Robotran-YARP interface: a framework for real-time controller developments based on multibody dynamics simulations

Timothée Habra, Houman Dallali, Alberto Cardellino, Lorenzo Natale, Nikolaos Tsagarakis, Paul Fisette and Renaud Ronsse

Abstract Multibody dynamics simulation is widely used for prototyping and testing controllers. However, the transfer of controllers initially developed in simulation to real mechatronics platforms requires updating the code in order to interface with physical sensors and actuators. Due to this strong coupling with specific hardware, the controller re-usability is often severely compromised.
In the present contribution, we solve this issue by adding a middleware between the controller and the controlled platform (either real or simulated). This framework decouples the controller from the hardware, allows fast controller developments and eases collaborations on large scale projects. Moreover, it offers the possibility to simultaneously control the real and the simulated robot from a unique controller.
This paper presents the interface of the Robotran dynamic simulator with the YARP middleware. Robotran leverages symbolic generation of the multibody equations to provide fast and accurate simulations of multibody systems. The speed and accuracy of Robotran make it possible to test real-time controllers in a realistic simulation environment. This framework is illustrated with applications using the COMAN and WALK-MAN humanoid robots.

Timothée Habra · Paul Fisette · Renaud Ronsse
Center for Research in Mechatronics, Institute of Mechanics, Materials, and Civil Engineering, Université catholique de Louvain (UCL),
Place du Levant 2, 1348 Louvain-la-Neuve, Belgium,
e-mail: [timothee.habra, paul.fisette, renaud.ronsse]@uclouvain.be

Houman Dallali · Alberto Cardellino · Lorenzo Natale · Nikolaos Tsagarakis
Advanced Robotics & iCub Facility, Instituto Italiano di Tecnologia (IIT),
Via Morego 30, 16163 Genova, Italy
e-mail: [houman.dallali, alberto.cardellino, lorenzo.natale, nikos.tsagarakis]@iit.it

1 Introduction

Simulation tools are widely used in prototyping and testing new technologies. By providing a safe and controllable testing environment, they allow fast and cheap software prototyping. In fields such as robotics or vehicle dynamics, mechanical platforms are not always available for controller testing. Indeed, the mechanical design of first prototypes requires time and resources. Moreover, the few platforms usually built up are often subject to repair or maintenance. In addition, running untested controllers turns out to be potentially dangerous for both the operators (injuries) and hardware (damages). In this context, multibody dynamics simulation is a tool being particularly appealing for the development of controllers for complex mechanical systems.

Nevertheless, porting a controller initially developed in simulation to a real device is not straightforward. It typically requires to adapt the format of the input and output signals of the controller to the actual actuators and sensors. Furthermore, in a real setup, the communication and synchronization between the physical sensors, processors, and actuators must be handled. Hence the controller must also interface with the different communication protocols being used. Not only time consuming and error prone, this coupling between the high-level controller and the mechatronic hardware severely impacts the reusability and lifespan of the code. The code stays with a specific mechatronic platform and usually gets obsolete as soon as the hardware changes. This interfacing issue is thus an obstacle to the efficient use of dynamic simulators for controller developments.

Interestingly, an interfacing burden is also faced in robotics when updating a robot hardware or reusing some code initially developed for another robot. Each time a piece of hardware (e.g. sensors, actuators) is changed, the controllers become outdated and need to be modified. This is particularly burdensome in modern projects deploying advanced devices being constantly improved and updated, especially within the research community. To tackle this problem, the use of a so-called "middleware" emerged as a suitable solution to enhance code modularity and to encourage code reuse and collaborations across projects. Applying the principles of a middleware from robotics to dynamic simulators may help to solve the interfacing problem mentioned above.

The principle of a middleware consists in moving all of the code specific to the hardware into an intermediary software lying between the controllers and the robot, namely the middleware (see Figure 1). This software manages the communication between the controllers and the actual device. It provides the controllers with an interface to read sensors and drive actuators. Thus, the controllers become totally independent of the hardware. People developing controllers can then focus on their algorithm and are set free from the interfacing with the device. The lifespan of the code is then extended as it does not need to be modified when some pieces of hardware are updated or changed. By decoupling the controllers from the robot hardware, it further allows a unique controller to work with different pieces of hardware (e.g. joint encoders) as long as they are properly interfaced with the middleware.

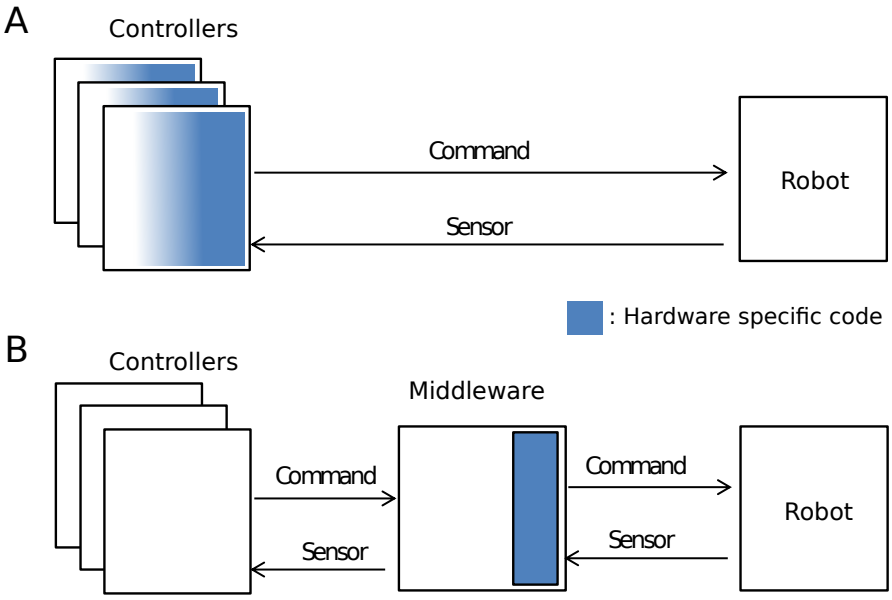

Fig. 1 Blue rectangles represent the piece of code being specific to the hardware. A) Controllers directly commanding the robot, with their hardware-specific code. B) Controllers interfaced with a middleware and having no hardware-specific code.

