Towards a Plug-and-Work Reconfigurable Cobot

Edoardo Romiti, Jörn Malzahn, Navvab Kashiri, Francesco Iacobelli, Marco Ruzzon, Arturo Laurenzi, Enrico M. Hoffman, Luca Muratore, Alessio Margan, Lorenzo Baccelliere, Stefano Cordasco, 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 low down-time for maintenance. To address these demands, this work 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 URDF format 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 the paper focuses on reconfigurable manipulators, the proposed concept can support arbitrary serial kinematic tree-like configurations. We demonstrate the contributions with examples of: (a) how the topology of the robot is reconstructed and the URDF model is generated, (b) a Cartesian task application for a cobot built with the basic modules, demonstrating the quick reconfigurabilty of the system from a 4 degrees of freedom (DOF) robot to a 5-DOF robot, in order to satisfy new workspace requirements.**

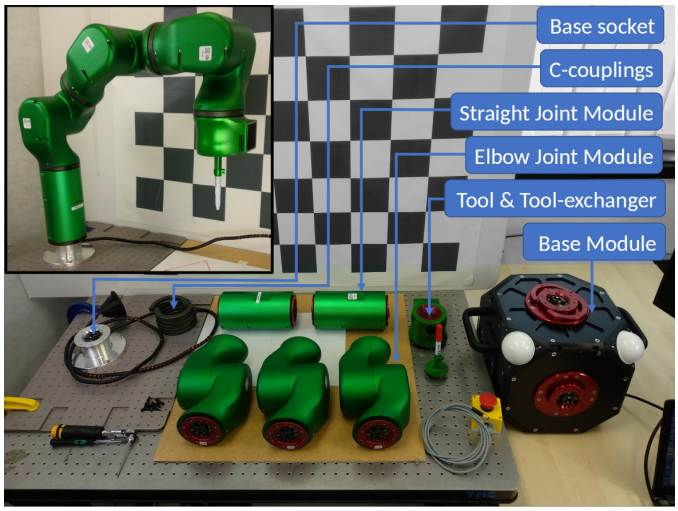

Fig. 1. Tabletop view of all components needed for realising the reconfigurable robot: Joint modules, Tool-exchanger module (an end-effector module designed to hold various tools) and the Base module. The modules can be quickly connected together through C-couplings and standard electro-mechanical interfaces starting from either the Base module or the base socket to build robots as the one in the top-left corner.

I. Introduction

Cobots keep entering new application domains in which robotic automation was not conceivable until recently. Yet, many small and medium sized enterprises (SMEs) 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-customised products [1], requiring new paradigms of highly configurable robot solutions. While being flexibly programmable and customisable from a software perspective, the hardware customisability of existing cobots is limited to the end-effector tooling. Conventional cobots are, strictly fixed in their physical properties such as the number of degrees of freedom (DOF), kinematics (workspace volume and shape) as well as payload capacity. The size and payload capacity appear to be "rigidly" proportionate: the higher the payload capacity, the bigger the robot. These observations motivate the work on reconfigurable robots, i.e. modular robots that can be frequently reconfigured in their physical shape even after their initial build.

On the other hand, quickly reconfigurable robots offer the required versatility to adapt and realise fit-to-task kinematic arrangements on demand, addressing changes in the manufacturing process with minimal down-times of the automation line. In particular, the physical characteristics of reconfigurable robots can be rapidly customised to meet specific and varying application requirements, in terms of payload, workspace, kinematics and task dexterity. Reconfigurable robots can be split 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 down-times.

The first prototypes of modular manipulators start to appear between the late 80-ies and early 90-ies, with some pioneering work in [2]–[4]. Later in the 90-ies, Chen proposed a generic approach for 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], intra-robot 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 industry world a modular arm was

This work has received funding from the European Union's Horizon 2020 project CONCERT under grant agreement No. 101016007.

The authors are with the Humanoids and Human-Centred Mechatronics laboratory, Istituto Italiano di Tecnologia, via Morego, 30, 16163 Genova, Italy (email: edoardo.romiti@iit.it).

developed by Schunk [18] and employed in [19], [20]. A prototype modular robot was proposed also by Traclabs in [21].

To our observation, works concentrating on the electro-mechanical module design and modelling 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], auto-generation and verification of complete and accurate kinematic and dynamic models, as well as automatic controller generation and tuning [24], [25].

In [26], 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 paper.

The key contributions of the paper are:

- a method for automatic robot model detection and generation, by exploiting 4-port EtherCAT network configurations,
- the software architecture to realise a controller that, independently of the robot configuration, implements a torque-controlled compliant behaviour, with no need for user input.

While the model generation method is applicable to any reconfigurable robot, the paper focuses on user-reconfigurable cobots of size, payload and power ratings, equivalent to commercially available conventional cobots requiring torque-controlled functionality as a crucial requisite. The technological contribution of this paper is the introduction of a novel hard- and software architecture for user-reconfigurable cobots, which extends the software-centred flexibility and programmability of conventional cobots also to the hardware level. The proposed method enables to operate the robot regardless of re-configurations. 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 electro-mechanical interfaces (EMI). Connecting two modules with the provided EMIs requires only a single torque wrench and less than one minute of time for a non experienced user.

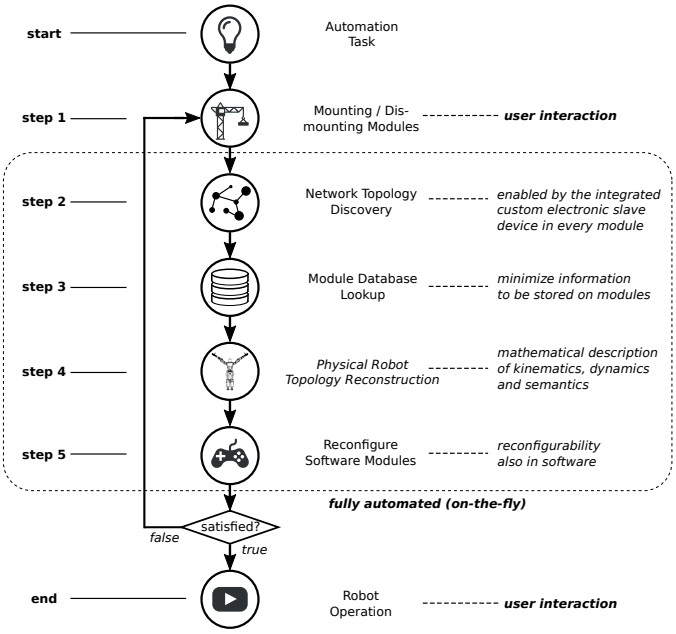

Fig. 2. Overview of the method for automatic model generation and robot reconfiguration.

