IEEE Robotics & Automation Magazine (RAM) paper, presented at ICRA 2026, Vienna, Austria. Cite as RAM paper.

PAPRLE (Plug-And-Play Robotic Limb Environment): A Modular Ecosystem for Robotic Limbs

Obin Kwon[1*], Sankalp Yamsani[1*], Noboru Myers[1], Sean Taylor[1], Jooyoung Hong[1], Kyungseo Park[2], Alex Alspach[3] and Joohyung Kim[1]

***Abstract*— We introduce PAPRLE (Plug-And-Play Robotic Limb Environment), a modular ecosystem that enables flexible placement and control of robotic limbs. With PAPRLE, a user can change the arrangement of the robotic limbs, and control them using a variety of input devices, including puppeteers, gaming controllers, and VR devices. This versatility supports a wide range of teleoperation scenarios and promotes adaptability to different task requirements. We also introduce a pluggable puppeteer device that can be easily mounted and adapted to match the target robot configurations. PAPRLE supports bilateral teleoperation through these puppeteer devices, agnostic to the type or configuration of the follower robot. The modular design of PAPRLE facilitates novel spatial arrangements of the limbs and enables scalable data collection, thereby advancing research in embodied AI and learning-based control. We validate PAPRLE in various real-world settings, demonstrating its versatility across diverse combinations of leader devices and follower robots. The system will be released as open source, including both hardware and software components, to support broader adoption and extension.**

Teleoperation, Data Collection, Human-Robot Interaction

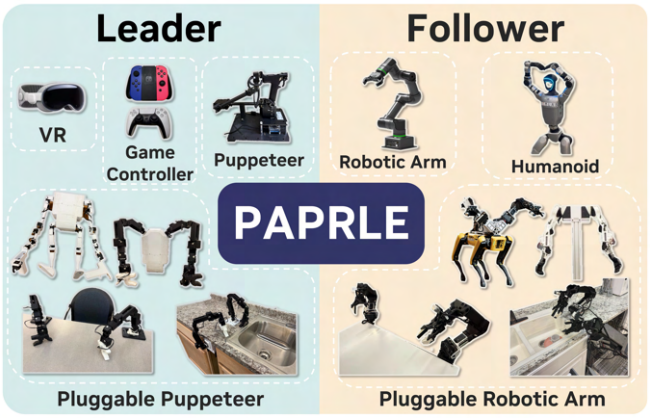

Fig. 1: **Concept of PAPRLE.** Various leader devices (left) can be flexibly paired with diverse robot embodiments (right) through a plug-and-play teleoperation system that supports modular and interchangeable components.

I. Introduction

In many research and deployment scenarios, the flexible arrangement of robotic limbs is often required to support a wide range of task configurations. For example, a table-mounted arm is well-suited for tabletop tasks, but becomes limited for more complex tasks, such as loading dishes from a sink to dishwasher, where reachability and varied interaction angles are required. However, existing systems are often designed for specific scenarios, making it difficult to explore new configurations, support scalable implementations. To address this limitation, the Plug-and-Play Robotic Arm System (PAPRAS) [1] was introduced, enabling modular assembly and flexible arrangement of robotic arms across diverse settings, including single-arm, multi-arms, or even humanoid-like platforms. The bottom-right side of Figure 1 shows the various examples of PAPRAS configurations.

While PAPRAS facilitates physical modularity, a unified control method capable of seamlessly operating across these diverse configurations remains an open challenge. Building on the modular philosophy of PAPRAS, we introduce PAPRLE (Plug-And-Play Robotic Limb Environment), which is a system that extends plug-and-play principles into the teleoperation and control domain. We use the term "*environment*" for the proposed system because it provides a flexible and composable platform where various robot configurations (followers) and control interfaces (leaders) can be easily plugged in, mixed, and matched. Figure 1 illustrates the core concept of PAPRLE. Operators can 'plug in' puppeteer devices as leaders, as a controller for the teleoperation system. In addition to puppeteers, other control modalities such as gaming controllers or VR devices, can also be 'plugged in' as a leader. This "plug-and-play" characteristic also applied to the follower side. The operator can configure custom robotic limb setups using PAPRAS as a follower, or they can use commercial humanoid or robotic arm platforms as a follower.

To support the variable limb configurations offered by PAPRAS, we introduce a pluggable puppeteer device (Figure 2a), which can be freely mounted on various places. The puppeteer device is a low-cost, 3D-printed, scaled-down version of the robot, actuated by small motors. Inspired by recent works [2], [3], [4], [5], this approach allows for direct joint-value mapping, providing users with an intuitive and physically grounded control experience. By attaching a mount to make those pluggable, we made a pluggable puppeteer device. Using this device, multiple puppeteers can

*These authors equally contributed to this work.
[1] Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, Champaign, IL, USA. {obinkwon, yamsani2, noborum2, seanlt2, jh97, joohyung}@illinois.edu
[2] Department of Robotics and Mechatronics Engineering, Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu, South Korea kspark@dgist.ac.kr
[3] Toyota Research Institute, Los Altos, CA, USA alex.alspach@tri.global

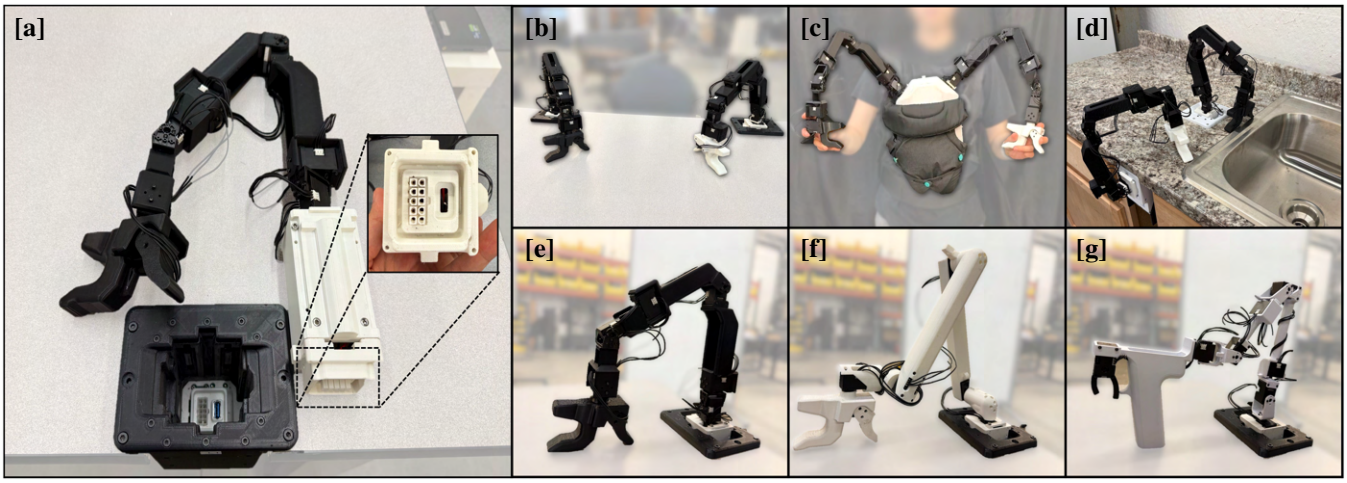

Fig. 2: **Pluggable Puppeteers** (a) The proposed puppeteer device and mounting interface. (b–c) Examples of the device mounted in diverse scenarios. (d–f) The same mounting base accommodates a variety of puppeteer devices, which are scaled replicas of different arms such as PAPRAS, UR5, and Unitree G1's arm.

