# Towards an Open-Source Hardware Agnostic Framework for Robotic End-Effectors Control

Davide Torielli*†, Liana Bertoni*, Nikos Tsagarakis* and Luca Muratore*
*Humanoids and Human Centered Mechatronics (HHCM), Istituto Italiano di Tecnologia, Genova, Italy
†Department of Informatics, Bioengineering, Robotics, and Systems Engineering (DIBRIS),
University of Genova, Genova, Italy
{davide.torielli, liana.bertoni, nikos.tsagarakis, luca.muratore}@iit.it

*Abstract*— **Nowadays a wide range of industrial grippers are available on the market and usually their integration to robotics automation systems relies on dedicated software modules and interfaces specific for each gripper. During the past two decades, more sophisticated end-effector modules that target to provide additional functionality including dexterous manipulation skills as well as sensing capabilities have been developed. The integration of these new devices is usually not trivial, requiring the development of brand new, tailor-made software modules and interfaces, which is a time consuming and certainly not efficient activity. To address the above issue and facilitate the quick integration and validation of the new end-effectors, we developed the ROS End-Effector open-source framework, which provides a software infrastructure capable to accommodate a range of robotic end-effectors of different hardware characteristics (number of fingers, actuators, sensing modules and communication protocols) and capabilities (with different manipulation skills, such as grasping, pinching, or independent finger dexterity) effectively facilitating their integration through the development of hardware agnostic software modules, simulation tools and application programming interfaces (APIs). A key feature of the ROS End-Effector framework is that rather than controlling each end-effector in a different and customized way, following specific protocols and instructions data fields, it masks the physical hardware differences and limitations (e.g., kinematics and dynamic model, actuator, sensor, update frequency, etc.) and permits to command the end-effector using a set of high level grasping primitives. The framework capabilities and flexibility in supporting different robotics end-effectors are demonstrated both in a kinematic/dynamic simulation and in real hardware experiments.**

## I. Introduction

The realization of robotic end-effectors that can mirror the performance of the human hands in terms of manipulation dexterity and grasping strength, has been one of the major challenges of the robotics research in the past few years. As a result, diverse robotic end-effector principles have been realized exploring different kinematics and actuation arrangements, attempting to trade off the end-effector implementation complexity with its manipulation dexterity and grasping strength capabilities. In particular, a wide range of industrial grippers offering grasping functionality are available in the market and usually their integration to robotics automation systems relies on customized software modules and interfaces. In addition, some works on more sophisticated end-effector modules, which target to provide dexterous manipulation as well as sensing capabilities, have resulted to a number of novel and more complex end-effectors, which provide a broader set of manipulation skills [1].

The control of these more dextrous robotic hands has been an attractive field for the robotics research community. In neuroscience, several works [2], [3], have shown that, in order to grasp an object, humans control their hands in a configuration subspace that is considerably smaller than the space generated by the amount of human hand degrees of freedom (DoF). From these observations the concept of *synergies*, which can be seen as the common patterns of actuation of the human hand, has been introduced. Following this idea, many studies in robotic grasping have focused on exploiting the synergies for controlling robotic end-effectors. Thanks to the dimensionality reduction of the controlling space, the grasp planning is more efficient and more adaptable to various kinds of end-effectors [4]. In general, the adoption of synergies in robotics sets the basis for the development of frameworks to map human hand synergies into robotics end-effectors ones, despite the possible different kinematics. This mapping has been explored at different levels, such as joint space [5] and Cartesian space [6]. For a more comprehensive overview of the synergy-based works, the reader can refer to [7].

At the same time, grasping and manipulation primitives have been a fundamental basis for different studies. They are usually identified as atomic elements that can 1) be used to compose more complex actions and 2) be employed by high-level grasping and manipulation interfaces to hide low-level hardware details. The pioneered work by [8] classifies a *Manipulation Task Primitive* by the relative motion between two (rigid) parts. The goal of this work was to build a library of robot capabilities in the manipulation domain, providing a higher level of abstraction for more complex manipulation tasks. In [9], the *Manipulation Primitives* are identified as the high-level interfaces, which choose the control signals for their sensor-based hybrid switched-system controller. In [10], an abstract layer of *Control Primitives* is built as a vocabulary, which can be coordinated using state machines to describe complex actions.

