Towards A Robot Hardware Abstraction Layer (R-HAL) Leveraging the XBot Software Framework

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

Towards A Robot Hardware Abstraction Layer (R-HAL) Leveraging the XBot Software Framework

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

***Abstract*— The rapid advances in robotics have recently led to the developments of a wide range of robotic platforms that exhibit significant differences at the hardware components level. In analogy to the case of the early computer hardware few decades ago, most of these robotics systems do not possess any form of hardware abstraction. This necessitates the users, writing code to control these platforms, to have extensive know-how of the robot hardware devices and their communication interfaces and protocols. Consequently, this poses a significant challenge to robot software developers since they have to know how every hardware device in the robot works to ensure their software's compatibility when transferring/reusing their code on different robots.**

In this paper we present a new Robot Hardware Abstraction Layer (R-HAL) that permits to seamlessly program and control any robotic platform powered by the *XBot* control software framework [1]. The implementation details of the R-HAL are introduced. The R-HAL is extensively validated through simulation trials and experiments with a wide range of dissimilar robotic platforms, among them the COMAN and WALK-MAN humanoids, the KUKA LWR and the CENTAURO upper body. The results attained demonstrate in practise the gained benefits in terms of code compatibility, reuse and portability, and finally unified application programming even for robots with significantly diverse hardware. Furthermore, it is shown that the implementation and integration of the R-HAL within the *XBot* framework does not generate additional computational overheads for the robot computational units.

I. Introduction

A robotic platform can be considered as a complex distributed system composed by a set of hardware devices communicating through different fieldbus systems and a set of interfaces and protocols of diverse instruction and data fields. The aforementioned complexity is mainly due to hardware specific operation protocols, hardware distribution and synchronization requirements [2]. As a result, frequently during software development in robotics it is very challenging to develop code, e.g. control software modules, that can be reused with no changes on different platforms.

However, efficiently developing control and application software, that can be shared, ported and reused in these diverse range of robotic hardware with minimum effort, is a fundamental requisite for advancing fast the capabilities of these machines.

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

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

{giuseppe.rigano, luca.muratore, arturo.laurenzi, enrico.mingo, nikolaos.tsagarakis}@iit.it

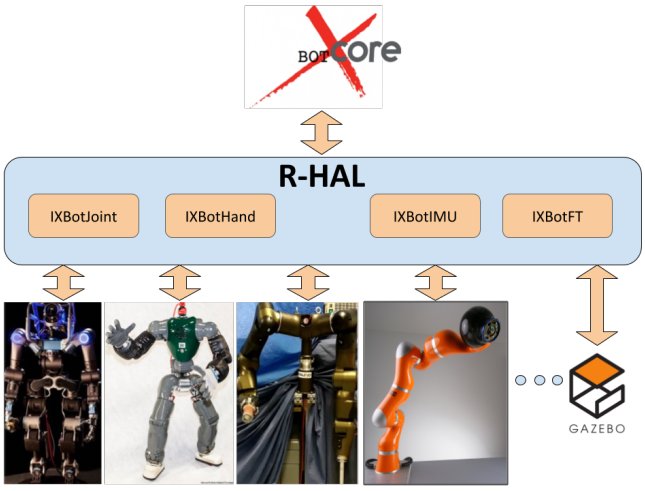

Figure 1. The robot Hardware Abstraction Layer introduced for the *XBot* software architecture. The R-HAL assures high flexibility towards any type of robotic platform or simulation environment.

An important component, needed to achieve this, is the Hardware Abstraction Layer (HAL), which can be incorporated to mask the physical robot hardware differences and limitations (e.g. kinematics model, sensor, update frequency, etc) varying from one robotic hardware to another. In such a case, an application programming interface (API) alone could do very little to hide those differences, by mostly assuming a common model, that will not be effective in most of the cases.

Thus, the robotic software architectural decisions should seriously consider the key role played by HAL for the interaction and the coordination of robot hardware and software control modules. HAL can provide a relatively uniform abstraction that could hide the specifics of the underlying hardware such that the underlying robotic hardware is transparent to the robot control software [3]. Furthermore, HAL assures portability and code-reuse: it efficiently permits robot control software developed for one robot to be ported to another. In one example [3], the HAL permits the same navigation algorithm to be ported from a wheeled robot and used on a humanoid legged robot.

