CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework

Arturo Laurenzi, Enrico Mingo Hoffman, Luca Muratore, Nikos Tsagarakis

▶ **To cite this version:**

Arturo Laurenzi, Enrico Mingo Hoffman, Luca Muratore, Nikos Tsagarakis. CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework. IEEE International Conference on Robotics and Automation 2019, May 2019, Montreal, Canada. hal-02017773

HAL Id: hal-02017773
https://hal.science/hal-02017773

Submitted on 13 Feb 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework

Arturo Laurenzi[1,2], Enrico Mingo Hoffman[1], Luca Muratore[1,3], and Nikos G. Tsagarakis[1]

***Abstract*— This work introduces a framework for the Cartesian control of multi-legged, highly redundant robots. The proposed framework allows the untrained user to perform complex motion tasks with robotics platforms by leveraging a simple, auto-generated *ROS*-based interface. Contrary to other motion control frameworks (e.g. ROS *MoveIt!*), we focus on the execution of Cartesian trajectories that are specified *online*, rather than planned in advance, as it is the case, for instance, in tele-operation and locomotion tasks. Moreover, we address the problem of generating such motions within a *hard real-time (RT)* control loop. Finally, we demonstrate the capabilities of our framework both on the *COMAN+* humanoid robot, and on the hybrid wheeled-legged quadruped *CENTAURO*.**

I. Introduction

In the past few decades, advancements in robotics have vastly extended its domain of application, moving from the traditional industrial focus to eventually applying robots to human and more unstructured environments. Consequently, robotic platforms have grown in complexity, in an attempt to satisfy the requirements that the new applications demand in terms of hardware, motion generation, control and human machine interfaces. Indeed, operating effectively in complex environments requires enhanced mobility in order to move on rough surfaces, overcome obstacles, and to carry out complex manipulation tasks that take place in an extended workspace, ranging from the ground-level to above the eye-level.

Such robots exhibit a high degree of redundancy, which however complicates their control from several perspectives: first, tasks have usually *different priorities*; then, when infinitely many solutions exist, some criterion is needed to select one. Finally, the complexity of the code required to control the robot grows considerably: a simple-yet-effective scheme for solving the manipulator inverse kinematics may be realized with few lines of code, whereas the control of highly redundant robots is best done inside a suitable *framework* that hides most of the involved complexity in the motion coordination of redundant robots.

An aspect that is worth highlighting is the dichotomy between tasks that are *pre-planned* offline, and tasks that are specified *online*, in a continuous fashion. As an example from the first family, the reader could consider a reach-to-grasp task taking place in a cluttered environment; such an action can be completely pre-planned before the actual execution, thus opening the possibility of performing a time-consuming search over the robot configuration space in order to find obstacle-free trajectories. Conversely, examples from the second category include tele-operation scenarios, reactive locomotion and physical interaction, and others. Online tasks greatly differ from their pre-planned counterparts, since they require *low computation times* and, especially when any kind of feedback is involved, small delay and jitter in the execution.

Existing works have recognized the need for a Cartesian control framework; yet, it is hard to find a solution that satisfies the following requirements:

- flexible task specification, in terms of both type and number of tasks;
- ability to enforce *soft* priorities as well as *hard* priorities between tasks;
- ability to specify constraints in the task execution;
- small computation time (suitable for online execution);
- possibility to execute inside a *real-time (RT)* thread in order to reduce delays and jitter;
- ease of configuration and use, quick setup time and ready to use control tools;
- parametrized with standard description formats (e.g. *URDF*) in order to support multiple platforms;
- handling of floating base robots.

Previous work from the authors [1], [2] goes in the direction outlined by the aforementioned points, resulting in the C++ library *OpenSoT*, which provides tools for writing Cartesian solvers while taking into account priorities and constraints, in a real-time safe way. In the present article, we extend the framework with additional layers that permit to perform Cartesian control "out of the box": we relieve the user from the need of writing and compiling C++ code that is tailored to a specific robotic platform and task, and we provide an auto-generated uniform interface to send commands to Cartesian controllers inside the popular *ROS* framework. Notably, the same ROS API is provided even when Cartesian controllers run inside a hard real time thread.

