Journal of Intelligent & Robotic Systems (2023) 108:70
https://doi.org/10.1007/s10846-023-01911-5

REGULAR PAPER

ROS End-Effector: A Hardware-Agnostic Software and Control Framework for Robotic End-Effectors

Davide Torielli[1,2] · Liana Bertoni[1,3] · Fabio Fusaro[4,5] · Nikos Tsagarakis[1] · Luca Muratore[1]

Received: 25 August 2022 / Accepted: 7 June 2023 / Published online: 22 July 2023

Abstract
In recent years, several robotic end-effectors have been developed and made available in the market. Nevertheless, their adoption in industrial context is still limited due to a burdensome integration, which strongly relies on customized software modules specific for each end-effector. Indeed, to enable the functionalities of these end-effectors, dedicated interfaces must be developed to consider the different end-effector characteristics, like finger kinematics, actuation systems, and communication protocols. To face the challenges described above, we present ROS End-Effector, an open-source framework capable of accommodating a wide range of robotic end-effectors of different grasping capabilities (grasping, pinching, or independent finger dexterity) and hardware characteristics. The ROS End-Effector framework, rather than controlling each end-effector in a different and customized way, allows to mask the physical hardware differences and permits to control the end-effector using a set of high-level grasping primitives automatically extracted. By leveraging on hardware agnostic software modules including hardware abstraction layer (HAL), application programming interfaces (APIs), simulation tools and graphical user interfaces (GUIs), ROS End-Effector effectively facilitates the integration of diverse end-effector devices. The proposed framework capabilities in supporting different robotics end-effectors are demonstrated in both simulated and real hardware experiments using a variety of end-effectors with diverse characteristics, ranging from under-actuated grippers to anthropomorphic robotic hands. Finally, from the user perspective, the manuscript provides a set of examples about the use of the framework showing its flexibility in integrating a new end-effector module.

Keywords End-effector control · Hardware abstraction · Grasping primitives · Robotics software architecture · Robot operating system (ROS)

1 Introduction

Even considering the research's effort made, it is still difficult to see an effective deployment of complex robotic hands in real use-case scenarios [1], where usually only simple 2-pad and single-motor grippers are adopted. In fact, there is a significant barrier that prevents a wider use of more dexterous end-effectors, due to the lack of frameworks and user interfaces that permit to abstract the end-effector characteristics enabling a transparent integration of these more capable end-effectors in industrial lines.

Motivated by the limited capabilities of existing end-effector software tools, we have developed ROS End-Effector, an open-source software framework to facilitate the control and integration of new and diverse robotic end-effectors, thanks to an automatic extraction of grasping skills and to the exploitation of hardware-agnostic software modules. To synthesize the grasp poses at a higher-level, ROS End-Effector leverages on the concept of *primitive grasping actions*, inspired by the relevant works in the fields of synergies and manipulation primitives. The main contributions of the ROS End-Effector software framework are summarized below:

- Automatic extraction of end-effector capabilities through an identification of *primitive grasping actions* from the end-effector model, e.g. fingers configuration, their possible motions and their interactions. The extracted primitives permit to control the end-effector in a dimensionality-reduced subspace, following the concept of the synergies.
- Accommodation of a wide range of end-effectors with different hardware characteristics and grasping capabil-

✉ Davide Torielli
davide.torielli@iit.it

Extended author information available on the last page of the article

ities, effectively facilitating their integration through an Hardware Abstraction Layer (HAL).
- Set of software tools like Application Programming Interfaces (APIs), simulation tools, and Graphical User Interface (GUI), to effectively exploit the grasping actions extracted and syntetized to perform manipulation tasks.

Thanks to the above functionalities, the proposed software framework is able to mask the end-effector physical hardware differences (e.g. kinematic and dynamic model, number of actuators, number of fingers, finger transmission principles, and communication protocols) permitting to seamlessly command the end-effector using a set of high level primitive grasping actions exploiting the Robot Operating System (ROS) middleware.

The automatic extraction of primitive grasping action, synthesis of complex actions and their execution in simulated environments has been validated with end-effectors with different kinematics and capabilities: SCHUNK SVH [2], Robotiq 2F-140 [3], Robotiq 2F-85 [3], Robotiq 3F [4], Dagana, qb SoftHand [5], OnRobot 3FG15 [6], and HERI II [7]. Furthermore, the framework has been validated in real environment with some of the above-mentioned end-effectors (Dagana, Robotiq 2F-85, OnRobot 3FG15, qb SoftHand, and HERI II). Finally, the ROS End-Effector package has been released in the ROS official repositories, making it available for both ROS and ROS2.

1.1 Manuscript Overview

This manuscript is an enhanced version of our previous work [8]. In particular, we have expanded the methods sections (Sections 4, 5 and 6), providing more details on the framework. New figures and code snippets have been added to help the reader in better understanding the concepts and the software tools, like the ROS End-Effector Graphical User Interface (Section 6.1). Furthermore, we have made the validation richer by integrating new and diverse robotic end-effectors and by performing more manipulation tasks in real scenarios (Fig. 12, Fig. 13). Finally, we have added a complete new Section 8 providing a tutorial with a lot of practical explanations and code examples.

In this work, we do not consider the aspect of the grasp planning. Hence, we do not deal with the object geometries and the fingers movements necessary to synthesize a stable grasp. This aspect has been taken into account in a parallel work [9], where we have developed a new generic grasp planner using the concept of primitive grasping action for the *grasp synthesis*. This permits to automatically define a suitable composition of primitive grasping actions able to grasp an arbitrary object.

The paper is structured as follows. Section 2 recaps the relevant works in the field; Section 3 introduces the ROS End-Effector framework; Section 4 explains the theory behind the *primitive grasping actions* and their automatic extraction from the end-effector model; Section 5 describes how to generate more complex end-effector motions from the primitive grasping actions; Section 6 details how the grasping actions are requested to the end-effector; Section 7 illustrates the experiments conducted with various end-effectors; Section 8 gives a short practical tutorial about the ROS End-Effector framework; Finally, in Section 9, conclusions and future works are discussed.

2 Literature Review

The research developments of robotic end-effectors is endorsed by the necessity to improve how robots manipulate objects and interact with their unstructured surroundings. This need comes from heterogeneous scenarios, from industrial to healthcare [10]. As a result, a lot of effort has been spent in realizing end-effectors with various capabilities, from simple grippers to complex hands with human-like dexterity and strength, leading to the exploration of new grasping principles.

Many works in the robotic field are inspired by neuroscientific studies on human hands and their grasping abilities [11, 12]. These studies show that humans simplify the grasping of an object by controlling their hands in a smaller subspace of the space where normally the human degrees of freedom (DoF) lie. From these observations, the concept of *synergies* and its exploitation in robotics emerged. In the literature, the definition of synergy can be intended as the "common patterns of actuation of the human hand" [13], and as a map between the higher-dimensional complexity of the mechanical architecture of the human hand and the lower-dimensional control space of the action and performance [14]. Thanks to the dimensionality reduction of the controlling space, the grasp planning is more efficient and more adaptable to various kinds of end-effectors [15]. This can lead to the development of general frameworks to control robotic hands of different kinematic design. Indeed, control algorithms for anthropomorphic hand with predefined synergies can be developed and the resulting finger motions can be mapped onto various robotic hands. This mapping has been explored at the Cartesian space level [16] and at the joint space level [17]. Another approach introduces the concept of virtual objects, for a mapping in the "object domain" [18, 19]. The idea is to study a virtual object held by a human hand and to transform its parameters to create a virtual object held by a robotic hand. Nevertheless, using a synergy-based approach does not automatically imply that the end-effector will be compliant with respect to the object to grasp. To cope with this issue, the concept of *soft-synergies* has been intro-

duced. By exploiting the soft-synergies, the hand modifies its final pose according to the object shape [20, 21]. Even if the soft-synergy is an elegant theoretical solution, in practice, implementing a combination of natural synergies and compliance is very challenging. For this reason, *adaptive-synergies* have been employed for the design parameters of an under-actuated hand (that will become the qb SoftHand) [5].

Alongside the synergies, the concepts of *grasping/manipulation primitives* has emerged. Various studies give slightly different definitions of grasping/manipulation primitives, but, in general, a primitive is an end-effector capability that is exploited as an atomic unit to compose more complex grasping actions. The primitive is then employed by high-level grasping and manipulation interfaces to abstract the low-level end-effector capabilities and hardware details. Since their importance, grasping actions have been employed in different ways. In a pioneer work a *Manipulation Task Primitive* is classified as the relative motion between two (rigid) parts. By classifying the primitives, a library of robot capabilities in the manipulation domain is built, thus providing a higher level of abstraction for more complex manipulation tasks [22]. Based on *Manipulation Primitives*, a generic framework for sensor-based robot motion control has been developed. An adaptive selection matrix switches between multiple sensors and open/closed-loop controllers. The control signal are chosen based on the Manipulation Primitives, which hence constitute the interface from high-level applications to the low-level controller of robotic manipulators [23]. Grasping capabilities has also been designed as *Control Primitives*, which constitute an abstract layer built as a vocabulary, which can be coordinated using state machines to describe complex actions. The state machines are then automatically translated to specific models, such that the full capabilities of each robotic platform can be exploited [24].

So far, the cited works aim to analyse methodologies to extract the fundamental motions (embedded in the *synergies* or in the *primitives*) of an end-effector in such a way to simplify the planning of the grasping task. In parallel, some other works focus on the development of generic software tools for motion analysis, planning, and control of different kinds of end-effectors by applying the concepts of synergy and primitive. *GraspIt!* [25] is a simulator for grasping research that can accommodate arbitrary hands and objects. The implemented collision detection and contact determination system permits to create a set of grasps for a specific object. Each grasp is then evaluated with a set of grasp quality metrics as well as visualization methods which allows the user to understand the weak points of the grasp. *GraspIt!* focuses on grasp planning and analysis, but it does not consider dexterous manipulation, and it does not provide soft contact models or sensors simulation support. *Syngrasp* [26] is a MATLAB toolbox focused on grasping analysis. It features several functions to investigate the grasp properties: 1) controllable forces and object displacement, 2) manipulability analysis, and 3) grasp stiffness and quality measures. This toolbox is intended to be used by applications that range from neuroscience to the design of robotic hands. Another software tool is *OpenGRASP: A Framework for Robot Grasping Simulation* [27]. Its aim is, like *GraspIt!*, to provide a planner, an analyser and a simulator for grasping with robotic hands. It includes the modeling of actuators, sensors and contact and it permits to choose among a set of available physics engines for the simulation. The same authors developed *OpenHand: A Framework for Human Grasping Simulation* [28], a simulation engine to analyse object grasping with human hand, modelled as similar as possible to real human hands, with its skeleton structure, muscles, tendons, skin and neuromuscular control.

