Yarp Based Plugins for Gazebo Simulator

Enrico Mingo Hoffman, Istituto Italiano di Tecnologia, enrico.mingo@iit.it, Genova, Italy
Silvio Traversaro, Istituto Italiano di Tecnologia, silvio.traversaro@iit.it, Genova, Italy
Alessio Rocchi, Istituto Italiano di Tecnologia, alessio.rocchi@iit.it, Genova, Italy
Mirko Ferrati, Centro di Ricerca "E. Piaggio", mirko.ferrati@centropiaggio.unipi.it, Pisa, Italy
Alessandro Settimi, Centro di Ricerca "E. Piaggio", alessandro.settimi@centropiaggio.unipi.it, Pisa, Italy
Francesco Romano, Istituto Italiano di Tecnologia, francesco.romano@iit.it, Genova, Italy
Lorenzo Natale, Istituto Italiano di Tecnologia, lorenzo.natale@iit.it, Genova, Italy
Antonio Bicchi, Centro di Ricerca "E. Piaggio", antonio.bicchi@centropiaggio.unipi.it, Pisa, Italy
Francesco Nori, Istituto Italiano di Tecnologia, francesco.nori@iit.it, Genova, Italy
Nikos G. Tsagarakis, Istituto Italiano di Tecnologia, nikos.tsagarakis@iit.it, Genova, Italy

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, ERC Advanced Grant no. 291166 SoftHands and the CoDyCo project (FP7-ICT-2011-9, No. 600716).

Abstract

This paper presents a set of plugins for the Gazebo simulator that enables the interoperability between a robot, controlled using the YARP framework, and Gazebo itself. Gazebo is an open-source simulator that can handle different Dynamic Engines (ODE, DART, Bullet, SimBody), backed up by the Open Source Robotics Foundation (OSRF) and supported by a very large community. Since our plugins conform with the YARP layer used on the real robot, applications written for our robots, COMAN and iCub, can be run on the simulator with no changes. Our plugins have two main components: a YARP interface with the same API as the real robot interface, and a Gazebo plugin which handles simulated joints, encoders, IMUs, force/torque sensors and synchronization. The robot model is provided to the simulator by means of an SDF file, which describes all the geometric, dynamic and visual characteristics of a robot. Once the SDF is read from Gazebo, our plugins are loaded and associated with the simulated robot model and the simulated world. Different modules for COMAN and iCub have been developed using Gazebo and our plugins as a testbed: joint impedance control plus gravity compensation, dual arm Cartesian control for manipulation tasks, dynamic walking, etc. This work has been developed as part of a joint effort between three different European Projects "WALKMAN", "CoDyCo" and "SoftHands" aiming at implementing a common simulation platform to develop and test algorithms for our robotic platforms. This work is available as open-source to all the researchers in the YARP community (`https://github.com/robotology/gazebo_yarp_plugins`).

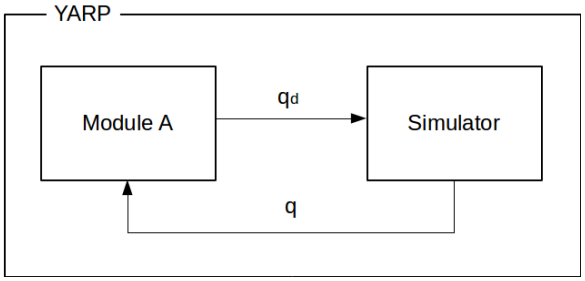
Keywords: Simulator, Robotics, YARP, Gazebo, Open-Source

1 Introduction

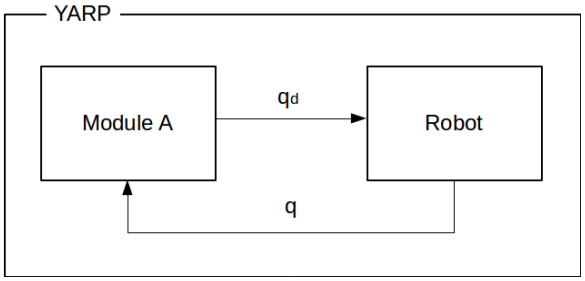
In the past years, robotics researchers have been developing many robotics frameworks such as OpenRDK (Calisi et al. [2008]), YARP (Metta et al. [2006]) or ROS (Quigley et al. [2009]) in order to ease the creation of generic applications for robots and encourage code reuse. The performance overhead introduced by these frameworks is balanced by the architectural benefits, for example they allow to build modular systems to execute one or more assigned tasks.

