2025 IEEE/SICE International Symposium on System Integration (SII)
January 21-24, 2025, Munich, Germany

RPC: A Modular Framework for Robot Planning, Control, and Deployment

Seung Hyeon Bang[1], Carlos Gonzalez[1], Gabriel Moore[2], Dong Ho Kang[3], Mingyo Seo[2], Ryan Gupta[1], and Luis Sentis[1]

Abstract— **This paper presents an open-source, lightweight, yet comprehensive software framework, named RPC, which integrates physics-based simulators, planning and control libraries, debugging tools, and a user-friendly operator interface. RPC enables users to thoroughly evaluate and develop control algorithms for robotic systems. While existing software frameworks provide some of these capabilities, integrating them into a cohesive system can be challenging and cumbersome. To overcome this challenge, we have modularized each component in RPC to ensure easy and seamless integration or replacement with new modules. Additionally, our framework currently supports a variety of model-based planning and control algorithms for robotic manipulators and legged robots, alongside essential debugging tools, making it easier for users to design and execute complex robotics tasks. The code and usage instructions of RPC are available at https://github.com/shbang91/rpc.**

I. INTRODUCTION

In order to deploy control algorithms safely and reliably on robotic hardware, it is essential to first evaluate them extensively and rigorously in simulation environments. While learning-based control algorithms are widely popular [1], [2], model-based control algorithms remain essential due to their capacity to provide systematic analysis and generalization without requiring data collection [3], [4]. However, these algorithms are often challenging to implement due to their complex optimization processes and are typically not available in a user-friendly form for the broader technical community. Finally, debugging tools are essential within the software components for complex robots, as they are critical for diagnosing and resolving issues that arise during the development and deployment of control processes.

To tackle these challenges, this paper introduces a software architecture designed to integrate multiple physics-based simulators, model-based planning and control modules, visualization tools, plotting and logging utilities, and operator interfaces for robotic systems. This integration facilitates intuitive deployment, thorough testing, and iterative refinement of control algorithms, significantly enhancing the reliability of the robot control deployment process.

A. Related Work

Recent advancements in physics-based simulators, such as MuJoCo [5], PyBullet [6], and Raisim [7], have significantly accelerated progress in robotics research. However, interfacing these simulators and control modules, or integrating new robots into these environments, remains a complex challenge. Each simulator provides its own APIs for low-level access to its full capabilities, requiring the development of higher-level interface (wrapper/utility) functions to enable the effective evaluation of controllers within these environments. To address this, the *mc-mujoco* library [8] facilitates integration between MuJoCo and the *mc-rtc* robot control framework [9], while the *PnC* library [10] bridges DART [11] with its associated control framework. Compared to these libraries, our software framework offers greater versatility by supporting interfaces with both the MuJoCo and PyBullet simulators.

Significant research has been devoted to model-based motion planning and control algorithms for robotic systems, yet much of the related software remains proprietary, difficult to access, or challenging to integrate and test. While some libraries, such as the inverted pendulum-based gait planner [12] and the task space inverse dynamics controller [13], have been released as open-source, they are not self-contained and require external libraries to complete the motion planning and control pipeline. In contrast, libraries like *OCS2* [14], *Drake* [15], *open-robotics-software* [16], *Cheetah-software* [17], and *PnC* [10] offer integrated solutions that include both motion planning and feedback control, along with a comprehensive test environment. However, these solutions have significant drawbacks: *OCS2* is heavily reliant on ROS, *Drake* lacks versatile options regarding simulators and visualization tools, *open-robotics-software* is Java-based, and both the *Cheetah-software* and *PnC* provide limited planning and control options. Our proposed framework overcomes these limitations by being ROS-independent, C++-based for real-time performance, and highly versatile.

Visualizing robots alongside their controller states and planned actions is essential during code development and debugging. Additionally, a user-friendly operator interface can minimize delays in non-autonomous operations and expand the range of possible tasks. To meet these needs, [18] developed debugging tools that include logging features, a robot visualizer, and a practical user interface (UI). Similarly, [19] introduced a graphical user interface (GUI) that provides an interactive simulation environment with live plotting and user-configurable parameters. Our framework offers comparable capabilities, featuring comprehensive visualizations, live plotting, logging, and operational tools

[1]S.H. Bang, C. Gonzalez, and L. Sentis are with the Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, TX 78712, USA bangsh0718@utexas.edu

[2]G. Moore and M. Seo are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, TX 78712, USA

[3]D.H. Kang is with the Department of Mechanical Engineering, The University of Texas at Austin, TX 78712, USA

that support both development and deployment—capabilities extending our previous work [10].