We organize the rest of the article as follows: in Section II, we review some of the related works; then, in Section III the overall architecture of the proposed framework is described. Section IV is dedicated to the description of two experimental setups that validate the effectiveness of the framework. Finally, possible future directions are depicted in Section V.

II. Related works

The presented framework draws inspiration from previous work on the same topic; in the remainder of this section

[1]Advanced Robotics Department (ADVR), Istituto Italiano di Tecnologia, Genova, Italy

[2]DIBRIS, Università di Genova, Italy

[3]School of Electrical and Electronic Engineering, The University of Manchester, M13 9PL, UK

{arturo.laurenzi, enrico.mingo, luca.muratore, nikos.tsagarakis}@iit.it

	Environment-aware planning capabilities	Real time capable	Generic robot support	Flexible task specification	Floating-base support
OpenRAVE [3]	✓	✓	✓	✗	✗
MoveIt! [4]	✓	✗	✓	✗	✗
Chorenoid [5]	✗	✓	✗	✗	✓
CartesI/O	✗	✓	✓	✓	✓

TABLE I

COMPARISON BETWEEN DIFFERENT FRAMEWOKS FOR ROBOT CARTESIAN CONTROL.

we present a summary of the most prominent contributions in the field, highlighting the shortcomings that we aim to address with our framework. A summary of our comparison is outlined in Table I.

OpenRAVE [3] is one of the first environments for testing, developing, and deploying motion planning algorithms in real-world robotics applications. It has been used with many different types of robots such as humanoids, mobile platforms and manipulators. From the Cartesian control point of view, OpenRAVE relies on *ikfast* which is an analytical IK engine; it is one of the fastest IK engines available at the moment, however it does not handle redundancy (extra DoFs have to be *locked* manually in order to match the required kinematic structure).

The most widely-spread framework for Cartesian control is probably *MoveIt!* [4], which is part of the open source *ROS* framework. MoveIt! provides a rather complete system of motion planning and execution tools that also take into account the perceived environment. The main core of MoveIt! is based on state-of-the-art sampling-based motion planning algorithms, mostly deployed inside the *OMPL* project [6]. Despite MoveIt! has been used on many different types of robot, it lacks explicit support for multi-chains and floating-base (e.g. legged) robots. Furthermore its non-linear planning nature does not make it suitable for on-line (tele-operation) or real-time control.

Another notable work is the software suite *Chorenoid* [5], which permits to synthesize whole body motions for bipedal humanoid robots. The user can control different end-effectors as well as perform dynamic motions which are filtered through a *Zero-Moment-Point (ZMP)* based stabilizer. Such a tool has been used successfully on many HRP-series robots, yet it does not permit to setup customized IK problems. This indeed limits the usability of Chorenoid to manipulators and bipeds.

A lot of effort was spent on creating GUI systems for the *DARPA Robotics Challenge (DRC)* (e.g. [7], [8], [9]). Yet, none of these works was designed to be general enough to handle different types of robots with considerably diverse structure. Moreover, most of these GUI systems focus on predefined tasks (usually arms end-effectors, center-of-mass, and direct joint-space control) without any flexibility in terms of tasks or constraints declaration.

III. ARCHITECTURE AND IMPLEMENTATION

The purpose of this section is to present details about the framework design and structure. First and foremost, we highlight which are the main building blocks of the framework; then, we briefly introduce our previous work [1], the *OpenSoT* library. Finally, from Section III-C to Section III-E we present the core components of *CartesI/O*, that is the main focus of this work.