For what concerns relevant software for end-effector motion analysis, planning and control, several tools have been developed in the past years. *GraspIt!* [11] permits to plan grasps of a variety of objects with different hands. It includes grasp analysis routines and control algorithms to compute the joint forces necessary to follow the trajectory generated. *Syngrasp* [12] is a MATLAB toolbox which features several functions to investigate the grasp properties including the 1) controllable forces and object displacement, 2) manipulability analysis, and 3) grasp stiffness and quality measures. The above-mentioned software tools focus on the analysis and the synthesis of grasps, but they do not target to actually abstract the end-effector hardware. Therefore, while the development of the above methodologies and tools have facilitated the control of more complex and dexterous end-effectors, a barrier that still prevents their integration is present, and a wider use of these end-effectors is impeded by the lack of effective software/control components and interfaces that can permit to abstract the end-effector hardware, enabling the efficient and transparent integration of these end-effectors in industrial robotics and automation lines.

This work is inspired by the above observations and proposes the ROS End-Effector software framework, which leverages on the concept of *primitive grasping actions*, that are automatically extracted from the end-effector hardware and configuration models. These primitives are used to synthesize grasping and manipulation actions, effectively enabling to control the end-effector in a subspace of reduced dimensionality by commanding a combination of the available grasping primitives instead of each individual joint, thus hiding the hardware details of the end-effector in use. This concept is detailed in Section III.

The main contribution of the proposed framework is therefore that it effectively facilitates the integration of different end-effectors by leveraging on a number of novel hardware agnostic software modules that provide end-effector abstraction to the higher control layers of a robotic system through the following unique features:

- Automatic extraction of end-effector motions given the kinematic model and configuration of the end-effector,
- Automatic synthesis of primitive grasping actions given the available end-effector motions and its hardware configuration, e.g., the number and location of the fingers and their possible interactions and motions combinations.
- Simulation tools and high-level APIs for the end-effector module based on the synthesized available grasping actions.

The framework functionalities in abstracting and controlling different robotics end-effectors are demonstrated in simulation experiments using a number of different end-effectors including the SCHUNK SVH [13] (kinematic-only visualization), Robotiq 2F-140[1], Robotiq 3F[2] and the qb SoftHand[3]. Validations with real hardware have been successfully conducted on the HERI II hand [14] demonstrating the capabilities of the ROS End-Effector framework to seamlessly abstract and control end-effectors of very diverse physical hardware and configurations in terms of automatic generation and usage of the grasping primitives.

[1]`https://robotiq.com/products/3-finger-adaptive-robot-gripper`

[2]`https://robotiq.com/products/2f85-140-adaptive-robot-gripper`

[3]`https://qbrobotics.com/products/qb-softhand-research/`

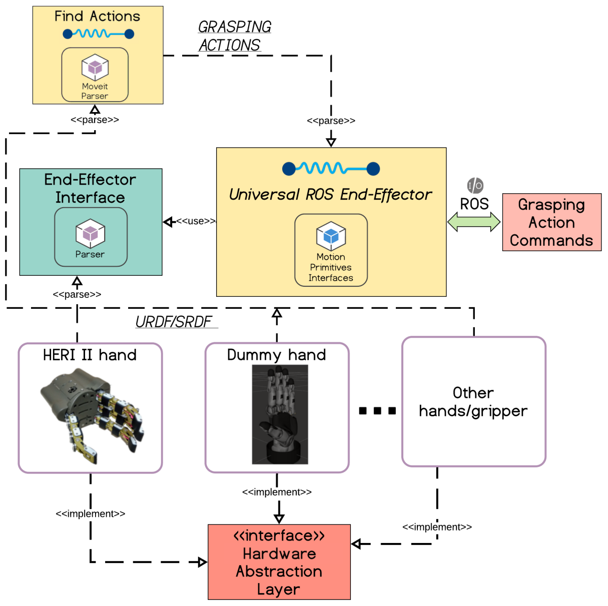

Fig. 1. Scheme of the ROS End-Effector Framework. At the top, the Find Actions node in the offline phase generates the Grasping Actions from the end-effector URDF and SRDF files. In the online phase, the generated Grasping Actions are parsed by the Universal ROS End-Effector node, which will communicate with the end-effector in use through the End-Effector Interface and the specific Hardware Abstraction Layer (HAL) implemented for the end-effector in use. The Dummy hand block represents any simulated hand in *RViZ* and/or *Gazebo* (the SCHUNK SVH hand is shown as an example).