The aim of the above-mentioned software is to examine the possible interactions between the end-effector and the object to grasp. They are not designed with the main intent of communicating with the real hardware, i.e. commanding the hand actuators and receiving feedback from sensors. To answer to these problems there exist more hardware-orientend software frameworks. *MoveIt* [29] is a well-known motion planner platform for robotic manipulators, fully integrated in *ROS* (Robotic Operating System) [30, 31]. *MoveIt* provides a set of plugins to enhance its functionalities. One of them, *MoveIt Grasps* [32], consists in a basic grasp generator to grasp simple objects (only blocks or cylinders) with parallel or suction grippers. Another *MoveIt* based tool, the *ROS2 Grasp Library* [33], provides a grasp planner for industrial purposes, focusing on deep learning grasp detection algorithms for intelligent visual grasp solutions. The provided grasping functionality of the last two mentioned ROS tools is limited to pick and place operations. They do not support complex multi-fingered hands, and their focus is mainly the planning of a grasp, without including features to ease the integration of a new end-effector.

3 Software Framework

The ROS End-Effector framework is composed by two main components, as schematized in Fig. 1:

- With the *offline* component, the end-effector motion capabilities are automatically extracted resulting in a set of primitive grasping actions. In brief, based on a particular end-effector mechanical configuration (e.g. number of finger, degrees of actuation of each finger, joint limits, and so on), our framework provides the ready-to-use primitive grasping actions. Furthermore, more complex

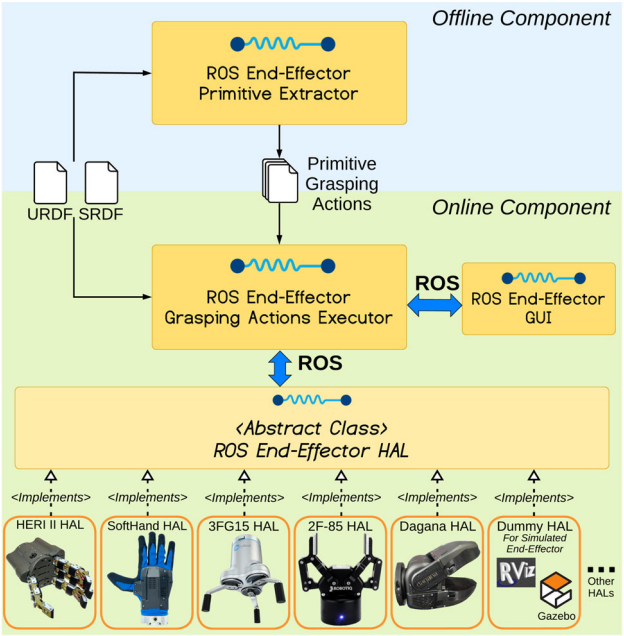

Fig. 1 Overall scheme of ROS End-Effector framework, with its principal software modules divided in *offline* and *online* components

custom grasping actions can be built from the extracted primitives.
- With the *online* component, the user can execute manipulation tasks by requesting the grasping action commands (instead of directly commanding the end-effector actuators). Moreover, a GUI, dynamically loaded for the end-effector in use, is provided, while the communication with the robot is performed by the implemented HAL.

In the following sections, we will describe these components in detail.

4 Offline Component: Primitive Grasping Actions

A *primitive grasping action* is a particular collection of fingers movements inducted by the actuators as a mean to grasp an object. Therefore, the end-effector fundamental capabilities are embedded in its primitive grasping actions. This permits to execute manipulation tasks by just commanding these semantically significant grasping actions (like "pinch"), instead of considering at the low-level which actuator must be activated to control particular fingers. Indeed, the relationship between actuators and end-effector fingers movements is hidden to the user because it is taken into account automatically by the framework.

We divide the primitive grasping actions into three main categories:

- Trig-type primitive grasping actions are dedicated to move a single finger or a phalanx. Even though moving a single finger is not enough to grasp an object, these movements are important as components to perform more complex grasping actions, as we will see later in Section 5. This category includes: *Trig*, *TipFlex*, and *FingFlex* primitive grasping actions.
- Pinch-type primitive grasping actions are suitable for precise grasps. With them, the fingertips move towards each other to pick narrow or small objects. It is important to note that a pinch-type primitive can not be decomposed into *Trigs*, because it is not always true that two or more separate *Trigs* can establish contacts between fingertips or make them move towards each other. This category includes: *PinchTight*, *PinchLoose*, and *MultiPinchTight_N* primitive grasping actions.
- *SingleJointMultipleTips_N* primitive grasping action has been included to synthesize finger motions that do not fit in other primitives. This primitive takes care of the characteristic of some end-effectors (like the qb SoftHand and the SCHUNK SVH hand) of having an actuator coupled with more fingers.

As an example, some of the primitive grasping actions performed by the SCHUNK SVH hand from each of the listed categories are shown in Fig. 2.

Each primitive grasping action contains the following data:

- The primitive grasping action name.
- The end-effector's bodies involved in the primitive grasping action, like the fingers names or the actuator names. They are used to discriminate the primitive grasping actions with the same name (e.g. a *trig* can be performed with different fingers, like the *index* or the *middle*).
- The position set-points of the actuators, i.e. the reference positions which are requested to the actuators when the grasping action is demanded.

4.1 Extraction of the Primitive Grasping Actions

The *ROS End-Effector Primitive Extractor* node depicted in Fig. 1 is in charge of exploring the robot model and automatically extracting the primitive grasping actions. Only two configuration files representing the robot model are necessary in this phase: the well-known Unified Robot Description Format (*URDF*) file and the Semantic Robot Description Format (*SRDF*) file. More information about these formats and their relationship with ROS End-Effector are given in Section 8.1. After providing the robot model to the ROS End-Effector node, the primitive grasping actions available for the specific

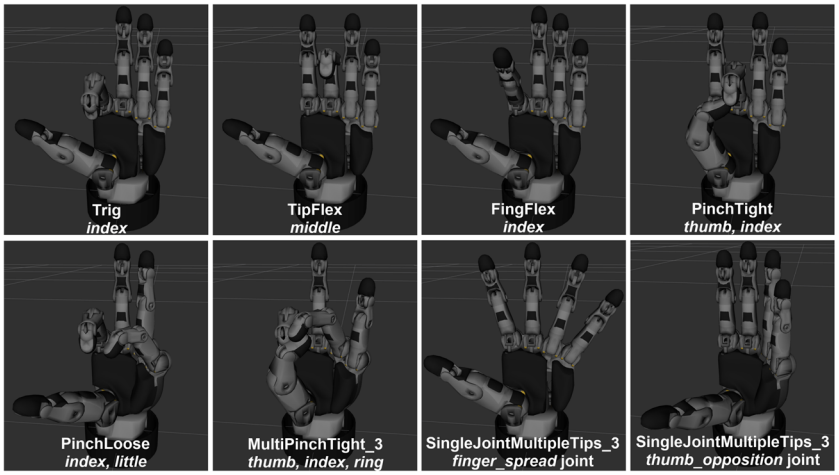

Fig. 2 Some primitive grasping actions extracted from SCHUNK SVH. Please note that the two *SingleJointMultipleTips_3* primitives in the bottom row have the "3" suffix because the *finger_spread joint* and the *thumb_opposition joint* actuators are coupled to three fingers, respectively: *index*, *ring*, *little* and *thumb*, *ring*, and *little*

end-effector in use are extracted with a dedicated algorithm for each primitive.

A *Trig* done with a finger is possible if there is at least one actuator dedicated to move only that finger. In this case, this/these actuator/s coupled to the finger is/are taken into consideration for the *Trig* primitive.

If a single finger is moved by more than one actuator, usually it means that some of its phalanges can be moved independently. Hence, the end-effector has additional motion capabilities that we embed in the *TipFlex* and *FingFlex* primitives. The *TipFlex* primitive considers the actuator which moves the farther part of the finger from the palm. This actuator usually makes the fingertip bend, hence the name *TipFlex*. The *FingFlex* primitive, instead, considers the first actuator of the finger, beginning from the palm. This actuator usually flexes the finger, thus maintaining the finger straight. For both of them we say "usually", because it is not always true that the first and the last actuators rotate the phalanges along an axis which induces a bending toward the palm. For example, there could exist an actuator that rotates a finger along an axis perpendicular to the palm, like in a scissor movement. In this case these primitive names are misleading, but they remain useful as building blocks for more complex actions, as we will see later.

To automatically extract these three trig-type primitives from the *URDF* and *SRDF* models, the *ROS End-Effector Primitive Extractor* node explores the fingers kinematic chains, and, for each actuator, i.e. a *URDF joint* which is not *mimic* nor *passive* (Section 8.1), we extract the linked phalanges, i.e. *URDF links*. For each finger or phalange that moves independently we have a trig-type primitive. The position set-points of the actuators stored in the primitive are the maximum excursion limit from the starting position for the actuator(s) that move the finger or phalange, and the starting position for the not-involved actuator(s).

For the pinch-type primitives a collision checking is performed, thanks to the *MoveIt collision checker* [29] which is based on Flexible Collision Library (FCL) [34]. A *PinchTight* is found when two end-effector fingertips can perform a movement that causes them to collide. The reason is that, if two fingertips can collide, they can *pinch* very little or narrow objects. To find this primitive grasping action, we look at *M* random hand's configuration: for each couple of fingertips, we choose the configuration where their meshes compenetrate more, because more co-penetration will mean more maximum force applicable to the object, hence resulting in a more stable pinch. If, for a certain couple of fingertips, no collision is found among the *M* random hand's configuration, we say that two fingers can not perform a *PinchTight*. The pseudocode in Code 1 schematizes these steps.

