Software Architectures for Enhanced Physical Human-Robot Interaction and Mixed Reality-based Rehabilitation

by

Dongjune Chang

A Dissertation
Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Submitted December 2025 to the
Graduate Supervisory Committee

Hyunglae Lee, Chair
Thomas Sugar
Hamid Marvi
Spring Berman
Christopher Buneo

ARIZONA STATE UNIVERSITY
May 2026

ABSTRACT

Physical human-robot interaction (pHRI) and mixed reality (MR)-based robotic rehabilitation impose stringent requirements on software systems: safety-critical real-time execution, heterogeneous sensor integration, and support for collaborative multi-user workflows. Conventional research prototypes, typically monolithic and single-threaded, lack the reliability and extensibility required for large-scale studies. This dissertation develops software architectures for pHRI and MR rehabilitation, guided by two principles: (1) parallel architectures that deliver real-time performance through non-blocking execution, and (2) modular architectures that enable scalable development through clear functional separation and decoupled communication.

For pHRI, these principles are instantiated in a shoulder rehabilitation exoskeleton with lock-free multi-threaded pipelines, a wearable upper-limb exoskeleton on a layered ROS architecture, and a robotic arm manipulator driven by a distributed task scheduler. They extend to MR rehabilitation through a gait-symmetry AR visual-distortion system, a single-user AR admittance platform, and a multi-user MR platform that formalizes visualization–control separation through policy-based configuration. They further extend to multi-robot and multi-agent intelligence through an MR materials-synthesis platform, a Single Source of Truth architecture deployed across heterogeneous robot platforms at two institutions, a multi-agent AI platform with retrieval-augmented discussion and four-level human-in-the-loop approval, a personal AI infrastructure integrating four language-model providers with a real-time C++ core, and a three-rate interaction-control architecture composing language-model reasoning, diffusion-based action generation, and Lie-group impedance and admittance control.

Validation supports the architectural claims. The shoulder exoskeleton achieves 250 Hz control with high impedance reliability ($R^2 > 0.97$) in a forty-participant

study; the wearable exoskeleton attains 500 Hz with >99.2% VAF in impedance replication; the robotic arm manipulator reduces interaction energy by approximately 45% through session-continuous Bayesian optimization. The multi-robot platform integrates heterogeneous robots from multiple manufacturers through YAML-only configuration with zero per-platform source-code modification. The multi-agent AI platform produced a sole-authored nuclear waste site compliance assessment that received "Paper of Note" and "Superior Paper" awards at the Waste Management Symposia 2025. The multi-user MR platform sustains real-time synchronization across head-mounted displays through a LAN-local authoritative server with a mesh VPN overlay. Together, these architectures transform prototype-level pHRI and MR rehabilitation systems into reliable platforms for long-term and multi-site studies.

DEDICATION

To Sola, my mother, my father, and Suhyeon.

ACKNOWLEDGMENTS

I thank my advisor, Prof. Hyunglae Lee, for the discipline of writing each sentence as one function and naming the specific quantity rather than the abstract noun. The clarity I aimed for in this dissertation is a direct attempt to follow his example. I thank my committee members for the feedback that shaped this dissertation in its final form.

I thank colleagues at the Neuromuscular Control and Human Robotics Laboratory at ASU. Prof. Seunghoon Hwang co-led the shoulder-stiffness biomechanics study, Dr. John Atkins designed the wearable exoskeleton hardware, Dr. Fatemeh Zahedi led the variable-damping Bayesian-optimization study, and Dr. Omik Save co-led the gait-adaptation visual feedback system with me and was instrumental in its development. Aditya Saxena, Ellory Oleen, and Soe Lin Paing carried the human-subject data collection that these studies depended on.

I thank Hamza Muzammal for the augmented reality rehabilitation framework and the multi-robot infrastructure work at Argonne. Prof. Christopher Buneo directed the proprioception and visual-motor research that motivates several chapters, and the exploration of mixed reality for biomechanics that Prof. Lee, Prof. Buneo, Hamza, and I began continues beyond this dissertation.

I thank Dr. Young Soo Park at Argonne National Laboratory for access to the heterogeneous robot fleet without which the cross-platform validation would not have been possible. I am also grateful to the ANL robotics group for the robot drivers and hardware infrastructure. The bilateral teleoperation controller was originally developed at the Interactive Robotic Systems Laboratory at KAIST under Prof. Jee-Hwan Ryu, and I thank Joong-Ku Lee, Seong-Su Park, and the IRiS Lab team for that code.

Two earlier mentors shaped the path that ultimately led to this dissertation. Prof. Jung Kim at the KAIST Biorobotics Laboratory first introduced me to the

fundamentals of robotics and helped me understand how robotic systems could extend beyond their original domains, including toward medical and human-centered applications. The industrial robotics experience I later gained at LG Electronics Inc. and Dongbu Robot Co., Ltd. further grounded this perspective in practical skills and engineering discipline before my work expanded into nuclear engineering. Prof. Youho Lee at Seoul National University then opened the door to Argonne National Laboratory, where I began learning teleoperation and nuclear-application robotics as a student aide before and during my doctoral studies, and where these diverse experiences eventually converged into multi-robot research.

My doctoral studies at ASU were supported by Prof. Hyunglae Lee. In his lab, I was able to broaden my robotics background through rehabilitation engineering, human–robot interaction, mixed-reality-based robotic systems, and the many hands-on challenges of building experimental systems from the ground up. Most importantly, Argonne National Laboratory became the place where these foundations could be integrated, tested, and developed into the larger robotic systems presented in this dissertation.

The Digital Modular Materials Robotic Platform was funded by an ASU–ANL joint Argonne LDRD project. I also gratefully acknowledge support from the U.S. Department of Energy, Office of Environmental Management, for related research activities that contributed to this work.

Finally, I thank Sola and my family—my mother, my father, and Suhyeon—whose long, quiet, and unreserved support through every year of this work is the reason this dissertation exists. I am also deeply grateful to my friends at ASU and beyond, who carried me through the harder periods of these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation and Research Context

Physical human-robot interaction (pHRI) and mixed reality (MR) rehabilitation systems must satisfy real-time control constraints, multi-modal sensor integration, and human safety requirements that standard industrial robot architectures do not address. Unlike pick-and-place manipulators that run fixed trajectories, pHRI systems must maintain control frequencies above 200 Hz with sub-millisecond jitter during direct physical contact (Colgate and Schenkel, 1997), simultaneously acquire force, motion, and EMG data at different rates, and remain stable across months of multi-participant data collection.

This dissertation addresses the question: *What software architecture patterns allow pHRI and MR rehabilitation systems to achieve deterministic real-time control, multi-modal sensor integration, and modular team development on commodity hardware?*

1.2 Background: Why Software Architecture Matters in pHRI and MR-Based Robotic Systems

1.2.1 Requirements in Physical Human-Robot Interaction

Physical human-robot interaction differs from conventional industrial robotics in four ways. The human and the robot are in direct physical contact, so the controller must guarantee human safety throughout the session rather than only at task boundaries. Force/torque sensors, motion-capture systems, electromyography sensors, and visual feedback streams must operate simultaneously at different sampling

rates. The human sensorimotor system imposes timing requirements that are stricter than those of industrial trajectories: realistic force rendering and haptic interactions require control frequencies above 200 Hz (Colgate and Brown, 1994). A timing violation in this regime can result in human injury (Colgate and Hogan, 1988), so deterministic execution is a safety requirement, not an optimization target.

1.2.2 Requirements for Enhanced pHRI Research

Research-grade pHRI systems impose three additional requirements. New sensing modalities must be incorporated without disrupting existing functionality, so the architecture must be modular at the sensor-integration boundary. Large-scale studies require the platform to operate continuously across months with minimal failures. The same platform must support diverse experimental protocols and research questions without per-study rewrites.

1.2.3 Requirements in Mixed Reality-Based Robotic Systems

MR-based rehabilitation adds four further requirements. Remote therapists and local patients must share consistent spatial representations of robots and environments across network boundaries. Virtual robot representations must accurately reflect physical robot states while supporting independent visualization modes for different users. The system must operate across heterogeneous hardware that ranges from HoloLens devices to desktop computers and robotic controllers, each with different computational capabilities. Clinical deployments span local-area networks (LAN), virtual private network (VPN) connections, and cloud-based routing, so the connection layer must adapt to the deployment topology rather than assume a fixed one.

1.2.4 Common Architectures and Their Limits

Several middleware frameworks address subsets of these requirements. ROS (Quigley *et al.*, 2009) provides publish–subscribe communication and package management but does not enforce real-time execution. OROCOS (Bruyninckx, 2001) provides hard real-time component execution but lacks built-in multi-modal sensor fusion or team development support. YARP (Metta *et al.*, 2006) provides cross-platform distributed processing but targets humanoid platforms rather than pHRI research workflows. No existing framework integrates real-time determinism, multi-modal sensor fusion, and modular team development in a single architecture.

1.2.5 The Monolithic Prototype Problem

In practice, most pHRI research groups build monolithic, single-developer codebases in which sensor acquisition, control computation, and data logging run in a single thread. Such prototypes are convenient for proof-of-concept demonstrations but expose four limitations as the study scales. Adding a force/torque sensor to such a system can drop the control rate from 250 Hz to below 100 Hz because the serial read blocks the entire loop, an effect that the author observed during System 1 development (Chapter 2). Unreliable execution leads to missing trials and compromised datasets in long-running studies. Tightly coupled code makes parallel development by multiple researchers impractical. And session reliability across months requires architectural guarantees that monolithic designs cannot provide.

1.3 Research Objectives

The dissertation pursues two architectural objectives.

1.3.1 *Objective 1: Parallel Architecture for Real-Time Performance*

Develop a parallel processing architecture that minimizes latency and jitter while providing runtime fault tolerance. The architecture exploits multi-core processors through non-blocking multi-threaded sensor acquisition and lock-free data structures (concurrent data structures that use atomic operations instead of mutual-exclusion locks, so that no single thread blocks the others) for inter-thread communication, and enforces consistent control cycles through explicit timing mechanisms.

1.3.2 *Objective 2: Modular Architecture for Scalable Development*

Design a layered architecture in which hardware interfaces, control logic, and analysis code are developed, tested, and modified independently. Component isolation is provided by language-agnostic message interfaces and distributed communication substrates (ROS, ZeroMQ (Hintjens, 2013)), and parameter changes propagate through declarative configuration rather than recompilation.

1.4 Research Contributions

The two objectives are validated across the systems described in the following chapters and the integration demonstrations of Chapter 13. The recurring architectural patterns are:

1. **Lock-free real-time architecture** (Systems 1–2, Chapters 2–3): parallel, non-blocking processing patterns that achieve deterministic control (250 Hz with $<$0.5 ms jitter; 500 Hz transparent impedance) without a real-time operating system.

2. **Multi-language integration pattern** (System 1, Chapter 2): a Python glue architecture that combines a C++ real-time control layer, a Python application

layer for rapid prototyping, and an R statistical analysis layer through clean file-based and topic-based interfaces.

3. **Message-driven distributed scheduling** (System 3, Chapter 4): a ZeroMQ ROUTER–DEALER task scheduler with JSON-driven configuration that supports session-to-session continuity and algorithm swaps without stopping the controller.

4. **Visualization–control separation through policy** (Systems 4–5, Chapters 5–7): an MR architecture pattern in which visualization mode, control mode, and user role are independent configuration axes, allowing a single deployment to host on-site and remote users with different permissions.

5. **Mixed reality integration framework** (System 5, Chapters 7–13): real-time state synchronization through a LAN-local authoritative server (Mirror Networking SyncVar/KCP), QR-based co-registration between Unity and ROS coordinate frames, and a peer-to-peer mesh VPN overlay for multi-site access without cloud relays.

6. **Declarative configuration hierarchy** (System 8, Chapter 9): a base–override YAML scheme through which heterogeneous robot platforms are integrated by configuration alone, with zero source-code modification across multiple platforms at two institutions.

7. **Multi-agent reasoning with human-in-the-loop approval** (System 9, Chapter 10): six specialized agent types, four swappable VLM backends, and a four-level approval system, all governed by the same configuration hierarchy used for robot configuration.

8. **Personal AI infrastructure with real-time hot path** (System 10, Chapter 11): a multi-provider LLM abstraction, a provider-agnostic tool engine, a microservice skills platform, a C++ safety and trajectory core, and a zero-copy shared-memory vision transport, all maintained by a single researcher as the deployment substrate for Systems 9 and 11.

9. **Three-rate hierarchical interaction control** (System 11, Chapter 12): a multi-agent reasoning layer, a diffusion-interaction policy, and a Lie-group interaction control library composed into three time-decoupled rates, with the agent and policy layers dispatching a unified library of seven interaction controllers through a common C++ interface hierarchy.

10. **Operation-layer services above a single compiled controller binary** (System 10, Chapter 11): a cascade-verified bringup substrate, a declarative task-contract orchestrator with wrapper-orthogonal recording (sub-millimeter motion-plane invariance under wrapper substitution), a voice-dispatchable tool surface, a decentralized device registry, and a persistence layer that writes session, message, and run state through a single ABC-backed Mongo backend.

Chapter 13 composes Systems 6–11 into six integration demonstrations spanning bilateral teleoperation, multi-agent supervision over a dexterous manipulator, and multi-user distributed mixed reality, and Chapter 14 summarizes the cross-chapter patterns.

1.4.1 Scope of the Architectural Claims

The architectural claims of this dissertation hold within a defined operating envelope. Real-time control rates target the 200 Hz–1 kHz range on commodity hardware (Linux without a real-time kernel, consumer-grade workstations and edge

compute nodes); sub-microsecond determinism, formal worst-case execution-time guarantees, and certified safety-critical operation are outside the scope of the present work and would require a real-time operating system, formal methods, or hardware-level enforcement that this dissertation does not develop. The multi-robot and multi-site claims of Chapter 9 are validated across heterogeneous platforms at two institutions through configuration alone; clinical-network deployment, cross-vendor interoperability beyond the platforms listed in Table 9.2, and fleet-scale concurrent operation remain future work. The multi-agent AI substrate of Chapter 10 is validated empirically on cross-corpus regulatory assessment; rehabilitation-domain validation against clinical ground truth is identified in Chapter 14. These boundaries are stated here so that the patterns documented in the following chapters can be evaluated against the conditions under which they were demonstrated rather than against unstated generality.

1.5 Dissertation Organization

This dissertation is organized into four parts plus a conclusion:

Part I: Software Architecture for Physical Human-Robot Interaction (Chapters 2–4)

- **System 1** (Chapter 2): Shoulder Rehabilitation Exoskeleton Robot—lock-free architecture evolution from 166 Hz to 250 Hz control, enabling a 40-participant biomechanics study (Chang *et al.*, 2021; Hwang *et al.*, 2024a).
- **System 2** (Chapter 3): Wearable Upper-limb Exoskeleton Robot—ROS-based three-layer modular architecture achieving 500 Hz transparent impedance control (Atkins *et al.*, 2024).

- **System 3** (Chapter 4): Robotic Arm Manipulator—ZeroMQ-based distributed task scheduler for Bayesian optimization with session-to-session continuity (Zahedi *et al.*, 2022).

Part II: Software Architecture for Mixed Reality-Based Rehabilitation (Chapters 5–7)

- **System 4** (Chapter 5): Gait Symmetry via Mixed Reality Visual Distortion—real-time visual feedback architecture integrating split-belt treadmill with HoloLens AR (Save *et al.*, 2026).

- **System 5** (Chapter 6): Single-User AR Admittance Rehabilitation—500 Hz Cartesian admittance loop on a 6-DoF collaborative robot, dexterous hand-over-hand grasping, and HoloLens eye-tracked co-located AR feedback.

- **System 6** (Chapter 7): Multi-User MR Robot-Aided Rehabilitation—distributed architecture for synchronized multi-user MR experiences with shared robot control via Mirror Networking (LAN-local authoritative SyncVar over KCP) and ROS2.

Part III: Software Architecture for Multi-Robot and Multi-Agent Systems (Chapters 8–10)

- **System 7** (Chapter 8): Mixed Reality Robotic Platform for Autonomous Materials Synthesis (DMMRP)—MR-mediated human-robot interaction for high-throughput experimentation (Lee *et al.*, 2024b).

- **System 8** (Chapter 9): Multi-Robot Platform for Distributed pHRI—declarative configuration hierarchy supporting heterogeneous robot platforms across two institutions through configuration files alone.

- **System 9** (Chapter 10): Multi-Agent AI Platform for Enhanced pHRI—six specialized AI agent types with a four-level human-in-the-loop approval system (Chang *et al.*, 2025).

Part IV: Software Architecture for Multi-HRA Integration (Chapters 11–13). Throughout this dissertation, *Multi-HRA* refers to *Multi Human-Robot-AI interaction in eXtended Reality*: deployments that compose multiple human users, multiple robot platforms, and multiple AI agents within a shared mixed reality environment, rather than any single one of these dimensions in isolation.

- **System 10** (Chapter 11): Personal AI Infrastructure (own-code)—multi-provider LLM abstraction, microservice skills platform, real-time C++ core, and shared-memory vision transport.

- **System 11** (Chapter 12): Multi-Agent Architecture for Real-Time Robot Interaction Control—three-rate hierarchical architecture (agents at $\sim$1 Hz, diffusion-interaction policy at 10–15 Hz, Lie-group interaction control at 1 kHz) with Cholesky-parameterized SPD stiffness and damping and a unified C++ interaction control library.

- **Integration** (Chapter 13): Six demonstrations composing Systems 6–11 into bilateral teleoperation, multi-agent supervision over a dexterous manipulator, and multi-user distributed mixed reality.

Conclusion: Chapter 14 summarizes the systems and describes future work directions.

1.6 Publications and Collaborations

The work presented in this dissertation has contributed to the following publications:

- **Chang, D.**, Hunt, J., Atkins, J., Lee, H. "Validation of 4B-SPM Robot for Shoulder Rehabilitation," *IEEE ICRA*, 2021.

- Hwang, S.*, **Chang, D.***, et al. "Characterization of Human Shoulder Joint Stiffness Across 3D Arm Postures and Its Sex Differences," *IEEE TBME*, 2024. (*co-first authors)

- Hwang, S., **Chang, D.**, Saxena, A., Oleen, E., Paing, S. L., Atkins, J., Lee, H. "Characterization of Human Shoulder Joint Stiffness across Various Arm Postures and Its Sex Differences," *48th Annual Meeting of the American Society of Biomechanics (ASB)*, Madison, WI, Poster P1-281, 2024.

- Zahedi, F., **Chang, D.**, et al. "A Bayesian-Optimization-Based User-Centered Variable Damping Control," *IEEE RAL*, 2022.

- Atkins, J., **Chang, D.**, et al. "Dual-Purpose Gravity Compensation in a Wearable Upper-Limb Exoskeleton," *Wearable Technologies*, 2024.

- Save, O.*, **Chang, D.***, et al. "Feasibility of Gait Symmetric Learning Through Augmented Reality-based Visual Distortion," *IEEE TBME*, under review (2026). (*co-first authors)

- Lee, S., Kim, J. I., Baek, Y., **Chang, D.**, et al. "Fiber-Optic Force Sensing of Modular Robotic Skin for Remote and Autonomous Robot Control," *IEEE TRO*, 2024.

- **Chang, D.** "Multi-Agent Retrieval-Augmented Generation for Nuclear Waste Site Compliance Assessment," *Waste Management Symposia (WM2025)*, Phoenix, AZ, 2025.

- Kim, S., **Chang, D.**, Archambault, B., Park, Y. S. "Eyes, Ears, and Evidence: A Multi-Agent AI System for Nuclear Waste Decision Support," *Waste Management Symposia (WM2026)*, Phoenix, AZ, 2026.

- Kim, S., **Chang, D.**, Wang, J. "Persona Alchemy: Designing, Evaluating, and Implementing Psychologically-Grounded LLM Agents for Diverse Stakeholder Representation," *ICLR 2026 Workshop AFAA*, accepted (to appear, 2026).

- Park, S.-S., **Chang, D.**, Park, Y. S. "Non-spherical Wrist Robot Control," *Waste Management Symposia (WM2026)*, Phoenix, AZ, 2026.

- Park, S.-S., **Chang, D.**, Park, Y. S. "Heavy Manipulator Control," *Waste Management Symposia (WM2026)*, Phoenix, AZ, 2026.

Chapter 2

SYSTEM 1: SHOULDER REHABILITATION EXOSKELETON ROBOT

2.1 Introduction

Stable shoulder control governs distal joint and hand function during activities of daily living (Mussa-Ivaldi *et al.*, 1985), and shoulder impairment caused by stroke or other neurological diseases propagates position errors to the elbow and wrist (Perreault *et al.*, 2001). The property governing this stability is mechanical impedance—the relationship between imposed motion and resulting force—which can be modeled as a second-order system of stiffness, damping, and inertia (Hogan, 1985). Quantifying shoulder stiffness and damping for both healthy and impaired subjects supports rehabilitation training protocols that target damping-dependent spasticity and stiffness-dependent hypertonicity (Hwang *et al.*, 2024a).

2.2 Background and Related Work

2.2.1 Shoulder Impedance Characterization

Two gaps remain in the literature on shoulder impedance. Previous studies characterized shoulder joint stiffness primarily within two-dimensional (2D) horizontal planes using planar robotic systems that applied perturbations to the participant's hand and decomposed end-point stiffness into joint components via Jacobian matrices (Mussa-Ivaldi *et al.*, 1985; Perreault *et al.*, 2001). The 2D approach cannot isolate shoulder joint stiffness for a specific degree of freedom. Single-DoF servo motor setups that perturb the shoulder directly in three-dimensional (3D) space (Hwang *et al.*, 2024a) resolved the DoF-isolation issue, but each posture change required man-

ual reconfiguration, limiting the number of testable postures and making large-scale studies impractical. The second gap is demographic: although females experience higher incidence of shoulder injuries in occupational and sports settings (Hwang *et al.*, 2024a), sex differences in shoulder joint stiffness had not been investigated. Addressing both gaps requires a robotic platform capable of three-dimensional perturbation across multiple postures without manual reconfiguration; the 4B-SPM described in Section 2.3 provides this capability.

2.2.2 System Identification

Perturbation-based system identification estimates impulse response functions (IRFs)—the system's output when excited by a unit impulse—from perturbation-response data (Ljung, 1999). The IRF captures the system's dynamic behavior in a model-free form and is the standard tool for characterizing biomechanical impedance: the input is the perturbation torque applied to a joint, and the output is the resulting joint trajectory, from which inertia, damping, and stiffness terms can be regressed. Valid IRF estimation requires that the input–output relationship be sampled densely and consistently enough that the discrete-time identification preserves the underlying continuous-time dynamics.

2.3 Hardware Platform

The 4B-SPM (Hunt and Lee, 2019, 2018) (Figure 2.1) is a shoulder rehabilitation robot designed for precise impedance characterization. The mechanical design employs a low-inertia, high-precision parallel manipulator configuration with three active degrees of freedom and two passive degrees of freedom that accommodate the complex kinematics of the shoulder joint. Six Dynamixel MX-106R servo motors provide actuation with both position and torque control modes available.

The sensor suite integrates three modalities operating at different rates. A 6-axis force/torque sensor samples interaction forces at 488 Hz through an RS-232 serial interface. A Vicon motion capture system tracks marker positions at 260 Hz via UDP network protocol. Surface EMG sensors record muscle activity at 2,000 Hz for verification that subjects maintain passive relaxation during perturbation trials.

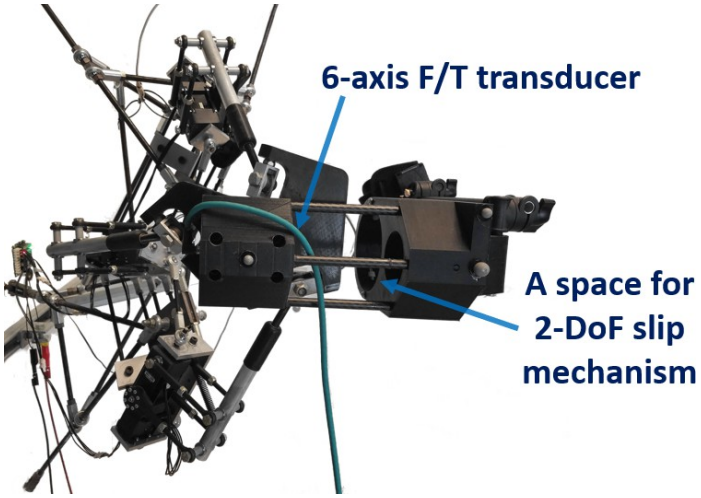


Figure 2.1: 4B-SPM shoulder exoskeleton robot with 6-axis F/T transducer and 2-DoF slip mechanism for passive compensation.

2.4 Software Requirements

The 4B-SPM mechanical platform of Section 2.3 supports high-bandwidth perturbation-response measurement, but realizing this capability for the 40-participant biomechanics campaign of (Hwang *et al.*, 2024a) required the software to satisfy four conditions that the original prototype did not:

1. Sustained 250 Hz control with jitter well below 10% of the period for valid IRF estimation. Cervin et al. demonstrated that sampling jitter introduces multiplicative noise in discrete-time system models proportionally to the jitter-to-period ratio $\varepsilon = \tilde{t}/t_s$ (Cervin *et al.*, 2003); when ε exceeds approximately 10%, the

identified model diverges from the true plant dynamics. The original prototype software—a monolithic single-threaded program in which sensor reading, control computation, and data logging executed sequentially—reached only 166 Hz with ± 50 ms timing jitter ($\varepsilon \approx 8.3$, far above the 10% threshold), corrupting the estimated IRFs and invalidating the biomechanical measurements. The 250 Hz target also matched the Vicon motion capture stream's 260 Hz native rate; the Dynamixel actuators supported higher rates, so the control rate was set by the slowest critical sensor rather than by the motor.

2. Simultaneous acquisition of force/torque, motion-capture, and EMG streams at their respective native rates (488 Hz F/T, 260 Hz Vicon, 2,000 Hz EMG), with no single sensor read blocking the 250 Hz control loop.

3. Reliable operation across a multi-month campaign of 40 participants completing approximately 30 trials each (over 1,200 trials in total), with protocol changes (perturbation type, posture set, analysis pipeline) supported without recompilation.

4. Parallel development across disjoint subsystems—C++ hardware interface, Python control logic, R statistical analysis, and ROS communication—without serialized commits across the four-researcher team.

2.5 Software Architecture

The architecture combines two patterns: a lock-free parallel design at the sensor-fusion layer (addressing Obj 1) and a four-language layer separation across hardware, control, analysis, and communication (addressing Obj 2). Table 2.1 summarizes.

Table 2.1: Architectural contributions of Chapter 2 to the dissertation's two objectives.

Objective	Method
Obj 1 (real-time parallel)	Lock-free sensor fusion, 250 Hz / ±0.5 ms jitter
Obj 2 (modular scalable)	Four-language layer separation (C++/Python/R/ROS)

2.5.1 Parallel Architecture

The challenge in achieving high-frequency control with multiple sensor modalities is that each sensor operates at a different rate and may block execution while waiting for data. This challenge is addressed through the lock-free parallel architecture introduced in Chapter 1: non-blocking multi-threaded design with lock-free data structures for inter-thread communication (Figure 2.2) (Chang *et al.*, 2021).

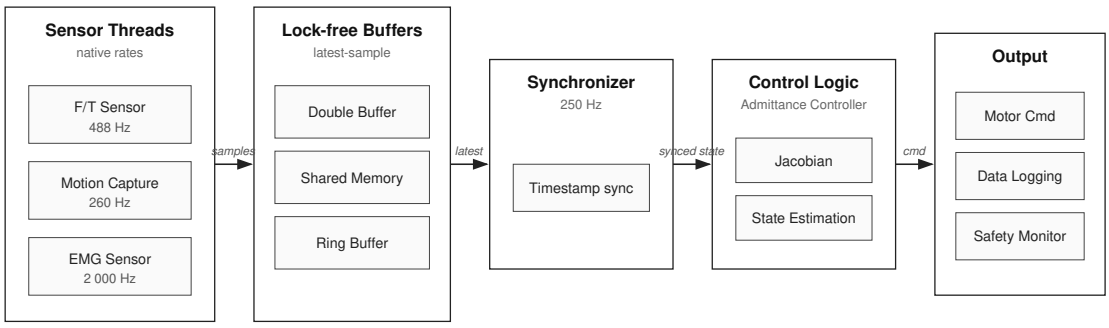


Figure 2.2: Multi-threaded architecture with parallel sensor threads, thread-safe buffers, synchronizer, and control loop. Each sensor operates in a dedicated thread that runs at its native rate (488 Hz F/T, 260 Hz Vicon, 2,000 Hz EMG); lock-free buffers decouple sensor acquisition from the 250 Hz control loop so that no single sensor read can block the controller.

Sensor Thread Design

Each sensor operates in a dedicated thread that runs independently of the control loop. The force/torque thread samples at 488 Hz with a 2.05 ms cycle time, acquiring

6-axis measurements through the serial interface. The motion capture thread receives position data at 260 Hz with a 4 ms cycle time via UDP packets. The EMG thread samples at 2,000 Hz with a 0.5 ms cycle time, oversampling relative to the control loop to capture rapid muscle activity transients.

Thread-safe buffers decouple the sensor threads from the control loop, allowing each to proceed at its natural rate without blocking. The force/torque data uses a double buffer—two alternating memory regions where one is written by the producer while the other is read by the consumer—with a 10 μs mutex lock (a mutual exclusion primitive that grants one thread exclusive access to a shared resource). The short lock duration is acceptable for this lower-priority data path. Motion capture data uses lock-free shared memory that eliminates mutex overhead through atomic compare-and-swap operations. EMG data flows through a ring buffer—a fixed-size circular queue that overwrites the oldest data when full—accumulating oversampled data between control cycles.

Synchronization and Control

A synchronizer component coordinates data from the multiple sensor threads at a base rate of 250 Hz. This rate exceeds the 200 Hz minimum requirement (Colgate and Brown, 1994) and leaves a 25% margin for timing variations. The synchronizer timestamps incoming data and interpolates or holds values as needed to provide the control loop with a consistent multi-modal state estimate.

The control loop operates with a 4 ms deadline for computing motor commands, logging data, and monitoring safety conditions. Timing enforcement (Figure 2.3) uses high-resolution clock measurements to ensure consistent 250 Hz operation, sleeping for the remaining cycle time after computation completes.

Lock-free circular buffers enable non-blocking communication between threads. The implementation uses atomic head and tail indices to allow concurrent push and pop operations without mutex locks, ensuring that sensor threads never block the control loop.

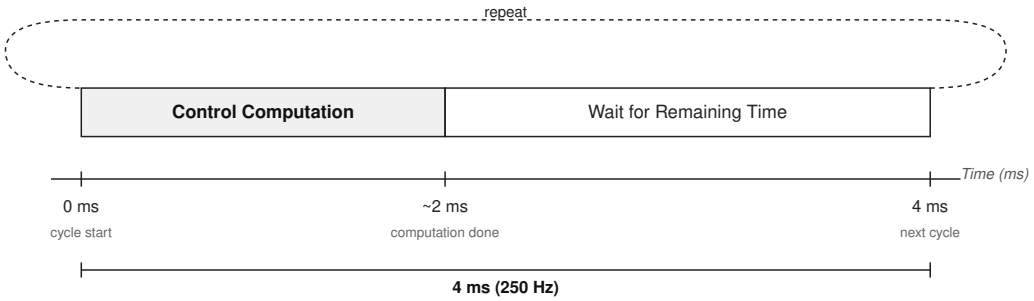


Figure 2.3: Timing enforcement mechanism showing cycle start measurement, control computation, elapsed time calculation, and sleep adjustment to maintain consistent 250 Hz control rate.

2.5.2 Modular Architecture

The architecture separates into four layers (Figure 2.4): hardware interface (C++), control logic (Python), statistical analysis (R), and ROS-based communication, each developed and tested independently.

Hardware Interface Layer

The hardware interface layer is implemented in C++ to provide deterministic execution timing for real-time sensor communication. Each sensor module (force/torque, motion capture, EMG) manages its respective serial or network interface at the rates described in Section 2.3. The motor control module interfaces with the Dynamixel servos and supports both position and torque control modes.

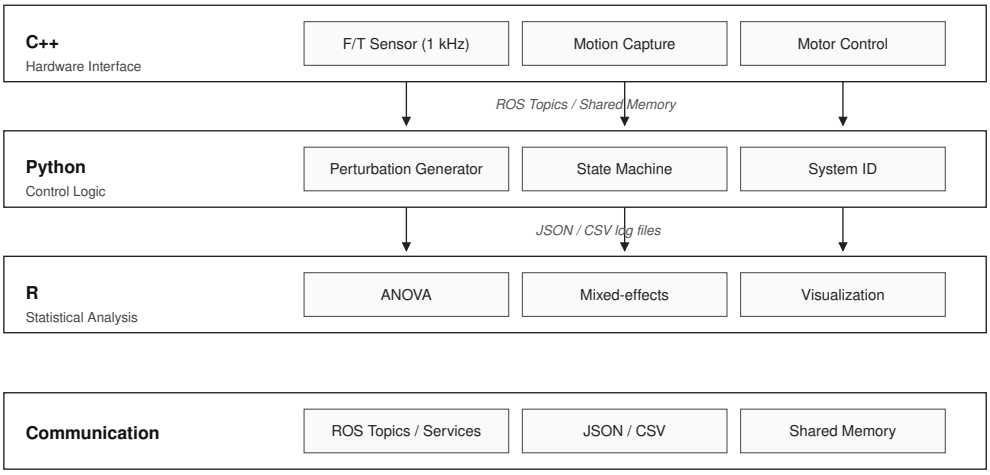


Figure 2.4: Modular language architecture with Hardware Interface (C++), Control Logic (Python), Statistical Analysis (R), and Communication layers. Each layer is implemented in the language best suited to its task and developed independently; the layers exchange data through ROS topics and CSV files, so changing one layer does not require recompiling the others.

Control Logic Layer

The control logic layer is implemented in Python, providing rapid prototyping capability for experiment designs. The perturbation generator creates excitation signals including Gaussian noise, chirp signals, and pseudo-random binary sequences. A state machine controller manages experiment sequencing, trial management, and error handling. System identification algorithms estimate impulse response functions and calculate stiffness parameters in real-time, allowing quality monitoring during data collection.

Statistical Analysis Layer

The statistical analysis layer uses R for ANOVA, mixed-effects models, and post-hoc comparisons. Statistical validation employs ANOVA analysis, mixed-effects models, and post-hoc comparisons appropriate for the study design. The visualization engine produces publication-ready plots with error bars and statistical significance markers.

Communication Layer

Inter-module communication uses two complementary mechanisms. ROS provides topic publishing, service calls, and a parameter server for real-time data exchange between layers. File-based communication through CSV files handles configuration management and result storage, supporting offline analysis of experiment data.

2.6 Results

The 40-participant biomechanics study reported in (Hwang *et al.*, 2024a) ran on the lock-free architecture: 40 healthy subjects (20 male and 20 female; age 24.5 ± 3.2 years) performed three-dimensional stochastic perturbations (2° RMS amplitude, 3 Hz cutoff) at 15 arm postures across 45-second trials with two repetitions per posture (over 1,200 trials in total). The architecture sustained 250 Hz control at ±0.5 ms jitter (jitter-to-period ratio $\varepsilon \approx 0.125$—within the stable identification regime of Cervin et al. (Cervin *et al.*, 2003); two orders of magnitude reduction from the prototype's 166 Hz / ±50 ms baseline), reached 99.8% data completeness, and extended system uptime from approximately 30 minutes per session to continuous multi-hour operation across the multi-month campaign. The Short Data Segment Algorithm estimated impulse response functions from the perturbation-response data (Figure 2.5) with mean 96.5% variance accounted for (VAF), confirming reliable identification.

