ROS-PyBullet Interface: A Framework for Reliable Contact Simulation and Human-Robot Interaction

Anonymous Author(s)
Affiliation
Address
email

Abstract: Reliable contact simulation plays a key role in the development of (semi-)autonomous robots, especially when dealing with contact-rich manipulation scenarios, an active robotics research topic. Besides simulation, components such as sensing, perception, data collection, robot hardware control, human interfaces, etc. are all key enablers towards applying machine learning algorithms or model-based approaches in real world systems. However, there is a lack of software connecting reliable contact simulation with the larger robotics ecosystem (i.e. ROS, Orocos), for a more seamless application of novel approaches, found in the literature, to existing robotic hardware. In this paper, we present the ROS-PyBullet Interface, a framework that provides a bridge between the reliable contact/impact simulator PyBullet and the Robot Operating System (ROS). Furthermore, we provide additional utilities for facilitating Human-Robot Interaction (HRI) in the simulated environment. We also present several use-cases that highlight the capabilities and usefulness of our framework. Please check our video, source code, and examples included in the supplementary material.

1 Introduction

Dealing with contacts is a key requirement for robots to become effective in our daily lives and valuable assets in industry [1]. Examples of contact-rich tasks include pick and place [2], locomotion [3, 4, 5], wiping [6], pushing [7, 8], dyadic co-manipulation [9], and robot surgery [10, 11]. Developing approaches for (semi-)autonomous robots involving contact is thwart with practical issues (e.g. slippage, high impulsive forces, model mismatch, etc.) that may cause failure and damage to the robot or yield safety concerns for humans in close proximity.

Development of machine learning approaches require a facility to collect large datasets. However, collection on scale is non-trivial. Learning from demonstration [12] is a popular technique for endowing robots with new skills where a human provides examples. According to one paradigm, *kinaesthetic teaching* (e.g. [13]), the human interacts directly with the robot. However, this method requires a physical system which is cumbersome and has potential safety concerns. A second approach utilizes *teleoperation* (e.g. [14]), which has the benefit that the human operator can either interact with a simulated robot or at a distance with the physical system (ensuring safety). Two key considerations for our framework are that the virtual world can include a human model via telepresence and that the human operator can experience the virtual forces generated by the simulator, through interface to haptic devices. Furthermore, to ease issues when deploying methods on robot hardware, we provide software features to easily map targets to the robot control commands.

In robotics, the system's complexity, need for several sub-processes, and multi-machine operations motivate a modular system design and message parsing functionality [15, 16, 17]. There are many packages for the Robot Operating System (ROS) [16] integrating useful functionalities for research and commercial systems, including useful data structures, control interfaces, inverse kinematics (IK) and motion planning, perception tools, etc. [18, 19, 20, 21, 22]. Additionally, various visualizers [23, 24, 25] and physics simulators [26, 27, 28, 29, 30] allow for the development and testing of algorithms, prior to execution on robot hardware. However, popular and reliable libraries for contact-rich scenarios, such as PyBullet [29], Drake [28], and MuJoCo [27], lack integration with the ROS framework.

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

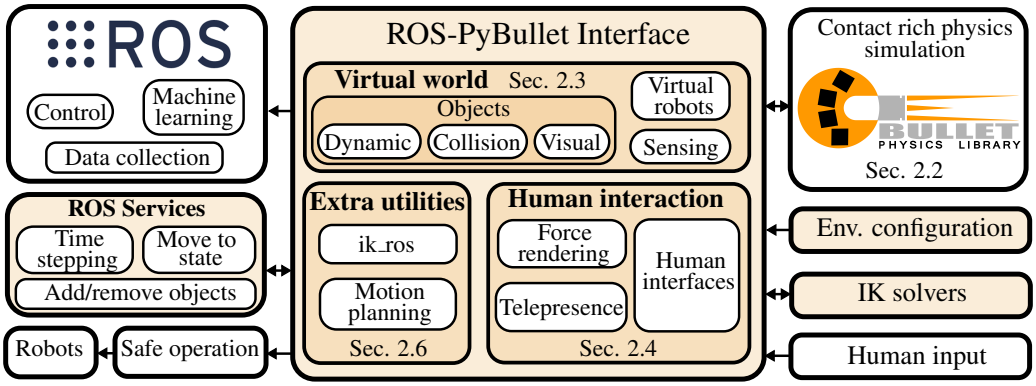


Figure 1: Outline of the proposed framework for bridging a reliable contact simulator within the ROS ecosystem, and additional HRI interfaces. Colored boxes indicate our framework.