As theoretical contribution, the paper 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 Sec. II. This involves the low-level communication systems required for the automated network topology recognition described in Sec. III and subsequently the robot physical topology reconstruction detailed in Sec. IV. Sec. V explains the reconfigurable software architecture including decentralised controllers on the module level, a hard real-time middleware with automatically reconfiguring centralised controllers and higher level application interfaces. Sec. VI provides more details on the centralised controller reconfiguration. Sec. VII illustrates a hardware prototype with an application demonstration. Finally, the paper concludes and provides an outlook for ongoing works in Sec. VIII.

II. Reconfiguration Process Overview

The overall process of 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 realise 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 Sec. 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 modelling, 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 (eg. a Port 3) may exist (dotted port). In the following, only 4-port slaves (Port 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 towards the network master[1]. When 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 towards 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, that 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.

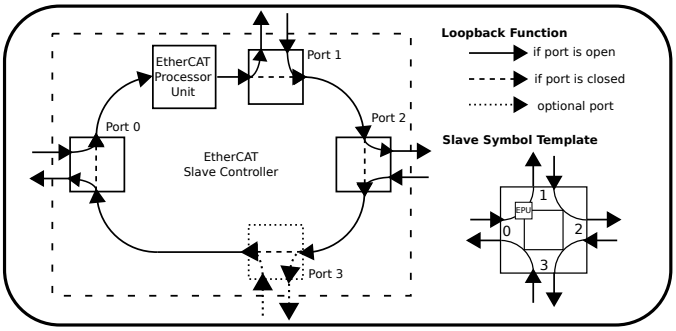

Fig. 3. Illustration of the functioning of the EtherCAT Slave Controller.

The example shows two 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 0,1,2 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 applying the convention that port 0 is always the upstream port and by the ESC's register of open and closed ports, which can be read and set from the EtherCAT master.

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 neighbour 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 χ 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 χ 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) are treated equally so that the robot topology can be detected.

IV. Robot Physical Topology Recognition

At this point, we identified the graph χ describing the network topology with each node representing an EtherCAT

[1]To exploit the genderless connectors, a hardware circuit can swap the connectivity of port 0 to the EMI on demand as in [28]

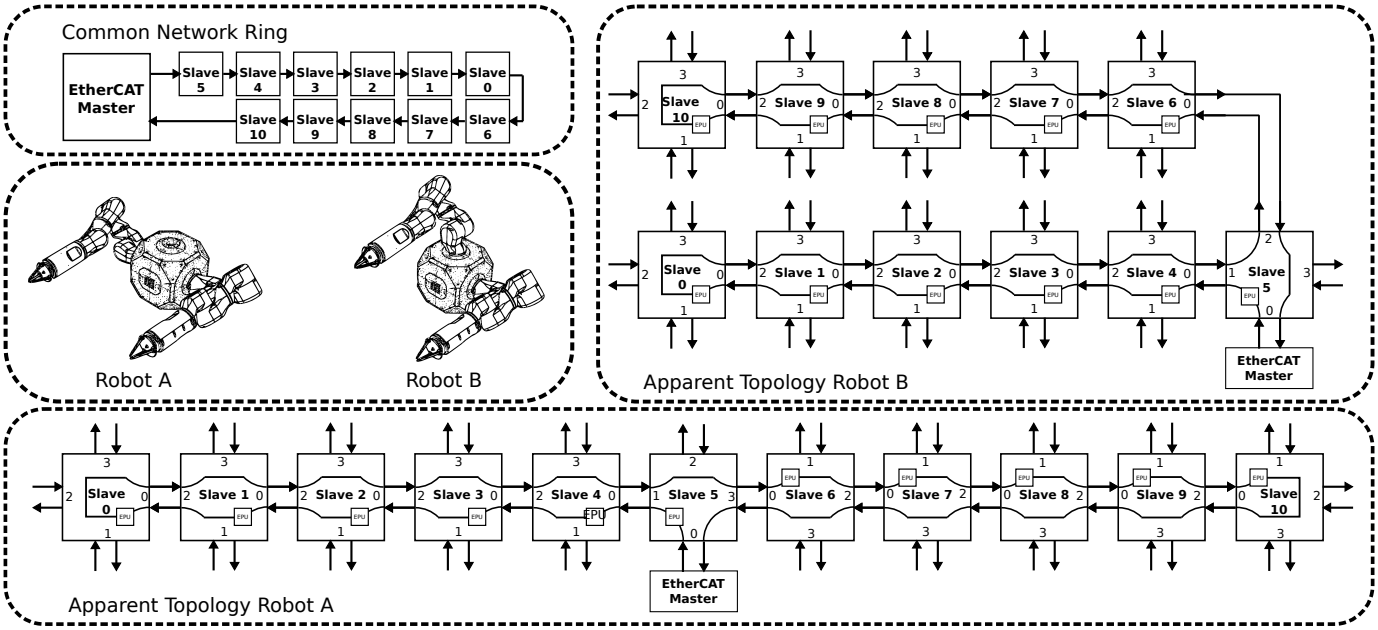

Fig. 4. Robot A and B represent two different robots with different apparent topologies. Anyhow the actual topology is the same network ring in both cases.

slave (Sec. III). For each node we can extract the information fully characterising the module hosting the slave from the centralised database (Sec. IV-A). The convention for frame assignment described in Sec. IV-B allows us to expand each slave node into multiple physical nodes as described in Sec. IV-C. This information is now aggregated into a tree-like data structure ϕ, where each node represents a physical body module. This new graph ϕ 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 URDF (Universal Robot Description Format) [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] (Sec. IV-D).