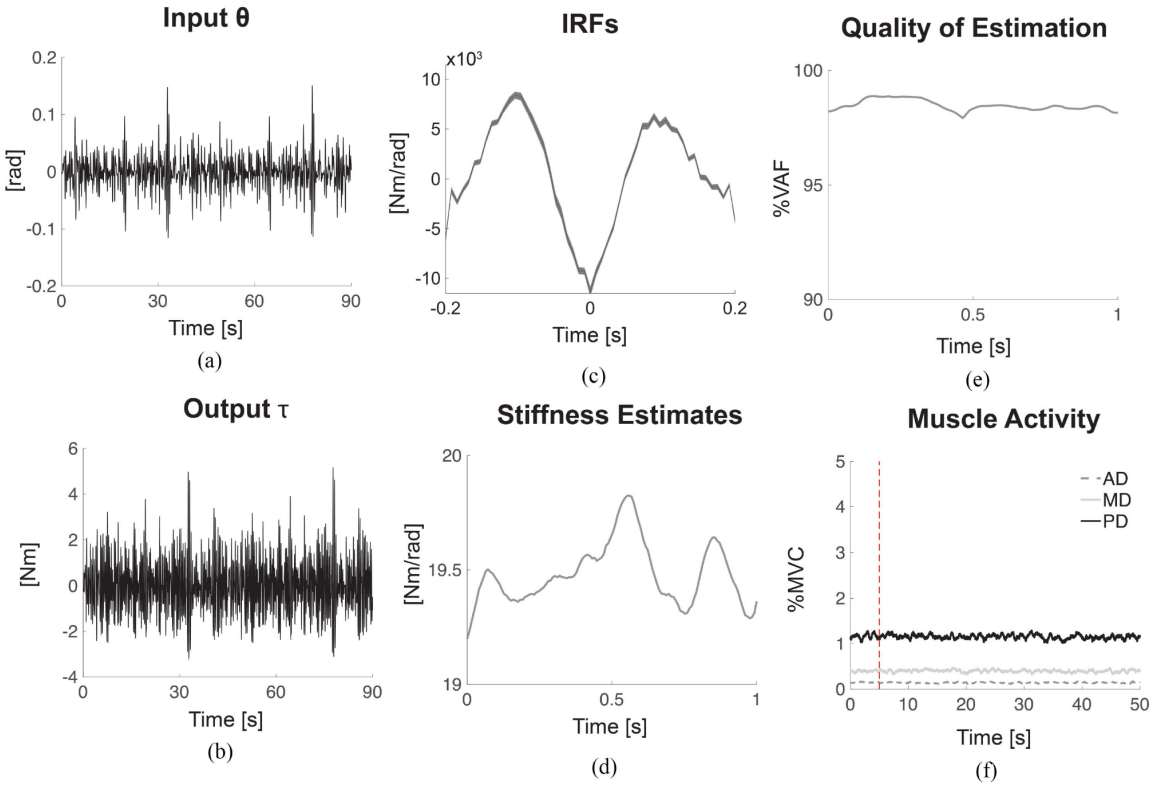
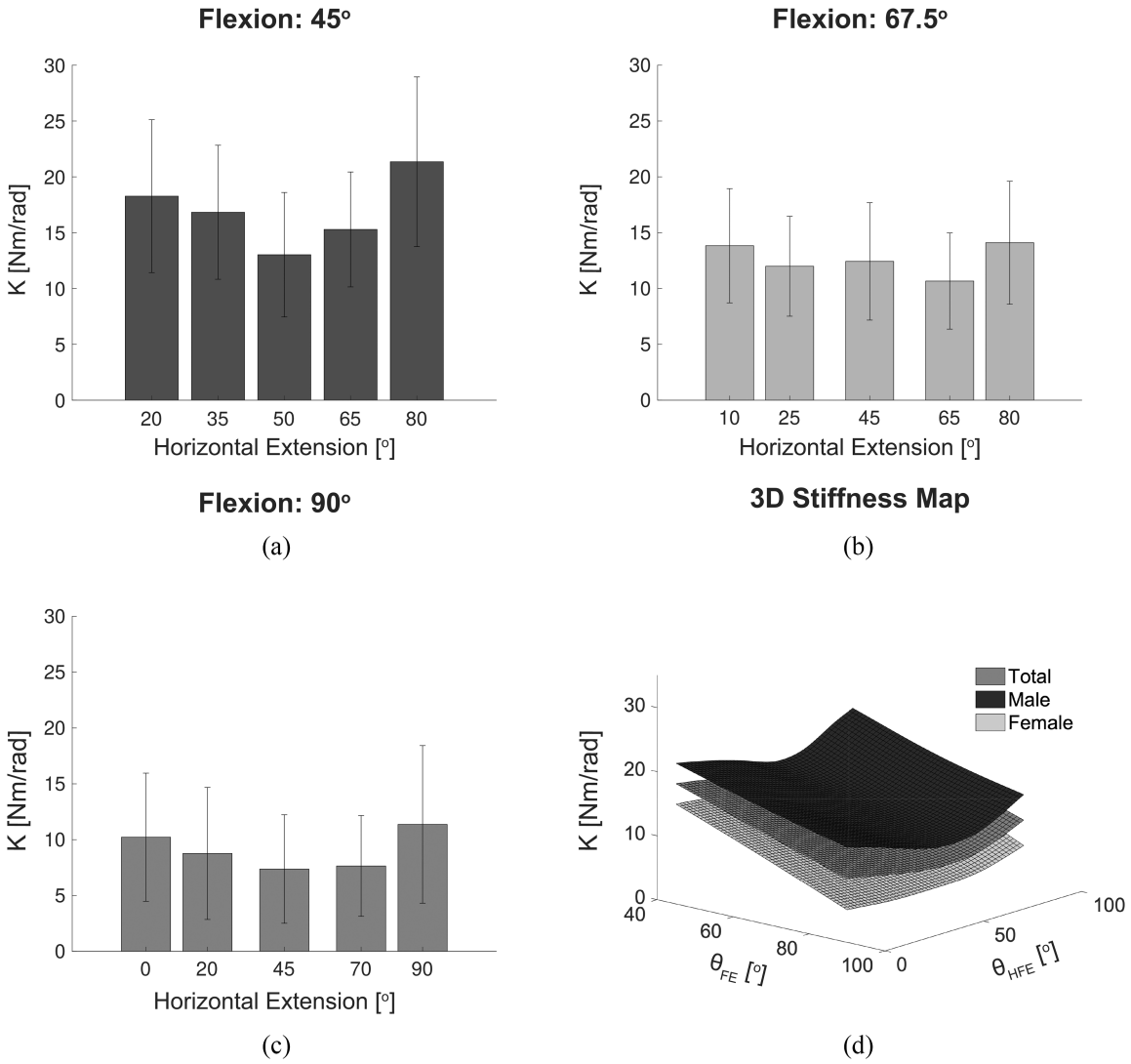


Figure 2.5: Representative results showing (a) input perturbation, (b) output torque response, (c) identified impulse response functions, (d) stiffness estimates across postures, (e) estimation quality (%VAF), and (f) EMG verification of passive muscle state.



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

Figure 2.6: Shoulder stiffness characterization across arm postures: (a-c) Stiffness variation with horizontal extension at three shoulder flexion angles (45°, 67.5°, 90°), showing increased stiffness at extreme horizontal positions. (d) Three-dimensional stiffness map revealing sex-dependent differences, with male participants exhibiting higher stiffness values than female participants across all postures.

The biomechanical findings revealed that arm posture affects shoulder joint stiffness (Figure 2.6): stiffness decreased as shoulder flexion angle increased, and stiffness increased as shoulder horizontal flexion or extension approached the limits of the range of motion. The study provided a three-dimensional characterization of shoulder stiffness across postures and found evidence for sex-dependent shoulder stiffness, with male participants exhibiting higher stiffness values than female participants across all postures.

The four-language layer separation supported parallel development by four researchers (C++ hardware interface, Python control logic, R statistical analysis, and ROS communication) without merge conflicts; switching from Gaussian noise to pseudo-random binary sequence perturbation required modification of the Python control layer only, leaving the C++ hardware interface and R analysis layers untouched.

2.7 Discussion

The lock-free design was selected over mutex-based synchronization because the priority-inversion behavior of POSIX mutex on a non-real-time Linux kernel is unbounded in the worst case (incompatible with the ± 0.5 ms jitter requirement) and because the producer–consumer pattern at 250 Hz with single-writer, single-reader ring buffers permits a wait-free implementation with atomic index updates—making the architectural cost of avoiding the kernel scheduler one cache line of contention rather than a redesign. The same reasoning extends to the 500 Hz case of Chapter 3 and to the zero-copy vision transport of Chapter 11.

ROS-based modularization absorbed the four-researcher parallel-development workflow without merge conflicts during the multi-month TBME 2024 campaign. The custom thread-based communication, however, was tightly coupled to this hardware configuration; porting to a different robot required re-implementing the buffer inter-

faces. The 250 Hz rate left no margin for computationally heavier algorithms such as real-time adaptive control or online optimization. Chapter 3 introduces ROS topic-based communication for hardware-agnostic modularity, and Chapter 4 introduces Bayesian optimization for online parameter tuning over a transport that sustains concurrent inference.

2.8 Summary

This chapter contributed two architectural patterns. The lock-free parallel architecture (Obj 1) replaced the prototype's 166 Hz / ±50 ms timing with 250 Hz control at ±0.5 ms jitter, published as the dissertation author's first-author contribution at IEEE ICRA 2021 (Chang *et al.*, 2021). The four-language modular layer separation (Obj 2) absorbed the four-researcher parallel-development workflow and supported the 40-participant biomechanics study reported as co-first-author publication in IEEE TBME 2024 (Hwang *et al.*, 2024a), with companion conference dissemination at the Annual Meeting of the American Society of Biomechanics (Hwang *et al.*, 2024b). The lock-free pattern carries forward to the 500 Hz wearable exoskeleton of Chapter 3; the modular layer pattern carries forward to the asynchronous Bayesian-optimization scheduler of Chapter 4.

Chapter 3

SYSTEM 2: WEARABLE UPPER-LIMB EXOSKELETON ROBOT

3.1 Introduction

Physically strenuous work environments lead to a greater risk of developing musculoskeletal disorders (MSDs) in workers, with risk factors including highly repetitive manufacturing tasks, heavy lifting, and awkward postures; symptoms are exacerbated by heavy sustained loading to the shoulder and neck joints (Atkins *et al.*, 2024). Reducing forces on the worker's joints reduces fatigue and slows or stops the progression of these injuries, so robotic exoskeletons that assist a wearer's motion are a viable option for reducing pain and injury in fields such as manufacturing and construction (Atkins *et al.*, 2024).

3.2 Background and Related Work

3.2.1 Upper-limb Exoskeleton Designs

Upper-limb exoskeletons can be broadly classified into active, passive, and hybrid designs. Active exoskeletons provide torque assistance through multiple actuators at the cost of weight, computational load, and power consumption. Passive exoskeletons rely on spring-driven mechanisms to provide partial gravity compensation with limited adaptability. Hybrid (semi-active) designs combine actively controlled actuators with passive components, offering reduced weight while providing more assistance than purely passive systems (Atkins *et al.*, 2024). Both active and hybrid exoskeletons require a compatible assistive controller that must minimize user effort through gravity compensation and feedforward torque compensation while maintaining transparent

operation—the robot must follow human movement without imposing perceptible resistance during free motion (Just *et al.*, 2018).

3.2.2 Stability–Transparency Trade-off and Z-Width

Achieving this transparent operation is constrained by a trade-off between stability and transparency (Hogan, 1985). Over-damped controllers ensure safe operation but cause users to feel resistance; under-damped controllers improve transparency but risk instability when the robot interacts with human limb dynamics. The passivity condition for sampled-data haptic systems (Colgate and Schenkel, 1997) requires that the product of sampling period, virtual stiffness, and damping remain below a threshold set by the physical damping. A controller running at 100 Hz can render approximately one-quarter the maximum virtual stiffness of a controller running at 200 Hz before becoming active (unstable). Delays and jitter further reduce this Z-width—the range of stable virtual impedances (Colgate and Brown, 1994)—so high control frequencies are required for transparent impedance rendering.

3.3 Hardware Platform

The wearable shoulder exoskeleton of (Atkins *et al.*, 2024) (Figure 3.1) is a 9.83 kg hybrid active–passive 4-DoF system. Two active joints (q_0 abduction/adduction, q_3 flexion/extension) are actuated by Maxon EC 60 Flat 200W motors with 9:1 gear reduction (4.8 Nm at 360 rpm, backdrivable for transparent impedance) and driven by two ODrive 3.6 controllers on independent serial buses; two passive joints (q_1 translational for user fitting, q_2 rotational for misalignment compensation) accommodate natural shoulder movement, while a four-bar mechanism redistributes weight to the torso (11 cm center-of-mass lowering), dual-purpose springs compensate the device weight ($\pm$0.5 Nm) and partial arm weight, and a 6-DoF Spatial Compliant Alignment

Mechanism (SCAM) handles ±15 mm shoulder translation. Sensing comprises an ATI Axia-80 6-axis force/torque sensor at the end effector and four AS5047P 4096-step absolute encoders for joint positions; complete kinematic parameters are provided in Appendix A.

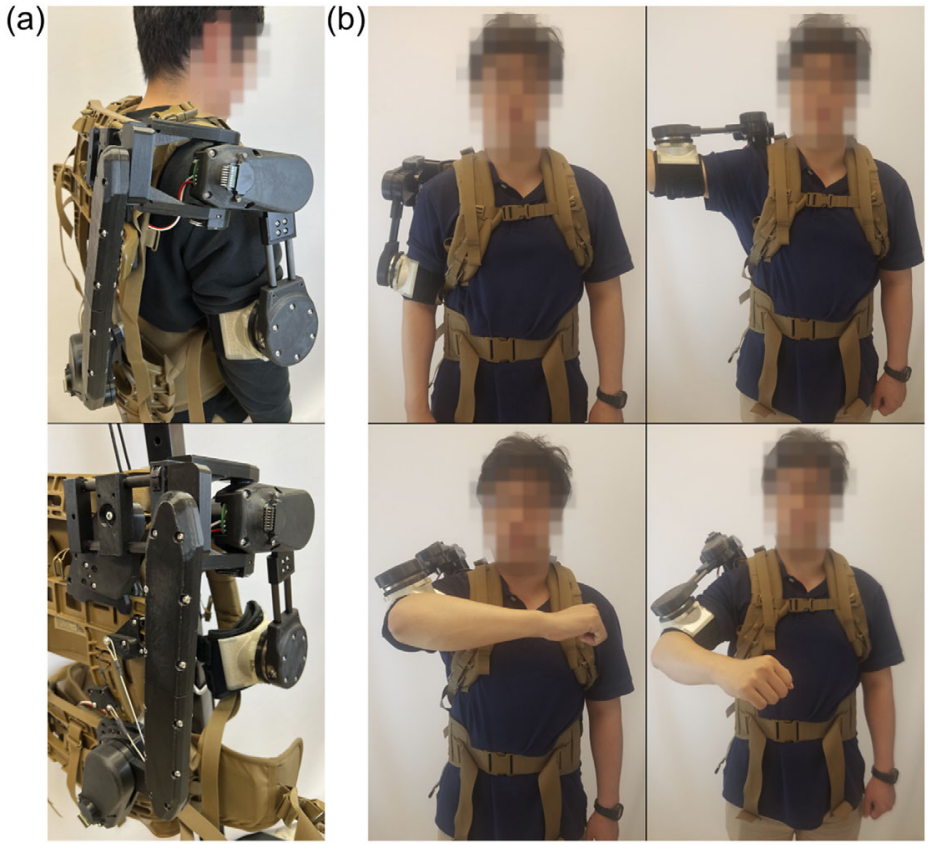


Figure 3.1: Wearable shoulder exoskeleton with four-bar mechanism for weight redistribution, dual-purpose gravity compensation, and SCAM for misalignment compensation.

3.4 Software Requirements

The hardware platform of Section 3.3—two torque-controlled actuators on independent buses, a 6-axis force/torque sensor, and a hybrid active–passive 4-DoF kinematic chain—imposed three software conditions that the System 1 architecture (Chapter 2) could not meet directly:

1. Sustained 500 Hz control—twice the System 1 target of 250 Hz—so that the Z-width of the sampled-data impedance loop was wide enough to render low virtual stiffnesses without losing passivity (Colgate and Schenkel, 1997; Colgate and Brown, 1994).

2. Non-blocking acquisition of both joint torques and the 6-axis force/torque reading inside each 2 ms cycle for dual motor controller integration, without any single read blocking the loop.

3. Switching between three distinct experimental protocols (dynamic trajectory tracking, stiffness validation, transparency with human wearer) without recompilation; the existing single-layer prototype required a recompile, incompatible with the cross-session study design.

3.5 Software Architecture

The architecture combines two patterns: a non-blocking lock-free design at the sensor-fusion layer (addressing Obj 1) and a three-layer separation across application, interface, and hardware concerns (addressing Obj 2). Table 3.1 summarizes.

Table 3.1: Architectural contributions of Chapter 3 to the dissertation's two objectives.

Objective	**Method**
Obj 1 (real-time parallel)	Lock-free non-blocking acquisition, 500 Hz / <2 ms jitter
Obj 2 (modular scalable)	Three-layer separation (Application/Interface/Hardware)

3.5.1 Parallel Architecture

The target control frequency of 500 Hz imposes strict timing constraints that preclude the use of blocking operations in the control path (Colgate and Brown, 1994).

Non-blocking Architecture for Real-Time Control

The lock-free parallel architecture from System 1 (Chapter 2) was extended with four design principles (Figure 3.2). First, a latest-data priority policy ensures that the controller always operates on the most recent sensor measurements rather than stale data that may have accumulated in communication buffers. Second, temporal decoupling through lock-free buffers allows sensor acquisition and control computation to proceed asynchronously without blocking each other. Third, real-time communication optimizations minimize network latency between distributed software components. Fourth, synchronized control ensures coordinated actuation across multiple motor controllers.

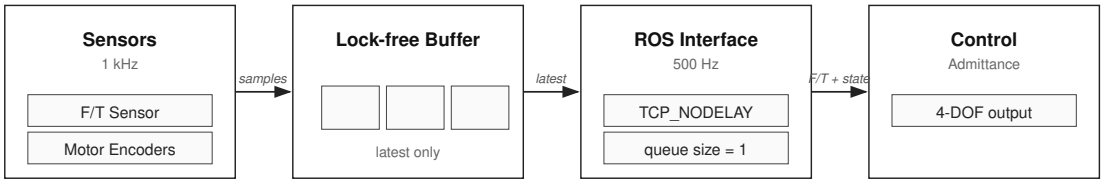


Figure 3.2: Non-blocking architecture with hardware layer operating at 1 kHz, lock-free buffer layer, network layer at 500 Hz, and control layer. Sensors sample at 1 kHz into latest-data ring buffers and the 500 Hz controller reads only the most recent sample; this temporal decoupling prevents any single sensor from blocking the control thread.

Layered Implementation

The hardware layer operates at 1 kHz, with the force/torque sensor, dual ODrive controllers, and Arduino-based encoder interface all sampling synchronously at this rate. Sensor data flows into lock-free ring buffers that implement a latest-data-only policy, automatically discarding stale samples when the control loop cannot consume them fast enough.

The network layer implements ROS communication with optimizations for real-time performance. The `TCP_NODELAY` socket option disables Nagle's algorithm (a TCP optimization that buffers small packets to reduce overhead at the cost of added latency), so each control message is transmitted immediately. Publisher queue sizes are set to 1 to retain only the most recent data.

The control layer implements a workspace admittance controller with the dynamics:

$$M_d\ddot{x} + B_d\dot{x} + K_d x = F_{ext} \tag{3.1}$$

where M_d, B_d, and K_d are the desired inertia, damping, and stiffness matrices, and F_{ext} is the measured external force. The controller maintains an augmented state vector to coordinate multi-joint actuation.

The exoskeleton uses two ODrive motor controllers, each managing two joints. ROS namespace isolation (Quigley *et al.*, 2009) provides unified 4-DoF control while maintaining clear separation between the controller interfaces. Board 1 controls joints J0 and J1 through the `/board1` namespace via serial port `/dev/ttyACM0`, while Board 2 controls joints J2 and J3 through the `/board2` namespace via `/dev/ttyACM1`.

3.5.2 Modular Architecture

The software separates into three layers—Application (500 Hz), Interface (500–750 Hz), and Hardware (>1 kHz)—each deployable and testable independently (Figure 3.3).

The Application Layer operates at 500 Hz and handles high-level control logic and experiment management. This layer implements the admittance controller with variable impedance capabilities and gravity compensation. A protocol manager provides JSON-based configuration with hot-reload capability, allowing experiment parameters to be modified without restarting the system.

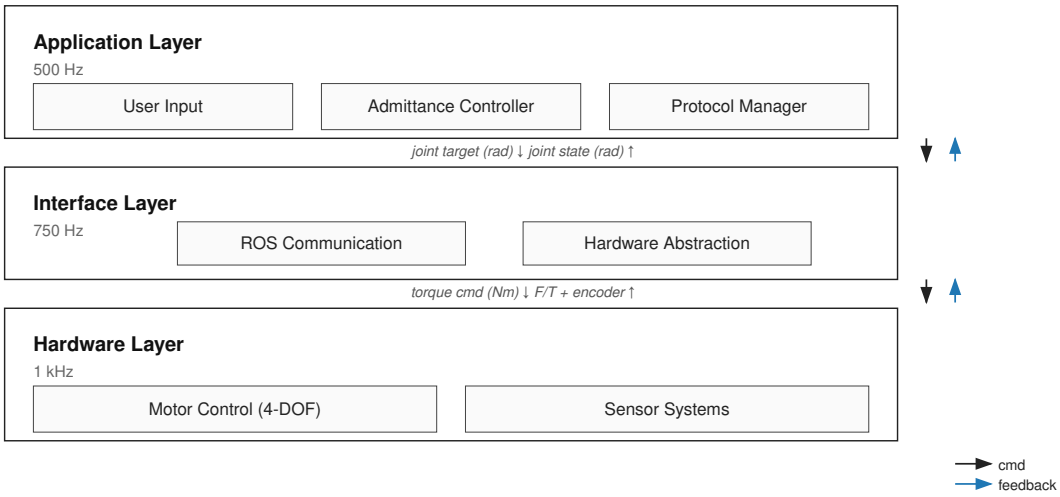


Figure 3.3: Three-layer modular architecture showing the Application Layer (500 Hz), Interface Layer (500–750 Hz), and Hardware Layer (>1 kHz). Each layer is deployable and testable independently; replacing the F/T sensor or switching experimental protocol affects only one layer, leaving the others untouched.

The Interface Layer operates at 500–750 Hz depending on the specific sensor being handled. This layer provides hardware abstraction through ROS topics (`/joint_states`, `/joint_commands`) and services (`/configure`), so replacing the ATI Axia-80 with a different F/T sensor requires changes only in the Hardware Layer driver, not in the controller.

The Hardware Layer runs at rates exceeding 1 kHz, interfacing with the ODrive motor controllers, ATI Axia-80 force/torque sensor, and AS5047P encoders. All low-level processes operate faster than the control loop to ensure deterministic hardware behavior.

3.6 Results

3.6.1 Dynamic Trajectory Tracking

The first study evaluated the system's ability to reproduce desired admittance controller dynamics. Three impedance levels were tested: low ($I = 0.1$ kgm^2, $B = 1.0$

Nms/rad), medium ($I = 0.15$ kgm^2, $B = 1.5$ Nms/rad), and high ($I = 0.2$ kgm^2, $B = 2.0$ Nms/rad). A 2.0 Nm sinusoidal torque input at 0.5 Hz was applied, and the resulting angular trajectories were compared to the ideal dynamical system response (Figure 3.4).

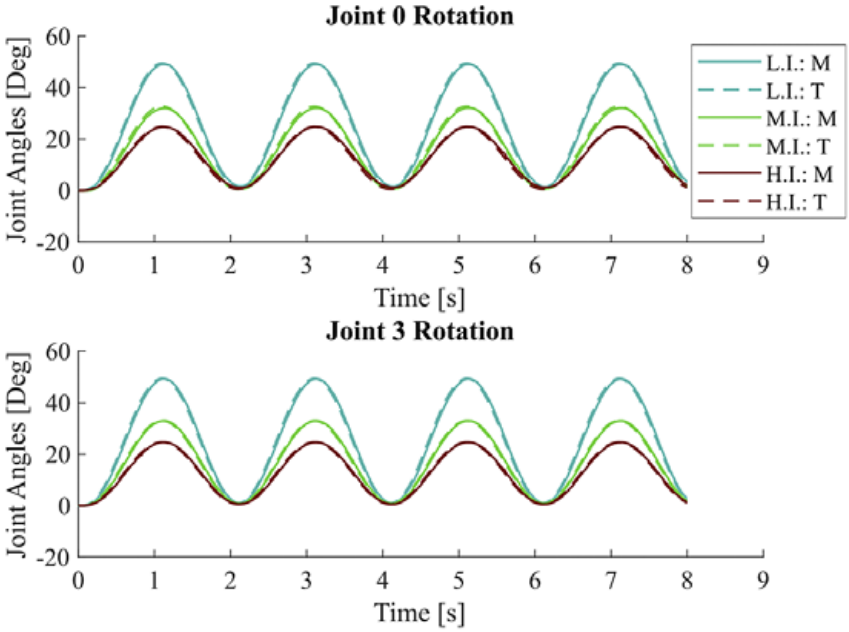


Figure 3.4: Dynamic trajectory tracking across three impedance levels showing measured (M) and target (T) trajectories for Joint 0 and Joint 3 rotation.

Peak-to-peak angular error was less than 0.33°, peak-to-peak time delay was 36.4 ms, and RMSE was less than 1% of the commanded range of motion for all three impedance levels. These errors confirm that the non-blocking architecture introduces no systematic tracking degradation relative to the ideal admittance model.

3.6.2 Stiffness Behavior Validation

The second study validated the system's ability to replicate specific stiffness behaviors. The exoskeleton was manually displaced from an equilibrium position while the admittance controller simulated stiffness values of 5, 10, and 20 Nm/rad. The

measured restoring torques were compared to the predicted torques from an ideal spring system (Figure 3.5).

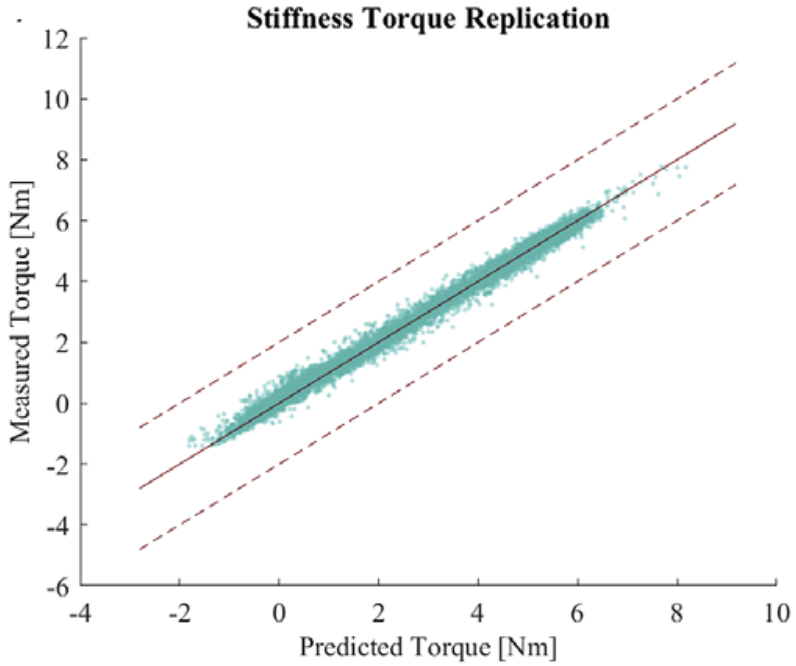


Figure 3.5: Stiffness torque replication showing measured torque versus predicted torque with greater than 99.2% variance accounted for (VAF).

Linear regression showed that estimated stiffness coefficients were within 0.1 Nm/rad of commanded values and VAF exceeded 99.2% for all tested cases, confirming that the admittance controller reproduces the intended mechanical behavior to within measurement noise—a prerequisite for safe pHRI in occupational settings.

3.6.3 Transparency Study with Human Wearer

The third study evaluated the transparency of the admittance controller during human-robot interaction. A subject wore the exoskeleton while the controller simulated a low-impedance environment ($I = 0.1$ kgm^2, $B = 0.5$ Nms/rad, $K = 0.0$ Nm/rad).

Four motion cases were tested: side raise, front raise, neutral raise, and neutral reach. Each motion was performed for 10 trials timed at 3 seconds per motion.

The results showed nominal interaction torques of 0.26 Nm or less for all motion cases, with peak torques reaching 0.81 Nm for the neutral raise case. For context, the human shoulder generates approximately 20–50 Nm during daily tasks (Atkins *et al.*, 2024); interaction torques of 0.26 Nm represent less than 1.3% of this range, well below the perceptual threshold for resistive torque during free motion.

3.7 Discussion

The three-layer architecture achieved 500 Hz control with <2 ms jitter on the same lock-free buffer substrate established for the 250 Hz case of Chapter 2; the 500 Hz operation required only its application at a finer time scale, confirming that the substrate scales across systems with different timing budgets.

The three validation studies confirmed three architectural properties: dynamic trajectory tracking within 0.33° error and 36.4 ms delay confirmed accurate admittance dynamics, stiffness replication exceeding 99.2% VAF confirmed model-consistent behavior, and transparency torques of 0.26 Nm or less during free motion validated that the architecture does not impede natural human movement.

Switching from the stiffness-validation protocol to the transparency-study protocol required changing one Python script in the Application Layer without modifying the Interface or Hardware Layers, demonstrating the principle of functional separation between layers in practice.

Impedance parameters (M_d, B_d, K_d) were manually tuned per subject; the per-user optimum varies because human-limb impedance differs across individuals (Zahedi *et al.*, 2022), and ROS serialization overhead would exceed the control period for an integrated optimizer loop. Chapter 4 introduces Bayesian optimization for user-adaptive

parameter tuning over ZeroMQ asynchronous sockets that sustain the concurrent inference pipeline.

3.8 Summary

This chapter contributed two architectural patterns. The non-blocking lock-free architecture (Obj 1) sustained 500 Hz transparent impedance control with <2 ms jitter on the same substrate established for the 250 Hz case of Chapter 2. The three-layer separation (Obj 2) confined protocol changes to a single Application Layer script across three distinct validation studies (dynamic trajectory tracking, stiffness replication, transparency with human wearer). The architecture and validation were reported in (Atkins *et al.*, 2024) as the dissertation author's second-author contribution to Wearable Technologies 2024. The non-blocking lock-free architecture and three-layer separation patterns carry forward to the distributed task scheduler of Chapter 4, where ZeroMQ asynchronous sockets and a three-module decomposition extend these architectural concerns to user-adaptive Bayesian optimization.

Chapter 4

SYSTEM 3: ROBOTIC ARM MANIPULATOR WITH DISTRIBUTED TASK SCHEDULER FOR BAYESIAN OPTIMIZATION

4.1 Introduction

User-adaptive variable damping control improves the coupled stability–agility trade-off in pHRI by modulating the damping coefficient between negative and positive values based on the user's intent of motion (Zahedi *et al.*, 2022). Realizing this controller in a multi-subject study required a software architecture that ran a per-user Bayesian optimization loop in parallel with the real-time impedance controller and persisted the optimizer state across the natural boundaries of an experimental session. This chapter describes the distributed task scheduler that supports this integration.

4.2 Background and Related Work

4.2.1 Enhanced pHRI: Stability–Agility–Effort

Impedance and admittance controllers are the standard mechanisms for safe pHRI (Hogan, 1985; Colgate and Schenkel, 1997): positive damping dissipates energy and ensures stability of the coupled human–robot system but at the cost of reduced user agility and additional effort to overcome the robot's resistive behavior (Zahedi *et al.*, 2022). The transparent operation regime of Chapter 3—in which the robot follows the user without imposing perceptible resistance—additionally requires that the controller render low virtual stiffness without crossing the passivity boundary (Colgate and Brown, 1994). A fixed-damping controller cannot simultaneously satisfy both stability (high damping) and agility/effort (low damping) goals across the full task space;

enhancing pHRI beyond bare safety requires resolving this stability–agility–effort trade-off through damping that varies in time.

4.2.2 Variable Impedance Control

Variable impedance control extends the classical impedance formulation by allowing inertia, damping, and stiffness parameters to vary in time according to task state, user signals, or model-based estimators (Tsumugiwa *et al.*, 2002; Zahedi *et al.*, 2022). The category includes variable stiffness, variable damping, and variable inertia variants, each used to mitigate a different facet of the stability–agility–effort trade-off in pHRI. The variable damping formulation exercised in this chapter, contributed by Zahedi et al. (Zahedi *et al.*, 2022), modulates the damping coefficient between negative and positive values according to the user's intent of motion: positive damping dissipates energy when the user moves erratically, while negative damping injects energy when the user moves intentionally toward the target, recovering agility without compromising stability. Because human-limb impedance and movement strategies vary across individuals (Zahedi *et al.*, 2022), the damping function parameters that are optimal for one user are rarely optimal for another, motivating per-user adaptation of the controller.

4.2.3 Bayesian Optimization for Per-User Tuning

Per-user controller tuning is constrained by data cost: each evaluation of the energy-based objective requires a human subject performing several reaching trials, so the optimization budget is small (typically 10–20 acquisition iterations per subject). Bayesian optimization is well suited to this regime because it builds a Gaussian-process surrogate of the objective and selects each next evaluation through an acquisition rule (e.g., expected improvement) that trades off exploration and exploitation, converging

on far fewer evaluations than grid search or random search would require—a property formalized in the BO literature for expensive black-box functions (Brochu *et al.*, 2010; Shahriari *et al.*, 2016). The Zahedi et al. study applied this approach to per-user damping function tuning with a Gaussian-process surrogate and an expected-improvement acquisition rule (Zahedi *et al.*, 2022).

This scheme works directly in a single-subject session that runs uninterrupted, but a typical pHRI session of an hour or more—fragmented by rest periods, operator intervention, and occasional process restarts—requires the Gaussian-process surrogate state to persist across these natural session boundaries if the optimization is to resume rather than restart. Earlier single-process prototypes held this state only in process memory; a break that closed the optimizer process erased the surrogate and forced the optimization to begin again, multiplying the data cost in the most failure-prone setting (multi-subject studies). Warm-starting Bayesian optimization across sessions—loading the most recent serialized surrogate state and resuming acquisition—treats the natural session boundary as an ordinary checkpoint rather than a session terminator; Section 4.5.2 describes the on-disk state-manager design that realizes this property and made the multi-subject transparency study of (Zahedi *et al.*, 2022) tractable in its published form.

4.3 Hardware Platform

Figure 4.1 shows the KUKA LBR iiwa R820 platform on which the architecture was exercised.

The architecture was exercised on the experimental platform of (Zahedi *et al.*, 2022): a 7-DoF KUKA LBR iiwa R820 torque-controlled manipulator with a 6-axis ATI Delta IP60 force-torque load cell at the end effector, kinematic and force signals acquired at 1 kHz and low-pass filtered by a fourth-order Butterworth filter (20 Hz cutoff). Five

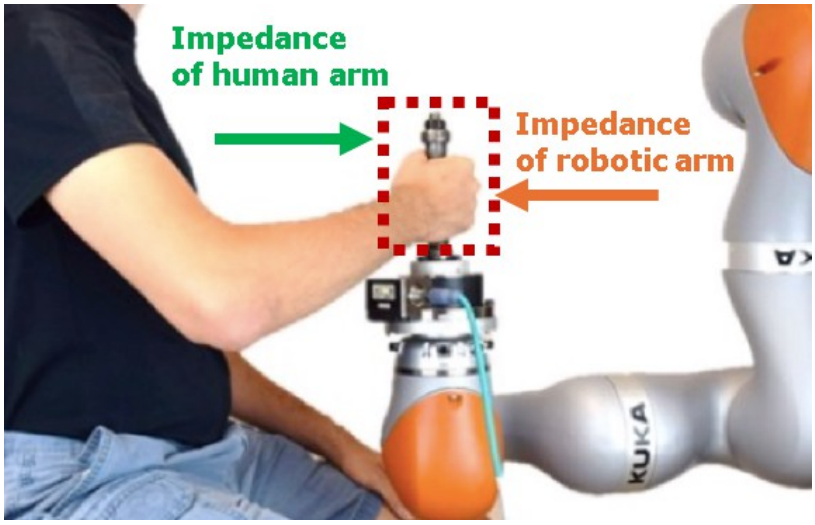


Figure 4.1: Experimental platform used in (Zahedi *et al.*, 2022): a 7-DoF KUKA LBR iiwa R820 torque-controlled manipulator with a 6-axis ATI Delta IP60 force-torque load cell at the end effector, operated through a two-dimensional reaching task.

healthy subjects (four male, one female; age 21–34 years, height 163–183 cm, weight 50–78 kg) performed 20×20 cm^2 two-dimensional reaching tasks in blocks of twelve trials with rest periods between blocks; each Bayesian optimization iteration used four consecutive trials to evaluate an energy-based objective, with convergence between 12 and 16 iterations per subject and an approximately 45% reduction in user energy expenditure (and 20% improvement in aggregate performance) over a fixed-damping baseline (the study was approved by the Arizona State University IRB under protocol STUDY 00010123, with all subjects providing written informed consent). For the software architecture, the salient features of the protocol are that (i) a single subject session spans tens of minutes to more than an hour with multiple rest breaks during which the optimizer must retain state, (ii) the acquisition function evaluation is not bounded in time and cannot be embedded in the 1 kHz control loop, and (iii) algorithm variants have to be compared between and within sessions without disturbing the live subject.

4.4 Software Requirements

The protocol of Section 4.3 imposes two software requirements that the single-threaded pHRI prototype used before this work could not satisfy:

1. The optimizer state—the Gaussian-process surrogate, the acquisition function history, and the running record of evaluated parameter points—must persist across the natural boundaries of an experimental session, including rest breaks, operator intervention, and process restarts that were routine in the earlier prototype.

2. The optimizer must execute in parallel with the real-time impedance controller. Fitting a Gaussian-process surrogate after each acquisition step takes tens to hundreds of milliseconds and does not have a hard time bound, whereas the impedance controller runs at a fixed 1 kHz on the KUKA LBR iiwa R820 platform used in the study and cannot tolerate blocking calls on the control thread.

The prototype ran both the optimizer and the controller in a single Python process: optimizer state was held only in memory, the control loop stalled whenever the acquisition step was evaluated, and algorithm changes required editing and restarting the same process that held the live experimental session. The distributed task scheduler described in this chapter was designed to remove these coupling points.

4.5 Software Architecture

The architecture combines two patterns: a ZeroMQ ROUTER–DEALER scheduler that decouples the Bayesian optimizer from the 1 kHz controller (addressing Obj 1) and a three-module decomposition (optimizer, controller, monitor) with language-agnostic JSON message interfaces (addressing Obj 2). Table 4.1 summarizes.

Table 4.1: Architectural contributions of Chapter 4 to the dissertation's two objectives.

Objective	Method
Obj 1 (real-time parallel)	ZeroMQ ROUTER–DEALER scheduler decoupling Bayesian optimizer from 1 kHz controller
Obj 2 (modular scalable)	Three-module decomposition (optimizer / controller / monitor) with JSON message interfaces

The two requirements identified in Section 4.4—state persistence across natural session boundaries and computational decoupling between the optimizer and the real-time controller—rule out the three architectures that the group had previously considered. A single-process architecture holds the optimizer state only in process memory and blocks the control loop during surrogate fitting. A blocking client-server architecture satisfies the persistence requirement but still stalls the controller on each acquisition call. A central-broker publish-subscribe architecture such as a ROS master introduces a third process whose failure can halt the experiment, together with a network round trip on every message that the control loop does not require. The architecture adopted here is the smallest design that satisfies both requirements: a lightweight, brokerless, message-driven distributed system with explicit on-disk state persistence.

4.5.1 Parallel Architecture

ROUTER–DEALER Pattern

The task scheduler is built on ZeroMQ (Hintjens, 2013)—a high-performance brokerless messaging library providing asynchronous socket abstractions (PUB/SUB, ROUTER/DEALER, REQ/REP) over multiple transports (TCP, IPC, in-process)—using the ROUTER–DEALER socket pattern (Figure 4.2). The scheduler exposes a single ROUTER socket to which an optimizer worker and a controller worker connect

through DEALER sockets. Messages carry an opaque identity prefix that ZeroMQ manages automatically, so neither side maintains a session table. Both directions are fully asynchronous: senders do not block on receivers, and the underlying transport handles queueing.