Code 1: Pseudocode to extract the *PinchTight* primitives

```
1 while i < M
2   pose = hand.setRandomPose()
3   //cfp : colliding fingertip pairs
4   cfp.update(moveit.checkCollisions(pose))

6   foreach p in cfp
7     if p.compenetration > p.old_compenetration
8       cfp.storeNewPinchTight(pose)
```

Some fingertips can move toward each other but without any contact between them, because of some structural hand constraints (like joint limits). Nevertheless, this kind of finger motion can be useful to grasp some not-so-little objects. Hence, we have defined the *PinchLoose* primitive. A *PinchLoose* is searched for all the fingertips pairs that can not perform a *PinchTight*, which is more versatile because the fingertips have more excursion towards each other.

The procedure to extract this primitive is similar as the one for *PinchTight*, but instead of using the mesh compenetration as the measure, we consider the minimum distance to which the two fingertips can arrive before the physical limit. To understand if the fingertips effectively move towards each other (i.e., their trajectories are not parallel) we consider

Table 1 Table which recaps the characteristics of the primitive grasping actions

Name	End-Effector's Bodies Involved	Description	Constraints
Trig	A finger	Move all fingers actuators toward their bounds	At least one actuator providing decoupled motion only to the finger
TipFlex	A finger	Move the last actuator of the finger toward its bound	At least two actuators providing decoupled motion only to the finger
FingFlex	A finger	Move the first actuator of the finger toward its bound	At least two actuators providing decoupled motion only to the finger
PinchTight	Two fingers	Contact between two fingertips	
PinchLoose	Two fingers	Movement of two fingertips towards each other but without contact	
MultiPinchTight_N	N(≥3) fingers	Contact between three or more fingertips	
SingleJointMultipleTips_N	An actuator	Move an actuator (that influences N(≥2) fingertips) toward its bound	The actuator provides coupled motion to N(≥2) fingertips

a robot model where the joints have more excursions (i.e. increased joint limits), and we see if some contact happens among the *M* random configuration. If neither in this case a collision happen, the fingertips trajectories are said to be parallel, hence the two fingers can not perform any pinch motions.

With a *MultiplePinchTight_N* primitive grasping action, N(≥3) fingertips collide, providing a more stable pinch than a *PinchTight* in some situation. The extraction method is similar to the one used for the *PinchTight* primitive.

For the pinch-type primitives, the position set-goals of the actuator stored in the primitive are the ones that make the hand to reach the found configuration, excluding all the actuators not coupled with the considered fingers.

SingleJointMultipleTips_N primitive grasping actions are extracted exploring the kinematic tree given by the *URDF* and *SRDF* files, in a analogous manner as the trig-type primitives.

In Table 1, we sum up the characteristics of each primitive grasping action.

To understand better the automatic extraction phase, the reader can follow the Fig. 3. In this example, we have modified the HERI II hand model to show how from various fingers and actuators configurations different primitive grasping action are extracted. On the left part of the figure, four different configurations are shown with a the top view of the hand. Each colored box represents the presence of a finger while each curved arrow an actuator. The arrow direction symbolizes in which direction the actuator moves the finger(s), assuming that an actuator can move the finger(s) in a unique direction, in both senses. On the right part of the figure, some primitive grasping actions extracted from each configuration are shown (for simplicity, *TipFlex*, *FingFlex*, and *SingleJointMultipleTips_N* primitives are not shown). In the first configuration, a single actuator is coupled to both fingers (as in common industrial 2-finger grippers). So, the only extracted primitive is a *PinchTight* (because we see that the fingers can collide, otherwise we would have only a *PinchLoose*). In the second configuration, we have a slightly more complex end-effector, indeed, it has an actuator

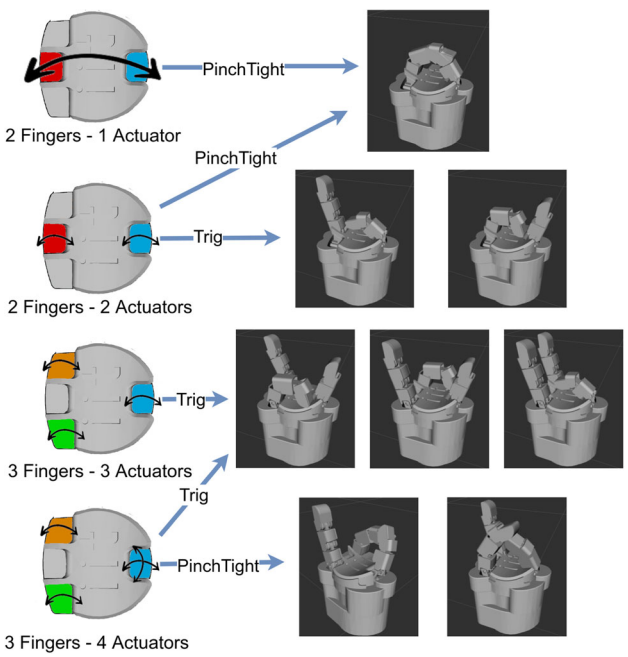

Fig. 3 Schematic end-effector configurations and the primitive grasping actions extracted (excluding the *SingleJointMultipleTips*, *TipFlex*, and *FingFlex* for simplicity). On the left, each finger is represented by a colored box, while each actuator by a curved black arrow. The arrow direction represents the direction in which the actuator move the finger(s). On the right, the automatically extracted primitive grasping actions are shown

for each finger. So, in this case, other than the *PinchTight*, also two *Trig* primitives are available. In the third configuration, the presence of a third finger permits to have three *Trig* primitives, but not any *PinchTight* anymore because fingertips does not ever collide. There are not any *PinchLoose* neither, because the fingers have all parallel trajectories. In the last configuration, we have an additional actuator for the opposing finger (in blue) which can generate finger motion in another direction. In this case, the fingertips may come in contact and so two more *PinchTight* primitives are available.

5 Offline Component: Custom Grasping Actions

With the primitive grasping actions of an end-effector, some simple objects can be grasped thanks to the fundamental motions extracted and embedded in the primitives. For example, a *PinchTight*, if available, can be performed to grasp a little object. Nevertheless, these primitives can be not enough to exploit the full potential of complex multi-DoF end-effectors, like the SCHUNK SVH hand, where complex combinations of finger motions can grasp complex shapes. For this reason, ROS End-Effector facilitates the creation of complex motions from the primitive grasping actions extracted, by defining three kinds of custom grasping actions: *composed grasping action*, *timed grasping action* and *generic grasping action*.

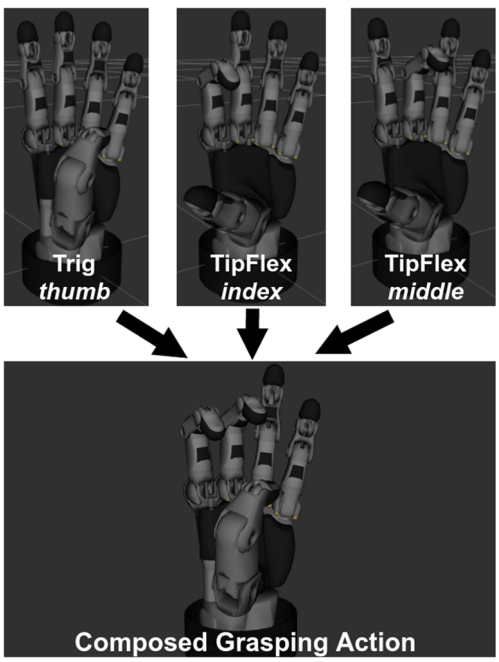

Fig. 4 A composed grasping action generated from three different primitive grasping actions

Composed grasping actions, as the name suggests, are created by *combining* primitives or other custom grasping actions, resulting in a more complex fingers motions. The actuator position set-points of the composed grasping action will be the composition of the position set-points of the inner grasping actions. For example, in Fig. 4, a composed grasping action is defined composing three primitives: *Trig* performed with the thumb, *TipFlex* performed with the index, and another *TipFlex* performed with the middle. Since the SCHUNK SVH hand is an end-effector with a high number of DoFs, it is useful to exploit this feature to adapt the fingers to the shape of a particular object to grasp by bending specific fingers parts.

Timed grasping actions are a collection of grasping actions executed in sequence, with an optional time delay between two consecutive ones. The reason of the introduction of the timed grasping actions is that, for some manipulation tasks, a composed grasping action can be not the best option. Indeed, there are situations in which the fingers must be moved in a particular sequence to reach a pre-grasp pose before effectively close the fingers. Alternatively, some in-hand manipulation tasks can be performed after a successful grasp. For example, in Fig. 5, a timed grasping action is used to perform a in-hand manipulation task with the HERI II

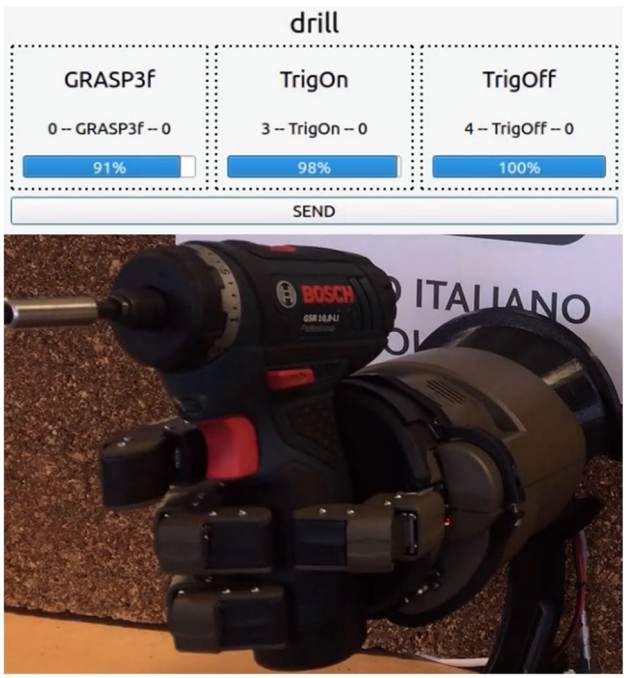

Fig. 5 An example of timed grasping actions, "drill", where the HERI II hand has executed an in-hand manipulation task. The first grasping action of the sequence ("GRASP3f") is used to grasp the drill, and the other two ("TrigOn" and "TrigOff") to press and release the trigger button of the drill. The first two loading bars are not fully completed (to 100%) because of small final errors in the actuator positions respect to the references

hand: first the drill is grasped with the "GRASP3f" action, then the drill is switched on and off with the "TrigOn" and "TrigOff" grasping actions. Please note that this experiment is part of our previous paper [8].