This approach can also be applied with a multibody simulator. Indeed, from a controller perspective, a multibody simulator can be seen as a collection of virtual hardware (actuators and sensors). Therefore, as for any physical device, the virtual hardware of a simulator can be interfaced with a middleware that takes care of matching the relevant input/output signals. Then, without changing a single line of code, the very same controller can work with the simulator and with the real robot.

Over the years, several middleware frameworks were developed. Generic middleware like CORBA[1], Ice[2] or ØMQ[3] provide complete communication backbones and are widely used to communicate through the internet network. However, they are rarely employed in robotics. Firstly, they are not easy to adopt by non-experts. Secondly, they lack specific components for robotics such as interfaces for sensors and actuators. For that reason, middleware being specific to robotics appeared, such as Player [6], ROS [29], Orocos [4], YARP [25], Orca [24], OpenRDK [5], Mira [11] or LCM [18] to mention just a few.

Player [6] is a *hardware abstraction layer* for robotic devices. It implements drivers for a large collection of sensors and robots. The Stage simulator is also interfaced with Player and offers to simulate a population of mobile robots in 2D

[1] http://www.corba.org/

[2] http://www.zeroc.com/ice.html

[3] http://zeromq.org/

[15]. It is not intended to simulate multibody dynamics but rather to test planning algorithms.

ROS [29] can be seen as the evolution of the Player/Stage framework. It was launched in 2007 and has gained a wide audience ever since. It is interfaced with the Gazebo simulator [21] which was initiated as a 3D complement to Stage. Not only a middleware, ROS also provides a rich array of toolboxes and packages for robotics developed among a large community. ROS defines some standard message formats for sensors and provides interfaces for controllers. The Player drivers can be used in ROS.

Orocos middleware [4] builds its communication around CORBA. It is popular in robotics for its kinematics and dynamics library (KDL) that can be easily integrated into controllers. Orocos direct dynamics module in addition with a time integrator can be used to create a custom simulator [12]. However, contacts with the environment are not included in this scheme. Orocos can also connect to the Gazebo simulator through ROS.

YARP middleware [25] is mainly used in humanoid robotics, and most notably for the iCub platform [26]. For such a kind of robot with many actuators and sensors, YARP facilitates the code modularity by abstracting the hardware through interfaces. It can be extended with device drivers to be compatible with a variety of sensors and actuators. Originally, a dynamic simulator based on Open Dynamics Engine (ODE) was developed for the iCub [32]. However, it was not generic enough to simulate other robots. Therefore, a plugin to use YARP with Gazebo was also developed [17].

For more details about the different middleware used in robotics, the interested reader can refer to the following surveys [22, 27].

Currently, the multibody research community is not sufficiently connected with the robotics research community. In this regard, we believe that the adoption of middleware in the multibody domain could facilitate collaborations. This would be beneficial for both robotics and multibody researchers. On one hand, roboticists would gain access to the state-of-the-art in multibody dynamics simulators. Indeed, the middleware interface would guarantee an effortless transfer of their controllers to new simulators. A strong current trend in robotics is to use simulators being initially developed for video games, like ODE and Bullet [19]. These simulators are primarily targeting fast computation with results being visually-plausible rather than physically correct. Therefore, adoption of simulators from the multibody community should improve the accuracy of the simulations. On the other hand, the multibody community could improve the code modularity and reusability, and more generally, enhance collaborations using software development techniques coming from robotics.

This work presents the interfacing of the Robotran multibody dynamics simulator [9] with the YARP middleware [25]. Similar interfaces between middleware and simulators already exist: for example YARP with Gazebo [17] and ROS with Gazebo[4]. The Gazebo simulator is attractive and widely used in the robotics com-

[4] http://wiki.ros.org/gazebo_ros_pkgs

munity because it allows to model complex world and vision sensors. Moreover, Gazebo and ROS communities are large, active, and partly overlap. However, none of these existing solutions is satisfying enough to get accurate simulation in realistic scenarios while remaining faster than real-time. This demanding tradeoff in terms of computational speed and simulation accuracy is an incentive for using Robotran. Indeed, thanks to its symbolic formalism, Robotran can generate for any given rigid multibody model, its corresponding equations of motion in a form specifically tailored for fast computation. By removing the unnecessary arithmetic operations of a model, Robotran yields an optimal form of the kinematic and dynamic equations and of the associated routines [9].

Importantly, the user should not lose the flexibility offered by the middleware by being forced to use a single simulation environment. Therefore, a controller developed with the Robotran-YARP interface remains totally compatible with the Gazebo-YARP interface. Indeed, since the Gazebo-YARP and Robotran-YARP plugins implement all the same drivers, the same controllers can also work with both. This is precisely the objective of a middleware. Also, YARP has the capability to interoperate with ROS. Therefore, our interface is also compatible with software being developed by the ROS community.

The presented approach remains general and can be followed for other multibody simulators and other middleware. Robotran was selected as the dynamic simulator for its speed and accuracy provided by its symbolic formalism. YARP was chosen for its openess and interoperability (e.g. fully compatible with ROS). The corresponding code is open source and publicly available[5].

This chapter is structured as follows. Section 2 outlines the main features and implementation of the YARP middleware. Section 3 presents the drivers implemented to interface Robotran with YARP. Then, section 4 details some applications enabled by the proposed framework. To illustrate how the proposed framework can be used for controllers of complex mechatronic systems, examples with the COMAN [33] and WALK-MAN [1] humanoid robots are given. Finally, section 5 gives a brief conclusion.

2 YARP Middleware

A major problem in robotics is that controllers can quickly get entangled with the platform they are running on and the devices they are controlling. Moreover, many modules are typically used to control a robot. Managing the inter-module communication can quickly become intractable and time consuming. In that context, the YARP robotics framework has been developed to help reduce the time spent doing infrastructure-level programming rather than actual research [13, 25].