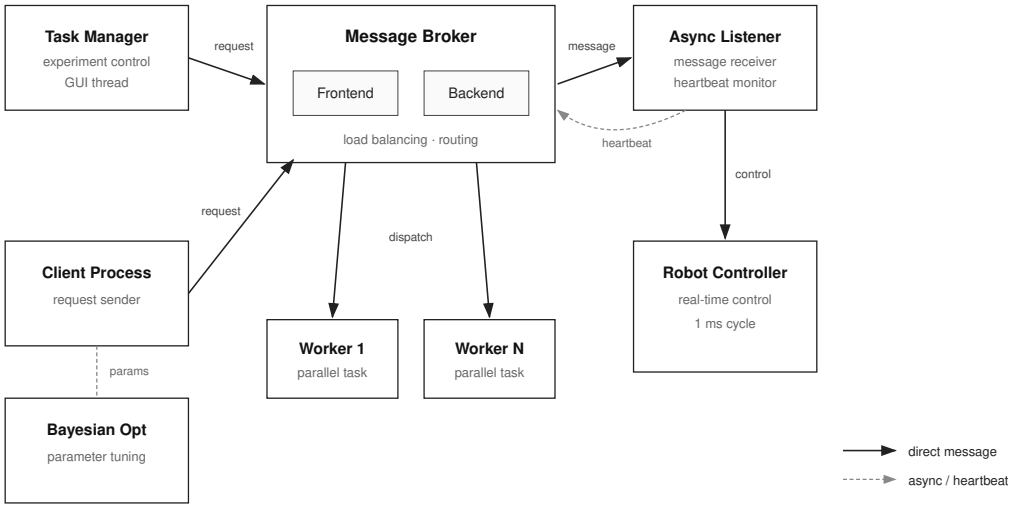


Figure 4.2: ZeroMQ ROUTER–DEALER task scheduler. The Bayesian optimizer and the real-time controller run as separate processes connected to the scheduler through DEALER sockets. The scheduler routes parameter updates from the optimizer to the controller and trial outcomes from the controller back to the optimizer asynchronously.

Four properties of ZeroMQ were relevant to this application. Its message-driven asynchronous ROUTER–DEALER pattern is provided out of the box, so the architecture could use a pattern tested in production rather than a custom socket implementation. Its bindings for Python, MATLAB, and C++ cover the set of languages used by the optimizer (Python with a Gaussian-process surrogate built on GPy (GPy Authors, 2014)), the controller (C++ for the 1 kHz impedance loop on the KUKA), and the experiment monitor (initially MATLAB, later Python). The direct point-to-point connections between processes leave no central broker whose failure would halt the experiment, and the approximately one-megabyte runtime with no

external dependencies matters in a research laboratory where the software stack is installed and maintained by a single researcher rather than a dedicated operations team. These properties carry forward to the AR-based gait visualization of Chapter 5, where the same communication layer lets the controller swap visualization components without modification, and to the multi-user mixed-reality composition of Chapter 7.

Worker Pool, Message Queue, and State Manager

The scheduler is composed of three subsystems. A *worker pool* maintains a registry of active optimizer and controller processes and dispatches incoming work to the appropriate worker. A *message queue* buffers trial parameter requests, trial outcomes, and configuration updates so that bursts of activity from either side do not cause back-pressure on the other. A *state manager*, implemented as an ordered dictionary keyed by trial index, tracks the optimization history, the current parameter vector, and per-trial metadata such as timestamp, subject identifier, iteration index, and trial outcome. The state manager is the in-memory representation of the session; its on-disk serialization is the persistence layer described in Section 4.5.2.

The non-blocking interaction between the optimizer and the controller is enforced at the scheduler queues. At the start of each reaching trial the controller reads the current parameter vector from its inbound queue; at the end of the trial it writes the computed outcome to its outbound queue and immediately proceeds to the next trial. The optimizer reads outcomes from its inbound queue whenever it is ready to update its surrogate, and writes new parameters to its outbound queue whenever a new acquisition point is available. The optimizer's per-iteration compute time is therefore decoupled from the controller's trial cadence. The scheduler state transitions for one acquisition iteration are shown in Figure 4.3.

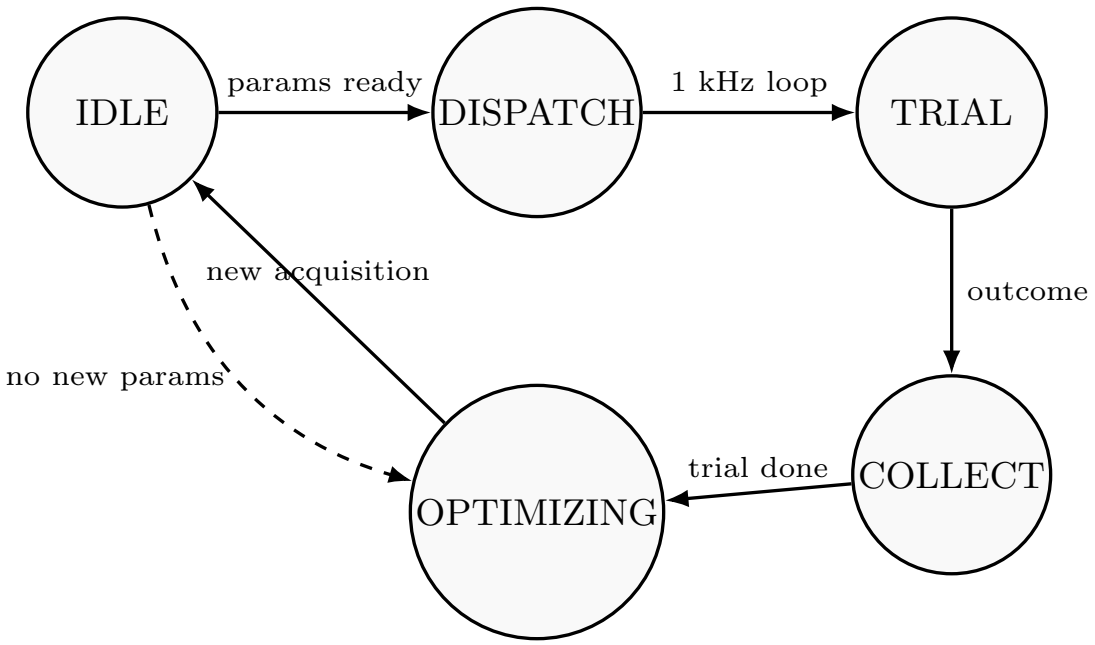


Figure 4.3: Task scheduler state machine for a single acquisition iteration. The OPTIMIZING state—during which the Gaussian-process surrogate is refit and the acquisition function is maximized—has no fixed time bound; the controller waits in IDLE, and the 1 kHz impedance loop on the controller thread continues to execute regardless of what the optimizer is doing.

4.5.2 Modular Architecture

Three-Module Decomposition

The system is decomposed into three processes: an optimizer module, a controller module, and an experiment monitor module (Figure 4.4). Each module exposes only message-level interfaces through its DEALER socket and has no direct access to the internal data structures of the other two. Any one module can be restarted, replaced, or upgraded while the other two continue running.

The decomposition allows the optimization algorithm to be changed without restarting the controller. During the experimental campaign of (Zahedi *et al.*, 2022), several optimization variants were compared: different Gaussian-process kernels, different acquisition functions, and a random-search baseline. Each swap required

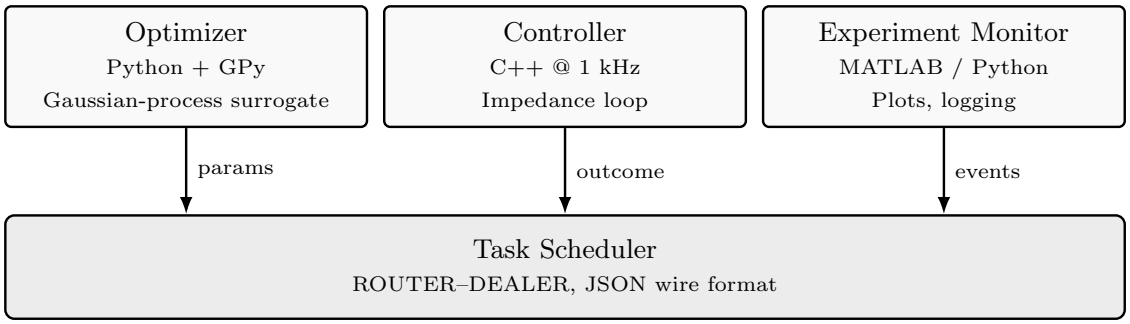


Figure 4.4: Three-module decomposition. The optimizer, controller, and experiment monitor are independent processes communicating only through the task scheduler. The controller does not need to be stopped or rebuilt when the optimization algorithm is changed.

stopping the optimizer module and starting a new one with a different command-line argument; the controller and the monitor could remain running during the swap.

Language-Agnostic Interfaces and JSON-Driven Configuration

The on-wire message format is plain JSON. Each message is a small object with a type field, a trial identifier, and a payload. JSON parsing is supported in Python, MATLAB, and C++ through standard libraries, so no interface description language, code generator, or build-time dependency is required for the language mix used by the study.

The same format is used for experiment configuration (Listing 4.5). A single configuration file specifies the subject identifier, session identifier, optimizer type, optimizer hyperparameters, controller gains, safety limits, data output path, and any algorithm-specific settings. Switching from one variant to another—for example, comparing a Gaussian-process Bayesian optimization against the random-search baseline—requires editing one field in the JSON file rather than modifying source code.

```
{
  "subject_id": "Subject1",
  "session": "Block3/Trial10",
  "optimizer": {
    "type": "gp_ei",       // "gp_ei" | "gp_ucb" | "random"
    "kernel": "matern52",
    "acquisition": "expected_improvement",
    "n_iterations": 16,
    "prior_trials": 13
  },
  "controller": {
    "rate_hz": 1000,
    "stiffness_N_per_m": 800,
    "damping_bounds_Ns_per_m": [20, 180]
  },
  "data_dir": "/home/user/Adaptive/AP_BO_newsetup/DATA/Subject1/Block3/"
}
```

Figure 4.5: Representative excerpt of the JSON experiment configuration used in the Zahedi et al. study (Zahedi *et al.*, 2022): subject and block identifiers, optimizer type and hyperparameters (Gaussian-process kernel, acquisition rule, iteration count, prior-trial count), controller rate, stiffness, damping bounds, and data directory. Algorithm swaps require only editing the `optimizer.type` field (`gp_ei`/`gp_ucb`/`random`); no source code change is needed.

Because an algorithm communicates with the rest of the system only through this JSON protocol, adding a new algorithm requires implementing the algorithm together with a short message handler that converts trial outcomes into the algorithm's input and the algorithm's output back into a parameter update. There is no shared inheritance hierarchy to satisfy, no virtual interface to implement, and no shared object to link against.

Cross-Session State Continuity

The most directly enabling design choice for the study is the on-disk serialization of the scheduler's state manager. The design addresses four requirements that the single-process prototype could not satisfy: connecting optimization results across trials, maintaining the parameter evolution $\theta(i) \to \theta(i+1)$ across rest breaks and process restarts, supporting session-to-session continuity for subjects whose sessions span multiple blocks, and keeping the optimizer–controller communication non-blocking so that the 1 kHz impedance loop is not stalled by the surrogate update.

After every trial the state manager writes a JSON document containing the Gaussian-process hyperparameters, the acquisition function history, every parameter vector proposed so far, and every trial outcome observed so far. When a session resumes—after a rest break, after a controller restart, or after a deliberate algorithm swap—the optimizer reads the most recent serialized state, reconstructs the Gaussian process, and proposes the next acquisition point as if no interruption had occurred.

This on-disk state is what bridges the natural breakpoints of a human-subject session. For the representative subject analyzed in (Zahedi *et al.*, 2022), the optimization used 13 prior iterations followed by 16 Bayesian optimization iterations, and the remaining four subjects converged between 12 and 16 Bayesian optimization iterations each. Each iteration used four reaching trials, and the total session duration was long enough that subjects routinely took rest breaks between blocks of twelve trials. The mapping from experimental requirements to architectural elements is summarized in Table 4.2.

Table 4.2: Experimental requirements imposed by the human-subject protocol of (Zahedi *et al.*, 2022) and the architectural elements that satisfy them.

Requirement from (Zahedi *et al.*, 2022)	**Architectural element**
Persist Gaussian-process state across rest breaks and restarts	State manager serialized to JSON after every trial
Keep the 1 kHz impedance controller running while the optimizer fits its surrogate	Non-blocking ZeroMQ queues on both the controller and optimizer sides
Compare algorithm variants without interrupting the subject	Plug-and-play algorithm modules behind the JSON message interface
Operate a Python optimizer, C++ controller, and MATLAB/Python monitor in parallel	Language-agnostic JSON wire format, three independent processes
Recover the session automatically after a worker crash	Auto-reload from the most recent serialized state
Eliminate manual parameter entry during the session	JSON-driven experiment configuration read once at startup

4.6 Results

The architecture described in this chapter was used to run the study reported in (Zahedi *et al.*, 2022). All five subjects completed their optimization sessions, the Bayesian optimization converged between 12 and 16 iterations per subject, and the controller reduced user energy expenditure by approximately 45 % and improved aggregate performance metrics by approximately 20 % relative to the fixed-damping baseline (Figure 4.6) (Zahedi *et al.*, 2022). The algorithmic and biomechanical contributions of the study belong to Zahedi et al.; the present chapter contributes the architectural substrate that made the sessions tractable. Specifically, session continuity across rest breaks, isolation of the Gaussian-process compute from the 1 kHz control loop, and the ability to swap optimizer variants without stopping the controller were not available in the earlier single-process prototype, and the protocol would not have completed in its published form without them.

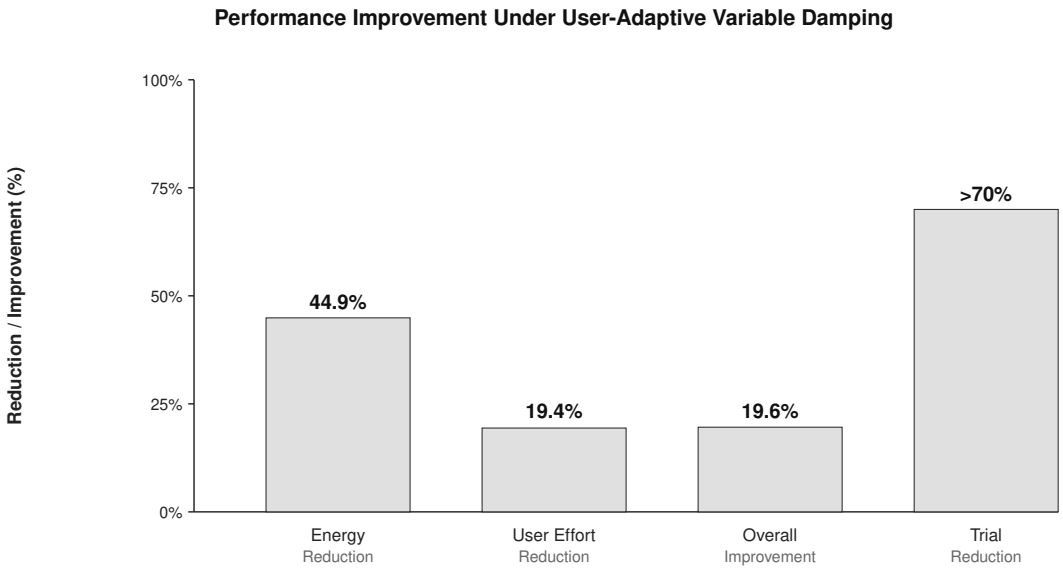


Figure 4.6: User energy expenditure and aggregate performance improvement under user-adaptive variable damping, reproduced from (Zahedi *et al.*, 2022). Experimental methodology and statistical analysis are reported in that paper.

4.7 Discussion

4.7.1 Properties of the Architecture

Four properties of the architecture kept the integration cost low during the study. The scheduler was lightweight (no master node, no service discovery, no schema registry, no build-time code generation). It was language-agnostic, so each module could be written in whichever language was most convenient for its task—Python for the surrogate model, C++ for the real-time control loop, and MATLAB or Python for the experiment monitor. It was message-driven, so adding a new module required only specifying the message format that module would consume and produce. And persistence was explicit: any module could be replaced without losing experimental state. The same four properties are reused by the configuration-hierarchy multi-robot framework of Chapter 9, which replaces the JSON configuration file with a hierarchical

YAML scheme, and by the personal AI infrastructure of Chapter 11, which applies them in a multi-provider language-model runtime.

The ZeroMQ ROUTER–DEALER substrate was selected over a ROS topic-based scheduler for two specific reasons. First, ZeroMQ permits brokerless point-to-point messaging with no master process, so a crash of the optimizer process does not propagate to the controller process; a ROS master architecture concentrates the failure mode in a single node whose crash halts the entire pipeline. Second, the asynchronous request–reply pattern of ROUTER–DEALER directly matches the cross-session continuity requirement of Section 4.5.2: the optimizer's pending requests are queued at the broker socket rather than at the controller, so the controller can restart without losing the optimizer's outstanding inference. The latency cost of the substrate was sufficient to keep the controller's 1 kHz cycle within budget during the campaign of (Zahedi *et al.*, 2022); a formal latency benchmark against a ROS topic baseline is identified as a future validation item in Chapter 14.

4.7.2 Limitations

Three limitations apply to the architecture described here. ZeroMQ provides no formal real-time guarantee, so the non-blocking property of the controller–scheduler interface is sufficient for the 1 kHz control rate used in the study but a pHRI application with stricter determinism requirements would need a real-time-safe transport. The JSON wire format is human-readable and language-agnostic but unversioned, so experimental compatibility across sessions relies on conventions rather than schema enforcement; the hierarchical YAML schema of Chapter 9 addresses this in a later system. The architecture was exercised with one manipulator and one human participant per session; multi-robot generalization is the subject of Chapter 9.

4.8 Summary

This chapter contributed two architectural patterns. The ZeroMQ ROUTER–DEALER distributed task scheduler (Obj 1) decoupled the Bayesian optimizer's surrogate fitting from the 1 kHz impedance controller and persisted optimizer state across session breaks through on-disk serialization. The three-module decomposition (Obj 2) with language-agnostic JSON message interfaces supported algorithm-swappable comparison without controller restart. The architecture supported the five-subject variable-damping study reported as the dissertation author's second-author contribution to IEEE RAL 2022 (Zahedi *et al.*, 2022), achieving approximately 45% reduction in user energy expenditure over the fixed-damping baseline. These patterns generalize to the configuration-hierarchy multi-robot framework of Chapter 9 and to the personal AI infrastructure of Chapter 11.

Chapter 5

SYSTEM 4: GAIT SYMMETRY VIA MIXED REALITY VISUAL DISTORTION

5.1 Introduction

Walking depends on coordination across the nervous, musculoskeletal, visual, vestibular, and auditory systems (Patterson *et al.*, 2010), and disruptions from stroke, Parkinson's disease, cerebral palsy, injuries, or aging cause motor impairments that compromise walking ability. A common manifestation is gait asymmetry—disparities in stance time, swing time, and step length between the right and left legs—arising from unequal propulsion forces caused by bilateral differences in muscle strength or by impaired neurological control of coordination (Patterson *et al.*, 2010). Gait asymmetry reduces walking speed, compromises balance and stability, decreases energy efficiency, increases fall risk, reduces community mobility, and diminishes quality of life, which motivates rehabilitation strategies that promote lasting gait adaptation (Patterson *et al.*, 2010).

5.2 Background and Related Work

5.2.1 Treadmill-Based Gait Adaptation

Traditional physiotherapy for gait asymmetry emphasizes repetitive walking practice under therapist guidance (Reisman *et al.*, 2005); the approach focuses on error reduction through external correction rather than self-generated motor learning. Split-belt treadmill training drives two belts at different speeds (Reisman *et al.*, 2005, 2007), and the speed difference creates sensory prediction errors—the gap between expected and actual sensory consequences of a movement (Wolpert *et al.*, 1998)—that

induce adaptive responses. The adapted patterns decay quickly to baseline once the perturbation is removed (Reisman *et al.*, 2009), so split-belt training is practical only when applied repeatedly over prolonged periods.

5.2.2 Visual Feedback Distortion

Visual feedback distortion (VD) is an alternative strategy (Kim and Krebs, 2012) that manipulates visual cues to exaggerate the representation of selected movement parameters. VD-induced aftereffects persist longer than those from split-belt training alone (Kim *et al.*, 2024). VD engages visuomotor adaptation—the recalibration of the mapping between visual input and motor output—driven by sensory prediction error. The adapted asymmetry through implicit VD (where subjects are unaware of the distortion) is limited. Monitor-based setups also depend on subject engagement: subjects have been reported as distracted or as shifting attention away from the monitor during training (Kim *et al.*, 2024).

5.2.3 AR Head-Mounted Displays for Gait Training

Augmented reality (AR) head-mounted displays address these engagement limitations. AR overlays feedback in the subject's visual field regardless of head orientation. AR maintains the visual connection to the physical environment, which reduces cybersickness compared to VR. AR keeps the feedback bars visible when the subject turns the head (Kim *et al.*, 2024). An earlier single-participant pilot at the NeuRRo Lab at ASU identified a candidate benefit of AR delivery for visual distortion gait feedback (Save *et al.*, 2026), motivating the controlled comparison reported in this chapter.

5.3 Hardware Platform

The experimental setup consists of an instrumented split-belt treadmill (Bertec Inc.), a motion capture system, and visual display hardware. The Bertec treadmill has two separate belts powered by independent motors, allowing for independent speed control of each belt. A safety harness system prevents falls without interfering with the subject's gait.

The Vicon motion capture system comprises eight infrared cameras (Bonita) that locate reflective markers placed on the subjects' lower legs. Seven markers are used: three on the left calf and heel, and four on the right calf and heel, enabling the motion capture system to differentiate between the two legs.

Visual feedback is delivered through either a 1920×1080 conventional monitor mounted on a gas-spring stand 2 m in front of the participant or a Microsoft HoloLens 2 AR head-mounted display. A custom Python real-time pipeline ingests motion capture and treadmill force data through the Vicon DataStream SDK, computes bilateral step length per heel strike, and serializes the result for transmission to the rendering target. Figure 5.1 shows the instrumented split-belt treadmill setup.

Figure 5.1: Experimental setup showing the instrumented split-belt treadmill with safety harness and motion capture markers. Visual feedback is provided via a traditional computer monitor or a Microsoft HoloLens augmented reality head-mounted display.

5.4 Software Requirements

The controlled AR-vs-CD comparison reported in this chapter imposes three software requirements that a monitor-only setup cannot satisfy directly (Save *et al.*, 2026):

1. Gait-cycle-locked visual feedback: the step-length asymmetry visualization had to update at heel strike, on the order of every 600–800 ms, so that the subject received a corrected feedback bar for the next step without a perceptible lag.

2. Identical experimental protocol on both a 1920×1080 conventional display (CD) and a HoloLens 2 head-mounted display (AR), with the distortion parameters, bar height mapping, heel-strike detection threshold, and trial structure held constant between the two conditions, so that the only difference between the two arms of the study was the rendering target.

3. Continuous Bertec split-belt treadmill and Vicon motion capture acquisition above 100 Hz throughout each session, independent of the 60 Hz frame rate of the visual feedback renderer, so that a dropped frame on the renderer side could never cause a drop in the kinematic and force data used for the step-symmetry analysis.

These three requirements map onto the dissertation's two objectives: the gait-cycle-locked feedback and the continuous high-rate acquisition encode real-time parallel concerns (Obj 1), while the CD/AR protocol equivalence encodes a modular separation concern (Obj 2). The mechanisms summarized in Table 5.1 addressed both, making the AR-versus-CD comparison a controlled study rather than a confounded one.

5.5 Software Architecture

The architecture combines two patterns: a ZeroMQ-based streaming pipeline for cycle-locked step-length feedback (addressing Obj 1) and a ScriptableObject-based configuration layer that allows the same Unity binary to drive either monitor or HoloLens 2 rendering through a configuration-asset swap (addressing Obj 2). The Bar Graph Visualization is implemented as a modular Unity component bound through the ScriptableObject layer, so swapping it for a different visualization (e.g., a treadmill heatmap or a virtual avatar in the multi-user mixed-reality composition of Chapter 7)

requires only a new ScriptableObject asset; the same ZeroMQ pipeline transports the underlying step-symmetry metric without modification. Table 5.1 summarizes.

Table 5.1: Architectural contributions of Chapter 5 to the dissertation's two objectives.

Objective	Method
Obj 1 (real-time parallel)	• ZeroMQ PUB-SUB pipeline for cycle-locked step metrics • 100+ Hz Vicon/treadmill ingest decoupled from 60 Hz renderer (non-blocking)
Obj 2 (modular scalable)	• ScriptableObject configuration (same Unity binary serves CD or AR via configuration-asset swap) • Swappable Bar Graph Visualization component

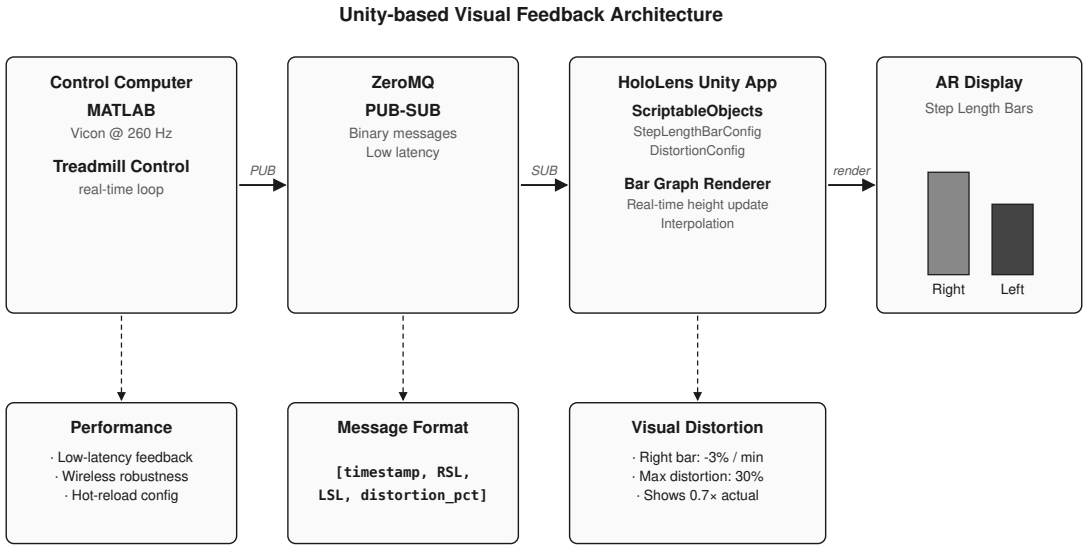


Figure 5.2: Unity-based visual feedback architecture showing ScriptableObject configuration, ZeroMQ communication layer, and HoloLens rendering pipeline. The same Unity binary serves both monitor and AR conditions through the configuration layer; the ZeroMQ publish-subscribe channel decouples the host pipeline from the rendering target, so a stall in the renderer never back-pressures the data acquisition thread.

5.5.1 Parallel Architecture

ZeroMQ Communication

Network communication between the host workstation (running the Python real-time pipeline for Vicon ingest and step-length computation) and the rendering target uses ZeroMQ for reliable, low-latency message passing. The PUB-SUB pattern broadcasts step length measurements from the host workstation, with both the PyGame UI for the conventional display and the Unity HoloLens 2 application for the AR display subscribing as parallel clients to receive real-time updates; the HoloLens operates on a separate wireless network from the host, and ZeroMQ's message queuing handles network variability without blocking the rendering loop or back-pressuring the host thread. Message serialization uses a compact binary format containing timestamp, right step length, left step length, and current distortion percentage, and the HoloLens application interpolates between received values to maintain smooth bar graph animation when packets are delayed or dropped.

5.5.2 Modular Architecture

ScriptableObject Configuration

Unity ScriptableObjects provide a data container pattern that separates configuration from runtime behavior. The visual feedback system encodes bar graph styling (colors, dimensions, transparency), distortion parameters (rate, maximum percentage), and calibration settings as ScriptableObjects, so researchers can modify experimental parameters without recompiling and hot-reload configuration during development. The StepLengthBarConfig ScriptableObject encapsulates visual properties (bar height scaling, foreground/background colors, Z-depth layering for AR rendering), and the DistortionConfig ScriptableObject manages the distortion schedule (rate of increase

per minute, maximum distortion percentage). The same Unity binary therefore serves both the conventional display and the AR conditions; switching between the two arms of the study is a configuration-asset swap rather than a code change.

Bar Graph Visualization

The visual feedback consists of a vertical bar graph with two bars representing bilateral step lengths, rendered as semi-transparent overlays in the HoloLens field of view so subjects see both the feedback graphics and the physical treadmill environment. Bar height updates in real time based on the gait cycle phase: at the start of the right swing phase (detected when the right foot crosses the left), the right bar drops to zero and increases proportionally as the right foot moves forward; upon heel strike (detected through ground reaction force threshold crossing), the right bar maintains its maximum height. The same process occurs for the left bar during the left swing phase. The bar mapping over a single gait cycle is shown in Figure 5.3.

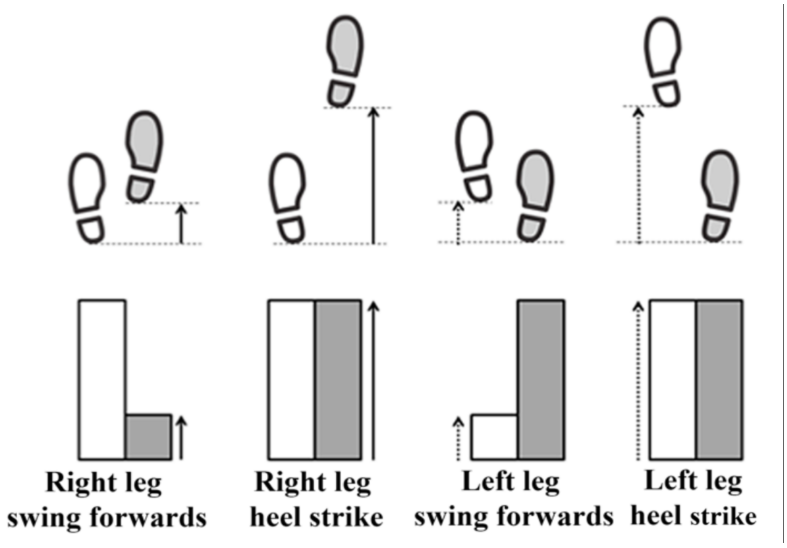


Figure 5.3: Visual representation of how the feedback bar heights correspond to each phase of the gait cycle. Bar height increases proportionally during the swing phase and locks at maximum upon heel strike.

The monitor refreshed at 60 Hz and the HoloLens 2 at 60 Hz, both above the 30 Hz minimum for smooth motion perception during gait feedback.

5.6 Results

Twelve healthy adults (24.6 ± 3.6 years; body mass 66.6 ± 13.7 kg; height 1.7 ± 0.1 m; six male, six female) participated in the comparison study under Arizona State University Institutional Review Board approval (STUDY00021299). None had prior exposure to the specific head-mounted display, and 66.7 % had no prior experience with HMD systems. Each participant completed two walking experiments with identical protocols on separate days, at least five days apart, with the visual feedback modality (conventional display, CD, or augmented reality, AR) randomized in order to mitigate session-order effects.

Each trial lasted 21 minutes at the participant's preferred walking speed (PWS, mean 1.19 ± 0.09 m/s across the cohort): a 2-minute baseline phase with veridical feedback, a 10-minute adaptation phase, and a 9-minute post-adaptation phase with the visual feedback removed. The adaptation phase consisted of a 5-minute ramp during which the displayed right step length was reduced by 3 %/min until reaching 15 % distortion at the 7^{th} minute, followed by a 5-minute steady-state hold at 15 %. Participants kept their gaze forward, were not informed of the visual distortion, and were instructed to match the displayed left and right step length bars. The same Python real-time pipeline drove both display modalities, so the only inter-condition difference was the rendering target (Save *et al.*, 2026).

Step length symmetry ratio (SSR) is calculated as:

$$\overline{SSR}(\%) = \frac{\text{RSL} - \text{LSL}}{\frac{1}{2}(\text{RSL} + \text{LSL})} \times 100, \tag{5.1}$$

where RSL and LSL denote the right and left step lengths measured at consecutive heel strikes; positive values indicate longer right steps and negative values indicate longer left steps. The cycle-by-cycle $\overline{SSR}$ stream is the primary outcome measure for the AR-vs-CD comparison reported below.

During the baseline phase, the population-averaged step symmetry ratio was $0.13 \pm 0.92\,\%$ under AR and $-0.17 \pm 0.96\,\%$ under CD, with no significant difference between modalities (Wilcoxon signed-rank test, $p = 0.55$). The two display systems delivered comparable veridical feedback prior to perturbation, satisfying the precondition for the subsequent ramp comparison.

During the ramp adaptation phase, $\overline{SSR}$ increased proportionally to the imposed distortion in both conditions. A linear mixed-effects model fit to $\overline{SSR} \sim \text{Modality} \times \text{Distortion} + (1 \mid \text{Participant})$ showed no significant interaction between modality and distortion level ($p = 0.98$), indicating comparable sensitivity to the imposed sensory prediction error across the two modalities. By the end of the ramp at $15\,\%$ distortion, $\overline{SSR}$ reached $12.17 \pm 2.04\,\%$ under AR and $12.21 \pm 2.63\,\%$ under CD.

During the steady-state adaptation phase ($15\,\%$ distortion held for 5 minutes), the CD condition exhibited a slightly higher $\overline{SSR}$ ($13.28 \pm 2.63\,\%$) than the AR condition ($12.01 \pm 2.75\,\%$); the difference approached but did not reach statistical significance (paired t-test, $p = 0.059$, Cohen's $d = 0.5$). This trend suggests that the modality may modestly affect the expression of the stabilized motor pattern, although the underlying sensorimotor recalibration was preserved across both delivery formats.

Following feedback removal, $\overline{SSR}$ decayed over time in both conditions. A log-transformed linear mixed-effects model fit to the post-adaptation trajectory revealed comparable initial retained magnitudes ($p = 0.49$) and decay rates ($p = 0.47$) between the two modalities. At three representative post-adaptation time points, the modalities did not differ at the early ($4.43 \pm 3.28\,\%$ AR vs. $5.85 \pm 3.27\,\%$ CD), middle ($1.90 \pm 2.95\,\%$

vs. $2.20 \pm 3.04\,\%$), or late ($2.07 \pm 3.90\,\%$ vs. $1.90 \pm 4.02\,\%$) phases, with all comparisons returning $p_{\text{adj}} = 1$ after Holm correction (Wilcoxon signed-rank test). The cumulative retention summarized by the area under the curve over the 9-minute post-adaptation interval was $23.84 \pm 19.46\,\%\cdot\text{min}$ for AR and $23.05 \pm 23.49\,\%\cdot\text{min}$ for CD, with no significant difference ($p = 0.79$).

The single-participant progression of $\overline{SSR}$ across the trial appears in Figure 5.4, the population-averaged trajectory in Figure 5.5, and the post-adaptation comparison at three time points in Figure 5.6.

These results indicate that AR delivers visually distorted gait feedback comparably to a conventional monitor across the baseline, adaptation, and post-adaptation phases. The single trend ($p = 0.059$ at steady-state) suggests that modality may modestly influence the expression of the stabilized motor pattern without altering the underlying error-driven recalibration (Save *et al.*, 2026).

5.7 Discussion

The controlled AR-vs-CD comparison reported in this chapter addressed a single architecturally enabled question: does AR delivery preserve the established visual distortion gait paradigm comparably to a conventional monitor? The architecture met the chapter's three requirements through two complementary axes. Real-time integrity was provided by a ZeroMQ publisher–subscriber channel for non-blocking communication between the host pipeline and the rendering target and a Vicon DataStream SDK ingest path that operated above 100 Hz so that neither the 60 Hz monitor nor the 60 Hz HoloLens 2 frame rate ever bottlenecked the data acquisition. Configuration flexibility was provided by a ScriptableObject-based layer for condition switching without code changes. This unified delivery made the AR-vs-CD comparison a controlled study rather than a confounded one: the visual distortion protocol, the

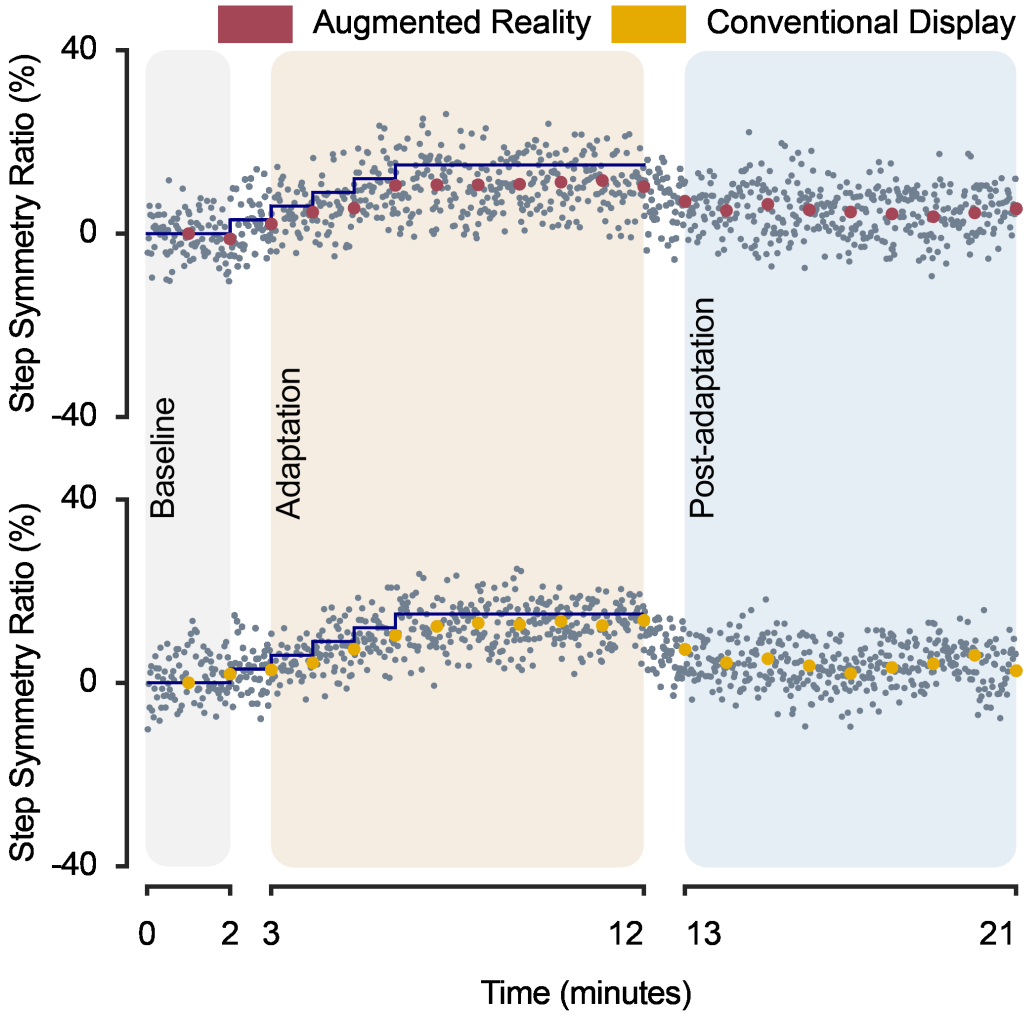


Figure 5.4: Progression of $\overline{SSR}$ during the 21-minute walking trial under the AR condition (top) and the CD condition (bottom). Smaller gray circles represent $\overline{SSR}$ values for each gait cycle; larger colored circles indicate the 1-minute averages. The distortion level is shown as a solid navy-blue line across the baseline and adaptation phases. Adapted from (Save *et al.*, 2026).