1.1 Contributions

In this paper, we propose

- A framework (Figure 1) for simulating manipulation scenarios, allowing for seamless collection of contact-rich data and human demonstrations within a simulated environment. The framework includes
  - full physics simulation utilizing PyBullet,
  - sensor simulation (i.e. robot joint force-torque and point cloud),
  - integration with the ROS ecosystem,
  - several teleoperation interfaces (e.g. keyboard, mouse, joystick, 6D-mouse, haptic device) and robots (e.g. Kawada Nextage humanoid, KUKA LWR robot arm, Talos humanoid, Kinova robot arm, and dual-arm KUKA IIWA), and
  - easy integration with robot hardware.
- Several use-cases to demonstrate the capabilities and usefulness of the framework, and full documentation (see supplementary material).

Our framework exploits modularity and extensibilty by implementing several ROS nodes and object types using class hierarchy design patterns. Furthermore, we include additional features such as: utilities for Model Predictive Control (MPC) design and development, motion planning, safe robot operation (for ensuring safety limits are satisfied before commanding the real robot), and inverse kinematics. Several recent works use the proposed framework [31, 32, 33, 34, 35].

2 Proposed Framework

To enable easy prototyping, implementation, and integration with hardware we implement several tools/features in the framework. This section describes these and our design decisions.

2.1 Framework features

Figure 1 shows an overview for our framework highlighting it as a central interface between the ROS ecosystem and PyBullet, human interaction, IK solvers, and real robots. The main features of the framework are listed as follows.

1. **Online, full-physics simulation using a reliable contact simulator.** The framework relies on PyBullet to enable well established contact simulation for rigid/deformable bodies.
2. **Integration with the ROS ecosystem.** Robot simulation and visualization of real robots/objects (utilizing sensing) are integrated via ROS. Furthermore, this enables (i) ROS packages to be integrated with PyBullet and (ii) a straightforward way to port developed algorithms to real systems.

3. **Several interfaces enabling HRI with virtual worlds and telepresence.** We provide facilities for human's to provide examples in a simulated environment via several popular interfaces (including haptic devices).
4. **Modular and extensible design.** Our framework adopts a modular (i.e. several ROS nodes) and highly extensible design paradigm (i.e. class hierarchy) using the Python programming language. This makes it easy to quickly develop new features for the framework.
5. **Data collection with standard ROS tools.** Since the framework provides an interface to ROS, we can leverage common tools for data collection such as ROS bags [36] and data processing to common formats in machine learning applications, i.e. `rosbag_pandas` [37].
6. **Integration with robot and sensing hardware.** Tools are provided to easily remap the virtual system to physical hardware and integrate real sensing apparatus in the PyBullet simulation (e.g. vicon).

2.2 Full-Physics Contact-rich Simulation

Several simulators exist for full-physics simulation; e.g. Gazebo[1] with ODE [25, 26], PyBullet [29], Drake [28], and MuJoCo [27]. Reliably simulating contact between several bodies complicates the models. The simulators PyBullet, Drake, and MuJoCo have been identified as reliable for modeling real world impacts [38].

We chose to use PyBullet [29] since it is free and open source, a well-known library with an active community, easy to install, well documented, and is in Python. To ensure our framework is extensible, we develop several classes that interface with PyBullet and establish communication links with ROS utilizing publishers, subscribers, timers, and services.

2.3 Robots and virtual worlds building

Our framework provides several tools to build virtual worlds. The main ROS-PyBullet Interface node must specify a main YAML configuration file containing a list of robots/objects to load into PyBullet, parameters, RGB-D sensor configuration, and visualizer options - robots/objects can also be added/removed through ROS services. Various examples of robots/tasks are shown in Figure 2. Several object types (Section 2.3.1) were developed with different interaction properties and various communication channels with ROS. The specification for each object is defined in separate YAML files. See the documentation provided in the supplementary material for a full list of parameters for the configuration files, ROS topics published/subscribed, and ROS services provided.

2.3.1 PyBullet Objects

Robot Incorporating robots in our framework is simple. In a configuration file the user will specify the URDF file name, and several parameters (e.g. base position, inital joint configuration, etc). The robot base frame with respect to the world frame (named `rpbi/world`) is set by a transform broadcast using the ROS TF library [18]. The interface optionally publishes the robot joint/link states to ROS. Mobile and floating-base robots are also supported by our framework given an initial state (i.e. pose and linear/angular velocity). ROS services expose PyBullet's IK features, allow the user to move the the robot to a given joint/end-effector state[2], and return robot information (e.g. joint/link names, number of degrees of freedom, etc).