[5] in public repositories https://gitlab.robotran.be/walkman

2.1 Motivation

A typical example of a middleware utility is raised in the WALK-MAN project, aiming at developing a humanoid robot able to support rescue teams by performing activities in the most hazardous disaster zones[6] (e.g. Fukushima nuclear plant). Example of tasks to be achieved are those proposed at the recent DARPA Robotics Challenge, such as valve turning, stairs climbing or debris removal[7]. These tasks require integration of multiple modules of perception, locomotion and manipulation. An overview of this high level control framework is presented in Figure 2.

This type of collaborative research project requires various developed modules to be shared among multiple research groups located in different countries (in this case Italy, Switzerland, Germany and Belgium). During the first months of the project, the WALK-MAN robot was designed. Consequently, the first software developments were performed in simulation or by using other humanoids robots (e.g. the COMAN [33]). Last but not least, some controllers and software tools previously developed in other projects were expected to be reused.

Without using a middleware, all of the software developments and integration achieved in a time period of 18 months would have been impossible.

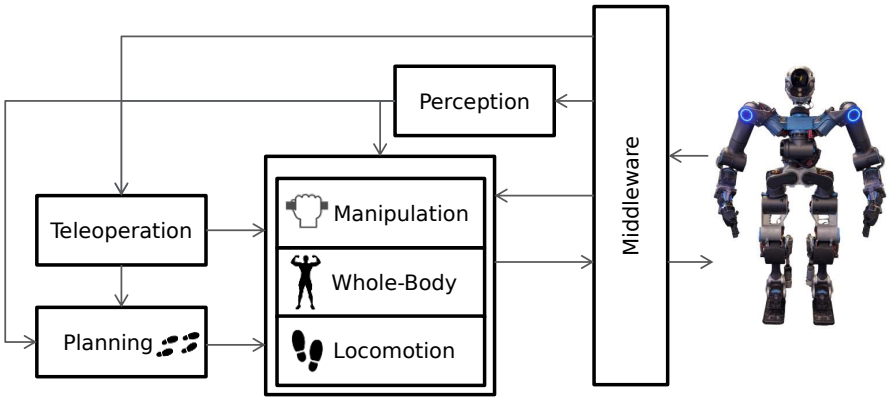

Fig. 2 Overview of the control framework of the WALK-MAN robot. It represents a typical example of the modules being used to control the humanoid robot performing the tasks of the DARPA Robotics Challenge.

[6] see www.walk-man.eu/

[7] http://www.theroboticschallenge.org/

2.2 Middleware Principle

The key idea of a middleware is to extract the hardware-controller interface out of the controller code itself. As shown in Figure 1, all of the hardware dependent code is handled by the middleware. The different controllers are then totally "agnostic" with respect to the actual hardware being used.

The hardware dependent codes manage the direct operation of the hardware devices, known as the *device drivers*. It can be for example the code requesting the joint position measured by an encoder communicating through an I2C bus. It can also be the code sending a desired voltage command through a CAN bus to the power electronics driving a DC motor. Thus, for every actuator or sensor communicating with the controller, a driver must be implemented. All of the drivers of a specific platform are gathered in the middleware.

In order to keep the controllers flexible to hardware changes, *device interfaces* are used between the user controllers and the middleware. They consist of hardware abstraction layers for families of devices. The idea is to provide a unique interface for all the devices having the same functions, regardless of the technology they rely on (see Figure 3). The user control algorithms are blind with respect to the actual devices being used on the robot. The only relevant information they depend on is the type of interface being available on the mechatronic platform. For example, the encoder interface can be used for any sensor providing a measurement of the joint position. If an upgrade of the platform requires to replace optical encoders by magnetic encoders, the controllers are not affected. Programmers can write the higher-level application code independently of the specific hardware of a particular platform in a particular configuration.

This abstraction through device interfaces is actually borrowed from classic computer operating systems. For example, a computer software can use an interface for the clicking devices. The interface makes it unnecessary to re-install all computer applications after plugging in a mouse or when using a new touch screen. Instead, it is simply required to have the correct drivers for these devices while they have to comply with the same interface.

2.3 Implementation

Practically in YARP, the device interfaces consist of Application Programming Interface (API) used to control actuators and read sensors data. They are implemented in C++ as abstract base classes with a set of virtual methods. So, each type of sensor (e.g. joint position encoders) is associated to a set of interfaces specifying the collection of functions available to access the data (e.g. *getEncoder()*). Similarly, each type of actuator (e.g. a joint DC motor) is associated to interfaces specifying the functions to control it (e.g. *positionMove(ref)*). For more details about YARP interfaces, see [13].

Following the object oriented programming paradigm, device drivers are children of the device interface classes. So for each device interfaced with YARP, a *driver* should implement the corresponding *interface* classes. Drivers of different devices belonging to the same family can implement the same interface. This allows different types of encoders to implement the same *encoder interface* and thus to be compatible with the same controller. As written above, this architecture makes the controller be device agnostic: it knows the interface functions, but does not need to know which specific device implements them. The controller should only fulfill the requirements regarding the type and format of the data it gets or needs to send. So each device can implement interfaces when appropriate, with the requirement of being compliant with the specifications, e.g. regarding the unit (degrees for angular positions, Newton for forces, etc). This guarantees that the controller can work with any platform implementing its interfaces, as illustrated in Figure 3.

In addition, for each interface, YARP implements generic network proxy devices allowing the remote execution of the same code. Therefore, different controller pieces can run on different machines. Thus, controllers on a user laptop can be interchangeably connected to the physical mechatronic platform or to a simulator with no necessity to recompile the program. This offers for instance to rely on the specific advantages of specialized computers for specific computations (e.g. for vision or intensive computation).