The rest of the paper is structured as follows. Section II briefly introduces the ROS End-Effector framework. Section III describes the methodology for the extraction of grasping actions and primitive motions. Section IV discusses the grasping action command, while Section V presents the hardware abstraction layer (HAL). Finally, Section VI presents validation studies of the proposed framework and Section VII draws the conclusions and future work plans.

## II. ROS End-Effector Framework

The ROS End-Effector's framework is composed by two main components:

- An *offline* component, where the end-effector capabilities are automatically extracted, under the form of primitive grasping actions.
- An *online* component, where the end-effector can execute tasks by receiving grasping actions commands, abstracting the low level hardware details.

An overview of the framework is depicted in Fig. 1 while the details of the above two components are discussed in the next sections.

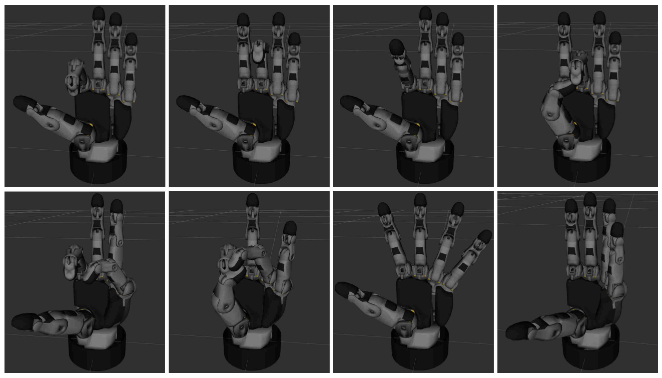

Fig. 2. Primitive grasping actions extracted for SCHUNK SVH. From left to right, in the first row: Trig (*index*), TipFlex (*middle*), FingFlex (*index*), and PinchTight (*thumb and middle*). In the second row: PinchLoose (*index and little*), MultiPinchTight_3 (*thumb, index and ring*), SingleJointMultipleTips_3 (which uses the *finger_spread joint* joint to move the *index*, *ring*, and *little* fingers), and another SingleJointMultipleTips_3 (which uses the *thumb_opposition* joint to move the *thumb*, *ring*, and *little* fingers).

## III. Offline Component: Primitive Grasping Actions Extraction

A *primitive grasping action* is a collection of fingers movements, which are specific for each end-effector and permit to perform the grasping of an object.

The information contained in a primitive grasping action essentially is:

- The grasping action name.
- The main end-effector's elements involved in the grasping action, e.g., a *trig* performed with the *index* finger.
- The position set-points of the actuators, which describe the particular end-effector's pose associated with the grasping action.

Furthermore, when commanding a primitive grasping action, a certain $0\% - 100\%$ scaling value can be set to scale the position set-points of the actuators and perform a partial closure of the fingers.

### A. Automatic Extraction of the Primitive Grasping Actions

A *primitive grasping action* is an atomic motion of the end-effector's elements, that can not be decomposed into simpler actions.

As anticipated before, the offline component of the ROS End-Effector explores the robot model to automatically extract the primitives grasping actions. The framework aims to be flexible when a new end-effector must be controlled, so it requires only two configuration files representing the robot: the Unified Robot Description Format (*URDF*) and the Semantic Robot Description Format (*SRDF*), both commonly used in ROS. The former is the standard format to describe the robot model in ROS; the latter adds some information that is not included in the URDF format, like defining the fingers as kinematic chains composed by links and joints, and selecting the non-actuated joints as passive.

We divide the primitive grasping actions into three main categories. The trig-type primitives (which include *Trig*, *TipFlex*, and *FingFlex*) are dedicated to move a single finger or a phalanx toward the palm. The pinch-type primitives (which include *PinchTight*, *PinchLoose*, and *MultiPinchTight_N*) are suitable for precise grasps. With them, the fingertips move towards each other to pick narrow or little objects. The last category include the *SingleJointMultipleTips_N* primitive, where a single actuator moves $N (\geq 2)$ fingertips.