Visual robot A visualization of a robot, that does not interact with the PyBullet environment, can be instantiated by setting the `is_visual_robot` parameter to `True` in the robot configuration file. The main utility of this feature is to allow the user to map a real robot state to the PyBullet environment enabling them to easily compare the real robot with a simulated robot representing the target configuration. Only the robot information and IK services are available for visual robots.

Visual object A common requirement for simulators is to visualize objects. In PyBullet, these objects do not affect other bodies in the scene, nor react to those. Visual objects are listed in the

[1] An incomplete Bullet interface for Gazebo exists. To the best of the authors knowledge, it is deprecated.
[2] When an end-effector state is given, the corresponding joint state is found using PyBullet's IK features.

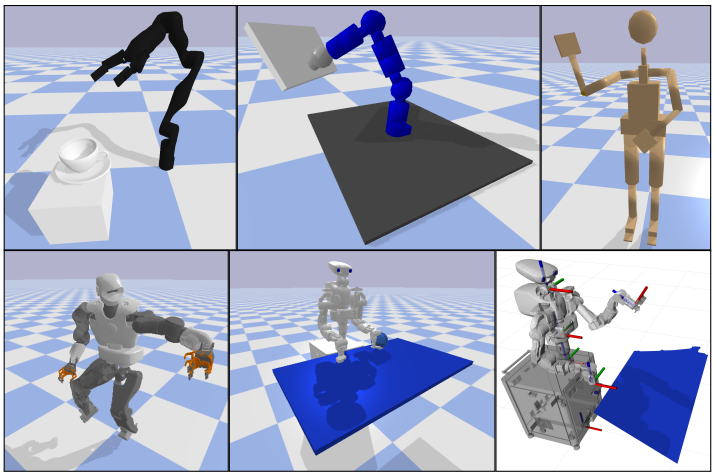


Figure 2: Examples of the ROS-PyBullet Interface: Top-row (left-right): a Kinova arm reaching for a cup, a Kuka LWR arm simulating a wiping task, and a human model waving. Bottom-row (left-right) a Talos humanoid robot reaching for a target while maintaining balance, the Nextage robot simulating a smart factory scenario, and simulated RGB-D data visualized in RViz.

main configuration file under the parameter `visual_objects`. A visual object was used to visualize the real pushing box (tracked with Vicon) in [33].

Collision object Modeling static objects, that cause momentum changes for other bodies upon collision, such as floors, ceilings, and walls are often necessary. These objects are listed under the parameter `collision_objects` in the main configuration. During the development of the experiments presented in [31], a collision object was used to represent a surface.

Dynamic object Objects whose pose evolution is completely determined by the simulator's physics engine are a key requirement for development of control algorithms. Dynamic objects, listed as `dynamic_objects` in the main configuration file, can be used inside PyBullet to simulate an object. A simulated pushing box was used to develop the controller in [33].

Soft object All previously described object types are rigid bodies. We also provide an interface to PyBullet deformable objects. These are similar to dynamic objects in that their evolution is defined by PyBullet and can be specified using the `soft_objects` parameter.

Load from URDF Finally, robots and objects can also be loaded directly into the PyBullet environment using the `urdfs` parameter. The evolution of these objects are defined by PyBullet. However, since the usage of these objects is ambiguous their communication with ROS is limited.

2.3.2 Sensor simulation

Many control, and planning algorithms rely on sensory feedback. Integrating multiple sensory inputs, such as tactile and vision, is underdeveloped in robotic manipulation [39]. To enable future research in realistic simulated environments, we provide an interface to several sensing modalities.

F/T sensor Through the configuration file for the robot it is possible to instantiate a simulated force-torque sensor attached to any joint on the robot. These virtual sensors publish joint reaction forces, read from PyBullet, as ROS wrench-stamped messages at a user-defined sampling frequency.

RGB-D camera Color and depth perception is a key sensing capability for object state estimation in contact-rich manipulation tasks. An RGB-D camera can be instantiated and attached to a frame through the main configuration file.