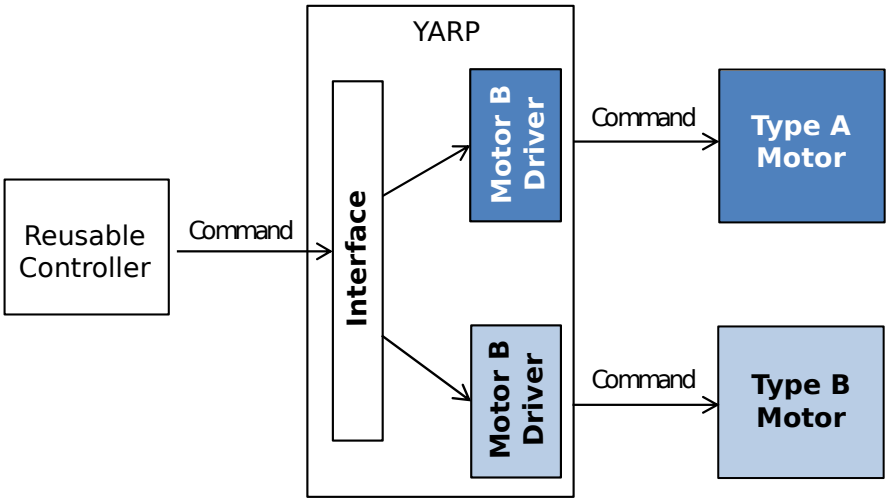

Fig. 3 Different devices of the same family, *Type A Motor* and *Type B Motor* are controlled by the same controller. Each motor has its own driver but both drivers implement the same interface.

3 Robotran-YARP drivers

Since a simulator can be seen as a virtual robot, it is possible to implement drivers for the simulated devices. This was performed for the Robotran simulator, as described in this Section. This implementation allows to test controllers in simulation and then to port them to the real robot without changing a single line of code. Figure 4 illustrates an example of this structure with the COMAN humanoid robot.

Robotran is particularly tailored to fast and accurate multibody dynamics simulations [30]. To challenge the capacity of Robotran to produce fast and accurate simulations, the proposed interface aims at testing robotic platforms and controllers involving large dynamical contributions (e.g. locomotion, whole-body control). Consequently, the drivers for actuators and for the sensors measuring dynamics (i.e. torque sensors rather than cameras) are implemented in the Robotran environment.

3.1 Control Board Driver

On a real robot, control boards are electronic boards supervising the joints actuation. A control board often embeds a joint control algorithm implemented at the firmware level. Different control modes are usually available (e.g. position or torque control). As displayed in Figure 5, this low-level control algorithm drives the actuator and receives a feedback from the joint encoder. If available, it can also acquire a torque feedback from a joint torque sensor. These low-level joint controllers can usually be

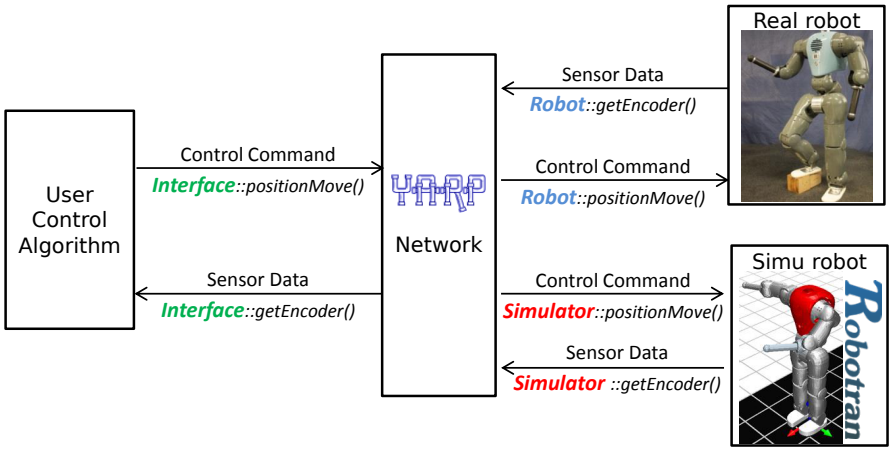

Fig. 4 Robotran-YARP control framework. The user control algorithm calls the interfaces functions without knowing what device is behind. The real robot and the one simulated with Robotran, have their own drivers implementing these interfaces. The very same controller can then be used with both the real and the simulated robot.

decoupled from the high-level control algorithms. While the high-level controllers (e.g. locomotion algorithm) run on a central on-board PC, the low-level controllers (e.g. torque control) directly run on the electronic boards. Typically, high-level algorithms do not directly drive the motors but rather send references (position, velocity, torque, etc.) to control boards that will, in turn, drive the motors (see Figure 5). Therefore, a control board driver receives reference commands from the user control algorithm and translates it into commands being specific to the low-level joints controller.

The control board driver is thus in charge of several devices and implements many device interfaces. The main interfaces implemented for the Robotran control board driver are:

- *IEncoders*: returns the joints position, velocity and acceleration.
- *IOpenLoopControl*: sends a voltage command to the DC motors in open loop (without feedback control). This mode thus bypasses the low-level controller mentioned above, allowing feedforward control.
- *IPositionControl*: controls the joint position. It can implement either a PID control (stiff mode) or a joint impedance control (compliant mode).
- *IVelocityControl*: controls the joint velocity. It can implement either a PID control (stiff mode) or a joint impedance control (compliant mode).
- *ITorqueControl*: controls and measures the joint torque.
- *IControlMode2* and *IInteractionMode* : select respectively the control mode (position, velocity, torque, open loop) and the interaction mode (stiff or compliant).
- *IPidControl* and *IImpedanceControl*: control respectively the low-level PID controller gains and the virtual impedance stiffness and damping.
- *IAmplifierControl*: feeds back the electrical current in the motors.

In order to make the simulation environment compatible to the real hardware, two add-ons were implemented in the Robotran simulation suite (and more particularly the devices presented in Figure 5). These add-ons are generic in the sense that they can be reused for different Robotran multibody models by adapting some configuration files.