A *Trig* primitive grasping action performed with a finger is possible if there is at least one actuator dedicated to move only that finger, closing the whole finger toward the palm. If a single finger is moved by more than one actuator, the end-effector has additional motion capabilities that we embed in the *FingFlex* and *TipFlex* primitives. Each one takes only an actuator of the finger, the former selects the first actuator met in the finger kinematic chain (from the palm to the fingertip), the latter the last one. When operated, these actuators usually flex the entire finger and the last phalange, respectively. We extract these *trigs* primitives because they are useful for synthesizing the *composed* grasping actions, as it is explained in Section III-B.

For the pinch-type primitives, a collision checking is performed, thanks to the *MoveIt collision checker*, which is based on Flexible Collision Library (FCL) [15]. A *PinchTight* with two fingers is possible in an end-effector, if two fingertips can move toward each other until they get in contact, permitting to *pinch* very small objects. If some fingertips pairs can move toward each other but without colliding at the end (e.g., because some structural hand constraints stop them before), we embed this characteristic in a *PinchLoose*, instead of a *PinchTight*. The reason is that this kind of movement can still be useful to pinch and grasp objects of bigger size. We also consider the *MultiplePinchTight_N* primitive grasping action, which is a *PinchTight* but performed with $N > 2$ fingertips. It is important to notice that a pinch-type primitive grasping action 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 to move towards each other. When commanding the pinch-type primitives, the $0\% - 100\%$ scaling value can be set accordingly to the dimension of the object to be pinched.

The *SingleJointMultipleTips_N* primitive has been introduced to include movements that do not fit in the above primitive grasping actions. For example, it is suitable to consider the characteristic of some heavily under-actuated end-effectors (like the qb SoftHand) to close all the fingers together with a single actuator.

The Table I summarizes the features of each primitive grasping action considered in this work, while in Fig. 2 some of them are performed by SCHUNK SVH in a *RViZ* kinematic simulation.

### B. Custom Grasping Actions

After the primitive grasping actions are extracted, the fundamental motions of the end-effector can be performed and simple objects can be grasped. The primitives grasping actions may be not enough though alone and more complex

**TABLE I** Table which recaps the characteristics of the primitive grasping actions.

| Name | Main Element(s) Involved | Description | Constraints |
|---|---|---|---|
| **Trig** | A finger | Move all finger's actuators toward their bounds | At least one actuator dedicated only to the finger |
| **TipFlex** | A finger | Move the last actuator of the finger toward its bound | At least two actuators dedicated only to the finger |
| **FingFlex** | A finger | Move the first actuator of the finger toward its bound | At least two actuators dedicated only to the finger |
| **PinchTight** | Two fingers | Collision between two fingertips | |
| **PinchLoose** | Two fingers | Movement of two fingertips towards each other but without collision | |
| **MultiPinchTight_N** | N($\geq$3) fingers | Collision between three or more fingertips | |
| **SingleJointMultipleTips_N** | An actuator | Move an actuator (that influences N($\geq$2) fingertips) toward its bound | The actuator must influence the position of N($\geq$2) fingertips |

finger's movements may be necessary. Hence, we introduced the possibility of defining three kinds of custom grasping actions: *composed*, *user-defined* and *timed*.

*Composed* grasping actions are defined as the *composition* of other grasping actions, hence the resultant actuators set-points will be the composition of all the selected inner grasping actions. For example, we can combine more primitives from the trig category to bend only particular fingers or phalanges to adapt optimally to the object to grasp, as done in the last image of Fig. 4 and in the second image of Fig. 5.

*User-defined* grasping actions are instead created from scratch, hence not built starting from other grasping actions.

*Timed* grasping actions are a collection of grasping actions executed in sequence. They can be exploited to reach a pre-grasping pose before actually grasp the object (as in Fig. 4), or to execute a sequence of sub-tasks (like grasping and triggering an electric drill, as shown in Fig. 5).

The ROS End-Effector framework provides C++ API methods and ROS services to define the custom grasping actions.