If the composition is not sufficient to exploit a particular end-effector capability, generic grasping actions can be created from scratch, hence not built starting from other grasping actions, but setting directly the actuator position set-points, as explained later in Section 8.3 with Code 8.

ROS End-Effector provides C++ API methods and ROS Services to facilitate the creation of these custom grasping actions. We give examples about the exploitation of this feature in the tutorial Section 8.2.

Primitive and custom grasping actions are stored in *YAML* files, where the information about the grasping actions are written, like the name, the elements involved, and the position set-points of the actuators. This permits to create them once per hand (during the offline phase) and to execute them later (in the online phase) re-utilizing the stored data.

6 Online Component: Demanding the Grasping Actions

The online component of ROS End-Effector permits to command the grasping actions defined offline for the specific end-effector in use (Sections 4 and 5). This component is in charge of recognizing the grasping action requested and sending to the robot the commands necessary to execute the requested grasping action. This is made possible thanks to the *Grasping Action Executor* node and to the HAL, represented in Fig. 1.

The grasping actions executor performs an initialization step where the robot model, described by the *URDF* and *SRDF* files, is parsed, together with all the grasping actions (primitive and custom), described by the *YAML* files. After this step, through the provided ROS topics, services and actions, information about all the available grasping actions can be retrieved and the commands to execute them can be sent.

When commanding a grasping action, fundamentally two information are necessary to be sent to the ROS End-Effector framework: the name of the action and a $0\% - 100\%$ intensity value. The intensity value is used to scale all the position set-points of the actuators before sending the command to the robot. In this way we can perform partial movements of the fingers, for example to grasp a big object without generating too much forces on the object.

A specific primitive grasping action can be usually performed with different fingers. For example, a *Trig* with an human-like end-effector, like the SCHUNK SVH hand, can be performed with any of the five fingers; or a little object can be pinched with different fingers pairs (with the primitives *PinchTight* and *PinchLoose*) or groups (with the primitive *MultiPinchTight_N*). Hence, to command a primitive grasping action, a third information is necessary: the end-effector's body/bodies involved in the primitive. For *Trig*, *TipFlex*, and *FingFlex*, it must be specified the finger's name; for *PinchTight*, *PinchLoose*, and *MultiPinchTight_N*, the names of the fingers that moves toward each other; for *SingleJointMultipleTips_N* the name of the actuator which moves the

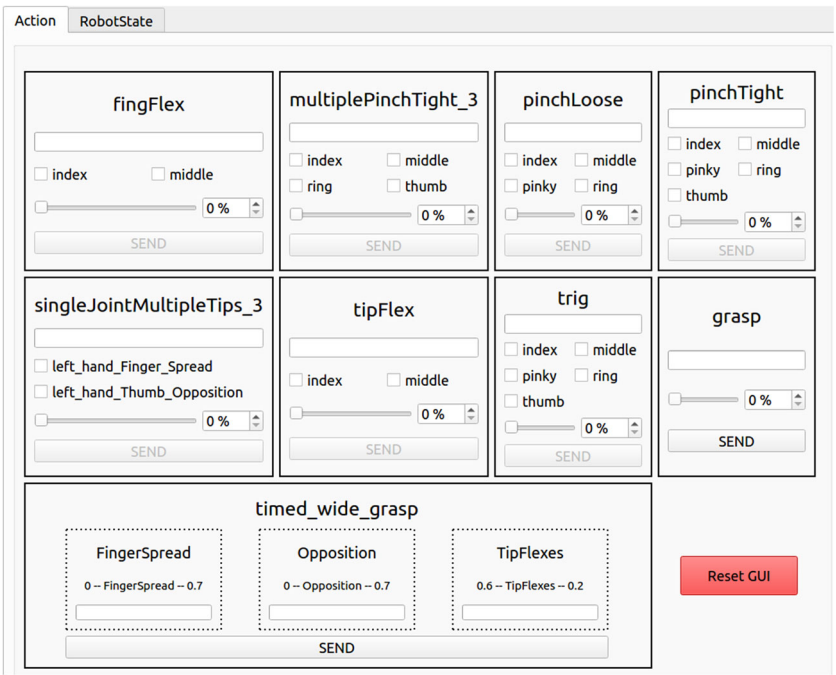

Fig. 6 ROS End-Effector GUI (1st tab) showing the grasping actions specifically defined for the for the SCHUNK SVH hand. For the timed grasping action (bottom row), the numbers below the inner grasping action names indicate the time delay to wait before and after executing the correspondent inner grasping action

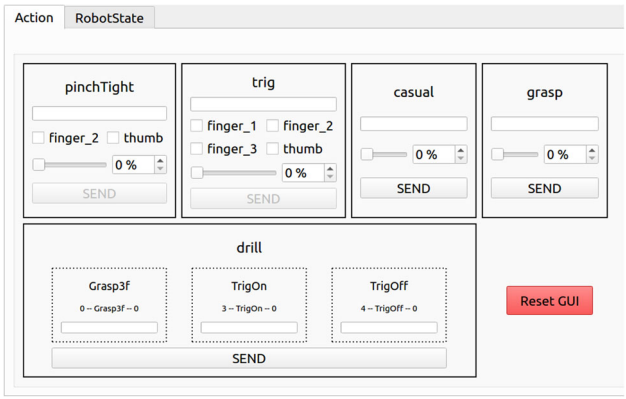

Fig. 7 ROS End-Effector GUI (1st tab) showing the grasping actions specifically defined for the for the HERI II hand

N fingers. Please note that for custom grasping action, this third information is not necessary.

A grasping action command, composed as described above, must be sent as a *message* through *ROS topics*, which are named communication channels over which nodes exchange messages unidirectionally, with a publisher-subscriber design. More precisely, we exploit the *ROS actions*, which, in practice, are aggregators of multiple ROS topics. With ROS actions, we can send through a topic the grasping action command, and receive through another one a feedback about its completion. We show some examples on how to command a grasping action in the tutorial of Section 8.3.

6.1 ROS End-Effector GUI

To facilitate the execution of the grasping action commands, we have developed the *ROS End-Effector Graphical User Interface* (GUI), shown as an additional node in the scheme of Fig. 1. The GUI's layout is specific for each end-effector: when it is launched, it dynamically loads only the grasping actions available (i.e., extracted primitives and defined custom grasping actions) for the end-effector in use. Hence, the layout will be different for each robotic end-effector as shown in the two different GUIs of Figs. 6 and 7. GUI elements are self-explanatory: below the grasping action name, a loading bar (empty in the two figures) shows the completion status of the grasping action while it is being executed; for primitive grasping actions, the end-effector body/bodies involved must be chosen by means of the checkboxes; finally, the intensity value can be set with the slider.

We have also implemented a second GUI tab (Fig. 8) where the robot states (like joints positions and velocities) can be monitored. Furthermore, the same data can be drawn in real-time in the integrated plot.

6.2 Hardware Abstraction Layer

The Hardware Abstraction Layer (Fig. 1) is the mean with which the *ROS End-Effector Executor* node communicates with the robotic end-effector. This layer abstracts the details of the end-effector in use, like specific hardware components, protocols and data fields. Thanks to this abstraction, it is possible to generalize the way the position set-goals of the actuators are sent to the robot, making the communication simple and safe, despite all the possible hardware differences in each end-effector. Since the HAL, and not the *ROS End-Effector Executor* node, must communicate directly with the robot, only the HAL must be specialized for the end-effector in use. To facilitate the integration of a new end-effector, a C++ abstract class, `EEHal`, is provided by our framework. A specific HAL for any real end-effector can be implemented deriving the provided `EEHal`. The communication between the *ROS End-Effector Executor* node and the

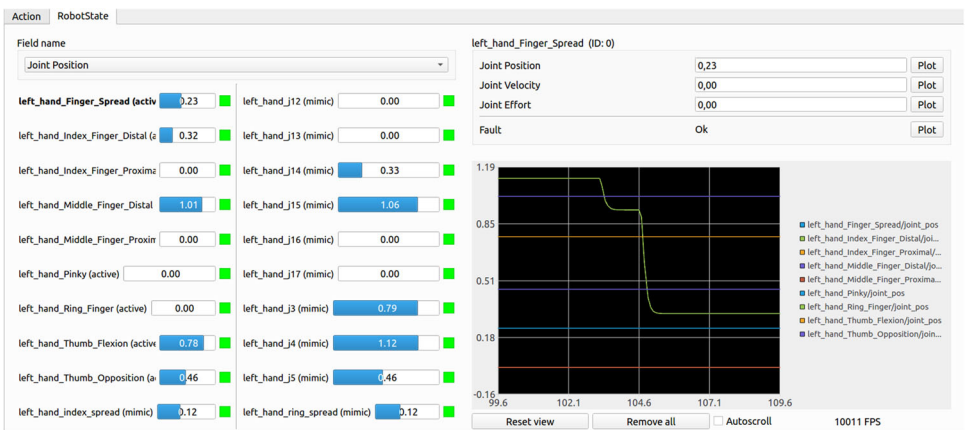

Fig. 8 ROS End-Effector GUI (2nd tab) showing and plotting the end-effector states in real-time. This figure shows the GUI of the SCHUNK SVH hand

`EEHal` is already implemented, hence only the robot-side communication must be defined. In the practice, only two methods must be implemented: a `sense()` and a `move()`, as shown in Code 2.

Code 2: C++ methods of `EEHal` abstract class to be implemented for a new end-effector

```
class EEHal {

public:
  [...]
  // Here the motor position arriving from the robot
  // must be copied into a EEHal member
  virtual bool sense() = 0;

  // Here the motor position references (stored in a
  // EEHal member) must be sent to the robot
  virtual bool move() = 0;
}
```

In the `sense()`, actuator positions must be gathered from the end-effector, to then fill a member of the class with this data. This permits to the already implemented methods to send back the feedback to the *Executor* node. In the `move()`, the end-effector commands must be sent, in according to follow the position set-points of the actuator, coming from the *Executor* node, and extracted from the requested grasping action command.