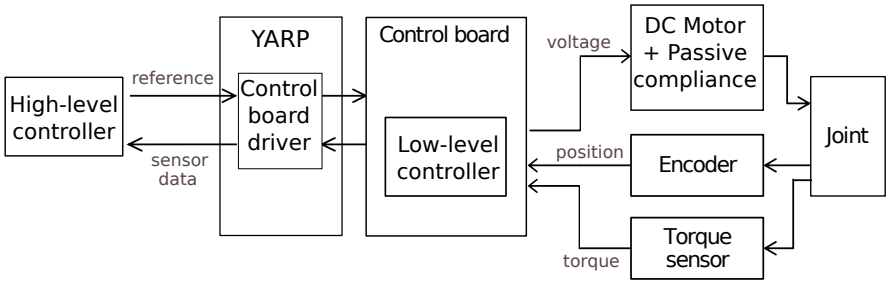

Fig. 5 The control board is a key bridge between the high-level controller and the actual joints. It contains a low-level controller directly controlling the actuators.

The first add-on consists of joints controllers mimicking the firmware low-level controllers of the real robot. This allows to specify for each joint the desired type of control (position, torque, impedance and open-loop). The PID gains can also be modified through this module.

The second add-on consists in a compliant actuator model, which is particularly useful to simulate compliant robots like COMAN [33] and WALK-MAN [7]. It provides the constitutive laws of a DC motor and the mechanical model of a series elastic actuator (SEA). This model is combined with the multibody dynamics through the Robotran *user derivatives* feature. The latter allows integration of the additional differential equations of the DC-motor and the joint compliance together with the set of multibody dynamic equations. All the details can be found in [8, 34].

Based on these add-ons, the Robotran Control Board Driver is implemented. At each simulation time step, joints position, velocity and torque are read by the driver and sent to YARP. The desired joints position or torque are then received from YARP and sent to the Robotran low-level controller.

3.2 Force-Torque Sensor Driver

A 6-axis force-torque sensors measure the wrenches acting on a body, i.e. the 3 force components (F_x, F_y, F_z) and the 3 torque components (T_x, T_y, T_z) expressed in a body fixed frame.

On a real robot, this type of sensor is typically made of strain gauges capturing small deformations. In Robotran, this can be simulated by means 6 locked joints at the sensor location. The Lagrange multipliers computed by Robotran provides the force or torque being required to keep these joints locked [30]. Assuming that the body deformation due to the strain gauge stretching are negligible, this technique is a good approximation for modelling this sensor.

This driver thus implements the *IAnalogSensor* interface which outputs 6 values representing the forces and torques components.

3.3 IMU Sensor Driver

An Inertial Measurement Unit (IMU) measures the orientation, velocity and acceleration of a body.

This physical sensor usually combines accelerometers and gyroscopes [10].

In the simulation environment, the same information can be retrieved using the Robotran forward kinematics module [30]. Placing a Robotran sensor at the same location as the IMU sensor provides position, orientation, velocity and acceleration of this point in space. Note that Robotran provides the body orientation by means of a rotation matrix. It must be converted into Euler angles to comply with the YARP format.

In summary, this driver implements an *IAnalogSensor* interface which outputs the orientation, the angular velocity and the linear acceleration of a body.

3.4 Clock Driver

This driver gets the simulation time in order to synchronize the controllers with the simulation environment. Such a driver is thus specific to the simulation environment, and does not exist in the physical counterpart. Indeed, a simulated time clock is not always constant. It might run faster or slower than real time. If the controllers and the simulator are not synchronized, they will display unrealistic behaviors. All the interconnected systems should thus follow the same clock managed by the simulator clock driver. This driver distributes the clock through the network to all the involved machines and applications. Note that this does not prevent fast simulators like Robotran to run tests of real-time controllers even faster than real time. Indeed, running the simulator clock faster than real-time is possible, as long as all interconnected systems stay synchronized.

Moreover, the Robotran library provides a tool to control the simulation time. The clock can be artificially slowed down in order to carefully observe the device behavior during highly dynamical events, for example, an impact with the ground or an object. Moreover, this central clock even offers to rewind a simulation backward in time, a feature that proves to be very useful when debugging.

4 Applications

The Robotran-YARP framework offers a wide range of applications. Its standardized interface enhances the code reusability and eases the transfer of a controller from a simulation environment to an experimental setup. Moreover, it can be used to simultaneously run a controller on the simulator and on the experimental setup. Finally, the use of a middleware can also facilitate the benchmarking of multibody simulation tools. These different examples are detailed in the sections hereunder

4.1 Code reusability

YARP clearly specifies the interface of the control modules. To (re)use a control module, the only request is to connect it to the correct inputs-outputs through the corresponding YARP communication ports. This “plug and play” architecture eases the software reuse across projects.

As an example of module reusability, Figure 6 shows the *Robot Motor Gui* module controlling the simulated COMAN. This YARP module implements a graphical

user interface to interact with the control boards. It offers to test different low-level joint controllers. It is interesting to note that, at the time this module was written, no interface between YARP and Robotran existed. Nevertheless, as the module was designed to be compliant with YARP, it could be reused in Robotran simulated platform without any modification. Moreover, this module was initially developed for another robot, namely the iCub [26] and was imported here without changing a single line of code, despite a very different hardware. This illustrates the potential of code reusing and cross-projects collaboration brought by our framework.