In these frameworks, the *simulator* is a module that represents the real robot at the interface level (Figure 1). Such simulator module accepts control input (desired joint torques, desired joint position, ...) and outputs sensory feedback (cameras, joint positions, ...) from the simulated world. These simulators usually allow to have the human in-the-loop permitting to train a human operator. The most important aspect is that they allow to develop modules that directly will work in the real robot without any need to rewrite code. In fact, when the real robot is used, there is a module that replaces the simulator by providing the same hardware interfaces.

By accurately simulating robots and environments, code designed to operate on a real robot can be executed and validated on the simulated equivalent system. This avoids common problems associated with



(a) Module A connected to the simulator



(b) Module A connected to the robot

Figure 1: Module A write desired joint position and read actual joint position without knowing if it is interfaced with the simulator or the robot since they expose the same interface.

hardware such as short battery life, hardware failures, and unexpected and dangerous behaviors, particularly during the initial stages of development and tuning of new modules and controllers. It is also much faster to have a simulation engine up and running than using a real robot, especially when the simulation engine can run faster than real-time. In this way the simulator becomes a fundamental part of the framework and the robot software development cycle as the first step to validate algorithms, thus minimizing the risks of hardware breaks.

Figure 2: COMAN and iCub interacting inside a Gazebo simulation of a kitchen. Blue dots represent contact points.

With these concepts in mind we decided to extend one of the most known robotics simulator, Gazebo (Koenig and Howard [2004]), to be compatible with one of the most used robotics framework, YARP (Figure 2), developed in the Italian Institute of Technology. YARP is supported by the iCub simulator (iCubSim, Tikhanoff et al. [2008]) that is dedicated to a specific platform. The needs of a more generic tool for simulating different robots rise up. Gazebo, which has been recently chosen as the simulator for the DARPA Virtual Robotic Challenge (VRC, DARPA [2013]), allows the use of different dynamic engines, it is easily expandable through plugins and it has a strong and active community. Gazebo is maintained by the Open Source Robotics Foundation (OSRF [2011]).

This paper is organized in the following sections: **State of Art**, discusses some of the most popular simulation environments in robotics, **Structure** introduces in detail the Gazebo plugins developed in this work, **Conclusions** summarize the outcome of this effort and finally **Future Works** discusses the follow up activities.

2 State of Art

A large number of simulators have been developed in the past two decades (Ivaldi et al. [2014]). Such simulators range from dynamic solver libraries to complex simulation environments/systems. The latter are usually large projects that provide both rigid body dynamic simulations and tools such as graphical editors, planner libraries, visualization tools, controllers and so on.

The *Open Dynamics Engine* (ODE, Smith [2000]) is one of the most widely used rigid body dynamics engine in robotics simulation. ODE simulates chains of rigid bodies connected and constrained by different types of joints. It has a built-in collision detection system and implements hard contacts using non-penetration constraint whenever two bodies collide. Beside the large number of project that use it, at the moment the development has been paused. *Bullet* (Erwin [2003]) is another dynamic engine. It implements different direct/inverse rigid body dynamic algorithms (eg. Featherstone articulated body algorithm, Featherstone [2007]) as well as different solvers (eg. Mixed Linear Complementarity Problem, MLCP) and contact models. Bullet is used for a wide range of projects and its community is active and continues to improve it constantly.

OpenRAVE (Diankov [2010]) provides an environment for testing, developing, and deploying motion planning algorithms in real-world robotics applications. The main focus is on simulation and analysis of kinematic and geometric information related to motion planning. It provides many command line tools to work with robots and planners, and the run-time core is small enough to be used inside controllers and bigger frameworks. Industrial robotics automation is an important target application.

Webots (Michel [2004]) is a development environment used to model, program and simulate mobile robots. With Webots the user can design complex robotic setups, with one or several, similar or different robots, in a shared environment. A large choice of simulated sensors and actuators is available. The robot controllers can be programmed with the built-in IDE or with third party development environments. The robot behavior can be tested in dynamic simulated worlds (ODE based). The controller programs can optionally be transferred to commercially available real robots.