## IV. Online Component: Commanding the Grasping Actions

The grasping actions introduced in Section III, can be selected and combined during the execution of a task. ROS End-Effector is in charge to forward the request to the end-effector as actuator set-point positions. During the initialization, the framework parses the model and all the available grasping actions (primitive, composed, user-defined, or timed) for the end-effector in use. From this point on, the grasping action commands can be sent to the end-effector.

A grasping action command is composed essentially by two fields: the name of the action and the $0\% - 100\%$ scaling value, to perform partial movements of the fingers. To command a primitive grasping action (even if this is part of a composed or timed grasping action), an additional information is necessary. This is the key to recognize which particular primitive is requested among the ones with the same name. For the Trig we must specify the finger's name; for the pinch-type primitives the names of the fingers that moves toward each other; for the singleJointMultipleTips_N the name of the actuator which moves the $N$ fingers.

A grasping action command must be sent as a *message* through *ROS actions*, which are named communication channels over which nodes exchange commands and feedback about action completion.

## V. Hardware Abstraction Layer

The communication with the real and simulated robotic end-effectors is performed by the *ROS End-Effector Hardware Abstraction Layer (HAL)*.

This layer permits to hide the low level details of the end-effector, like specific hardware components, protocols, and data fields. This generalizes the way the references are sent to the robotic end-effector and makes it simple and safe to send a command despite all the possible different hardware components.

ROS End-Effector sends the actuator references to the *HAL* component, and not to the end-effector directly. It is the *HAL*, implemented specifically for the end-effector in use that is responsible for the low level details, like how to communicate with the motors. Given a new end-effector, a new *HAL* must be implemented deriving the ROS End-Effector C++ abstract class *EEHal*. This additional work is kept as simple and fast as possible for the user. The user has only to define the communication with the robot-side, while the communication toward the ROS End-Effector main node is already implemented in the *EEHal* class. In practice, only two methods are necessary, a *sense()* to receive end-effector's data, and a *move()* to send actuator commands to the robotic end-effector.

We have implemented a *DummyHAL* suitable to exploit our framework to communicate with any simulated robotic end-effectors in Gazebo, the de facto ROS simulator.

Another *HAL* is included to communicate with the real HERI II hand, which uses the EtherCAT[4] protocol. For this specific HAL, we have exploited XBot2 software middleware, the successor of the XBot framework [16].

[4]`https://www.ethercat.org/en/technology.html`

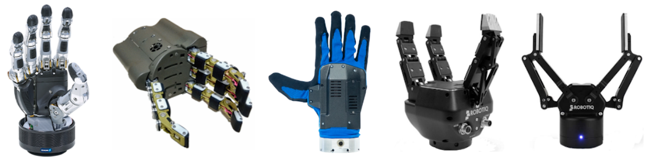

Fig. 3. All the tested end-effectors (in simulation and/or with real hardware) from left to right: SCHUNK SVH, HERI II, qb SoftHand, Robotiq 3F, Robotiq 2F-140.

## VI. Validation and testing

The framework flexibility and adaptability to different end-effectors, from simple grippers to complex human-like hands (Fig. 3), are validated both in simulation and on real end-effectors as reported in Fig. 4 and Fig. 5.

The SCHUNK SVH is a complex humanoid hand with 9 actuators and 20 DoF. The large number of possible fingers movements in this hand permitted us to validate the automatic extraction of primitive grasping actions. Some resulting hand's poses are shown in Fig. 2. Furthermore, other tests with more complex grasping actions (i.e., composed and timed grasping actions) have been conducted. We have created a particular timed grasping action called "timed_wide_grasp", with the goal of enveloping a disk. Due to the particularity of this object's shape and to the SCHUNK SVH kinematics, the fingers must be set in a particular "pre-grasping" pose, before closing the fingers around the disk. So, we can not perform a unique action but we need to execute the action sequentially. The first one, "Fing_Spread", spreads the index, the middle and the little fingers. The second one, "Opposition", moves the thumb and the right part of the hand (which is attached to the ring and the little fingers) towards each other. These are two *SingleJointMultipleTips_3* primitives: each one has one dedicated actuator linked to three fingers. The last inner action, "TipFlexes", is a composed grasping action. For the particular object's geometry, we want the thumb to close (i.e., performing a *Trig*), and the fingertips of index and middle fingers to flex a bit (i.e., performing two *TipFlex*). Therefore we have composed these three primitives to create this last inner action. The execution of this timed grasping action is visible in Fig. 4.