B. Contributions

The main contributions of this paper are the following:

1) We have devised a lightweight yet comprehensive open-source software framework, named **RPC**, which integrates motion planning and control modules for robotic applications, along with graphical tools for robot visualizations, data logging, and user interfaces.
2) We provide a versatile software interface that allows the proposed framework to be easily deployed across multiple high-fidelity physics simulators for extensive algorithm testing and demonstrate its smooth integration within the ROS environment for conducting hardware experiments.
3) We demonstrate the use of the proposed framework in loco-manipulation tasks performed by the humanoid robot DRACO 3 in both simulated and real hardware environments.

C. Organization

The remainder of this paper is organized as follows. Section II provides a concise overview of the proposed software framework and its key modules. Section III demonstrates the practicality of the framework through humanoid locomanipulation tasks. Finally, Section IV concludes the paper and discusses potential directions for future work.

II. System Modules

This section describes the overall software framework of **RPC** and its key modules. An overview of the proposed software architecture is illustrated in Fig. 1.

A. Test Environment

In **RPC**, we integrate two high-fidelity physics simulators: PyBullet and MuJoCo. This allows us to readily evaluate control algorithms across multiple simulators (e.g., to asses the controller's performance under different contact dynamics models) to enhance the versatility and reliability of the resulting algorithms. Each robot has its associated main simulation script, where our utility functions — integrated with the APIs of PyBullet and MuJoCo — allow for reading sensor data from the simulation and applying control signals to the robot's actuators.

B. Interface Layer

The modules in this layer enable communication between the low-level controllers in the **Test Environment** and the high-level layers (i.e., **User Command** and the **Planning and Control Layer**). The versatility of these modules ensures that our framework can be seamlessly applied in both physics simulators and hardware environments.

The key modules of this layer are the following:

- **InterruptHandler**: This module manages state transitions within either the `LocomotionStateMachine` or `ManipulationStateMachine` based on user input commands. In our current setup, these commands are sent via keyboard, but other input devices (e.g., joysticks) can be easily added.
- **Interface**: This module manages the communication of sensor data and commands from the **Test Environment** to the **Planning and Control Layer**. In one direction, the `SensorData` read from the **Test Environment** is updated at each servo loop and relayed to the `StateEstimator` class to update the robot's states. This data includes measurements from the IMU, joint encoders, F/T sensors, and cameras. In the opposite direction, the `GetCommand` function is invoked to compute the `Command` using the **Planning and Control Layer**. This `Command`, which consists of joint positions, velocities, and torques, is then applied to the robot's actuator via a joint impedance controller.
- **TeleopHandler**: This module manages the communication with the teleoperation devices (e.g., RealSense T265 camera) through ZeroMQ [20] and Protocol Buffers [21], and relays the teleoperation commands to the `ManipulationPlanner`.
- **TaskGainHandler**: This module manages the task PD controller gains in the `TCIContainer` during both simulation and hardware operation. Users can adjust the task gains in real-time through a user-specified communication protocol. This is particularly useful for hardware experiments that require frequent gain tuning. In the current implementation this is achieved using ROS messages and ROS service calls.

C. Planning and Control Layer

The **Planning and Control Layer** is a crucial component that enables robots to perform complex tasks reliably and efficiently. The modules contained in this layer work together to transform high-level goals into precise, coordinated low-level actuator commands. The key components include `StateEstimator`, `RobotSystem`, `ControlArchitecture`, `StateMachine`, `Planner`, `Manager`, `TCIContainer`, and `WBC`. Their flexibility allows for easy adaptation to new robots, increasing the versatility of our software framework.

- **StateEstimator:** This class estimates the floating base state (i.e., SE(3) and twist). We provide two different state estimators: 1) a simple estimator that uses only an IMU and leg joint encoders [22], and 2) a KF-based estimator [23].
- **RobotSystem:** This class serves as a wrapper around Pinocchio [24], enabling efficient computation of rigid body dynamics. It provides APIs for obtaining kinematic and dynamic properties of the robot, such as link Jacobians, centroidal states [25], and the mass matrix. This class is instantiated using a universal robot description file (URDF) and is updated in each control loop to reflect the robot's current configuration and velocity states in conjunction with the chosen `StateEstimator`.

