

qiBullet, a Bullet-based simulator for the Pepper and NAO robots

Maxime Busy¹ and Maxime Caniot²

Abstract—The Pepper and NAO robots are widely used for in-store advertizing and education, but also as robotic platforms for research purposes. Their presence in the academic field is expressed through various publications, multiple collaborative projects, and by being the standard platforms of two different RoboCup leagues. Developing, gathering data and training humanoid robots can be tedious: iteratively repeating specific tasks can present risks for the robots, and some environments can be difficult to setup. Software tools allowing to simulate complex environments and the dynamics of robots can thus alleviate that problem, allowing to perform the aforementioned processes on virtual models. One current drawback of the Pepper and NAO platforms is the lack of a physically accurate simulation tool, allowing to test scenarios involving repetitive movements and contacts with the environment on a virtual robot. In this paper, we introduce the qiBullet simulation tool, using the Bullet physics engine to provide such a solution for the Pepper and NAO robots.

I. INTRODUCTION

The use of the Pepper and NAO robots as research platforms is widely spread across various academic fields. Moreover, each of these robot is a standard platform in a league of the Robocup[1] competition (the Robocup@Home [2] league for Pepper, and the Robocup soccer [3] for NAO). Additionally, the Pepper and NAO robot are respectively standard platforms in the World Robot Summit [4] competition and in the NAO Challenge [5]. In the recent years, tremendous progresses have been achieved in the robotics field through machine learning approaches, and in particular data-driven approaches such as Deep Learning and Reinforcement Learning. Such approaches often compel the developer to gather data by iteratively repeating specific tasks with a robot. Gathering this data in a simulation able to handle complex environments and the dynamics of the simulated robot would alleviate the data gathering task, and prevent the real robotic platform from being damaged. Such a simulation would also allow to identify potential problems in a scenario including Pepper or NAO before deploying it into the real world.

Presently, Pepper and NAO models are available in simulation tools such as Gazebo [7], V-REP [8], Webots [9], or Choregraphe [10]. These implementations either lack the ability to accurately handle the physics of the model or to simulate complex environments. More recently, a Morse-based [11] simulation for the Pepper robot [12] has been announced, but focuses on human-robot interactions and not on reliable physics.

¹Maxime Busy is with the Innovation Department of SoftBank Robotics Europe, France. maxime.busy@softbankrobotics.com

²Maxime Caniot is with the Innovation Department of SoftBank Robotics Europe, France. maxime.caniot@softbankrobotics.com

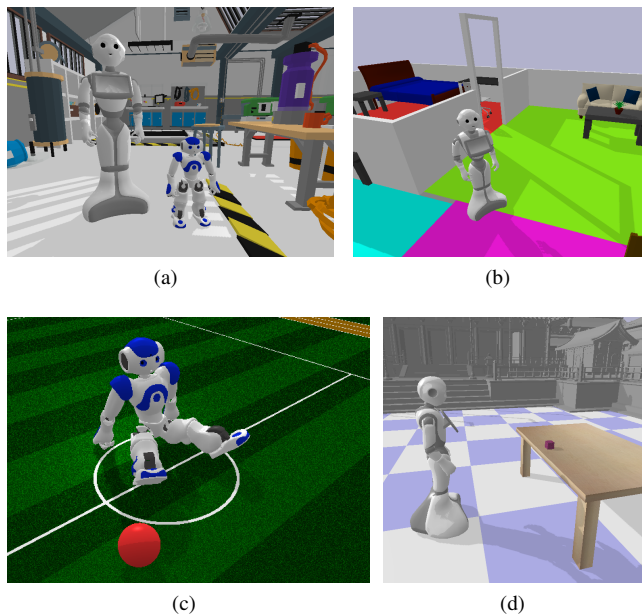


Fig. 1: (a) Pepper and NAO robots in the botlab environment [6]. (b) Pepper robot in the Montreal arena of the Robocup@Home. (c) NAO robot in a soccer environment. (d) Pepper robot in a grasping environment.