be mounted as needed and also can be swapped to other puppeteers, enabling intuitive control of multi-limb robot setups.

This composability facilitates large-scale data collection across a wide range of robot-control configurations, supporting the advancement of embodied AI research. Compared to existing teleoperation systems that often targeted specific device–robot pairings, PAPRLE aims for a broader, more extensible architecture. It supports both task-space and joint-space control modalities, introduces a pluggable puppeteer hardware device, and incorporates force feedback mechanisms even across heterogeneous leader-follower configurations.

The contributions of this paper are summarized as follows:

- PAPRLE enables flexible robotic limb arrangements on both leader and follower side, ranging from tabletop arms to multi-limb humanoid systems
- We present a device- and robot-agnostic teleoperation system that supports diverse combinations of leader and follower devices.
- PAPRLE platform will be released as open source to encourage reproducibility, extensibility, and community-driven research.

II. Related Work

A growing movement in robotics research has focused on teleoperation as a pathway to large-scale data collection for learning-based robot policy. Systems such as ALOHA [3], Gello [2] enable intuitive teleoperation through kinesthetic interfaces by replicating the kinematics of the target robot. In addition, FACTR [4] and Echo [5] further enhance joint-space teleoperation with bilateral force feedback, enabling high-quality demonstrations for contact-rich tasks. On the other hand, BunnyVisionPro [6] or OpenTeleVision [7] proposed a VR-based pipeline for capturing human wrist pose and dextrous hand motion and translating it into robot control signals. While these approaches have shown compelling results, they are often designed for fixed device–robot pairings, limiting their adaptability to diverse teleoperation scenarios.

Focusing on broader applicability, several works have proposed modular and cross-platform teleoperation architectures that allow flexible pairing of input devices with a range of robotic platforms. ACE [8] introduces a portable exoskeleton with vision-based tracking, supporting cross-robot teleoperation. TeleMoMa [9] allows users to control mobile manipulators using various input devices such as RGB-D cameras or gamepads by abstracting user actions into a shared control space. MoMa-Teleop [10] takes a hybrid approach where the user controls the end-effector while the robot adjusts its base for extended reach. These systems demonstrate increased input modularity and cross-platform compatibility, but typically target a narrow set of follower configurations, often using commercial robots or predefined setups.

In contrast, our system supports arbitrary limb configurations on both the leader and follower sides, enabling plug-and-play reconfiguration across diverse embodiments. The key is the plug-and-play abstraction: not only can users swap between input devices (e.g., puppeteers, gaming controllers, VR), but the proposed puppeteer leader device itself is also modular and pluggable. This flexibility propagates to the follower side, where a wide range of robots from tabletop scenarios to humanoids, can be teleoperated using the same unified interface. Furthermore, we extend beyond existing work by providing force feedback even in heterogeneous scenarios, where the puppeteer leader and follower do not share the same structure or degrees of freedom.

III. Pluggable Puppeteers

In this section, we introduce a pluggable puppeteer leader device, designed for PAPRLE. Similarly to Gello [2], this de-

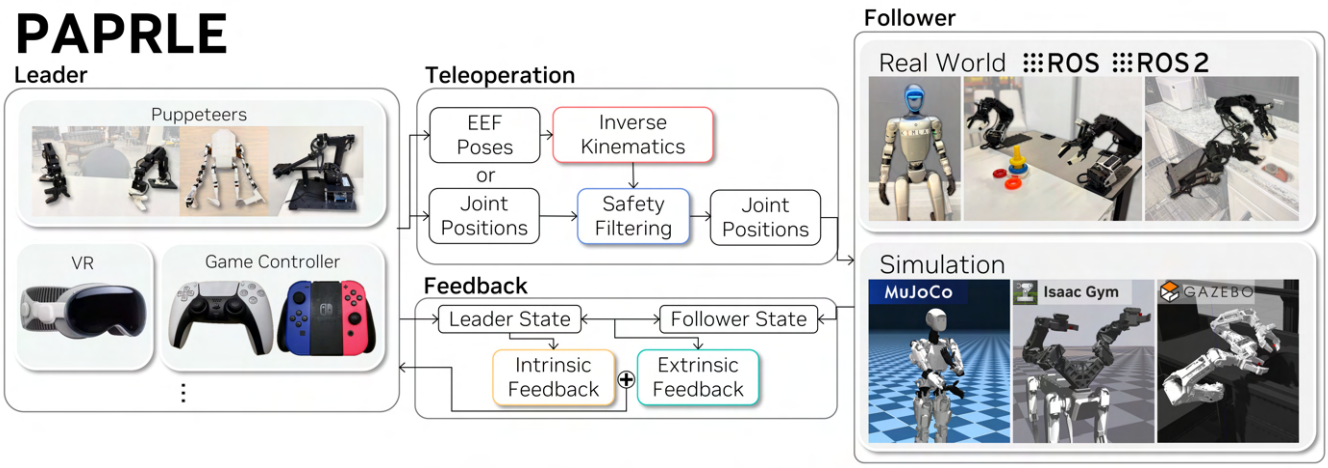

Fig. 3: **Overview of PAPRLE.**

vice can be designed as a scaled replica of a specific follower robot, fabricated using 3D-printed components. Equipped with low-cost motors, it can read joint states in real time and relay them for controlling follower robots. The device has a mount, which make this device can be easily mounted and removed in various places, enabling flexible deployment across various setups. Figure 2a illustrates the proposed puppeteer device alongside its mounting mechanism. The mount design is similar to the one from PAPRAS, which allows the same mounts to be used interchangeably between PAPRAS and puppeteer devices, enabling more diverse and reconfigurable setups. TTL communication and power are then routed through the mount to the leader device. The leader device is held in place with a simple retaining bar. This allows for quick swapping of leader devices using a common mounting base. For more information on the mounting mechanism, please refer to [1] and [11].