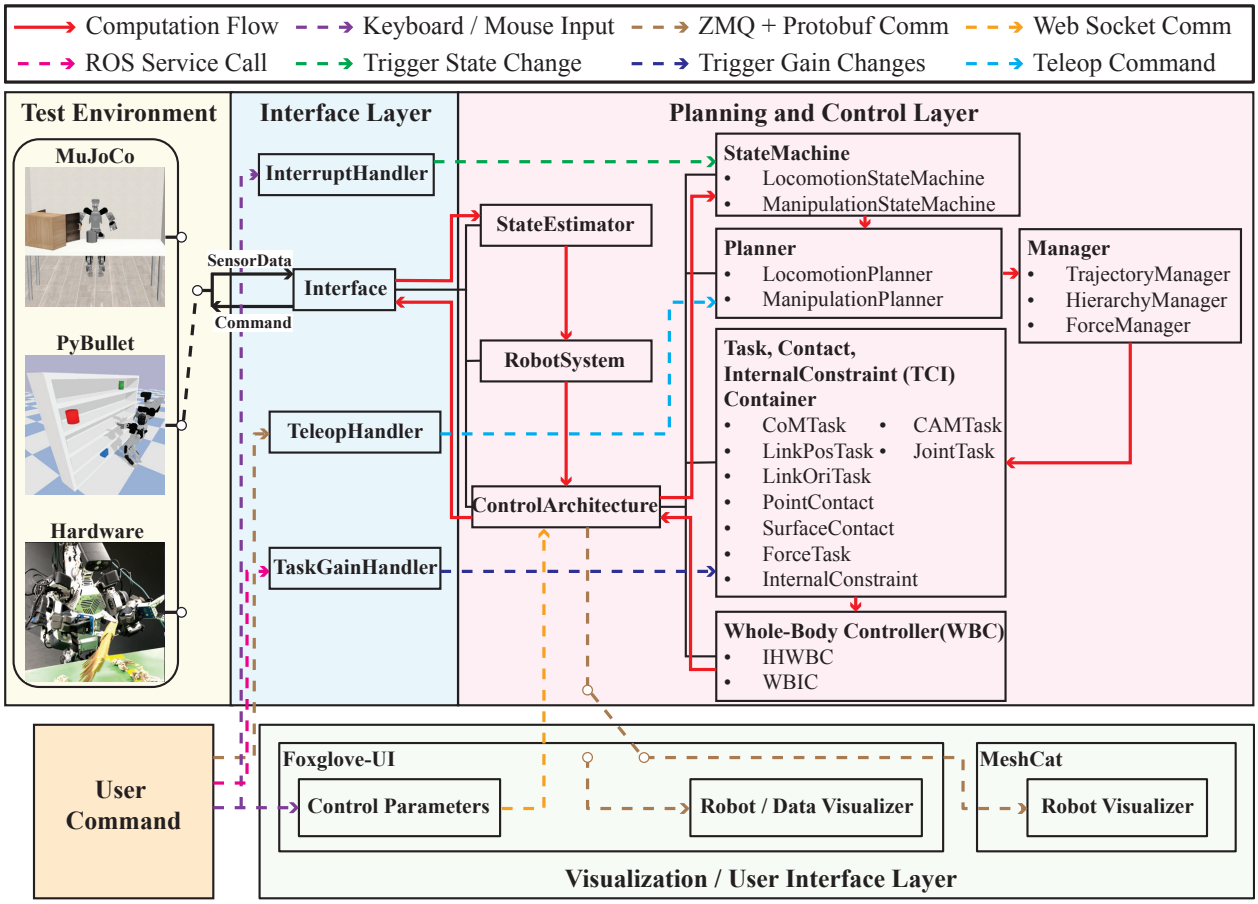


Fig. 1. **Overall software architecture: RPC** consists of the Test Environment, Interface Layer, Planning and Control Layer, and Visualization / User Interface Layer. Each layer includes several modules, and their interaction methods are illustrated in different line types.