In this paper, we introduce qiBullet, an open-source simulation tool based on the Bullet physics engine [13] and the PyBullet module [14], designed to answer the aforementioned problems. This simulator aims to provide a cross-platform and transparent mean to embed a virtual Pepper or NAO robot in different environments (Figure 1), via a Python-based API mimicking NAOqi [15] or a ROS interface emulating the naoqi_driver¹ ROS [16] package. In order to describe our work, we first specify the strong ties between qiBullet and the Bullet physics engine. We then describe the simulator itself, its interfaces, the components of a virtual robot model and its control and sensing capabilities, along with scenarios showcasing the tool. Lastly, we discuss the exploitation and availability of the simulator.

II. PHYSICS ENGINE

We based our approach on a comparison of physics-based simulation libraries proposed by Erez *et al.* [17], discussing the differences between the Bullet, Havok [18], MuJoCo[19], ODE[20] and Physx[21] physics engine.

¹http://wiki.ros.org/naoqi_driver

To build our simulator, we chose the Bullet physics engine and the additional PyBullet module. The physics engine is integrated with many of the popular robotics software platforms, such as V-REP and Gazebo, and presents the advantage of being open-source. This engine can additionally be extended with PyBullet, an open-source Python module providing robotics and machine learning capabilities [22] [23] [24] [25] [26].

The qiBullet simulation has been designed to inherit the cross-platform properties of the PyBullet Python module and Bullet physics engine: the simulation tool can be run on Linux, Windows and MacOS.

III. SIMULATOR

In this section, we will detail how a virtual robot is defined in the qiBullet simulator.

A. Robot model

We use a Unified Robot Description Format (URDF) [27] file to describe the model of a virtual robot. The different links of the model, their masses, inertia matrices and the joints connecting them are extracted from this file by the engine. Mesh files are associated to each link, allowing the engine to render the visual aspect of the robot model and to perform collision checking.

B. Interface

Two different ways of interacting with the robot virtual model are proposed in the simulation: The Python-based qiBullet API or the ROS Framework.

The Python-based qiBullet API, built on top of the Pybullet API, allows the user to interact with the simulated robot and more generally with the simulated environment. To ease the integration into existing projects and ensure code consistency, the qiBullet API mimics a part of the NAOqi API, rendering the interaction with a virtual and a real robot as transparent as possible.

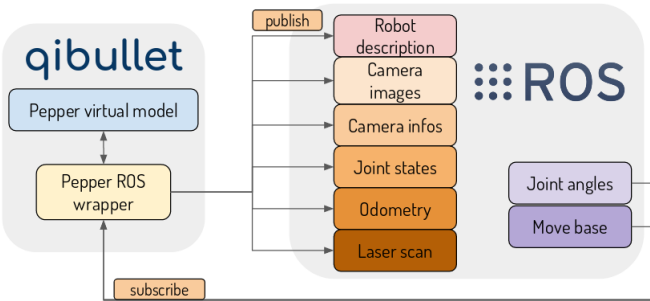


Fig. 2: Interfacing of a Pepper virtual model with the ROS framework. The wrapper specifies the description of the robot, publishes the sensory data of the model and subscribes to control topics.

The qiBullet simulator also provides a "ROS wrapper" built on top of the API, emulating the naoqi_driver ROS package and allowing the user to interact with the virtual

robot model through the ROS framework. As described in Figure 2, the wrapper retrieves the sensory data of the model to publish it into the framework, and retrieves commands from the framework to apply them onto the model.

In order to facilitate the use of the simulator for machine learning tasks, the API can also be used to instantiate, reset, and stop several independent simulated instances, running in parallel. It can also be used to spawn or remove virtual robots from an instance, and to control the position of the light source of the simulation.

C. Control and Sensing

This subsection illustrates the control and sensing capabilities of the Pepper and NAO virtual robots in the qiBullet simulator.

The joints of the models can be controlled independently or as groups to reach a specified articular position with a specified speed. Several postures can be applied onto the models, similar to the postures defined within the NAOqi API. Additionally, the base of the Pepper virtual robot can be controlled in position or in velocity.



Fig. 3: Synthetic RGB image retrieval from the top camera of a virtual Pepper, in a simulated environment. The obtained RGB image is displayed in the bottom left corner, using OpenCV.