A. Framework components

Following a bottom-up description, we can distinguish:

- the *solver*; this is the component that solves a single instance of the mathematical problem that describes our Cartesian control algorithm. In our case, this could be either a *matrix pseudo-inverse* solver, or a *Quadratic Program (QP)* solver. Other possibilities exist, like more general *Non-Linear Program (NLP)* solvers.
- A *modeling language* that allows to construct the aforementioned mathematical problems in a natural and more high-level way, which is less error-prone and time consuming.
- A *base class* for Cartesian controllers, that allows for uniform programmatic usage of any specific implementation. It also takes care of set-point management, as for instance enforcing velocity/acceleration limits, or transforming waypoints into properly interpolated references.
- A *middleware interface*, which enables all other processes that compose the control system to send their references in a uniform way that does not depend on the specific implementation running.

This work is mostly concerned with the last two parts, and it is completely decoupled from any specific choice of a solver and modeling language. However, integration with the *OpenSoT* library, which implements a modeling language tailored to Cartesian control, also represents a major goal; we briefly describe *OpenSoT* in the following section.

B. The OpenSoT library

The *OpenSoT* library is a C++ framework which has been initially developed to participate to the DARPA Robotics Challenge [10] as a whole-body inverse kinematics engine. During the years, inverse dynamics and force optimization have also been integrated [1], [2] for different robotic platforms. The focus of *OpenSoT* is to ease the formalization of prioritized controllers through dedicated interfaces for *tasks*,

constraints and *solvers*. Each of these entities are atomic elements which can be combined using a simple syntax, named *Math Of Task*. An example of a simple controller described using the *Math Of Task* is the following one:

$$\begin{pmatrix} \left({}^{\text{Waist}}\mathscr{T}_{\text{RWrist}} + {}^{\text{Waist}}\mathscr{T}_{\text{LWrist}}\right) / \\ \mathscr{T}_{\text{Posture}} \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Joint}\\\text{Limits}}} + \mathscr{C}_{\substack{\text{Joint Velocity}\\\text{Limits}}}\right). \tag{1}$$

Two *hard* priorities (slash "/" operator) are specified: the first one is constituted by two tasks (in relative *soft* priority, plus "+" operator) which control the arms end-effectors w.r.t. the *Waist* frame, and the second one is a *Joint Postural* task. All priority levels are subject to *Joint Limits* and *Joint Velocity Limits* constraints ("<<" operator).

The control problem (1) is then solved in two steps: first, a *solver front-end* computes matrices and vectors that describe the QP problems for all priority levels, according to the following formulation:

$$\begin{aligned} \min_{\boldsymbol{x}_i} \ & \|A_i \boldsymbol{x}_i - b_i\|^2 + \varepsilon \|\boldsymbol{x}_i\|^2 \\ \text{s.t. } & b_{\text{l}} \leq D\boldsymbol{x}_i \leq b_{\text{u}} \\ & u_{\text{l}} \leq \boldsymbol{x}_i \leq u_{\text{u}} \\ & A_{i-1}\boldsymbol{x}^*_{i-1} = A_{i-1}\boldsymbol{x}_i \\ & \vdots \\ & A_0 \boldsymbol{x}^*_0 = A_0 \boldsymbol{x}_i. \end{aligned} \tag{2}$$

Then, a *solver back-end*[1] actually computes the corresponding solutions. Notice that the *Math Of Task* formulation, as well as the chosen *solver*, are completely decoupled from the type of controller (velocity or acceleration-level IK, torque control, ...) which depends only on the specific *tasks* and *constraints* implementations.

C. Cartesian Interface

A modeling language eases the job of formulating and solving a complex optimization problem; yet, the need to write C++ code that is customized for the specific robotic platform and task to be solved remains. This should be avoided, both with a view to promote code reuse, and also considering that, for complex platforms and according to the authors' experience, *writing a hierarchy of tasks/constraints that makes the robot show the desired behavior can be "an art"*. Therefore, it should be left to "experts in the field", while users should simply customize the problem to better fit their needs. Furthermore, this code would be of little use without an I/O infrastructure that allows a control module to communicate with the external world. Again, the user should be relieved from developing its own. With this motivation, we started developing an interface layer that:

- provides a uniform way to programmatically interact with a Cartesian controller;
- automatically generates a complete ROS API for sending references to the controller;
- allows to use the ROS-based API also in the case that the *solver* is running inside a real-time thread.