Simpler end-effectors, like Robotiq 2F-140, Robotiq 3F, and qb SoftHand have been tested, too. The first robot is a 2-pad gripper with a single actuator to close the pads. The second has three fingers and two actuated movements: a spreading of the two adjacent fingers and a grasping which closes all the fingers. The third has an humanoid hand shape but with a single actuator that powers all five fingers. As for the SCHUNK SVH, primitive grasping actions have been extracted, custom grasping actions defined, and all of them tested. This time the dynamic simulation (done with Gazebo) was available.

Experiments with real hardware have been conducted with the HERI II hand, configured with three fingers and a thumb opposed to them. Each finger is moved by a dedicated actuator, through a cable-driven tendon linked to all its phalanges. As shown in the sequences of Fig. 5, we have created a timed grasping action, "drill" to grasp and to trig the switch of an electric drill. We have defined it inserting three inner grasping actions. The first closes two long fingers and the thumb, to grasp the drill's handle but maintaining the trigger button free. The second performs a *Trig* with the remaining long finger to push the trigger, activating the drill's rotation. The last is the same *Trig* as before but with an intensity value of zero: this causes the finger to come back to its natural position, releasing the trigger button.

In all the above described experiments, the ROS End-Effector framework was able to automatically extract the primitive grasping actions for the end-effector in use (either very complex or a simple gripper) and allowed the user to successfully define custom grasping actions based on the primitives ones, simply using the high-level API provided by the framework. Both the simulation (either only kinematic with RViZ or also dynamic with Gazebo) and real hardware cases were validated and the user was able to command the different end-effectors in a fully agnostic fashion with respect to the hardware in use: in fact, the requested grasping tasks were carried out just commanding a set of grasping actions without specifying any motor position or current reference for the motors of the end-effector. This proves the portability and the ease of integration and usage on new robotic end-effectors of the proposed framework: the grasping tasks executed during the validation of the ROS End-Effector were successful and permitted to obtain a stable grasp for different kinds of objects, using different kinds of end-effectors.

## VII. Conclusion and Future Works

This work presented ROS End-Effector, an open-source software framework which permits to control robotic end-effectors in an agnostic fashion by means of grasping actions, effectively abstracting the low level hardware details of the robotic hand in use.

The software integration of a new end-effector can be facilitated by an offline automatic component which extracts the *primitive grasping actions* from the end-effector configuration files. Thanks to these primitives, it is possible to control the end-effector immediately both in simulation or in the real hardware. Moreover, for complex end-effector and objects, additional custom grasping actions can be defined: *composed grasping actions* can be easily defined *summing* the primitives extracted and other previously defined actions; *user-defined grasping actions*, can be constructed from scratch. Moreover, thanks to the so-called *timed grasping actions*, an ordered sequence of grasping actions can be executed with a certain timing between each other.

Once a grasping action is defined, it can be sent to the ROS End-Effector framework, which will "translate" it to low level commands for the robotic hand in use, by means of the hardware abstraction layer.

The hardware abstraction layer interface is provided to help the user to integrate a new end-effector into the ROS End-Effector framework, keeping its effort at minimum.