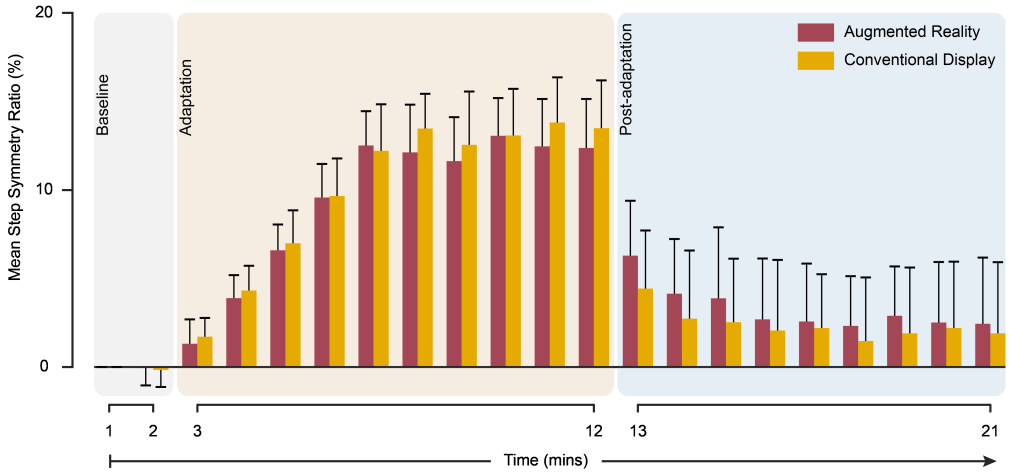


Figure 5.5: Population-averaged $\overline{SSR}$ across all 21 minutes under the AR (maroon) and CD (gold) conditions. Error bars indicate standard deviation across N = 12 participants (Save *et al.*, 2026).

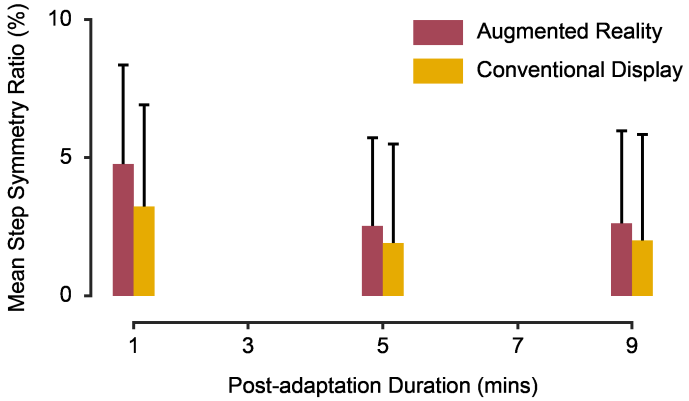


Figure 5.6: Population-averaged $\overline{SSR}$ at the early (minute 1), middle (minute 5), and late (minute 9) post-adaptation time points. AR (maroon) and CD (gold) bars; error bars indicate standard deviation. No statistically significant differences between modalities at any time point (Save *et al.*, 2026).

heel-strike detection, the bar height mapping, the 1-minute analysis windows, and the analysis pipeline were identical between the two arms of the study, with only the rendering target differing.

The AR versus CD comparison with N = 12 participants tests whether the display modality itself influences gait adaptation under matched visual feedback. AR delivers visually distorted gait feedback comparably to a conventional monitor across baseline ($p = 0.55$), ramp adaptation ($p = 0.98$ modality–distortion interaction), and post-adaptation phases (initial retained magnitude $p = 0.49$, decay rate $p = 0.47$, AUC $p = 0.79$) (Save *et al.*, 2026). The single observed trend was a slightly higher steady-state asymmetry under CD ($p = 0.059$, $d = 0.5$, not significant at $\alpha = 0.05$); this suggests that once the motor pattern has stabilized, modality may modestly shape its expression, while the underlying error-driven recalibration remains preserved across the two modalities.

This comparable behavior has two implications. First, in healthy young adults with robust sensorimotor adaptation, the choice between AR and conventional display does not appear to alter the magnitude or retention of the adapted gait pattern; the visual distortion paradigm engages similar core learning mechanisms regardless of how the bars are rendered. Second, the comparable performance establishes AR as a viable alternative to conventional monitor displays for visual feedback distortion research and rehabilitation. AR delivery preserves the experimental validity of the established paradigm while removing the spatial constraints of a fixed monitor: AR maintains feedback alignment with natural gaze, reduces attentional switching between feedback and walking surface, and is portable enough to extend gait adaptation research into clinical or community settings without the laboratory monitor infrastructure (Save *et al.*, 2026).

Several factors may have contributed to the absence of a significant modality effect in this population. Healthy young adults exhibit robust locomotor adaptation that attenuates modality-sensitive differences (Kim *et al.*, 2024); N=12 limits sensitivity to moderate effect sizes (power < 0.8 for $d < 0.5$); and the step symmetry ratio does not probe sensory weighting or perceptual differences between AR and CD. Future studies in clinical populations with impaired sensory integration, in more demanding walking environments, or with neurophysiological measures may reveal modality-dependent advantages for AR that the present healthy-adult comparison did not surface.

With this architecture in place, the AR-vs-CD comparison demonstrated that the display modality itself does not alter gait adaptation magnitude or retention in healthy young adults, supporting the feasibility of AR as a delivery platform for visual distortion-based gait rehabilitation. The architecture thus transformed the question from "does visual distortion improve gait symmetry" (already established in (Kim *et al.*, 2024)) to "does AR delivery preserve the established adaptation behavior," and the present evidence supports an affirmative answer (Save *et al.*, 2026).

End-to-end latency from foot movement to HoloLens display update was not measured; this remains a future-work item.

The visual feedback substrate is one-way: the system delivers a distorted bar to the participant but cannot impose forces on the limb. The participant interacts with the treadmill belt rather than with a robot end-effector, so the platform cannot study how attention or kinematic behavior changes when an admittance-controlled robot is in the loop. Chapter 6 introduces a 6-DoF collaborative robot with a built-in force-torque sensor under the same Lab software substrate (Unity, HoloLens 2, low-latency host pipeline), extended with a dexterous hand and eye tracking that instrument the visuomotor loop directly during admittance-controlled reaching.

5.8 Summary

This chapter contributed two architectural patterns. The ZeroMQ-based streaming pipeline (Obj 1) sustained cycle-locked step-length feedback with 100+ Hz Vicon/treadmill ingest decoupled from 60 Hz rendering. The ScriptableObject configuration layer (Obj 2) allowed the same Unity binary to drive both monitor and HoloLens 2 AR through a configuration-asset swap, with a swappable Bar Graph Visualization component. The architecture supported the controlled $N = 12$ AR-vs-CD comparison reported as the dissertation author's co-first-author contribution in (Save *et al.*, 2026), finding comparable gait adaptation magnitude and retention across the two display modalities ($p = 0.55$ at baseline, $p = 0.98$ for the modality–distortion interaction). These patterns carry forward to the single-user AR admittance rehabilitation platform of Chapter 6 and the multi-user mixed-reality composition of Chapter 7.

Chapter 6

SYSTEM 5: SINGLE-USER AR ADMITTANCE REHABILITATION

6.1 Introduction

Neurological conditions such as stroke, traumatic brain injury, and spinal cord injury compromise proprioception—the sense of limb position and movement without visual confirmation. Patients with impaired proprioception rely on visual feedback during rehabilitation tasks. Conventional two-dimensional (2D) monitor displays force repeated gaze shifts between the monitor and the robot handle, and each shift interrupts the visuomotor loop, degrades movement coordination, and increases cognitive load (Wenk *et al.*, 2022). Head-mounted display (HMD)-based augmented reality co-locates the visual feedback with the patient's physical limb, so the target, the robot, and the arm appear in a single visual field without gaze transitions. The clinical and HCI literature attributes the HMD advantage to reduced cognitive load and more naturalistic visualization, primarily through subjective questionnaires (Wenk *et al.*, 2022). Eye tracking has been used during robotic teleoperation (Dragan *et al.*, 2013) and collaborative assembly (Amershi *et al.*, 2019), but gaze-based measurement during admittance-controlled rehabilitation reaching has not been reported.

Measuring whether AR reduces attention shifting requires gaze samples timestamped against the robot-state stream. Measuring whether reduced attention translates to improved kinematic performance requires the same gaze stream aligned with end-effector state, applied wrench, and distributed grip force. The four concurrent real-time data streams must share a single clock for post-hoc causal analysis, a combination uncommon in rehabilitation robot software. The system described in this

chapter integrates a UR16e collaborative robot arm with a built-in force/torque sensor, a Tesollo DG-5F dexterous hand that grasps the patient's hand in a therapist-like configuration, a Microsoft HoloLens 2 with built-in eye tracking, and a 2D monitor baseline condition. A single ROS2 Humble host synchronizes the four data streams, and the hand-guided reaching task that exercises the platform produces the attention measurement, the kinematic outcome, and the grip-force behavior within the same trial.

6.2 Background and Related Work

6.2.1 Admittance-Controlled Rehabilitation Robots

Admittance-controlled rehabilitation robots translate the patient's applied force into compliant motion through a virtual mass-damper-spring system. The damping coefficient B_d governs the effort required to move the robot: high damping demands more force and restricts fast motion, while low damping permits rapid response but risks oscillation. The present chapter holds the damping constant and varies the display modality between Monitor and AR. The architecture observes the user's behavior upstream of the controller through four temporally aligned streams: gaze, end-effector state, applied wrench, and distributed grip force.

6.2.2 HMD AR for Robot-Aided Rehabilitation

Several groups have investigated HMD-based AR for upper-limb rehabilitation (Al-Issa *et al.*, 2012). The reported advantage is subjective: post-hoc questionnaires register higher engagement scores and self-reported spatial awareness. The literature treats the display modality as an independent variable that affects the patient's experience and reports questionnaire scores; it does not instrument the visuomotor

loop itself to measure whether the display modality changes the patient's physical interaction behavior with the robot.

6.2.3 Eye-Tracked Attention in Physical Human-Robot Interaction

Eye tracking has been used to study attention during robotic teleoperation (Dragan *et al.*, 2013) and during collaborative assembly (Amershi *et al.*, 2019). Its application in rehabilitation settings where the patient is physically coupled to a robot through an admittance controller has not been reported. The architectural challenge is not the eye tracker itself—the HoloLens 2 provides built-in gaze at 30 Hz—but its temporal alignment with the 500 Hz robot state stream, which this chapter's architecture solves through a shared ROS2 clock that allows post-hoc correlation between individual gaze shifts and the corresponding kinematic events.

6.3 Hardware Platform

The platform combines four commercial components under a single ROS2 Humble host (Table 6.1).

Table 6.1: Hardware components of the single-user AR admittance rehabilitation platform.

Component	**Model**	**Role**
Robot arm	Universal Robots UR16e, 6-DoF, 16 kg payload, 900 mm reach	Admittance-controlled reaching platform (500 Hz servo, built-in F/T sensor)
Robot hand	Tesollo DG-5F, five fingers, 20 active joints, 880 g, backdrivable	Grasps the patient's hand; provides distributed grip-force measurement at each fingertip
Head-mounted display	Microsoft HoloLens 2, 52° FoV, 60 Hz render	Co-located AR feedback; integrated eye tracker samples gaze at 30 Hz
Monitor (control)	24-inch LCD at 60 cm viewing distance	Baseline 2D feedback
Force sensing	UR16e built-in F/T sensor (500 Hz) + DG-5F fingertip force sensors (100 Hz)	Admittance wrench input and distributed grip-force profile

The robot grasps the patient's hand in a hand-over-hand configuration rather than presenting a rigid handle, so the DG-5F finger sensors expose distributed grip-force measurements at each fingertip during reaching, rather than at a single contact point. A rigid handle cannot distinguish a confident grip from an anxious one. The HoloLens 2 head-mounted display that delivers the AR feedback is shown in Figure 6.1.

Figure 6.1: Microsoft HoloLens 2 head-mounted display used as the augmented reality endpoint. The integrated eye tracker streams gaze origin and direction at 30 Hz alongside the head pose at 60 Hz; both streams are timestamped against the same ROS2 clock as the 500 Hz admittance control loop, so saccade events and kinematic transitions can be aligned at the trial level.

6.4 Software Requirements

The hand-guided reaching protocol of Section 6.6 imposes four software requirements that a conventional rehabilitation robot software cannot satisfy:

1. Concurrent acquisition of four real-time data streams—500 Hz robot state and applied wrench, 30 Hz HoloLens gaze, 100 Hz fingertip grip force, and 60 Hz AR rendering—under a single host clock so that gaze events can be aligned to specific kinematic events without post-hoc time synchronization.

2. 500 Hz Cartesian admittance loop on the same host without blocking on any of the slower sensor streams, so that the patient can guide the robot through compliant motion with the damping coefficient as the only parameter that varies the perceived effort.

3. Display modality (Monitor vs. AR) and admittance damping (5, 15, 30 Ns/m) selectable through configuration alone, so that the same compiled binary delivers all six conditions of the 2×3 within-subject protocol.

4. Distributed grip-force measurement at five fingertips rather than at a single contact point, so that the user's interaction comfort can be observed alongside the kinematic outcome and the gaze allocation within the same trial.

6.5 Software Architecture

The architecture combines two patterns. A multi-rate parallel pipeline sustains a 500 Hz Cartesian admittance loop, 30 Hz eye tracking, 100 Hz grip-force logging, and 60 Hz AR rendering concurrently under a single ROS2 clock (addressing Obj 1). A three-layer separation matching the System 2 pattern of Chapter 3 (control / interface / application) lets the display modality (AR vs. Monitor) and the admittance damping change through configuration alone (addressing Obj 2). Table 6.2 summarizes.

Table 6.2: Architectural contributions of Chapter 6 to the dissertation's two objectives.

Objective	**Method**
Obj 1 (real-time parallel)	500 Hz Cartesian admittance loop; concurrent 30 Hz gaze, 100 Hz grip, and 60 Hz rendering on one ROS2 clock
Obj 2 (modular scalable)	Three-layer separation (control / interface / application)

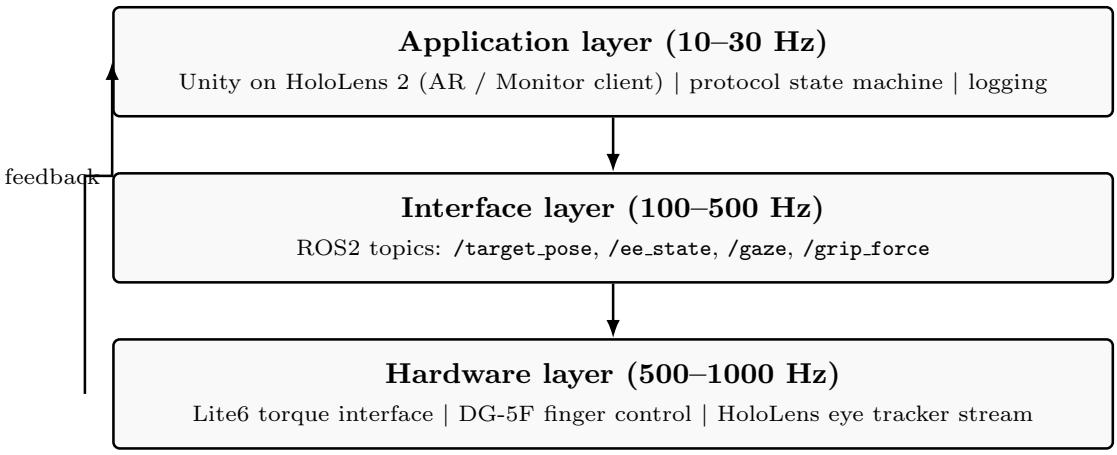


Figure 6.2: Three-layer software architecture. Each layer is deployable and testable independently; changing the display modality (AR vs. Monitor) or the admittance damping requires modifying only the application-layer configuration.

6.5.1 Parallel Architecture

Admittance Control Layer

The 500 Hz Cartesian admittance loop is summarized in Figure 6.3.

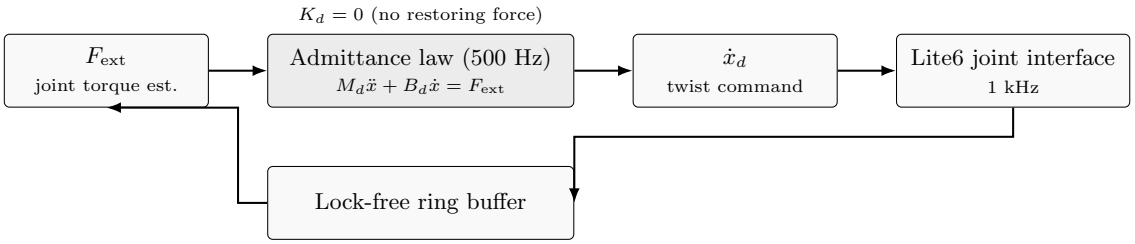


Figure 6.3: 500 Hz Cartesian admittance loop. External wrench is read from the UR16e built-in force/torque sensor, filtered through a lock-free ring buffer shared with the servo interface (the same lock-free ring-buffer pattern as Chapter 2), and integrated through the admittance dynamics to produce a twist command.

The admittance loop implements the classical Cartesian admittance law

$$M_d\,\ddot{x} \;+\; B_d\,\dot{x} \;+\; K_d\,x \;=\; F_{\text{ext}}, \tag{6.1}$$

in the end-effector frame, where M_d is the virtual inertia, B_d the damping, K_d the stiffness, and F_{ext} the external wrench measured by the UR16e built-in force/torque

sensor at 500 Hz. The loop runs on a dedicated thread, reuses the lock-free ring-buffer pattern of Chapter 2, and commands the robot through the `ur_robot_driver` (Universal Robots and ROS-Industrial Consortium, 2024) position interface. The stiffness is held at $K_d = 0$ so the robot does not pull back toward a nominal trajectory: the patient must actively navigate to the target, and the damping B_d alone determines how hard the reach is. Three damping settings (5, 15, and 30 Ns/m) are used to vary task difficulty while keeping all other control parameters fixed. The directional signal flow of the admittance dynamics is shown in Equations 6.1 and 6.3.

$$M_\lambda \ddot{\lambda} + B_\lambda \dot{\lambda} + K_d \lambda = \gamma \tag{6.2}$$

$$\text{where } \lambda = \mathrm{Log}(T_{\text{disp}})^\vee \in \mathbb{R}^6, \quad \gamma = \mathrm{dexp}_\lambda^\top \mathrm{Ad}_{T_d}^{-\top} F_{\text{world}}.$$

$$T_{\text{disp},\,k+1} = T_{\text{disp},\,k}\ \exp\Big((\dot{\lambda}_{k+1}\,\Delta t)^\wedge\Big) \tag{6.3}$$

Equation (6.1) is a directional summary of the admittance loop: the measured external wrench F_{ext} is transformed through the body-frame admittance dynamics into a modified pose T_{mod} commanded to the inverse-kinematics layer. The decoupled-axis form is sufficient for the single-user reaching protocol of this chapter; the SE(3) Lie-group form used by the multi-platform deployment in Chapter 12 is mathematically related and shares the same compiled controller binary.

Unity-ROS2 Visual Feedback Bridge

The Unity–ROS2 data pipeline that connects the HoloLens client to the admittance loop and the DG-5F grip-force node is shown in Figure 6.4.

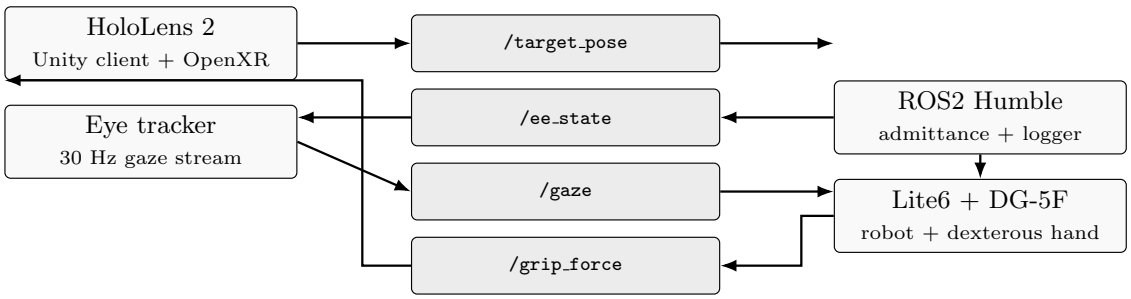


Figure 6.4: Unity–ROS2 data pipeline. Gaze samples flow from the HoloLens eye tracker through `/gaze` at 30 Hz; end-effector state flows from the admittance loop to the Unity renderer; grip force is published by the DG-5F node at 100 Hz. The display mode (Monitor or AR) is selected by a configuration value in the Unity client without any change to the ROS2 side.

The Unity client on the HoloLens 2 renders the visual feedback in two modes. The Monitor mode displays a colored target circle on the desktop display; the AR mode renders a colored sphere at the corresponding physical three-dimensional location in the UR16e workspace through a QR-based co-registration step. In both modes the client exchanges robot state and target poses with the ROS2 backend through the ROS TCP Connector, a Unity package that bridges Unity C# and ROS2 messages over a TCP socket. The display mode is a configuration value, so switching between Monitor and AR conditions during the experimental campaign requires only re-deploying the Unity scene, not re-building the ROS2 nodes.

Dexterous Hand Interface and Grip Force

The DG-5F hand is driven by a separate ROS2 node that enforces four operating phases:

1. HOME: the hand is open and the patient places their hand inside the palm.

2. GRASP: the five fingers close to a programmable grip force of approximately 3 N per fingertip; the phase transition is reported to the application layer so the reaching trial can begin.

3. REACH: the hand holds the grip while the UR16e follows the patient's applied wrench through Equation 6.1, and the fingertip force sensors publish on `/grip_force` at 100 Hz.

4. RELEASE: the hand opens and the patient withdraws their hand for the inter-trial rest.

The phase transitions are driven by the application-layer state machine, not by the control loop, which keeps the 500 Hz admittance path independent of the slower grip protocol.

6.5.2 Modular Architecture

The three-layer separation supports the chapter's experimental scope—two display modalities (AR vs. Monitor) crossed with three damping settings—on a single binary. The application layer holds the trial-state machine, the target geometry, and the display-modality flag; the interface layer carries ROS2 topics for end-effector state, gaze, grip force, and trial events; the hardware layer drives the UR16e, the DG-5F, and the HoloLens 2. Changing the display modality or the admittance damping is a configuration change in the application layer and a Unity scene re-deploy, with the interface and hardware layers unchanged. The three layers separate the variables that the experimental protocol varies from the components that remain identical between conditions.

6.6 Results

6.6.1 Hand-Guided Reaching Task

The reaching task is the experimental substrate through which the display-modality effect on attention is made observable. The task is designed so that a single reaching

trial simultaneously produces the four data streams—gaze, kinematics, grip force, and trial state—that the downstream analysis requires.

Ten target positions are distributed in the UR16e workspace: five at the home-position plane (planar reaches) and five elevated at a 15 cm vertical offset (three-dimensional reaches). A trial begins when the hand transitions to REACH: the target sphere (AR) or target circle (Monitor) appears, the participant pushes the UR16e toward it through the admittance interface, acquires the target by holding the end-effector within a 15 mm radius for 0.5 s, and the hand transitions to RELEASE. A 2 s inter-trial rest separates consecutive trials.

The experimental design is a 2×3 within-subject factorial: two display conditions (Monitor, AR) crossed with three damping conditions (5, 15, 30 Ns/m). The six conditions are counterbalanced across participants. Each condition comprises 20 reaching trials, yielding 120 trials per participant. The logging service captures end-effector pose and wrench at 500 Hz, gaze samples at 30 Hz, and fingertip forces at 100 Hz, with all streams timestamped against the same ROS2 clock.

The constant-damping design is intentional. The present study holds damping constant to isolate the effect of the display modality on the user's behavior upstream of the controller.

6.6.2 Measurement Pipeline

Four categories of measurement are recorded simultaneously during every reaching trial (Table 6.3). The architectural requirement imposed by each category is listed in the rightmost column; together they define the multi-rate design.

The four categories are organized around a four-stage causal chain that the platform observes within a single trial. The display modality alters the participant's gaze allocation. The gaze allocation contaminates the kinematic intent signal that

arrives at the admittance controller. The contaminated intent signal manifests in the kinematic outcome. The grip-force stream provides a parallel comfort observation along the same chain. The single-clock alignment of the four streams supports post-hoc mediation analysis on data drawn from the same trial, rather than separate experiments at each stage.

Table 6.3: Measurement categories and the architectural requirement each imposes.

Category	**Measurements**	**Architectural requirement**
Attention	gaze shift count (N_{shift}), gaze-on-target ratio	30 Hz eye-tracking stream timestamped against ROS2 clock; I-VT classifier at 100°/s for saccade detection
Kinematics	movement time, path ratio, SPARC smoothness, rise time T_{rise}, overshoot OS%	500 Hz end-effector state without admittance-loop interference
Interaction	mean grip force, grip-force coefficient of variation, per-finger distribution	100 Hz fingertip force logging on a separate thread
Subjective	NASA-TLX, System Usability Scale	Post-condition questionnaires

Attention Measurements

The gaze stream is classified into fixations and saccades using the I-VT (velocity-threshold identification) algorithm with a threshold of 100°/s, consistent with the recommendations of (Salvucci and Goldberg, 2000). A *gaze shift* is defined as a saccade whose landing falls outside the target region. The gaze shift count N_{shift} per trial is the primary attention variable: the hypothesis is that AR reduces N_{shift} by eliminating the need to shift between the monitor and the hand. The gaze-on-target ratio complements N_{shift} by measuring how much of the trial duration the participant's gaze remained within the target region.

Kinematic Measurements

The end-effector trajectory is captured at 500 Hz through the UR16e servo interface. Movement time is measured from trial onset to target acquisition. The path ratio (actual path length divided by straight-line distance) quantifies trajectory efficiency. The SPARC smoothness metric (Balasubramanian *et al.*, 2015) quantifies movement fluidity in the frequency domain. Rise time T_{rise} (10% to 90% of peak velocity) and percent overshoot OS% together characterize the agility-stability tradeoff within each trial.

Dual-Metric Admittance Reporting

Admittance-controller evaluation is reported as two metrics rather than as a single scalar error. A single scalar error sums two different quantities: the controller's tracking fidelity to its internal admittance reference, and the compliant displacement that the admittance design prescribes under an applied force. The summed scalar can reach tens of millimeters on a step-force trial when the end-effector is following its admittance reference to sub-millimeter accuracy, because the apparent error is the design-prescribed compliant displacement (Colgate and Schenkel, 1997). The two quantities are therefore reported separately:

$$e_{\text{track}} = \|p_{\text{EE}} - p_{\text{modified}}\|, \qquad x_{\text{comply}} = \|p_{\text{modified}} - p_{\text{desired}}\|, \tag{6.4}$$

where $p_{\text{modified}} = p_{\text{desired}} + \Delta p_{\text{disp}}$ is the admittance-modified reference produced by integrating the $SE(3)$ admittance dynamics. e_{track} measures the controller's faithfulness to the admittance dynamics; x_{comply} is the designed compliant displacement, a function of the experimental condition rather than of controller errors. Each admittance-related figure in this chapter splits e_{track} and x_{comply} into separate panels.

User Intent Signal

A controller-agnostic measure of the user's input quality (Zahedi *et al.*, 2022) is the kinematic intent signal

$$u(t) \;=\; \dot{x}(t)\,\ddot{x}(t), \tag{6.5}$$

computed on the end-effector translation projected onto the reach axis from the same 500 Hz pose stream that feeds the admittance loop, after a Savitzky–Golay filter (order 2, window 11 samples). Two summary statistics are reported per trial: the signal-to-noise ratio $\mathrm{SNR}_u = \mathbb{E}|u|/\sqrt{\mathrm{Var}\,u}$ and the zero-crossing rate ZCR_u. Because $u(t)$ is computed from end-effector kinematics rather than from any controller-internal state, it is comparable across the two display conditions and across the three damping levels without controller-specific assumptions. $u(t)$ is the second stage of the four-stage chain introduced at the start of this section.

Interaction Measurements

The five-fingertip force stream provides a distributed view of the patient's grip during reaching. The mean grip force across all fingertips reflects the overall interaction effort; the coefficient of variation (CV) across fingertips reflects grip consistency. A high CV with a low mean suggests a relaxed but asymmetric grip (comfort), while a high mean with low CV suggests a uniform clench (anxiety). This distinction is unavailable from a rigid handle.

Temporal Alignment

All four streams are timestamped against the single ROS2 Humble clock and consumed by the logging service through a single `message_filters::TimeSynchronizer`, so a specific gaze shift event aligns with the kinematic behavior that followed it and with

the grip-force change that accompanied it without post-hoc time synchronization. The alignment is a property of running the admittance controller, the eye-tracking publisher, and the grip-force publisher as ROS2 nodes on the same host that call the same `now()` clock.

6.6.3 Preliminary Single-Pilot Result

The platform was exercised through a representative single-pilot reaching trial under the AR display condition to validate that the full pipeline sustains concurrent operation of the four data streams. The logging service recorded all four streams without dropped samples, and the maximum clock offset between any two streams remained below 1 ms across the trial, consistent with the ROS2 intra-host message-passing latency.

The aligned-stream evidence panel in Figure 6.5 is the preliminary result reported here. The reaching-task workspace layout is shown in Figure 6.6. Multi-participant analysis of the display-modality effect on gaze shifting, kinematic performance, and grip-force variability is deferred to the future-work agenda of Chapter 14, where the corresponding pilot, protocol, and statistical procedure are identified.

6.7 Discussion

6.7.1 Rationale for the Dexterous Hand

The DG-5F hand is not the only way to realize a hand-guided reaching task. A rigid handle attached to the UR16e end-effector would produce the same 6-DoF admittance behavior and would be mechanically simpler. The dexterous hand is selected for the distributed measurement it enables: per-fingertip force rather than a single handle-mounted load cell. During a reach under admittance control, a confident

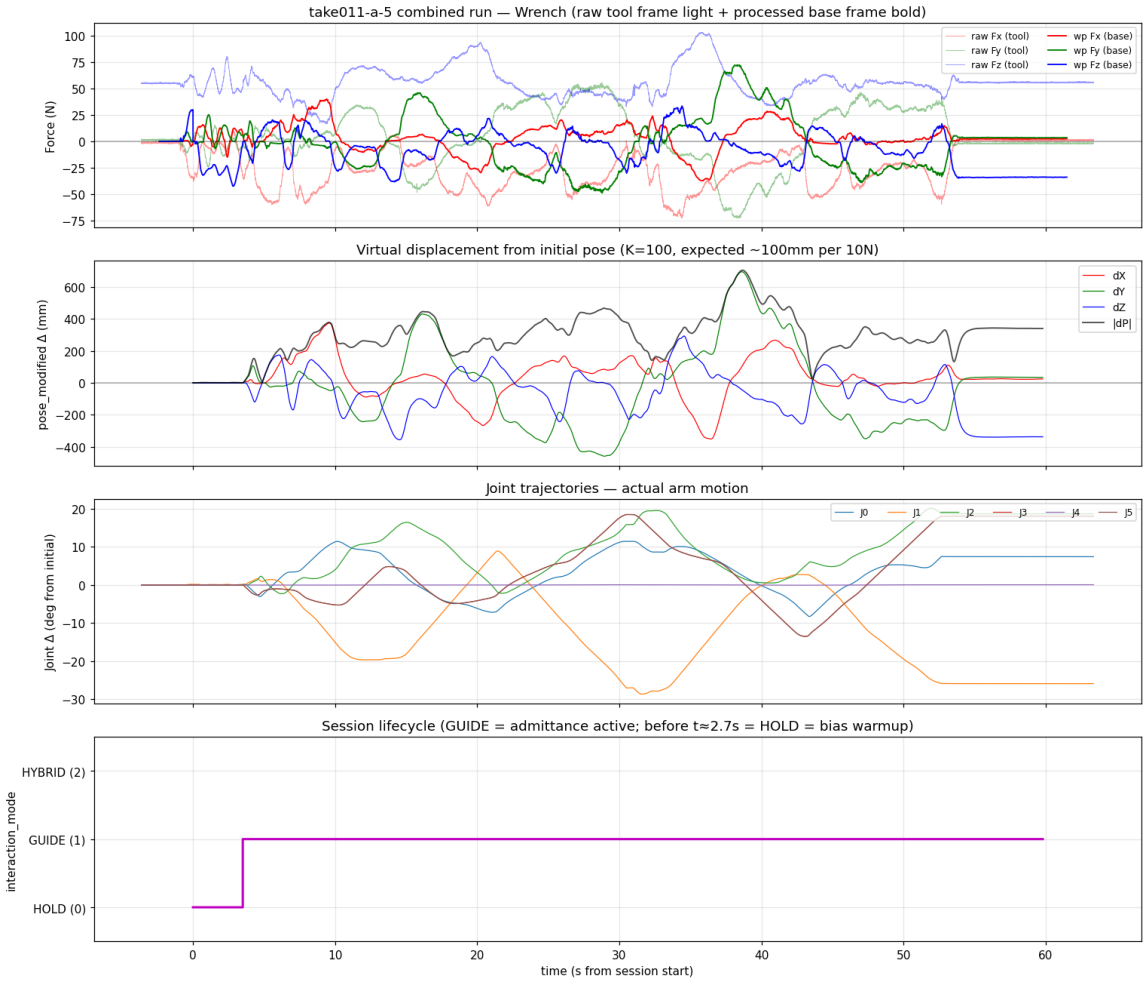


Figure 6.5: Aligned-stream evidence panel from a representative pilot reach under the AR display condition: admittance-controlled reach trajectory and the corresponding fingertip force profile. The end-effector follows the operator-applied wrench through the admittance dynamics of Equation (6.1); the five-fingertip force stream from the DG-5F provides the distributed grip-force coverage that a rigid handle cannot supply. This panel verifies that the four data streams (admittance command, end-effector trajectory, gaze samples, and fingertip force) are simultaneously captured and temporally aligned across a full reach.

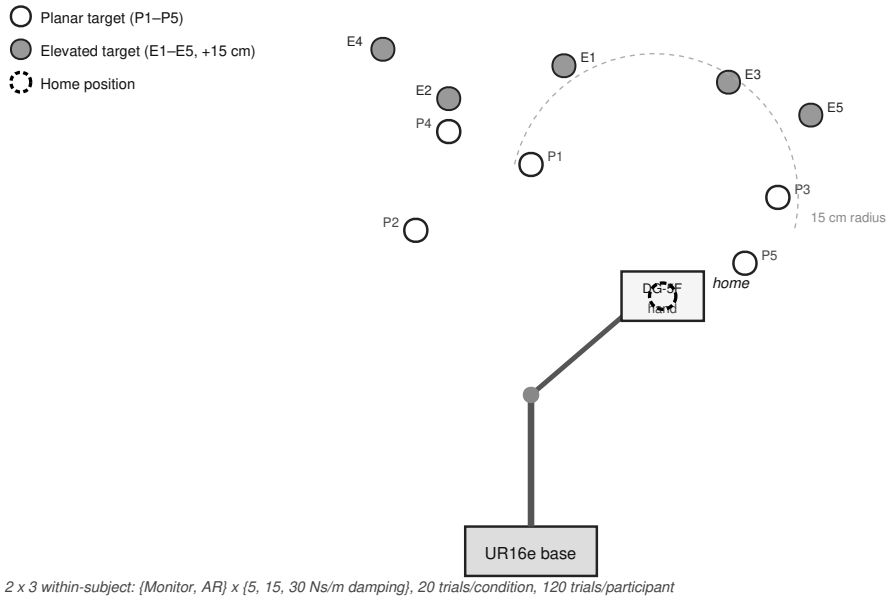


Figure 6.6: Hand-guided reaching task workspace. Ten targets are distributed in the UR16e workspace: five planar (P1–P5) and five elevated at a 15 cm vertical offset (E1–E5). Each participant completes 20 trials per condition across a 2×3 within-subject design (display × damping), yielding 120 trials per participant.

patient grips the hand lightly and the forces are distributed, while an anxious patient tightens the grip and the finger forces spike. A rigid handle cannot distinguish the two. The fingertip force stream in this platform therefore functions as a physical proxy for interaction comfort that conventional handle-based rehabilitation robots cannot capture.

6.7.2 Limitations

Four limitations bound this chapter's claims. First, the platform has been exercised only with healthy adult participants under laboratory conditions. Clinical validation with patients who have proprioceptive deficits is the intended application and is addressed in the future-work agenda of Chapter 14. Second, the HoloLens 2 field of view (52°) limits the AR overlay for large-amplitude reaching movements. Targets are

therefore placed within a 15 cm radius of the home position. Third, the quantitative analysis of the display-modality effect is not reported in this chapter. The chapter contributes the architecture, not the clinical findings. Fourth, the platform supports a single colocated user pair (one patient and one clinician). Distributed multi-user scenarios—a remote therapist guiding an on-site patient with coordinate-aligned virtual overlays across heterogeneous head-mounted displays—require spatial synchronization and role-based visualization–control separation, which Chapter 7 introduces.

6.8 Summary

This chapter contributed two architectural patterns. The multi-rate parallel pipeline (Obj 1) sustained a 500 Hz Cartesian admittance loop with 30 Hz HoloLens gaze, 100 Hz fingertip grip force, and 60 Hz AR rendering concurrently under a single ROS2 clock. The three-layer separation (Obj 2) confined the experimental protocol (display modality, admittance damping) to application-layer configuration alone. A representative single-pilot reaching trial (0 dropped samples, sub-millisecond inter-stream clock offset) confirmed the full pipeline sustains the required data rates concurrently. Multi-participant analysis of the display-modality effect is deferred to the future-work agenda of Chapter 14. These patterns carry forward to the multi-user mixed-reality composition of Chapter 7 and the diffusion-interaction policy architecture of Chapter 12.