V-REP (Rohmer et al. [2013]), similarly to Webots, embeds different tools that permit fast developing of algorithms, the code can be transferred inside real robotic hardware.

Gazebo (Koenig and Howard [2004]) is a multi-robot simulator for outdoor environments. As Stage (part of the Player project, Gerkey et al. [2003]), it is capable of simulating a population of robots, sensors and objects. It generates both realistic sensor feedback and physically consistent interactions between objects. It includes an accurate simulation of rigid-body physics and allows the user to select between multiple dynamics engines (ODE, Bullet, SimBody Sherman et al. [2011] and DART Tech [2013]). Gazebo has been used to compare algorithms for navigation and grasping in a controlled environment.

Finally, two notable softwares are the OpenHRP project used in Japan for the HRP series (Kanehiro et al. [2004]) and MuJoCo (Todorov et al. [2012]) used for model-based control.

Our decision to add a YARP interface to Gazebo is motivated by the following considerations. We want to switch between fast, not accurate simulations and slow, accurate ones, thus we need the capability of choosing among different dynamic engines. We also want a simulator which is both easy to use and to expand in order to add new robot models. Finally, we prefer an open-source software with an active community and money investments. Gazebo fulfills our requirements, in particular it is expandable with a plugin structure: in this work our YARP interface is a collection of Gazebo plugins.

3 Structure

It is useful to understand Gazebo plugins and YARP device drivers before describing the structure of our plugins (from now on *gazebo_yarp_plugins*).

Gazebo plugins are C++ classes that extend the functionalities of Gazebo, while YARP device drivers are C++ classes used in YARP for abstracting the functionality of robot devices. Usually, each class of gazebo_yarp_plugins embeds a YARP device driver in a Gazebo plugin.

3.1 Gazebo Plugins

A plugin is a piece of code compiled as a shared library and inserted into the simulator. A plugin has direct access to all the functionalities of Gazebo from the physics engine to the simulated world. Furthermore, plugins are self-contained routines that are easily shared and can be inserted and removed from a running system. There are 4 types of plugins in Gazebo: **world**, **model** and **sensor** plugins are attached to and control a specific simulated world/model/sensor respectively, while **system** plugin is specified on the command line and loads during the Gazebo startup.

3.2 YARP Device Drivers

In YARP, a device driver is a class that implements one or more interfaces. There are three separate concerns related to devices in YARP:

- Implementing specific drivers for particular devices
- Defining interfaces for device families
- Implementing network wrappers for interfaces

For example the Control Board device driver implements a set of interfaces that are used to control the robot (IPositionControl, ITorqueControl, etc.) and another set of interfaces to read data from the motors (IEncoders, etc).

3.3 Gazebo-YARP Plugins

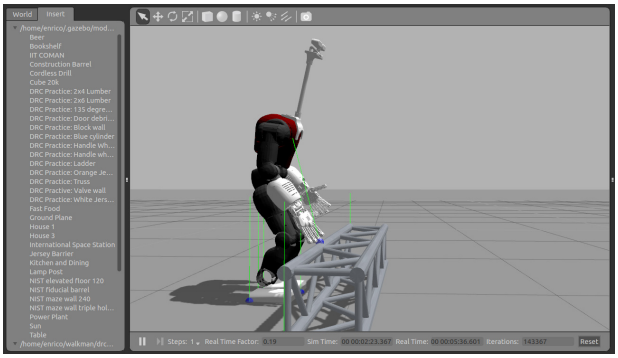
The gazebo_yarp_plugins is made of:

- Gazebo plugins that instantiate YARP device drivers,
- YARP device drivers that wrap Gazebo functionalities inside the YARP device interfaces.