[1]At this moment, back-ends are available for the *qpOASES* [11] and *OSQP* [12] solvers.

We call the base class that specifies such an interface *CartesianInterface*. It defines simple methods that can be used to change the behavior of all defined tasks, as for instance:

- a *setPoseReference()* method sets the Cartesian set-point corresponding to a task, given its controlled link name;
- the corresponding *setWayPoints()* method is used to specify a point-to-point motion passing through custom waypoints.
- A *setBaseLink()* method can be used to change online the base-link of a task.
- A *setControlMode()* method is used for selecting whether a specific task should be position-controlled, velocity-controlled, or disabled at all. Tasks that are running in velocity mode will discard any position reference, while tasks that are disabled will disappear completely from the Cartesian control problem.
- An *update()* method is used to evaluate all point-to-point trajectories given the current time, and to enforce velocity/acceleration limits by means of the *Reflexxes* library [13]. This method is overridden by the specific subclasses in order to implement their own control loop.

The developer willing to implement its own Cartesian controller can override all these methods in order to take appropriate actions whenever a reference, a control mode or a base-link has changed.

To store the state of the robot, and to perform kinematic/dynamic computation as well, we make use of the *XBot::ModelInterface* class from the *XBotCore* framework [14], [15], which acts as a wrapper for rigid body dynamics libraries. Specific implementations of it[2] can be chosen at runtime, using a dynamic loading mechanism.

D. Auto-generated ROS API

Once that a Cartesian controller has been implemented, a communication layer towards external modules is needed, which is our middleware interface. This is implemented in the form of a C++ class that we call *ROS Server Class*; for each of the defined tasks, it provides the following functionalities:

- a TF publisher for the model state;
- point-to-point motions with custom waypoints through a custom ROS action;
- commanding continuous pose and velocity references by publishing to ROS topics;
- run-time activation/deactivation of tasks, as well as change of control mode and base-link by calling custom ROS services;
- Rviz *interactive markers* manager [18] that allows for intuitive, GUI-based reference generation;
- *joystick*-based reference generation;
- *joint sliders*, which permit to work at joint space level.

[2] An RBDL-based [16] version is available at `https://github.com/ADVRHumanoids/ModelInterfaceRBDL`, a version based on iDynTree [17] is available at `https://gitlab.advrcloud.iit.it/advr_humanoids/modelinterfaceidynutils`