In Fig. 5, a non-exhaustive 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 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, port 0 is used to connect to the EtherCAT master, leaving three available ports which 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 χ, 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 centralised module database. The *Module identifier* code embeds:

- a *Module type*, which can be: 1 for an active Joint, 2 for a Base, 3 for an End-Effector, 4 for a passive Link
- a *Module id*, which is used to distinguish between different module of the same type. For instance it will be 1 for a straight Joint module and 2 for an elbow Joint module.
- a *Module size*, which is 1 for small size modules, 2 for medium size and 3 for large size.
- 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 centralised module database comprehend at least:

- The coordinate transformations between the coordinate frames assigned to the module.
- The inertial parameters of the module. This includes the centre of mass coordinates w.r.t. the upstream frame, the module mass along with the module inertia parameters w.r.t. the centre of mass. For Joint modules, the parame-

ters are stored separately between the upstream body and the downstream body of the joint.
- 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'.
- Parameters for possible kinematic, differential or dynamic constraints associated with the module. Joint motion range, torque and speed limits are common constraints.
- Links to a 3D mesh file graphically representing the module body, and coordinate transformation between the upstream frame of the body and the mesh origin.

B. Module Coordinate Frame Assignment

We introduce a coordinate frame convention to formalise 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 center of mass (CoM), with axes parallel to the upstream frame, locates the inertial properties of the module. These coordinate frames $\{\boldsymbol{f}\}$ and the relative transformations $\boldsymbol{T}$ between them enable the reconstruction and description of the robot in URDF (see Sec. IV-C). The frame axes are named X, Y, Z. As a convention in this paper, 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: $\{{}_{purpose}\boldsymbol{f}^{slave\_id}_{port\_id}\}$.

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 Ω 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.

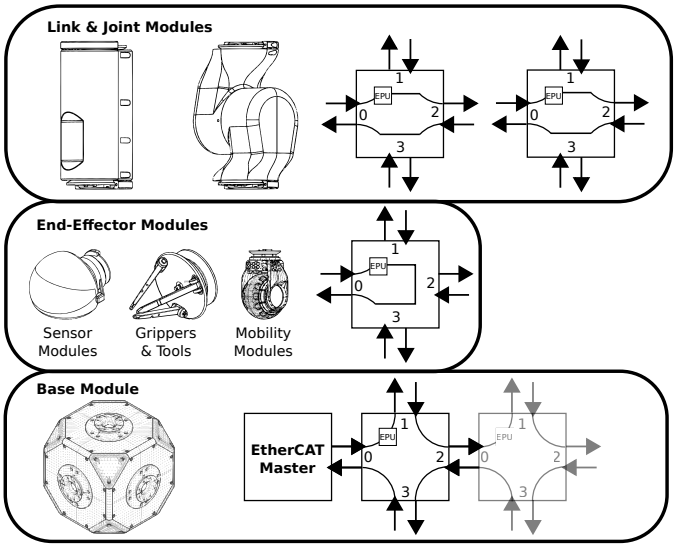

Fig. 5. Family of robotic modules together with their EtherCAT slave communication scheme. The picture shows the ESC integrated in each module and its connectivity. Arrows represent possible connection points. Joint and Link modules use ports 0 and 2 for communicating with the network (port 1 and 3 are closed). End-effector modules use only the upstream port 0. For Base modules all ports can be connected. The second ESC is optional and therefore greyed out in this picture.

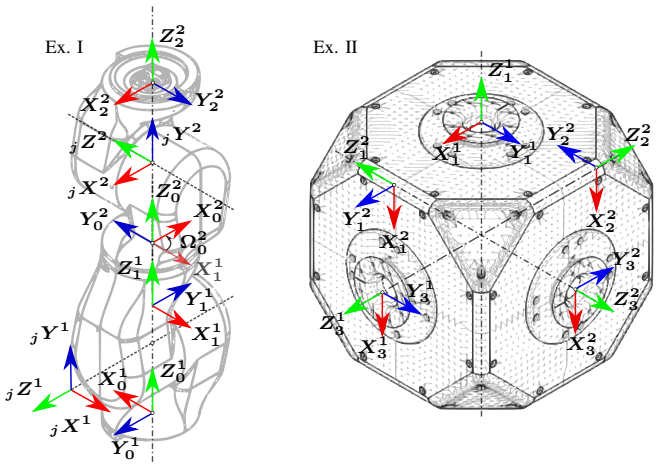

Fig. 6. Example of frames definition for the different type of modules and their connections. The circles indicate frame origins. For better visibility of the frame orientation some coordinate triad are visualised off set. Dashed lines connect the triad to their coordinate origin.

C. Modelling and URDF/SRDF Generation

The coordinate frames convention introduced in Sec. 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 $\boldsymbol{T}_{\lambda(k),k}$ 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

$$\boldsymbol{T}_{\lambda(k),k} = \boldsymbol{T}_{f_0^{\lambda(k)},f_0^k} = \boldsymbol{T}_{f_0^{\lambda(k)},f_{p_{out}}^{\lambda(k)}} \cdot \underbrace{\boldsymbol{T}_{f_{p_{out}}^{\lambda(k)},f_0^k}}_{=\mathrm{I}} = \boldsymbol{T}^{\lambda(k)}_{0,p_{out}} \tag{1}$$

where $\boldsymbol{T}^{\lambda(k)}_{0,p_{out}} \in SE(3)$ and $p_{out} \in \{0,1,2,3\}$ is the port number where the subsequent module is connected. It assumes the dummy value of 0 only in case of End-Effector module. For a module k, which can be any node in χ, the transformation depends only on the parent node $\lambda(k)$ and the edge connecting the two nodes, which embeds the value of p_{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], [32] can be used to obtain the full forward kinematics model. The modular kinematics for any module k is:

$$\boldsymbol{T}^k_{0,p_{out}} = \begin{cases} \boldsymbol{T}^k_{0,j} \cdot e^{\hat{s}^k_j q_k} \cdot \boldsymbol{T}^k_{j,p_{out}}, & \text{if } type = \text{Joint} \\ \boldsymbol{T}^k_{0,p_{out}}, & \text{if } type = \text{Link, Base} \\ \boldsymbol{T}^k_{0,tcp}, & \text{if } type = \text{End-Effector} \end{cases} \tag{2}$$

where the above transformation matrices are as follows: $\boldsymbol{T}^k_{0,j}$ from frame $\{f^k_0\}$ to $\{_j f^k\}$, $\boldsymbol{T}^k_{j,p_{out}}$ from $\{_j f^k\}$ to $\{f^k_{p_{out}}\}$, describing the *proximal* and *distal* parts of the Joint module respectively, $\boldsymbol{T}^k_{0,p_{out}}$ between input port (0) and the output port for Link/Base modules, and $\boldsymbol{T}^k_{0,tcp}$ between input port and the TCP of End-Effector modules. q_k is the joint displacement of module k. $\hat{s}^k_j \in se(3)$ is the twist of the joint of module k expressed in frame $\{f^k_j\}$. The 6-D coordinate vector s^k_j representing the twist coordinates of the joint axis is constant, with $s^k_j = [0,0,0,0,0,1]^T$ for revolute joints and $s^k_j = [0,0,1,0,0,0]^T$ for prismatic joints.

By traversing the graph χ, each node can be expanded by applying (2) and using the data retrieved from the database, to obtain the graph ϕ. 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 ROS-based libraries, we convert the graph ϕ to a 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 ϕ 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 SRDF file (Semantic Robot Description Format), 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 real-time control loop, are the ones using spatial algebra notation [31] such as KDL [34], Pinocchio [35] or 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:

- the Inverse Dynamics or Recursive Newton-Euler Algorithm (RNEA)

$$\boldsymbol{\tau} = RNEA(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \tag{3}$$

- the Forward Dynamics or Articulated Body Algorithm (ABA)

$$\ddot{\boldsymbol{q}} = ABA(model, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}) \tag{4}$$

- the Composite Rigid Body Algorithm (CRBA)

$$\boldsymbol{M}(\boldsymbol{q}) = CRBA(model, \boldsymbol{q}) \tag{5}$$

where $model$ is a data structure obtained by parsing the URDF. In particular, for the controller implemented in Sec. VI.C, the RNEA algorithm is used to compute the Coriolis-centrifugal and gravitational forces $\boldsymbol{n}$ and the CRBA to compute the manipulator mass matrix $\boldsymbol{M}$. 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 above, the proposed 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 decentralised 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 Real-Time (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 centre of Fig. 7 the different components of XBot are depicted, which are detailed in [37]. The lowest layer implements the *EtherCAT master*, which realises the bi-directional communication between the centralised software components and the decentralised ones implemented at the firmware level

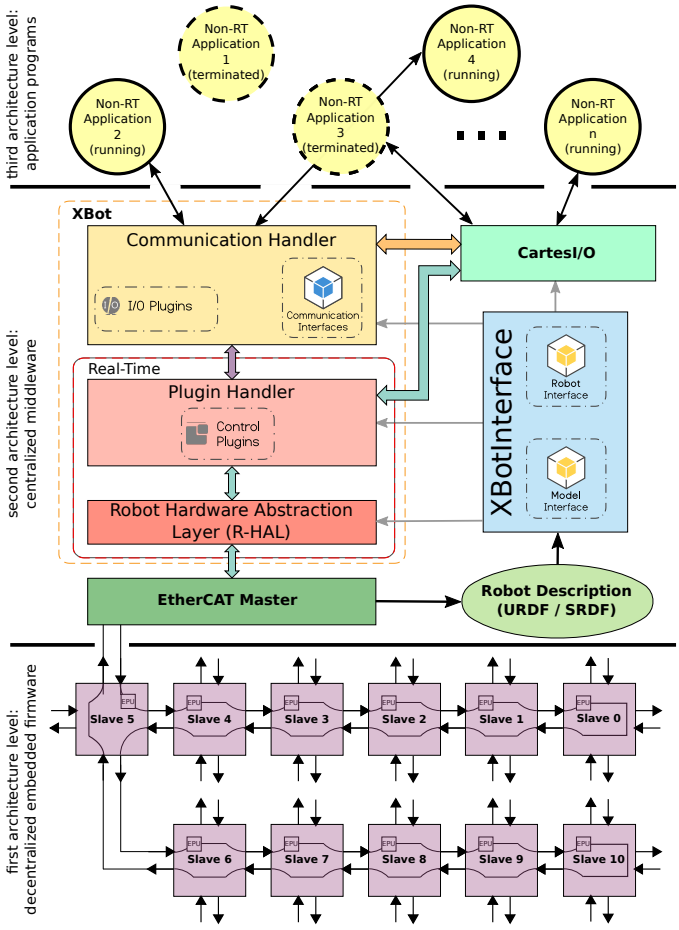

Fig. 7. Software architecture scheme: decentralised firmware level, centralised middleware level and application level.

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 organise the communication to the third non-RT layer that is the application level. The *XBotInterface* provides a 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 development at the application level, the CartesI/O [38] library provides an auto-generated 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 real-time 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 centralised 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 reconfigurability also in the control architecture.

A. Optimisation-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 behaviours.

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

$$\boldsymbol{\Gamma}(\boldsymbol{x}) = ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||^2_{\boldsymbol{W}} \tag{6}$$

defined by the matrix $\boldsymbol{A}$, the column vector $\boldsymbol{b}$, a weighting matrix $\boldsymbol{W}$ and the robot state vector $\boldsymbol{x}$. Tasks can be defined in the joint space, with the purpose to minimise joint state variables (e.g. torques or velocities) or other indexes, or in the Cartesian space, with the goal to impose a specific behaviour to the robot (e.g. controlling the CoM or an end-effector).