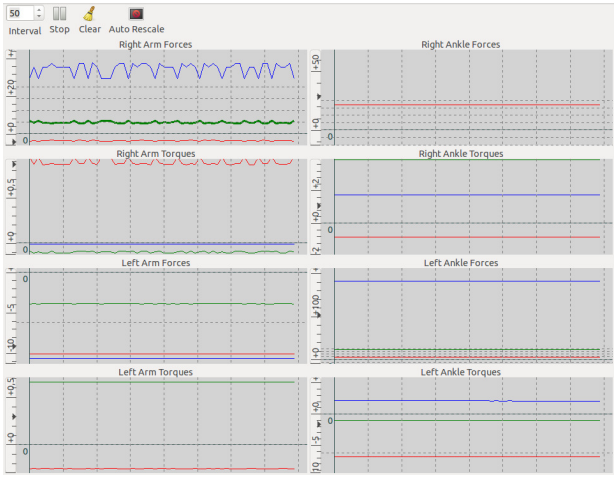
The plugins/devices already implemented are the *Control Board*, *6-axis Force Torque sensor*, *Inertial Measurement Unit* (IMU) and a *Clock* plugin used for synchronization. The first three plugins are directly related to the simulated objects and sensors, while the last one is a system plugin that synchronizes all the other YARP modules with the simulation time.

3.3.1 Control Board

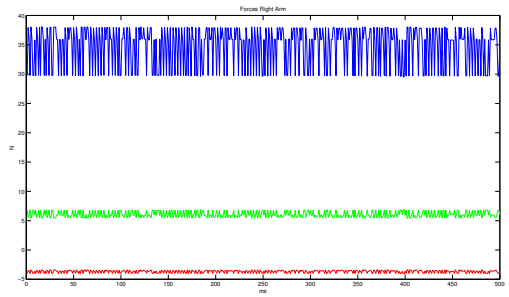
The Control Board plugin allows to control the robot using YARP Interfaces, it is implemented as a Gazebo Model plugin. Every control board allows the user to control one or more joints (a kinematic chain such as the arm or leg, etc.) as specified in a configuration file. For each controlled joint the control board opens different interfaces, permitting the use of different type of controllers for each joint. Such interfaces include position control, torque control, encoders reading, torque



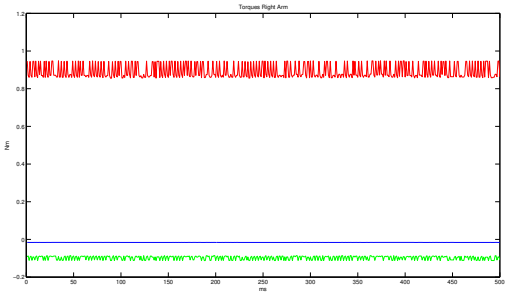
(a) Gazebo interface with COMAN



(b) A yarpscope showing on-line the forces and the torques at each Force/Torque sensor



(c) Plot of forces measured at the Force/Torque sensor placed on the right arm (data logged during simulation). Forces along x,y and z are respectively in red, green and blue



(d) Plot of torques measured at the Force/Torque sensor placed on the right arm (data logged during simulation). Torques along x,y and z are respectively in red, green and blue

Figure 3: A Gazebo simulation running COMAN interacting with a debris

measurement and joint impedance control. Usually the number of instantiated control boards is equal to the number of kinematic chains. Each control board, during every cycle of simulation, reads position, velocity and torque values from the simulated joints and sends desired joints position or torques to the simulator. The values read from the simulator are broadcasted through YARP interfaces in the YARP network, in a similar way the desired joint values come from YARP interfaces (Figure 4). The following YARP interfaces are used to control the robot.

- **IPositionControl**: a position control with a linear trajectory generator considering a max joint speed
- **IPositionDirect**: a position control using Gazebo position PIDs
- **ITorqueControl**: a perfect torque follower
- **IImpedanceControl**: a joint impedance control with the following law

$$\tau_d = -P_d(q - q_d) - D_d\dot{q} + \tau_{offset} \qquad (1)$$

where q_d is the desired equilibrium position, P_d is the desired joint stiffness and D_d is the desired joint damping. τ_{offset} is an extra signal that can be used for gravity compensation or inverse dynamics control.

Furthermore, the Control Board implements the **IControlMode** interface that allows to change the type of controller online. All these interfaces are also available on the robot and they have the same behaviour.