The color and depth images (`image`) are published together with the intrinsic camera parameters (`camera_info`). The camera intrinsic parameters are derived from the OpenGL projection matrix and can be used to back-project the images to a colored point cloud (Figure 4). This is natively supported in ROS, e.g. via the RViz `DepthCloud` plugin or via the `rgbd_launch` package. Optionally, the interface can compute and publish the point cloud data directly.

On a discrete GPU (NVIDIA GeForce GTX 1650 Mobile) this will achieve about 27 Hz, while on an integrated GPU (Intel UHD Graphics 630) this will reduce to about 18 Hz.

(a) (b) (c)

Figure 4: RGB-D observation of a scene with two objects: (a) colour image, (b) depth image, (c) projected colour point cloud in RViz.

2.4 Human interaction

Developing contact-aware algorithms is a key aspect of future work for the robotics community [39, 40]. Clearly there are safety concerns for HRI tasks. Simulation is a necessary step in any development cycle involving robots. This means the only way a human can interact with the virtual environment is through some interface (e.g. haptic device). To remedy this issue, we have developed a plugin for several human interfaces so that the human can be realized in the virtual environment and also receive virtual feedback from that environment.

Operator node Several common human interfaces and their drivers are provided in our framework out-of-the-box, these include: keyboard, mouse, joystick, space mouse, and a haptic device. Mapping the interface signals to a control space is non-trivial [41]. Choosing this mapping is simplified by allowing the user to specify the teleoperation mode in the launch file. We provide ROS nodes in the `operator_node` package that take raw interface signals as input and maps them to a control space of the users specification.

Logging signals There are several advantages for an intermediary node mapping driver signals to operator commands. First, modularity allows the user to easily swap out interfaces/mappings to compare modes. Second, methods utilizing moving horizon estimation (e.g. [31]) need to track a window of signals. We provide a node that enables this functionality in the `operator_interface_logger` node. Third, such a structure enables collection/comparison of data streams using ROS bags removing the need for extensive post-processing.

Scale node Several systems utilize joysticks, often the scaled value of a joystick axis defines velocity in certain dimensions [42, 43]. We provide `scale_node.py` that appropriately orders and scales the interfaces axes.

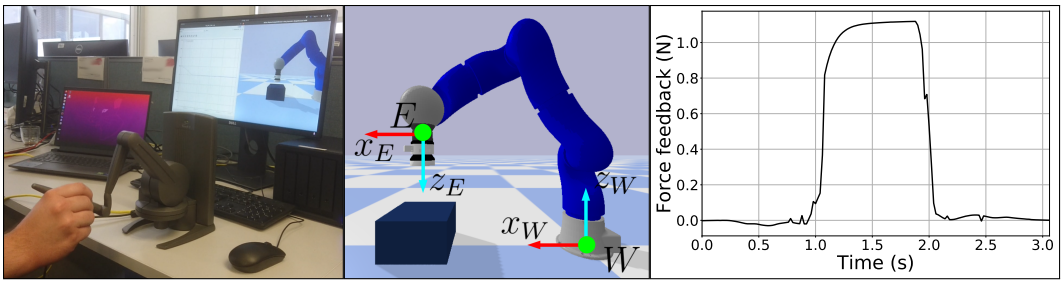
Isometric node Point-mass systems are common models for teleoperation, e.g. [31, 32]. The individual axes of the interface is often in the range $[-1, 1]$. If we scaled the joystick axes, as before, then the magnitude of the maximum velocity is non-uniform for all interface states. We provide `isometric_node.py` that ensures the maximum velocity magnitude is isometric.

2.5 Interfacing with real hardware

Interfacing with hardware is straightforward using our framework. Each object can publish/broadcast its state in several formats (e.g. joint states, float arrays, transforms, wrenches). This means the simulator can act, at development time, as the real system. When porting to the real hardware we provide nodes that remap the current system states to the required robot/hardware drivers: see the `remap_joint_state_to_floatarray` and `remap_joint_state` nodes in the `custom_ros_tools` package. Setup for these is as simple as remapping topics in a launch file or enabling a ROS re-mapper. The framework can also visualize the current state of physical robots/objects in PyBullet. This is very useful when debugging software and hardware.

2.6 Additional utilities

We also provide several utilities to facilitate development and safe robot operation. These utilities are described below.



(a) (b) (c)

Figure 5: A human interacting with a virtual world. (a) The experimental setup where the human interacts with the virtual world using a haptic device. (b) The Kuka LWR robot in PyBullet about to interact with a static collision object including the coordinate frames (the frames are not presented to the user). (c) The force-feeback evolution that is rendered to the human via the haptic device.