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

$$\boldsymbol{l} \leq \boldsymbol{C}\boldsymbol{x} \leq \boldsymbol{u} \tag{7}$$

where the column vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ are lower and upper bounds, and the matrix $\boldsymbol{C}$ is the constraint matrix. Again, each column of the matrix $\boldsymbol{C}$ corresponds to a DOF of the robot. In the same way, each row corresponds to a constraint. Consequently, adding or removing a robot degree of freedom changes the number of columns of $\boldsymbol{C}$. The rows of the matrix $\boldsymbol{C}$ and thereby also the number of elements in the vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ 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 minimising 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 optimisation problem are updated automatically.

C. Impedance Controller Implementation

A Cartesian impedance controller has been selected for demonstration in this paper. 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 $\boldsymbol{F_{ext}} \in \Re^m$ to the deviation of its actual Cartesian position $\boldsymbol{x} \in \Re^m$ from the desired equilibrium point $\boldsymbol{x_d}$, according to the law:

$$\boldsymbol{\Lambda_d}\ddot{\boldsymbol{e}} + \boldsymbol{D_d}\dot{\boldsymbol{e}} + \boldsymbol{K_d}\boldsymbol{e} = \boldsymbol{F_{ext}} \tag{8}$$

where $\boldsymbol{\Lambda_d}, \boldsymbol{D_d}, \boldsymbol{K_d}$ are $m \times m$ matrices representing the desired mass, damping and stiffness, respectively and $\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{x_d}$ 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 can not be chosen arbitrarily, but it is determined by the equivalent Cartesian mass matrix $\boldsymbol{\Lambda(q)}$:

$$\boldsymbol{\Lambda_d} = \boldsymbol{\Lambda(q)} = (\boldsymbol{J(q)M(q)}^{-1}\boldsymbol{J(q)}^T)^{-1} \tag{9}$$

where $\boldsymbol{J(q)}$ is the task Jacobian matrix of size $m \times n$ and $\boldsymbol{M(q)}$ 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:

$$\boldsymbol{M(q)}\ddot{\boldsymbol{q}} + \boldsymbol{n(q,\dot{q})} = \boldsymbol{\tau_d} + \boldsymbol{J(q)}^T\boldsymbol{F_{ext}} \tag{10}$$

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

$$\underset{\boldsymbol{\delta\tau} \in \boldsymbol{C}}{min} ||\boldsymbol{J(q)M(q)}^{-1}\boldsymbol{\delta\tau} - \boldsymbol{J(q)M(q)}^{-1}\boldsymbol{J(q)}^T\boldsymbol{f}||^2 \tag{11}$$

where $\boldsymbol{f} = \boldsymbol{\Lambda(q)}(\ddot{\boldsymbol{x}}_{\boldsymbol{d}} - \dot{\boldsymbol{J}}\boldsymbol{(q)}\dot{\boldsymbol{q}}) - \boldsymbol{D_d}\dot{\boldsymbol{e}} - \boldsymbol{K_d}\boldsymbol{e}$ and $\boldsymbol{C}$ is the set of linear constraints as in (7), that imposes limits on position, velocity, acceleration and torque. Eq. (11) represents the main task in Cartesian space, for which one of the solutions is the well-known $\boldsymbol{\tau}^* = \boldsymbol{J(q)}^T\boldsymbol{f}$. In the case of redundant manipulators, the nullspace dynamics can be determined by adding dynamically-consistent ([41], [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 realise a 7-DOF manipulator).

We can see how (11) is written as in the generic form (6) where $\boldsymbol{A} = \boldsymbol{JM}^{-1}$ is a $m \times n$ matrix, while $\boldsymbol{x} = \boldsymbol{\delta\tau}$ is a $n \times 1$ vector and $\boldsymbol{b} = \boldsymbol{JM}^{-1}\boldsymbol{J}^T\boldsymbol{f} = \boldsymbol{\Lambda}^{-1}\boldsymbol{f}$ is a $m \times 1$ 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 $\boldsymbol{A}$ matrix and the length of the robot state vector $\boldsymbol{x}$, while no change to the controller or re-tuning 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 EMIs of the Base module to attach module chains. The Base module contains a power-board and a compact PC with RT operating system to run the centralised 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 realised by a hollow flange with 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 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 PCB with rubber support

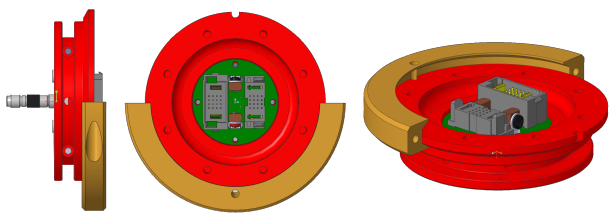

Fig. 8. Side, top and isometric view of the EMI. One of the two C-couplings required to establish a connection and its placement around the conical surface of the EMI, is also shown in the picture. A couple of screws can tighten the two C-couplings to establish the mechanical connection.