Chapter 7

SYSTEM 6: MULTI-USER MIXED REALITY ROBOT-AIDED REHABILITATION

7.1 Introduction

The single-user AR platform of Chapter 6 demonstrated that a 500 Hz admittance loop, eye tracking, and grip-force logging can run concurrently under a single host. The design addresses one patient and one clinician in the same room. Clinical rehabilitation, however, often requires distributed participation: a therapist at one institution guides a patient at another through shoulder exercises on a robot arm while a local assistant retains emergency-stop authority. All three participants view the same virtual guide robot overlaid on the physical robot in real time, and each participant's role—observer, controller, or supervisor—is configurable without code changes. Multi-user MR rehabilitation therefore imposes three capabilities beyond a single-user platform: spatial synchronization across multiple head-mounted displays so that virtual objects appear at the same physical location for every participant, bidirectional robot state sharing so that every participant's view is consistent with the physical robot, and role-based visualization–control separation so that the therapist can adjust exercise parameters while the patient can only follow the guide robot.

Three architectural mechanisms provide these capabilities: (i) a QR-based coordinate co-registration that establishes a shared global frame from each device's independent scan (Section 7.5.1); (ii) an authoritative state replication layer through Mirror Networking SyncVar over KCP transport that synchronizes coordinate frames and robot state across all connected clients on a LAN-local server without cloud relays (Section 7.5.1); and (iii) a visualization–control separation policy that binds

each user's display modality and control authority to a declarative tuple read from the configuration hierarchy of Chapter 9 (Section 7.5.2). Two demonstrations at the end of the chapter exercise the architecture with multiple HoloLens clients on a shared rehabilitation robot. Figure 7.1 illustrates the target deployment: distributed participants share one mixed reality workspace around a real rehabilitation robot, with spatial consistency provided by QR-based co-registration across heterogeneous head-mounted displays.

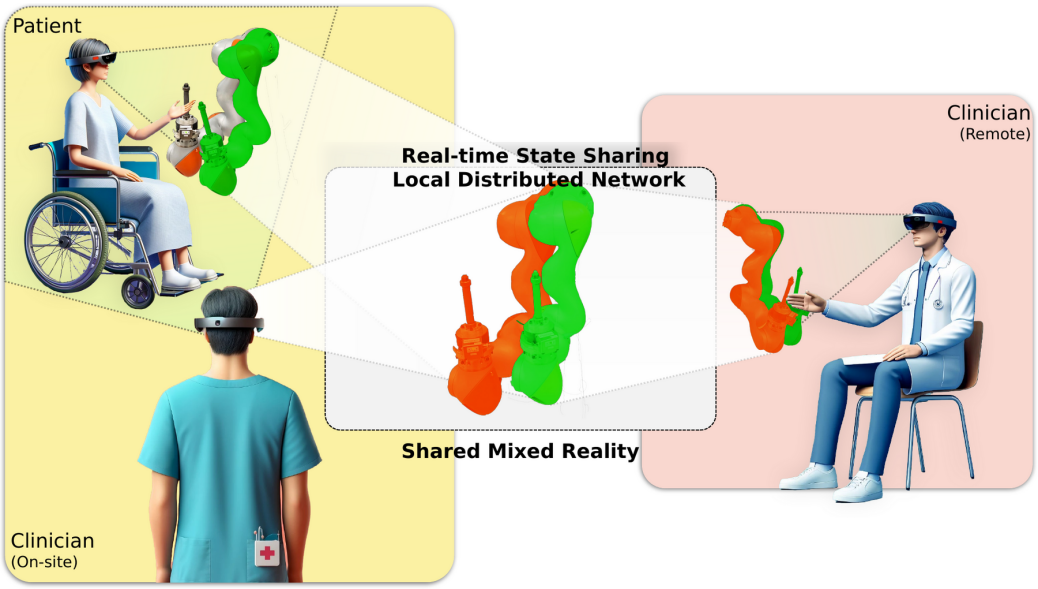


Figure 7.1: Multi-user spatial visualization across sites: distributed participants share one mixed reality workspace around a real rehabilitation robot. The same virtual robot and target overlays are anchored to the physical workspace for every participant through QR-based co-registration, regardless of which head-mounted display is used (HoloLens 2, Meta Quest 3) or whether the participant is on-site or remote.

7.2 Background and Related Work

7.2.1 Multi-User HMD Rehabilitation

Multi-user MR rehabilitation introduces architectural challenges beyond those of single-user systems (Hönig *et al.*, 2015; Quesada and Demiris, 2022). Multiple head-mounted displays must maintain spatial consistency so that virtual objects appear

at the same physical location for every connected user (Wang *et al.*, 2023; Pezzera *et al.*, 2020). Robot state must synchronize across all connected clients with minimal latency to support coherent interaction (Murnane *et al.*, 2021; Kaszuba *et al.*, 2021). Single-user designs constrain real-time patient–clinician collaboration and prevent remote therapy (Szczurek *et al.*, 2023; Williams *et al.*, 2018).

7.2.2 Spatial Synchronization Across Heterogeneous HMDs

Current MR systems for robot-aided rehabilitation report misalignment between user movements and robot actions, gaze shifts induced by visual mismatch, and limited collaboration in single-user paradigms (Garcia Hernandez *et al.*, 2023; Howard and Davis, 2022; Garcia-Hernandez *et al.*, 2024; Ocampo and Tavakoli, 2019). AR overlays digital information onto the real world with limited interaction (Al-Issa *et al.*, 2012; Phan *et al.*, 2022); MR supports two-way real-time interactions between digital objects and the physical environment (Krevelen and Poelman, 2010; Carmigniani *et al.*, 2011). The challenge for distributed deployment is establishing a shared coordinate frame across HMDs with different intrinsic tracking conventions and field-of-view characteristics.

7.2.3 Role-Based Access in Telerehabilitation

Robot-aided rehabilitation provides individualized exercise progression, quantitative performance measurement, and longitudinal tracking with interactive feedback (Giang *et al.*, 2020; Gassert and Dietz, 2018); access is constrained by geographic distance between patients and therapists and by the cost of repeated in-person sessions, which motivates multi-site deployment. Telerehabilitation that combines on-site and remote participants requires role-based access control that differs between co-located and remote users (Higgins *et al.*, 2023; Assis *et al.*, 2016; Brennan *et al.*, 2010): a

remote therapist may need elevated permissions to control the physical robot, but only when an on-site supervisor is present to enforce safety constraints.

7.3 Hardware Platform

The platform integrates Microsoft HoloLens 2 and Meta Quest 3 head-mounted displays through OpenXR, Mirror Networking middleware, the ROS TCP Connector bridge, and a ROS2 Humble-controlled rehabilitation robot under a single shared coordinate frame (Figure 7.2). Mirror Networking (Mirror Networking Contributors, 2024) replicates game-object state across connected clients via KCP transport on a LAN-local authoritative server; this avoids dependence on cloud relays and bounds replication latency to local subnet round-trip time. Unity renders the MR environment on each HMD; ROS2 (Macenski *et al.*, 2022) controls the physical robot through DDS-based middleware; the ROS TCP Connector bridges Unity C# and ROS2 messages over a TCP socket.

Two virtual robots coexist in the MR environment. The virtual twin robot mirrors the physical robot's current joint configuration; the virtual guide robot displays the clinician-prescribed target pose or trajectory. The two-robot scheme separates current state from intended state in the visual field. Figure 7.2 shows the platform components—XR devices, Unity rendering, OpenXR input mapping, Mirror state replication, and ROS2 robot control—and Figure 7.3 shows how patient movement updates each participant's virtual twin robot through ROS, Unity, and Mirror.

Coordinate frame management on the ROS2 side uses the TF2 library, which maintains a tree of spatial relationships between frames. The bridge to Unity's networked multi-user environment is provided by custom synchronization components built within Mirror's state replication framework, which provide bidirectional coordinate frame and robot state synchronization across all connected clients (Section 7.5.1).

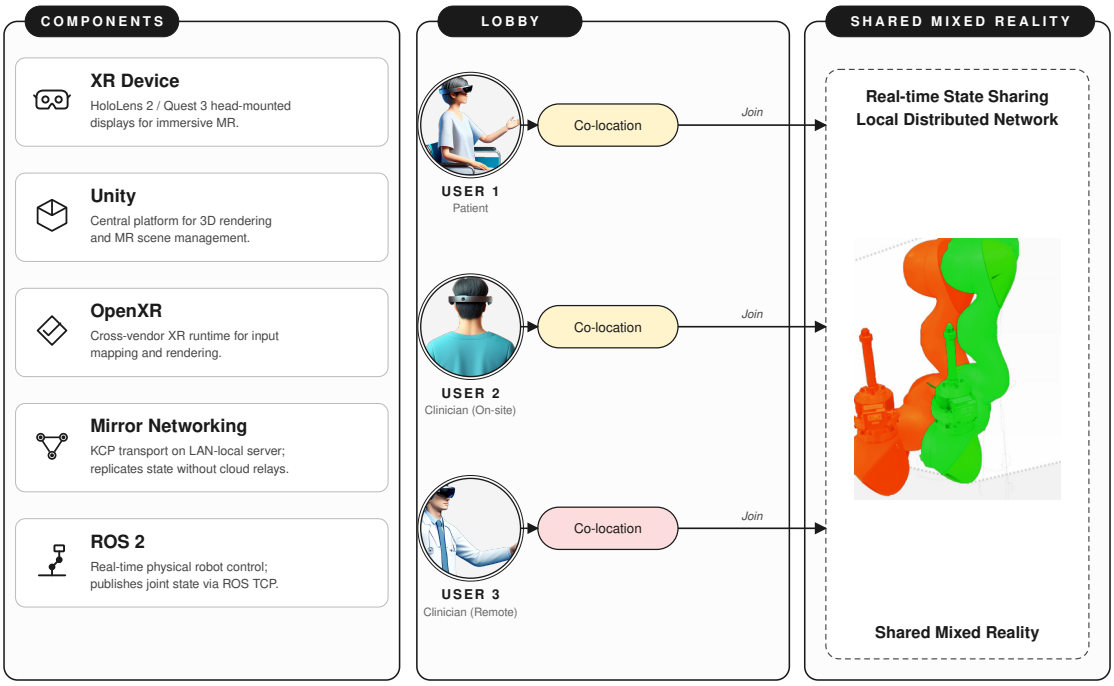


Figure 7.2: System architecture overview. The platform integrates HoloLens 2 and Quest 3 head-mounted displays through OpenXR for cross-vendor input mapping and rendering; Unity renders the MR scene on each HMD; Mirror Networking replicates state across connected clients (the virtual twin robot, the virtual guide robot, the global world coordinate frame, and user avatars) via KCP transport on a LAN-local authoritative server, avoiding cloud relays; ROS2 controls the physical robot and publishes its joint state and coordinate transforms to Unity through the ROS TCP Connector. Each user joins the shared mixed reality environment from the lobby through QR-based co-location.

7.4 Software Requirements

The architecture described in this chapter is required by two downstream workloads: the multi-site deployment of Chapter 9, which reuses the policy tuple, the Mirror Networking authoritative server, and the connection-resolver abstraction; and the integration demonstrations of Chapter 13, which configure the same architecture through different policy instances without modifying the underlying control code. Three architectural requirements follow:

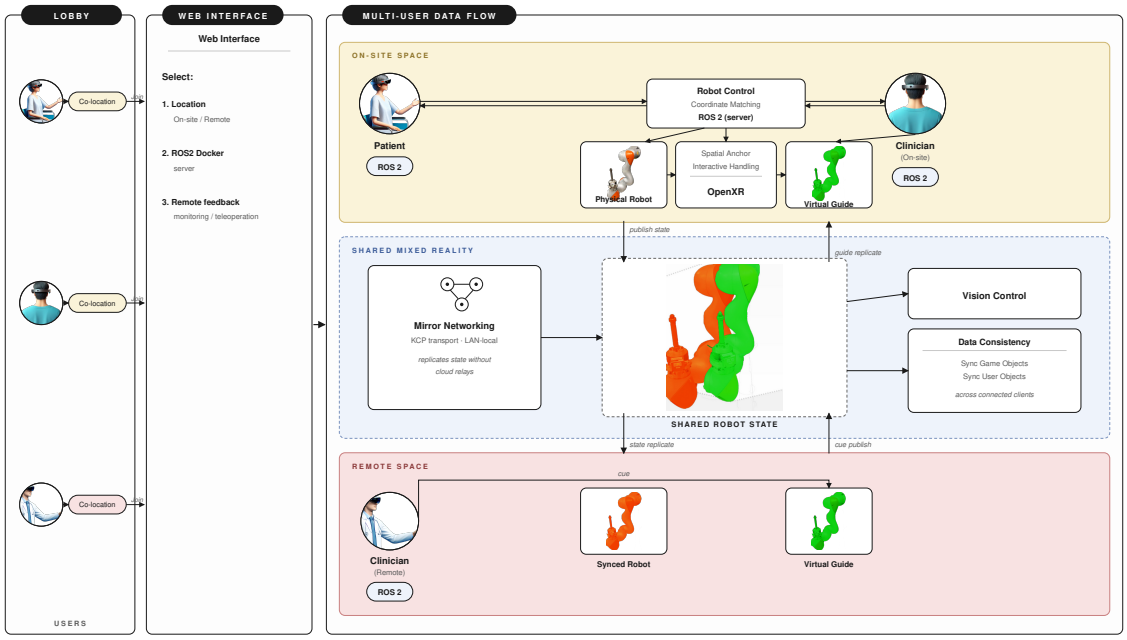


Figure 7.3: Multi-user data flow architecture. Patient movement updates the physical robot via ROS2 and is replicated to each participant's virtual twin robot through Mirror Networking; clinician cues are returned through the same channel for real-time feedback during the session.

1. Authoritative state replication that tolerates a single head-mounted client losing connectivity, so that distributed sessions continue when one HMD drops the network.

2. Visualization–control separation that allows any subset of clients to render the session while a disjoint subset holds control authority, so that role assignments (therapist, patient, observer, supervisor) are expressed through configuration alone.

3. Site-agnostic network layer that resolves hostnames and port assignments from a single declarative file, so that a session started at one site can be moved to another site without touching source code.

Component	Model / Library	Role
Head-mounted display	Microsoft HoloLens 2 (52° FoV, 60 Hz render)	On-site AR overlay; gesture and gaze input
Head-mounted display	Meta Quest 3 (104° FoV, 90 Hz render)	On-site and remote MR participants via OpenXR
Networking middleware	Mirror Networking (KCP transport)	LAN-local authoritative state replication (Mirror Networking Contributors, 2024)
ROS bridge	ROS TCP Connector	Unity–ROS2 message bridge over TCP
Robot middleware	ROS2 Humble (DDS)	Robot control and state publication (Macenski *et al.*, 2022)
Workspace marker	QR code anchor	Per-device co-registration to shared global frame (Costa *et al.*, 2024)

Table 7.1: Hardware and middleware components of the multi-user MR rehabilitation platform.

7.5 Software Architecture

The architecture combines two patterns. QR-anchored coordinate co-registration with authoritative state replication via Mirror Networking SyncVar over LAN-local KCP transport addresses Obj 1. A five-field policy tuple (*see*, *move*, *follow*, *owner*, *mapping*) separates visualization from control authority through declarative configuration, addressing Obj 2. Table 7.2 summarizes.

Objective	Method
Obj 1 (real-time parallel)	QR-anchored hierarchical coordinate unification across heterogeneous HMDs; two-stage initialization (lobby co-registration → MR sync); Mirror SyncVar over KCP, authoritative LAN-local replication
Obj 2 (modular scalable)	Five-field policy tuple (*see* / *move* / *follow* / *owner* / *mapping*); role-based permission matrix resolved at session start

Table 7.2: Architectural contributions of Chapter 7 to the dissertation's two objectives.

7.5.1 Parallel Architecture

Coordinate Co-Registration

The coordinate system consists of three components that together maintain spatial consistency across user locations and devices (Figure 7.4).

A Global Reference Frame is the shared spatial anchor for all participants. Each user wearing a head-mounted display scans a QR marker attached to the robot workspace (Costa *et al.*, 2024); the scan generates a Local User Frame at the marker's detected pose, and the shared Global Frame is loaded at that position. Because the Global Frame is a networked object replicated through the state synchronization layer of Section 7.5.1, every subsequent user who scans the same marker is placed into the same coordinate space. The Robot Base Frame and Robot End-Effector Frame are children of the Global Frame and update in real time as the physical robot moves.

Two-Stage Initialization Process

The two-stage initialization process separates calibration from task execution (Figures 7.5 and 7.6). In Stage 1 (lobby), each user dons the head-mounted display, performs spatial mapping of the room geometry so that virtual objects can be placed

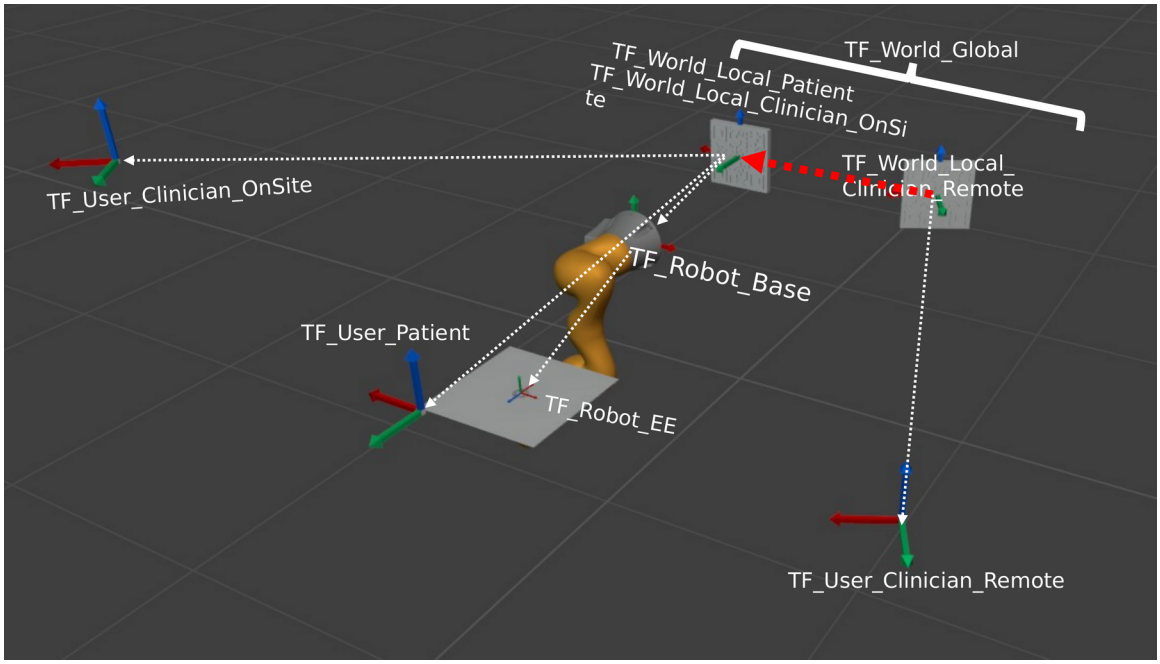


Figure 7.4: Coordinate system establishment via QR code scanning. Each user's local frame is aligned to a shared global frame through independent QR scans, placing all participants into the same spatial reference regardless of physical location.

at correct physical locations, and scans the QR marker to create their local world frame at the marker pose. The first user to scan creates the Global Reference Frame at their local world pose; subsequent users align their User Pose under the same Global Frame, which reassigns the parent of their User Pose to the Global Frame and replicates the ROS TF tree under it. In Stage 2 (mixed reality), the system verifies physical and virtual robot states, loads the rehabilitation sub-task, positions all virtual and physical robots at the Robot Base Frame, and begins synchronized operation. The patient manipulates the physical robot's end-effector through admittance control while the virtual guide robot displays the next target pose; Mirror replicates pose updates so all users observe consistent virtual overlays.

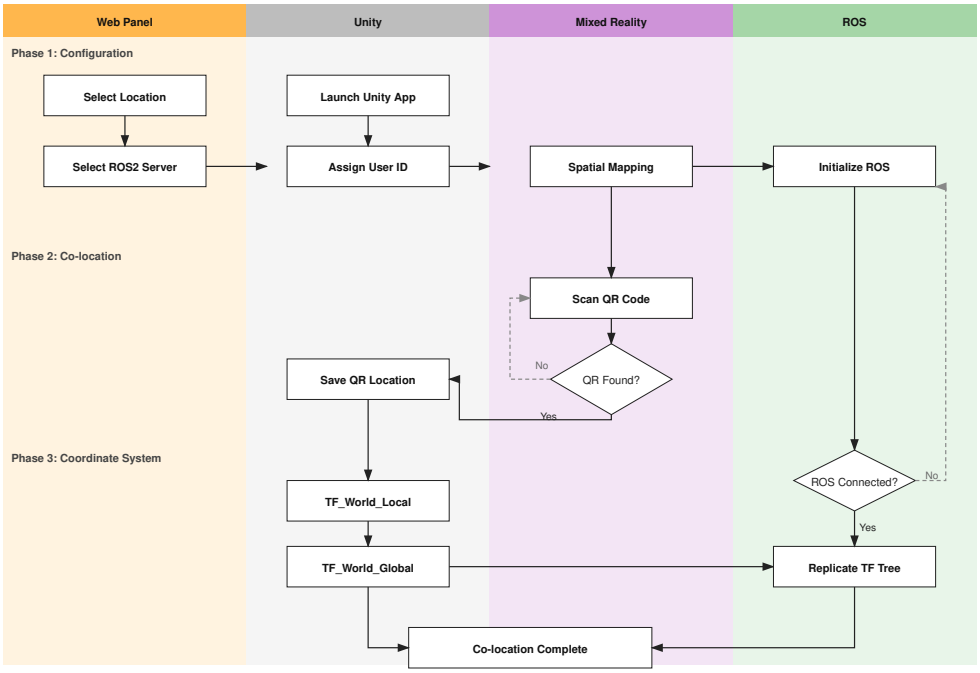


Figure 7.5: Stage 1: Lobby—initialization and co-location. Users scan a QR marker to establish a shared global coordinate frame before entering the mixed reality task.

Distributed State Synchronization

Multi-user MR requires that all connected clients maintain consistent views of two categories of shared state: coordinate frames that define spatial relationships between virtual and physical elements, and robot state that reflects current joint positions and visualization modes. Figure 7.7 shows the two synchronization components and the LAN-local transport that carries them.

The synchronization layer is built on Mirror Networking (Mirror Networking Contributors, 2024), a LAN-direct authoritative networking framework for Unity that replicates object state across connected clients via KCP transport. Two domain-specific components extend Mirror for this platform. NetworkTF objects represent coordinate frames with SyncVar-annotated fields; Mirror's authoritative server replicates state changes to all clients, and hook callbacks trigger local updates only when state

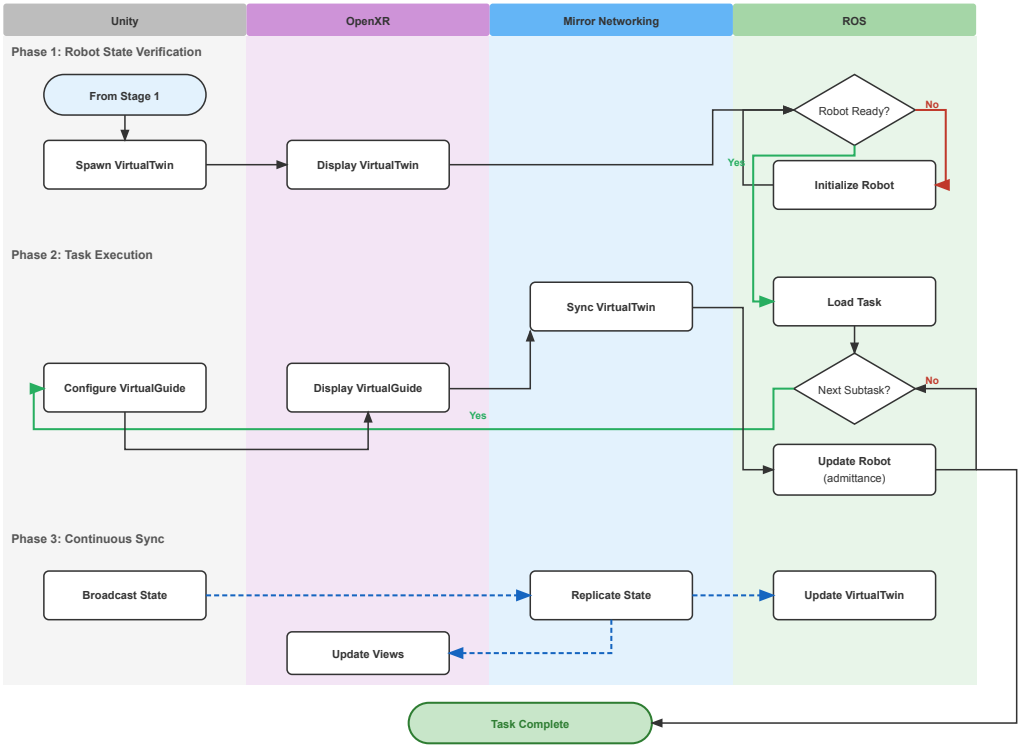


Figure 7.6: Stage 2: Mixed reality task execution. The patient manipulates the physical robot through admittance control while all participants observe consistent virtual overlays.

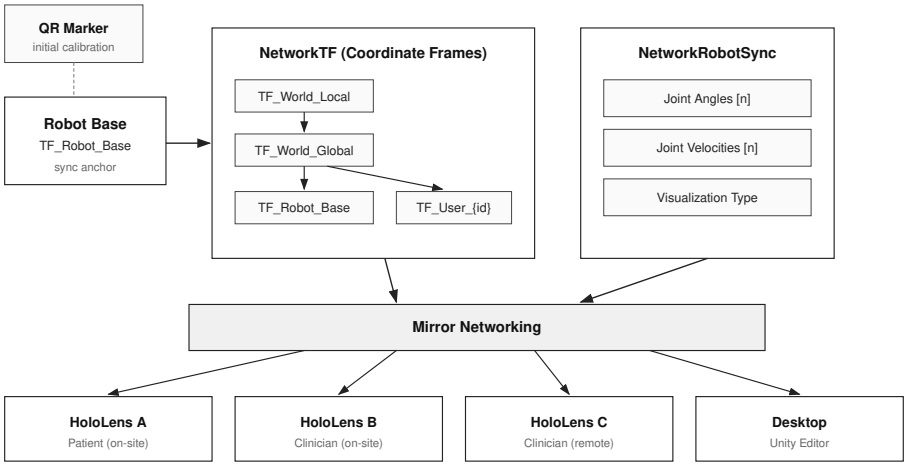


Figure 7.7: Distributed synchronization architecture. NetworkTF replicates coordinate frames and NetworkRobotSync replicates robot joint state through Mirror Networking's authoritative server over LAN-local KCP transport.

actually changes. A teleport detection mechanism flags discontinuous position changes above a configurable threshold so that the client snaps rather than interpolates. NetworkRobotSync objects maintain arrays of joint values that update as the physical robot moves; authority-based synchronization ensures that only the client with direct robot connection can modify the networked state, preventing conflicting updates from multiple sources.

The visualization type field within NetworkRobotSync allows dynamic switching between Physical, Virtual Twin, and VirtualGuide modes. When a client changes its visualization preference, the networked state updates and all clients render the robot appropriately for their local context. The per-client rendering choice implements the visualization–control separation described in the next section.

7.5.2 Modular Architecture

Visualization–Control Separation

The architecture separates what each user sees from what each user can do, allowing each to be configured independently per participant (Figure 7.8).

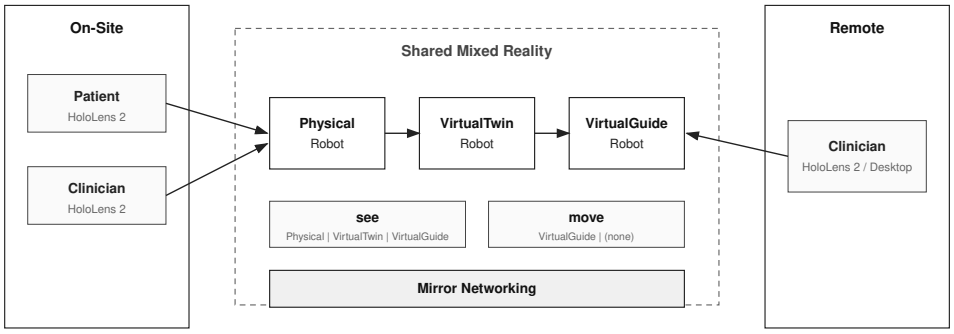


Figure 7.8: Visualization and control separation. Three independently selectable layers—user location, robot visualization mode, and control mode—combine to support role assignments that coupled architectures cannot express.

Three independently selectable layers define each participant's role:

1. **User Location**: physically present near the robot or accessing remotely.
2. **Robot Visualization**: Physical mode renders AR overlays on the actual hardware, Virtual Twin mode mirrors the physical robot's state as a digital twin, and VirtualGuide mode shows the clinician-prescribed target trajectory.
3. **Control Mode**: Physical mode provides direct robot control, Virtual mode provides simulation, and None mode allows observation without control.

These three layers combine to support scenarios that coupled architectures cannot express. A remote therapist can observe through Virtual Twin while controlling the Physical robot, provided an on-site supervisor is present. A trainee can practice with Virtual control while viewing the Virtual Twin. A researcher can run simulation experiments while an on-site operator maintains Physical control as backup. The per-participant binding is expressed as a declarative tuple in the configuration hierarchy of Chapter 9; changing a participant's role requires editing a configuration value, not modifying code. The tuple is composed of five fields: visualization (*see*), control authority (*move*), camera-following (*follow*), per-participant ownership (*owner*), and the QR anchor that defines the entity's coordinate frame (*mapping*). The resolved per-role assignments of these fields are summarized in Figure 7.9.

The permission matrix resolves safety constraints into per-user defaults: on-site users access all visualization and control modes; remote users observe through Virtual Twin without physical robot control; users with an elevated permission flag can control the physical robot remotely when an on-site supervisor is present, supporting telerehabilitation scenarios in which a remote therapist guides exercises while local staff ensure safety (Brennan *et al.*, 2010).

owner role taxonomy	see visualization observed	move actively driven	follow passively followed	mapping QR / device	control resolved authority
therapist	VirtualGuide	VirtualGuide	(none)	phantom	Real
patient on-site	VirtualTwin	(none)	VirtualTwin	Q9	Real admittance
remote_therapist	VirtualTwin	(none)	VirtualTwin	Q9	Fake guide only

Figure 7.9: Permission matrix showing allowed combinations of user location, visualization mode, and control mode based on role and safety constraints. The matrix encodes the per-user defaults that the policy resolver returns at session start; on-site users default to full access, remote users default to Virtual Twin observation, and elevated permissions extend remote access only when an on-site supervisor is present.

Policy-Based Configuration

Three declarative YAML policy files manage configuration: the robot visualization strategy policy selects available visualization modes based on user location; the access control policy implements role-based permissions (administrator, operator, observer); and the connection policy specifies network routing per platform and location. Gazebo simulation (Koenig and Howard, 2004) provides physics-accurate testing without physical hardware, and the same policy mechanism selects between physical and simulated execution paths.

Network and Authentication Infrastructure

The network and authentication infrastructure for multi-site, multi-platform deployment is documented in Appendix D, Section D.7. The infrastructure includes the `ConnectionResolver` that maps abstract site/robot identifiers to concrete VPN-mesh peer IPs across local, VPN, SSH-tunnel, and cloud-proxy paths, an SSH gateway with HoloLens device discovery and VPN hostname resolution, JWT-based device authentication with optional TOTP, and Mirror webhook integration for browser-side monitoring. State replication uses Mirror with the LAN-local KCP transport rather

than a cloud-based service, because field deployments at participating laboratories cannot rely on outbound traffic to an external cloud relay. The architectural property exercised by this chapter is platform-aware routing under restricted-OS clients (HoloLens) without compromising the visualization–control separation of Section 7.5.2.

7.6 Results

Two demonstrations exercise the multi-user architecture. Both are preliminary, intended to validate the architecture's operational characteristics rather than to produce clinical outcome measures.

7.6.1 Demonstration 1: Multi-User Session Monitoring

Multiple users wearing HoloLens 2 devices join a shared MR rehabilitation session, and a monitoring interface captures each user's perspective simultaneously so that supervisors can observe how participants perceive the shared virtual environment (Figure 7.10). The demonstration validates three capabilities: simultaneous capture of multiple HoloLens views, spatial consistency across user perspectives through the co-registration of Section 7.5.1, and session supervision without requiring a supervising observer to wear a head-mounted display.

A single-perspective view of the same alignment is shown in Figure 7.11.

7.6.2 Demonstration 2: Multi-User Robot Spawn and Share

Two users—an on-site clinician and a patient—enter the MR environment through the two-stage initialization process. Each user scans the QR marker to establish their local frame; after co-location, both observe the same virtual robot positioned at the physical robot's location, and the clinician spawns a virtual guide robot to demonstrate target trajectories (Figure 7.12). The demonstration validates QR-based

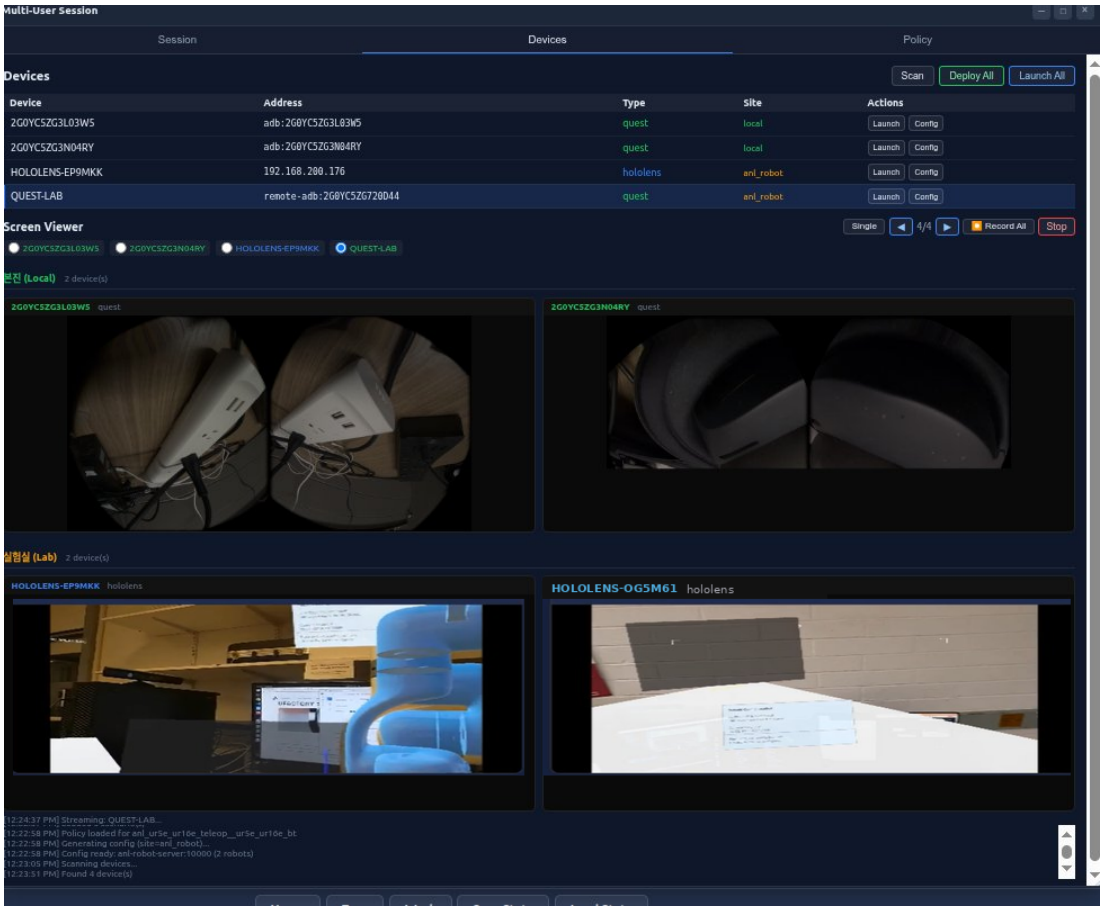


Figure 7.10: Multi-user monitoring interface showing simultaneous first-person views from each connected HoloLens 2 client during a shared mixed reality session. The supervisor panel renders every client's perspective side by side without requiring the supervisor to wear a head-mounted display, and confirms that each participant sees the same virtual robot anchored to the same physical workspace.

co-location, coordinate frame synchronization through the authoritative server, and shared robot visualization at consistent physical locations for every user.

7.7 Discussion

7.7.1 Rationale for QR-Based Co-Registration

QR-based co-registration produces a shared Global Reference Frame from independent device-side scans without a centralized calibration server. Each device scans the same physical marker to establish its local world frame, and the first user to

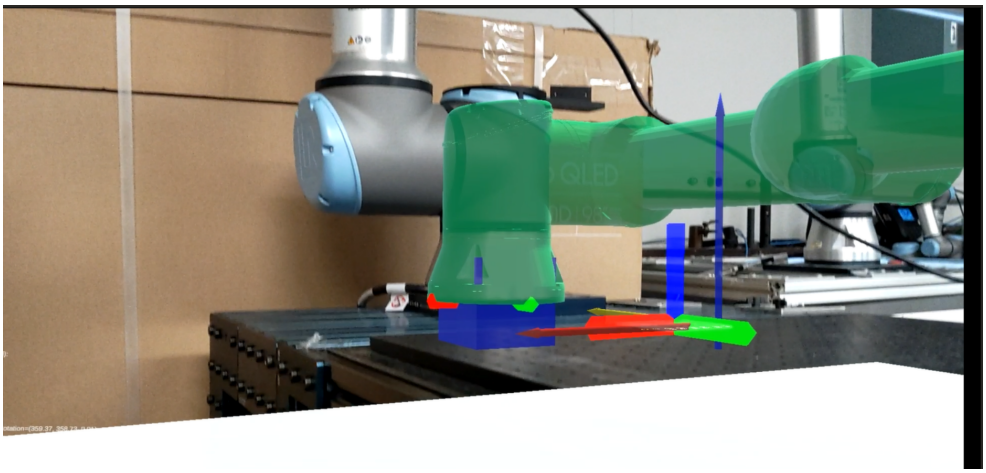


Figure 7.11: Co-location demonstration viewed through one user's HoloLens 2: the green virtual UR16e is rendered at the same world-frame position as the real arm, with the QR-anchored frame axes (red/green/blue) shown at the work surface. The same virtual robot appears at the same physical location for every participant in the multi-user session, regardless of head-mounted display (HoloLens 2, Meta Quest 3) or user location (on-site, remote), through the authoritative state replication and peer-to-peer VPN bridge described in Section 7.5.1.