- **ControlArchitecture:** This class is a key component of **RPC**, responsible for generating control commands that are passed to `Command` by integrating all necessary modules for robot planning and control. It is instantiated with a `RobotSystem` object and, within its constructor, instances of the `StateMachine`, `Planner`, `Manager`, `TCIContainer`, and `WBC` classes are created. During each control loop, the `GetCommand` function is invoked to compute the control commands (i.e., joint positions, velocities, and torques).
- **StateMachine:** This class coordinates the robot's complex behaviors by managing a finite number of locomotion or manipulation states and by handling transitions between them based on predefined conditions or inputs. Each state corresponds to a distinct contact mode or a specific task contributing to a modular and structured control system.
  This `StateMachine` class is instantiated with the `ControlArchitecture` and `RobotSystem` instances. At the start of each state, the `FirstVisit` function is invoked to initialize the desired control signal trajectories, typically using predefined temporal parameters, via the `Planner` or `Manager` in the `ControlArchitecture`. During each control loop, the `OneStep` function updates these trajectories as defined in `FirstVisit`. The `EndOfState` function checks whether the termination conditions for the current state have been met and triggers a state transition based on predefined temporal parameters, contact events, or signals from the interrupt handler.
- **Planner:** The `Planner` module is responsible for generating motion plans that enable robots to move effectively and efficiently. Within **RPC**, we have implemented both locomotion and manipulation planners to support robotic systems.
  To address the different requirements for balance and stability during walking, we provide two types of locomotion planners: 1) the Divergent Component of Motion (DCM) planner [26] and 2) the convex Model Predictive Control (MPC) planner [27]. The DCM planner is designed for quasi-static walking, utilizing DCM dynamics based on the Linear Inverted Pendulum Model (LIPM) to calculate the robot's motions. It takes a pre-determined sequence of foot placements (generated by a footstep planner) as input and generates a DCM trajectory, which serves as the reference signal for the robot's center of mass (CoM) task in `TCIContainer`. On the other hand, the convex MPC planner is op-

timized for dynamic walking, employing a Single Rigid Body Dynamics (SRBD) model. The MPC planner takes velocity commands—specifically, the desired CoM velocity in the x and y directions and the yaw velocity of the torso—as input. It then generates a ground reaction force (GRF) trajectory, which is used as the reference for the foot force task in `TCIContainer`. Additionally, we provide a variant of the MPC planner known as VI-MPC [28], which extends the SRBD model by incorporating composite rigid body inertia. This enhancement allows the planner to compute more reliable GRFs, enabling faster and more efficient maneuvers.

For manipulation planning, we provide various interpolation methods for trajectory generation to ensure smooth motions as the robot's arm reaches a target position. The following interpolation options are included in **RPC**: `CosineInterpolate`, `HermiteCurve`, `MinJerkCurve`, and `CubicBezier`.

- **Manager:** The `Manager` module is a utility class that serves as an interface between the `Planner` and `TCIContainer`. Specifically, it receives the desired control signals from the `Planner` and updates the corresponding `Task` or `Contact` elements in the `TCIContainer`.
- **TCIContainer:** The `TCIContainer` module is designed to efficiently manage a list of `Task`, `Contact`, and `InternalConstraint` elements, which are frequently updated for use in the `WBC` module. This module initializes and maintains these lists, ensuring that they are easily accessible and modifiable. We also provide modular `Task`, `Contact`, and `InternalConstraint` classes, making them reusable across different types of `WBC` instances and flexible enough to accommodate robot-specific tasks (e.g., Whole-body Orientation task [29]). The following options are available: `JointTask`, `SelectedJointTask`, `LinkPosTask`, `LinkOriTask`, `CoMTask`, `CAMTask`, `WBOTask`, `PointContact`, `SurfaceContact`, `RollingJointConstraint`.
- **WBC:** The `WBC` module is designed to convert the high-level task specifications (such as CoM or End-effector task objectives) into low-level motor commands that drive the robots. To address the different control requirements, we provide two types of `WBC`: 1) the Implicit Hierarchical Whole-body Controller (IHWBC) [10] and 2) the Whole-body Impulse Controller (WBIC) [17].

  IHWBC employs an implicit hierarchy between the `Task` and soft constraints to handle `Contact`, enabling smooth task and contact transitions. This controller is suitable for robots with highly accurate dynamics models as it calculates joint position and velocity commands in a dynamically consistent manner via integration schemes [16].

  In contrast, WBIC employs a strict hierarchy between the `Task` and hard constraints to handle `Contact` using a null-space projection technique to strictly hold task priority. This WBC is effective for robots that need reliable motion stabilization through joint position feedback control since it utilizes an inverse kinematics algorithm to compute joint position, velocity, and acceleration commands.