Each Pepper and NAO virtual model embeds two RGB cameras (Figure 3). The Pepper virtual model additionally embeds a depth camera. Similarly to the NAOqi API, the resolution of the retrieved synthetic images can be selected by the user. The parameters of the simulated cameras are tuned to match the ones of the real model, although it is important to point out that a real depth image from Pepper will be more noisy than a synthetic one. The Pepper virtual model also possesses laser sensors attached to its base, mimicking the lasers of the real robot. Finally, the position of each model in the world frame can be directly retrieved from the qiBullet API, providing odometry information. In our simulation, the odometry drift is not simulated. The API can also provide collision data for a link or a group of links of the desired virtual model.

D. Scenarios

In this subsection, we present three different scenarios showcasing the qiBullet simulator.

a) Workspace computation: We aim to sample the right and left arm workspaces of the Pepper robot, and to evaluate the manipulability [28] of each sampled configuration. The kinematic chains start at the Tibia link and respectively end at the right and left gripper links. To do so, we instantiate 10 simulation instances. In each simulation and for each iteration, an articular position is randomly defined and applied onto the joints of a chain. If the chain is self colliding with the model, the position is deemed incorrect, and another position is computed. If the end effector does not self collide, the reached position and the corresponding manipulability are added to the workspace. When all of the instances have reached 4000 successful iterations for each arm, the results are combined and normalized with respect to the maximum manipulability value obtained to generate a workspace containing 40000 elements for each kinematic chain (see Figure 4(a)).

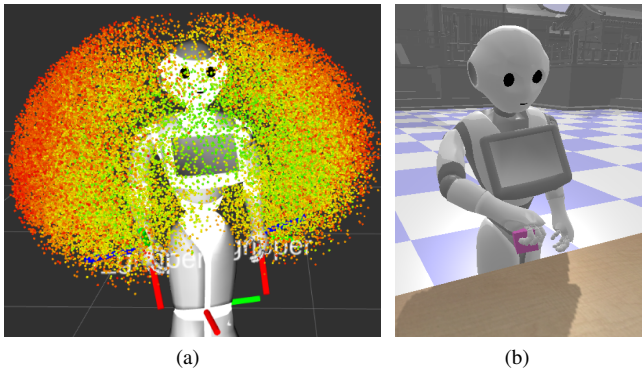


Fig. 4: (a) RViz display of the right and left arm workspaces of the Pepper robot, with kinematic chains starting from the Tibia link, and respectively ending at the right and left gripper links. The color of each point represents the associated normalized manipulability value: green corresponds to the maximum manipulability value, while red corresponds to the minimum. (b) Grasping scenario with a virtual Pepper robot: the pink cube in the right hand of Pepper is the object to be grasped.

b) Grasping task: We define a grasping scenario, where Pepper is placed in front of a table on which an object to be grasped is positioned. The physical properties of the Bullet physics engine allow to virtually test out such a scenario with the Pepper virtual model (see Figure 4(b)).

c) Walking task: We define a walking scenario, where NAO is placed standing still in a flat world. The simulator can be used to test the balance of the robot while being controlled with different walking algorithms (see Figure 5).

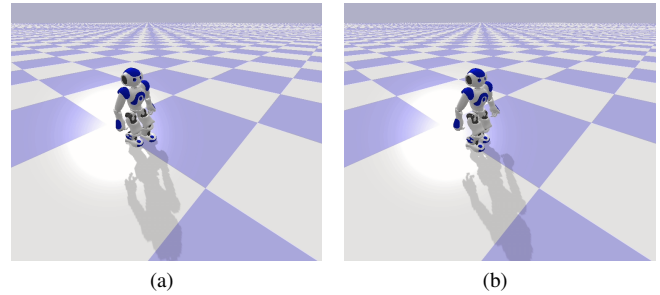


Fig. 5: (a) and (b) Virtual NAO robot being controlled by a walking algorithm.

The use of the qiBullet simulator can be extended beyond these scenarios, for instance to generate/extend datasets with synthetic data [29], to train Reinforcement Learning algorithms or to perform localization and navigation tasks.

IV. EXPLOITATION

In order to foster the use of the qiBullet simulator, its code is made open-source and is available on Github². The qiBullet repository contains the files defining the Pepper and NAO robot models, the Python-based qiBullet API, examples illustrating how to use the simulator, an automatically generated documentation, and unit tests. The unit tests are automatically launched by a continuous integration tool³ when the simulation is updated, in order to ensure the stability of the library. Moreover, the qiBullet⁴ Python package has been created and is updated after each new release of the simulator: this particular packaging allows a simple installation of our simulator and of its dependencies.