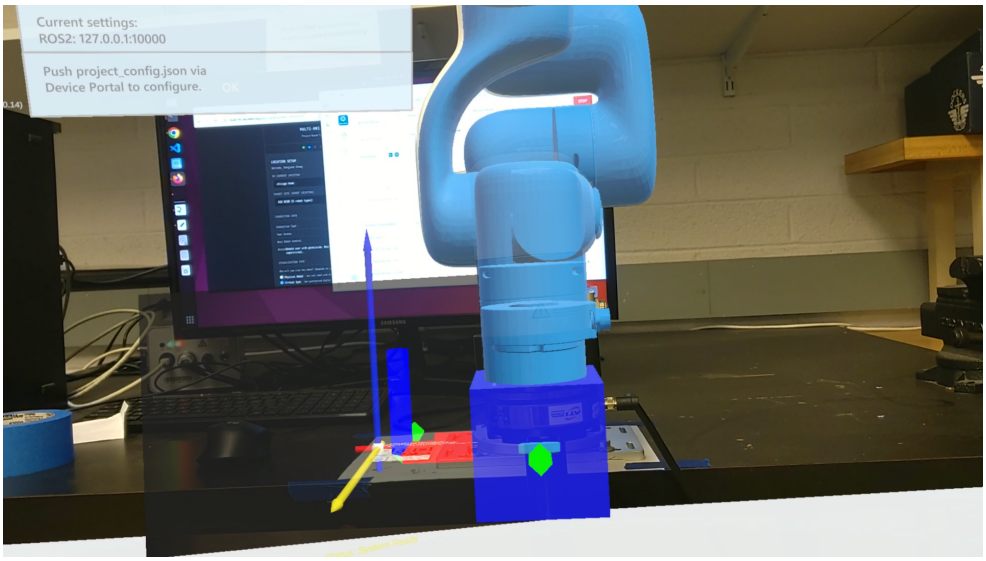


Figure 7.12: HoloLens first-person view showing co-located virtual robot overlaid on the physical workspace after QR-based co-registration. The virtual robot's joints align with the physical robot's pose because both share the Global Reference Frame established by the QR scan; the alignment persists as the user moves through the workspace, evidencing the spatial consistency property that the co-registration mechanism is designed to provide.

scan creates the Global Frame at their pose; subsequent users align under the same Global Frame through Mirror Networking's authoritative replication of the networked TF tree. The mechanism trades one calibration-server dependency for one shared physical marker, which removes the failure mode of a centralized calibration service unreachable across institutional boundaries.

7.7.2 Rationale for Authoritative State Replication

Mirror Networking's authoritative server prevents conflicting state updates across multiple clients. Only the client with direct robot connection holds authority over robot state; other clients receive updates as observers. The visualization–control separation expresses clinical roles (therapist, patient, observer, trainee) as five-field policy tuples rather than as code paths, so the same compiled architecture serves a co-located therapy session, a remote-supervision session, and a training session through configuration alone.

7.7.3 Limitations

The initial implementation relied on the Mixed Reality Toolkit (MRTK) (Microsoft, 2024) for interaction components. MRTK targets the Microsoft HoloLens family and does not support the Meta Quest platform. The MRTK dependency was subsequently replaced with an OpenXR-native implementation to support both HoloLens 2 and Quest 3 from a single codebase; the migration and its validation are described in Chapter 13. Quantitative validation of the architecture is deferred to the multi-user study scheduled in Chapter 14. QR-based co-registration is to be characterized through inter-user placement variance across HoloLens 2 and Meta Quest 3 clients at a 1 m workspace distance, together with the global-frame drift over a 30-minute session. Mirror Networking SyncVar replication is to be characterized through server-

to-client latency and per-client state divergence under LAN-local KCP transport. Real clinical-site network access is deferred to an Institutional Review Board-approved deployment; representative telerehabilitation network profiles from the literature back the future-work entry in Table 14.2.

7.8 Summary

This chapter contributed two architectural patterns. The QR-anchored coordinate co-registration with authoritative state replication via Mirror Networking SyncVar over LAN-local KCP transport (Obj 1) sustained spatial consistency and robot-state synchronization across heterogeneous head-mounted displays without cloud relays. The five-field policy tuple (*see*, *move*, *follow*, *owner*, *mapping*) (Obj 2) expressed clinical role assignments through declarative configuration. Two preliminary demonstrations exercised multi-user session monitoring and shared robot visualization on a LAN-local Mirror server. These patterns carry forward to Chapter 13 (integration demonstrations across heterogeneous robot platforms) and Chapter 9 (multi-site deployment of the policy tuple, the connection resolver, and the SSH gateway).

Chapter 8

SYSTEM 7: MIXED REALITY ROBOTIC PLATFORM FOR AUTONOMOUS MATERIALS SYNTHESIS

8.1 Introduction

Accelerating the discovery of new materials—catalysts, battery electrodes, photovoltaics—is a critical challenge in energy research, and high-throughput experimentation (HTE) addresses this bottleneck by parallelizing synthesis and characterization across modular lab equipment (Hung *et al.*, 2024). The Digital Modular Materials Robotic Platform (DMMRP) (Lee *et al.*, 2024b) integrates a Sawyer collaborative arm, modular lab equipment (stirrer, injector, scale), and a Unity-based mixed reality interface into a single ROS-based system at Argonne National Laboratory; the same platform also supports Fiber Bragg Grating (FBG) robotic skin force sensing for autonomous obstacle avoidance and admittance-controlled object manipulation. The software architecture combines four components: Robot Operating System (ROS) (Quigley *et al.*, 2009) communication on a single master with the rosbridge protocol linking Linux and Windows hosts, Unity (Unity Technologies, 2024) virtual reality (VR) and augmented reality (AR) integration for the mixed reality client, SMACH-based task sequencing (Bohren and Cousins, 2011), and YAML task configuration that later evolved into the declarative configuration hierarchy of Chapter 9.

This chapter presents the DMMRP and the robotic skin work under a single software architecture. Section 8.5 describes the four-component architecture; Section 8.6.1 reports the task-parameterized Gaussian Mixture Model trajectory synthesis (Calinon,

2016) used as the programming-by-demonstration mechanism within the architecture; Section 8.6.2 reports the robotic skin force-sensing application validated through autonomous obstacle avoidance and admittance-controlled manipulation (Lee *et al.*, 2024b). The DMMRP's monolithic architecture could not accommodate a second robot without code changes, and the robotic skin work required separate software integration despite sharing the same Sawyer hardware; these limitations motivated the parameterized multi-robot platform of Chapter 9.

8.2 Background and Related Work

8.2.1 High-Throughput Experimentation in Materials Synthesis

Traditional materials synthesis involves manual dispensing, mixing, and characterization of chemical compositions one at a time, a process too slow to explore the vast compositional spaces required for next-generation energy technologies (Hung *et al.*, 2024). High-throughput experimentation (HTE) addresses this bottleneck by parallelizing synthesis and characterization, enabling hundreds of compositions to be screened per day. At national laboratories, HTE workflows for catalyst screening involve sequential dispensing, stirring, shaking, and characterization steps across modular lab equipment (Hung *et al.*, 2024).

8.2.2 Robotic Lab Automation and Mixed Reality Interfaces

Automating these HTE workflows with robotic platforms has been pursued by multiple groups, but existing laboratory automation systems are typically designed for a single fixed workflow: changing the synthesis protocol or adding new equipment requires reprogramming the entire system. Domain scientists who design these experiments are not software engineers; every protocol change therefore requires developer intervention.

Mixed reality (MR) interfaces offer a potential solution by allowing scientists to program and monitor synthesis workflows through spatial interaction rather than text-based scripting—pointing at physical equipment to define waypoints rather than editing coordinate files.

8.3 Hardware Platform

The DMMRP was built around the Sawyer collaborative robot (Rethink Robotics, USA), selected for its precision control capability and safe physical interaction with humans. A custom gripper was developed for handling dispensers, vials, and vial plates used in catalyst synthesis. Arduino Uno R3 (Arduino, Italy) board controllers managed peripheral lab equipment (shaker, injector, scale) through ROS nodes via serial communication. Figure 8.1 shows the physical prototype and the AR overlay.

The sensor suite included an HTC Vive (HTC Corporation, Taiwan) system for position and orientation tracking, with seven tracked points per hand (five fingertips, hand center, and wrist) published as ROS topics. A Phantom Omni haptic device provided 6-DoF force feedback up to 3.3 N for the operator.

8.4 Software Requirements

The high-throughput experimentation workflows of Section 8.6 impose four software requirements that single-fixed-workflow lab automation cannot satisfy:

1. Real-time concurrent sensing/actuation across heterogeneous components (Sawyer arm, modular lab equipment via Arduino, HTC Vive position tracking, depth camera, PHANToM Omni haptic) on a single ROS master with rosbridge linking Linux and Windows hosts, so that VR/AR programming-by-demonstration can be transferred to physical synthesis operation in real time.

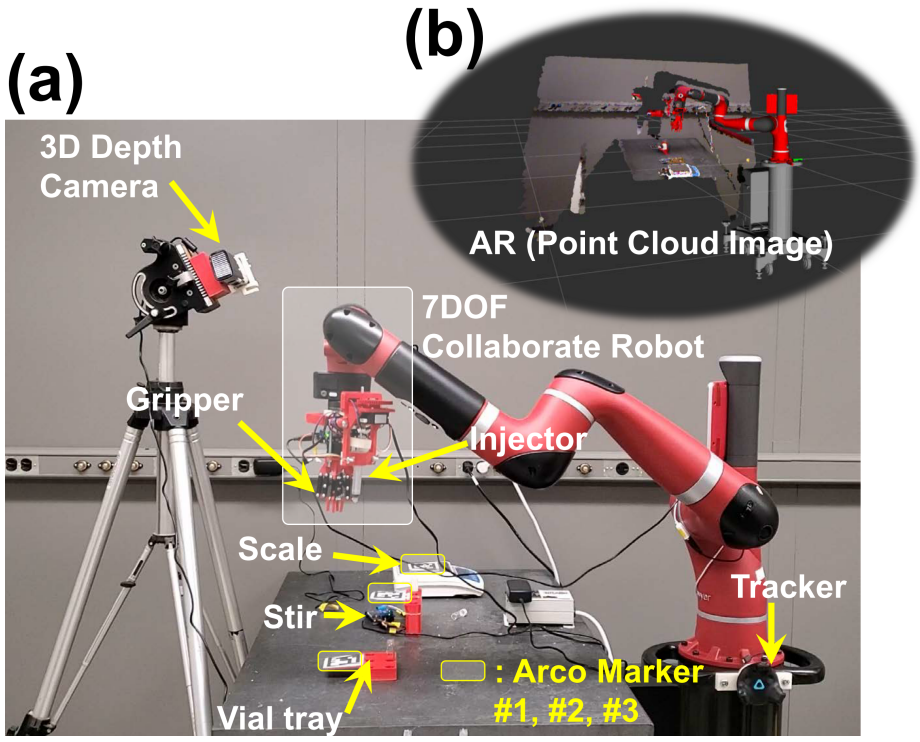


Figure 8.1: Physical prototype of the DMMRP showing (a) modular lab equipment (stirrer, injector, gripper, scale) and (b) VR/AR integrated real-time 3D reconstruction for the MR environment. The yellow "Arco Marker" annotations in panel (a) refer to ArUco fiducial markers (Garrido-Jurado *et al.*, 2014); the spelling in the photographic overlay reflects the original prototype labels.

2. SMACH-based hierarchical task sequencing with YAML task configuration, so that scientists can define synthesis protocols (dispensing, stirring, characterization) as sequences of typed operations through a GUI without modifying source code.

3. Fiducial-marker (ArUco) coordinate registration that binds the virtual laboratory to the physical workspace through 6-DoF pose estimation, so that the MR client can be moved between Sawyer workstations without recompilation.

4. Programming-by-demonstration mechanism that classifies hand movements into macro (arm-level) and micro (finger-level) categories and synthesizes smooth trajectory primitives from a small number of demonstrations, so that experimen-

tal skills can be transferred to the robot through human demonstration rather than text-based scripting.

8.5 Software Architecture

The architecture combines two patterns: ROS-based concurrent sensing/actuation across the Sawyer collaborative arm, modular lab equipment, Unity client, and PHANToM Omni haptic device (addressing Obj 1) and a SMACH state machine with YAML task configuration that decouples scientific workflows from source code (addressing Obj 2). Table 8.1 summarizes.

Objective	**Method**
Obj 1 (real-time parallel)	ROS concurrent sensing/actuation across Sawyer, lab equipment, Unity client
Obj 2 (modular scalable)	SMACH state machine + YAML task configuration

Table 8.1: Architectural contributions of Chapter 8 to the dissertation's two objectives.

8.5.1 Parallel Architecture

The software architecture was built on a single ROS master with the rosbridge protocol (Crick *et al.*, 2017) providing the communication bridge between the Unity VR/AR engine running on Windows and the ROS software stack on Linux. The bridge enabled the exchange of position data, equipment status, and haptic feedback in real time between the two operating systems. Each hardware component (Sawyer controller, ArUco tracker, Arduino nodes, HTC Vive publisher) ran as a ROS node within this single-master architecture. The full system topology is shown in Figure 8.2.

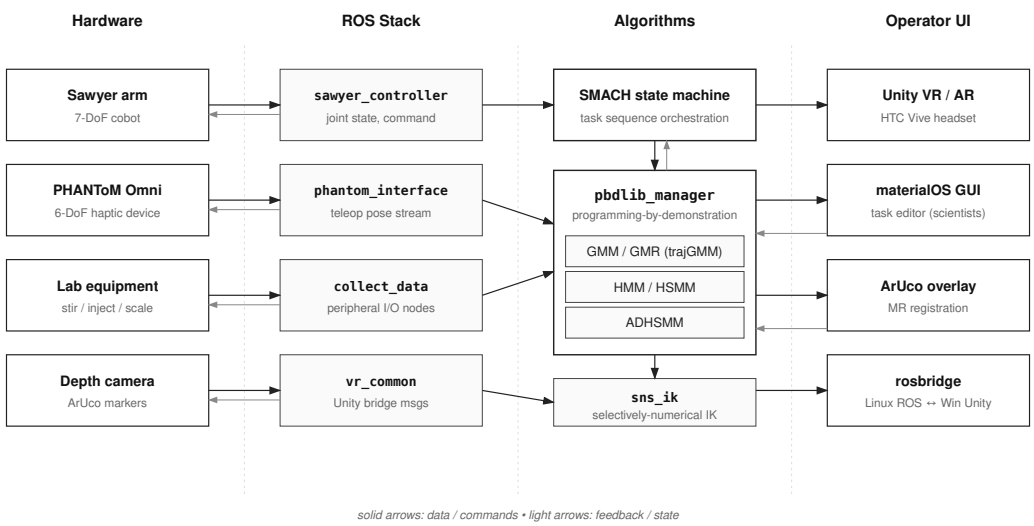


Figure 8.2: DMMRP system architecture. The Sawyer arm, PHANToM Omni haptic device, modular lab equipment, and depth camera are integrated through dedicated ROS packages (`sawyer_controller`, `phantom_interface`, `collect_data`, `vr_common`) that publish to a SMACH state machine and to the `pbdlib_manager` programming-by-demonstration library, which exposes GMM/GMR (trajGMM), HMM/HSMM, and ADHSMM primitives behind a common interface. Joint commands flow through `sns_ik`; the operator interacts through Unity VR/AR with HTC Vive, the materialOS GUI, ArUco-anchored MR overlays, and the rosbridge link between the Linux ROS host and the Windows Unity client.

8.5.2 Modular Architecture

Mixed Reality Environment

The mixed reality environment was implemented as three prototyping layers that closed the perception-action loop incrementally:

1. **VR Prototype**: A Unity-based 3D environment rendered models of the robot, lab equipment, and chemical elements. HTC Vive headsets and controllers exchanged position data with Unity for immersive visualization and planning, even when the physical robot was not connected.

2. **Physical Prototype**: The Sawyer robot and modular lab equipment performed the actual synthesis operations. ArUco fiducial markers (Garrido-Jurado

et al., 2014) attached to vial plates provided six-degree-of-freedom (6-DoF) pose estimation through a depth camera, providing real-time object tracking.

3. **MR Prototype**: The VR and physical environments were linked through coordinate matching between the ArUco marker poses and their Unity counterparts. This closed the loop between simulation and implementation.

The HTC Vive hand glove tracking points and the physical-to-virtual hand correspondence are shown in Figure 8.3.

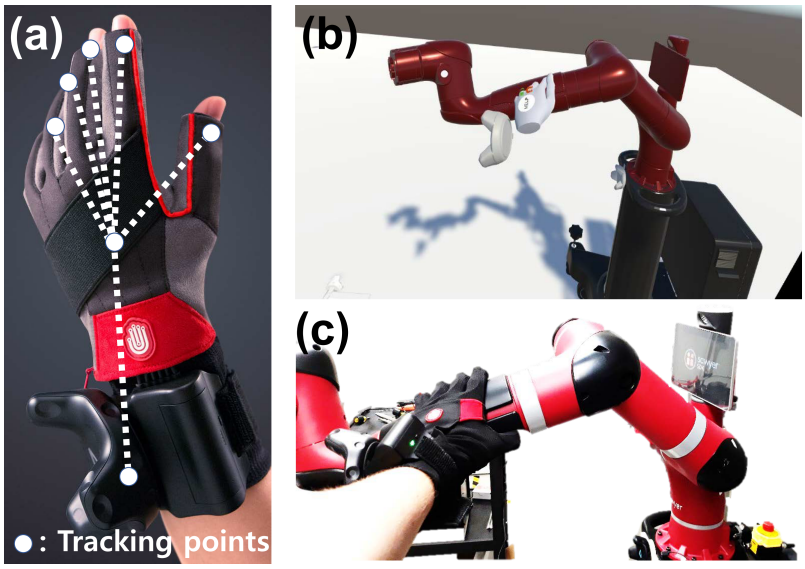


Figure 8.3: (a) Tracked points on the HTC Vive hand glove. (b–c) Coordinate matching between the physical hand and its VR representation in the Unity environment.

State Machine and Task Management

A ROS SMACH state machine managed the task sequence during automated operation. Each state corresponded to a specific synthesis step (dispensing, stirring, shaking, centrifuging), and state variables were shared across nodes through the SMACH API. A GUI application ("materialOS") allowed scientists to define task sequences and

parameters without modifying source code; the GUI and the underlying state-machine flow are shown in Figure 8.4.

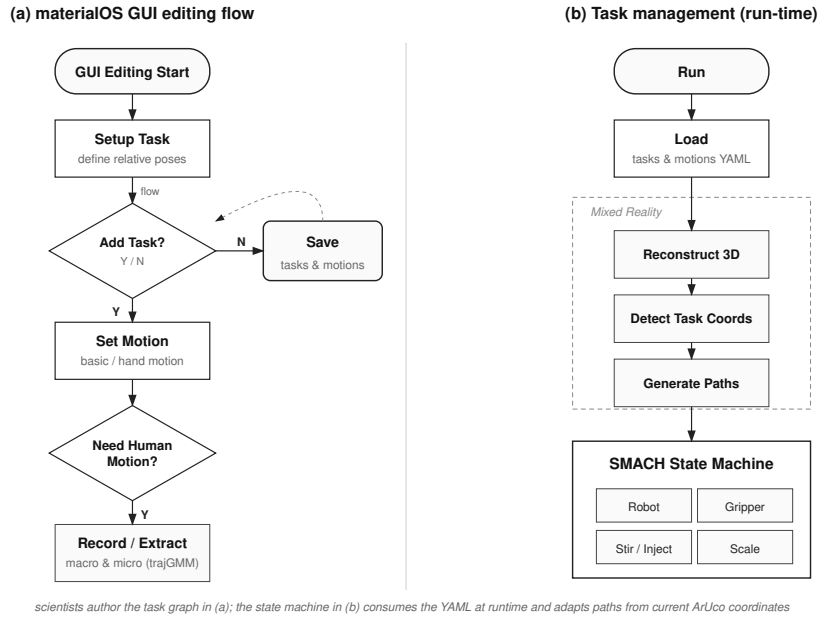


Figure 8.4: (a) The materialOS GUI for defining task sequences. (b) Internal flow of the task management system showing state machine transitions.

Task parameters were stored in YAML files specifying relative poses, task coordinates (defined by ArUco marker centers), and motion types. When new task coordinates were registered at runtime through ArUco detection, the system adapted the robot's motion paths accordingly. This YAML-based task configuration represented an early form of the declarative parameter management that the next chapter generalizes to the full multi-robot system.

8.6.1 Programming by Demonstration

The DMMRP adopted a programming by demonstration (PbD) approach using a task-parameterized Gaussian Mixture Model (TP-GMM) (Calinon, 2016) to transfer human experimental skills to the robot.

Motion Classification

Hand movements were classified into two categories based on tracking granularity:

- **Macro movements**: Arm-level trajectories captured from the wrist and palm data. These were used for robot path imitation, such as moving objects between workstations.
- **Micro movements**: Finger-level gestures captured from the five fingertip trackers. These were classified into task-specific commands (gripping, injecting, stirring, weighing).

Representative trajectory clusters for each category are shown in Figure 8.5.

Trajectory Generation

A trajectory-GMM (trajGMM) algorithm synthesized smooth motion primitives from five demonstrations per task. The implementation lived in the `pbdlib_manager` ROS package, which wrapped a programming-by-demonstration library exposing GMM, GMR, HMM, HSMM, and ADHSMM primitives behind a common interface. The trajGMM variant used here encoded position, velocity, and acceleration as GMM components, finding the trajectory $\hat{x}$ that maximized the likelihood of the demonstrated movements:

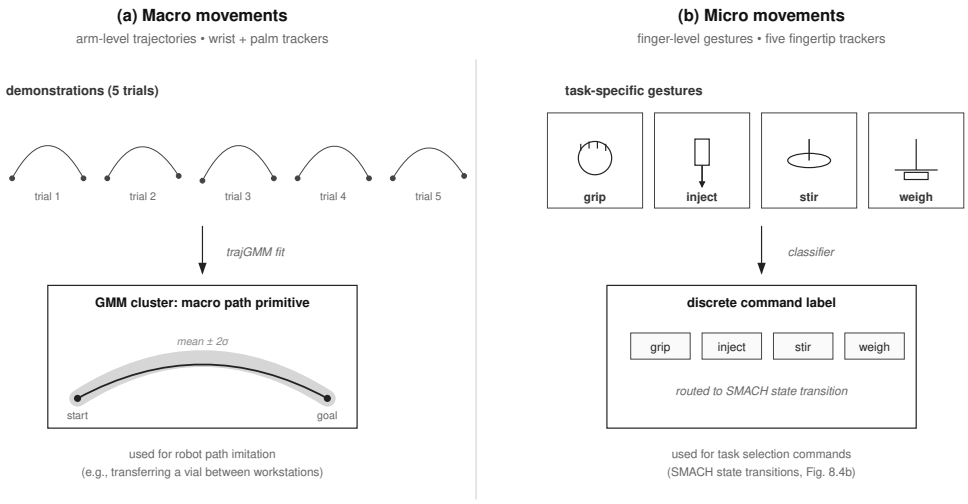


Figure 8.5: Two-tier movement classification. (a) Macro movements: five demonstrations of arm-level trajectories (wrist + palm trackers) are fitted by a trajGMM into a single path primitive, used for robot path imitation. (b) Micro movements: finger-level gestures (five fingertip trackers) are classified into discrete task-specific labels (grip, inject, stir, weigh) that are routed to SMACH state transitions.

$$\hat{x} = (\Phi^T \Sigma_s^{-1} \Phi)^{-1} \Phi^T \Sigma_s^{-1} \mu_s \tag{8.1}$$

where Φ maps the position vector x to the concatenated position-velocity-acceleration observation $\zeta = \Phi x$, and μ_s, Σ_s are the mean and covariance of the GMM states. The resulting motion primitives were smooth even with a small number of Gaussian distributions and adapted to new task coordinates through the task-parameterized framework. The four PbD stages (observation, task-parameterized adaptation, reproduction, GMR retrieval) are shown in Figure 8.6.

8.6.2 Robotic Skin for Force Sensing

Fiber Bragg Grating (FBG) sensors integrated into the same Sawyer infrastructure provided force and contact localization for autonomous obstacle avoidance and admittance-controlled object manipulation (Lee *et al.*, 2024b).

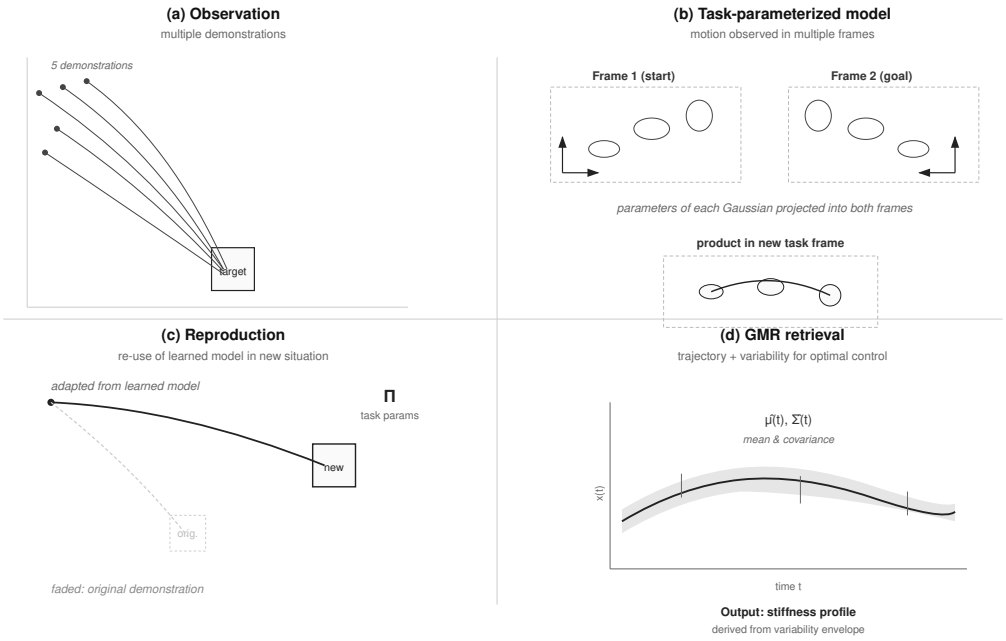


Figure 8.6: Programming-by-demonstration pipeline. (a) Multiple demonstrations are recorded against a target. (b) Each demonstration is encoded as Gaussians projected into multiple task frames (start and goal); the product of frame-conditional Gaussians yields a task-parameterized model. (c) The learned model is re-applied to a new task configuration through the task parameters Π, producing an adapted trajectory. (d) Gaussian Mixture Regression retrieves a mean trajectory $\hat{\mu}(t)$ and covariance $\hat{\Sigma}(t)$ over time; the variability envelope informs the stiffness profile passed to the controller.

Skin Module Design

The skin used FBG sensors embedded in a custom triangular beam structure, with each module providing three degrees of freedom in sensing: contact force magnitude and two-axis contact localization. Characterization showed resolutions of 1.45 N for force estimation and 1.85 mm horizontal / 1.91 mm vertical for contact localization.

Applications

The software architecture used ROS topics for real-time streaming of FBG sensor data to both a haptic feedback loop (for remote manipulation) and an admittance controller (for autonomous operation). Two application areas were demonstrated:

- **Remote manipulation**: An operator controlled the Sawyer through a haptic device while receiving force feedback from the skin. The operator performed shape tracing of unknown objects and path finding through multi-obstacle environments using only contact information.
- **Autonomous control**: The robot navigated obstacle fields and manipulated free-moving objects autonomously. For path finding, the robot followed straight-line paths between waypoints and autonomously avoided obstacles based on real-time contact detection. For object manipulation, the robot maintained a target contact force of 6 N (achieved RMSE of 0.64 N) while rotating an object around a fixed reference.

While the DMMRP focused on MR-mediated task programming, the robotic skin addressed physical sensing for safe interaction with the environment. The DMMRP used VR/AR for task programming while the robotic skin used real-time FBG contact sensing—both through the same Sawyer and ROS stack, but with no shared parameter management.

8.7 Discussion

8.7.1 Contributions

Three DMMRP elements reappear in the multi-robot platform of Chapter 9: the YAML-based task parameter files introduce declarative configuration for scientific

workflows, the ROS SMACH state machine structures the task sequencing layer, and the rosbridge communication between Unity and ROS establishes the cross-platform MR-robot integration pattern reused in all subsequent systems.

8.7.2 Limitations

Both the DMMRP and the robotic skin experiments shared the same architectural limitation. The DMMRP used a monolithic ROS architecture with a single master, where all nodes—`vr_common` (Unity bridge), `sawyer_controller`, `phantom_interface` (haptic), `pbdlib_manager` (PbD module), and the SMACH state machine—ran as tightly coupled processes. The Sawyer's kinematic parameters, communication ports, and ArUco marker layout were hardcoded in source files. The robotic skin system similarly embedded robot-specific control parameters and sensor configurations in ROS launch files.

Neither system supported adding a second robot or deploying at a different institution without rewriting these components. The YAML task files, while a step toward declarative configuration, covered only task parameters and did not extend to robot kinematics, network topology, or control parameters. These limitations motivated the parameterized architecture presented in the next chapter, where all robot-specific and site-specific parameters are extracted into a unified YAML configuration hierarchy.

8.8 Summary

This chapter contributed two architectural patterns. The ROS-based concurrent sensing/actuation across the Sawyer collaborative arm, modular lab equipment, Unity client, and PHANToM Omni haptic device (Obj 1) supported real-time MR-mediated programming by demonstration with task-parameterized Gaussian Mixture Model trajectory synthesis (Calinon, 2016). The SMACH state machine with YAML-

based task configuration (Obj 2) decoupled scientific workflows (dispensing, stirring, characterization) from source code, allowing scientists to define synthesis protocols through GUI-based parameter editing. The architecture supported the DMMRP for high-throughput experimentation (Lee *et al.*, 2024b), and the Fiber Bragg Grating robotic skin enabled autonomous obstacle avoidance and admittance-controlled object manipulation through real-time contact sensing. The monolithic single-robot architecture shared by both projects could not scale to multiple robots or distributed sites, motivating the declarative configuration hierarchy of Chapter 9.

Chapter 9

SYSTEM 8: MULTI-ROBOT PLATFORM FOR DISTRIBUTED PHRI

9.1 Introduction

Physical human-robot interaction research is expanding from single-robot, single-site experiments to multi-robot deployments across distributed facilities, driven by validation of rehabilitation protocols across diverse robot platforms, equipment sharing across institutions, and multi-robot collaborative manipulation in hazardous environments (Quigley *et al.*, 2009). The software infrastructure has not kept pace: ROS2 (Quigley *et al.*, 2009; Macenski *et al.*, 2022) provides publish-subscribe communication and standardized message types but does not prescribe how multi-robot systems should manage configuration, so adding a robot requires writing new launch files, modifying node parameters in source code, and rewriting network parameters at each site—producing configuration drift (the gradual divergence of nominally identical deployments) that makes cross-site comparison of experimental results unreliable. The DMMRP (Chapter 8) demonstrated MR-mediated single-robot, single-site interaction, but its YAML-based task configuration could not accommodate additional robots or distributed deployment without code modification.

This chapter presents a base-override YAML configuration hierarchy in which robot-specific behavior is fully determined by configuration files: adding a new robot platform requires no code modification, only configuration files specifying kinematic parameters, control endpoints, and communication topology, with each parameter defined in exactly one location and propagated at runtime. The architecture is described as a five-level configuration hierarchy with the `hri_msgs` robot abstraction layer (pluggable

kinematics backends), a Zenoh (Corsaro, 2022)-based cross-distribution bridge for ROS2 Humble ↔ Jazzy real-time messaging, an adaptive network routing mechanism, and Docker (Merkel, 2014) containerization. Three integration cases validated the architecture across heterogeneous robot platforms (Table 9.2) at two institutions: the same rehabilitation task on three kinematically different robots, cross-site deployment at two institutions, and a dual-arm teleoperation configuration prepared for an external collaboration (Lee and Park, 2024b,a) that integrates a bilateral teleoperation controller developed at KAIST IRiS Lab; full live execution of the integrated dual-arm bilateral loop on the ROS2 architecture reported in this dissertation is listed as a future-work item in Chapter 14.

9.2 Background and Related Work

9.2.1 Robot Middleware and Multi-Robot Systems

The Robot Operating System (ROS) (Quigley *et al.*, 2009) and its successor ROS2 (Macenski *et al.*, 2022) provide publish-subscribe communication, standardized message types, and a package library for robot software. However, ROS/ROS2 does not prescribe how multi-robot systems should manage configuration. Launch files, parameter servers, and node-level arguments couple configuration to code: adding a robot typically requires writing new launch files and modifying node parameters in source code.

Several frameworks address multi-robot coordination at the planning level (Yan *et al.*, 2013; Parker *et al.*, 2016). Most assume a homogeneous fleet or focus on task allocation rather than the lower-level problem of integrating heterogeneous hardware with consistent interaction quality. In pHRI applications, each robot's dynamics

directly affect patient safety and therapeutic efficacy, making this integration problem critical.

9.2.2 Configuration Management in Robotics

Docker-based containerization (Merkel, 2014) has been applied to robotics for reproducible deployment. Existing Docker-based approaches containerize individual robot stacks without centralized configuration management.

The concept of a single source of truth originates in database design (Kent, 1983) and has been applied to infrastructure-as-code practices in cloud computing (Morris, 2016). This dissertation adapts the principle to multi-robot pHRI systems by treating robot configurations as declarative specifications that fully determine runtime behavior.

9.3 Hardware Platform

The platform integrates heterogeneous robot platforms across two institutions through the configuration hierarchy of Section 9.5 (Table 9.2). Site A (ASU) hosts the xArm5 (UFACTORY, 5-DoF) and the iiwa14 (KUKA, 7-DoF). Site B (ANL) hosts six serial arms (UR5e, UR16e, UR3e, Franka FR3 / Med7 / Med14, Kinova Gen3, UFACTORY Lite6) and two humanoids (Rainbow Robotics RB-Y1, Unitree G1). The configuration hierarchy integrates each platform through a single YAML override file, and Section 9.6 reports the integration cost across all platforms.

9.4 Software Requirements

The multi-robot multi-site deployment of Section 9.6 imposes four software requirements that traditional ROS2 launch-file-based approaches cannot satisfy:

1. Base-override configuration in which each robot platform is integrated through a single declarative file (kinematic parameters, control endpoints, named positions),

with each parameter defined in exactly one location and propagated to all dependent components at runtime, so that adding a new robot requires no source code modification.

2. Robot abstraction layer (`hri_msgs`) with a uniform motion interface and pluggable kinematics backends (Product of Exponentials, KDL, TRAC-IK, Drake), so that the same rehabilitation task can execute across robots of widely varying degrees of freedom through configuration alone.

3. Cross-distribution real-time messaging across ROS2 Humble (robot controllers, MR application) and ROS2 Jazzy (Drake physics simulator) whose DDS implementations are incompatible, so that simultaneous multi-robot operation across distributions preserves deterministic timing for safety-critical pHRI commands.

4. Adaptive network routing that resolves site-specific endpoints from centralized configuration at startup, supporting local, single-site, and multi-site deployment from the same software stack distinguished only by a site identifier.

9.5 Software Architecture

The architecture combines two patterns: a Zenoh-based cross-distribution bridge with adaptive network routing (addressing Obj 1) and a five-level YAML configuration hierarchy with the `hri_msgs` robot abstraction layer and Docker service architecture (addressing Obj 2). Table 9.1 summarizes.

Table 9.1: Architectural contributions of Chapter 9 to the dissertation's two objectives.

Objective	Method
Obj 1 (real-time parallel)	Cross-distribution Zenoh bridge (ROS2 Humble ↔ Jazzy); adaptive network routing at startup
Obj 2 (modular scalable)	Five-level YAML configuration hierarchy; `hri_msgs` abstraction with pluggable kinematics backends; Docker service architecture per robot

9.5.1 Parallel Architecture

Cross-Distribution Real-Time Communication

The platform spans two ROS2 named releases (distributions): Humble (robot controllers and the MR application) and Jazzy (Drake physics simulator). These distributions use incompatible Data Distribution Service (DDS) implementations and cannot communicate directly.

The architecture bridges this gap through Zenoh (Corsaro, 2022), a publish-subscribe protocol that operates below the DDS layer. Bridge instances on each ROS2 domain exchange messages with explicit filtering, restricting cross-distribution traffic to the motion interface services. This maintains deterministic timing for safety-critical pHRI commands while supporting simultaneous multi-robot operation across distributions. The bridge topology is shown in Figure 9.1.

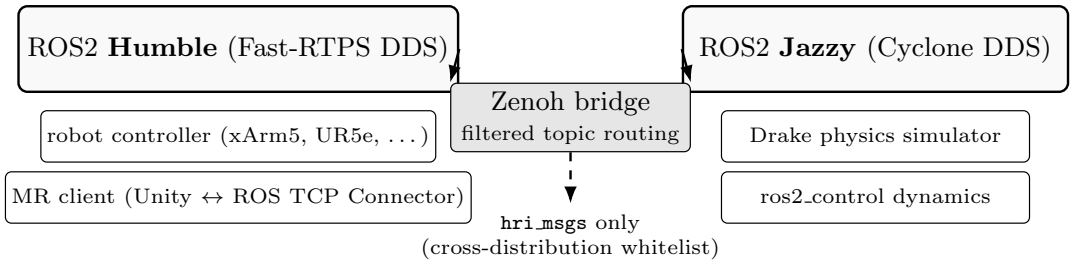


Figure 9.1: Cross-distribution Zenoh bridge. The robot controllers and the MR application run on ROS2 Humble; the Drake physics simulator runs on ROS2 Jazzy, whose DDS implementation is incompatible with Humble. A filtered Zenoh bridge forwards only the `hri_msgs` motion-interface topics between the two distributions, preserving real-time timing for safety-critical commands.

Adaptive Network Routing

Multi-site deployment introduces network topology differences that break hardcoded connection parameters. Building on the ConnectionResolver architecture in System 5 (Chapter 7), which handles per-device routing for MR clients, the approach is extended to a centralized network configuration that maps logical service names to site-specific endpoints for all services.