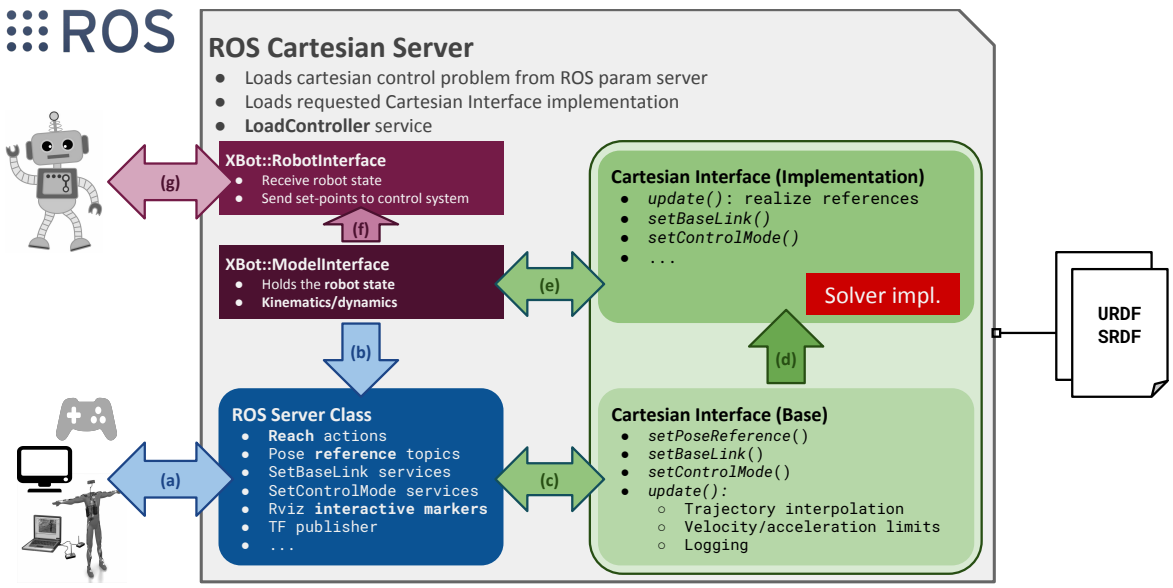

Fig. 1. Components of the Cartesian Interface and signal flow when the Cartesian Interface is executed inside a ROS node. External processes exchange information with the CartesI/O framework via ROS topics (a), thanks to the *ROS Server Class* component. It gets the current solver state through the *ModelInterface* object (b), in order to broadcast it to the ROS environment. It also forwards the received commands to the *Cartesian Interface base class* (c), where trajectory interpolation and filtering take place. The *Cartesian Interface implementation* component gets these pre-processed signals in order to track them (d). This results in an updated model state (e), which is then sent to the robot actuators through the *RobotInterface* object (f, g).

Furthermore, a ROS node that acts as a *Cartesian Server* is provided as well. It loads a user-specified implementation of the *CartesianInterface* class using a dynamic loading mechanism, and initialize the *ROS Server Class* in order to generate the ROS API. Finally, a ROS service is provided to dynamically change the controller that is under execution during runtime. The resulting architecture is shown in Figure 1: note that, in order to send actual references to the robot, we employ the *XBot::RobotInterface* class from the XBotCore framework, which serves as the robot abstraction layer[3].

E. Configurable OpenSoT implementation

As the final step towards a complete Cartesian control framework as described in Section III-A, we provide a generic implementation of the *CartesianInterface* that relies on *OpenSoT* as the modeling language, and on its supported solvers for carrying out the actual optimization procedure. Such a module allows the user to formulate a *hierarchical, whole-body inverse kinematics* problem at the velocity level (as described in Section III-B); it is written so as to be completely configurable, meaning that the stack of tasks can be specified either on a YAML file or via the ROS parameter server. In this way, starting from a standard description of the robot in terms of its URDF/SRDF, the user can directly run a ROS Cartesian Server to perform rather complex whole-body control tasks with no code compilation involved; moreover, the auto-generated ROS API provides a convenient way to interact with such a controller from all processes that compose a distributed control system.

F. Real-time execution

Whenever a Cartesian controller is based on continuous feedback from the robot, its precise and jitter-free execution at the specified control frequency becomes critical; indeed, delays contribute to destabilize feedback loops, and should therefore be avoided. Typical examples are torque-based controllers (e.g. Cartesian impedance control); however, it is worth noticing that position-based controllers can involve feedback, as in the case of admittance controllers, *stabilizers* for legged robots (e.g. [19]), and tele-manipulation with force feedback.

Such characteristics can be achieved by calling the CartesianInterface's *update()* method from within a *real-time (RT)* thread that runs inside a suitable *real-time operating system (RTOS)*; in addition, the communication between this thread and the robot control PC needs to be fast introducing minimum latency. In order to do so, we need to ensure *real-time safeness* of all components of our architecture, which broadly speaking means that *all non-deterministic operations should be avoided*, most notably memory allocations and, e.g., network communication.

By careful code development and profiling, we ensure satisfaction of these constraints both by the *OpenSoT* library (including its solver back-ends), and by the *CartesianInterface* layer. However, it is not possible to run our *ROS Server Class* on the RT layer directly, mainly because of ROS's usage of TCP primitives[4]. To address this issue, a *dual-thread*

[3] A ROS-based version that eventually publishes on a joint state topic is available at `https://github.com/ADVRHumanoids/RobotInterfaceROS`

[4] TCP usage from a real time thread will eventually will be possible in ROS2, which is based on the DDS middleware.

architecture is needed, where a *non-RT thread* runs our ROS API server, while a *RT thread* runs the *CartesianInterface* implementation. We put this idea into practice by leveraging the *XBotCore* framework, that provides us both a RT "control thread" and a non-RT "communication thread". We design the synchronization between the two to be *lock-free* for the RT thread, in order to avoid priority inversion problems. This results in a deterministic execution time for our controller, as we experimentally demonstrate in Section IV-A.