Using this pluggable design, multiple types of puppeteer devices can be interchanged and arranged in different configurations. Figure 2b, 2c, 2d show example configurations realized using the same set of puppeteer devices. The reconfiguration process is simple; users can simply unplug a device from one mount and plug it into another. The same mount can also accommodate entirely different types of arm. Figure 2e, 2f, 2g shows the example of using different puppeteer device for the same mount. This plug-and-play functionality of leader devices enables users to swap between different puppeteer devices or configurations, supporting a wide range of manipulation scenarios with minimal setup time.

IV. PAPRLE Teleoperation System

The overall architecture of the system is illustrated in Figure 3. PAPRLE consists of four high-level modules: (1) leader, (2) teleoperation, (3) follower and (4) feedback module. This modular framework enables flexible and scalable data collection, supporting both real and simulated environments. A detailed explanation of each module is provided in following sections. Before getting into the details of each module, we begin with an overview of a typical teleoperation session to provide better context of the system.

Algorithm 1 Teleoperation Session

Require: $cfg_{leader}, cfg_{follower}, cfg_{env}$

```
leader = init_leader(cfg_leader)
teleop = init_teleop(cfg_follower)
follower = init_follower(cfg_follower, cfg_env)
feedback = init_feedback(leader, teleop, follower)
feedback.start_thread()
# Initialization Phase
wait_until_leader_starts()
cmd = leader.get_status()
qpos = teleop.process(cmd)
follower.move_to(qpos)
# Teleoperation Loop
while not shutdown do
    cmd = leader.get_status()
    qpos = teleop.process(cmd)
    follower.step(qpos)
    if leader.requested_end then
        # Reset
        follower.move_to(base_pos)
        wait_until_leader_starts()
        cmd = leader.get_status()
        qpos = teleop.process(cmd)
        follower.move_to(qpos)
    end if
end while
```

A. Teleoperation Session Workflow

The overall procedure of a teleoperation session is illustrated in the pseudocode shown in Algorithm 1. To initiate a teleoperation session, the user must specify the leader device, the follower robot, and the target environment. Based on these inputs, the system selects the appropriate configuration files and uses them to set up the teleoperation session. For example, if a user want to control the PAPRAS robot using a VR device in the MuJoCo environment, the selected

configurations would be: (1) cfg_{leader} for the VR device, (2) $cfg_{follower}$ for the PAPRAS robot, and (3) cfg_{env} for the MuJoCo environment. To switch from a MuJoCo simulation to a hardware setup, the system simply change cfg_{env} to the corresponding configuration for ROS1 or ROS2, while using the same leader and follower configuration files. To setup a new follower robot, an URDF file is needed and the user just needs to write a corresponding configuration file.

At the start of each teleoperation session, the follower robot is assumed to have a predefined base pose. The robot first moves to this base pose and waits for all other modules to be initialized. Once initialization is complete, the system awaits a start signal from the operator. Upon leader module detects the start signal, the follower robot slowly moves to the posture that resembles the operator's initial pose. The teleoperation session is structured as a simple for-loop, following a format similar to the gym environment[1], consisting of repeated observation and action selection. This design is intentionally adopted to facilitate future integration with AI agents. At each iteration, commands are obtained from the leader module, and processed through the teleoperation module. Teleoperation module translates the commands from the leader into joint-space actions, and the resulting command is sent to the follower module to actuate the robot. Concurrently, the feedback module runs in the background, relaying feedback to the leader module.

B. Leader Module

The leader module serves as the interface between each control device and PAPRLE system. This module translates input signals from the leader device into appropriate command formats for the teleoperation module. Depending on the configuration of the paired follower robot, the leader module outputs either (1) joint positions or (2) end-effector poses represented as $\mathrm{SE}(3)$ transformations. When the puppeteer has the same joint configuration as the follower robot, the leader module outputs joint positions as commands. However, one-to-one joint mapping is not feasible when a different type of puppeteer device is used as the leader, or when the device does not have joint positions such as gaming controller or VR device. In these cases, leader module outputs end-effector poses, and the following teleoperation module performs inverse kinematics (IK) to compute joint-level commands for the follower robot. The end-effector pose $\mathbf{T}$ at time step t is defined as a homogeneous transformation matrix:

$$\mathbf{T}_t = \mathbf{Rt}\mathbf{0}1, \tag{1}$$

where $\mathbf{R} \in SO(3)$ is the rotation matrix representing orientation, and $\mathbf{t} \in R^3$ is the translation vector representing position. At the start of a teleoperation session, the leader module stores the initial end-effector pose of the leader device, denoted as $\mathbf{T}_0^{leader} \in SE(3)$, while the teleoperation module stores the initial end-effector pose of the follower robot as $\mathbf{T}_0^{follower}$. If the leader device is a puppeteer, the end-effector pose is computed via forward kinematics using the URDF model. For hand-held devices, the initial pose can be set to the identity matrix $\mathbf{I}$. For VR-based leaders, the human wrist pose is used as the end-effector pose. During the teleoperation session, the leader device continuously computes its current end-effector pose, denoted as $\mathbf{T}_{0\to t}^{leader} \in SE(3)$. The control command is defined as the pose delta from the initial pose:

$$\mathbf{T}_{0\to t}^{leader} = \left(\mathbf{T}_0^{leader}\right)^{-1} \mathbf{T}_t^{leader} = \mathbf{R}_{0\to t}^{leader} \mathbf{t}_{0\to t}^{leader} \mathbf{0}1, \tag{2}$$

where $\mathbf{R}_{0\to t}^{leader}$ denotes the relative rotation and $\mathbf{t}_{0\to t}^{leader}$ denotes translation from time 0 to t.

To account for differences in scale between the leader and follower devices, a scalar factor $s \in R$ is applied to the translational component of $\mathbf{T}_{0\to t}^{leader}$. The scaled pose is then defined as:

$$\widetilde{\mathbf{T}}_t^{leader} = \mathbf{R}_{0\to t}^{leader} s \cdot \mathbf{t}_{0\to t}^{leader} \mathbf{0}1. \tag{3}$$

The scaled pose $\widetilde{\mathbf{T}}_t^{leader}$ is then forwarded to the teleoperation module and applied to the follower's initial pose to compute the follower's target pose $\mathbf{T}_{t,cmd}^{follower}$:

$$\mathbf{T}_{t,cmd}^{follower} = \mathbf{T}_0^{follower} \cdot \widetilde{\mathbf{T}}_t^{leader}. \tag{4}$$

By adopting this delta-based end-effector command, our teleoperation framework enables seamless integration of diverse leader devices.