The routing mechanism resolves connection parameters at startup based on a site identifier in the configuration file. This abstraction supports three deployment scenarios—local development, single-site, and multi-site—from the same software stack, distinguished only by the site identifier.

9.5.2 Modular Architecture

Configuration Hierarchy

The system configuration is organized into five hierarchical levels, each managing a distinct aspect of the platform:

1. **Robot configuration**: Kinematic chain parameters (Product of Exponentials, joint limits, link lengths), control parameters (control rate,

impedance/admittance gains), frame hierarchy, and named positions for each robot type. A base-override pattern allows robot-specific files to override only parameters that differ from defaults.

2. **Network configuration**: Port assignments, site-specific hostnames, and connection parameters. A single file defines all network parameters for all deployment sites.

3. **Task configuration**: Rehabilitation exercise definitions as sequences of typed operations (waypoint, trajectory, pattern, gripper, vision). Clinicians define therapeutic protocols through these files alone. This level directly extends the YAML task parameter files introduced in the DMMRP (Chapter 8).

4. **Pattern configuration**: Parameterized motion patterns (circle, diamond, spiral, square, hemisphere, triangle) stored as unit patterns with normalized coordinates and scaled to each robot's workspace at runtime.

5. **AI agent configuration**: Model backends, memory limits, collaboration rules, and human-in-the-loop approval levels for each AI agent. This level provides the configuration foundation for the multi-agent AI platform described in Chapter 10.

All configuration files use YAML format with environment variable expansion (`${VAR:-default}`), allowing deployment-specific customization without modifying the files themselves. Appendix D provides the complete configuration reference. The base-override mechanism that propagates a single change across the five levels is shown in Figure 9.2.

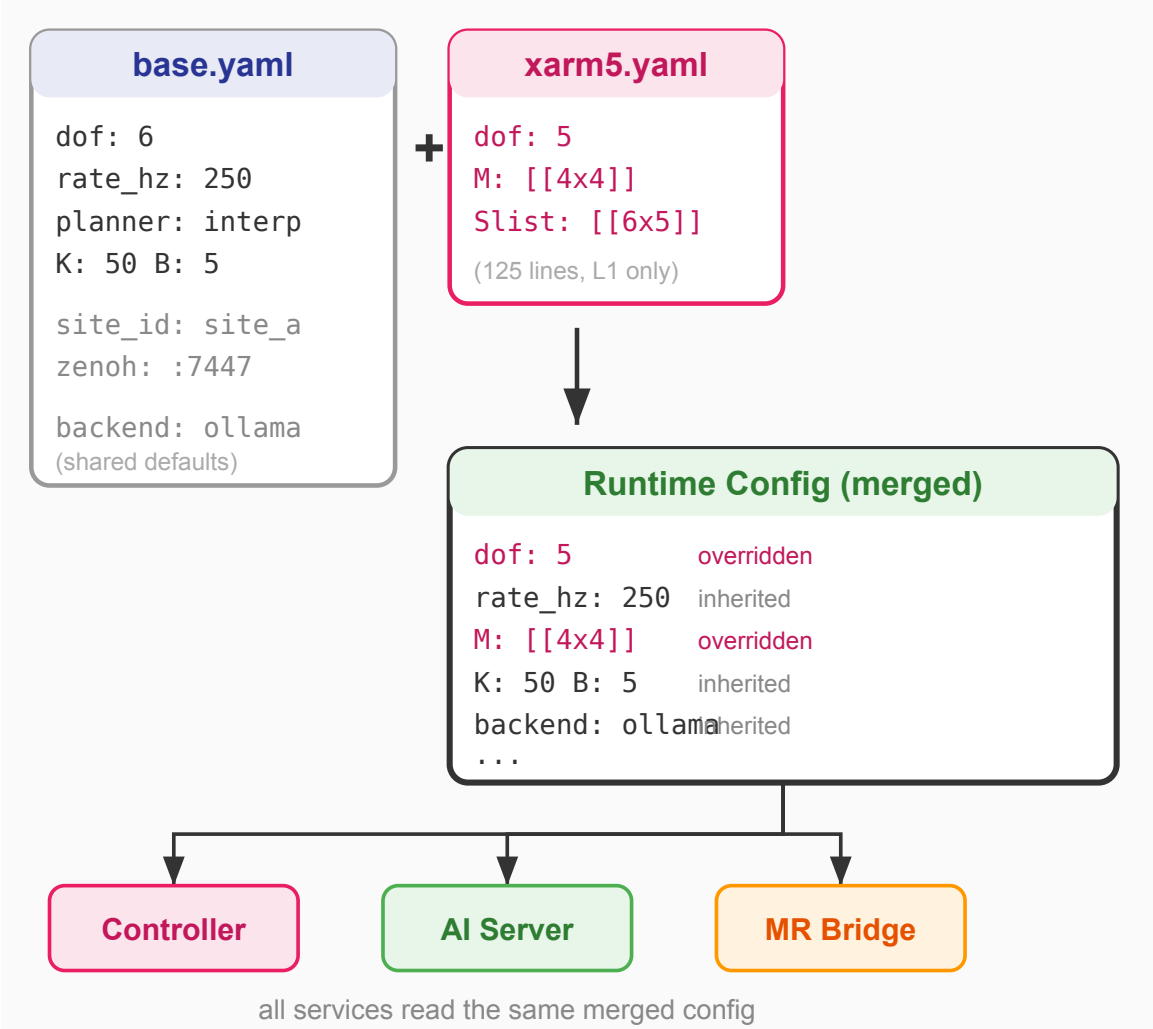


Figure 9.2: base-override configuration mechanism. The shared `base.yaml` defines defaults for all five configuration levels. A per-robot override file (e.g., `xarm5.yaml`, 125 lines) replaces only the parameters that differ. The merged result propagates to all services through a shared Docker volume mount. Red text indicates overridden values; black text indicates inherited defaults.

Robot Abstraction Layer

The `hri_msgs` ROS2 package provides a uniform motion interface across heterogeneous platforms. The package contains message, service, and action definitions for motion planning and control:

- **Robot-agnostic services**: The `MotionPlan` service accepts goal specifications by joint angles, Cartesian pose, or named configuration. The planner backend (interpolation, Drake, MoveIt, OMPL) is selected per-request with automatic fallback.

- **Pluggable kinematics**: Forward kinematics, inverse kinematics, and Jacobian computation are abstracted behind a common interface. A Product of Exponentials (POE) solver is implemented as the default backend (Appendix B), with KDL (Kinematics and Dynamics Library), TRAC-IK, and Drake (Drake Development Team, 2024) as alternatives.

- **Namespace isolation**: Each robot instance operates under its own ROS2 namespace (`/{robot_id}/`), preventing topic collisions when multiple robots are active.

- **Execution mode abstraction**: The `ExecutionMode` message selects among real hardware, simulated ros2_control, Drake, MuJoCo, Isaac Sim (NVIDIA, 2024), and Gazebo (Koenig and Howard, 2004) backends. The same rehabilitation task can execute on physical robots or in simulation through configuration selection.

A rehabilitation task defined for the xArm5 can thus execute on the iiwa14 or UR5e with only a configuration change. Figure 9.3 summarizes the abstraction layer.

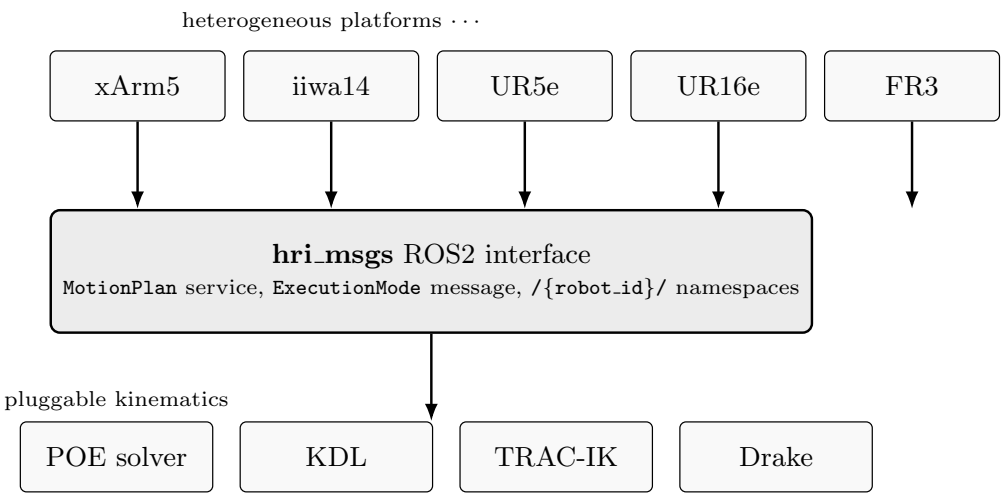


Figure 9.3: `hri_msgs` robot abstraction layer. Heterogeneous robots expose the same `MotionPlan` / `ExecutionMode` interface to the rest of the system. A kinematics backend (POE, KDL, TRAC-IK, or Drake) is selected per request, and the robot-specific YAML override declares which backend applies to which robot.

Docker Service Architecture

The system is deployed as containerized services organized by functional domain. Each container encapsulates a specific service (robot controller, AI server, retrieval pipeline, voice interface, 3D scanner, or communication bridge) and accesses the centralized configuration directory. All services reference the same parameter values regardless of the deployment site.

Containerization provides reproducible deployment—the same service definitions produce identical configurations at Site A, Site B, and demonstration sites, and a validated rehabilitation protocol behaves consistently across institutions—and isolates service dependencies, so the AI runtime, communication bridge, and robot controller do not interfere with each other. The two-site topology served by a single `docker-compose.yaml` is shown in Figure 9.4.

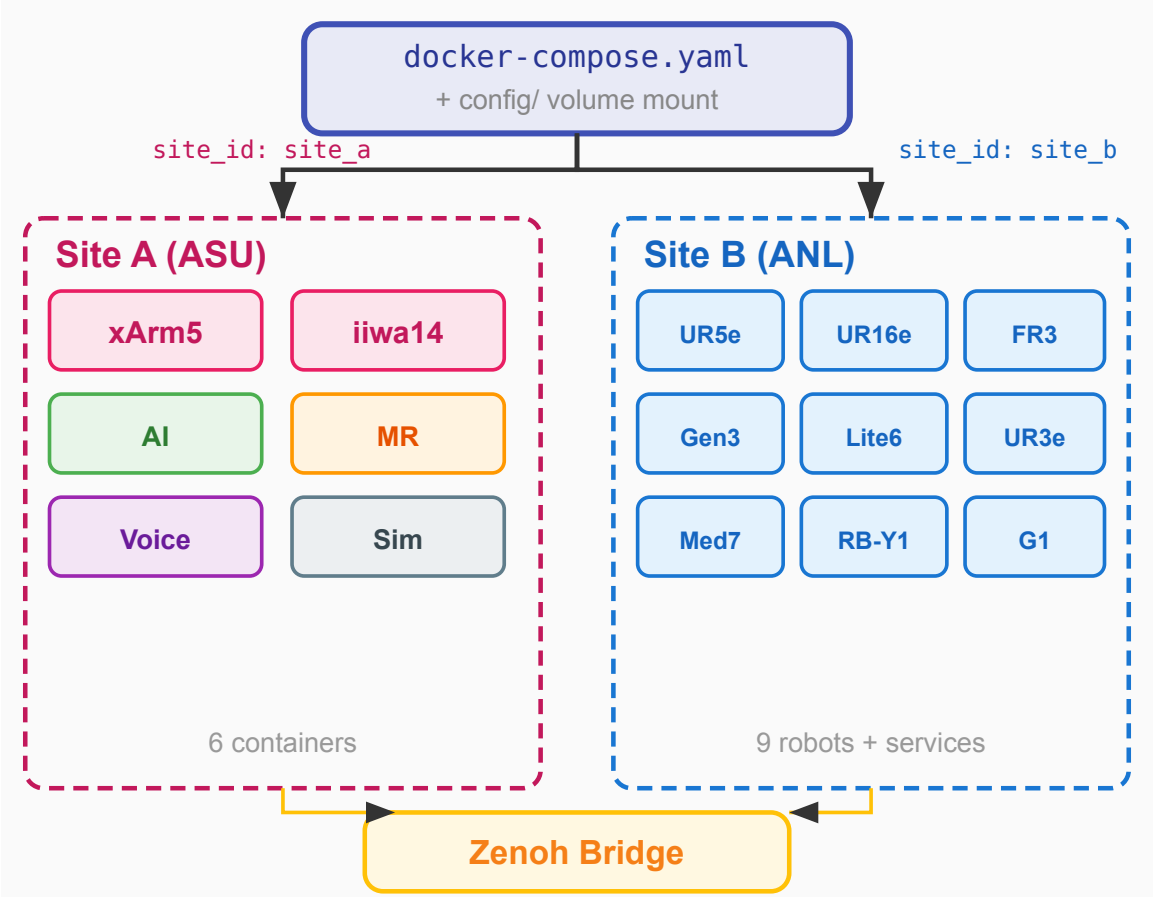


Figure 9.4: Multi-site deployment from a single `docker-compose.yaml`. The same service definitions deploy at both sites; only the `site_id` parameter differs. Site A runs 2 robots and 4 services. Site B runs 9 robots with the same services. The Zenoh bridge provides cross-distribution and cross-site messaging.

Task-Contract Deployment Pipeline

The configuration hierarchy described above is paired with a declarative task-contract specification that covers the full experimental lifecycle—build, deploy, launch, load, execute, verify—through a sequence of action-typed cards. A robotics dispatcher routes each card to the appropriate executor based on its declared `action.type`: `ssh_job` and `ros2_launch` dispatch to a remote ROS2 host, `unity_build` dispatches to a Unity Editor instance through a Model Context Protocol bridge, and `xr_api` dispatches to head-mounted-display vendor interfaces (Android Debug Bridge for Quest 3, the

Device Portal REST API for HoloLens 2). The transport layer is therefore transparent from the task-contract's perspective; the contract caller specifies only `action.type` and the dispatcher resolves the corresponding executor. This single dispatch-table extension lets a four-device deployment (one PC editor, one HoloLens 2, two Quest 3 headsets) be specified in one task-contract file and executed end-to-end with a single button press, reusing the existing task-contract semantics for robot bringup and trial recording without introducing a parallel deployment mechanism.

The connection-resolver service that sits in front of the dispatcher is caller-subnet-aware. When a head-mounted display on the wireless segment of the laboratory network issues a request to the dispatcher, the resolver enumerates local interfaces, filters them against an explicit allow-list of LAN subnets, and excludes both the institution's wide-area network and the IPv6 mesh overlay used for cross-site federation; the address returned to the caller is therefore always the LAN-local interface that the caller can reach without traversing the wide-area or overlay networks. This separation between subnet-aware address resolution and the wide-area mesh overlay is what lets the same task-contract specification deploy unchanged in single-site and cross-site configurations.

9.6 Results

9.6.1 Case Study: ASU Laboratory (xArm5 and KUKA iiwa14)

The primary robots at Site A are a 5-DoF xArm5 (UFACTORY) and a 7-DoF iiwa14 (KUKA). Both robots are integrated by creating robot-specific configuration files. The two robots differ in kinematic structure, control interface, and communication protocol, yet both use the same motion interface services, trajectory execution pipeline,

and MR visualization components. A clinician can design a rehabilitation exercise once and deploy it on either robot through configuration alone.

9.6.2 Case Study: ANL Robot Integration

A collaborating national laboratory (Site B) provided access to six additional robot platforms: UR5e, UR16e, Franka Research 3, Kinova Gen3, xArm Lite6, and UR3e. Each robot is integrated by adding only configuration files. No modifications to the core software were required.

An external collaboration provided a specific validation scenario. A dual-arm teleoperation system using the UR5e and UR16e was deployed for development at Site B and subsequently demonstrated at a third facility. The same containerized services and configuration files were used at both locations; only the site identifier changed.

9.6.3 Simulation Backend Integration

Three simulation backends (Drake (Drake Development Team, 2024), Isaac Sim (NVIDIA, 2024), and Gazebo) are integrated as alternative execution targets for the motion interface. The `ExecutionMode` message allows users to switch between real hardware and simulation at runtime, allowing safe prototyping of rehabilitation exercises before patient interaction.

The parameterized architecture provides the infrastructure for the multi-agent AI platform described in Chapter 10, as discussed in Section 9.7. Three elements enable this extension.

Configuration layer. The fifth level of the YAML hierarchy defines model backends, memory limits, agent collaboration rules, and human-in-the-loop approval levels. These parameters follow the same configuration-hierarchy principle as robot and

network configurations. The same multi-agent system operates consistently whether deployed at Site A with the xArm5 or at Site B with the UR5e.

Docker microservices. The containerized architecture provides isolated execution environments for AI services (LLM inference, RAG retrieval, voice processing, vision-language models) alongside robot control services. The AI containers share the same configuration volume mounts and network routing as robot containers.

Real-time communication. The Zenoh-based communication architecture provides the low-latency messaging required for AI agents to participate in real-time pHRI. When a vision-language model agent detects a safety concern, it must communicate that assessment to the robot controller within the same timing constraints as a force sensor signal. The parallel architecture ensures that AI-generated commands traverse the same communication paths as sensor data and control signals.

9.6.4 Deployment Scenarios

The architecture is validated through three integration cases that exercise the core claim: heterogeneous robots at distributed sites run the same rehabilitation software through configuration alone.

Integration Case 1: Same Task on Heterogeneous Robots

A rehabilitation exercise consisting of a circular trajectory pattern was defined once in the task configuration layer. This exercise was deployed on three robots with different kinematic structures: the 5-DoF xArm5 (UFACTORY), the 7-DoF KUKA iiwa14, and the 6-DoF UR5e (Universal Robots). The three robots differ in joint count, workspace geometry, control interface protocol, and maximum payload.

For each robot, the corresponding robot configuration file (specifying POE parameters, joint limits, and control rate) and the shared task configuration file (specifying

the circular pattern radius, speed, and impedance parameters) were loaded. No source code was modified between deployments. The motion interface service computed robot-specific joint trajectories from the shared Cartesian task specification using each robot's pluggable kinematics backend.

All three robots executed the prescribed circular trajectory within their respective workspaces. The exercise parameters (pattern radius scaled to workspace, impedance gains) transferred directly through the configuration hierarchy.

Integration Case 2: Cross-Site Deployment

The same containerized service stack was deployed at two geographically separated institutions: Site A (ASU) and Site B (ANL). At Site A, the xArm5 and iiwa14 ran rehabilitation exercises with the MR visualization from System 5. At Site B, the UR5e and UR16e ran the same exercises. The only difference between deployments was the site identifier in the network configuration file, which resolved hostnames and port assignments for each institution's network topology.

The Zenoh bridge maintained cross-distribution communication between ROS2 Humble (robot controllers) and ROS2 Jazzy (Drake simulator) at both sites. The Docker service definitions were identical; `docker compose up` at either site produced a functional deployment using the site-specific configuration.

Integration Case 3: Argonne National Laboratory Dual-Arm Teleoperation

An external collaboration prepared a dual-arm teleoperation configuration using the UR5e (leader) and UR16e (follower) at Site B for demonstration at a third facility (Lee and Park, 2024b,a). The dual-arm configuration was added by creating a task configuration file that specified leader-follower role assignments, joint mapping between the two different robots, and safety constraints for the teleoperation mode.

The bilateral teleoperation controller (Lee and Park, 2024b,a) (developed at KAIST IRiS Lab and exercised in a 2024 ROS1 demonstration of that work) was integrated into the same containerized architecture. Configuring deployment at the third facility required changing only the site identifier and network configuration—no recompilation or code modification. End-to-end validation of the integrated dual-arm bilateral loop on the ROS2 architecture reported in this dissertation is listed as a future-work item in Chapter 14.

9.6.5 Quantitative Integration Cost

Table 9.2 summarizes the integration cost for all robot platforms currently supported by the architecture. For each robot, the table reports the number of lines in its YAML override file and the number of source code lines modified to integrate the platform.

Table 9.2: Robot integration cost across the supported platforms at two institutions. Each platform was integrated by adding a single YAML override file. No source code modification was required for any platform. DoF = degrees of freedom.

Robot	**Manufacturer**	**DoF**	**YAML lines**	**Code changed**	**Site**
xArm5	UFACTORY	5	125	0	A
iiwa14	KUKA	7	92	0	A
UR5e	Universal Robots	6	100	0	B
UR16e	Universal Robots	6	153	0	B
UR3e	Universal Robots	6	90	0	B
FR3	Franka Emika	7	107	0	B
Kinova Gen3	Kinova	7	108	0	B
Lite6	UFACTORY	6	154	0	B
Med7	Franka Emika	7	92	0	B
Med14	Franka Emika	14	92	0	B
RB-Y1	Rainbow Robotics	26	299	0	B
Unitree G1	Unitree	23	339	0	B
Mean			**146**	**0**	
Range		5–26	90–339	0	

The "Code Changed" column is zero for every platform. The YAML override file contains robot-specific parameters—Product of Exponentials (POE) kinematic parameters, joint limits, named positions (home, work, zero), and Unity prefab mappings—but no executable logic. Files range from 90 lines (UR3e, a standard 6-DoF serial arm) to 339 lines (Unitree G1, a 23-DoF humanoid requiring additional gait parameters). The mean of 146 lines requires approximately 30 minutes of configuration work for an engineer familiar with the robot's kinematic specifications.

In a standard ROS2 launch-file approach, integrating a new robot requires launch files (50–100 lines), controller node modifications (100–300 lines), URDF/xacro integration (200–500 lines), and parameter tuning scripts (50–100 lines): 400–1000 lines across 5–10 files, each carrying source code changes that risk breaking existing integrations. The configuration-hierarchy approach reduces this to a single declarative file with zero risk to the existing codebase.

The Product of Exponentials (POE) kinematics server loaded all serial robot configurations (excluding the humanoids, which use a separate whole-body controller) and responded to forward kinematics requests within 2 ms per query on the EC2 deployment, confirming that the declarative approach does not sacrifice computational performance.

9.6.6 Configuration Consistency Across Sites

The configuration hierarchy eliminates configuration drift through the single-file rule. Every robot override file exists in exactly one location within the shared configuration directory. When the system deploys at Site A or Site B, each service reads the same file through Docker volume mounts. The only parameter that differs between sites is the site identifier in the network configuration, which resolves hostnames and port assignments for the local network topology.

This property was verified on the live EC2 deployment: the SHA-256 hashes of all robot configuration files matched between the two sites' deployment directories. In contrast, traditional ROS2 deployments that duplicate parameters across launch files, environment variables, and source code constants are susceptible to configuration drift—where a parameter updated at one site is forgotten at another. For pHRI applications, this drift is a safety concern: a rehabilitation exercise validated with impedance gain $K = 50$ N/m at Site A must use exactly $K = 50$ N/m at Site B, not a stale value from a previous deployment.

9.7 Discussion

The three integration cases and the quantitative analysis in Table 9.2 confirm that heterogeneous robot platforms can be integrated into a unified pHRI framework through configuration alone. Integration Case 1 showed that the same rehabilitation task executed on robots with 5, 6, and 7 degrees of freedom without source code changes. Integration Case 2 showed that the same containerized services deployed at two institutions by changing only the site identifier. Integration Case 3 showed that a dual-arm teleoperation task deployed at a third facility with the same procedure.

Table 9.2 quantifies the cost of this integration: across the platforms listed (multiple manufacturers, spanning a wide range of degrees of freedom), every integration required zero lines of source code modification. The mean YAML override file of 146 lines contains only declarative parameters—kinematic chains, joint limits, named configurations—with no executable logic. This result demonstrates that the base-override configuration pattern successfully decouples robot-specific parameters from platform code, reducing integration effort from the estimated 400–1000 lines of source code changes typical of ROS2 launch-file-based approaches to a single configuration file.

The parallel execution model from Systems 1–2 (non-blocking threads with lock-free queues) persists inside each robot's controller container, while the modular boundaries from Systems 3–5 (decoupled interfaces) map to inter-container communication through Zenoh.

The configuration consistency verification confirms that the single-file rule eliminates drift between deployments. In traditional multi-site deployments, parameters duplicated across launch files, environment variables, and source code constants diverge as sites evolve independently. A parameter updated at one site but forgotten at another produces subtle behavioral differences that are difficult to diagnose and dangerous in pHRI. The configuration hierarchy prevents this class of errors: each parameter exists in exactly one file, and all sites read the same file.

9.7.1 Limitations

The current implementation has three architectural limitations. First, the YAML configuration files lack schema validation; a malformed configuration produces runtime errors rather than compile-time warnings. Second, the network routing mechanism assumes a star topology; peer-to-peer communication patterns are not yet supported. Third, the Zenoh bridge introduces additional latency for cross-distribution communication. Applications requiring sub-millisecond timing across ROS2 distributions would need alternative solutions.

Three quantitative measurements remain incomplete:

- **Communication latency**: Round-trip latency for multi-site Zenoh bridge communication has not been systematically profiled under varying network conditions. This measurement is necessary to determine whether the architecture satisfies sub-millisecond timing requirements for impedance-controlled pHRI across sites.

- **Concurrent robot scaling**: Whether the control rate degrades when multiple robots operate simultaneously within the same container stack has not been characterized.

- **Clinical efficacy**: No patient studies have been conducted. Whether configuration-preserved impedance parameters produce equivalent therapeutic outcomes across different robots requires IRB-approved clinical validation.

The most important limitation is the absence of intelligence. The platform can deploy heterogeneous robot platforms across two institutions and maintain consistent interaction quality through parameterized configuration. Yet it cannot judge whether an exercise is appropriate for a given patient, understand a therapist's spoken instructions, or detect that a patient is struggling. These capabilities require the multi-agent AI platform presented in the next chapter.

9.8 Summary

This chapter contributed two architectural patterns. The Zenoh-based cross-distribution bridge with adaptive network routing (Obj 1) supported real-time messaging across ROS2 Humble $\leftrightarrow$ Jazzy and multi-site deployments through the same containerized service stack. The five-level YAML configuration hierarchy with Docker containerization and `hri_msgs` robot abstraction (Obj 2) decoupled robot-specific parameters from platform code, integrating heterogeneous robot platforms (Table 9.2) at two institutions through a single declarative file per robot with zero source code modification. Three integration cases validated the architecture: the same rehabilitation task on three kinematically different robots, cross-site deployment at two institutions, and dual-arm teleoperation at a third facility for a Department of Energy (DOE) project. These patterns carry forward to Chapter 10 (multi-agent AI), Chapter 11

(personal AI infrastructure), and Chapter 12 (multi-agent interaction control), all deployed as additional Docker services within this configuration hierarchy.

Chapter 10

SYSTEM 9: MULTI-AGENT AI PLATFORM FOR PHRI DECISION SUPPORT

10.1 Introduction

Recent advances in large language models (LLMs) and vision-language models (VLMs) have opened new possibilities for robot interaction: rather than preprogramming every possible scenario, robots can perceive their environment through visual understanding, reason about appropriate actions, and communicate with operators in natural language (Ahn *et al.*, 2022; Liang *et al.*, 2023; Brohan *et al.*, 2023). Existing LLM- and VLM-based robotics frameworks share two limitations for safety-critical pHRI: they are predominantly single-agent (a specialist evaluating against a single corpus produces unreliable assessments when the safety case spans multiple authoritative domains—clinical protocol, biomechanical safety, motor-learning evidence), and they treat human oversight as control-level intervention rather than as a first-class architectural concern at the decision level. Systems 1–8 in Chapters 2–9 provide the hardware abstraction, communication, and configuration layers on which the present chapter builds, but each is reactive: none observes a patient struggling, hears a therapist's spoken instruction, or reasons about whether the next action is safe.

This chapter presents a multi-agent architecture that fills this gap, validated through two case studies on a shared agent substrate. Section 10.5 describes the substrate (pluggable agent types, swappable model backends, retrieval-augmented memory subsystem). Case Study 1 (Section 10.6.1) instantiates the substrate as a multi-agent discussion platform across a sequence of works (Chang *et al.*, 2025; Kim

et al., 2026a,b): a RAG kernel for nuclear waste site compliance, an Orchestrator-based extension with heterogeneous corpora and knowledge-graph citation tracing, and a psychologically-grounded persona design layer; together these establish the convergence properties of the discussion protocol on on-premises workstations. Case Study 2 (Section 10.6.2) reuses the same substrate for VLM-driven action proposal with a multi-level human-in-the-loop approval policy and a hands-free voice agent, instantiating physical supervision of a rehabilitation robot.

10.2 Background and Related Work

10.2.1 Large Language and Vision-Language Models in Robotics

Recent frameworks ground LLM outputs in robot affordances (Ahn *et al.*, 2022), generate executable robot code from natural-language instructions (Liang *et al.*, 2023), and map visual observations and language commands directly to robot actions (Brohan *et al.*, 2023). These systems demonstrate that natural-language interfaces can replace task-specific scripting, but they operate as single-agent pipelines without structured safety validation for physical interaction.

10.2.2 Multi-Agent AI Systems

Multi-agent frameworks orchestrate specialized agents to address tasks beyond individual model capability. AutoGen (Wu *et al.*, 2023) and CrewAI (CrewAI, 2024) provide the orchestration substrate for agent collaboration, but neither characterizes the convergence behavior of structured iterative discussion over disjoint corpora. The work of (Chang *et al.*, 2025) introduced a three-agent retrieval-augmented discussion protocol for cross-corpus regulatory compliance assessment and showed that inter-agent agreement increases and semantic drift decreases over successive rounds. The

follow-up work of (Kim *et al.*, 2026a) extended the protocol to four specialized agents coordinated by an Orchestrator, added heterogeneous data sources, and introduced a knowledge-graph-backed document library for cross-agent citation tracing. A further extension (Kim *et al.*, 2026b) added psychologically-grounded persona design to the agent role specification, demonstrating that the same substrate supports specialized stakeholder representation beyond regulatory and scientific corpora. The present chapter uses these works as the empirical foundation of the substrate and reuses the same kernel for physical supervision.

10.2.3 Human-in-the-Loop AI for Safety-Critical Systems

Human-in-the-loop frameworks maintain human authority over AI-assisted decisions in safety-critical domains (Amershi *et al.*, 2019). In robotics, shared-autonomy approaches (Dragan *et al.*, 2013) blend human input with autonomous control. These frameworks operate at the control level (adjusting autonomy blending ratios) rather than at the decision level (approving or rejecting a proposed action plan), and they do not provide a configurable policy that binds approval requirements to agent classes and action classes through the system's configuration hierarchy. The four-level policy of Section 10.6.2 addresses this gap.

10.3 Hardware Platform

Each cluster site uses a single workstation; larger-scale convergence runs of Section 10.6.1 additionally use a university computing cluster (Jennewein *et al.*, 2023). RGB camera input is sourced through the shared-memory vision substrate of Chapter 11. The remaining infrastructure—inference runtimes, retrieval-augmented memory, and the inter-agent message substrate—is described in Section 10.5.

10.4 Software Requirements

The case studies impose four software requirements that a single-agent LLM application cannot satisfy directly:

1. Per-agent retrieval-augmented memory with disjoint corpora, so that each specialist queries its own isolated knowledge base and structured multi-round discussion produces convergent rather than divergent assessments over multiple authoritative domains.

2. Pluggable agent types behind a common input-context / output-action interface and pluggable model backends behind a common inference interface, so that adding a new agent (Vision-Language, Approval, Voice) or switching between local Ollama and cloud endpoints is a configuration change rather than a code change.

3. Cloud-independent inference, so that institutional settings without cloud LLM access can host the substrate.

4. Configurable per-action human-in-the-loop approval policy bound at the decision level rather than the control level, so that the same compiled architecture serves a permissive research deployment (notify on execute) and a strict clinical deployment (explicit approval) through a YAML configuration change.

10.5 Software Architecture

The architecture combines two patterns: parallel retrieval across disjoint corpora with concurrent vision-language inference across swappable backends and an asynchronous voice pipeline (addressing Obj 1) and pluggable agent types behind a common

interface with a configurable human-in-the-loop approval policy and a knowledge-graph-backed retrieval subsystem (addressing Obj 2). Table 10.1 summarizes.

Table 10.1: Architectural contributions of Chapter 10 to the dissertation's two objectives.

Objective	**Method**
Obj 1 (real-time parallel)	Parallel retrieval across disjoint corpora; concurrent vision-language inference across swappable backends; asynchronous voice pipeline
Obj 2 (modular scalable)	Pluggable agent types behind a common interface; configurable human-in-the-loop approval policy; knowledge-graph-backed retrieval subsystem

The architecture provides three reusable substrates that the case studies share: pluggable agent types behind a common interface, swappable model backends behind a common inference interface, and a retrieval-augmented memory subsystem that grounds every agent response in authoritative sources. Each is described once below; Case Study 1 (Section 10.6.1) and Case Study 2 (Section 10.6.2) use different subsets and bindings of the same substrate. Figure 10.1 summarizes the three substrates and the YAML configuration that binds them.

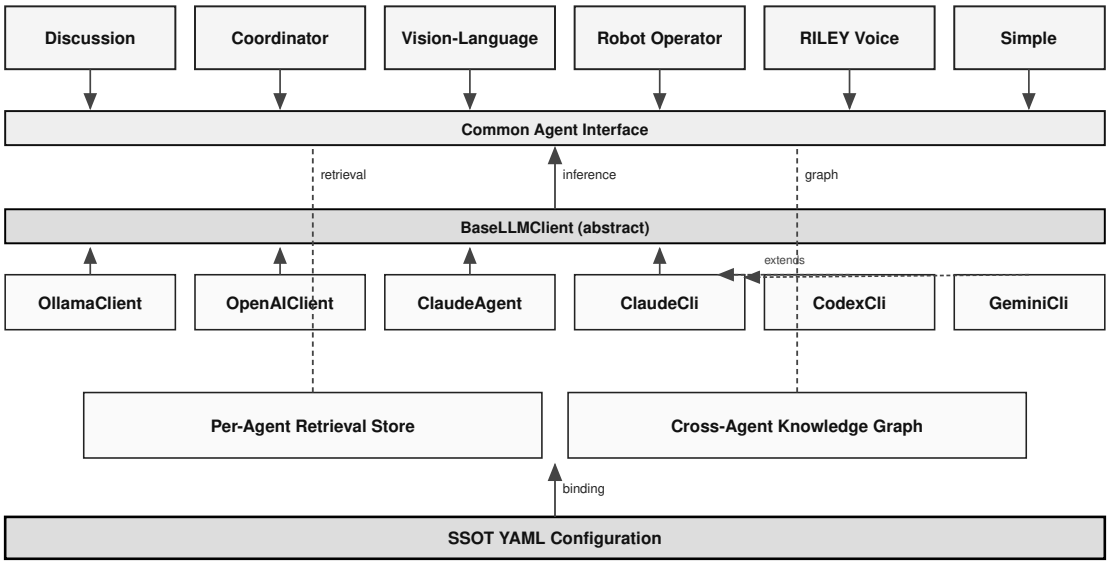


Figure 10.1: Multi-agent architecture. Six agent types operate behind a common input-context / output-action interface, drawing inference from swappable model backends and grounding their responses in a per-agent retrieval store and a cross-agent knowledge graph. The fifth level of the configuration hierarchy of Chapter 9 binds each agent to its backend, retrieval context, and approval level.

10.5.1 Parallel Architecture

Pluggable Model Backends

The inference layer supports multiple backends, selectable through configuration: a local Ollama runtime (Ollama, 2024) for quantized models, direct loading from the HuggingFace model hub for vision-language models, and cloud endpoints (Claude, Gemini, Codex) for scenarios in which local compute is insufficient. The application code interacts with a common interface; the specific backend is selected per agent through the fifth-level YAML, so switching from a local quantized model to a cloud endpoint does not change agent code or prompts.

Per-Agent Retrieval and Memory

Each agent grounds its responses in authoritative sources through a retrieval-augmented generation subsystem. The subsystem supports per-agent retrieval contexts, so each specialist queries its own isolated corpus; this isolation is central to the convergence study of Section 10.6.1, where the agents bind to disjoint regulatory and scientific corpora and exchange assessments without sharing retrieval state. Long-term memory and cross-agent document relations are provided by a graph store that connects each document to the agents that own it and to the regulatory or technical requirements it supports, so that a citation chain can be traced across agents in a single query. The graph store is the basis for the cross-corpus citation tracing of (Kim *et al.*, 2026a) and for the per-session interaction history maintained during physical supervision (Section 10.6.2).

10.5.2 Modular Architecture

Pluggable Agent Types

Six agent types are defined as independent, pluggable components, each implementing a common interface that takes an input context and emits an action or assessment. The Discussion Agent runs the structured multi-round protocol against an isolated retrieval context; multiple Discussion Agents with different role profiles assess proposed actions or assessments through iterative dialogue. The Coordinator Agent decomposes a request into subtasks, dispatches them to the relevant specialist subset, and synthesizes their responses. The Vision-Language Model Agent consumes camera frames through the shared-memory vision substrate of Chapter 11 and produces structured scene descriptions. The Robot Operator Agent translates approved actions into the unified motion interface of Chapter 9 after validating them against the kinematic and

workspace constraints from the per-robot configuration. The RILEY Voice Agent provides the audio modality of the agent substrate through speech-to-text, intent classification, and spatial text-to-speech. The Simple Agent handles routine queries that do not require multi-agent coordination. Each agent type is deployed as a service within the personal AI infrastructure of Chapter 11, and adding a new agent type requires no modification to the platform's core code.

10.6 Results

10.6.1 Case Study 1: Multi-Agent Discussion Substrate

The discussion kernel was first exercised on nuclear waste site assessment because that domain shares the structural challenge of multi-domain pHRI safety oversight: a single decision must be grounded in disjoint authoritative corpora—regulatory, geological, and environmental in the waste-assessment case; clinical protocol, biomechanical safety, and motor-learning evidence in pHRI—and a unilateral assessment by any single specialist is unreliable.

The three-agent kernel of (Chang *et al.*, 2025) addressed two open problems in multi-agent reasoning over disjoint corpora. First, whether structured iterative discussion among specialized agents produces convergent or divergent assessments was unsettled: prior multi-agent frameworks supplied the orchestration substrate but did not characterize convergence behavior (Wu *et al.*, 2023; CrewAI, 2024). Second, whether the convergence behavior holds at modest model scale—3-billion parameters, sufficient for cloud-independent inference—was untested. Both were addressed by exercising three Discussion Agents (Regulatory Compliance, Safety and Environmental, Documentation and Reporting), each bound to its own retrieval context, under a

structured ten-round protocol on a 3-billion-parameter local model deployed on a university computing cluster (Jennewein *et al.*, 2023).

The protocol produced agreement rates that increased monotonically and semantic drift that decreased monotonically over successive rounds (Figures 10.2 and 10.3).