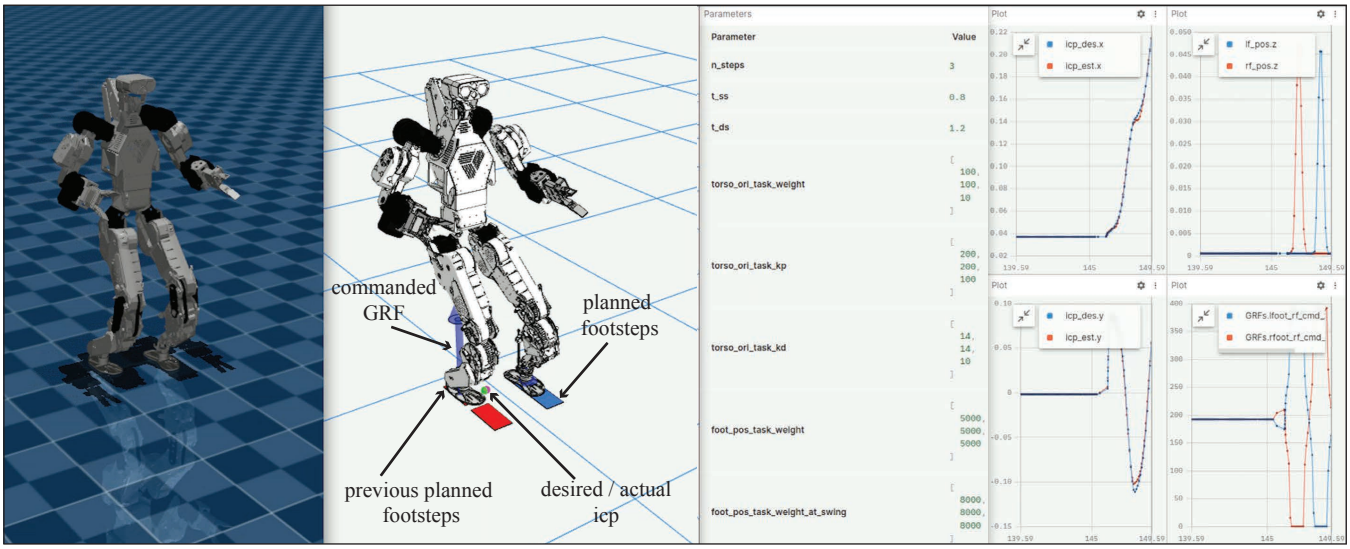
D. Visualization / User Interface Layer

We have set up two main forms of operating and visualizing robots: one where the user sends commands via the keyboard and can visualize the robot via Meshcat, and another one using a GUI to operate and visualize the robot via Foxglove.

- **Meshcat:** Meshcat is a 3D viewer that communicates over websockets and runs in the browser. We use the Python bindings for Meshcat [30] to host the `Robot Visualizer` server and visualize several elements, such as the overall robot, its DCM, its CoM, its desired GRF's, and its planned footsteps. This data is sent out as ZMQ messages and is also stored in a pickle file. This last step allows us to replay any given log in greater detail, including all visual elements, while scrolling through time.
- **Foxglove:** Foxglove [31] is a software package equipped with visualization, plotting, logging, and operation capabilities for robotic platforms. Currently, it easily integrates into ROS systems, but requires additional steps to integrate into other environments. Hence, we have developed several modules to use it without requiring ROS in environments such as **RPC**. In particular, we run two different servers by utilizing the Foxglove websocket protocol: the `Control Parameters` server and the `Robot/Data Visualizer`. The `Control Parameters` server, shown in Fig. 2(c), provides the user with a web GUI to adjust the robot's control parameters while operating the robot. A client that subscribes to these parameters is run in the `ControlArchitecture` and updates its values accordingly. The `Robot/Data Visualizer` hosts topics rendered in the 3D viewer, shown in Fig. 2(b), and/or in plots, shown in Fig. 2(d). We also provide the tools to log this data into an MCAP file, which can then be uploaded to Foxglove for synchronized replay, complete with all its visual elements. The ability to run either server independently of one another allows the user to implement alternative visualizers without losing the ability to adjust robot parameters while in operation.