IV. Experimental results

To validate our *CartesI/O* framework and demonstrate its flexibility in different robot platforms, we set up two manipulation tasks to be carried out by our legged robots *CENTAURO* [20], a 39-DoF wheeled-legged quadruped with a humanoid torso, and the 28-DoF humanoid *COMAN+*. In both cases, we select a box-picking task where the box must be picked from a low height. In such a case, not only the arms but all the robot chains must coordinate to accomplish the task, which highlights the advantage of using a floating-base whole-body formulation. The outcome of our experiments is summarized in the accompanying video; an extended version can be found at `https://youtu.be/eVmDBVL83WY`.

A. Case study: stabilized box-picking task

As our first experiment, we present an application of our framework to a scenario where the humanoid *COMAN+* has to pick a 3 Kg box and pass it to a human. We define the task such that the robot must reject external disturbances without falling, and it must also show compliance in the reaction in order not to hurt people around it. To achieve compliant rejection of external forces, we use the work of [19] where a compliant admittance-base stabilizer is introduced, which essentially computes a *modified CoM reference* from a reference on the *center of pressure (CoP)* (which we kept fixed during the experiment) and from force-torque measurements at the feet as well. Since this experiment involves a feedback controller, we run our Cartesian solver plus the stabilizer from within the *XBotCore* RT control thread as explained in Section III-F. To execute the motions, we run from a different low-priority process a *ROS SMACH*[5]-based state machine written in Python that sends target poses to the end-effectors via our auto-generated ROS API as follows:

- the box is grasped and brought to the chest level by sending suitable references to the hands w.r.t. the world frame;
- the box is then passed to the human operator; in order to do so, we change the base link for the hands tasks from the world frame to the torso frame, and finally we command it to rotate about the z-axis.

During the whole demo, the compliant stabilizer is continuously adjusting the CoM reference to track the desired CoP.

Time statistics regarding the experiment are shown in Figure 2; it can be noticed that our control thread does indeed meet its deadline in a deterministic way, with a root-mean-square deviation of $T_{\text{jitter}} \approx 90\mu s$, over more than one minute of experiment. Snapshots from the experiment are visible in Figure 3.

[5] `http://wiki.ros.org/smach`

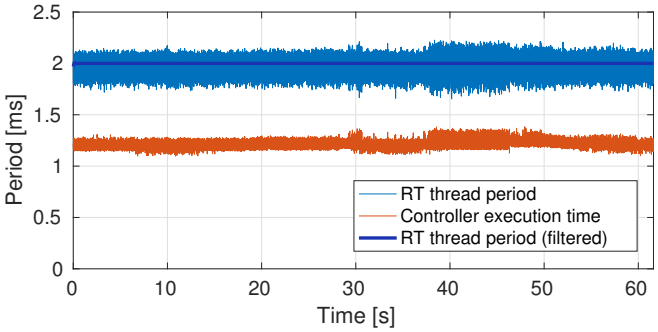

Fig. 2. Timing statistics for the box picking experiment with *COMAN+*. The thin blue line represents the XBotCore RT thread period. The cartesian controller computation time (including the stabilizer) is depicted in red.

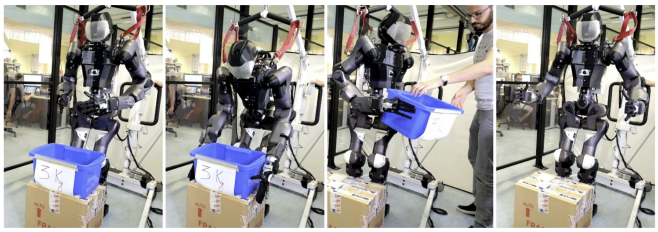

Fig. 3. Snapshots from a box-picking task with RT stabilization using humanoid COMAN+.

