software-centered flexibility and programmability of the conventional cobots also to the hardware level. The proposed method enables to operate the robot regardless of reconfigurations. It hides the specific hardware configuration details from the user. The presented novel reconfigurable cobot is entirely torque controlled and composed of interchangeable modular plug-and-work components featuring common electromechanical interfaces (EMI). Connecting two modules with the provided EMIs requires only a single torque wrench and less than one minute of time for a nonexperienced user.

As theoretical contribution, this article describes the automated configuration discovery of the connected modules, the kinematic and dynamic model generation, as well as the control and software interface adaptation required after each physical reconfiguration of the robot modules. It discusses all steps involved in the robot reconfiguration process and indicates the technical features enabling it in Section II. This involves the low-level communication systems required for the automated network topology recognition described in Section III, and subsequently, the robot physical topology reconstruction detailed in Section IV. Section V explains the reconfigurable software architecture including decentralized controllers on the module level, a hard real-time (RT) middleware with automatically reconfiguring centralized controllers, and higher level application interfaces. Section VI provides more details on the centralized controller reconfiguration. Section VII illustrates a hardware prototype with an application demonstration. Finally, Section VIII concludes this article.

II. RECONFIGURATION PROCESS OVERVIEW

The overall process of the robot assembly and reconfiguration consists of five steps as depicted in Fig. 2. Initially, the user manually assembles the robot by connecting the modules to realize the desired kinematic configuration suitable to the requirements of the targeted automation task. This is the only manual step where the user input is required. As soon as this step is completed and the physical robot is assembled from the interconnected modules, a network communication is established among the modules and the network topology can be discovered, as described in Section III. This enables to reconstruct the parent/child relationship among the modules. Each type of module also contains a unique identifier that permits after the discovery of the network and the identification of the interconnected modules, to access a database and obtain all parameters of each module. This information is then used to
reconstruct the physical properties of the robot: kinematic, dynamic model, and semantic information such as which module is an End-Effector, or which modules are part of the same kinematic chain. Finally, the software architecture is dynamically reconfigured at the end of the robot reconfiguration permitting the user to operate the robot immediately and perform the required task right after the completion of this process. The rapid reconfigurability of the robot allows the user to quickly iterate the entire process to arrive at the most effective solution. The same process can be performed to quickly adapt the robot configuration to the new task and/or workspace settings.

III. NETWORK TOPOLOGY RECOGNITION

The requirements for the network technology suitable for the implementation of the proposed method are as follows:

1) Since each slave will comprise one electronic device, the topology of the network used in the proposed electronic must permit chaining of slaves to create tree-like robots. Only this way, modular serial kinematic robots, such as robotic arms, and tree-like robots, such as legged locomotion platforms, can be designed.

2) For kinematic and dynamic modeling, it must be feasible to infer the exact network topology, i.e., to extract the graph with slaves being the graph’s nodes and the connections among the slaves being the graph’s edges.

A. EtherCAT Networks

One communication network standard that satisfies these requirements is the EtherCAT standard that is briefly recalled here. In order to reconstruct the robot topology, we assume that for each robot module, the embedded slave device includes at least one EtherCAT slave controller (ESC) chip. Fig. 3 illustrates the base feature of different ESC chips complying with the EtherCAT standard. Each ESC is comprised of an EtherCAT processor unit and at least the Ports 0 to 2. Optionally, more ports (e.g., a Port 3) may exist (dotted port). In the following, only four-port slaves (Ports 0–3), as depicted in Fig. 3, will be considered. The Port 0 has a special function as a so-called upstream port, since it always points toward the network master. 1When the EtherCAT master inserts a data telegram into the ring, it will arrive to a slave through Port 0. The telegram advances to the next port when a port is closed; open ports pass the telegram toward a slave connected to them, until it will be received back through this port and forwarded to the next one. Finally, the telegram arrives back to the master. Any port of the ESC is automatically opened or closed when a communication link on that port becomes active or inactive.

Each ESC exhibits a register, accessible by the master, which holds the open/closed state of each port.

B. Topology Reconstruction Algorithm

The network implementation allows to connect slaves in apparent topologies that are bus, tree-, or star shaped, resulting in robots with series or tree-like kinematic chains. However, an EtherCAT actual network topology is always an open ring where only the organization of the ports in the ESC makes the apparent network topology—and thereby, the robots physical topology—appear differently. The master sees all slaves lined up in a certain order on this ring. In this way, two different apparent network topologies (physical robots) can lead to the same order of slaves on the ring, as Fig. 4 exemplifies.

The example shows two robots, Robots A and B. The apparent topology of Robot A differs from the apparent topology of Robot B in that Slave 5 (Torso module) uses the Ports 0, 1, and 3 for Robot A, whereas for Robot B, the Ports 0, 1, and 2 are being employed. Both apparent network topologies yield the same common actual network topology ring seen by the EtherCAT master. This can be verified by following the outgoing network line across all slaves back to the master. The difference in the apparent network topology leads to a difference in the physical robot topology. The reconstruction of the apparent network topology from the order of the slaves along the network ring is, therefore, the first step to the automatic reconstruction of the robot kinematics and dynamics. This becomes possible by

1To exploit the genderless connectors, a hardware circuit can swap the connectivity of Port 0 to the EMI on demand as in [28]
The EtherCAT master can then reconstruct the apparent network topology from the network ring by looping over all slaves on the ring. Each slave has a parent slave in the tree-like apparent network topology. This parent is a precursor on the network ring, but not necessarily the direct neighbor of the slave. Considering the EtherCAT network rules, it is possible, by looking at the open ports of each slave on the ring, to determine its parent, as implemented in the simple Open EtherCAT Master [29]. The result of this step is a graph structure $\chi$ representing the apparent network topology with the slaves being the nodes and the connection among the ports being the edges of the graph. As in [5], the graph $\chi$ can be represented in a more compact form by an AIM, with the difference that in our graph all types of modules (even joints) are vertexes. Every slave of the network (joints and links alike) is treated equally so that the robot topology can be detected.