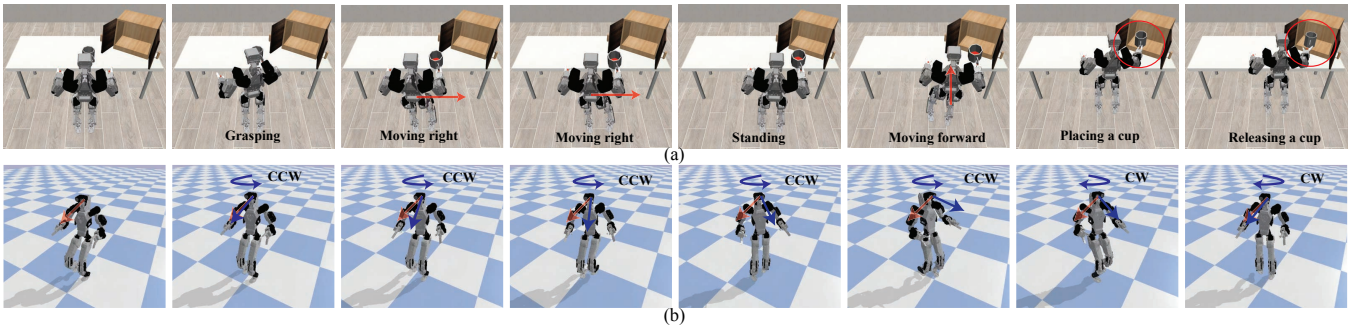
III. DEMONSTRATIONS

In this section, we demonstrate the practicality of our proposed framework for locomanipulation tasks for a 25-DOF humanoid robot, DRACO 3 [22], in both simulation and hardware environments. For more details on the experiments conducted in this section, please refer to the accompanying video.



(a) (b) (c) (d)

Fig. 2. **Foxglove UI usage:** (a) MuJoCo simulation. (b) Robot model visualization window in Foxglove. (c) Control parameters tuning window in Foxglove. (d) Data visualization window in Foxglove.



(a)

(b)

Fig. 3. **Simulation snapshots:** (a) Teleoperation-based locomanipulation for a cup shelving task. (b) Convex MPC-based omnidirectional walking task. Red arrows represent the initial heading, and the blue arrows indicate the current heading.

A. Simulation

We demonstrate here our ability to synthesize complex manipulation and locomotion behaviors in simulation environments using **RPC**.

First, we performed a locomanipulation task in MuJoCo, where the robot was required to complete a cup shelving task, as shown in Fig. 3(a). In this task, the manipulation motion plan (robot's right hand SE(3) pose) was provided through teleoperation by an operator using the Realsense T265 tracking camera, with gripper commands (open and close) triggered by keyboard keystrokes. These manipulation commands were relayed to the `ManipulationPlanner` via the `TeleopHandler`. For the locomotion planner, we employed the DCM planner with predefined temporal parameters and step lengths. The footstep plan was triggered by keyboard keystrokes through `InterruptHandler`. Given these manipulation and locomotion plans, IHWBC computed joint commands in each corresponding `StateMachine`, which were then applied to the actuators via `Command` to accomplish the task. Notably, the robot was able to freely move the cup while holding it, regardless of its locomotion state, due to our architecture's use of independent `StateMachine` instances for manipulation and locomotion tasks, unlike the approach in [8].

Next, we performed a dynamic locomotion task in Pybullet, where the robot was required to maneuver omnidirectionally, as shown in Fig. 3(b). In this task, given velocity commands from an operator through `InterruptHandler`, the SRBD model-based convex MPC planner generated the reference GRF trajectory, while the Raibert hueristics [32] was used to decide the desired foot placement. Based on these locomotion plans, WBIC computed joint commands to track the body posture, swing foot pose, and GRF.

B. Hardware

DRACO 3 utilizes ROS as its middleware, so we set up a ROS nodelet where **RPC** receives joint state information and transmits joint commands via shared memory. Within the nodelet modules, `Interface` effortlessly connects the **Planning and Control Layer** to the robot without any modifications, ensuring a smooth and reliable transition between simulation and real hardware.