In a robotic platform different fieldbus systems can be employed to permit communication among the robot hardware devices. *CAN* bus is one example. Among the others, the iCub humanoid [4] and the HyQ quadruped [5] are

exploiting this protocol to communicate with the actuators control boards. Nevertheless, the *CAN* bus limit the data throughput and have higher roundtrip(reaction) time with respect to Ethernet-based solutions [6].

Even if *Ethernet*(defined in IEEE 802.3) is non-deterministic and thus is unsuitable for hard Real-Time (RT) applications due to its random delays and potential transmission failures, it is used in several robotic platform e.g. on COMAN humanoid [7] or the recent upgrade of the iCub, called iCub2 [8]. An example of a fieldbus system with RT communication capabilities is *EtherCAT* (Ethernet Control Automation Technology), an industrial protocol built on the *Ethernet* specifications. It assures high transmission rate, minimum roundtrip time with respect to other industrial protocol, precise synchronization ($\ll 1\mu$s), and demonstrates flexible topologies, easy configuration, implementation and cost effectiveness [9]. WALK-MAN humanoid [10] robot is using the *EtherCAT* technology.

On the basis of the above mentioned considerations, the *XBot* [1] RT software platform was designed only for *EtherCAT* based robots. However to support a wider variety of robotic systems it is essential to incorporate a R-HAL interface inside the framework. In the work presented herein we will show the design choices taken for the implementation of the robot hardware abstraction layer and the capabilities of this software component, which enable us to efficiently port and run the same control software modules on different robots, both on simulation and on the real hardware platforms. The rest of the paper is organized as follows: Section II presents the related work, while section III gives a detailed description of the proposed HAL. Section IV presents the experimental results. Finally section V addresses the conclusions.

II. Related Work

In computer science, the *separation of concern* principle followed by the modern software design patterns has been a relevant topic even before the Object Oriented Programming (OOP) languages. The main goal is to enhance the reuse of the code using abstractions of real objects and hiding implementation details. Although programming by interface is a good practise to use for any type of software development, the need of abstraction layer is strong when interfacing with hardware devices because of the several communication protocols.

For the above reasons, the *Object Management Group* [11], not-for-profit technology standards consortium, last year announced the adoption of the *Hardware Abstraction Layer for Robotic Technology (HAL4RT)* [12] as an open standard for the implementation of robotics and control software systems. By standardizing on HAL4RT it will be possible to port or reuse drivers on different robotics hardware.

The approach used by some of the RT robotic frameworks follows the use of making abstraction of the low level analog and digital I/O devices and exploit the polymorphism to change the specific implementation at runtime.

The *OROCOS* [13] framework relies on the Device Interfaces [1] as a means to achieve hardware abstraction modeling the concept of low level analog/digital I/O as well as a bit upper level axis interface. However it requires more effort from the user side in order to interface an entire robot.

EEROS [14], an industry-ready open source RT robotics software, uses a similar methodology trying to wrap each specific hardware libraries using a configuration file to chose the low level I/O parameters. An interesting feature of the EEROS framework is the safety layer embedded in the HAL that allows to handle the safety behaviour of the robot in emergency situations.

A different approach is used by *PODO* [15], where the hardware abstraction is done at the level of the robot using a shared memory mechanism. Nevertheless no mention about interfacing different robot hardware has been found in their work.

The *OpenRTM-AIST* [16] RT framework has the goal to build an independent middleware in order to improve the re-usability of software on the independent platform. It uses a different approach, each hardware component is seen as an RT component with its internal state machine that handles the component's life cycle. The communication between components follows the publisher/subscriber model and it is realized using the data port.

Similar consideration to the *OROCOS* framework can be done for the NRT *YARP* [17] middleware where the user has to implement a specific driver for each device family.

ROS [18], the popular component-based NRT framework, aims to provide a really high level hardware abstraction by means of the *ROS-CONTROL* [2] packages. The robot hardware interface is designed in such a way to move the integration effort to the driver level and let the user to easily read the state of the robot and send commands.