### IV. ROBOT PHYSICAL TOPOLOGY RECOGNITION

At this point, we identified the graph $\chi$ describing the network topology with each node representing an EtherCAT slave (see Section III). For each node, we can extract the information fully characterizing the module hosting the slave from the centralized database (see Section IV-A).

The convention for the frame assignment described in Section IV-B allows us to expand each slave node into multiple physical nodes as described in Section IV-C. This information is now aggregated into a tree-like data structure $\phi$, where each node represents a physical body module. This new graph $\phi$ contains bodies as nodes, and joints and physical connections as edges; rather than slaves and ports, respectively. This tree reflects the physical topology of the robot and can be converted effectively to universal robot description format (URDF) [30] that will be exchanged among the different software agents. With the given robot model, we can compute kinematics and dynamics quantities as according to Featherstone [31] (see Section IV-D).

In Fig. 5, a nonexhaustive set of possible representations of robotic modules are depicted. Each robotic module comprehends at least one of the electronic devices with an embedded ESC, where available ports are associated to an EMI to connect to other modules. Robot modules can for instance be Link and Joint modules. The ESC inside a Joint or Link module features either one single (open end, last member of a chain) or two EtherCAT connections (interior member of a chain). End-Effector modules...
can only be placed as the ends of a chain and can have arbitrary application-related purposes such as sensor modules, gripper and tools modules, or mobility modules such as feet or wheels for various types of locomotion.

Base modules function as a control unit and allow to branch the network, and therefore, the robot’s kinematic chain. It includes a minimum of one ESC, as in the other modules, in which all the four ports are available for communication. The Base module also contains an embedded PC where, together with the other software modules, the EtherCAT master is running, controlling the communication with the slaves and providing a command interface to the user. To this end, the Port 0 is used to connect to the EtherCAT master, leaving three available ports that can be used to connect robot branches. More ESCs can be combined in the Base module so that a higher number of branches can be connected. For example, the Base module presented in Fig. 5 combines two ESCs communicating with the EtherCAT master, thereby, allowing communication with five robot branches. In a similar way, Hub modules can be created with one or more ESCs, just like the Base modules without the EtherCAT master. Hub modules enable multiple branching points at any point in the structure.

A. Module Database

After creating $\chi_j$, each slave is asked for its Module identifier stored in the microprocessor of the slave device. This identifier serves as a key to lookup the module properties in a centralized module database. The module identifier code embeds the following:

1) A Module type, which can be: 1 for an active Joint, 2 for a Base, 3 for an End-Effector, and 4 for a passive Link.
2) A Module id, which is used to distinguish between different modules of the same type. For instance, it will be 1 for a straight Joint module and 2 for an elbow Joint module.
3) A Module size, which is 1 for small size modules, 2 for medium size, and 3 for large size.
4) A Module revision number, with increasing value for successive versions of the module design.

The properties stored for each module type and version inside the centralized module database comprehend at least the following:

1) The coordinate transformations between the coordinate frames assigned to the module.
2) The inertial parameters of the module. This includes the center of mass (CoM) coordinates w.r.t. the upstream frame, and the module mass along with the module inertia parameters w.r.t. the CoM. For Joint modules, the parameters are stored separately between the upstream body and the downstream body of the joint.
3) Semantic information describing the common purpose of the module, which identifies, for example, an End-Effector module as a “gripper” or a “foot” or a “wheel.”
4) Parameters for possible kinematic, differential or dynamic constraints associated with the module. Joint motion range, torque, and speed limits are common constraints.

B. Module Coordinate Frame Assignment

We introduce a coordinate frame convention to formalize the automatic model generation. A coordinate frame on each connection interface and on each joint axis define all the kinematic properties of a module. An additional reference frame on each moving body CoM, with axes parallel to the upstream frame, locates the inertial properties of the module. These coordinate frames \( \{ f \} \) and the relative transformations \( T \) between them enable the reconstruction and description of the robot in URDF (see Section IV-C).

The frame axes are named as X, Y, and Z. As a convention in this article, the Z-axis of the module interface always points downstream. The optional leading subscript indicates a special purpose, such as a “j” means that the frame defines the joint axes of action. Frames without the optional purpose subscript indicate the placement of one of the interfaces of the module. The trailing superscript is the index or identifier associated with the ESC, while the trailing subscript refers to the port of the ESC: \( \{ \text{purpose}_f^{\text{slave}_{\text{id}}} \} \).

The coordinate frames are kept parallel to the upstream frame where possible. If a reorientation of one coordinate frame with respect to the upstream frame is necessary, the approach is to rotate about the minimum number of axes. The two frames in a connected EMI are by default coincident. They can have an orientation offset \( \Omega \) about the common axis. For joint axes indicated by the purpose subscript “j,” the axis of action is always the Z-axis.

The coordinate frame assignment for Link and End-Effector modules is straightforward. In Fig. 6, example I shows the frame convention when two Joint modules are connected to each other. Example II shows the frame convention applied to a Base module. It is visible that for each port, there is a coordinate frame attached to the EMI.

Fig. 6. Example of frames definition for the different type of modules and their connections. The circles indicate frame origins. For better visibility of the frame orientation, some coordinate triad are visualized off set. Dashed lines connect the triad to their coordinate origin.