As a use case, we have implemented some HAL that are available within our software package. A `DummyHal` is present to communicate with any simulated robotic end-effectors. Indeed, the framework offers built-in support for *RViz* [35], the ROS standard kinematic visualization tool, and *Gazebo* [36], the de facto ROS dynamic simulator. The simulation in Gazebo is obtained thanks to dedicated plugins. An additional feature is present to integrate a ROS tool called *dynamic reconfigure* [37] which permits to tune online the PID gains of the simulated robotic end-effector.

Another HAL, the `XBotHAL`, has also been implemented to control a robot with XBot [38], a real-time software framework adaptable quickly to different hardware.

Finally, specific HALs have been added for the experiments that we have conducted with the Dagana, Robotiq 2F-85, qb SoftHand, OnRobot 3FG15, and HERI II end-effectors.

7 Experimental Validation

The framework flexibility and adaptability to different end-effectors, from simple grippers to complex human-like hands (Fig. 9), are validated both in simulation and on real end-effectors.

The SCHUNK SVH is a complex humanoid hand with 9 actuators and 20 DoF. Due to the lack of dynamic parameters for the simulation, we have been able to test this hand only in a set of pure-kinematic experiments. Nevertheless, the large number of possible fingers movements permitted us to validate the automatic extraction of primitive grasping actions. We have already reported some grasping actions performed with this hand as examples in the previous Figs. 2 and 4.

Some demonstrations with the Gazebo simulator have been performed with the Robotiq 2F-140, and Robotiq 3F end-effectors. In Fig. 10, the Robotiq 3F is performing a timed grasping action, called "timed_wide_grasp". This end-effector can move the two near fingers laterally, other than closing all of them toward the palm. So, we take advantage of this feature to perform a wide grasp. With the first inner grasping action "Open_Wide", the two near fingers are spread, as visible in the second image of Fig. 10. With the second inner grasping action "Grasp", all the fingers are closed, as visible in the third and fourth images. The two inner grasping actions are executed in sequence when the user requests the defined timed grasping action.

Validations with real end-effectors in extracting the grasping primitives and executing them through the ROS End-Effectors modules have been performed with the Dagana, Robotiq 2F-85, qb SoftHand, OnRobot 3FG15, and HERI II

Fig. 9 The end-effectors used to validate the ROS End-Effector framework (in simulation and/or with real hardware); from top to bottom and from left to right: SCHUNK SVH, Robotiq 2F-140, Robotiq 3F, Dagana, Robotiq 2F-85, qb SoftHand, OnRobot 3FG15, and HERI II

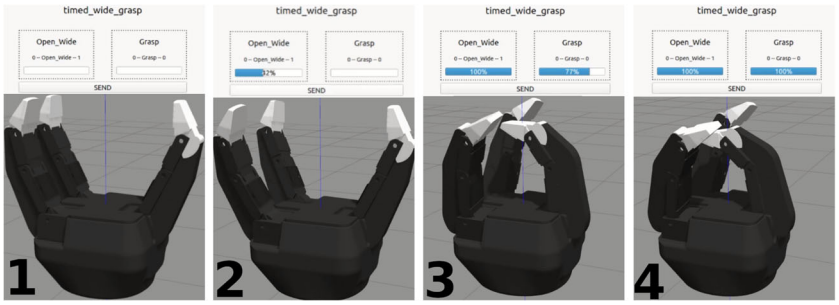

Fig. 10 The Robotiq 3F simulated in Gazebo, performing a custom timed grasping action, where the fingers are spread to execute a wide grasp. At the top of each image, the GUI loading bars show the execution state of each inner grasping action

end-effectors. In Fig. 11, the framework is used to command the Dagana gripper with the ROS End-Effector GUI. This end-effector is a single-actuator gripper with a unique moving part to open and close the beak. In the figure, the first tab of the GUI (at the top-right on the image) shows the grasping actions extracted for this particular end-effector and the table for a quick look at the joint states. The second tab (at the bottom-right) shows the live plot of the joint states of the end-effector.

In Fig. 12, we show the same pick and place task performed with three different end-effectors, Robotiq 2F-85, qb SoftHand, and OnRobot 3FG15. Once the primitive grasping actions are extracted from each one and their respective HAL implemented, in no-time it has been possible to command these different end-effectors. At the top of each image, the GUI shows the primitives extracted and the current primitive in execution.

In another experiment, we present a grasping scenario with the HERI II hand mounted on the IIT-INAIL arm [39] (Fig. 13). The goal is to grasp three objects, which require different grasping actions due to their different shapes. The arm trajectories were predefined according to the locations of the objects since object detection was not the focus. We have exploited a different grasping action for each object, as visible in Fig. 14. At the top, the first object is picked up with a primitive grasping action, *PinchTight*. For the other two objects (middle and bottom images of Fig. 14), custom grasping actions have been defined beforehand, composing some *Trigs* primitives. Please note that the "3f_grasp" is commanded with a percentage below 100% to not exert too much force in vain. All three grasping actions are commanded to the HERI II hand through the ROS End-Effector GUI, whose snapshots are visible in the right parts of Fig. 14. In Fig 15, the plots of the actuator positions and actuator currents of the end-effector are shown; we have highlighted the time sections where the three grasping actions are executed.

With the presented experimental demonstrations, we have shown the ROS End-Effector framework flexibility to automatically extract the capabilities of very different end-effectors under the form of primitive grasping actions. Furthermore, we show how the primitive grasping actions can be utilized to define in a simple way more complex custom grasping actions.

The user was able to command different types of grasping actions to both simulated (only kinematic in RViz and dynamic in Gazebo) and real end-effectors, in an agnostic way. Indeed, the framework permitted to abstract the hardware details such that the user had not deal at all with position or current actuator references, but just with the grasping actions. Moreover, the GUI simplified furthermore the communication with the end-effector. The demonstration showed the flexibility, the portability and the ease of use of our proposed framework, by obtaining different particular grasps using different kinds end-effectors and grippers.

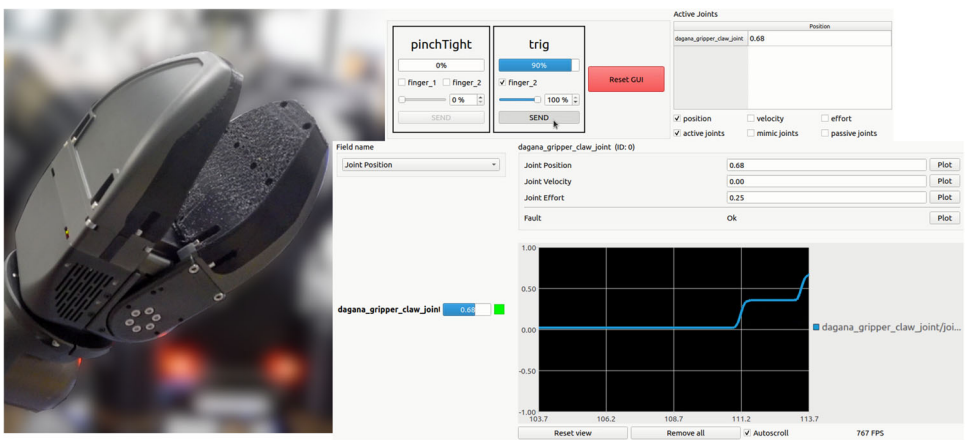

Fig. 11 The Dagana gripper, performing a grasping action. On the right side the two tabs of the ROS End-Effector GUI, loaded dynamically for the end-effector in use: at the top the grasping actions available and a summary of the joint states, at the bottom the joint states together with a plotting control

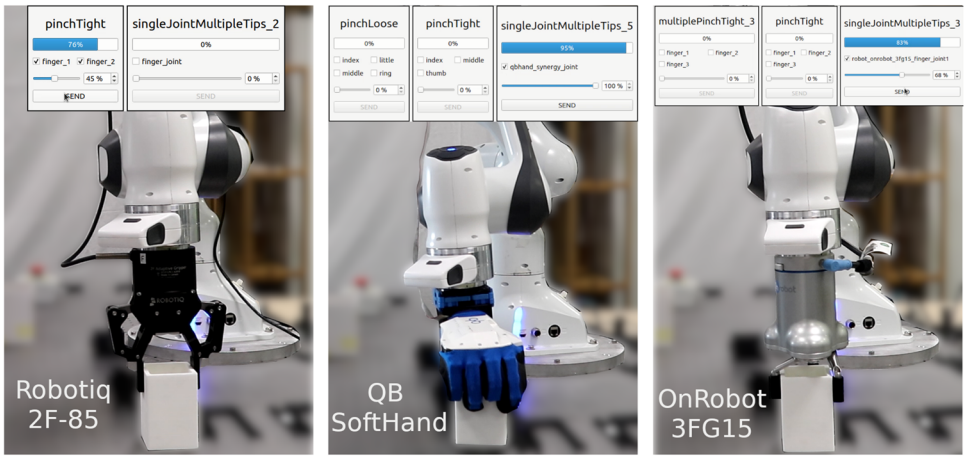

Fig. 12 Three different grippers are used for a pick and place task with the same object. ROS End-Effector has extracted automatically the primitive grasping actions of each gripper and three HALs have been implemented to communicate with the low level gripper interfaces. The GUIs at the top are used to command the extracted primitives

8 Tutorial

In this section, we present the fundamental steps to integrate a new end-effector in the framework. For more detailed guidelines, see the general documentation at https://advrhumanoids.github.io/ROSEndEffectorDocs and the Doxygen [40] generated API documentation at https://advrhumanoids.github.io/ROSEndEffector/index.html.

8.1 Prepare your Model

As stated, a few configuration files describing the end-effector, the *URDF* and *SRDF* files, are necessary to let ROS End-Effector automatically extract the primitive grasping actions. We will skip explanations about *URDF* file, since it is the well-known format to describe a robot in the ROS world. Instead, we will focus on some information that is necessary to have in the *SRDF* file. The *SRDF* is a ROS standard format that adds some information not included in the *URDF* file. For ROS End-Effector, two features of the *SRDF* are used. The first feature is used to group joint and links, so that the framework can retrieve them as kinematic chains. The second feature defines the passive joints, i.e., joints that are not moved by any actuator, but only by external forces. In Code 3, we provide a self-explanatory example of the *SRDF* file for the SCHUNK SVH hand.