The list of supported leader devices is shown in Table I, categorized by device type, supported command format, number of controllable limbs.

TABLE I: **Supported Leader Devices.**

Type	Name	Cmd Type	# of Limbs
Puppeteers	7DoF PAPRAS 6DoF PAPRAS UR5 OM-Y	Joint Pos, EEF Pose	Unbounded
VR Headset	VisionPro	EEF Pose	≤ 2
Joystick	Joycon Dualsense	EEF Pose EEF Pose	1 1
Etc	Offline Trajectory	Joint Pos, EEF Pose	Unbounded

- **Puppeteer devices** replicate a scaled-down structure of a specific target follower robot. The list is the devices we have tested, including UR5 from Universal Robotics[2], and Open Manipulator Y model (OM-Y) from ROBOTIS[3]. Users can also design and use their own puppeteer devices.
- **VisionPro**, a VR headset from Apple, provides estimated human wrist poses, enabling end-effector pose control of up to two limbs. This device is particularly useful when a lightweight and portable teleoperation setup is preferred.

[1] https://github.com/openai/gym

[2] https://www.universal-robots.com/products/ur5e/

[3] https://www.dynamixel.com/omy.php

- **Gaming Controller** such as Joycon from Nintendo Switch[4] and DualSense from PlayStation 5[5].
- **Offline trajectory** can also serve as leader inputs, typically in the form of pre-recorded datasets streamed into the system. For example, end-effector trajectory collected from Universal Manipulation Interface (UMI) [12] can be used as a leader.

Importantly, our system does not assume a fixed number of limbs. PAPRLE supports multi-limb control within a single teleoperation session, enabling flexible configurations. In case of pluggable puppeteer devices, designed to be mountable in the same plug-and-play manner as PAPRAS arms, the leader configuration can be physically matched to that of the follower robot. For example, two puppeteers can be mounted for a dual-arm follower roobot, while four puppeteers can enable full-limb teleoperation of a humanoid robot. Figure 8e shows an example of such multi-limb configuration. Moreover, the system is easily extensible: new type of leader devices can be integrated by implementing a simple class that periodically publishes commands in the appropriate format, allowing rapid support for novel hardware interfaces.

C. Teleoperation Module

The teleoperation module receives commands from the leader device and translate them into a control signal for the follower. This process consists of two main steps.

a) Command Interpretation: If the command is given as a delta end-effector pose $\widetilde{\mathbf{T}}_t^{leader}$, the module first computes the resulting target pose for the follower robot's end-effector $\mathbf{T}_{t,cmd}^{follower}$ using Equation 4.

To translate this command into joint positions of the follower robot, the module solves an inverse kinematics (IK) problem using a Jacobian-based iterative algorithm. Given the forward kinematics function f, the IK solver aims to find the joint positions $\mathbf{q}^*$ such that the resulting end-effector pose $\mathbf{f}(\mathbf{q})$ closely matches the desired pose $\mathbf{x}_d$. Starting from the initial guess $\mathbf{q}_0$, the solver iteratively updates the joint positions by computing the pose error $\mathbf{e}$, and mapping it into the joint space using the Jacobian $\mathbf{J}(\mathbf{q})$ as:

$$\dot{\mathbf{q}} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \rho^2\mathbf{I})\mathbf{e}, \tag{5}$$

where ρ^2 denotes the damping factor, to handle singularity of Jacobian. Since this is a real-time teleoperation scenario, we can assume that the end-effector does not move significantly between consecutive commands. Thus, we set the initial guess $\mathbf{q}_0$ as the current joint position of the follower robot for faster convergence.

Additionally, when the follower robot has kinematic redundancies, the leader's end-effector motion may not directly translate into a natural or desirable posture for the follower. Figure 4 illustrates such a case: although both robots achieve the same end-effector pose, their postures differ. In the first example, the first joint is excessively tilted, resulting in large base movements. Depending on user preferences and task requirements, such configurations may or may not be desirable. To incorporate user-defined preferences, we introduce a simple optimization problem that penalizes the motion of specific joints:

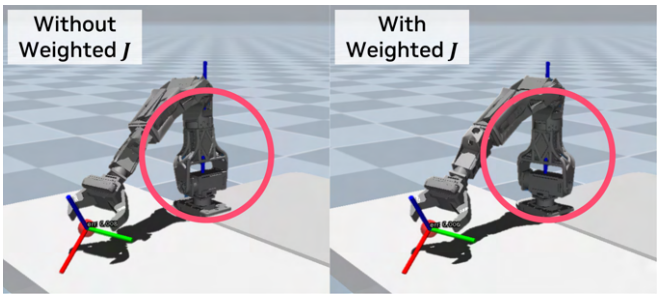

Fig. 4: **Examples of differences of IK results.**

$$\min_{\dot{q}} \ \dot{\mathbf{q}}^{\mathbf{T}}\mathbf{P}\dot{\mathbf{q}} \ , \quad \text{subject to } \mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{x}}, \tag{6}$$

where $\mathbf{P}$ is a diagonal matrix which has user-defined priorities of each joint.

For example, in Figure 4, the first joint is weighted by setting the corresponding diagonal entry in $\mathbf{P}[0,0] = 2$, thus discouraging excessive movement of the base and favoring more distal joints. Using the method of Lagrange mutlipliers, the solution for $\dot{q}$ can be expressed as:

$$\dot{\mathbf{q}} = \mathbf{P}^{-1}\mathbf{J}^{\mathbf{T}}(\mathbf{J}\mathbf{P}^{-1}\mathbf{J}^{\mathbf{T}} + \rho^2\mathbf{I})^{-1}\mathbf{e} \tag{7}$$

This formulation allows the IK solver to bias the solution toward or against specific joints as needed. Table III compares results obtained with weighted versus unweighted IK. By default, $\mathbf{P}$ is the identity matrix $\mathbf{I}$, and users only need to adjust the diagonal entries to encode their preferences.

Using this IK algorithm, the teleoperation module translates the leader command into the joint positions of the follower robot. This design allows the module to accommodate different types of leader devices, depending on whether they operate in task space or joint space. If the command is already provided in joint space, this step is skipped.

b) Safety Filtering: After interpretation into joint positions, the module performs safety checks, including collision detection and joint limits. It also constrains the maximum velocity of each joint to ensure safety. Outputs from this filtering becomes the control signal for follower robots.

D. Follower Module

The follower module receives joint position commands and sends them to the follower robot. For simulation, simulator-specific functions are implemented to set the joint positions of the robot. The supported environments are listed in Section V. For hardware control, we leverage the Robot Operating System (ROS) to simplify integration and implementation. Since ROS provides a standardized interface for communication between software and hardware, we can easily integrate wide range of robots for PAPRLE, utilizing existing drivers and tools.