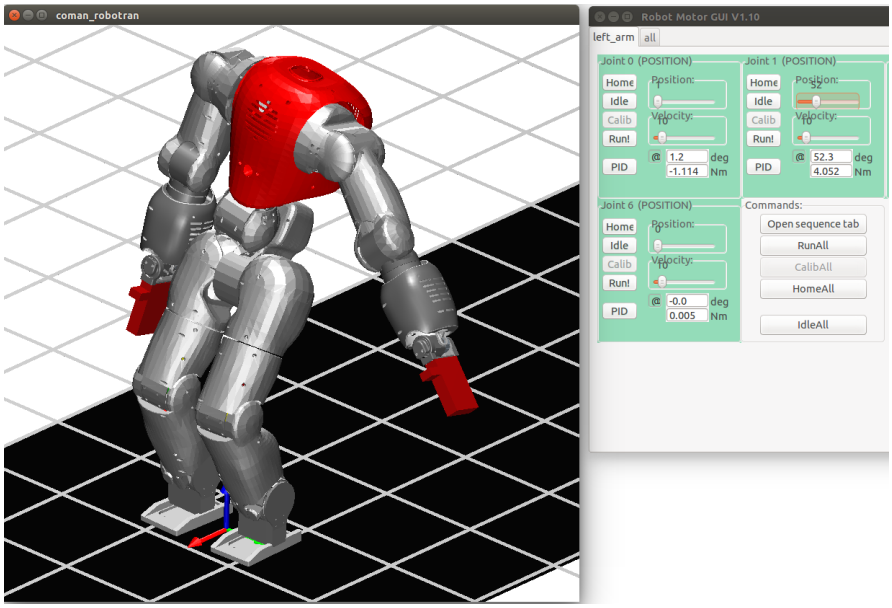

Fig. 6 The COMAN humanoid simulated in Robotran is controlled by a classical YARP-module, the "Robot Motor GUI" through the Control Board interface.

4.2 Code transfer

Our framework further allows to seamlessly transfer a module from simulation to real hardware and *vice versa*. Thanks to the middleware decoupling, the controllers do not need to be recompiled when switching from one environment to the next (simulated or physical). Furthermore, this seamless transfer to simulation makes possible to write regression tests in order to verify that patches or new developments do not modify the expected behavior of the robot software. Controller testing can

thus be automated, as prescribed by the software continuous integration methodology [14].

As represented in Figure 7, a squatting task was run on both Robotran and the real WALK-MAN robot in order to compare the simulated model of WALK-MAN with experiments. The position, torque and the 6 DoF force torque sensor data were logged for comparison. The data is presented for 30 seconds where the robot squats down by 30 cm and returns to its initial posture every 7 seconds, completing a full cycle. In terms of position tracking, once the low-level PID controllers are properly tuned a suitable tracking performance is obtained. Regarding the knee torques, there are some differences between the simulation data and the experimental results. The experimental data is more oscillatory. This is due to parasitic movements in the other joints, while this is better controlled in the simulation. Also the average torque profile is slightly higher in the experiment. This is due to unmodelled shock absorbing plastic covering the robot body and to electrical wires and electronic boards adding weight of the actual robot, while this extra weight was not initially captured in the model. This difference is indeed reduced after adjusting the modelled torso weight, as shown in the figure. Note that a better prediction should likely require an extended system identification for refining the model.

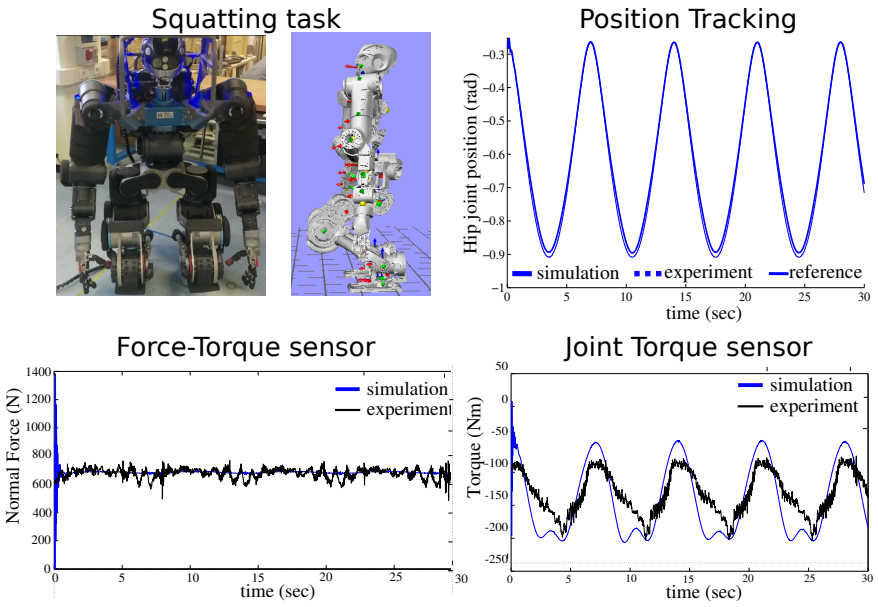

Fig. 7 Squatting task performed by a simulated and real WALK-MAN robot. The leg joints are controlled in position to move the robot from the standing up to the squatting posture. Top left figures show the real and simulated robots in the squat posture. Top right figure represents the evolution of the position tracking of the Hip joint. Bottom left figure represents the measurement of the vertical force in the foot sensor. Bottom right figure compares the experimental and the simulated torque in the knee joint.

4.3 Parallel execution

On top of that, the simulation can also run in parallel to the real robot (See Figure 8). Indeed, the modular interface of the middleware offers to simultaneously control a real robot and a simulated one (the commands being duplicated).

This offers the possibility to add an *internal model* in the control scheme, i.e. a framework with a strongly bio-inspired ground [20]. As sketched in Figure 9, an internal model consists in a dynamic simulator receiving a copy of the commands sent to the real platform. It can then predict the state of the robot. Such a model provides a particularly useful tool to perform *state estimation*, *model identification* and *predictive control* [3, 16, 23, 31].

Among others, we plan to use this tool in the next future to support robotic teleoperation with low bandwidth and noisy communication channels between the robot and its operator.