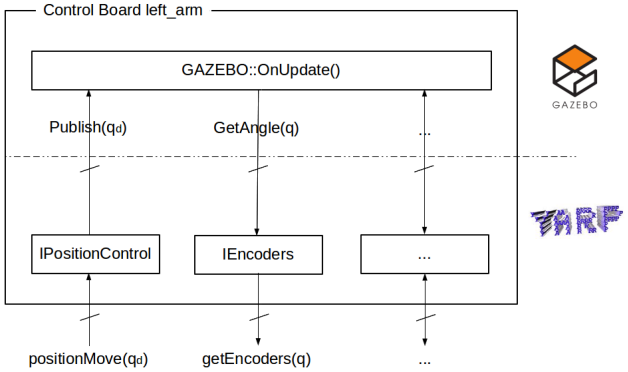


Figure 4: Control Board plugin for the left_arm kinematic chain. yarp::IPositionControl interface has a method positionMove() that can be used to set joint values inside a YARP module. The plugin implements such interface by calling the Publish() method inside the Gazebo API to move the simulated joints at each OnUpdate().

3.3.2 6-axis Force/Torque sensor

A Force/Torque sensor measures a wrench in the robot structure (Figure 3). The sensor, at the time of writing, is simulated in Gazebo as if it was attached to the

reference frame associated to a joint. On the YARP side, the reading of a generic sensor is implemented as a **IAnalogSensor** interface (Figure 5). The broadcasted data is a vector of six numbers representing the forces and the torques applied on that reference frame.

3.3.3 IMU sensor

An IMU measures velocity, orientation, and gravitational forces, using a combination of accelerometers and gyroscopes, of the link where it is placed. It is also possible to add white Gaussian noise on the measurement (Figure 7). Similar to the Force/Torque sensor, it is implemented as a **IAnalogSensor** interface.

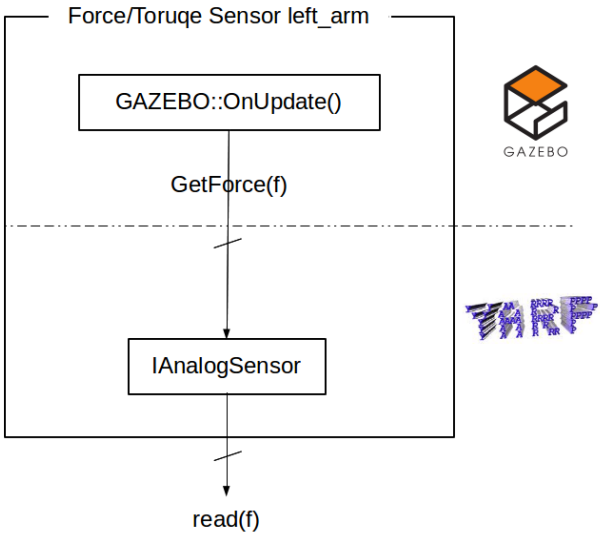


Figure 5: The Force/Torque sensor in the left arm is implemented as a YARP IAnalogSensor interface. At every step the internal state of the plugin is updated with the last readings of forces and torques from the simulation.

3.3.4 Clock

A fundamental aspect in simulations is the synchronization between YARP modules and the simulated robot. A YARP module is a process in which one or more threads are started. When such modules are used in the real robot, the thread rate is timed by the machine (system) clock, also called the *wall clock*. When the simulation is running we want the rate of such modules to be synchronized with the simulated time, otherwise the control loop could run faster or slower with respect to the simulated robot dynamics. The *real-time factor* (RTF) of the simulation is given by

$$RTF = update\_frequency \times step\_time \qquad (2)$$

and is kept to one when the desired update frequency is the inverse of the time increased at each step in the

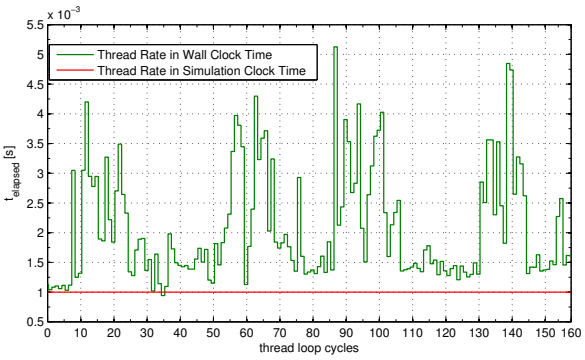
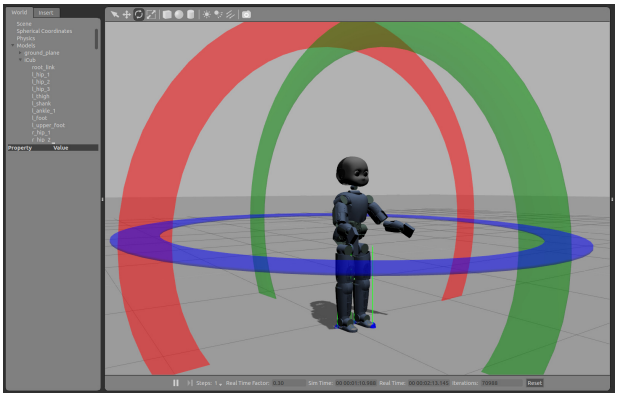