Model Predictive Control When developing MPC methods [31, 33], it is useful to slowly iterate through individual MPC iterations. Start, stop, and step ROS services are provided that allow the user to easily debug MPC controllers further facilitated by a GUI interface (see `rpbi_controls_node.py` in the `rpbi_utils` package).

Inverse Kinematics Inverse kinematics is a key requirement for robotic systems with several established libraries [44, 45, 46, 21]. We provide a standardized interface `ik_ros` that allows a user to easily swap out and compare different solvers. Several solvers are interfaced out-of-the box [45, 21, 44, 29]. The implementation is extensible so that additional solvers can be easily included.

Interpolation Control often requires smaller time resolution than planning. In this case, interpolation between knot points is necessary. The framework provides `interpolation_node.py` in the `rpbi_utils` package that performs interpolation of planned trajectories.

Time-sync ROS with PyBullet Synchronizing time between processes and a simulated time can be useful. We provide an option so that the user can synchronize the ROS clock with PyBullet's simulation time.

Safe robot operation Safely operating robots is of paramount importance - especially when conducting HRI experiments. We provide a package `safe_robot` that ensures safe robot motions acting as a guard between target and commanded states - every target is checked (e.g. link and joint position/velocity limits, and self-collision) prior to being commanded on the real system.

3 Use-cases

This section describes four use-cases highlighting the features of the proposed framework: (i) human interaction with a virtual world, (ii) learning from demonstration using dynamic movement primitives, (iii) full-body telepresence, and (iv) hardware realization. The code for running examples (i) and (ii) has been made fully open source. We plan to make the driver for the Xsens suit, used in (iii), open source along with an example.

3.1 Human interaction with virtual worlds

Developing robust learning and control techniques in contact-rich scenarios utilizing human input requires the human to interact with the virtual world. In order to prototype contact-rich machine learning and optimization-based methods with realistic interaction sequences requires such a haptic interface and a simulator for generating realistic force-feedback.

In this use-case, we present the user with a haptic interface, a simulated Kuka LWR robot arm, and a static collision box object (Figure 5a). Using the interface, the user controls a target position defined in the z_W axis that is constrained in the x_W, y_W axes (Figure 5b). The task for the robot is to minimize the distance between the end-effector position and the target while keeping the z_E and z_W axes aligned. The joint motion is generated using the IK features in PyBullet, we interface with

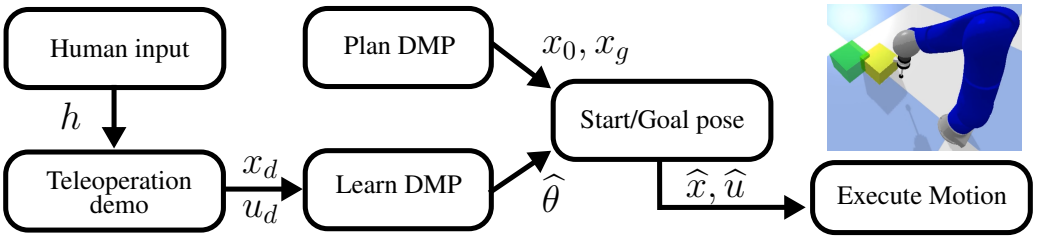


Figure 6: Pipeline for the learning from demonstration use-case.

this functionality using the `ik_ros` package described in Section 2.6. A simulated Force-Torque sensor is attached to the robot at the wrist joint. When the end-effector comes into contact with the static collision object, the force measured by the sensor in the z_E axis is rendered to the user via force-feedback. The evolution of the force feedback for a single interaction with the the box object is shown in Figure 5c. To run this example, attach a haptic device (3D Systems Touch X), and execute the command `roslaunch rpbi_examples human_interaction.launch`.

3.2 Learning from demonstration

Dynamic movement primitives (DMPs) [47] are a widely used mathematical formulation for modelling motor control of biological systems. Over recent years they have become a key component of learning from demonstration [48]. Many packages, examples, and code exists for learning and executing DMPs – several have been integrated in ROS. To highlight the flexibility and potential for learning from demonstration utlizing our framework we leverage a standard ROS package for learning DMPs [49].