The RT *XBotCore* platform was initially designed to provide a bland level robot hardware abstraction just for EtherCAT based robot, making really difficult interfacing with any new robot hardware. Despite the EtherCAT abstraction, *XBotCore* was highly tied to the low level with no autonomous threading capability, in the sense that it relied on the custom low level software to become part of the thread loop.

The proposed work tries to overcome the aforementioned issues making a really flexible system and improving software re-usability. To achieve that, a strategy similar to ROS has been adopted allowing to interface easily with the code already written for ROS and as a result the advantage of having a choice between *RT* (Real Time) and *NRT* (Not Real Time) control system. However it is important to note that the RT capability depends on the low level hardware and software layer.

[1] http://www.orocos.org/stable/documentation/rtt/v2.x/doc-xml/orocos-components-manual.html#idp41997920

[2] http://wiki.ros.org/ros_control

III. R-HAL

In this section we will discuss both the initial and the current design choices of the *XBotCore* platform trying to underline pros and cons by comparing the two software designs.

A. The former XBotCore design

As mentioned in the previous section, the initial design of the *XBotCore* was not ready to easily plug a new robot hardware. The Fig.3 shows the former software architecture where four main elements can be analyzed:

- XBotEcat, responsible to interface the low level software control;
- XBotCore, designed to provide the implementation of the interfaces;
- Interfaces representing Joint, IMU, FT, Hand;
- GazeboXBotPlugin, aims to emulate the XBotCore component acting as a Gazebo[3] simulator plugin.

The above described architecture allows to reuse part of the code but it is not completely optimal for a full reuse leading to a code replication inside the *GazeboXBotPlugin* and in any new concrete implementation. Moreover the UML class diagram shows the lack of any hardware abstraction layer, since only the *XBotEcat* implementation is provided as an endpoint to the low level control. The lack of multi-fieldbuses support is a strong limitation that needs to be overcome.

The described solution makes the *XBotCore* an embedded component inside the low level control software relying on the Template Pattern of the *EcThreadBoardControl* class in order to be part of the control loop. The resulting system architecture has a light complexity but the flexibility issue increases the effort the user has to carry out to support any new hardware.

```
void XBot::XBotCore::control_init(void)
{
     halInterface->init();
     init_internal();

     return;
}

int XBot::XBotCore::control_loop(void)
{
    int state = halInterface->recv_from_slave();
    if(state == 0)
      loop_internal();
    halInterface->send_to_slave();
}
```

Figure 2. *XBotCore* - R-HAL components interaction.

B. The R-HAL approach

The purpose of the new software design is to provide a middleware independent from the low level hardware/software with autonomous threading capability. The UML class diagram in Fig. 4 shows the software architecture with the R-HAL approach.

[3]http://gazebosim.org/

The core component is the *HAL* interface responsible to provide an endpoint to the low level hardware/software layer in such a way to be flexible towards any new robots. The HAL achieves its job by providing three abstract methods that each concrete implementation has to implement. In detail, adopting a master - slave approach, the HAL lifecycle is described by the following methods:

- *init()*, useful in the initialization phase (open connection, initialization of data structures);
- *recvFromSlave()*, designed to communicate to the slaves and fill the data structure;
- *sendToSlave()*, deals with the communication to send the reference signals to the slaves.

The HAL interface extends the XBotJoint, XBotFT, XBotIMU and XBotHand interfaces but it does not provide any implementation, leaving the job to the child classes. In particular we provide three implementation XBotEcat, XBotEthernet, XBotKuka, XBotGazebo respectively for Ethercat, Ethernet, Kuka lwr 4 based robots and Gazebo based simulator.

The described design does not limit the user to add new behaviour because it is just required to implement other set of interfaces. All the HAL implementation are built as shared library loaded at runtime according to what specified in a configuration file. In particular the factory design pattern has been adopted to load/unload several implementations. The *XBotCore* class will load the specific HAL implementation getting the factory from the *HALInterfaceFactory* class where the symbols are resolved immediately to avoid slowdowns during the RT execution.