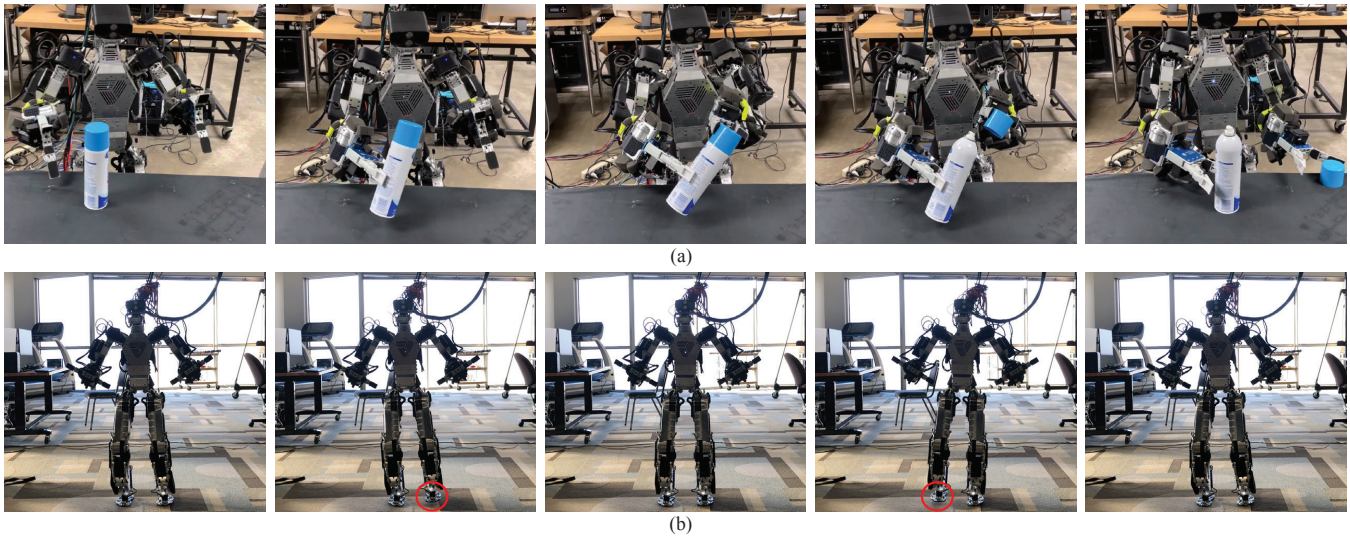


Fig. 4. **Hardware experiment snapshots:** (a) Teleoperation-based bi-manipulation for a spray cap removal task. (b) DCM-based in-place quasi-static stepping.

First, we performed a spray cap removal task as shown in Fig. 4(a), where the robot was required to grasp a spray can with one hand and remove its cap with the other hand while balancing on its feet. Since this task required bimanual manipulation, we used a VR device (Meta's Oculus Quest 2) for the robot's teleoperation. The underlying software architecture for this task is similar to the cup shelving task described in the previous subsection.

Then, we also performed a DCM-based stepping-in-place task, as shown in Fig. 4(b). In this task, we employed the IHWBC controller to track body posture, DCM, and swing foot pose.

It is important to note that these hardware experiments were extensively tested in simulation environments before being evaluated on the real hardware.

IV. CONCLUSIONS

In this work, we develop and open-source **RPC**, a testing and development control software framework designed for complex robotic systems, with a particular focus on humanoid robots. The framework is fully equipped with simulators, planning and control modules, and debugging tools, enabling robotics researchers to develop and test their algorithms with minimal external dependencies. Its modular design ensures that the software architecture can be easily extended or adapted to incorporate new simulators, hardware platforms, or control strategies, thereby enhancing the framework's flexibility and scalability. We believe that **RPC** will greatly facilitate the development, testing, and deployment of advanced robotics systems.

ACKNOWLEDGMENT

This work was supported by the Office of Naval Research (ONR), Award No. N00014-22-1-2204.

REFERENCES

[1] M. Seo, S. Han, K. Sim, S. H. Bang, C. Gonzalez, L. Sentis, and Y. Zhu, "Deep Imitation Learning for Humanoid Loco-manipulation Through Human Teleoperation," *IEEE-RAS International Conference on Humanoid Robots*, pp. 1–8, 2023.

[2] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," *International Journal of Robotics Research*, 2024.

[3] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1168–1175, 2014.

[4] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. D. Prete, "Optimization-Based Control for Dynamic Legged Robots," *IEEE Transactions on Robotics*, vol. 40, pp. 43–63, 11 2024.

[5] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 10 2012, pp. 5026–5033.

[6] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," 2016.

[7] J. Hwangbo, J. Lee, and M. Hutter, "Per-Contact Iteration Method for Solving Contact Dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018.

[8] R. P. Singh, P. Gergondet, and F. Kanehiro, "Mc-Mujoco: Simulating Articulated Robots with FSM Controllers in MuJoCo," in *2023 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 1 2023, pp. 1–5.