5) Links to a 3-D mesh file graphically representing the module body, and coordinate transformation between the upstream frame of the body and the mesh origin.
C. Modeling and URDF/SRDF Generation

The coordinate frames convention introduced in Section IV-B, permits to unequivocally derive the robot physical model. Since reference frames associated to paired connection interfaces will be coincident, the relative kinematics between two subsequent modules will depend solely on parameters of the parent module. Following the notation in [31] for kinematic trees, we can denote as $\lambda(k)$ the parent of a module of index $k$. The $T_{\lambda(k),k}$ transformation matrix between modules $\lambda(k)$ and $k$, can be defined as the one between the frames associated to the input Port 0 of the modules $\{f_0^{\lambda(k)}\}$ and $\{f_0^k\}$, by

$$
T_{\lambda(k),k} = T_{f_0^{\lambda(k)},f_0^k} = T_{f_0^{\lambda(k)},f_{\text{out}}^{\lambda(k)}} \cdot T_{f_{\text{out}}^{\lambda(k)},f_{\text{out}}^k} = T_{0,\text{out}}^k
$$

where $T_{0,\text{out}}^k \in SE(3)$ and $p_{\text{out}} \in \{0, 1, 2, 3\}$ is the port number where the subsequent module is connected. It assumes the dummy value of 0 only in case of the End-Effector module. For a module $k$, which can be any node in $\chi$, the transformation depends only on the parent node $\lambda(k)$ and the edge connecting the two nodes, which embeds the value of $p_{\text{out}}$. Therefore, the relative forward kinematics between two modules $a$ and $b$ can be computed by traversing the graph from node $a$ and iteratively call (1) until $b$ is reached.

Algorithms such as those in [5] and [32] can be used to obtain the full forward kinematics model. The modular kinematics for any module $k$ is

$$
T_{0,\text{out}}^k = \begin{cases} 
T_{0,j}^k \cdot e_{s_j}^k \cdot \hat{q}_k \cdot T_{j,\text{out}}^k, & \text{if type = Joint} \\
T_{0,\text{out}}^j, & \text{if type = Link, Base} \\
T_{0,\text{out}}^k, & \text{if type = End-Effector}
\end{cases}
$$

where the aforementioned transformation matrices are as follows: $T_{0,j}^k$ from the frame $\{f_0^k\}$ to $\{f_j^k\}$, $T_{j,\text{out}}^k$ from $\{f_j^k\}$ to $\{f_{\text{out}}^k\}$, describing the proximal and distal parts of the Joint module, respectively, $T_{0,\text{out}}^j$ between input port (0) and the output port for Link/Base modules, and $T_{0,\text{out}}^k$ between input port and the TCP of End-Effector modules.

$q_k$ is the joint displacement of the module $k$. $\hat{q}_k \in se(3)$ is the twist of the joint of module $k$ expressed in the frame $\{f_{\text{out}}^k\}$. The 6-D coordinate vector $s_j^k$ representing the twist coordinate of the joint axis is constant, with $s_j^k = [0, 0, 0, 0, 1]^T$ for revolute joints and $s_j^k = [0, 0, 1, 0, 0, 0]^T$ for prismatic joints.

By traversing the graph $\chi$, each node can be expanded by applying (2) and using the data retrieved from the database, to obtain the graph $\phi$. For example a Joint module node is expanded into two nodes representing the proximal and distal bodies, connected by an edge representing the actuated joint. The nodes store the dynamical parameters of each moving body and the edges the transformations relating the bodies to each other. This could be converted to an OIM [26] or AIM [27] for a more compact representation. Considering the URDF format (an XML format) as a de-facto standard for robot operating system (ROS)-based libraries, we convert the graph $\phi$ to an URDF file to describe the kinematics and dynamics model of the robot, representing the robot as a series of link elements (which consists of inertial, visual, and collision properties) connected by either fixed, prismatic, or revolute joint elements. The mapping between nodes and edges of $\phi$ and the URDF XML elements is, therefore, 1:1. Note that the static physical connection between two links (for instance between the distal and proximal body of two successive Joint modules) is represented by a fixed joint element, leaving to the URDF parser of the dynamic library of choice, the task of computing the dynamic parameters of the composite body.

Semantic information of the robot, such as a description of the end-effectors, kinematic chains, and joints composing them, is instead written in a semantic robot description format (SRDF), introduced by the MoveIt! framework [33] to complement the URDF.

D. Kinematic and Dynamic Algorithms

The dynamic library of choice derives unequivocally the kinematic and dynamic model from the URDF. Particularly efficient implementations of dynamic libraries, suitable for computations in a high-frequency RT control loop, are the ones using spatial algebra notation [31] such as kinematics dynamics library (KDL) [34], Pinocchio [35], or rigid body dynamics library (RBDL) [36]. In our current implementation, the latter was used, although any of these libraries allows to numerically compute from the URDF input all the main kinematic and dynamic quantities. The main algorithms to describe rigid body dynamics implemented are as follows:

1) The inverse dynamics or recursive Newton–Euler algorithm (RNEA)

$$
\tau = \text{RNEA(model, } q, \dot{q}, \ddot{q}) = M(q)\ddot{q} + n(q, \dot{q}).
$$

2) The forward dynamics or articulated body algorithm (ABA)

$$
\ddot{q} = \text{ABA(model, } q, \dot{q}, \tau).
$$

3) The composite rigid body algorithm (CRBA)

$$
M(q) = \text{CRBA(model, } q)
$$

where model is a data structure obtained by parsing the URDF. In particular, for the controller implemented in Section VI.C, the RNEA algorithm is used to compute the Coriolis-centrifugal and gravitational forces $n$ and the CRBA to compute the manipulator mass matrix $M$. Other quantities such as point Jacobians, accelerations and velocities can be computed from the model structure as in [31].

V. RECONFIGURABLE SOFTWARE ARCHITECTURE

To exploit the reconfigurability of the hardware, also the software architecture should provide the necessary flexibility to automatically adapt to the new robot topology. It enables to make the robot a plug-and-work system, ready to be operated right after assembly, providing access to all the required API, without the need to define a new controller or the need for user input and tuning. To achieve the aforementioned, the proposed
The software architecture is structured on three main components, which are going to be described from the lower to the higher level.

### A. Module Level

The firmware on each hardware module allows communication over the EtherCAT network, with an interface providing state measurements to higher levels of the architecture. The lower level implementation of active modules, e.g., joint and wheel modules includes a decentralized controller, which drives the module given the references from the higher levels.