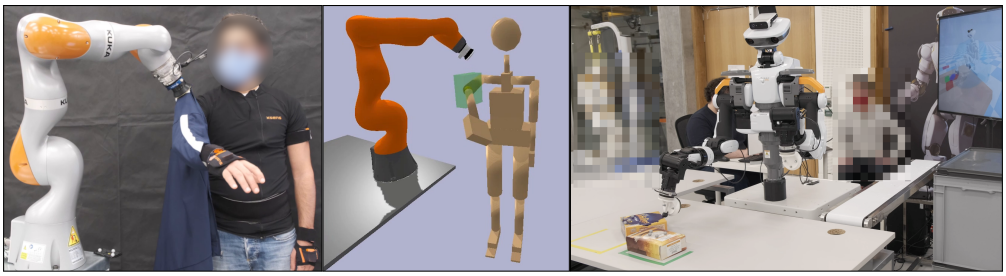
The goal in this section is to demonstrate how to learn a DMP from a teleoperated demonstration using our framework. In this use-case, the user interfaces with the system using the keyboard. A Kuka LWR robot arm is presented in PyBullet, and controlled in position control mode (Figure 6). The interface commands h are mapped to end-effector velocity in two dimensions. We use the EXOTica [21] plugin in the `ik_ros` package *link-anonymized* to perform inverse kinematics - the box and end-effector states x_d, u_d are saved as the human demonstration. The goal is for the human to demonstrate how to push a box from a starting location x_0 to the goal position x_g; this behavior is then learned from the human demonstration u_d using a DMP $\widehat{\theta}$. A random starting location for the end effector is chosen and the DMP is used to plan a motion $\widehat{x}, \widehat{u}$. When the DMP is executed, a starting position is chosen randomly. The data can be collected using standard ROS tools and processed to a Pandas data frame for analysis. To run this example, open a terminal and execute `roslaunch rpbi_examples lfd.launch`.

3.3 Full-body virtual-presence

In this use-case, we show that not only can we setup various teleoperation examples for the development of learning and control, but also we can integrate a realization of a virtual human that can interact with the PyBullet environment. We equip a human with an XSens suit and publish the 3D positions of each human body part into ROS as a transformation. Consecutively, we align each link of the human model in PyBullet with the positions of each body part of the human to realize a full-body virtual-presence of a humanoid figure. The setup for this use-case is shown via an assistive dressing scenario in Figure 7a, full details of the work can be found in [35].

3.4 Integration with hardware and MPC

A key feature of our framework is its ability to easily integrate with robot hardware and develop methods for MPC [31, 33]. In this use-case we highlight this feature of our framework using a pushing task deployed on the Kawada Nextage humanoid robot. The setup is shown in Figure 7c. A typical development cycle for research is to develop first in simulation, and then port the work to the real system. Due to the complexity of robotic systems, data collection, hardware issues, etc., the latter step can be quite time consuming. Our goal during development of the framework was to minimize this difficulty. We developed the facility to remap target joint states to several



(a) (b) (c)

Figure 7: Integrating robot hardware and sensing. (a) and (b) Real-time virtual-presence of a human in the simulation environment during assistive dressing. (a) The configuration of the human is sensed with the Xsens suit. (b) A humanoid figure that corresponds to the human is shown, along with a green *visual object* (Section 2.3.1) used to illustrate the estimated region of the occluded human elbow. (b) The Kawada Nextage humanoid robot performing a pushing task.

formats required by real systems in our lab, and several object types that can interface with real sensors (e.g. Vicon, AprilTags) or broadcast transforms (similar to a real object equipped with Vicon markers or AprilTags). The interface makes it simple to swap between testing/prototyping the system in simulation, and deploying on the real system – our demonstration reduces to setting a flag (`real_robot = True` or `False`). Furthermore, this use-case highlights the utility of the time-stepping feature of the interface for developing MPC algorithms, described in Section 2.6. Typically this can only be achieved in simulation, however since our system maps the simulator state to the real robot and has the facility to track objects using online sensing, e.g. we used Vicon, the framework can execute an MPC iteration on the real system by the user clicking a button. This significantly reduces the development time for laborious tasks such as parameter tuning.

4 Limitations

We implemented the framework using Python, a popular programming language with numerous libraries—making it suitable for our goals of easy prototyping and extensiblity. Despite Python being unsuitable for real-time systems, the robotics community has broadly adopted it as the language for implementing robotic experiments. Furthermore, we have found no latency issues in all of our experimental setups, running at frequencies of up to 200 Hz.