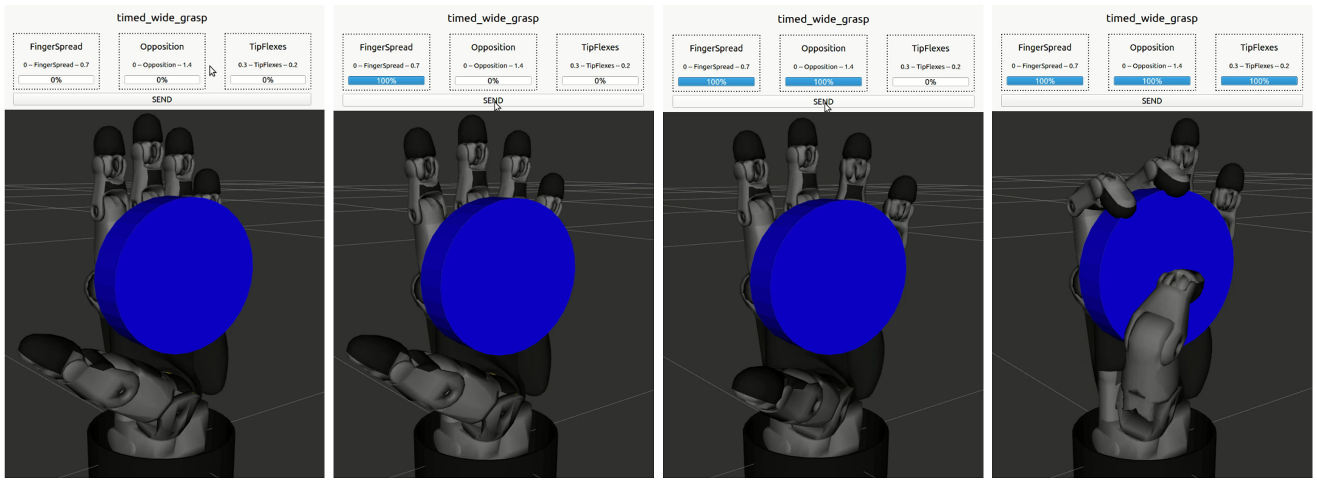

Fig. 4. A timed grasping action, "timed_wide_grasp" executed by SCHUNK SVH. The executions of its inner grasping actions ("FingerSpread", "Opposition", and "TipFlexes") are enlightened with the blue loading bars. The numbers above the blue loading bar indicate the seconds to wait before and after an inner grasping action.

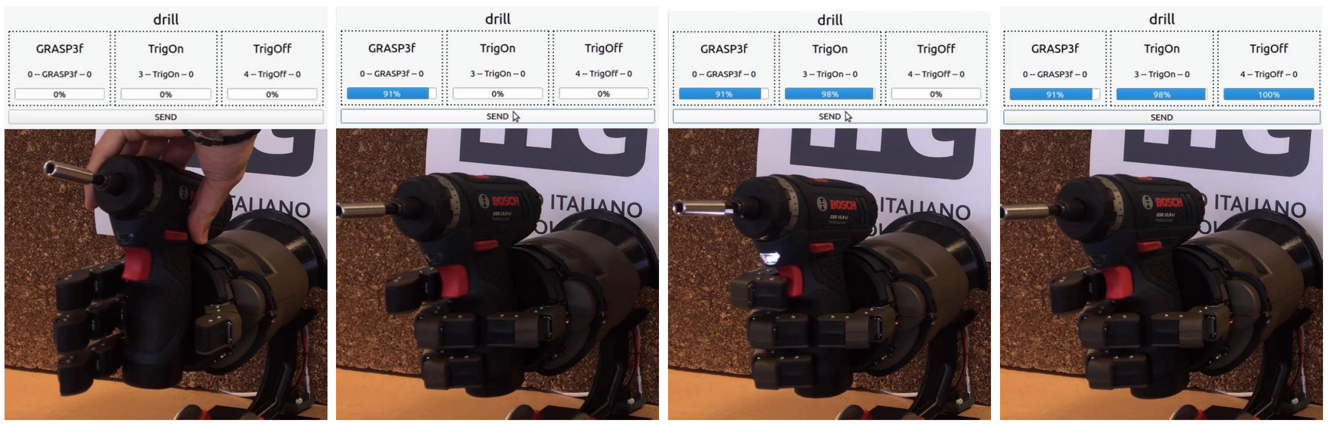

Fig. 5. A timed grasping action, "drill" executed by HERI II.

In any case, the framework is ready to be used with any simulated robotic end-effector thanks to the available *DummyHAL*.

The framework flexibility and adaptability have been tested in simulation with various robotic hands, from complex humanoid hands (SCHUNK SVH, kinematic only visualization), to simpler grippers (Robotiq 2F-140, Robotiq 3F, qb SoftHand). Tests with real hardware have been conducted with HERI II. The C++ code of ROS End-Effector is available open-source at `https://github.com/ADVRHumanoids/ROSEndEffector` with the Apache-2.0 License. In a parallel work [17] we have developed methods for the so-called *grasp synthesis*. Given an object and an end-effector, the task is to find *how to grasp it* and to integrate this into the ROS End-Effector framework. This will permit to define automatically the suitable set of grasping actions which will consider both the object shape and the end-effector used to grasp the object.