B. Case study: ground-level bimanual manipulation

For our second experiment, we choose to pick up a 6 kg brick from the ground using the CENTAURO robot, and then pass it to a human operator. Since the operator is standing on one side of the robot, the robot must perform a brief wheeled-locomotion phase in order to turn ninety-degrees. This is done by switching at runtime between two different implementation of the *Cartesian Interface*: the first, which is used to pick up the box, is the dynamically-configurable IK controller described in Section III-E; the second one, which we call *Centauro Wheeled Motion*, is an IK controller that is tailored to the mixed wheeled-legged locomotion of the *CENTAURO* robot. It is implemented by means of the *OpenSoT* library, and it permits to:

- control the waist pose w.r.t. the world through appropriate steering/rolling of the wheels;
- control the wheels position w.r.t. the waist frame in order to adjust the support polygon shape;
- perform basic control of the end-effectors w.r.t. waist frame.

Since this demonstration does not require to meed hard deadlines, we run the Cartesian controllers on the *ROS Cartesian Server* at a frequency of 100 Hz. Again we script the robot behavior via a *ROS SMACH* state machine as follows:

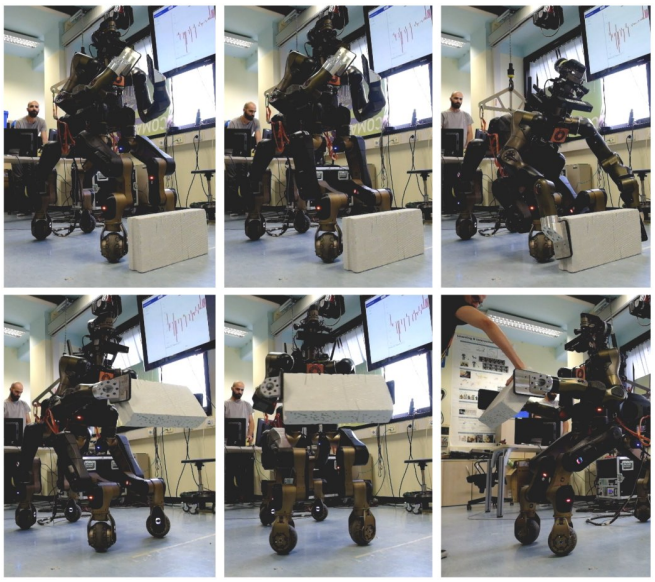

Fig. 4. Snapshots from the experiment with CENTAURO robot described in Section IV-B.

- first, we change the postural for the front knees in order to avoid later collisions with the arms;
- then we command the end-effector to surround the box, then grasp it and bring it up;
- after this, we load our *Centauro Wheeled Motion*, and we command a ninety-degrees rotation to the waist;
- finally, we command the arms to open in order to release the box.

The outcome of this experiment is summarized in Figure 4, and in the accompanying video as well.

V. Conclusions

In this work we have introduced *CartesI/O*, a framework for Cartesian control of floating/fixed-base robots which is focused on *online* execution, possibly under real-time constraints. As our main contribution, we integrate Cartesian solvers inside an architecture that permit to interact with them in a uniform way. This is achieved at two different levels:

- programmatically, through our *CartesianInterface* base class;
- from the ROS middleware, via an auto-generated set of topics, services, actions and tools that can be used to monitor the solver state, send references, and customize the solver behavior.

Our API aims at providing highly flexible task customization during runtime: the user can activate/deactivate tasks, change their base-link, switch between position and velocity control, and even dynamically load different controllers. Furthermore, the framework integrates common tools, such as trajectory interpolation with way-points, as well as enforcement of user-specified velocity/acceleration limits.