### B. Middleware Level

The middleware level exploits XBot [37], a cross-robot software framework that abstracts the diverse variability of the robotic hardware, assuring deterministic hard RT performance and delivering enhanced flexibility through a plug-in architecture. XBot provides to the user a standard API to communicate with the robot, regardless of its specific structure (manipulator, humanoid, quadruped, etc.), and independently of the particular software layer that the user wanted to operate within, requiring only the URDF and SRDF files. It, therefore, adapts its API to the discovered robot topology. Changing the robot topology, adding, for example, a kinematic chain, results in a different API that is compatible with the available components of the robot to control.

In the center of Fig. 7, the different components of XBot are depicted, which are detailed in [37]. The lowest layer implements the EtherCAT master, which realizes the bidirectional communication between the centralized software components and the decentralized ones implemented at the firmware level inside robot hardware modules. The main component of this architecture level is the Plugin Handler, which is responsible for running different RT plugins and executing them sequentially. One or more Communication Handler instead organize the communication to the third non-RT layer that is the application level. The XBotInterface provides an RT and non-RT API that are generated based on the model description obtained via the physical topology recognition.

### C. Application Level

This third architecture level is the non-RT application software level. Software modules in this level run in non-RT threads, which asynchronously receive data from the middleware level and send command references to it. For instance, the interface to the ROS framework is built-in with XBot, allowing for integration with user-defined and third party ROS nodes.

To further simplify the development at the application level, the CartesI/O [38] library provides an autogenerated ROS API allowing the user to specify reference trajectories in the Cartesian space. It generates trajectories online and executes them in a hard RT control loop (inside a XBot RT plugin), while ensuring the RT safeness of the operations.

The reconfigurability on this level is intrinsic as the applications can run and terminate at arbitrary moments in time and interact with the underlying architecture levels through requests to a Communication Handler.

### VI. RECONFIGURABLE CENTRALIZED CONTROL

All safety critical centralized controllers such as interaction, force, and impedance controllers run as RT plugins in the middleware architecture level. The controllers must respect the robot dynamics limits and remain stable even if the robot physical topology changes, which necessitates the reconfigurability also in the control architecture.

#### A. Optimization-Based Control

The proposed control architecture makes use of the OpenSoT library [39], which exploits quadratic programming (QP) and the concept of Stack of Task [40] to execute multiple tasks and achieve complex whole-body motion behaviors.

Tasks are mathematically described as convex quadratic programs, formulated as cost functions to minimize under linear constraints. The benefit of this approach is that global minimizers to such optimization problems are fast and efficient with guaranteed convergence. For example, a generic task can be described as the weighted least-square cost function

\[
\Gamma(x) = \|Ax - b\|_W^2
\]
defined by the matrix $A$, the column vector $b$, a weighting matrix $W$, and the robot state vector $x$. Tasks can be defined in the joint space, with the purpose to minimize joint state variables (e.g., torques or velocities) or other indexes, or in the Cartesian space, with the goal to impose a specific behavior to the robot (e.g., controlling the CoM or an end effector).

Constraints are defined by a generic function that can take the form of a linear inequality constraint

$$ l \leq Cx \leq u $$

where the column vectors $l$ and $u$ are lower and upper bounds, and the matrix $C$ is the constraint matrix. Again, each column of the matrix $C$ corresponds to a DOF of the robot. In the same way, each row corresponds to a constraint. Consequently, adding or removing a robot DOF changes the number of columns of $C$. The rows of the matrix $C$, and thereby, also the number of elements in the vectors $l$ and $u$ change with the number of constraints being active to the task.

Tasks are atomic entities, which can be executed concurrently with soft or hard priorities. Soft priorities are assigned to “equally important” tasks by the weighted superposition of their individual cost functions. Equivalently, the concurrent tasks might be added as rows to the existing cost function. Hard priorities are assigned by minimizing cost functions of lower priority tasks subject to the preserved optimality of higher priority tasks using null-space projection techniques, equality constraints, or similar methods.

### B. Controller Reconfiguration Principle

Controller reconfiguration is, therefore, equivalent to adding or removing rows or columns in the matrix equations of cost functions and constraints. The job of automatically composing the mathematical formulation of the controller is carried out by OpenSoT with the inputs given from the automatically discovered physical robot topology and the user-defined stack of tasks. A key feature of this step is that the user is not required to update the control task just because a new module has been added. End-effector task frames can be automatically shifted to new end-effector locations. The task formulation w.r.t. the end effector remains identical and numeric dimensions of the optimization problem are updated automatically.

### C. Impedance Controller Implementation

A Cartesian impedance controller has been selected for demonstration in this article. It is particularly suited for tasks involving physical interaction with the environment. An impedance control law relates the wrench exerted on the environment by the manipulator’s end effector $F_{\text{ext}} \in \mathbb{R}^n$ to the deviation of its actual Cartesian position $x \in \mathbb{R}^m$ from the desired equilibrium point $x_d$, according to the law

$$ \Lambda_d \ddot{e} + D_d \dot{e} + K_d e = F_{\text{ext}} $$

where $\Lambda_d$, $D_d$, and $K_d$ are $m \times m$ matrices representing the desired mass, damping, and stiffness, respectively, and $e = x - x_d$ is the Cartesian error. The damping matrix is updated at each control loop, depending on the desired stiffness, the desired damping ratio, and the actual Cartesian mass matrix of the arm, e.g., with one of the techniques described in [41].

Without the external force feedback, the desired mass matrix cannot be chosen arbitrarily, but it is determined by the equivalent Cartesian mass matrix $\Lambda(q)$ as

$$ \Lambda_d = \Lambda(q) = (J(q)M(q)^{-1}J(q)^T)^{-1} $$

where $J(q)$ is the task Jacobian matrix of size $m \times n$ and $M(q)$ is the manipulator’s mass matrix of size $n \times n$.