## ACKNOWLEDGMENTS

This work was supported by the European Union's Horizon 2020 research and innovation programme [grant numbers 732287 (ROS-Industrial) and 101016007 (CONCERT)] and the Italian Fondo per la Crescita Sostenibile - Sportello "Fabbrica intelligente", PON I&C 2014 - 2020, project number F/190042/01-03/X44 RELAX. The authors want to thank 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 software architecture.

## REFERENCES

[1] C. Piazza, G. Grioli, M. Catalano, and A. Bicchi, "A Century of Robotic Hands," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 1–32, 2019.
[2] M. Santello, M. Flanders, and J. Soechting, "Postural hand synergies for tool use," *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 18, pp. 10 105–10 115, 1998.
[3] E. Bizzi and V. C. Cheung, "The neural origin of muscle synergies," *Frontiers in Computational Neuroscience*, vol. 7, p. 51, 2013.

[4] M. Ciocarlie, C. Goldfeder, and P. Allen, "Dimensionality reduction for hand-independent dexterous robotic grasping," *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 20, pp. 3270–3275, 2007.

[5] M. T. Ciocarlie and P. K. Allen, "Hand posture subspaces for dexterous robotic grasping," *International Journal of Robotics Research*, vol. 28, pp. 851–867, 2009.

[6] A. Peer, B. Stanczyk, and M. Buss, "Haptic telemanipulation with dissimilar kinematics," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[7] M. Santello, M. Bianchi, M. Gabiccini, E. Ricciardi, G. Salvietti, D. Prattichizzo, M. Ernst, A. Moscatelli, H. Jörntell, A. M. Kappers, K. Kyriakopoulos, A. Albu-Schäffer, C. Castellini, and A. Bicchi, "Hand synergies: Integration of robotics and neuroscience for understanding the control of biological and artificial hands," *Physics of Life Reviews*, vol. 17, pp. 1–23, 2016.

[8] J. D. Morrow and P. K. Khosla, "Manipulation task primitives for composing robot skills," in *Proceedings of International Conference on Robotics and Automation*, vol. 4, 1997, pp. 3354–3359.

[9] T. Kröger, B. Finkemeyer, and F. M. Wahl, "Manipulation primitives — a universal interface between sensor-based motion control and robot programming," in *Robotic Systems for Handling and Assembly*, 2011, pp. 293–313.

[10] J. Felip, J. Laaksonen, A. Morales, and V. Kyrki, "Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks," *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 283–296, 2013.

[11] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *IEEE Robotics and Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[12] M. Malvezzi, G. Gioioso, G. Salvietti, and D. Prattichizzo, "SynGrasp: A MATLAB toolbox for underactuated and compliant hands," *IEEE Robotics and Automation Magazine*, vol. 22, no. 4, pp. 52–68, 2015.

[13] S. W. Ruehl, C. Parlitz, G. Heppner, A. Hermann, A. Roennau, and R. Dillmann, "Experimental evaluation of the schunk 5-Finger gripping hand for grasping tasks," *2014 IEEE International Conference on Robotics and Biomimetics*, pp. 2465–2470, 2014.

[14] Z. Ren, N. Kashiri, C. Zhou, and N. G. Tsagarakis, "HERI II: A Robust and Flexible Robotic Hand based on Modular Finger design and Under Actuation Principles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 1449–1455.

[15] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3859–3866.

[16] L. Muratore, A. Laurenzi, E. Mingo Hoffman, and N. G. Tsagarakis, "The XBot real-time software framework for robotics: From the developer to the user perspective," *IEEE Robotics and Automation Magazine*, vol. 27, no. 3, pp. 133–143, 2020.

[17] L. Bertoni, D. Torielli, Y. Zhang, N. G. Tsagarakis, and L. Muratore, "Towards a generic grasp planning pipeline using end-effector specific primitive grasping actions," *2021 IEEE International Conference on Advanced Robotics*, 2021.