Currently, the framework only supports ROS Noetic—thus the only way to run the framework with ROS2 is via the ROS1 bridge [50]. Porting the framework to ROS2 is, at the time of writing this manuscript, in-development.

5 Conclusions

In this paper, we have proposed a framework for simulating/collecting data for contact-rich manipulation scenarios including: full physics simulation using PyBullet (known for reliable impact/contact modeling), integration with the ROS ecosystem,and several teleoperation interfaces and robots. Furthermore, the framework easily and demonstrably interfaces with robot hardware. We have specifically designed the implementation to exploit modularity and extensibility and to be highly flexible and easily developed.

We hope students, researchers, and industry will make use this framework to facilitate the development of their control algorithms and machine learning approaches in scenarios involving contact. The supplementary material contains full documentation. The main code base is open source at *link-anonymized*, which contains several examples, documentation, and videos. Dependencies are linked to from the documentation and system requirements are detailed. The framework is released under the LGPL license. The suggested setup is ROS Noetic, Python 3, on Ubuntu 20.04.

References

[1] I. Kao, K. Lynch, and J. W. Burdick. *Contact Modeling and Manipulation*, pages 647–669. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-30301-5_28.

[2] R. A. Brooks. Planning collision- free motions for pick-and-place operations. *The International Journal of Robotics Research*, 2(4):19–44, 1983. doi:10.1177/027836498300200402.

[3] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018. doi:10.1109/LRA.2018.2798285.

[4] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard. Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[5] M. Posa, C. Cantu, and R. Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014. doi:10.1177/0278364913506757.

[6] C. E. Mower, J. Moura, and S. Vijayakumar. Skill-based Shared Control. In *Proceedings of Robotics: Science and Systems (R:SS)*, Virtual, July 2021. doi:10.15607/RSS.2021.XVII.028.

[7] F. R. Hogan and A. Rodriguez. Reactive planar non-prehensile manipulation with hybrid model predictive control. *The International Journal of Robotics Research*, 39(7):755–773, 2020. doi:10.1177/0278364920913938.

[8] J. Moura, T. Stouraitis, and S. Vijayakumar. Non-prehensile planar manipulation via trajectory optimization with complementarity constraints. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Jan. 2022.

[9] T. Stouraitis, I. Chatzinikolaidis, M. Gienger, and S. Vijayakumar. Online hybrid motion planning for dyadic collaborative manipulation via bilevel optimization. *IEEE Transactions on Robotics*, 36(5):1452–1471, 2020. doi:10.1109/TRO.2020.2992987.

[10] F. Rydén and H. J. Chizeck. Forbidden-region virtual fixtures from streaming point clouds: Remotely touching and protecting a beating heart. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3308–3313, 2012. doi:10.1109/IROS.2012.6386012.

[11] H. Saeidi, J. D. Opfermann, M. Kam, S. Wei, S. Leonard, M. H. Hsieh, J. U. Kang, and A. Krieger. Autonomous robotic laparoscopic surgery for intestinal anastomosis. *Science Robotics*, 7(62), 2022. doi:10.1126/scirobotics.abj2908.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. doi:https://doi.org/10.1016/j.robot.2008.10.024.

[13] L. Armesto, J. Moura, V. Ivan, M. S. Erden, A. Sala, and S. Vijayakumar. Constraint-aware learning of policies by demonstration. *The International Journal of Robotics Research*, 37 (13-14):1673–1689, 2018. doi:10.1177/0278364918784354.

[14] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning (CoRL)*, 2021. URL `https://openreview.net/forum?id=8kbp23tSGYv`.

[15] A. S. Huang, E. Olson, and D. C. Moore. LCM: Lightweight communications and marshalling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4057–4062, 2010. doi:10.1109/IROS.2010.5649358.

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[17] H. Bruyninckx, P. Soetens, and B. Koninckx. The real-time motion control core of the Orocos project. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 2766–2771 vol.2, 2003. doi:10.1109/ROBOT.2003.1242011.

[18] T. Foote. tf: The transform library. In *IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, 2013. doi:10.1109/TePRA.2013.6556373.

[19] D. Coleman, I. Sucan, S. Chitta, and N. Correll. Reducing the barrier to entry of complex robotic software: a MoveIt! case study, 2014. URL https://arxiv.org/abs/1404.3785.

[20] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, and E. F. Perdomo. ros_control: A generic and simple control framework for ros. *Journal of Open Source Software*, 2(20):456, 2017. doi:10.21105/joss.00456. URL https://doi.org/10.21105/joss.00456.