As in [41], the link-side dynamics of a robot with $n$ flexible joints can be described by

$$ M(q)\ddot{q} + n(q, \dot{q}) = \tau_d + J(q)^T F_{\text{ext}} $$

where $q \in \mathbb{R}^n$ is the joint state vector, $n(q, \dot{q}) \in \mathbb{R}^n$ is the vector of gravitational and Coriolis-centrifugal forces, and $\tau_d \in \mathbb{R}^n$ is the desired torque vector. By selecting $\tau_d = \delta \tau + n(q, \dot{q})$ to compensate for the nonlinear terms, the closed-loop behavior (8) is achieved by choosing a $\delta \tau \in \mathbb{R}^n$ solution of the following QP [42]:

$$ \min_{\delta \tau \in C} ||J(q)(M(q)^{-1} \delta \tau - J(q)M(q)^{-1}J(q)^T f)||^2 $$

where $f = \Lambda(q)({\dot{x}}_d - J(q)\dot{q}) - D_d \dot{e} - K_d e$ and $C$ is the set of linear constraints as in (7) that imposes limits on position, velocity, acceleration, and torque. Equation (11) represents the main task in Cartesian space, for which one of the solutions is the well-known $\tau^* = J(q)^T f$. In the case of redundant manipulators, the nullspace dynamics can be determined by adding dynamically consistent (see [41] and [43]) lower priority tasks expressed in the joint space (e.g., a postural task) or in the Cartesian space (e.g., conditions on other frames of the manipulator). We add the nullspace resolution task to the second level of priority in the Stack of Tasks. In this way, the structure of the controller remains unchanged when modifying the physical structure of the robot even in the presence of redundancy (e.g., adding modules to realize a 7-DOF manipulator).

We can see how (11) is written as in the generic form (6), where $A = JM^{-1}$ is an $m \times n$ matrix, while $x = \delta \tau$ is an $n \times 1$ vector and $b = JM^{-1}J^T f = \Lambda^{-1} f$ is an $m \times 1$ vector.

This controller formulation is particularly suited for reconfigurable robots since adding and removing modules to the structure will just change the shape of the $A$ matrix and the length of the robot state vector $x$, while no change to the controller or retuning of control parameters is needed.

### VII. Experimental Results

To validate the presented method, a reconfigurable robot prototype consisting of several straight and elbow Joint modules, an End-Effector module, and a Base module has been developed. With these basic modules, several tree-like robots can be created. The Joint modules are actuated by the Alberobotics actuators [44], [45], which feature integrated torque sensing, high power-to-weight ratio, and broad torque and velocity ranges. The end effector used in this prototypical implementation (Tool exchanger) includes a magnetic actuator to quickly attach or detach different tools. For this example, we use two EMs of the Base module to attach module chains. The Base module contains
a power board and a compact PC with an RT operating system to run the centralized software modules up to the middleware level. The application level software runs on an external PC communicating with the embedded PC of the Base module via WiFi.

All module prototypes comprehend the slave device required by the proposed method. The EMIs allow to quickly establish a reliable mechanical connection that ensure an uninterrupted power and communication link between the modules. It is realized by a hollow flange with a conical shape accommodating the power and communication bus connectors as shown in Fig. 8. The mechanical connection is established by mating the flanges of two modules with a pair of identical C-couplings, which can be tightened together with two screws. Such an interconnection establishes a form closure around the interface, and was also exploited in [46].

For the electrical connection instead, we exploited a commercial MPTC/MPSC connector with EIA-364 standard test approval for shock and vibration. This electrical connector allows for axial clearance of about 1 mm and is mounted on a printed circuit board with a rubber support for compliance w.r.t. the mechanical connector.

The proposed mechanical interface was designed to withstand a worst-case resultant moment of 170 Nm, with a safety factor of 1.5. The corresponding deformation/displacement is up to a maximum of 0.3 mm that guarantees the electrical connectivity.

A. Automatic Discovery Experiment

Fig. 9 illustrates automatic topology discovery and recognition with the experimental setup. The example is a tree-like robot with two chains extending from the Base module [see Fig. 9(1)]. Following Section III, the EtherCat master discovers the apparent network topology as a graph $\chi$ with each EtherCAT slave represented as a node [see Fig. 9(2)]. The displayed graph contains the position of the slave in the EtherCAT ring and the number of open ports. A 4-bit word identifies which ports of the ESC are open. For each port, the corresponding bit is set to one if open or zero otherwise. For each module, the Module Identifier is collected, which contains the following four fields characterizing it: type, id, size, and revision as described in Section IV. With the identifier, the information characterizing each module is retrieved and aggregated to $\chi$ [see Fig. 9(3)].
Each node in the tree represents a module and contains all the associated kinematic, dynamic, and semantic data of the module. From this tree structure, the physical robot topology $\phi$ can be obtained [see Fig. 9(4)], and represented in the URDF format [see Fig. 9(5)].

In Fig. 9(6), the output of the process, the URDF model, is rendered to assess the correct model generation.

At this point, both the simulation and robot hardware are ready to be used. The time required for building a robot structure from modules and automatically generate the model takes on average less than 5 min [compare also Fig. 10(2)].

### B. Cartesian Task Experiment

To show the efficacy of the introduced method, we present an application example, where the same task is performed with two different robot configurations. In the example task, the robot is instructed to draw on two sheets of article. One sheet is further away than the other and necessitates a higher reachability of the arm. The task is repeated twice: once with 4 DOF (closer sheet) and one with 5 DOF (to reach the second farther sheet). The tool held by the end effector is a simple pen.

Fig. 10 captures the main phases of the experiment starting from building the first 4-DOF robot [see Fig. 10(1)]. The automatic discovery phase [see Fig. 10(2)] comprises the network and physical robot topology recognition, and the controller reconfiguration. Next, the robot and controllers are started to kinesthetically teach and execute the drawing task [see Fig. 10(3)]. The whole procedure repeats for the 5-DOF robot in the adapt, discovery, and repeat phases [see Fig. 10(4)–(6)]. After in about 13 min, the entire experiment is completed.$^2$

For this experiment, a non-RT state machine runs at the application level of the software architecture of Section V to control the behavior of the robot. From the GUI, the user can put the robot in teaching mode to calibrate the workspace, specify the drawing to execute, and select the control parameters (stiffness and damping ratio).