Figure 6: Time elapsed between each execution of the control loop, measured in simulation clock time and in wall clock time. Desired thread rate is 1kHz and simulation time step is 1ms

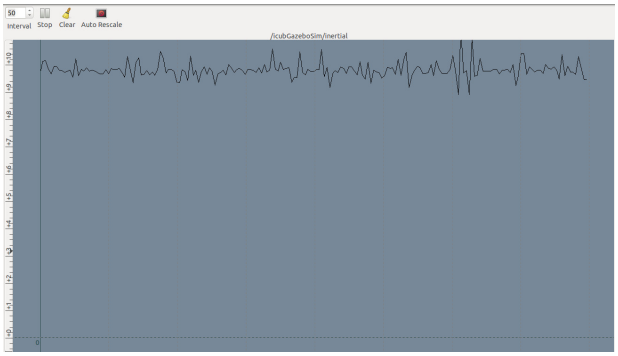
simulation. For instance if the simulation runs with a real time factor of 0.1, 10 seconds are needed to simulate 1 real second. Within this situation, the controller process should also be slowed down 10 times to be coherent with the simulation. To solve this issue we developed a *clock* plugin that synchronizes modules with the simulated time. The *clock* plugin is implemented as a System plugin and publishes on a YARP port the time information from the simulator. For every simulation step, the simulation time is incremented and the timestamp is sent via socket. All the YARP threads implemented as control threads (which need to be run at a desired rate, i.e. YARP's *RFModule* or *RateThread*) are automatically synchronized using the simulation clock if the *YARP_CLOCK* environment variables is set, or if the module explicitly asks to. The *yarp::os::Time* functionalities are also transparently working using the wall-clock or the simulation clock depending on the environment variable. Thread sleeps are performed using the right wall or simulated time. When synchronized with the simulation clock the *yarp::os::Time* delay does not explicitly sleep on a wall clock, rather a scheduler is synchronized with the simulation clock by performing blocking reads on the *YARP_CLOCK* port. This scheduler wakes up the threads that required a delay just once, when they have slept for the desired duration. Compared to the ROS::Time implementation which uses small sleeps on wall clock to check synchronization with the simulated clock, this allows to run simulations both slower and faster than real time and still have synchronization between threads and controls. In any case, when accessing the simulated clock Experiments showed the approach to be successful in synchronizing 1kHz control loops against simulations running 1kHz, thus having a 1ms clock granularity.

3.3.5 Simulation Description Format (SDF)

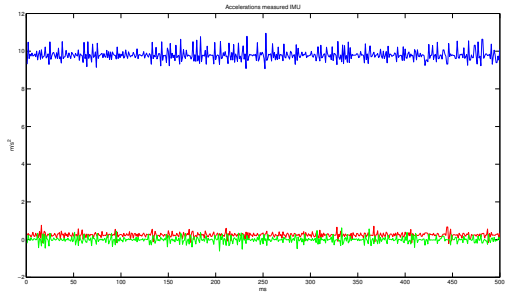
Gazebo uses an XML-style format, Simulation Description Format (SDF), to save and load information



(a) Gazebo interface with iCub



(b) A yarpscope showing on-line the acceleration measured by the simulated IMU along z



(c) Plot of accelerations measured by the IMU (data logged during simulation). Accelerations along x,y and z are respectively in red, green and blue

Figure 7: iCub inside Gazebo.

about a simulated world or model. An SDF encapsulates all the necessary information for a simulation such as:

- **Scene**: ambient lighting, sky properties, shadows.
- **Physics**: gravity, time step, physics engine.
- **Models**: collection of links, collision objects, joints, and sensors.
- **Lights**: point, spot, and directional light sources.
- **Plugins**: world, model, sensor, and system plugins.