[21] V. Ivan, Y. Yang, W. Merkt, M. P. Camilleri, and S. Vijayakumar. *EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control*, pages 211–240. Springer International Publishing, Cham, 2019. ISBN 978-3-319-91590-6. doi:10.1007/978-3-319-91590-6_7.

[22] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[23] W. Schroeder, K. Martin, B. Lorensen, and I. Kitware. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Kitware, 2006. ISBN 9781930934191. URL https://books.google.co.uk/books?id=rx4vPwAACAAJ.

[24] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim. RViz: A toolkit for real domain data visualization. *Telecommun. Syst.*, 60(2):337–345, oct 2015. doi:10.1007/s11235-015-0034-5.

[25] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154, Sendai, Japan, Sep 2004.

[26] R. Smith et al. Open dynamics engine. *http://ode.org*, 2005.

[27] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012. doi:10.1109/IROS.2012.6386109.

[28] R. Tedrake et al. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2014.

[29] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2020.

[30] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu. Fast and Feature-Complete Differentiable Physics Engine for Articulated Rigid Bodies with Contact Constraints. In *Proceedings of Robotics: Science and Systems (R:SS)*, Virtual, July 2021. doi:10.15607/RSS.2021.XVII.034.

[31] Anonymized Authors. Anonymized Title. In *Proceedings of Robotics: Science and Systems (R:SS)*, Virtual, July 2021.

[32] 1st Anonymized Author. *Anonymized Title*. PhD thesis, Anonymized University, 2021.

[33] Anonymized Authors. Anonymized Title. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA) 2022*, Jan. 2022.

[34] 2nd Anonymized Author. *Anonymized Title*. PhD thesis, Anonymized University, 2021.

[35] Anonymized Authors. Anonymized Title. *IEEE Robotics and Automation Letters*, 2022.

[36] T. Field, J. Leibs, J. Bowman, and D. Thomas. rosbag. http://wiki.ros.org/rosbag, 2020. Online; accessed 14 June 2022.

[37] A. Taylor. rosbag_pandas. http://wiki.ros.org/rosbag_pandas, 2019. Online; accessed 14 June 2022.

[38] B. Acosta, W. Yang, and M. Posa. Validating robotics simulators on real-world impacts. *IEEE Robotics and Automation Letters*, pages 1–1, 2022. doi:10.1109/LRA.2022.3174367.

[39] N. Fazeli, M. Oller, J. Wu, Z. Wu, J. B. Tenenbaum, and A. Rodriguez. See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. *Science Robotics*, 4 (26), 2019. doi:10.1126/scirobotics.aav3123.

[40] A. Rodriguez. The unstable queen: Uncertainty, mechanics, and tactile feedback. *Science Robotics*, 6(54), 2021. doi:10.1126/scirobotics.abi4667.

[41] Anonymized Authors. Anonymized title. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1497–1504, 2019.

[42] W. Merkt, Y. Yang, T. Stouraitis, C. E. Mower, M. Fallon, and S. Vijayakumar. Robust shared autonomy for mobile manipulation with continuous scene monitoring. In *13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 130–137, 2017. doi:10.1109/COASE.2017.8256092.

[43] T. Klamt, D. Rodriguez, M. Schwarz, C. Lenz, D. Pavlichenko, D. Droeschel, and S. Behnke. Supervised autonomous locomotion and manipulation for disaster response with a centaur-like robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018. doi:10.1109/IROS.2018.8594509.

[44] P. Beeson and B. Ames. TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935, 2015. doi:10.1109/HUMANOIDS.2015.7363472.

[45] M. L. Felis. RBDL: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, pages 1–17, 2016. doi:10.1007/s10514-016-9574-0.

[46] D. Rakita, B. Mutlu, and M. Gleicher. RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion. In *Proceedings of Robotics: Science and Systems (R:SS)*, Pittsburgh, Pennsylvania, June 2018. doi:10.15607/RSS.2018.XIV.043.

[47] S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006. doi:10.1007/4-431-31381-8_23.

[48] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel. Dynamic movement primitives in robotics: A tutorial survey, 2021.

[49] S. Niekum. dmp. http://wiki.ros.org/dmp, 2015. Online; accessed 14 June 2022.

[50] D. Thomas and J. Perron. ros1_bridge. https://index.ros.org/p/ros1_bridge/, 2022. Online; accessed 14 June 2022.