The rest of the procedure executes automatically. The controller does not need to be modified by the user after reconfiguration, since the software modules already reconfigure the middleware API and the controller architecture. In the automatic model generation, the only difference is the presence of one more module with an additional DOF in the kinematic chain. This is handled automatically in the discovery phase [see Fig. 10(5)]. For this task, the defined controller uses a Stack of Tasks [39] with a translation and orientation Cartesian impedance task on the higher level of priority, and a Postural task on the second and last level of priority as explained in Section VI. The postural is required for robots that are redundant with respect to the task. The Stack of Tasks is defined a priori before the experiment.

In Fig. 11, the actual position of the end effector and the given reference are plotted in Cartesian coordinates, both for the 4-DOF and 5-DOF robots. The stiffness values used for this experiment are the same for the two robots, with a translational stiffness on the $x$-, $y$-, $z$-axes of 2500 [N/m], and rotational stiffness on the roll and pitch rotations of 20 [Nm/rad].

$^2$Video of the experiment is available at: https://youtu.be/DHi7alHcpE
with null stiffness on the yaw axis. This proves there is no need to retune the controller parameters after reconfiguration. The values for the damping matrix are instead computed as in [41] by setting a damping ratio of 0.7. Maximum error on the x–y plane during the drawing task is 0.0024 m for the 4-DOF robot and 0.0033 m for the 5-DOF robot.

With identical controller gains, the increased tracking error for the 5-DOF robot can be explained by accumulated model errors when higher number of modules are involved. One way to reduce the tracking error is raising the stiffness gains. Another option is enhancing the model accuracy for each module, i.e., from identification to decrease error contributions from model-based compensation terms. This will be part of future work.

The proposed method, comprising of the automatic kinematic and dynamic model generation, provides all the tools to implement other types of controllers capable of improving the trajectory tracking performance, although sacrificing the compliance behavior property of the impedance control, with techniques as those proposed in [47] and [48].

VIII. CONCLUSION

The article presents a novel method and a complete architecture for quickly hard- and software reconfigurable robots. The architecture comprises a set of modules with modular electromechanical communication, power, and mounting interfaces along with a fully reconfigurable software stack. The software stack comprises the lowest module firmware level, a hard RT middleware for centralized control and safety-critical algorithms as well as a higher non-RT application level.

Using the proposed method, the article successfully demonstrates and exemplifies how a robot can be built, programmed for a task, commanded to execute it, reconfigured for a larger workspace, and commanded again to execute the task in minimal time within 13 min approximately. The time to physically assemble and program the robot for the desired tasks is on equally short time scales. This paves the way to exploit the versatility of reconfigurable modular robots to cope with frequent product variation, minimize maintenance downtimes, and build new robot designs on-demand. The proposed approach enables robots not only to assist in the production of mass customizable products, but permits robots to become mass-customizable themselves.

Future research of this work will focus on the development of software assistant tools to help the user identify the optimal robot configuration for formally defined task requirements as well as on the further extension of the presented approach to floating-base robots for locomotion, mobile manipulation, and combined loco-manipulation tasks.