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 Fig. 9.1. Following Sec. III, the EtherCat master discovers the apparent network topology as a graph χ with each EtherCAT slave represented as a node 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 four fields characterising it: type, id, size and revision as described in Sec. IV. With the identifier, the information characterising each module is retrieved and aggregated to χ (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 ϕ can be obtained (Fig. 9.4), and represented in URDF format (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 minutes (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, robot is instructed to draw on two sheets of paper. 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 Fig. 10.1. The *automatic discovery* phase 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 Fig. 10.3. The whole procedure repeats for the 5-DOF robot in the *adapt*, *discovery* and *repeat* phases Fig. 10.4-6. After about 13 minutes the entire experiment is completed[2]. For this experiment a non-RT state machine runs at the application level of the software architecture of Sec. V to control the behaviour 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

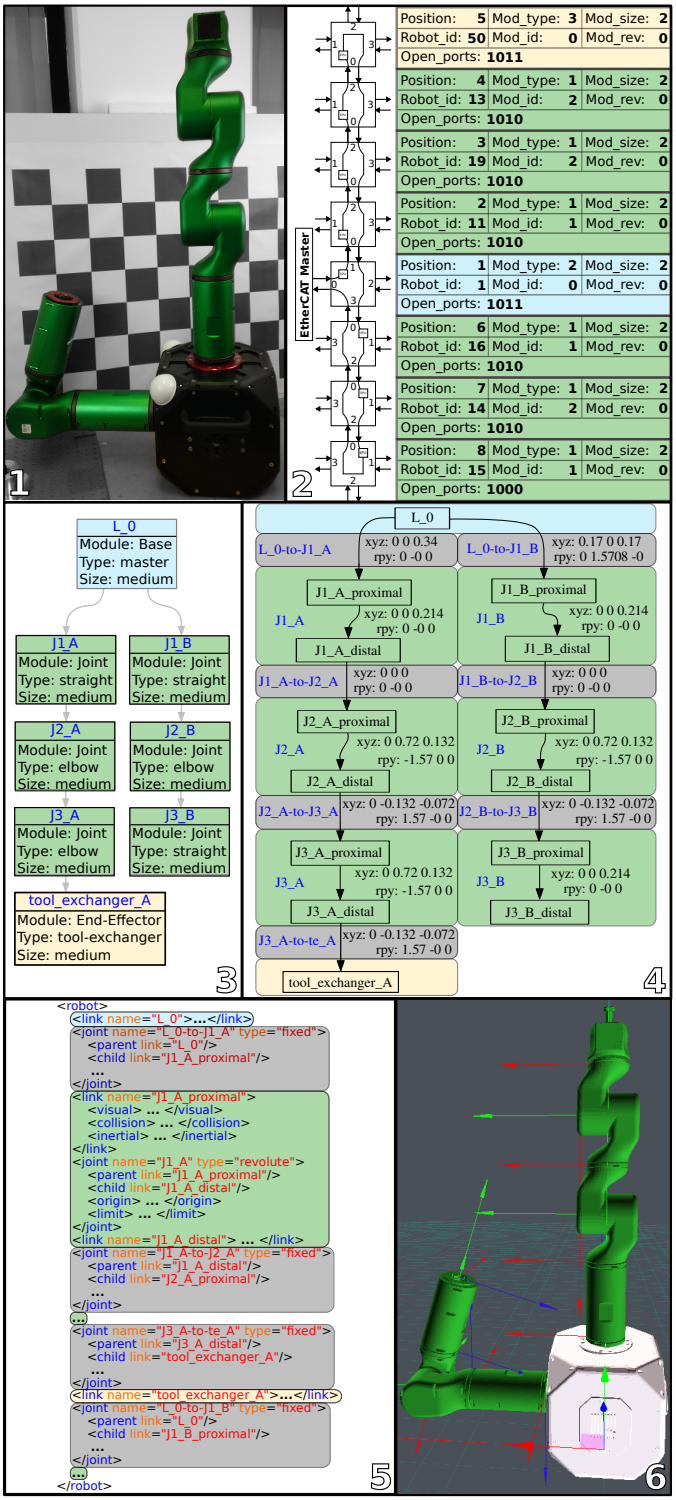

Fig. 9. The topology of a generic robot composed of the basic modules can be recognised with the proposed method. In this picture a two-arm robot model has been built (1) and information on each module collected by the EtherCAT master (2). From this data, the proposed method reconstructs the network topology χ (3) and physical topology ϕ (4). A snapshot of the URDF representation of ϕ is shown in (5), and its virtual rendering in (6), matching the real robot. Different colours indicate different modules and show how the data is propagated through χ to ϕ, and then stored in URDF. Light green, blue and yellow indicate respectively Joint, Base and End-Effector modules. In grey are instead indicated the *fixed joints*, rigid transformations representing the mechanical connections established between each module.