[4] https://www.nintendo.com/store/hardware/joy-con-and-controllers/

[5] https://www.playstation.com/accessories/dualsense-wireless-controller/

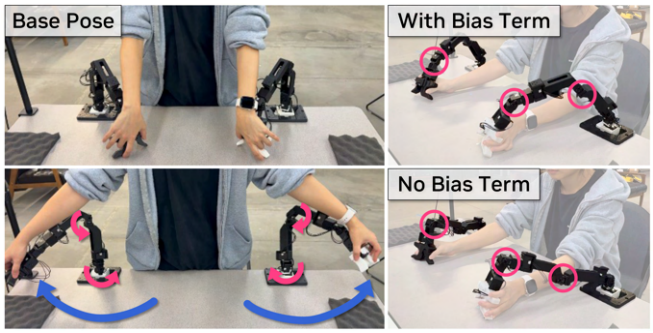

Fig. 5: **Example of Intrinsic Feedback.**

E. Feedback Module

Feedback module is not included in the teleoperation loop, rather this module runs as a separate thread asynchronously, to ensure fast and smooth feedback signals to the leader device. This module gathers information from the leader module and the follower module, and calculate corresponding feedback to the leader device. PAPRLE currently provides feedback tailored for puppeteer devices, with plans to potentially integrate diverse feedback types for other leader devices in the future. We categorized the feedback type into two types, (1) intrinsic feedback and (2) extrinsic feedback. Intrinsic feedback is solely based on the leader model, and extrinsic feedback comes from the difference between the follower and the leader.

a) Intrinsic Feedback: The main form of intrinsic feedback τ_{bias} is a biasing mechanism for the puppeteer devices. Each leader puppeteer has a predefined base pose according to its configuration, and the biasing force term is computed as

$$\tau_{bias} = \mathbf{q}_{leader}(t) - \mathbf{q}_{leader}^{base}. \tag{8}$$

This feedback applies a compensatory force to the leader device, reducing the physical effort required by the user during teleoperation. Figure 5 illustrates how this compensation force operates and helps the user. When the user's motion causes the elbow of the leader to bend in an unnatural direction, the system generates a corrective torque that gently guides it back toward the base configuration. This bias term helps the operator to maintain a natural posture of the leader device and provides an implicit hint about the workspace.

b) Extrinsic Feedback : Extrinsic feedback $\tau_{tracking}$ is based on the relationship between the leader device and the follower robot. The feedback module continuously monitors the follower's actual pose and provides corrective feedback to the leader device when the follower fails to track the commands. This feedback mechanism is designed to provide guidance or warning signals during unusual situations in teleoperation sessions. For instance, when the follower fails to track the leader due to velocity limits or obstacles, the position-based extrinsic feedback alerts the operator to the current condition of the follower. When the command type is joint position, which means that the leader and the follower have the same joint configuration, the intrinsic feedback $\tau_{tracking}$ can be computed as the direct difference between leader's joint position $\mathbf{q}_{leader}$ and follower's joint position $\mathbf{q}_{follower}$:

$$\tau_{tracking}(t) = \mathbf{K}_p(\mathbf{q}_{leader}(t) - \mathbf{q}_{follower}(t)), \tag{9}$$

where $\mathbf{K}_p$ is a diagonal matrix to scale the feedback torques. Such feedback naturally conveys interaction forces in the gripper, since contact with an object typically results in a deviation between the intended and actual pose of the follower's gripper.

Even when the leader and follower robots differ in joint configuration, the feedback module can still generate force feedback using task-space representations. Specifically, we compute the relative difference between current follower's pose $\mathbf{T}_t^{follower}$ and the target pose $\mathbf{T}_{t,cmd}^{follower}$ (from Eq. 4) and it is then mapped into the Lie algebra $se(3)$ using the logarithmic map, yielding a 6D twist vector $\Delta\mathbf{x}_{eef}$ that captures both translational and rotational error:

$$\Delta\mathbf{x}_{eef} = \log\left(\mathbf{T}_t^{follower^{-1}}\mathbf{T}_{t,cmd}^{follower}\right). \tag{10}$$

To reflect this error on the leader device, the task-space delta is projected into the leader's joint space using its damped pseudo-inverse Jacobian:

$$\Delta\mathbf{q}_{leader} = \mathbf{J}_{leader}^T(\mathbf{J}_{leader}\mathbf{J}_{leader}^T + \rho_{leader}^2\mathbf{I})^{-1}\,\Delta\mathbf{x}_{eef}, \tag{11}$$

where $\mathbf{J}_{leader}$ denotes a jacobian matrix of leader model, and ρ_{leader} denotes a damping factor to handle singularities. This joint-space correction can be used to generate torque-based feedback by applying a proportional gain:

$$\tau_{tracking} = \mathbf{K}_p\,\Delta\mathbf{q}_{leader}. \tag{12}$$

Note that $\mathbf{K}_p$ is set with low values to prevent the extrinsic feedback propagating back into the control signal. In addition, to ensure that excessive torques are not applied to the motors, the final feedback values $\tau_{feedback}$ are clipped as follows:

$$\tau_{feedback} = \min(\max(\tau_{bias} + \tau_{tracking}, -\tau_{max}), \tau_{max}), \tag{13}$$

where τ_{max} denotes a vector of maximum torque values which can be applied to each motor.

V. Supported Follower Robots, Environments

The proposed PAPRLE system enables users to operate a wide range of robots across diverse environments. For hardware, we support both ROS1 [13] and ROS2 [14] which facilitates the seamless integration of new robots. For simulation, our system supports MuJoCo [15], Isaac Gym [16], and Gazebo [17] (for ROS)), providing flexibility for rapid prototyping and experimentation. We validated the system on various configurations of PAPRAS as well as the Open manipulator Y from ROBOTIS, AI Worker from ROBOTIS, and Unitree's Humanoid Robot G1. As part of the PAPRLE development, we also created a 7-DOF variant of PAPRAS, adding an additional joint to the original design. All PAPRAS models, including 3D-printable components and hardware interfaces, will be released as open-source.