Our Control Board, Force Torque sensor and IMU plugins are included inside the SDF file that describes our robots. For our humanoid bipedal robot, COMAN, we have five Control Board plugins (one for each kinematic chain), four Force/Torque sensor plugins (two in the legs and two in the arms) and one IMU sensor plugin (placed on the back of the waist). The SDF descriptions of COMAN and iCub are available in `https://github.com/EnricoMingo/iit-coman-ros-pkg` and in `https://github.com/robotology-playground/icub_gazebo` respectively. The clock plugin is loaded trough a command line parameter when the simulator is started.

4 Conclusions

In this work we have presented a set of Gazebo plugins, named *gazebo_yarp_plugins*, that allow to connect the robotics framework YARP to Gazebo itself. Gazebo was chosen since it is easy to use, it has the possibility to switch between different rigid multi-body dynamics engines, it is Open-Source and has an active community. Our plugins are based on YARP device drivers in order to have exactly the same interfaces in the real and simulated robot. This allows to write modules that will work both in the simulator and in the real robot without the need to change the code. This is a very important paradigm in robotics research and develop since it minimizes the presence of errors due to code porting. Furthermore the simulator becomes a tool that helps the developer in testing and validation before using the real platform. Such plugins consist in: a Control Board plugin to control the robot, a Force Torque sensor plugin and an IMU plugin. A special plugin dedicated to synchronization between modules and simulator was also implemented. The plugins were tested to simulate two humanoid bipedal robots, the COMAN and the iCub, both from the Italian Institute of Technology.

5 Future Works

Gazebo_yarp_plugins is a project at an early stage that is gaining more and more interest inside the YARP community. Beside the good results obtained up to now, some works are still missing in order to be able to have 100% compatibility with all the Gazebo functions. Furthermore we still need to implement plugins to connect YARP device drivers dedicated to cameras and RGB-D sensors to the simulated ones in Gazebo. We are also interested in multi-robot and human-robot simulation: we already have the possibility to easily simulate different robot models but it is still difficult to simulate multiple instances of the same robot. Furthermore, since our robots in IIT have flexible joints, we are investigating on how to simulate flexible joints without specifying extra joints/links inside the SDF of the robot. Finally we are planning an official release of our plugins inside the Gazebo community.

References

D. Calisi, A. Censi, L. Iocchi, and D. Nardi. OpenRDK: a modular framework for robotic software development. In *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, pages 1872–1877, September 2008. ISBN 978-1-4244-2057-5. doi: 10.1109/IROS.2008.4651213.

DARPA. Darpa robotics challenge, 2013. URL `http://www.theroboticschallenge.org/`.

Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

Coumans Erwin. Bullet, 2003. URL `http://www.bulletphysics.org/`.

Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. ISBN 0387743146.

Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *In Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

Serena Ivaldi, Vincent Padois, and Francesco Nori. Tools for dynamics simulation of robots: a survey based on user feedback. *CoRR*, abs/1402.7050, 2014.

Fumio Kanehiro, Hirohisa Hirukawa, and Shuuji Kajita. Openhrp: Open architecture humanoid robotics platform. *I. J. Robotic Res.*, 23(2):155–165, 2004.

Nathan Koenig and Andrew Howard. 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*, volume 3, pages 2149–2154, 2004.

G. Metta, P. Fitzpatrick, and L. Natale. Yarp: Yet another robot platform. *International Journal of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics*, 3(1), 2006.

Olivier Michel. Cyberbotics ltd. webots tm : Professional mobile robot simulation. *Int. Journal of Advanced Robotic Systems*, 1:39–42, 2004.

OSRF. Open source robotics foundation, 2011. URL `http://osrfoundation.org/`.

Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *IROS*, pages 1321–1326. IEEE, 2013.

Michael A. Sherman, Ajay Seth, and Scott L. Delp. Simbody: multibody dynamics for biomedical research. *Procedia {IUTAM}*, 2(0):241 – 261, 2011. ISSN 2210-9838.

Russel Smith. Open dynamic engine, 2000. URL `http://www.ode.org/`.

Georgia Tech. Dart, 2013. URL `http://dartsim.github.io/`.

V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori. 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*, PerMIS '08, pages 57–61, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-293-1. doi: 10.1145/1774674.1774684. URL `http://doi.acm.org/10.1145/1774674.1774684`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5.