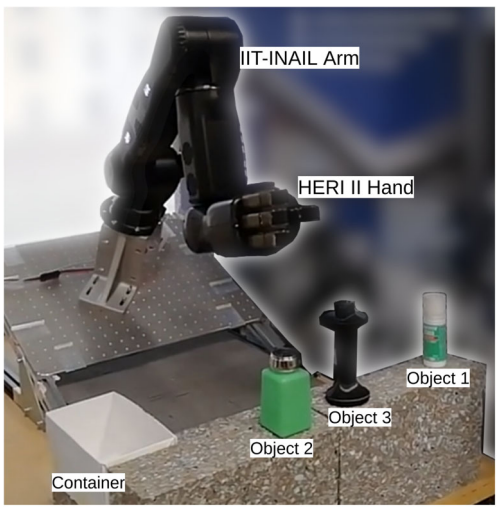

Fig. 13 The setup of the experiment with HERI II hand and IIT-INAIL arm. Three objects of different shapes must be picked up and placed in a container by commanding the grasping actions synthesized for the HERI II hand

Fig. 14 The three grasping actions performed during the experiment with HERI II hand and IIT-INAIL arm. On the right side, the GUI shows the grasping actions in execution. The loading bars are not fully completed (to 100%) because of small errors in tracking the actuator reference positions

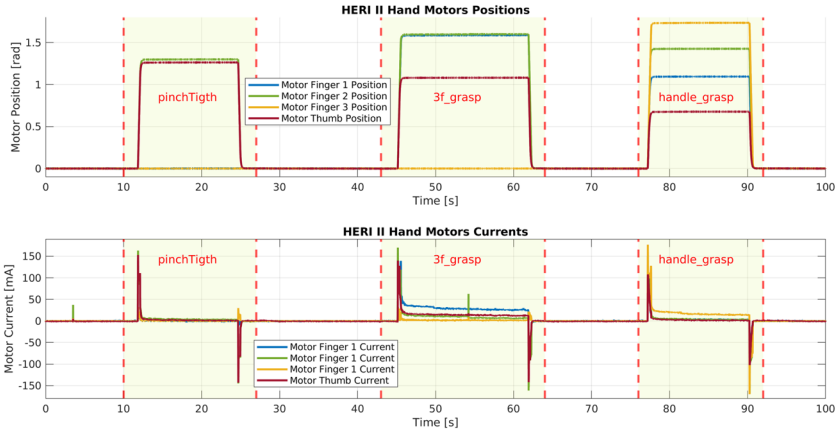

Fig. 15 Actuators positions and currents of HERI II hand during the experiment with the IIT-INAIL robot arm. In the plot are enlightened the time sequences where the hand is holding an object with a grasping action

Code 3: Example of a *SRDF* file

```
<robot name="schunk_svh">

  <group name="thumb">
    <chain base_link="left_hand_e1"
      tip_link="left_hand_c"/>
  </group>
  <group name="index">
    <chain base_link="left_hand_h"
      tip_link="left_hand_t"/>
  </group>
  [...]

  <group name="end_effector_fingers">
    <group name="thumb" />
    <group name="index" />
    <group name="middle" />
    <group name="ring" />
    <group name="little" />
  </group>

  <end_effector name="schunk_end_effector"
    parent_link="base_link"
    group="end_effector_fingers"/>

  <!-- SCHUNK SVH does not have these -->
  <!-- <passive_joint name="passive1" /> -->
</robot>
```

8.2 Run the Offline Phase

After the robot *URDF* and *SRDF* are ready, we can run the *Primitive Extractor* node (Fig. 1) with the dedicated *ROS launch* file that can be run in a bash terminal with the command shown in the Code 4 .

Code 4: Running the primitive extraction

```
roslaunch end_effector findActions.launch
  hand_name:=<my_hand>
```

Where `<my_hand>` is the name of the robot defined in the *URDF* and *SRDF* files. This command will automatically extract the primitive grasping actions of the loaded hand, as described in Section 4.1. The generated *YAML* files of the primitives will be stored locally in a system's folder.

If necessary, ROS End-Effector C++ API methods can be exploited to create additional custom grasping actions (Section 5). As an example, we report the definition of a "schunkGrasp" composed grasping action for SCHUNK SVH in the C++ Code 5.

Code 5: Composed grasping action definition

```
// We load the previously extracted primitives, and
// we use them to create the composed grasping action
MapActionHandler mapsHandler;
mapsHandler.parseAllPrimitives(
  folderForActions + "/primitives/");

// Let's create the ActionComposed object
ActionComposed schunkGrasp ("schunkGrasp");

// We fill the action with trigs
schunkGrasp.sumAction (
  mapsHandler.getPrimitive("trig", "index"));
schunkGrasp.sumAction (
  mapsHandler.getPrimitive("trig", "middle"));

// For the thumb, we do not want a complete closure,
// so we pass a scale factor < 1 (0.3)
schunkGrasp.sumAction (
  mapsHandler.getPrimitive("trig", "thumb"), 0.3);

// We store the new action into a YAML file, so it
// can be retrieved in the online phase
YamlWorker yamlWorker;
yamlWorker.createYamlFile (&schunkGrasp,
  folderForActions + "/generics/");
```

Timed grasping actions and generic grasping actions can be created with C++ methods in a similar way. For generic grasping actions, we must directly indicate the position set-points of the actuators, instead of "summing" some previously defined grasping actions. Alternatively, generic grasping actions can be defined during the online phase, through ROS Services, as explained in Section 8.3 with the Code 8.

8.3 Run the Online Phase

Like for the offline phase, a single command is necessary to run the *ROS launch* file of the online phase, as shown in Code 6.

Code 6: Running the online phase

```
roslaunch end_effector rosee_startup.launch
  hand_name:=<my_hand> hal_lib:=<my_hal>
```

As before, <my_hand> must be substituted with the name of the robot. This command will run both the *Grasping Actions Executor* node and the HAL shown in the bottom part of Fig. 1. The wanted <my_hal> library must have been previously implemented for the specific End-Effector hardware, as explained in Section 6.2. When launching the online phase, the selected HAL will be loaded dynamically by ROS End-Effector. If the robot is only simulated (with *RViz* and/or *Gazebo*), we can use the available `DummyHal`.

Grasping actions available for the end-effector can be retrieved through a dedicated ROS service, with the command in Code 7.

Code 7: Retrieving the grasping actions

```
rosservice call
  /ros_end_effector/grasping_actions_available
    "action_type: 0" #0 is for primitives
```

Where `action_type` is an index to indicate which kind of grasping action we want to retrieve.

New generic grasping actions can be added online with another ROS service. We have to indicate the new action name (`action_name`), the actuator position set-points (`action_motor_position`), and what actuators are used in the grasping action (`action_motor_count`: 0 if the actuator is not used, hence no reference will be set) as it is shown in Code 8. It is also possible to store the grasping action (`emitYaml: true`) together with the other created during the offline phase, to retrieve it for future tasks. Please note that if a generic grasping action is created online in such a way, the framework must be restarted to show it in the GUI.

Code 8: Adding a new grasping action online

```
rosservice call
  /ros_end_effector/new_generic_grasping_action
    "newAction:
      action_name: 'newFancyAction'
      action_motor_position:
        name:
        - 'motor_1'
        - 'motor_2'
        position:
        - 1.2
        - 0.75
      action_motor_count:
        name:
        - 'motor_1'
        - 'motor_2'
        position:
        - 1
        - 1
      emitYaml: true"
```

As explained in Section 6, the GUI is available to request the grasping actions. To run the GUI, a *ROS launch* file can be used, as shown in Code 9.

Code 9: Running the GUI

```
roslaunch rosee_gui gui.launch
```

Instead of using the GUI, it is also possible to command the grasping actions directly through the dedicated ROS topic. In Code 10, we show the most important ROS message fields to fill when demanding a grasping action.

Code 10: Commanding a grasping action

```
rostopic pub
  /ros_end_effector/action_command/goal
    rosee_msg/ROSEECommandActionGoal
     "[...]
     goal:
       goal_action:
         [...]
         action_name: 'pinchTight'
         action_type: 0
         selectable_items: ['index', 'thumb']
         percentage: 1"
```

The field `action_name` is the name of the grasping action (in this case, a `pinchTight`); `action_type` must be 0, 1, 2, or 3 for *primitive*, *generic*, *composed* and *timed* grasping action, respectively; `selectable_items`, used only for primitive grasping actions, indicates the body/bodies involved in the primitive; `percentage` (with values from 0 to 1) is the $0\% - 100\%$ intensity value to perform only partial movements of the fingers.

9 Conclusion and Future Works

We have presented ROS End-Effector, an open-source software framework which facilitates the integration of robotic end-effectors, abstracting the low-level hardware details. This manuscript extends the preliminary presentation of the framework of [8].

To achieve this functionality, ROS End-Effector automatically extracts the specific end-effector motion capabilities (e.g, precise pinching or single finger movements) in the form of *primitive grasping actions* during an offline process. In this offline phase the primitive grasping actions are extracted from the end-effector model described in the ROS standard formats *URDF* and *SRDF* (Section 4).

Given the extracted primitive grasping actions, the framework permits to combine them for generating *custom grasping actions* with different and more complex postural configurations that can be executed by the robotic end-effector. These custom grasping actions are classified as: (1) *composed grasping action*, created as the sum of others grasping

actions (Fig. 4); (2) *timed grasping action*, created as multiple inner grasping actions executed in sequence (Fig. 5); (3) *generic grasping action*, created/defined by the user (for example, as in Code 8). Finally, the composition of custom grasping actions is facilitated by the C++ API methods, interfaces, and ROS services provided by the ROS End-Effector framework (Section 5).