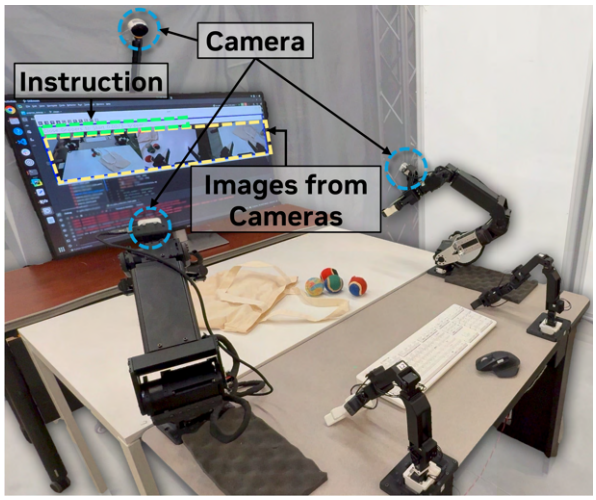

Fig. 6: **Example of Data collection Setup using PAPRLE**.

VI. Demonstration

In this section, we show the use cases of the proposed PAPRLE, with each example illustrated in the accompanying video.

A. Example Teleoperation Setup for Data Collection

First, we can use the proposed PAPRLE for data collection in robot learning. The example setup is shown in Figure 6. We mounted two PAPRAS on a table as a follower, and also mounted the puppeteer devices on the same table accordingly. Each PAPRAS arm supports one camera to be connected through USB, and we can also connect external cameras to the system. When a teleoperation session is launched, PAPRLE shows the real-time images from the connected cameras, collision status, and instruction for the operator. For example, in the Figure 6, the pop-up window tells the operator to 'close the grippers to start teleoperation.' PAPRLE automatically records data with timestamps in the background at each step. The dataset consists of joint states (pos, vel, effort) of the follower, images from cameras, command from the leader. This collected dataset can be processed and utilized for training a manipulation policy. Additionally, users have the flexibility to specify which data to collect.

B. Diverse Leaders

With PAPRLE, we can control the same robot with diverse types of leader devices. Figure 7 shows an example of controlling PAPRAS arms with different types of leaders. In Figure 7a, the operator is using the puppeteer which has the same joint configuration as the PAPRAS. In this case, the joint positions of the leader are directly forwarded to the follower robot. In Figure 7b, the operator is using the UR5 puppeteer, which is adapted from [2], and Figure 7c is the off-the-shelf leader device from ROBOTIS, which has the same configuration as Open Manipulator-Y robot from the company. Since the puppeteer has a different joint configuration from PAPRAS, PAPRLE controls the follower robot based on end-effector position of the leader device. Independent of the leader device's joint configuration, our system allows the operator to control the same follower robot and perform the same task with force feedback.

PAPRLE also accommodates non-puppeteer devices. Figure 7d and 7e illustrate the use of gaming controllers as leader for controlling the same PAPRAS arm. Also, PAPRLE supports VR devices as leader, as shown in Figure 7f, where the operator uses Apple Vision Pro to control two PAPRAS arms. Furthermore, PAPRLE enables the use of pre-collected datasets for robot control, even when the data is not gathered using PAPRLE itself. Figure 7g showcases this capability, depicting PAPRAS arms that reproduce motions derived from a dataset collected with UMI [12], a hand-held gripper interface used to record bimanual manipulation demonstrations for training robot policies. These examples demonstrate the versatility of PAPRLE, enabling seamless control of robots across various leader devices and data sources, facilitating flexible and robust teleoperation for a wide range of tasks and applications.

C. Diverse Follower Configurations

In this section, we present examples of running diverse configurations of follower robots, shown in Figure 9.

1) PAPRAS: Using PAPRAS, we can configure a variety of limb arrangements. For example, in Figure 8a, two PAPRAS arms are mounted on a table for tabletop tasks. In Figure 8b, the same pair of arms is mounted on a kitchen counter and wall to facilitate loading dishes into a dishwasher from the sink. Figure 8c shows the same arms mounted on a Boston Dynamics Spot robot for mobile manipulation tasks. These diverse configurations are easily operable by PAPRLE, particularly with the pluggable puppeteer devices. Notably, Figures 8a, 8b, 8c all demonstrate control using the same pair of puppeteer devices, which are simply remounted in different locations to match each setup.

2) Humanoid: PAPRLE can also be applied to the teleoperation of humanoid robots. Figure 8d and 8e show the examples of teleoperation of humanoid. With PAPRLE, users can teleoperate not only the upper body of a humanoid, but also all four limbs simultaneously.

3) Commercial Robotic Arm: PAPRLE further supports the teleoperation of commercial robotic arms beyond PAPRAS. Figure 8f presents an example of teleoperating the Open Manipulator-Y using the PAPRLE framework. The accompanying video demonstrates all the examples described above.

VII. Analysis

A. Runtime

We analyze the runtime performance of the proposed PAPRLE system with varying numbers of attached limbs. Specifically, we collected one minute of teleoperation data encompassing diverse motions (including general motions with varying speed, intentional collisions), and measured the average execution time and worst case latency of a single iteration of the for loop (lines 11–19 in Algorithm 1)