The *XBotCoreThread* gets the threading capability by inheriting from *ThreadHook* class such that it is possible to realize the control loop. In order to provide more flexibility a Controller interface is used to represent a generic controller behaviour. *XBotCore* acts as a specific controller implementation and it interacts with the *HAL* by calling the *init()* method in the *controlInit()* and the other two methods inside the *controlLoop()* as shown in Fig 2. The task of *XBoCoreThread* class is to invoke the *controlInit()* and *controlLoop()* methods on a controller implementation.

The described abstraction layer allows to easily switch between simulation and real robot as shown in Fig. 1. The *XBotGazeboPlugin* class emulates the same behaviour of the *XBotCoreThread* class relying on the *ModelPlugin* class to be part of the Gazebo internal loop. The advantages of the presented architecture are evident and despite the high flexibility provided it does not increase considerably the complexity of the software design.

C. Mixing RT-NRT controlled hardware

The software architecture presented in the previous chapter is well designed in the scenario in which we suppose to have either a fully RT or fully NRT environment. However how to mix NRT and RT controlled hardware is a requirement to be considered. One possible example of the aforehead mentioned condition, could be interfacing a RT hardware robotic arm with a NRT robotic end-effector (hand). A solution that exploits the Cross Domain Datagram Protocol

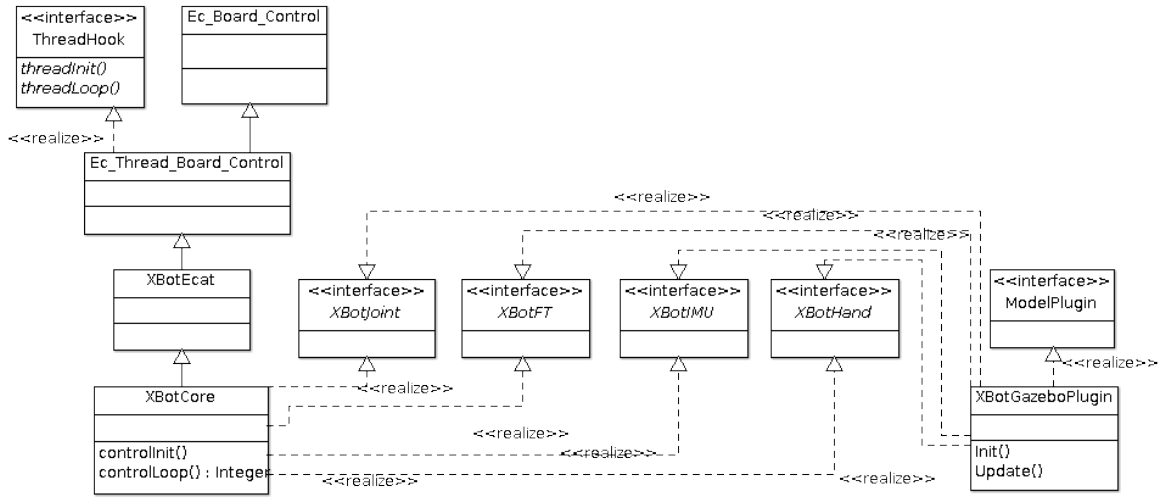

Figure 3. Initial *XBot* software design as UML class diagram.