REFERENCES

[1] “Robot adoption: The SME challenge—National report autumn 2019–20,” Accessed: Mar. 30, 2020. [Online]. Available: https://www.themanufacturer.com/articles/robot-adoption-the-sme-challenge
[2] D. Schmitz, P. Khosla, and T. Kanade, “The CMU reconfigurable modular manipulator system,” Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-RI-TR-88-07, 1988.
[3] R. Cohen, M. Lipton, M. Dai, and B. Benhabib, “Conceptual design of a modular robot,” J. Mech. Design, pp. 117–125, Mar. 1992.
[4] T. Matsumura, “Design and control of the modular robot system: TOMMS,” in Proc. IEEE Int. Conf. Robot. Autom., May 1995, vol. 2, pp. 2125–2131.
[5] I.-M. Chen and G. Yang, “Automatic model generation for modular reconfigurable robot dynamics,” Trans. ASME, J. Dyn. Syst. Meas., Control, vol. 120, no. 3, pp. 346–352, Sep. 1998.
[6] I.-M. Chen, “Rapid response manufacturing through a rapidly reconfigurable robotic workcell,” Robot. Comput. Integ. Manuf., vol. 17, no. 3, pp. 199–213, 2001.
[7] H. Zhang, W. Wang, Z. Deng, G. Zong, and J. Zhang, “A novel reconfigurable robot for urban search and rescue,” Int. J. Adv. Robot. Syst., vol. 3, no. 4, pp. 359–366, 2006.
[8] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans, “Modular reconfigurable robots in space applications,” Auton. Robots, vol. 14, no. 2–3, pp. 225–237, 2003.
[9] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, “M-TRAN: Self-reconfigurable modular robotic system,” IEEE/ASME Trans. Mechatron., vol. 7, no. 4, pp. 431–441, Dec. 2002.
[10] M. Yim, Y. Zhang, K. Roufas, D. Duff, and C. Eldershaw, “Connecting and disconnecting for chain self-reconfiguration with polybot,” IEEE/ASME Trans. Mechatron., vol. 7, no. 4, pp. 442–451, Dec. 2002.
[11] A. Castano, A. Behar, and P. M. Will, “The Cono modules for reconfigurable robots,” IEEE/ASME Trans. Mechatron., vol. 7, no. 4, pp. 403–409, Dec. 2002.
[12] M. W. Pyror, R. C. Taylor, C. Kapoor, and D. Tesar, “Generalized software components for reconfiguring hyper-redundant manipulators,” IEEE/ASME Trans. Mechatron., vol. 7, no. 4, pp. 475–478, Dec. 2002.
[13] G. Liu, Y. Liu, and A. A. Goldenberg, “Design, analysis, and control of a spring-assisted modular and reconfigurable robot,” IEEE/ASME Trans. Mechatron., vol. 16, no. 4, pp. 695–706, Aug. 2011.
[14] S. Russo et al., “Design of a robotic module for autonomous exploration and multimode locomotion,” IEEE/ASME Trans. Mechatronics, vol. 18, no. 6, pp. 1757–1766, Dec. 2013.
[15] T. Zhang, W. Zhang, and M. M. Gupta, “An underactuated self-reconfigurable robot and the reconfiguration evolution,” Mechanism Mach. Theory, vol. 124, pp. 248–258, 2018.
[16] S. Ha, S. Coros, A. Alsipach, J. Bern, J. M. Kim, and K. Yamane, “Computational design of robotic devices from high-level motion specifications,” IEEE Trans. Robot., vol. 34, no. 4, pp. 1204–1251, Oct. 2018.
[17] A. Zhao et al., “RoboGrammr: Graph grammar for terrain-optimized robot design,” ACM Trans. Graph., vol. 39, no. 6, pp. 1–16, 2020.
[18] SCHUNK GmbH & Co KG, 2012, Accessed: Mar. 30, 2020. [Online]. Available: https://schunk.com
[19] M. Brandstötter, A. Angerer, and M. Hofbaur, “The curved manipulator (CUMA-type arm): Realization of a serial manipulator with general structure in modular design,” in Proc. 14th IFToMM World Congr., 2015, pp. 403–409.
[20] E. Icer, A. Giusti, and M. Althoff, “A task-driven algorithm for configuration synthesis of modular robots,” in Proc. IEEE Int. Conf. Robot. Autom., 2016, pp. 5203–5209.
[21] R. R. Burridge, M. W. Chu, and B. T. Wolfe, “Apparatus, systems, and methods for reconfigurable robotic manipulator and coupling,” U.S. Patent 9475199, Oct. 25, 2016.
[22] J. Baca, B. Woosley, P. Dasgupta, and C. Nelson, “Real-time distributed configuration discovery of modular self-reconfigurable robots,” in Proc. IEEE Int. Conf. Robot. Autom., 2015, pp. 1919–1924.
[23] V. M. Vilches, “Metodo de determinacion de configuracion de un robot modular,” ES Patent 2661067B1, Mar. 2016.
[24] A. Giusti and M. Althoff, “On-the-fly control design of modular robot manipulators,” IEEE Trans. Control Syst. Technol., vol. 26, no. 4, pp. 1484–1491, Jul. 2018.
[25] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, “Effortless creation of safe robots from modules through self-programming and self-verification,” Sci. Robot., vol. 4, Jun. 2019, Art. no. eaaw 1924.
[26] Z. Bi, Y. Lin, and W. Zhang, “The general architecture of adaptive robotic systems for manufacturing applications,” Robot. Comput. Integ. Manuf., vol. 26, no. 5, pp. 461–470, 2010.
[27] I.-M. Chen, “Theory and applications of modular reconfigurable robotic systems,” Ph.D. dissertation, California Inst. Technol., Pasadena, CA, USA, 1994.
[28] A. Yun, D. Moon, J. Ha, S. Kang, and W. Lee, “ModMan: An advanced reconfigurable manipulator system with genderless connector and automatic kinematic modeling algorithm,” IEEE Robot. Autom. Lett., vol. 5, no. 3, pp. 4225–4232, Jul. 2020.
[29] Open EtherCAT Society, Accessed: Mar. 30, 2020. [Online]. Available: https://openthercatsociety.github.io

[30] Universal Robot Description Format (URDF), Accessed: Mar. 30, 2020. [Online]. Available: https://wiki.ros.org/urdf

[31] R. Featherstone, Rigid Body Dyn. Algorithms. Berlin, Germany: Springer, 2008.

[32] T. Zhang et al., “Robotic dynamic sculpture: Architecture, modeling, and implementation of dynamic sculpture,” IEEE Robot. Autom. Mag., vol. 21, no. 3, pp. 96–104, Sep. 2014.

[33] D. Coleman, I. Sucan, S. Chitta, and N. Correll, “Reducing the barrier to entry of complex robotic software: A MoveIt! case study,” J. Softw. Eng. Robot., vol. 5, no. 1, pp. 3–16, 2014.

[34] Orocos Kinematics Dyn. Library. Accessed: Mar. 30, 2020. [Online]. Available: https://www.orocos.org/kdl

[35] J. Carpentier et al., “The Pinocchio C Library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in Proc. IEEE/NICE Int. Symp. Syst. Integ., 2019, pp. 614–619.

[36] M. L. Felis, “RBDL: An efficient rigid-body dynamics library using recursive algorithms,” Auton. Robots, vol. 41, no. 2, pp. 495–511, 2017.

[37] L. Muratore, A. Laurenzi, E. M. Hoffman, and N. G. Tsagarakis, “The XBot real-time software framework for robotics: From the developer to the user perspective,” IEEE Robot. Autom. Mag., vol. 27, no. 3, pp. 133–143, Sep. 2020.

[38] A. Laurenzi, E. M. Hoffman, L. Muratore, and N. Tsagarakis, “CartesI/I: A ROS based real-time capable cartesian control framework,” in Proc. IEEE Int. Conf. Robot. Autom., 2019, pp. 591–596.

[39] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. Tsagarakis, “OpenSoT: A whole-body control library for the compliant humanoid robot COMAN,” in Proc. IEEE Int. Conf. Robot. Autom., 2015, pp. 6248–6253.

[40] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, “A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks,” in Proc. IEEE Int. Conf. Adv. Robot., 2009, pp. 1–6.

[41] A. Albu-Schaffer, C. Ott, U. Frese, and G. Hirzinger, “Cartesian impedance control of redundant robots: Recent results with the DLR-light-weight-arms,” in Proc. IEEE Int. Conf. Robot. Autom., 2003, vol. 3, pp. 3704–3709.

[42] E. M. Hoffman, A. Laurenzi, L. Muratore, N. Tsagarakis, and D. G. Caldwell, “Multi-priority cartesian impedance control based on quadratic programming optimization,” in Proc. IEEE Int. Conf. Robot. Autom., 2018, pp. 309–315.