V. CONCLUSION

In this paper we introduce qiBullet, a simulator based on PyBullet and the Bullet physics engine, aiming to virtually emulate SoftBank Robotics' robots in a physically accurate fashion. The simulator can sustain multiple instances running in parallel, provides a Python API to interact with the sensors and actuators of the simulated models, and can be interfaced with the ROS framework. In an effort to open the simulator to the community, its code has been made public and is hosted on Github. The work on the qiBullet simulator is still ongoing, we envision introducing additional features and enhancing the existing robot models.

REFERENCES

- [1] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. 1995.
- [2] Thomas Wisspeintner, Tijn Van Der Zant, Luca Iocchi, and Stefan Schiffer. Robocup@ home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, 10(3):392–426, 2009.

²<https://github.com/softbankrobotics-research/qibullet>

³<https://travis-ci.org/softbankrobotics-research/qibullet>

⁴<https://pypi.org/project/qibullet/>

- [3] Eric Chown and Michail G. Lagoudakis. The standard platform league. In *RoboCup 2014: Robot World Cup XVIII [papers from the 18th Annual RoboCup International Symposium, João Pessoa, Brazil, July 15]*, pages 636–648, 2014.
- [4] World Robot Summit. [Online], Available: <https://worldrobotsummit.org/en/>.
- [5] NAO Challenge. [Online], Available: <https://www.naochallenge.it/>.
- [6] Alan Zimmerman. Botlab environment. [Online], Available: <https://poly.google.com/view/2hWxc7Hk9CJ>.
- [7] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.
- [8] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.
- [9] Olivier Michel. Cyberbotics ltd. webots: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.
- [10] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 46–51. IEEE, 2009.
- [11] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: Morse. In *2011 IEEE International Conference on Robotics and Automation*, pages 46–51. Citeseer, 2011.
- [12] Florian Lier and Sven Wachsmuth. Towards an open simulation environment for the pepper robot. In *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 175–176. ACM, 2018.
- [13] Erwin Coumans. Bullet physics engine. *Open Source Software: http://bulletphysics.org*, 1(3), 2010.
- [14] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016–2019.
- [15] Aldebaran Robotics. Naoqi framework. [Online], Available: http://doc.aldebaran.com/2-5/index_dev_guide.html.
- [16] Morgan Quigley, Ken Conley, Brian 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*, volume 3, page 5. Kobe, Japan, 2009.
- [17] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.
- [18] Havok physics engine. [Online], Available: www.havok.com.
- [19] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. [Online], Available: www.mujoco.org.
- [20] Open Dynamics Engine. [Online], Available: <http://ode.org>.
- [21] PhysX physics engine. [Online], Available: www.geforce.com/hardware/technology/physx.
- [22] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.
- [23] Michel Breyer, Fadri Furrer, Tonci Novkovic, Roland Siegwart, and Juan Nieto. Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 4(2):1549–1556, 2019.
- [24] Krzysztof Choromanski, Aldo Pacchiano, Jack Parker-Holder, Jasmine Hsu, Atil Iscen, Deepali Jain, and Vikas Sindhwani. When random search is not enough: Sample-efficient and noise-robust blackbox optimization of rl policies. *arXiv preprint arXiv:1903.02993*, 2019.
- [25] Eloïse Dalin, Pierre Desreumaux, and Jean-Baptiste Mouret. Learning and adapting quadruped gaits with the “intelligent trial & error” algorithm. 2019.
- [26] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *arXiv preprint arXiv:1903.11239*, 2019.
- [27] Willow Garage. Universal robot description format (urdf). <http://www.ros.org/urdf/>, 2009, 2009.
- [28] Tsuneo Yoshikawa. Manipulability of robotic mechanisms. *The international journal of Robotics Research*, 4(2):3–9, 1985.
- [29] Alban Laflaquière and Verena V Hafner. Self-supervised body image acquisition using a deep neural network for sensorimotor prediction. *arXiv preprint arXiv:1906.00825*, 2019.