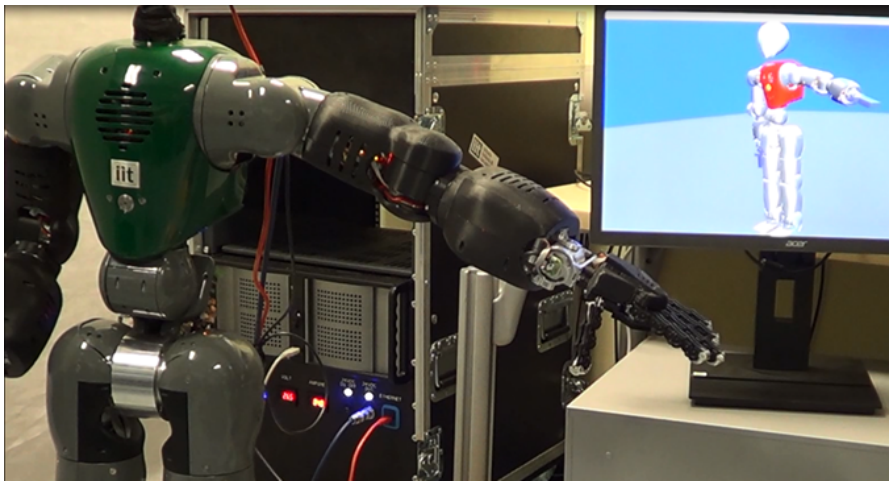

Fig. 8 Real and simulated COMAN humanoid controlled in parallel by a unique controller.

4.4 Benchmarking

Finally, our framework can also be used to support the benchmarking of multibody simulators. Indeed, in the same way a controller can be transferred from a simulated to a real robot, a benchmark test could be transferred from a simulation environment to another. Provided that the simulators are interfaced with the same middleware, any benchmarking test could be performed with no adaptation of the controller code. Following this approach, it is guaranteed that the benchmark is performed in the

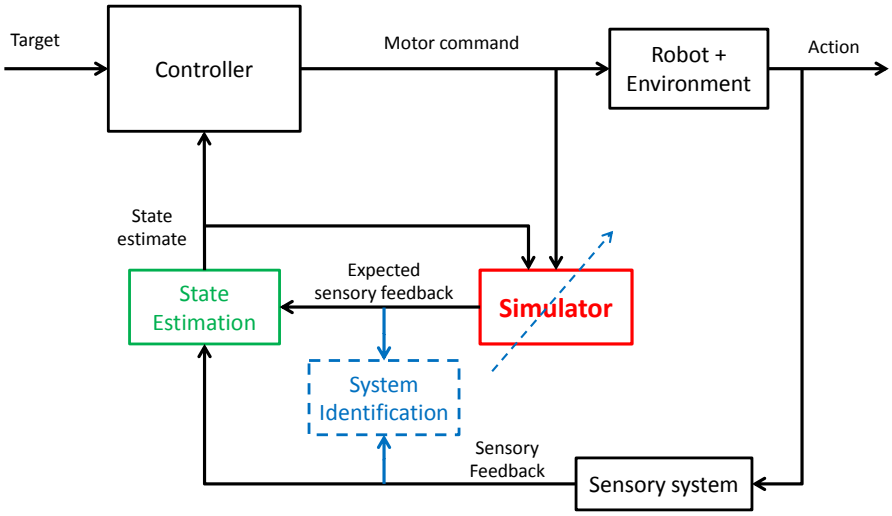

Fig. 9 Control scheme using a forward internal model (in red, in our case represented by the simulator). The internal model receives a copy of the command and predicts the expected sensory output. This information can then be used for system identification and state estimation. (adapted from [31]

same conditions for all the simulators under test. This is similar to the benchmarking in [2] and [28] using a physics abstraction framework to switch from one simulator to another.

5 Conclusion

In conclusion, a clean interface between Robotran and YARP was developed in a generic library. The use of a middleware interfaced with a multibody simulator offers many advantages for software development.

Firstly, the YARP middleware provides all the tools being necessary to create modular projects. It decouples the high level controllers from the controlled platform (real or simulated). This modularity eases the creation, maintenance and reuse of the software modules. This naturally supports collaborations and enhances the code durability.

Secondly, the integration of a dynamic simulator provides a powerful testing environment. It allows to perform early and regular tests in a safe and cheap environment. It is important to recall that, in the proposed framework, the code tested in the simulation environment is exactly the same as the one ported to the real platform. Interestingly, the bugs can further be replicated! This testing capability is key to produce high quality software modules.

Finally, the adoption of middleware among multibody research groups could tighten the link with robotics researchers: the middleware being the glue between the multibody simulation environments of the ones and the high level controllers of the others.

The modularity and testing capabilities enabled by our framework were successfully implemented with the WALK-MAN and COMAN robots. Fast software prototyping could thus be achieved thanks to the reusability brought by the YARP interface. Moreover, the transfer of a controller tested in a simulation to the real platform is now straightforward: the exact same code can run in simulation or on a real experimental setup. More advanced applications such as the parallel execution of the same controller in both the simulation and in the actual robot are also possible. This allows to perform state estimation and online system identification ,i.e. two features being critical for robots working in unknown environments. We also envisioned that this framework could be used for the benchmarking of multibody simulators.

Future works will consist of further extending the collection of robots available to other platforms, like the iCub humanoid robot [26]. We also plan to use it to perform online system identification of the robots and its environment. Drivers for other sensors might also be implemented (tactile sensors, camera, etc).

The proposed framework allows to improve and foster software developments among large collaborative projects such as the WALK-MAN project. We hope it can benefit to other research consortia. Therefore, we released the open source code of the Robotran-YARP plugin together with the simulators of the COMAN and WALK-MAN humanoids in a public repository (https://gitlab.robotran.be/walkman).

Acknowledgements This work was supported by the European Community's 7th Framework Programme (FP7-ICT-2013-10) under Grant 611832 (WALKMAN collaborative project) and by the Belgian F.R.S.-FNRS (Crédit aux Chercheurs 6809010 awarded to Renaud Ronsse). We are grateful to Alexandra Zobova and Nicolas Van der Noot for their support in the development of the simulator.