[43] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” IEEE J. Robot. Autom., vol. 3, no. 1, pp. 43–53, Feb. 1987.

[44] N. Kashiri, et al., “CENTAURO: A hybrid locomotion and high power resilient manipulation platform,” IEEE Robot. Autom. Lett., vol. 4, no. 2, pp. 1595–1602, Apr. 2019.

[45] Alberobotics. Accessed: Mar. 30, 2020. [Online]. Available: https://alberobotics.it

[46] S. Hong, D. Choi, S. Kang, H. Lee, and W. Lee, “Design of manually reconfigurable modular manipulator with three revolute joints and links,” in Proc. IEEE Int. Conf. Robot. Autom., 2016, pp. 5210–5215.

[47] L. Le-Tien and A. Albu-Schäffer, “Robust adaptive tracking control based on state feedback controller with integrator terms for elastic joint robots with uncertain parameters,” IEEE Trans. Contol Syst. Technol., vol. 26, no. 6, pp. 2259–2267, Nov. 2018.

[48] J. Kim and E. A. Croft, “Full-state tracking control for flexible joint robots with singular perturbation techniques,” IEEE Trans. Control Syst. Technol., vol. 27, no. 1, pp. 63–73, Jan. 2019.

**Edoardo Romiti** received the Master’s (Hons.) degree in automation engineering from the University of Bologna, Bologna, Italy, in March 2017. He is currently working toward the Ph.D. degree in robotics with Humanoids and Human-Centered Mechatronics lab, Istituto Italiano di Tecnologia, Genoa, Italy.

**Marco Ruzzon** received the Master’s (Hons.) degree in energy engineering from Politecnico di Milano, Milano, Italy, in April 2017, and the double Master’s Degree in robotics engineering from the University of Genova, Genova, Italy, and Ecole Centrale de Nantes, Nantes, France, in August 2019. He is currently with the Humanoids and Human-Centered Mechatronics lab, Istituto Italiano di Tecnologia, Genoa, Italy.

**Arturo Laurenzi** received the Master’s degree in automation engineering (cum laude) from Università Degli Studi di Firenze, Florence, Italy, in 2015. He is currently working toward the Ph.D. degree with the Humanoid and Human-Centered Mechatronics Laboratory, Istituto Italiano di Tecnologia, Genoa, Italy, on the software architecture and control of the hybrid wheeled-legged quadrupedal robot Centauro.

Since 2019, he has been a Senior Technician with the Humanoid and Human-Centered Mechatronics Laboratory.

**Jörn Malzahn** received the Graduate degree in electrical engineering and the Doctorate in Engineering (Dr.-Ing.) degree with summa cum laude from TU Dortmund University, Dortmund, Germany, in 2008 and 2014, respectively.

In 2015, he joined the Humanoids and Human-Centred Mechatronics Laboratory, Istituto Italiano di Tecnologia, Genoa, Italy, focusing on the actuation and control of compliant robots.

**Navvab Kashiri** received the Ph.D. degree in robotics from the Humanoid and Human-Centred Mechatronics group, Istituto Italiano di Tecnologia (IIT), Genoa, Italy, in 2015. He is currently working as a Senior Postdoctoral Researcher with Humanoids and Human-Centred Mechatronics Laboratory, IIT, focusing on the design, development, and control of mechatronics systems including robotic modules, compliant actuators and sensors, and vari- able impedance devices.

**Francesco Iacobelli** received the Master’s (Hons.) degree in mechanical engineering from the Sapienza University of Rome, Rome, Italy, in 2016. He worked as an Embedded Software Engineer for two years. Since 2019, he has been with the Humanoids and Human-Centred Mechatronics Laboratory, Istituto Italiano di Tecnologia, Genoa, Italy, as a Support Technician, working mainly on control software architecture.
Enrico Mingo Hoffman received the B.Sc. degree in electronics engineering and the M.Sc. degree in AI and robotics, both from the University of Rome “La Sapienza,” Rome, Italy, in 2008 and 2012, respectively, and the Ph.D. degree in humanoid robotics from the University of Genoa, Genoa, Italy, in 2016.

Since 2020, he has been a Researcher with the Humanoid and Human-Centered Mechatronics Laboratory, Istituto Italiano di Tecnologia, Genoa. His research interests include robot motion planning and control.

Luca Muratore received the split-site Ph.D. degree in electrical and electronic engineering from the University of Manchester, Manchester, U.K., and Istituto Italiano di Tecnologia (IIT), Genoa, Italy, in 2020.

He is currently the Lead Software Engineer with the Humanoids and Human Centered Mechatronics research line, IIT. Dr. Muratore is the Coordinator of the ROS End-Effector H2020 ROSIN FTP.

Lorenzo Baccelliere received the B.Sc. degree in biomedical engineering from the University of Genoa, Genoa, Italy, in 2007.

In 2014, he joined the Humanoids and Human-Centred Mechatronics lab, Istituto Italiano di Tecnologia, Genoa, where is currently working as a Senior Technician.

Stefano Cordasco received the Graduate degree in physics from the University of Genova, Genoa, Italy, in 2003.

He joined the Humanoids and Human-Centred Mechatronics Laboratory, Istituto Italiano di Tecnologia, Genoa, in 2010, focusing on the electronic hardware and firmware development for compliant actuators used in robotics.

Alessio Margan received the Master’s degree in computer science from the University of Genoa, Genoa, Italy, in December 2000.

In 2009, he joined the ADVR lab. He is currently with Humanoids and Human-Centred Mechatronics lab, Istituto Italiano di Tecnologia, Genoa.

Nikos Tsagarakis received the M.Sc. degree in control engineering and the Ph.D. degree in robotics from the University of Salford, Manchester, U.K., in 1997 and 2000, respectively.

He is currently a Tenured Senior Scientist and Principal Investigator of the Human-Centred Mechatronics Research Line at Istituto Italiano di Tecnologia, Genoa, Italy.