A configurable inverse-kinematics solver is also provided, through which the user can quickly set up a Cartesian control problem from a YAML file, which is then solved using the *OpenSoT* library. Such a controller is designed to be real-time safe, allowing for the deterministic execution of the commanded references. Moreover, we provide tools to connect a RT Cartesian controller to our auto-generated ROS API, in order to enable mixed RT/non-RT robot control.

Future work will address the development of solvers working at the dynamics level, as well as the augmentation of the markers capabilities (e.g. Cartesian impedance markers). Moreover, we aim to add previewing capabilities, and environment-aware trajectory planning, e.g. via integration with *MoveIt!*. Finally, we aim at integrating this work with well-known RT frameworks such as *OROCOS* and *ROS2*.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No. 644839 (CENTAURO) and No. 644727 (CogIMon).

References

[1] E. M. Hoffman, A. Rocchi, A. Laurenzi, and N. G. Tsagarakis, "Robot control for dummies: Insights and examples using opensot," in *17th IEEE-RAS International Conference on Humanoid Robotics, 2017*, pp. 736–741, 2017.

[2] E. M. Hoffman, A. Laurenzi, L. Muratore, D. G. Caldwell, and N. G. Tsagarakis, "Multi-priority cartesian impedance control based on quadratic programming optimization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[3] R. Diankov, *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[4] S. Chitta, I. Sucan, and S. Cousins, "Moveit!," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[5] S. Nakaoka, S. Kajita, and K. Yokoi, "Intuitive and flexible user interface for creating whole body motions of biped humanoid robots," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 1675–1682, IEEE, 2010.

[6] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. `http://ompl.kavrakilab.org`.

[7] S. Caron and Y. Nakamura, "Teleoperation system design of valve turning motions in degraded communication conditions," *The 33-rd Annual Conference of the RSJ*, 2015.

[8] M. Schwarz, T. Rodehutskors, *et al.*, "Nimbro rescue: Solving disaster-response tasks with the mobile manipulation robot momaro," *Journal of Field Robotics*, vol. 34, no. 2, pp. 400–425, 2016.

[9] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, *et al.*, "An architecture for online affordance-based perception and whole-body planning," *Journal of Field Robotics*, vol. 32, no. 2, pp. 229–254, 2015.

[10] N. G. Tsagarakis, D. G. Caldwell, *et al.*, "Walk-man: A high-performance humanoid platform for realistic environments," *Journal of Field Robotics*, vol. 34, no. 7, pp. 1225–1259, 2017.

[11] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "Qpoases: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, pp. 327–363, Dec 2014.

[12] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *ArXiv e-prints*, Nov. 2017.

[13] T. Kröger, "Opening the door to new sensor-based robot applications—the reflexxes motion libraries," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.

[14] L. Muratore, A. Laurenzi, E. M. Hoffman, A. Rocchi, D. G. Caldwell, and N. G. Tsagarakis, "Xbotcore: A real-time cross-robot software platform," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, pp. 77–80, April 2017.

[15] "XBotCore GitHub repository." https://github.com/ADVRHumanoids/XBotCore/wiki/XBotInterface.
[16] M. L. Felis, "Rbdl: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, pp. 1–17, 2016.
[17] F. Nori, S. Traversaro, J. Eljaik, F. Romano, A. Del Prete, and D. Pucci, "icub whole-body control through force regulation on rigid noncoplanar contacts," *Frontiers in Robotics and AI*, vol. 2, no. 6, 2015.
[18] D. Gossow, A. Leeper, D. Hershberger, and M. T. Ciocarlie, "Interactive markers: 3-d user interfaces for ros applications [ros topics]," *IEEE Robot. Automat. Mag.*, vol. 18, pp. 14–15, 2011.
[19] C. Zhou, Z. Li, X. Wang, N. Tsagarakis, and D. Caldwell, "Stabilization of bipedal walking based on compliance control," *Autonomous Robots*, vol. 40, pp. 1041–1057, Aug 2016.
[20] L. Baccelliere *et al.*, "Development of a human size and strength compliant bi-manual platform for realistic heavy manipulation tasks," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 5594–5601, IEEE, 2017.