References

1. Eu project whole body adaptive locomotion and manipulation (walkman). [Online]. Available http://walk-man.eu/
2. Boeing, A., Brunl, T.: Evaluation of real-time physics simulation systems. In: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, pp. 281–288. ACM (2007)
3. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science **314**(5802), 1118–1121 (2006)
4. Bruyninckx, H.: Open robot control software: the orocos project. In: Proceedings of the Robotics and Automation (ICRA). IEEE International Conference, vol. 3, pp. 2523–252 (2001)
5. Calisi, D., Censi, A., Iocchi, L., Nardi, D.: Design choices for modular and flexible robotic software development: the OpenRDK viewpoint. Journal of Software Engineering for Robotics **3**(1), 13–27 (2012)

6. Collett, T.H., MacDonald, B.A., Gerkey, B.P.: Player 2.0: Toward a practical robot programming framework. In: Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005), p. 145 (2005)
7. Dallali, H., Mosadeghzad, M., Medrano-Cerda, G., Loc, V.G., Tsagarakis, N., Caldwell, D., Gesino, M.: Designing a high performance humanoid robot based on dynamic simulation. In: Modelling Symposium (EMS), 2013 European, pp. 359–364. IEEE (2013)
8. Dallali, H., Mosadeghzad, M., Medrano-Cerda, G.A., Docquier, N., Kormushev, P., Tsagarakis, N., Li, Z., Caldwell, D.: Development of a dynamic simulator for a compliant humanoid robot based on a symbolic multibody approach. In: Mechatronics (ICM), 2013 IEEE International Conference on, pp. 598–603. IEEE (2013)
9. Docquier, N., Poncelet, A., Fisette, P., others: Robotran: a powerful symbolic gnerator of multibody models. Mechanical Sciences **4**, 199–219 (2013)
10. Dudek, G., Jenkin, M.: Inertial sensors, GPS, and odometry. In: Springer Handbook of Robotics, pp. 477–490. Springer (2008)
11. Einhorn, E., Langner, T., Stricker, R., Martin, C., Gross, H.M.: Mira-middleware for robotic applications. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 2591–2598. IEEE (2012)
12. Ferraguti, F., Golinelli, N., Secchi, C., Preda, N., Bonfe, M.: A component-based software architecture for control and simulation of robotic manipulators. In: Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on, pp. 1–5. IEEE (2013)
13. Fitzpatrick, P., Metta, G., Natale, L.: Towards long-lived robot genes. Robotics and Autonomous systems **56**(1), 29–45 (2008)
14. Fowler, M., Foemmel, M.: Continuous integration. Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf (2006)
15. Gerkey, B., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th international conference on advanced robotics, vol. 1, pp. 317–323 (2003)
16. Haruno, M., Wolpert, D.M., Kawato, M.: MOSAIC Model for Sensorimotor Learning and Control. Neural Computation **13**(10), 2201–2220 (2001). DOI 10.1162/089976601750541778
17. Hoffman, E.M., Traversaro, S., Rocchi, A., Ferrati, M., Settimi, A., Romano, F., Natale, L., Bicchi, A., Nori, F., Tsagarakis, N.G.: Yarp based plugins for gazebo simulator. In: Modelling and Simulation for Autonomous Systems: First International Workshop, MESAS 2014, Rome, Italy, May 5-6, 2014, Revised Selected Papers, vol. 8906, p. 333. Springer (2014)
18. Huang, A.S., Olson, E., Moore, D.C.: LCM: Lightweight communications and marshalling. In: Intelligent robots and systems (IROS), 2010 IEEE/RSJ international conference on, pp. 4057–4062. IEEE (2010)
19. Ivaldi, S., Peters, J., Padois, V., Nori, F.: Tools for simulating humanoid robot dynamics: a survey based on user feedback. In: Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on, pp. 842–849. IEEE (2014)
20. Kawato, M.: Internal models for motor control and trajectory planning. Current Opinion in Neurobiology **9**(6), 718–727 (1999). DOI 10.1016/S0959-4388(99)00028-8
21. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, vol. 3, pp. 2149–2154. IEEE (2004)
22. Kramer, J., Scheutz, M.: Development environments for autonomous mobile robots: A survey. Autonomous Robots **22**(2), 101–132 (2007)
23. Li, Q., Poo, A.N., Lim, C.M.: Internal model structure in the control of robot manipulators. Mechatronics **6**(5), 571–590 (1996)
24. Makarenko, A., Brooks, A., Kaupp, T.: Orca: Components for robotics. In: Intelligent robots and systems (IROS), 2006 IEEE/RSJ international conference on, pp. 163–168 (2006)
25. Metta, G., Fitzpatrick, P., Natale, L.: YARP: yet another robot platform. International Journal on Advanced Robotics Systems **3**(1), 43–48 (2006)
26. Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F.: The iCub humanoid robot: an open platform for research in embodied cognition. In: Proceedings of the 8th workshop on performance metrics for intelligent systems, pp. 50–56. ACM (2008)

27. Mohamed, N., Al-Jaroodi, J., Jawhar, I.: Middleware for robotics: A survey. In: Robotics, Automation and Mechatronics, 2008 IEEE Conference on, pp. 736–742. IEEE (2008)
28. Peters, S., Hsu, J.: Simple benchmarks for speed and accuracy of rigid body dynamic simulators. In: ECCOMAS Thematic Conference Multibody Dynamics 2015 (2015)
29. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software, vol. 3, p. 5 (2009)
30. Samin, J.C., Fisette, P.: Symbolic modeling of multibody systems, vol. 112. Springer Science & Business Media (2003)
31. Shadmehr, R., Krakauer, J.W.: A computational neuroanatomy for motor control. Experimental Brain Research **185**(3), 359–381 (2008)
32. Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L., Nori, F.: An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator. In: Proceedings of the 8th workshop on performance metrics for intelligent systems, pp. 57–61. ACM (2008)
33. Tsagarakis, N.G., Morfey, S., Cerda, G.M., Zhibin, L., Caldwell, D.G.: Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control. In: Proceedings of the Robotics and Automation (ICRA), 2013 IEEE International Conference on, pp. 673–678. IEEE (2013)
34. Zobova, A., Habra, T., Van der Noot, N., Dallali, H., Tsagarakis, N., Fisette, P., Ronsse, R., others: Multi-physics Modelling of a Compliant Humanoid Robot. In: ECCOMAS Thematic Conference Multibody Dynamics 2015 (2015)