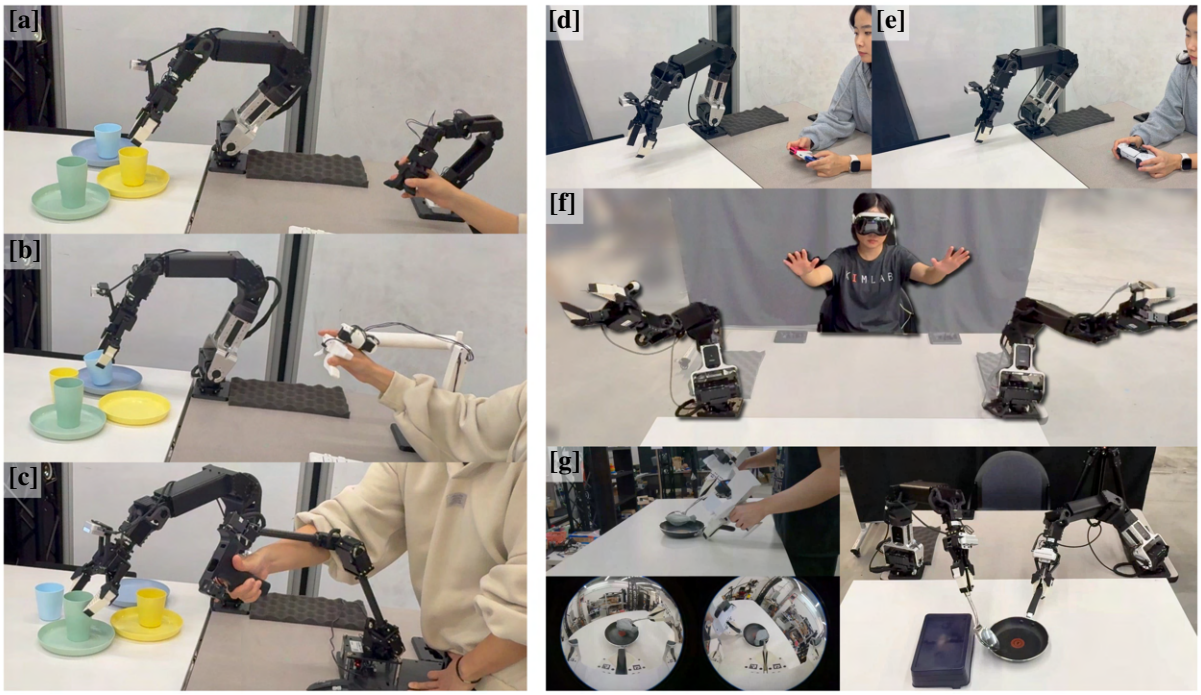

Fig. 7: **Examples of different devices as leader for controlling PAPRAS arm. [a-c]** Puppeteer devices designed for PAPRAS, UR5, and Open Manipulator Y model, respectively. **[d-e]** Gaming controllers (Joycon and Dualsense Controller), **[f]** VR device (Apple VisionPro), **[g]** Pre-collected dataset using UMI [12].

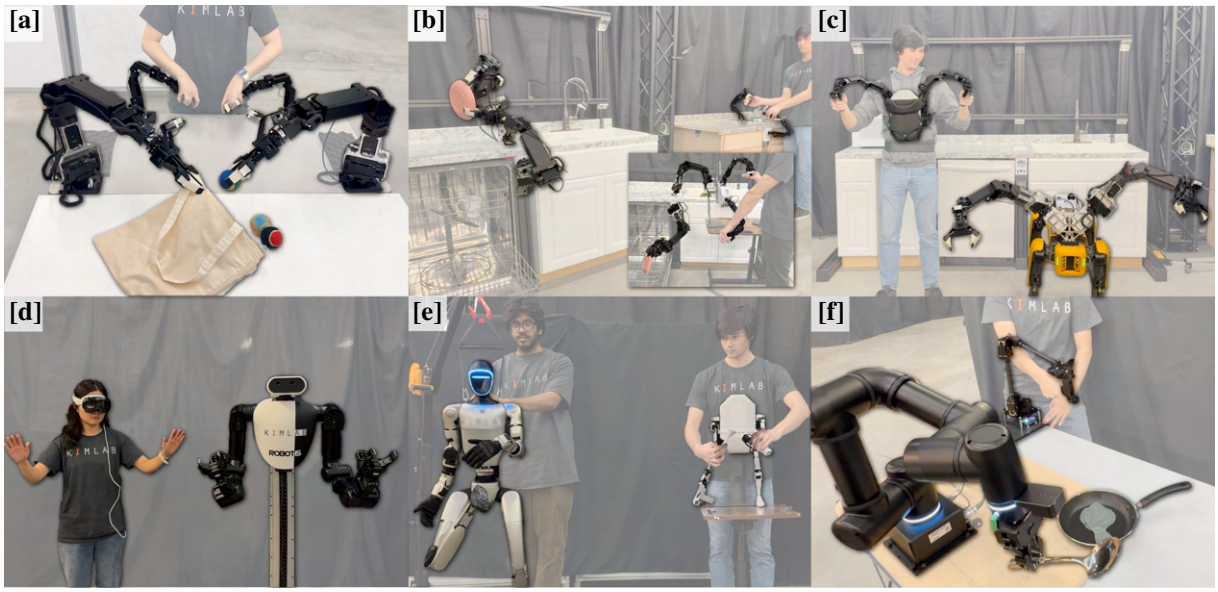

Fig. 8: **Overview of diverse follower setups.** Best view in color.

under different command types. All the experiments were conducted on a laptop with CPU: AMD Ryzen AI 9 HX 370 (12 Cores, 2GHz) and an NVIDIA GeForce RTX 4070 Laptop GPU.

The results are shown in Table II. When using end-effector (EEF) pose-based commands, the per-step runtime increases, primarily due to the computational overhead of IK and collision checking. Currently, PAPRLE processes each limb sequentially in a for-loop, leading to increased runtime as the number of limbs grows. Nonetheless, even with all four limbs enabled, PAPRLE is able to sustain a control loop frequency close to 50 Hz. Although the worst-case latency for four-limb EEF control reached 45 ms, occurrences where the loop exceeded 20 ms were rare (0.6% of cases). The results indicate that PAPRLE achieves sufficient runtime efficiency to support real-time teleoperation and large-scale data collection.

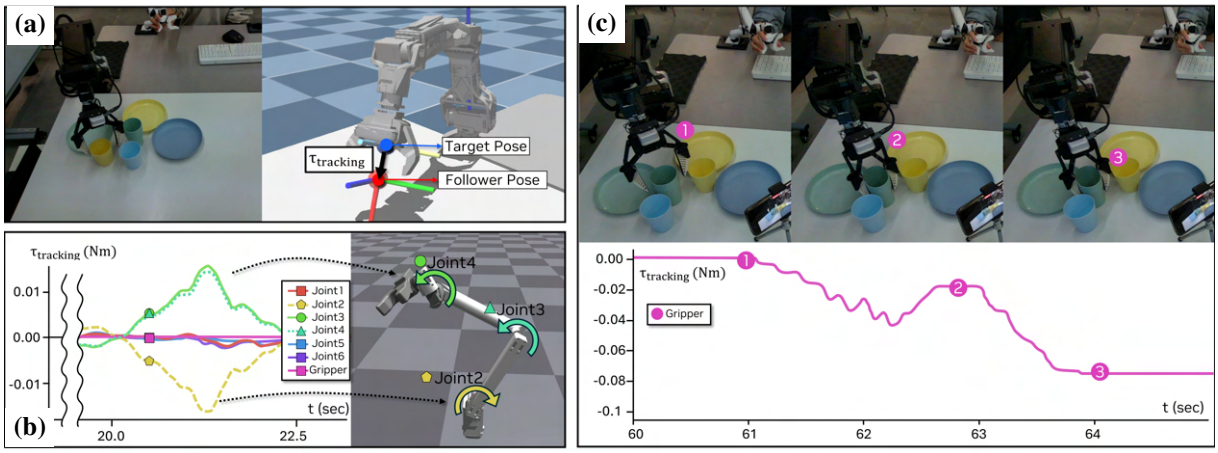

Fig. 9: **Examples of the extrinsic force feedback.**

TABLE II: Control step duration (ms) of the PAPRLE system for different number of limbs and command types.