Thanks to the above primitive grasping actions approach as well as other user-created grasping actions that can be added to the repertoire of the available grasping actions, ROS End-Effector hides from the user the low-level interface of the end-effector, permitting the user to command grasping actions instead of sending directly the references necessary to the end-effector's actuators (Section 6). A grasping action is sent to ROS End-Effector through ROS topics, either using a ROS node or the dedicated GUI. The ROS End-Effector GUI dynamically builds its layout showing only the grasping actions extracted/defined for the specific end-effector in use. The user can choose a particular grasping action to command while receiving visual feedback about its completion during the execution. The same GUI permits to monitor and plot at run-time the available states of the end-effector (Section 6.1). The ROS End-Effector framework takes into account the grasping action command and communicates directly with the end effector through the implemented *Hardware Abstraction Layer*. The implementation of the Hardware Abstraction Layer, specific for the end-effector in use, is facilitated by the C++ structures provided; indeed only the specific communication with the end-effector device must be implemented (Code 2). In any case, the framework is ready to be used with any simulated robotic end-effector thanks to the available `DummyHal`, and with some real end-effectors for which the HAL has been already implemented (Section 6.2).

The flexibility and adaptability of the framework have been tested in simulation with various robotic hands, from complex humanoid hands (SCHUNK SVH, kinematic visualization only), to simpler grippers (Robotiq 2F-140, Robotiq 3F). Tests with real hardware have been conducted with Dagana, Robotiq 2F-85, OnRobot 3FG15, qb SoftHand, and HERI II. All the tests include an offline phase, where the primitive grasping actions have been automatically extracted (and some custom grasping actions have been defined by the user), and an online phase, where the grasping actions have been commanded and executed by the end-effectors (Section 7).

Finally, a short practical tutorial for new ROS End-Effector users has been included to demonstrate the simplicity of the setting up process for a new end-effector using the ROS End-Effector framework (Section 8).

Future works on the framework will consider the extension of the functionality towards autonomous grasping action composition providing the necessary information of the object to be grasped in terms of geometrical shape and mechanical/dynamic properties.

Supplementary information

The ROS End-Effector package is available in the official ROS/ROS2 repositories with the name `ros-<ROS_DISTRO>-end-effector`. The C++ code of ROS End-Effector is available open-source with the Apache−2.0 license at https://github.com/ADVRHumanoids/ROSEndEffector. A video presenting the features of ROS End-Effector and showing the experiments carried out is attached with this article, and it is also available at the following link: https://youtu.be/X0qpSsFQg1M.

Acknowledgements This work was supported by the European Union's Horizon 2020 research and innovation programme (Grant numbers 732287 (ROS-Industrial)).The authors want to thanks Diego Vedelago and Stefano Carrozzo for the support with the experiments on the HERI II hand and Arturo Laurenzi for the guidance and the support in the implementation of the GUI.

Author Contributions Conceptualization, Methodology, Formal Analysis, and Investigation: Davide Torielli, Liana Bertoni, Nikos Tsagarakis, Luca Muratore; Software: Davide Torielli, Liana Bertoni, Fabio Fusaro, Luca Muratore; Writing - original draft preparation: Davide Torielli; Writing - review and editing: Davide Torielli, Nikos Tsagarakis, Luca Muratore; Validation: Davide Torielli, Fabio Fusaro, Luca Muratore; Visualization: Davide Torielli; Supervision and Funding acquisition: Nikos Tsagarakis, Luca Muratore.

Funding Open access funding provided by Università degli Studi di Genova within the CRUI-CARE Agreement. This work was supported by the European Union's Horizon 2020 research and innovation programme (Grant numbers 732287 (ROS-Industrial)).

Availability of data and materials The ROS End-Effector package is available in the official ROS/ROS2 repositories with the name `ros−ROS_DISTRO−end−effector`. A video presenting the features of ROS End-Effector and showing the experiments carried out is attached with this article, and it is also available at the following link: https://youtu.be/X0qpSsFQg1M.

Code Availability The C++ code of ROS End-Effector is available open-source with the Apache−2.0 license at https://github.com/ADVRHumanoids/ROSEndEffector.

Declarations

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Conflict of interest/Competing interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Negrello, F., Stuart, H.S., Catalano, M.G.: Hands in the real world. Frontiers in Robotics and AI. 6 (2020). https://doi.org/10.3389/frobt.2019.00147
2. Ruehl, S.W., Parlitz, C., Heppner, G., Hermann, A., Roennau, A., Dillmann, R.: Experimental evaluation of the schunk 5-Finger gripping hand for grasping tasks. IEEE International Conference on Robotics and Biomimetics. p 2465–2470 (2014). https://doi.org/10.1109/ROBIO.2014.7090710
3. Robotiq.: 2F-85 and 2F-140 Adaptive Robot Gripper. (2023) Available from: https://robotiq.com/products/2f85-140-adaptive-robot-gripper
4. Robotiq.: 3-Finger Adaptive Gripper. (2023) Available from: https://robotiq.com/products/3-finger-adaptive-robot-gripper
5. Catalano, M., Grioli, G., Farnioli, E., Serio, A., Bonilla, M., Garabini, M., et al.: 8. In: From Soft to Adaptive Synergies: The Pisa/IIT SoftHand. Springer International Publishing, p 101–125 (2016)
6. OnRobot.: 3FG15 Three Finger Gripper. (2023) Available from: https://onrobot.com/en/products/3fg15-three-finger-gripper
7. Ren, Z., Kashiri, N., Zhou, C., Tsagarakis, NG.: HERI II: A Robust and Flexible Robotic Hand based on Modular Finger design and Under Actuation Principles. IEEE/RSJ International Conference on Intelligent Robots and Systems, p 1449–1455 (2018) https://doi.org/10.1109/IROS.2018.8594507
8. Torielli, D., Bertoni, L., Tsagarakis, N.G., Muratore, L.: Towards an Open-Source Hardware Agnostic Framework for Robotic End-Effectors Control. IEEE International Conference on Advanced Robotics. (2021). https://doi.org/10.1109/ICAR53236.2021.9659331
9. Bertoni, L., Torielli, D., Zhang, Y., Tsagarakis, N.G., Muratore, L.: Towards a Generic Grasp Planning Pipeline using End-Effector Specific Primitive Grasping Actions. IEEE International Conference on Advanced Robotics. (2021). https://doi.org/10.1109/ICAR53236.2021.9659402
10. Piazza, C., Grioli, G., Catalano, M.G., Bicchi, A.: A Century of Robotic Hands. Annu. Rev. Control Robot. Auton. Syst. **2**(1), 1–32 (2019). https://doi.org/10.1146/annurev-control-060117-105003
11. Santello, M., Flanders, M., Soechting, J.: Postural Hand Synergies for Tool Use. The J Neurosc: The Official J Society Neurosci. **18**, 10105–10115 (1998). https://doi.org/10.1523/JNEUROSCI.18-23-10105.1998
12. Bizzi, E., Cheung, V.C.K.: The neural origin of muscle synergies. Front. Comput. Neurosci. **7**, 51 (2013). https://doi.org/10.3389/fncom.2013.00051
13. Santello, M., Bianchi, M., Gabiccini, M., Ricciardi, E., Salvietti, G., Prattichizzo, D., et al.: Hand synergies: Integration of robotics and neuroscience for understanding the control of biological and artificial hands. Phys. Life Rev. **17**, 1–23 (2016). https://doi.org/10.1016/j.plrev.2016.02.001
14. Latash, M.L.: Synergy. Oxford University Press, (2008)
15. Ciocarlie, M., Goldfeder, C., Allen, P.: Dimensionality reduction for hand-independent dexterous robotic grasping. IEEE/RSJ Int. Conf. Inte. Robot. Syst. **20**, 3270–3275 (2007). https://doi.org/10.1109/IROS.2007.4399227
16. Peer, A., Stanczyk, B., Buss, M.: Haptic telemanipulation with dissimilar kinematics. IEEE/RSJ International Conference on Intelligent Robots and Systems. (2005). https://doi.org/10.1109/IROS.2005.1545349
17. Ciocarlie, M.T., Allen, P.K.: Hand posture subspaces for dexterous robotic grasping. Int. J. Rob. Res. **28**, 851–867 (2009). https://doi.org/10.1177/0278364909105606
18. Griffin, W.B., Findley, R.P., Turner, M.L., Cutkosky, M.R.: Calibration and Mapping of a Human Hand for Dexterous Telemanipulation. Hand The. (2000). https://doi.org/10.1115/IMECE2000-2424
19. Gioioso, G., Salvietti, G., Malvezzi, M., Prattichizzo, D.: Mapping synergies from human to robotic hands with dissimilar kinematics: An approach in the object domain. IEEE Transactions on Robotics. **29**(4), 825–837 (2013). https://doi.org/10.1109/TRO.2013.2252251
20. Gabiccini, M., Bicchi, A., Prattichizzo, D., Malvezzi, M.: On the role of hand synergies in the optimal choice of grasping forces. Autonomous Robots. (2011). https://doi.org/10.1007/s10514-011-9244-1
21. Bicchi, A., Gabiccini, M., Santello, M.: Modelling natural and artificial hands with synergies. Philosophical Transactions of the Royal Society B: Biological Sciences. (2011). https://doi.org/10.1098/rstb.2011.0152
22. Morrow, J.D., Khosla, P.K.: Manipulation task primitives for composing robot skills. Proc. Int. Conf. Robot. Autom. **4**, 3354–3359 (1997). https://doi.org/10.1109/ROBOT.1997.606800
23. Kröger, T., Finkemeyer, B,, Wahl, F.M.: Manipulation Primitives — A universal interface between sensor-based motion control and robot programming. Robotic Systems for Handling and Assembly. p 293–313. (2011). https://doi.org/10.1007/978-3-642-16785-0_17
24. Felip, J., Laaksonen, J., Morales, A., Kyrki, V.: Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks. Rob. Auton. Syst. **61**(3), 283–296 (2013). https://doi.org/10.1016/j.robot.2012.11.010
25. Miller, A.T., Allen, P.K.: Graspit! A versatile simulator for robotic grasping. IEEE Rob. Autom. Mag. **11**(4), 110–122 (2004). https://doi.org/10.1109/MRA.2004.1371616
26. Malvezzi, M., Gioioso, G., Salvietti, G., Prattichizzo, D.: SynGrasp: A MATLAB toolbox for underactuated and compliant hands. IEEE Rob. Autom. Mag. **22**(4), 52–68 (2015). https://doi.org/10.1109/MRA.2015.2408772
27. León, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., Morales, A., et al.: OpenGRASP: A toolkit for robot grasping simulation. Simulation, Modeling, and Programming for Autonomous Robots. p 109–120 (2010). https://doi.org/10.1007/978-3-642-17319-6_13
28. León, B., Morales, A., Sancho-Bru, J.: From robot to human grasping simulation. Cognitive Systems Monographs. 19(May) (2014). https://doi.org/10.1007/978-3-319-01833-1
29. Sucan, I.A., Chitta, S.:MoveIt [Computer software]. (2023) Available from: http://moveit.ros.org
30. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., et al.: Japan Kobe ROS an open-source Robot operating system. ICRA Workshop on Open Source Softw. **3**, 5 (2009)
31. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J,. et al.: ROS Robot operating system [Computer software]. (2023). Available from: https://www.ros.org/
32. Coleman, D., McEvoy, A., Lautman, M.: MoveIt grasps [Computer software]. (2023). Available from: https://github.com/ros-planning/moveit_grasps
33. Liu, S., Yan, Y.: ROS2 Grasp library [Computer software]. (2023). Available from: https://github.com/intel/ros2_grasp_library
34. Pan, J., Chitta, S., Manocha, D.: FCL: A general purpose library for collision and proximity queries. IEEE International conference