[2]Video of the experiment is available at: https://youtu.be/DHi7aI1HcpE

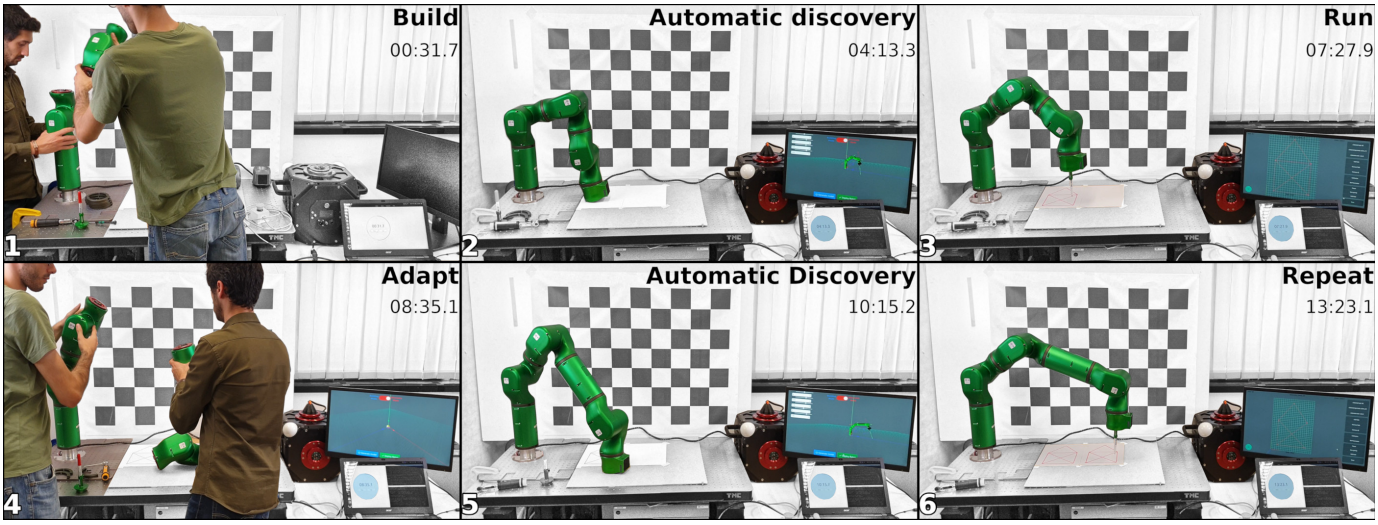

Fig. 10. In this example application, a 4 DOF robot is originally built by connecting the basic modules (1). The robot topology is then reconstructed and visualised by the web app, where the initial robot pose can be set and the controller started (2). Through the GUI it is possible to specify a drawing task and run it (3). This is then repeated for a 5 DOF robot which can be built just by adding a new module to the previously built one (4). The discovery and drawing task are then repeated as before (5-6). The drawing is replicated on a second sheet of paper not reachable by the 4-DOF robot.

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 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 Sec. 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 robot cases. 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 $[N/m]$, rotational stiffness on the roll and pitch rotations of 20 $[Nm/rad]$ with null stiffness on the yaw axis. This proves there is no need to re-tune the controller parameters after re-configuration. 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 behaviour property of the impedance control, with techniques as those proposed in [47], [48].

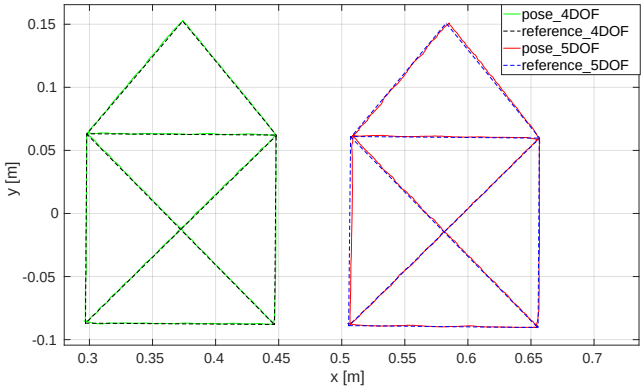

Fig. 11. Actual end-effector pose and its reference on the x-y plane for the drawing task of the 4-DOF and 5-DOF robot.

VIII. Conclusion

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

Using the proposed method, the paper 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 minutes approximately. The time to physically assemble and program the robot for the desired tasks are on equally short time scales. This paves the way

to exploit the versatility of reconfigurable modular robots to: (i) cope with frequent product variation, (ii) minimise maintenance down-times and (iii) build new robot designs on-demand. The proposed approach enables robots not only to assist in the production of mass customisable products, but permits robots to become mass-customisable 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] S. Howes, "Robot Adoption: The SME Challenge - National Report Autumn 2019-20," vol. 10, no. 2, pp. –.

[2] D. Schmitz, P. Khosla, and T. Kanade, "The cmu reconfigurable modular manipulator system," 1988.

[3] R. Cohen, M. Lipton, M. Dai, and B. Benhabib, "Conceptual design of a modular robot," 1992.

[4] T. Matsumaru, "Design and control of the modular robot system: Tomms," in *IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 2125–2131, 1995.

[5] I.-M. Chen and G. Yang, "Automatic model generation for modular reconfigurable robot dynamics," 1998.

[6] I.-M. Chen, "Rapid response manufacturing through a rapidly reconfigurable robotic workcell," *Robot. Comput. Integr. 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, p. 48, 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. Mech.*, vol. 7, no. 4, pp. 431–441, 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. Mech.*, vol. 7, no. 4, pp. 442–451, 2002.

[11] A. Castano, A. Behar, and P. M. Will, "The conro modules for reconfigurable robots," *IEEE/ASME Trans. Mech.*, vol. 7, no. 4, pp. 403–409, 2002.

[12] M. W. Pryor, R. C. Taylor, C. Kapoor, and D. Tesar, "Generalized software components for reconfiguring hyper-redundant manipulators," *IEEE/ASME Trans. Mech.*, vol. 7, no. 4, pp. 475–478, 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. Mech.*, vol. 16, no. 4, pp. 695–706, 2010.

[14] S. Russo, K. Harada, T. Ranzani, L. Manfredi, C. Stefanini, A. Menciassi, and P. Dario, "Design of a robotic module for autonomous exploration and multimode locomotion,"

[15] T. Zhang, W. Zhang, and M. M. Gupta, "An underactuated self-reconfigurable robot and the reconfiguration evolution," *Mechanism and Machine Theory*, vol. 124, pp. 248–258, 2018.

[16] S. Ha, S. Coros, A. Alspach, J. Bern, J. M. Kim, and K. Yamane, "Computational Design of Robotic Devices From High-Level Motion Specifications," *IEEE Trans. Robot.*, vol. 34, no. 5, pp. 1240–1251, 2018.

[17] A. Zhao, J. Xu, M. Konaković-Luković, *et al.*, "Robogrammar: graph grammar for terrain-optimized robot design," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–16, 2020.

[18] "SCHUNK GmbH & Co. KG 2012." https://schunk.com. Accessed: 2020-03-30.

[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," *Proc. 14th IFToMM World Congress*, pp. 403–409, 2015.

[20] E. Icer, A. Giusti, and M. Althoff, "A task-driven algorithm for configuration synthesis of modular robots," *Proc. IEEE Int. Conf. Robot. and Autom.*, pp. 5203–5209, 2016.

[21] R. R. Burridge, M. W. Chu, and B. T. Wolfe, "Apparatus, systems, and methods for reconfigurable robotic manipulator and coupling," Oct. 25 2016. US Patent 9,475,199.

[22] J. Baca, B. Woosley, P. Dasgupta, and C. Nelson, "Real-time distributed configuration discovery of modular self-reconfigurable robots," *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 1919–1924, 2015.

[23] V. Mayoral Vilches, "Metodo de determinacion de configuracion de un robot modular," 03 2016. ES patent 2661067B1.