	Direct Joint Mapping			EEF-pose based		
# of Limbs	**1**	**2**	**4**	**1**	**2**	**4**
Average	0.14	0.22	0.43	1.06	2.44	8.32
Worst Case	1.09	2.69	17.91	5.04	11.04	45.24
Std. Dev.	0.09	0.14	0.87	0.53	0.96	3.74

TABLE III: Inverse Kinematics Accuracy

Error (cm)	UR5 → PAPRAS (Unweighted)	UR5 → PAPRAS (Weighted)	OMY → PAPRAS (Unweighted)
Mean	0.49	0.48	0.54
Std	0.23	0.22	0.19
Max	1.98	1.65	2.45
Median	0.46	0.46	0.57
99% Quantile	0.96	0.96	0.87

B. Accuracy

We also analyze the accuracy with which the follower robot tracks the leader's commands, when they have different kinematic configurations. We collected several episodes of controlling a 7-DoF PAPRAS robot with two different leaders (UR5 puppeteer, Open Manipulator-Y puppeteer). The task, illustrated in Figure 7, is to sort colored cups into corresponding plates. By randomizing the initial positions of the cups and plates, we collected 20 episodes for each leader. Table III shows the error between the target end-effector pose command from the leader and the resulting end-effector pose of the follower robot. We observed that the majority of errors were under one centimeter, indicating accurate tracking performance despite differences in robot configuration.

Additionally, we evaluated the accuracy of the weighted IK using the same dataset, and the results are reported in the second column of Table III. In this experiment, we applied the weighting matrix $\mathbf{P}$ described in Section IV-C, which penalizes the use of the first joint ($\mathbf{P}[0,0] = 2$). The results indicate that the weighted IK achieves comparable accuracy to the unweighted case, demonstrating its practicality for teleoperation while allowing the incorporation of user preferences.

C. Leader feedback

In this section, we analyze how the force feedback occurs to the leader device, especially when the leader and the follower has different configuration. In Figure 9a, the operator is using an UR5 puppeteer to control PAPRAS. If the operator moves too fast, because of the velocity limit, the follower robot would not follow the leader command promptly. Due to the differences between the target command pose $\mathbf{T}_t^{follower}$ and the follower pose $\mathbf{T}_{t,cmd}^{follower}$, the feedback module applies $\tau_{tracking}$ to the leader device. Using Equation 11, the feedback module can calculate joint-level force feedback to leader device, even though the follower does not have the same joint configuration as leaders. Figure 9b shows the plot of how $\tau_{tracking}$ appears in each joint of the UR5 puppeteer. Since the feedback module wants to inform the leader that the follower is down below, each joint of the leader device gets the force feedback into the direction that the end-effector of the leader devices can be lowered.

Figure 9c illustrates how the gripper joint provides force feedback during interaction with objects. Feedback for the gripper joint is computed at the joint level as a scaled difference between the gripper joint positions of the leader and the follower. In Figure 9c, the force feedback occurs primarily in three stages: first, due to slight speed differences between the leader and follower grippers (stage 1), and second, when the follower gripper makes contact with an object while the operator continues squeezing (stage 2). At this point, the feedback force becomes noticeably strong, allowing the operator to physically sense resistance and naturally stop squeezing further. At stage 3, the operator continues to apply force beyond, the position discrepancy grows, leading to stronger feedback. This gripper feedback can provide the user as the sense of haptic feedback, which can enhance the accurate object manipulation.

VIII. Conclusion

We presented PAPRLE (Plug-And-Play Robotic Limb Environment), a modular and extensible ecosystem for teleoperation that enables flexible pairing between diverse leader devices and follower robots. By abstracting control interfaces and supporting both joint-space and task-space modalities, PAPRLE provides a unified framework for intuitive, real-time control across a variety of robot morphologies and environments. Its plug-and-play design allows rapid reconfiguration of multi-limb systems, facilitating scalable and reproducible data collection for learning-based robotics. Through real-world demonstrations, we validated PAPRLE's ability to handle diverse teleoperation scenarios with accurate tracking and responsive force feedback, even when leader and follower configurations differ. As the robotics community continues to pursue embodied AI through large-scale data, we believe PAPRLE offers a foundational platform to accelerate this progress.

ACKNOWLEDGMENTS

This work was partly supported by ROBOTIS and Toyota Research Institute.

References

[1] J. Kim, D. C. Mathur, K. Shin, and S. Taylor, "Papras: Plug-and-play robotic arm system," *arXiv preprint arXiv:2302.09655*, 2023.

[2] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel, "GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[3] T. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware," *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[4] J. J. Liu, Y. Li, K. Shaw, T. Tao, R. Salakhutdinov, and D. Pathak, "FACTR: Force-Attending Curriculum Training for Contact-Rich Policy Learning," in *Proceedings of Robotics: Science and Systems (RSS)*, 2025.

[5] A. Bazhenov, S. Satsevich, S. Egorov, F. Khabibullin, and D. Tsetserukou, "Echo: An Open-Source, Low-Cost Teleoperation System with Force Feedback for Dataset Collection in Robot Learning," 2025.

[6] R. Ding, Y. Qin, J. Zhu, C. Jia, S. Yang, R. Yang, X. Qi, and X. Wang, "Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning," *arXiv preprint arXiv:2407.03162*, 2024.

[7] X. Cheng, J. Li, S. Yang, G. Yang, and X. Wang, "Open-TeleVision: Teleoperation with Immersive Active Visual Feedback," in *8th Annual Conference on Robot Learning*, 2024.

[8] S. Yang, M. Liu, Y. Qin, R. Ding, J. Li, X. Cheng, R. Yang, S. Yi, and X. Wang, "ACE: A cross-platform and visual-exoskeletons system for low-cost dexterous teleoperation," in *8th Annual Conference on Robot Learning*, 2024.

[9] S. Dass, W. Ai, Y. Jiang, S. Singh, J. Hu, R. Zhang, P. Stone, B. Abbatematteo, and R. Martín-Martín, "TeleMoMa: A Modular and Versatile Teleoperation System for Mobile Manipulation," in *RSS 2024 Workshop: Data Generation for Robotics*, 2024.

[10] D. Honerkamp, H. Mahesheka, J. O. von Hartz, T. Welschehold, and A. Valada, "Whole-Body Teleoperation for Mobile Manipulation at Zero Added Cost," *IEEE Robotics and Automation Letters*, 2025.

[11] N. Myers, O. Kwon, S. Yamsani, and J. Kim, "CHILD (Controller for Humanoid Imitation and Live Demonstration): A Whole-Body Humanoid Teleoperation System," in *2025 IEEE-RAS 24th International Conference on Humanoid Robots (Humanoids)*, 2025.

[12] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song, "Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots," in *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[13] 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*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[14] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, no. 66, 2022.

[15] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and System (IROS)*, 2012.

[16] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning," 2021.

[17] 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 System (IROS)*, 2004.