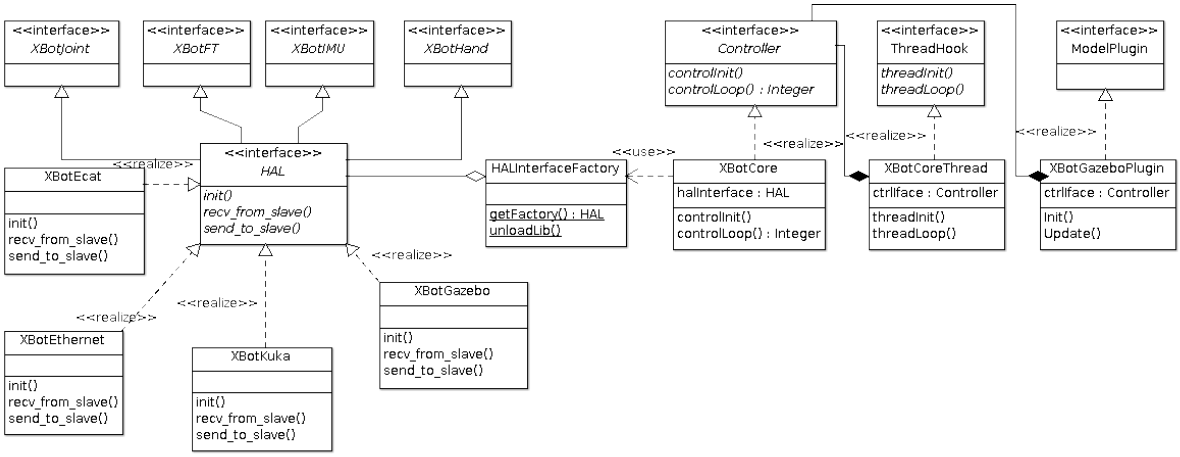

Figure 4. R-HAL *XBotCore* software design as UML class diagram.

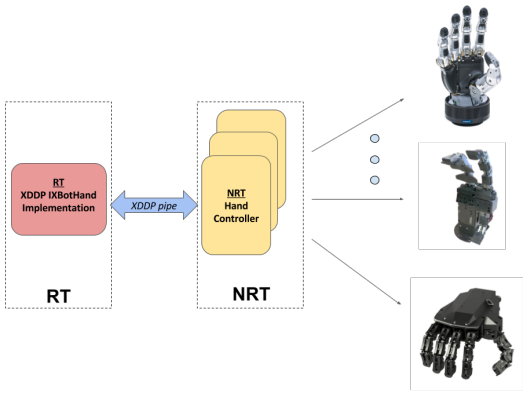

Figure 5. NRT end-effector (Hand) control from the RT side.

pipes (XDDP pipes) from *Xenomai*[4] (required for the *XBot* RT architecture) has been adopted. The design idea is shown in Fig.5, where we have an implementation of the *XBotHand* interface that uses XDDP pipes to forward the data to the NRT side and vice versa. A NRT thread will manage the robotic hardware exploiting the XDDP communication and running the low level end-effector control modules. The interaction between the RT and NRT domains takes place inside one of the possible HAL implementation. A mapping between the joint ID and *IXBotHand* interface allows us to load at runtime the specific RT-based or NRT-based end-effector (hand) control implementation.

IV. Experiments

To validate and evaluate the performance of the XBotCore software platform, we performed a set of experiments on

[4]https://xenomai.org/

different robotic platforms as described in the following section. Moreover a comparison between the former software design and the R-HAL introduced design is provided.

A. Experimental Setup

The proposed software architecture has been validated on the following robotic platforms (Fig. 6):

- WALK-MAN robot, a full-size humanoid with 33 DOFs (Degree-Of-Freedoms), 4 custom F/T sensors and 1 IMU. The WALK-MAN head is equipped with a CMU Multisense-SL sensor that includes a stereo camera, a 2D rotating laser scanner, and an IMU.
- CENTAURO upper body prototype has 15 DOF [19]. Each arm has 7 DOF. For the trunk there is an additional 1 DOF that permits the yaw motion of the entire upper body and extends the manipulation workspace of the robot. 2 custom F/T sensors are placed at the wrists.
- COMAN robot, a 95cm tall humanoid robot with 29 DOFs. Custom torque sensors are integrated into every joint to enable active stiffness control and 6-DOF sensors are included at the ankle to measure ground reaction forces.
- KUKA lwr 4+, an industrial 7-axis jointed-arm robot. Each joint is equipped with a position sensor on the input side and position and torque sensors on the output side. The robot can thus be operated with position, velocity and torque control. Data transfer between the robot controller and an external computer is carried out using the FRI (Fast Research Interface).

Figure 6. Robotic platforms used during the experiments: on top left corner KUKA lwr 4+, on bottom left corner COMAN, on the center WALK-MAN robot and on the right CENTAURO upper body.

All the robotic platforms were tested both on the Gazebo environment and on the real hardware by running an *XBot* plugin that performs a circular trajectory. Thanks to the dynamic API the experiments were executed in all the robots without code modification but just loading the configuration file for the specific robot platform. In particular, the OpenSoT control framework[20], [21] was used to solve whole-body control and inverse kinematics.