[24] A. Giusti and M. Althoff, "On-the-Fly Control Design of Modular Robot Manipulators," *IEEE Trans. Contr. Systr Tech.*, pp. 1–8, 2017.

[25] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, "Effortless creation of safe robots from modules through self-programming and self-verification," *Science Robotics*, vol. 4, p. eaaw1924, June 2019.

[26] Z. Bi, Y. Lin, and W. Zhang, "The general architecture of adaptive robotic systems for manufacturing applications," *Robot. Comput. Integr. Manuf.*, vol. 26, no. 5, pp. 461–470, 2010.

[27] I.-M. Chen, *Theory and applications of modular reconfigurable robotic systems*. PhD thesis, California Institute of Technology, 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, 2020.

[29] "Open EtherCAT Society." https://openethercatsociety.github.io/. Accessed: 2020-03-30.

[30] "Universal Robot Description Format (URDF)." https://wiki.ros.org/urdf. Accessed: 2020-03-30.

[31] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2008.

[32] T. Zhang, K. Backstrom, R. Prince, C. Liu, Z. Qian, D. Zhang, and W. Zhang, "Robotic dynamic sculpture: Architecture, modeling, and implementation of dynamic sculpture," *IEEE Robot. Autom. Mag.*, vol. 21, no. 3, pp. 96–104, 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.*, 2014.

[34] "Orocos Kinematics Dynamics Library." https://www.orocos.org/kdl. Accessed: 2020-03-30.

[35] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," *Proc. IEEE/SICE Int. Symp. Syst. Integr., SII 2019*, pp. 614–619, 2019.

[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, 2020.

[38] A. Laurenzi, E. M. Hoffman, L. Muratore, and N. Tsagarakis, "Cartesi/o: A ros based real-time capable cartesian control framework," *Proc. IEEE Int. Conf. Robot Autom.*, pp. 591–596, 2019.

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

[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," *Proc. IEEE Int. Conf. Adv. Robot.*, pp. 1–6, 2009.

[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," *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 3, pp. 3704–3709, 2003.

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

[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, 1987.

[44] N. Kashiri, L. Baccelliere, L. Muratore, A. Laurenzi, Z. Ren, E. M. Hoffman, M. Kamedula, G. F. Rigano, J. Malzahn, S. Cordasco, *et al.*, "Centauro: A hybrid locomotion and high power resilient manipulation platform," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1595–1602, 2019.

[45] "Alberobotics." https://alberobotics.it/. Accessed: 2020-03-30.

[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. and Autom.*, pp. 5210–5215, IEEE, 2016.

[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. Cont. Syst. Tech.*, vol. 26, no. 6, pp. 2259–2267, 2017.

[48] J. Kim and E. A. Croft, "Full-state tracking control for flexible joint robots with singular perturbation techniques," *IEEE Trans. Contr. Syst. Tech.*, vol. 27, no. 1, pp. 63–73, 2017.

Edoardo Romiti received the Master's degree in automation engineering (Hons.) from the University of Bologna, Bologna, Italy, in March 2017.

He then joined the Humanoids and Human-Centred Mechatronics lab, Istituto Italiano di Tecnologia, Genoa, Italy where is currently working toward a Ph.D. degree in Robotics.

Jörn Malzahn graduated in electrical engineering from TU Dortmund University as the best of the year in 2008. In 2014 he received the doctorate in engineering (Dr.-Ing.) with summa cum laude from the same University. He joined the Humanoids and Human-Centred Mechatronics lab in 2015 focusing on the actuation and control of compliant robots.

Navvab Kashiri completed his Ph.D. study at the Humanoid and Human Centred Mechatronics group of IIT in 2015, where he is currently working as a senior postdoctoral researcher, focusing on the design, development and control of mechatronics systems including robotic modules, compliant actuators and sensors, and variable impedance devices.

Francesco Iacobelli received the Master's degree in mechanical engineering (Hons.) from Sapienza, University of Rome (2016).

He worked as embedded software engineer for two years. Since 2019, he joined the IIT HHCM lab (Genoa, Italy) as a support technician, working mainly on control software architecture.

Marco Ruzzon received the Master's degree in energy engineering (Hons.) from Politecinco di Milano, Milano, Italy, in April 2017, and the double Master's Degree in robotics engineering from University of Genova, Genova, Italy and Ecole Centrale de Nantes, Nantes, France, in August 2019. He then joined the Humanoids and Human-Centred Mechatronics lab, IIT, Genoa, Italy.

Arturo Laurenzi received his Master's Degree in Automation Engineering (cum laude) in 2015 from Università Degli Studi di Firenze. He then started a PhD study inside the Humanoid and Human-Centered Mechatronics Lab at IIT, on the software architecture and control of the hybrid wheeled-legged quadrupedal robot Centauro. Since 2019, he is Senior Technician in the same lab.

Enrico Mingo Hoffman is a Researcher in the HHCM Lab. at Istituto Italiano di Tecnologia, since 2020. He received the B.Sc. in Electronics Eng. and M.Sc. in AI and Robotics both at University of Rome "La Sapienza", in 2008 and 2012 respectively. He got a Ph.D. in humanoid robotics at University of Genoa in 2016. His research interests include robot motion planning and control

Luca Muratore is the Lead Software Engineer at Istituto Italiano di Tecnologia (IIT), in the Humanoids and Human Centered Mechatronics research line. In 2020 He received his split-site PhD degree in Electrical and Electronic Engineering from the University of Manchester and IIT. He is the coordinator of the ROS End-Effector H2020 ROSIN FTP.

Alessio Margan received the Master's degree in Computer Science from the University of Genoa, Genoa, Italy, in December 2000.

He then joined the ADVR lab in 2009 and later Humanoids and Human-Centred Mechatronics lab, Istituto Italiano di Tecnologia, Genoa, Italy where is currently working

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

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

Stefano Cordasco graduated in Physics from University of Genova in 2003.

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

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

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