on Robotics and automation. p 3859–3866 (2012) https://doi.org/10.1109/ICRA.2012.6225337
35. Hershberger, D., Gossow, D., Faust, J., Woodall, W.: RViz [Computer software]. (2023). Available from: http://wiki.ros.org/rviz
36. Agüero, C.E., Koenig, N., Chen, I., Boyer, H., Peters, S., Hsu, J., et al.: Inside the virtual Robotics challenge: Simulating real-time Robotic disaster response. IEEE Trans. Autom. Sci. Eng. **12**(2), 494–506 (2015). https://doi.org/10.1109/TASE.2014.2368997
37. Gassend, B.: Dynamic Reconfigure [Computer software]. (2023). Available from: http://wiki.ros.org/dynamic_reconfigure
38. Muratore, L., Laurenzi, A., Mingo Hoffman, E., Tsagarakis, N.G.: The XBot real-time software framework for robotics: from the developer to the user perspective. IEEE Rob. Autom. Mag. **27**(3), 133–143 (2020). https://doi.org/10.1109/MRA.2020.2979954
39. Barrett, E., Hoffman, E.M., Baccelliere, L., Tsagarakis, N.G.: Mechatronic design and control of a light weight manipulator arm for mobile platforms. IEEE/ASME international conference on advanced intelligent mechatronics (AIM). p 1255–1261 (2021) https://doi.org/10.1109/AIM46487.2021.9517389
40. van Heesch D.: Doxygen [Computer software]. (2023). Available from: https://www.doxygen.nl/index.html

Davide Torielli received his Bachelor's Degree in Computer Science Engineering and Master's Degree in Robotics Engineering at the University of Genova, in 2017 and 2019 respectively. He carried out part of his Master Thesis at IRS Lab, Castellón de la Plana, Spain. Since 2019 he is working at Istituto Italiano di Tecnologia for the HHCM research line, and since 2020 he is also a PhD student in Bioengineering and Robotics with IIT and University of Genova. He actively worked on the European project ROS End-Effector ROS-Industrial Focused Technical Project (ROSIN FTP). Its PHD is involved in the exploration of intuitive and smart teloperation interfaces for the control of complex mobile manipulators. He presented some of his works at the ROSIN FTP Webinar Series 2020, International Conference on Advanced Robotics (ICAR) 2021, IEEE International Conference on Robotics and Automation (ICRA) 2022, ROS Conference (ROSCon) 2022, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2022, and IEEE-RAS International Conference on Humanoid Robots (Humanoids) 2022. He is currently involved is some European Projects (H2020 CONCERT, H2020 SOPHIA) and in the Italian MISE founded project RELAX.

Liana Bertoni received his Bachelor's and Master's Degree, both at the University of Pisa, in Software Engineering and Robotics and Automation Engineering, respectively. She is currently enrolled as a PhD student in the University of Pisa with working place at the Istituto Italiano di Tecnologia (IIT) for the Humanoids & Human Centered Mechatronics (HHCM) group. Her research interests focus the variable impedance control for autonomous robotic systems working in high dynamical environments as industrial application as well as harsh environments where the key feature of adaptability is still a challenging topic in that research area. She is currently involved in the international project CONCERT (2021).

Fabio Fusaro was born on 9 March 1995 in Genova (Italy). He received the BD in Bioengineering and the MD, with full mark, in Robotics Engineering from the University of Genova, Italy, in 2017 and 2019, respectively. In 2023, he defended his PhD thesis entitled "Dynamic Task Allocation for Proactive Human-Robot Collaboration", carried out at the Human-Robot Interfaces and Physical Interaction (HRI2) laboratory, in collaboration with Politecnico di Milano in the Neuroengineering and Medical Robotics Laboratory (NearLab). He received his PhD degree, cum laude. He worked actively in the Horizon-2020 European projects SOPHIA and CONCERT, the ERC grant Ergo-Lean and the Amazon Research Awards 2019. He was also involved in transferring technology initiatives with various industrial partners, like the JOiiNT LAB IIT-INTELLIMECH at Kilometro Rosso Innovation District. He was finalist of Solution Award 2020 (MECSPE2020) and of Best Student Paper Award Finalist (I-RIM 2021). He is working as Robotics Engineer at Hiro Robotics focusing on the development of robotics solutions for E-waste recycling.

Nikolaos G. Tsagarakis received the D.Eng. degree in electrical and computer science engineering from the Polytechnic School, Aristotle University of Thessaloniki, Thessaloniki, Greece, in 1995, 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, respectively. He is currently the Head of the Humanoids & Human Centred Mechatronics Lab, Istituto Italiano di Tecnologia, Genova, Italy. His research interest is on humanoid robots, mechanism design, compliant and variable impedance actuators, wearable assistive devices and exoskeletons for poweraugmentation, and haptic Systems. He is an author or co-author of over 350 papers in research journals and at international conferences and holds 16 patents. He has received the Best Jubilee Video Award at IROS (2012), the 2009 PE Publishing Award from the Journal of Systems and Control Engineering and prizes for Best Paper at ICAR (2003) and the Best Student Paper Award at Robio (2013). He was also a finalist for Best Entertainment Robots and Systems - 20th Anniversary Award at IROS (2007) and finalist for the Best Manipulation paper at ICRA (2012), the Best Conference Paper at Humanoids (2012), the Best Student Papers at Robio (2013) and ICINCO (2014), Best Interactive Paper finalist at Humanoids (2016) and Best Interactive Paper at Humanoids (2017). He has been in the Program Committee of over 60 international conferences including IEEE ICRA, IROS, RSS, HUMANOIDS BIOROB and ICAR. Nikos Tsagarakis was Technical Editor of IEEE/ASME Transactions on Mechatronics (2012–2015) and from 2014 served on the Editorial Board of the IEEE Robotics and Automation Letters. He is currently a Senior Editor of IEEE/ASME Transactions on Mechatronics.

Luca Muratore is Technologist at Istituto Italiano di Tecnologia (IIT) in the Humanoids and Human-Centered Mechatronics (HHCM) research line. In 2020 He received his split-site Ph.D. degree in Electrical and Electronic Engineering from the University of Manchester and IIT, with a Ph.D. thesis entitled "A Flexible Cross-Robot Software Framework for Robot Control: from on-board to Cloud Execution". He obtained his bachelor's and master's degrees, both in Software Engineering at the University of Pisa in 2011 and 2014 respectively. He carried out his Master's thesis (on Distributed Systems and Cloud Computing) at Universidad Politecnica de Madrid (UPM) after a six months internship in the Distributed System Lab (LSD). Since May 2014 he has been working as a Lead Software Engineer at IIT for the HHCM research line. In 2015 he participated in the DARPA Robotics Challenge as a WALK-MAN team member. He was involved in several EU projects as the main responsible for the software integration given the design and development of the XBot software framework; the main ones are: FP7 WALK-MAN (with the EU innovation radar award for the XBot software architecture) H2020 CENTAURO, H2020 CogiMON and currently H2020 CONCERT, H2020 SOPHIA, Horizon Europe HARIA, and Horizon Europe euROBIN. His main research interests are software architecture for robotics, human-robot collaboration, robotics teleoperation, cloud robotics, grasp planning, and robotics manipulation. He was the coordinator of the European project ROS End-Effector, a ROS-Industrial Focused Technical Project and He is the co-coordinator of the Italian MISE founded project RELAX (Robot Enabler for Load Assistive RelaXation). He is on the Organization Committee of the IEEE Robotic Computing (IRC) conference as program co-chair, and He is engaged as responsible for the software architecture and integration in the Alberobotics IIT startup (https://alberobotics.it/), the JOiiNT LAB IIT-Intellimech (https://www.iit.it/it/tech-transfer/joint-labs/joiint-lab) and the IIT-Leonardo joint lab.

Authors and Affiliations

Davide Torielli[1,2] · Liana Bertoni[1,3] · Fabio Fusaro[4,5] · Nikos Tsagarakis[1] · Luca Muratore[1]

Liana Bertoni
liana.bertoni@iit.it

Fabio Fusaro
fabio.fusaro@outlook.it

Nikos Tsagarakis
nikos.tsagarakis@iit.it

Luca Muratore
luca.muratore@iit.it

[1] Humanoids and Human Centered Mechatronics (HHCM), Istituto Italiano di Tecnologia (IIT), Via S. Quirico, 19d, Genova 16163, Italy

[2] Department of Informatics, Bioengineering, Robotics, and Systems Engineering (DIBRIS), University of Genova, Via All'Opera Pia, 13, Genova 16145, Italy

[3] Department of Information Engineering (DII), University of Pisa, Via G. Caruso, 16, Pisa 56122, Italy

[4] Human-Robot Interfaces and physical Interaction (HRI2), Istituto Italiano di Tecnologia (IIT), Via S. Quirico, 19d, Genova 16163, Italy

[5] Department of Electronics, Information, and Bioengineering (DEIB), Politecnico di Milano, Via Giuseppe Ponzio, 34, Milano 20133, Italy

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com