The software has been tested on Linux OS 3.18.20-xenomai-ipipe-2.6.5 running on COM Express i7 CPU @ 2.30GHz quad-core for WALKMAN and CENTAURO while the COM Express i7 CPU @ 1.7GHz dual-core was used for COMAN and KUKA arm. RTnet, an Open Soure hard real-time network protocol stack for Xenomai and RTAI was used assuring an implementation of the UDP/IP, TCP/IP (basic features), ICMP and ARP in a deterministic way. It is important to mention that in the case of the KUKA arm all the control was done in NRT mode because we reused the FRI API communication provided by KUKA intrinsically not RT safe.

B. Results

In Fig. 7 we present the results of the R-HAL validation in terms of control period. We report the data of the experiments on the four robotic platforms described in the Experimental Setup section. It is important to note that we requested a control frequency of 1 kHz (i.e. 1 ms control period), for all the platform RT controlled (i.e. CENTAURO upper body, COMAN and WALK-MAN). As already mentioned, the KUKA lwr 4+ was controlled in N-RT mode at 200 Hz (i.e. 5 ms control period). It is evident that no overhead has been introduced with the R-HAL compared to the former XBot software design.

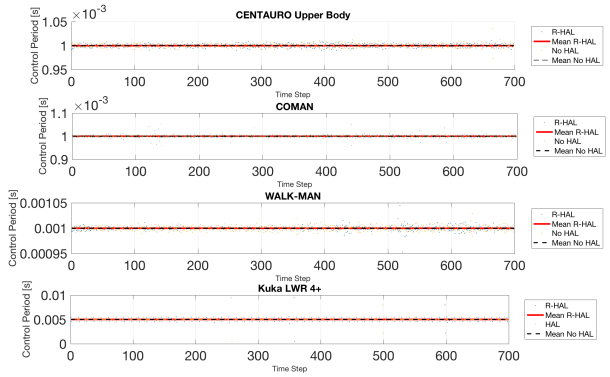

Figure 7. Control period comparison between the XBot architecture with R-HAL and with No HAL.

To complete our analysis, in Table I, we show a comparison of the mean CPU usage time for the XBot platform with or without the R-HAL. The outcome is that the R-HAL does not introduced any processing overhead.

TABLE I
SINGLE CORE MEAN CPU USAGE %

	No HAL	**R-HAL**
CENTAURO upper body	9.3 ±1.4	9.4 ±1.9
COMAN	12.0 ±2.2	12.2 ±2.3
WALK-MAN	18.2 ±1.7	18.5 ±1.5
KUKA	3.8 ±0.8	3.9 ±1.0

V. Conclusions

The extensive uprising in robotic hardware of different form is imposing a significant challenge to the robot software developers when they have to deal with robotics systems with hardware incompatibility making code development, porting and reuse highly inefficient. To address this barrier, hardware abstraction layers are necessary to mask the robot hardware variances and the associated hardware-dependent instructions to the robot from the application/control software developers, enabling efficient and unified code development and reuse among different robot hardware.

Towards this direction, we presented the concept of R-HAL, a new Robot Hardware Abstraction Layer as a way to improve the reuse of the code among different hardware robotic platforms. In particular, we leveraged on the power of the *XBot* software framework to develop and validate R-HAL on a range of different robotic platforms. A comparison between the initial software design and the new one has been performed showing the advantages and disadvantages of both architectures. The autonomous threading capability together with the R-HAL interface allows to easily adopt the *XBot* framework as a middleware for several different robots by exploiting the power of the software polymorphism. Moreover an example of integrating RT and NRT domain to support hybrid RT-NRT hardware has been shown leading to a very high flexibility, towards any type of low-level interface. The results demonstrate that the change in complexity does not affect the CPU load maintaining the desired execution bandwidth (control period, RT-NRT) in all robotic platforms.

Future work will focus on the design of a safety layer embedded in the R-HAL by applying hardware specific capability constraints to the generated command signals and defining the robot specific workspace for safe human robot collaboration. Furthermore, R-HAL will be validated on other research and industrial robotic platforms to rigorously evaluate its benefits. The XBot software platform is released free and open source[5].