[9] "mc-rtc." [Online]. Available: https://jrl-umi3218.github.io/mc_rtc/index.html

[10] J. Ahn, S. J. Jorgensen, S. H. Bang, and L. Sentis, "Versatile Locomotion Planning and Control for Humanoid Robots," *Frontiers in Robotics and AI*, vol. 8, no. August, pp. 1–17, 2021.

[11] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. Karen Liu, "DART: Dynamic Animation and Robotics Toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[12] S. Caron, A. Escande, L. Lanari, and B. Mallein, "Capturability-Based Pattern Generation for Walking with Variable Height," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 517–536, 2020.

[13] A. Del Prete, N. Mansard, O. E. Ramos, O. Stasse, and F. Nori, "Implementing Torque Control with High-Ratio Gear Boxes and Without Joint-Torque Sensors," *International Journal of Humanoid Robotics*, vol. 13, no. 01, p. 1550044, 3 2016.

[14] Farbod Farshidian and Others, "OCS2: An open source library for Optimal Control of Switched Systems." [Online]. Available: https://github.com/leggedrobotics/ocs2

[15] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[16] IHMC Robotics, "IHMC Open Robotics Software," 2018. [Online]. Available: https://github.com/ihmcrobotics/ihmc-open-robotics-software
[17] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly Dynamic Quadruped Locomotion via Whole-Body Impulse Control and Model Predictive Control," 2019. [Online]. Available: http://arxiv.org/abs/1909.06586
[18] M. Johnson, B. Shrewsbury, S. Bertrand, T. Wu, D. Duran, M. Floyd, P. Abeles, D. Stephen, N. Mertins, A. Lesman, J. Carff, W. Rifenburgh, P. Kaveti, W. Straatman, J. Smith, M. Griffioen, B. Layton, T. De Boer, T. Koolen, P. Neuhaus, and J. Pratt, "Team IHMC's lessons learned from the DARPA robotics challenge trials," *Journal of Field Robotics*, vol. 32, no. 2, pp. 192–208, 3 2015.
[19] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo," 12 2022. [Online]. Available: http://arxiv.org/abs/2212.00541
[20] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, Inc., 2013.
[21] Google, "Protocol Buffers," 2008. [Online]. Available: https://developers.google.com/protocol-buffers
[22] S. H. Bang, C. Gonzalez, J. Ahn, N. Paine, and L. Sentis, "Control and evaluation of a humanoid robot with rolling contact joints on its lower body," *Frontiers in Robotics and AI*, vol. 10, no. October, pp. 1–21, 2023.
[23] T. Flayols, A. Del Prete, P. Wensing, A. Mifsud, M. Benallegue, and O. Stasse, "Experimental evaluation of simple estimators for humanoid robots," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 11 2017, pp. 889–895.
[24] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 1 2019, pp. 614–619.
[25] D. E. Orin, A. Goswami, and S. H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, vol. 35, no. 2-3, pp. 161–176, 10 2013.
[26] J. Englsberger, C. Ott, and A. Albu-Schäffer, "Three-Dimensional Bipedal Walking Control Based on Divergent Component of Motion," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 355–368, 4 2015.
[27] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control," *IEEE International Conference on Intelligent Robots and Systems*, pp. 7440–7447, 2018.
[28] S. H. Bang, J. Lee, C. Gonzalez, and L. Sentis, "Variable Inertia Model Predictive Control for Fast Bipedal Maneuvers," 7 2024. [Online]. Available: http://arxiv.org/abs/2407.16811
[29] Y.-m. Chen, G. Nelson, R. Griffin, M. Posa, and J. Pratt, "Integrable Whole-Body Orientation Coordinates for Legged Robots," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 10 2023, pp. 10 440–10 447.
[30] "meshcat-python: Python Bindings to the MeshCat WebGL viewer." [Online]. Available: https://github.com/meshcat-dev/meshcat-python
[31] Foxglove Developers, "Foxglove," 2024. [Online]. Available: https://foxglove.dev
[32] M. H. Raibert, H. B. Brown, M. Chepponis, E. Hastings, J. Koechling, K. N. Murphy, S. S. Murthy, and A. J. Stentz, "Stable Locomotion," *Order A Journal On The Theory Of Ordered Sets And Its Applications*, no. 4148, 1983.