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme [FP7-ICT-2013-10] under grant agreements n.611832 WALKMAN, and European Unions Horizon 2020 research and innovation programme under grant agreement No 644839 (CENTAURO) and 644727 (CogIMon). The authors would like to thank Luka Peternel for the support provided during the experiments.

References

[1] 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 *IEEE International Conference on Robotic Computing*, 2017.

[2] S. Jrg, J. Tully, and A. Albu-Schffer, "The hardware abstraction layersupporting control design by tackling the complexity of humanoid robot hardware," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6427–6433. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6907808/

[3] T. Murray, B. Pham, and P. Pirjanian, *Hardware abstraction layer for a robot*. Google Patents, May 2005. [Online]. Available: https://www.google.com/patents/US6889118

[4] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The iCub humanoid robot: an open platform for research in embodied cognition," in *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM, 2008, pp. 50–56. [Online]. Available: http://dl.acm.org/citation.cfm?id=1774683

[5] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of HyQ a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, Sep. 2011. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0959651811402275

[6] S. Potra and G. S. LVD-Napomar, "EtherCAT protocol implementation issues on an embedded linux platform," *Automation, Quality and Testing, Robotics, 2006 IEEE International Conference on*, 2006.

[7] N. G. Tsagarakis, S. Morfey, G. M. Cerda, L. Zhibin, and D. G. Caldwell, "Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 673–678. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/6630645/

[8] A. Parmiggiani, M. Maggiali, L. Natale, F. Nori, A. Schmitz, N. Tsagarakis, J. S. Victor, F. Becchi, G. Sandini, and G. Metta, "The design of the icub humanoid robot," *International Journal of Humanoid Robotics*, vol. 09, no. 04, p. 1250027, 2012. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0219843612500272

[9] E. T. Group, "Ethercat technology introduction," https://www.ethercat.org/en/technology.html.

[10] N. G. Tsagarakis, D. G. Caldwell, A. Bicchi, F. Negrello, M. Garabini, W. Choi, L. Baccelliere, V. Loc, J. Noorden, M. Catalano, M. Ferrati, L. Muratore, A. Margan, L. Natale, E. Mingo, H. Dallali, J. Malzahn, A. Settimi, A. Rocchi, V. Varricchio, L. Pallottino, C. Pavan, A. Ajoudani, J. Lee, P. Kryczka, and D. Kanoulas, "WALK-MAN: A High Performance Humanoid Platform for Realistic Environments," *Journal of Field Robotics (JFR)*, 2016.

[11] OMG, "Object management group," http://www.omg.org/.

[12] J. OMG, "Hardware abstraction layer for robotic technology (hal4rt)," http://www.omg.org/spec/HAL4RT/1.0/Beta1/PDF/.

[13] H. Bruyninckx, "OROCOS: design and implementation of a robot control software framework," *Proc. IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatron.*, 2002.

[14] EEROS, "Eeros," http://wiki.eeros.org/getting_started/overview.

[15] O. S. H. J. I. K. J. L. eongsoo Lim, Inwook Shim and J.-H. Oh, "Robotic software system for the disaster circumstances: System of team kaist in the darpa robotics challenge finals," *International Conference on Humanoid Robotics*. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7363509&tag=1

[16] K. K. T. K. Noriaki Ando, Takashi Suehiro and W.-K. Yoon, "Rt-middleware: Distributed component middleware for rt (robot technology)," *IEEE/RSJ International Conference on Intelligent Robots and Systems*. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1545521

[17] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal on Advanced Robotics Systems*, 2006.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[19] L. Baccelliere *et al.*, "Development of a high performance bi-manual platform for realistic heavy manipulation tasks," in *IEER/RSJ IROS (to appear)*, 2017.

[20] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. G. Tsagarakis, "Opensot: a whole-body control library for the compliant humanoid robot coman," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6248–6253.

[21] E. Mingo 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 Robots, Humanoids 2017, Birmingham, UK, November 15-17, 2017*, 2017.

[5] https://github.com/ADVRHumanoids/XBotCore