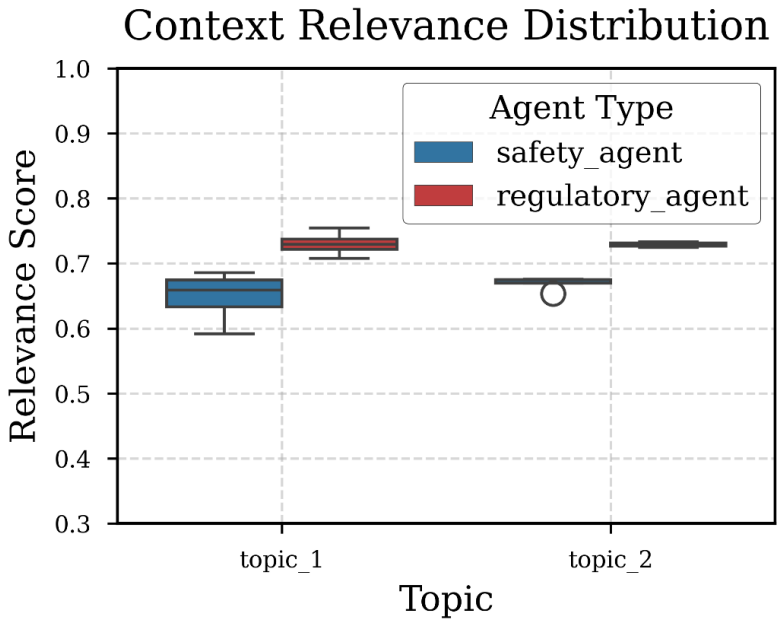


Figure 10.2: Context relevance distribution for the Regulatory Compliance Agent and the Safety and Environmental Agent across two assessment topics. The vertical axis is the cosine similarity between an agent's response vector and the retrieved document vectors; higher values indicate stronger document grounding. Both agents maintain relevance scores predominantly above the 0.3 grounding threshold, with the Regulatory Agent achieving higher median scores (reflecting structured legal documentation) and the Safety Agent exhibiting wider variability (reflecting the integrative nature of environmental risk assessment). Reproduced with author's permission from (Chang *et al.*, 2025).

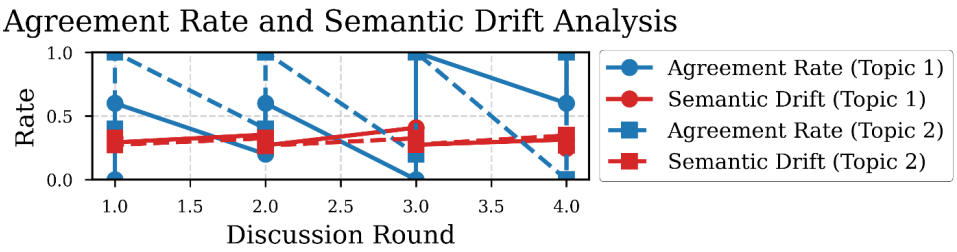


Figure 10.3: Agent agreement rate (left axis, increasing) and semantic drift (right axis, decreasing) across discussion rounds for two assessment topics. Reproduced with author's permission from (Chang *et al.*, 2025).

The follow-up work of (Kim *et al.*, 2026a) extended the kernel to address two further open problems. First, whether the convergence holds when the corpus set extends beyond authoritative documents to include heterogeneous sources—3D terrain visualization and public sentiment—that require different ingestion paths and that resist uniform retrieval-based representation. Second, whether cross-corpus citation tracing, a regulatory-audit requirement that the per-agent retrieval of the three-agent kernel could not satisfy, could be supported by promoting the document library to a graph store that links each document to the agents and requirements it supports. Both were addressed by adding an Orchestrator that selects agents at runtime, two domain-specific specialist agents (a Field agent for terrain and a Public Opinion and Social agent for sentiment), and a knowledge-graph extension of the retrieval subsystem; the convergence pattern held under the four-agent extension on the Yucca Mountain assessment.

A third extension addressed whether the same convergence machinery supports stakeholder representation beyond authoritative corpora—where authority derives from persona alignment rather than a fixed regulatory source. The work of (Kim *et al.*, 2026b) introduced a psychologically-grounded persona design layer on top of the discussion kernel, demonstrating that the substrate accommodates persona-

conditioned role specifications without modification to the iterative protocol or the retrieval interface.

The cumulative kernel of the three works—corpus-agnostic structured discussion, monotonic convergence under iteration, on-premises deployment, and persona-conditioned role configuration—is the substrate that the next section reuses for physical supervision.

10.6.2 Case Study 2: VLM Perception with Human-in-the-Loop Approval for pHRI

Physical supervision of a rehabilitation robot is structurally a multi-domain safety-assessment problem in which the assessment must be acted upon rather than reported. The Discussion Agents of Section 10.5 are reused with their disjoint-context configuration, and two domain-specific agents are added. A Vision-Language Model Agent grounds reasoning in real-time camera observation through the shared-memory vision substrate of Chapter 11. An Approval Agent—a specialization of the Discussion Agent that operates on the action space rather than on the document space—interposes a configurable human checkpoint between any agent-proposed action and its physical execution. The four-level policy that the Approval Agent enforces (Figure 10.4) is read from the fifth level of the configuration hierarchy of Chapter 9, the same hierarchy that defines the robot kinematics and network topology in earlier chapters. The chapter's empirical scope on the VLM path covers verbal action proposal, approval gating, and compilation of the approved plan into a behavior-tree action handler; physical execution of the resulting handler on the robot is the integration concern of Chapter 13.

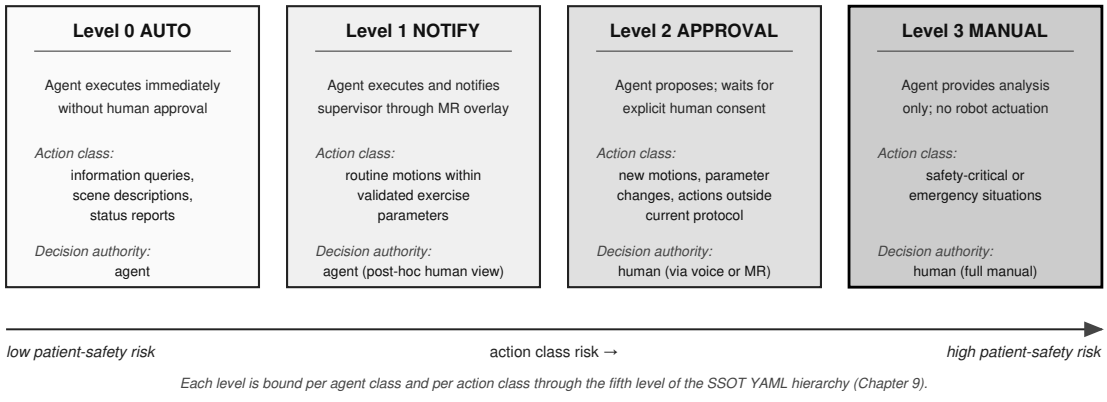


Figure 10.4: Four-level human-in-the-loop approval policy enforced by the Approval Agent. Level 0 (AUTO) executes immediately and is reserved for information queries; Level 1 (NOTIFY) executes and informs the supervisor through the mixed-reality interface, used for routine motions within validated parameters; Level 2 (APPROVAL) waits for explicit consent through the voice channel or the mixed-reality overlay, used for new motions or parameter changes; Level 3 (MANUAL) provides analysis only and retains full manual control. Each level is bound per agent and per action class through the fifth level of the configuration hierarchy.

The agents collaborate rather than execute a centralized pipeline. When the Vision-Language Model Agent reports a scene, the Coordinator Agent dispatches it to the relevant specialists for the current task: the Vision-Language Model Agent emits a structured action plan (a JSON sequence of verb / target / motion-direction / expected-outcome / abort-condition entries) against a prompt-defined schema, the Robot Operator Agent compiles the approved plan into a behavior-tree skeleton over the unified motion interface of Chapter 9, the Discussion Agents validate the proposal through the same iterative protocol used for the regulatory assessment, and the Approval Agent gates the validated proposal against the four-level policy bound to that agent class and that action class. The choreography of these exchanges is shown in Figure 10.5; each agent operates concurrently, communicates through the

inter-agent message substrate, and shares context through the per-session retrieval store.

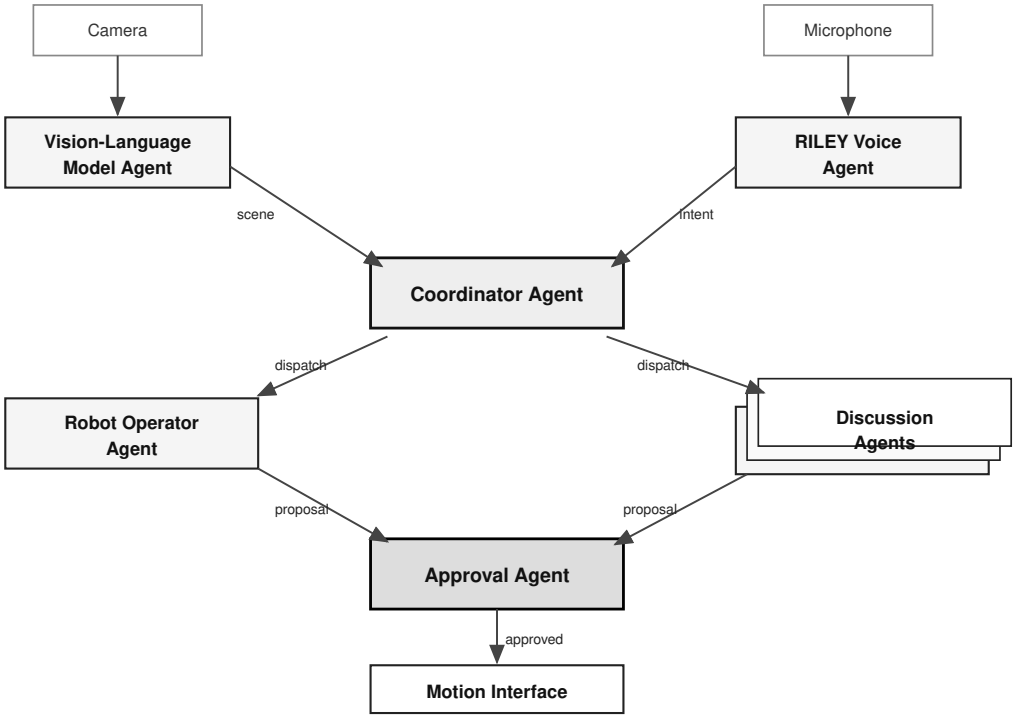


Figure 10.5: Agent collaboration during a physical supervision cycle. The Vision-Language Model Agent grounds reasoning in a current scene; the Coordinator Agent dispatches to the relevant specialists; the Robot Operator Agent translates intent into the motion interface of Chapter 9; the Discussion Agents validate the proposal through the protocol of Section 10.6.1; the Approval Agent gates the validated proposal against the four-level policy. The figure externalizes the agent exchanges; no centralized state machine drives the cycle.

The same agent layer also exposes a hands-free voice channel. The RILEY Voice Agent is not a separate interface but the audio modality of the agent substrate (Figure 10.6). Spoken commands enter the speech-to-text stage, are classified into intent categories, and are routed by the Coordinator to the appropriate specialist subset. A simple emergency-stop command bypasses the discussion layer; an exercise-adjustment request triggers a multi-agent reasoning chain across the Robot Operator,

the Discussion Agents, and the Approval Agent. Voice therefore shares the substrate's retrieval and approval mechanisms with the visual and command paths, rather than constituting a parallel control surface.

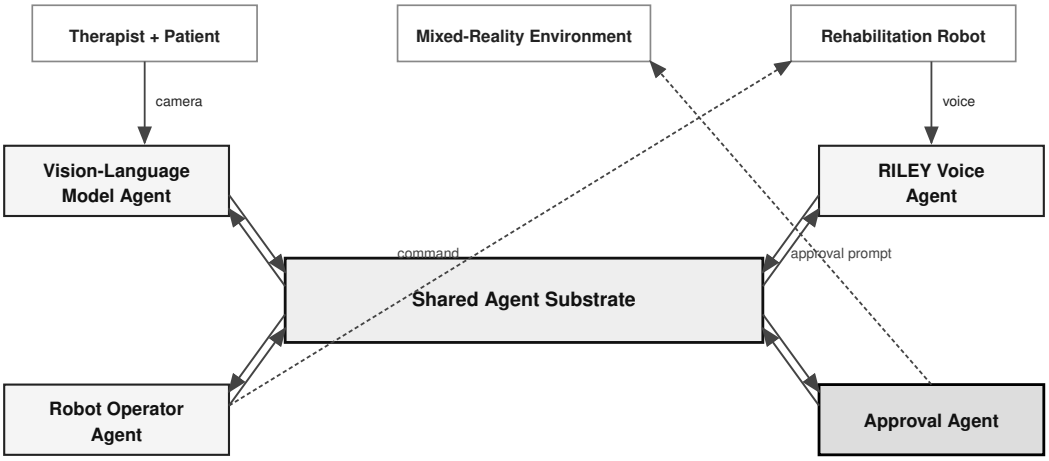


Figure 10.6: Session architecture: the Vision-Language Model Agent, the Robot Operator Agent, the Approval Agent, and the RILEY Voice Agent share the same retrieval and message substrate during a rehabilitation session. Voice is the audio modality of the agent layer rather than a separate control interface, and all four agents draw their per-agent configuration from the fifth level of the configuration hierarchy.

A persistent voice agent (*Riley*) runs in the msg skill of own-code, providing the audio modality of the agent layer. Spoken commands are transcribed by Whisper, routed through the ZMQ broker, expanded into tool calls by the tool engine, and dispatched to the orchestrator under the same approval policy as the visual and command paths.

The realization across the dissertation stack is implicit. The agent types operate as services within the personal AI infrastructure of Chapter 11; the inter-agent message substrate is shared with the multi-robot platform of Chapter 9; and the discussion protocol is reused under the names Perception, Planning, Safety, and Execution in the

three-rate hierarchical architecture of Chapter 12. The composition is exercised on a UR16e arm equipped with a DG-5F dexterous hand in Exemplar E2 of Chapter 13.

10.7 Discussion

The agent types, model backends, retrieval bindings, collaboration rules, and approval levels are all defined in the fifth level of the configuration hierarchy of Chapter 9. Adding a new agent type requires only a new section in the agent configuration; switching from a local quantized backend to a cloud endpoint requires changing one line of configuration; raising the approval level for a given action class from notify to explicit consent requires no code modification. The configuration approach of Chapter 9 therefore extends from joint limits and network endpoints to AI agent backends and decision authority, governing the entire stack through a single hierarchy. Complete agent configuration details and the per-action approval bindings appear in Appendix E.

10.7.1 Deployment Footprint

The convergence study of Section 10.6.1 ran on a single workstation with quantized Llama-3 inference through Ollama and `mxbai-embed-large` embeddings, without external application programming interface (API) calls. Per-round latency scales linearly with the number of agents and is bounded by the slowest backend; the substrate's routing and retrieval add tens of milliseconds, while a local quantized model produces one agent's textual response on the order of seconds. A formal latency benchmark across backends is identified as a future validation item in Chapter 14.

The architectural contribution of the chapter is not any single agent or backend but the substrate's reusability across domains. The kernel's convergence was established once on cross-corpus regulatory assessment (Chang *et al.*, 2025; Kim *et al.*, 2026a) and

inherited without re-validation by the physical-supervision application of Section 10.6.2; the four-level approval policy that distinguishes physical supervision from document assessment is a configuration change, not a kernel change; the voice channel that addresses the hands-engaged constraint of pHRI is the same agent layer accessed through an audio modality. This pattern—a single kernel reused across domains through configuration changes alone—is the same pattern that the configuration hierarchy of Chapter 9 applies to robot platforms, and it is the form in which the present chapter's contributions enter the cross-chapter integration of Chapter 13.

The empirical scope of the present chapter is the convergence study of Section 10.6.1. Rehabilitation-domain convergence with clinical-protocol corpora, vision-language scene-understanding accuracy on rehabilitation-relevant scenes against motion-capture ground truth, voice-pipeline end-to-end latency under clinical noise conditions, approval-workflow timing across the four levels, and therapist and patient experience under AI-assisted MR rehabilitation are addressed in the future-work agenda of Chapter 14.

10.8 Summary

This chapter contributed two architectural patterns. The shared agent substrate (Obj 1) hosts pluggable agent types, swappable model backends, and a retrieval-augmented memory subsystem behind common interfaces, supporting concurrent inference across heterogeneous corpora at the per-agent rate of the chosen backend. The same substrate (Obj 2) was reused without modification across the multi-agent discussion platform of Case Study 1 and the VLM-driven physical supervision of Case Study 2, supporting three published works on nuclear waste site compliance and stakeholder modeling (Chang *et al.*, 2025; Kim *et al.*, 2026a,b). The substrate carries

forward to Exemplar E2 of Chapter 13 (UR16e arm with DG-5F dexterous hand) and to the rehabilitation-domain quantitative validation listed in Chapter 14.

Chapter 11

SYSTEM 10: OWN-CODE PERSONAL AI INFRASTRUCTURE

11.1 Introduction

Modern robotics research integrates language models for perception, reasoning, instruction following, and human interaction. The Multi-Agent AI Platform of Chapter 10, the Diffusion-Interaction Policy of Chapter 12, and the demonstrations of Chapter 13 run language-model-scale agent inference and microsecond-bounded motor control on the same host, share camera frames between perception and policy at sub-millisecond latency, and persist conversation, run, and event state on the same backend. Application frameworks (LangChain, AutoGen (Wu *et al.*, 2023), CrewAI (CrewAI, 2024)) and robotics middleware (Macenski *et al.*, 2022) address these workloads separately; running them together on a multi-site cluster bridged by a VPN mesh (e.g., Husarnet) required the substrate this chapter describes.

This chapter presents own-code, a personal AI infrastructure that integrates vendor-agnostic language-model agents, persistent memory, vision processing, and a microsecond-bounded real-time control path on a single self-hosted substrate—a data-sovereign deployment platform.

The chapter contributes to **Objective 1** (real-time parallel) through two mechanisms: a microsecond-bounded safety checker (1–5 μs hot path, against a 50–200 μs Python baseline) and a zero-copy shared-memory vision transport (<0.2 ms target latency, against 30–300 ms for the ROS2 DDS baseline).

The chapter contributes to **Objective 2** (modular scalable) through five mechanisms: a common asynchronous base class across four language-model backends, a

provider-agnostic tool engine, a microservice skill platform, a backend-agnostic persistence interface (`own_store`) anchored to user-owned MongoDB, and a distributed network provider substrate (Corsaro, 2022) that resolves endpoints across three sites bridged by a VPN mesh.

11.2 Background and Related Work

11.2.1 LLM Application Frameworks

LangChain, LlamaIndex, AutoGen (Wu *et al.*, 2023), and CrewAI (CrewAI, 2024) provide provider abstractions and tool-calling layers for application development. They do not include a real-time C++ hot path or a zero-copy shared-memory vision transport, and were not designed for robotics research where the language-model layer must coexist with motor-control-scale workloads on the same host.

11.2.2 Robotics Middleware and Real-Time Constraints

ROS2 (Macenski *et al.*, 2022) provides publish-subscribe communication and standardized message types for robot software. Response-time analysis under reservation-based scheduling has been characterized for ROS2 processing chains (Casini *et al.*, 2019), establishing an upper bound on the middleware's contribution to controller jitter. ROS2 alone does not include a multi-provider language-model abstraction or a tool-calling registry. Zenoh (Corsaro, 2022) provides brokerless peer-to-peer messaging with liveliness tokens and automatic reconnection, complementing the ROS2 stack at the cross-distribution and cross-site boundaries.

11.2.3 Edge-Native AI and Mixed-Criticality Systems

Cloud-robotics surveys (Kehoe *et al.*, 2015) and learning-based offloading analysis (Chinchali *et al.*, 2021) establish the conditions under which on-premise inference is preferred over cloud relay. Mixed-criticality scheduling theory (Vestal, 2007; Burns and Davis, 2017) addresses the coexistence of workloads with different timing guarantees on a single host. The substrate described in this chapter operates at the intersection of these two traditions: language-model-scale agent inference at the seconds scale and microsecond-bounded motor control on the same hardware.

11.3 Hardware Platform

The platform is a single workstation per cluster site. All skill services, the language-model runtime, and the real-time C++ core execute as containerized processes on that workstation.

Three sites have been integrated: `anl_robot` (Argonne National Laboratory), `alienware` (workstation deployment host), and `asu_nchr` (Arizona State University). They connect through a VPN mesh overlay (e.g., Husarnet).

The four language-model providers (Claude, Codex, Gemini, Ollama) integrate through the asynchronous adapters of Section 11.5.2. Ollama deploys on-host as the `own-code-ollama` container (port 11434). Persistent state anchors to `own-code-mongo` (port 27017). The control plane runs as the `multi-hra-network-provider` container (HTTP port 9083), which publishes service-discovery beacons and exposes the topology query surface.

11.4 Software Requirements

The downstream workloads of Chapters 10, 12, and 13 impose six requirements on the substrate. No off-the-shelf language-model application framework satisfies them. A ROS2-only middleware stack satisfies the real-time path but not the language-model layer.

1. **R1: provider abstraction.** Common asynchronous interface across Claude, Codex, Gemini, and Ollama. Required by Chapter 10 for configuration-driven backend swap.

2. **R2: real-time core.** C++ safety checker bounded at 1–5 μs worst-case. Required by Chapter 12 inside the 1 kHz control loop.

3. **R3: zero-copy vision transport.** Sub-millisecond camera-to-policy latency. Required by Chapter 12 at 10–15 Hz inference.

4. **R4: provider-agnostic tools.** Tool engine and microservice skill platform with one tool definition per provider-independent function. Required by Chapter 13 for cross-provider workflow execution without per-provider glue.

5. **R5: multi-site substrate.** Single endpoint-resolution gateway across `anl_robot`, `alienware`, and `asu_nchr` from one image. Reuses Chapter 9's configuration hierarchy.

6. **R6: user-owned persistence.** Backend-agnostic store interface anchored to user-owned MongoDB. Holds conversation, run, and event records outside vendor cloud.

11.5 Software Architecture

The architecture combines two patterns: (i) a multi-rate parallel substrate that supports the coexistence of language-model-scale agent inference and microsecond-bounded motor control (Obj 1), and (ii) a component-based modular skills platform with provider-agnostic tool dispatch and user-owned persistence (Obj 2). Table 11.1 summarizes.

Table 11.1: Architectural contributions of Chapter 11 to the dissertation's two objectives.

Objective	**Method**
Obj 1 (real-time parallel)	Lock-free C++ core + zero-copy shared-memory vision transport (sub-ms latency) + two-plane container model (1 kHz hot path / 24/7 lightweight control plane)
Obj 2 (modular scalable)	Common async base class across LLM backends; provider-agnostic tool dispatch; independently deployable skill services; backend-agnostic persistence (own_store + MongoDB); distributed network provider substrate with endpoint-resolution gateway

own-code is organized as a seven-layer stack (Figure 11.1, Table 11.2) in which each layer exposes a narrow interface to its neighbors and may be replaced or evolved independently without recompilation or restart of any other layer. Communication is asynchronous: the frontend connects to the server through a WebSocket, the server dispatches each user message to the selected provider adapter, the adapter delegates tool calls to the tool engine, and the tool engine routes each call to a skill service over HTTP or to a C++ module through pybind11. Six principles guide the system: (i) a provider-agnostic core that confines vendor-specific LLM SDK imports to the four adapters in layer L4; (ii) microservice skills, each a self-contained HTTP and WebSocket service deployed in its own process or container; (iii) a real-time hot path in C++ exposed to Python through pybind11, with Python reserved for orchestration; (iv) declarative YAML configuration read once at startup, in the same spirit as the

configuration hierarchy of Chapter 9; (v) persistent memory written to the author's Obsidian vault so that knowledge from one conversation is available in the next; and (vi) local-first execution (Ollama, on-host vision-core, local skill services) with cloud providers treated as optional accelerators.

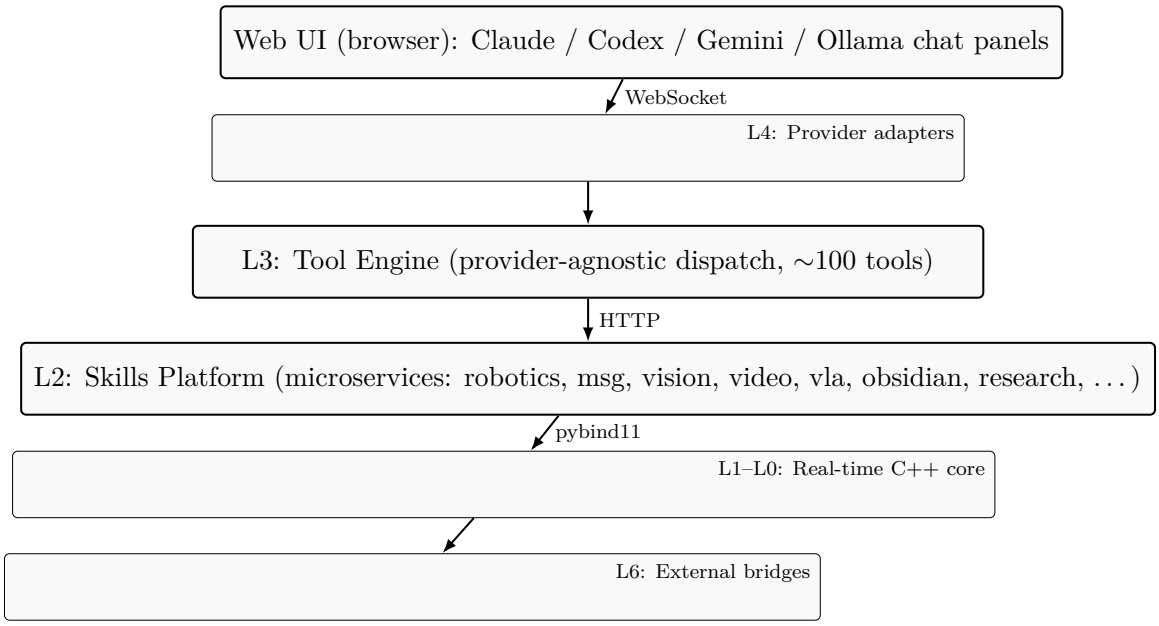


Figure 11.1: High-level architecture of own-code. The frontend, server, tool engine, and skill services are all asynchronous and communicate over WebSocket, HTTP, or shared memory; the real-time C++ core is invoked through pybind11 from any service that requires sub-millisecond execution.

Table 11.2: Seven-layer organization of own-code. Each layer exposes a narrow interface to its neighbors and may be replaced or evolved independently.

Layer	Responsibility	Modules	Lang.
L0	Hardware abstraction (SHM, atomics, ring buffers)	`cpp/`, `vision-core`	C++17
L1	Real-time hot path (safety, trajectory)	`cpp/safety_checker`, `cpp/trajectory_engine`	C++17
L2	Service skills (microservices)	`services/skills-platform/*`	Python
L3	Tool engine (provider-agnostic)	`server/api/tool_engine.py`	Python
L4	Provider adapters (LLM-specific glue)	`server/api/{claude,codex, gemini,ollama}_api_async.py`	Python
L5	Web UI (browser frontend)	`index.html`, `js/`, `css/`	HTML/JS
L6	External bridges (MCP, ROS2, Zenoh)	`services/{unity, isaac-sim}-mcp/`, `msg-skills/`	Python

11.5.1 Parallel Architecture

Real-Time C++ Core

Rationale for the C++ Core A NumPy action-chunk safety check takes 50–200 μs per call. A C++ implementation runs in 1–5 μs, executing within the 1 ms budget of a 1 kHz control loop without contributing measurable jitter.

The own-code design rule follows: any operation in a control loop is implemented in C++ and called from Python through pybind11, with Python reserved for orchestration. Language-model agent inference at the seconds scale and the controller's microsecond hot path coexist on disjoint threads communicating only through shared-memory queues, so that a Python agent stalled on token generation does not perturb the C++ control loop's jitter.

A 93.7 s sample of the UR16e `ros2_control` driver under coexisting load (multi-agent server, message broker, behavior tree runner, and several ZMQ publishers all active on the same host) preserved a 500 Hz publication cycle with a standard

deviation of 60.2 μs and a 99th-percentile period of 2.12 ms across 46,864 samples. The publish-to-record skew between the publisher header timestamp and the bag receive time had a mean of 127.5 μs and a 99th-percentile of 230 μs.

While that hot-path sample was running, the same host concurrently sustained a peer-to-peer mesh VPN link (e.g., Husarnet) between Argonne (Chicago) and Arizona State University ($\sim$1500 km, inter-state) with a 49.4 ms median round-trip time over 250 ICMPv6 samples. The same host also carried a production behavior-tree beacon protocol (ZMQ pub/sub with a JSON envelope through the message-skills relay) with a 29.2 ms one-way arrival floor.

A clock-skew-free pub/sub ping-pong measurement independently confirmed the floor. The minimum round-trip time over 250 samples was 55.4 ms ($\approx 2 \times 27.7$ ms one-way), agreeing with the one-way floor to within 1.5 ms. The residual NTP offset between the two hosts is bounded under 2 ms.

The microsecond-scale real-time publication path and the millisecond-scale inter-site control plane coexist on the same host with no measurable perturbation of the controller's cycle. The corresponding inter-layer statistical-independence measurement (a Pearson correlation between LLM inference latency and controller jitter on concurrent timing logs) is reported as a future validation item in Chapter 14.

The C++ core's bounded execution time is a function not only of the language choice but of the implementation discipline that surrounds it. Configuration structs reject defaults so that an invalid combination is caught at construction time rather than at run time. The hot-path classes follow the PIMPL pattern with copy deleted and move `noexcept`, so that the orchestration layer holds polymorphic references without aliasing risk. Runtime tuning hooks are exposed separately from constructors, so gain updates within a session do not require reconstruction. JSON-side configuration uses the same base-override deep-merge pattern as the YAML hierarchy of Chapter 9,

which keeps the C++ side aligned with the declarative configuration layer rather than carrying its own defaults. Cross-language ABI boundaries (the Python reader of the shared-memory buffer of Section 11.5.1, the pybind11 binding of the controller library of Chapter 12) use fixed-width unsigned integers for enum fields rather than vendor-specific sentinels, so that a Python consumer compiled against an older header still reads a well-defined value. These choices are not visible from outside the library, but they prevent the latency drift, orchestration race, and cross-version ABI break that would otherwise occur under repeated session lifecycle, holding the latency claims in this chapter and the controller-library claims in Chapter 12.

Safety Checker The safety checker enforces three per-joint constraints on a horizon of action chunks:

1. position limit, $q_{\min}(j) \le a(h, j) \le q_{\max}(j)$, applied at every horizon step h;
2. velocity limit, $\|(a(h) - a(h-1))/\Delta t\|_\infty \le v_{\max}$, applied through a finite-difference estimate of velocity;
3. acceleration limit, $\|(a(h) - 2a(h-1) + a(h-2))/\Delta t^2\|_\infty \le a_{\max}$, applied through a second finite difference.

On violation, the checker clamps the offending action and records it in a result object the calling Python service inspects. The hot path uses raw pointer arithmetic, performs no heap allocation after construction, and is verified by a unit test that asserts both correctness and a worst-case execution time bound.

Trajectory Engine The trajectory engine replaces Python's `time.sleep()`, which can exhibit jitter on the order of 1 to 5 milliseconds, with a `clock_nanosleep`-based timer that achieves 10 to 50 microseconds of jitter. It implements a smooth

interpolator with a jerk constraint, supports speed scaling for graduated deployment (typical schedule: 10% $\rightarrow$ 50% $\rightarrow$ 100% during commissioning), and operates in a receding-horizon mode in which the policy generates an H-step action chunk, the engine executes the first h steps, and the cycle repeats.

Image Preprocessor The image preprocessor wraps an OpenCV pipeline (resize, normalization, color-space conversion) consumed by the vision skills through a pybind11 import, built like the other C++ modules as a single wheel through scikit-build-core and pybind11 without any Python-level wrapping code beyond the `import` statement.

Reproducibility Provenance A version tuple of (package git commit, `pyproject` version string, host Eigen version) is recorded in every validation result file and in every real-time controller startup log. The tuple serves as the machine-readable provenance stamp for the numerical claims of this chapter and Chapter 12.

Zero-Copy Vision Transport (vision-core)

Problem with ROS2 DDS Distributing camera frames between processes through a ROS2 topic incurs three costs: image serialization, multiple memory copies, and the overhead of the underlying DDS transport. A single 640×480 RGB camera at 60 fps produces 52.7 MB/s. A multi-camera deployment quickly saturates the DDS infrastructure. The observed end-to-end latency is 30–300 ms depending on QoS settings.

This latency is incompatible with closed-loop control at 50–100 Hz, where the policy must consume the camera frame, infer an action, and command the robot within tens of milliseconds.

Shared-Memory Layout `vision-core` replaces the DDS topic with a lock-free shared-memory buffer that any number of consumer processes mmap directly: a 64-byte header (magic value, version, slot count, maximum frame size, atomically-updated active-index) followed by two slot descriptors (state EMPTY/WRITING/READY, color and depth metadata, offsets) and two data buffers in `/dev/shm` per camera (Figure 11.2). The producer writes one slot, atomically swaps the active index, and consumers read the previously active slot through their mmap mapping; there is no mutex, no contention, and no copy on either side.

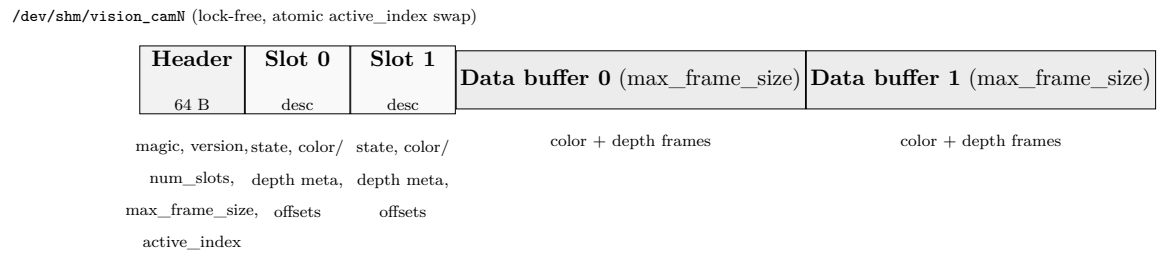


Figure 11.2: Shared-memory layout used by `vision-core`. The producer writes one slot and atomically updates the active index; consumers mmap the buffer and read the previously active slot, never copying frame data.

Performance Table 11.3 compares vision-core against a baseline ROS2 DDS topic transport on the same hardware. The shared-memory transport achieves sub-millisecond end-to-end latency with zero copies on either side, against the 30 to 300 milliseconds and three to four copies of the DDS baseline.

Table 11.3: Vision transport performance: vision-core shared memory versus ROS2 DDS topic baseline. †Mean publish-to-subscriber delay measured at the AI server site with two FLIR Blackfly S cameras (3072×2048) at 30.30 Hz, per-frame HEVC payload 8.47 KB; standard deviation 0.4 ms across the sample. DDS baseline comparison on identical hardware is identified as a validation item in Chapter 14.

Transport	Latency	Bandwidth	Producer copies	Consumer copies
ROS2 DDS topic	30–300 ms	overhead-bound	3–4	3–4
vision-core SHM	**∼1 ms**†	**mmap-limited**	**0**	**0 (mmap)**

Cross-Chapter Use The vision-core transport is used in three dissertation chapters. Chapter 12 (DIP) consumes camera frames at 50 Hz from the same buffer that the multi-camera HEVC encoder pipeline of Chapter 13 reads, and the vision skill of Section 11.5.2 consumes the same buffer for SAM2 segmentation.

Together with the uniform addressing substrate of Section 11.5.2, the vision-core library is the perception-side load-bearing analog. A thin ROS2 adapter on top of the shared-memory ring buffer republishes a uniform `/vision/{src}/...` address space across vendor SDKs (Intel RealSense, FLIR Spinnaker, ZED). Every chapter that consumes camera frames writes its perception logic against the source identifier rather than a vendor-specific message type.

The Argonne deployment runs four sensors continuously: two FLIR Blackfly machine-vision cameras on the AI workstation, and two Intel RealSense D435 depth sensors on the robot-controller workstation. A hardware-accelerated HEVC path keeps per-camera CPU utilization below ten percent. The laboratory's camera array provisioning and multi-workstation network supply the heterogeneous sensor plane that this perception substrate wraps. The shared-memory layout, HEVC encode path, and uniform addressing substrate connect that plane to the multi-camera multi-user demonstrations of Chapter 13 through the configuration hierarchy of Chapter 9.

11.5.2 Modular Architecture

Multi-Provider LLM Abstraction

The Provider Heterogeneity Problem The four providers that own-code supports differ along every axis that matters for integration:

1. Claude exposes a high-level conversation API with built-in tool calling through its official agent SDK;
2. Codex's command-line interface, intended for developer-style use, requires spawning a subprocess and parsing structured output;
3. Gemini exposes a Python SDK whose function-calling interface conflicts with its built-in code execution feature, requiring an external workaround;
4. Ollama runs locally and exposes an HTTP API but no high-level agent SDK at all.

Authentication ranges from OAuth subscription tokens (Claude, Codex) to API keys (Gemini) to no authentication (Ollama), and rate limits range from effectively unlimited (Claude and Codex subscriptions, Ollama local) to five requests per minute (Gemini free tier). A naive integration that uses each provider's SDK directly produces four parallel codebases.

The Common Substrate own-code replaces the four-codebase outcome with a single asynchronous abstract base class from which all four providers inherit. The base class owns three responsibilities:

1. a permission checker that decides tool-call admission under user policy;
2. a WebSocket manager that streams tokens to the frontend;

3. a cancellation token that interrupts generation mid-stream.

Each concrete provider implements only the small generation method its SDK requires; cancellation, model swapping, permission checking, and WebSocket streaming are written once in the base class and shared across the four.

The four concrete subclasses are correspondingly thin (Figure 11.3):

1. the Claude adapter delegates to the official agent SDK;
2. the Codex adapter spawns a subprocess and parses its output;
3. the Gemini adapter calls the Google generative AI SDK and dispatches tool calls through the tool engine, bypassing the SDK's internal function-calling pathway;
4. the Ollama adapter calls a thin local engine wrapping the Ollama HTTP API.

The same frontend binds to any of the four through a backend selector, so the user swaps providers mid-session without losing conversation state.

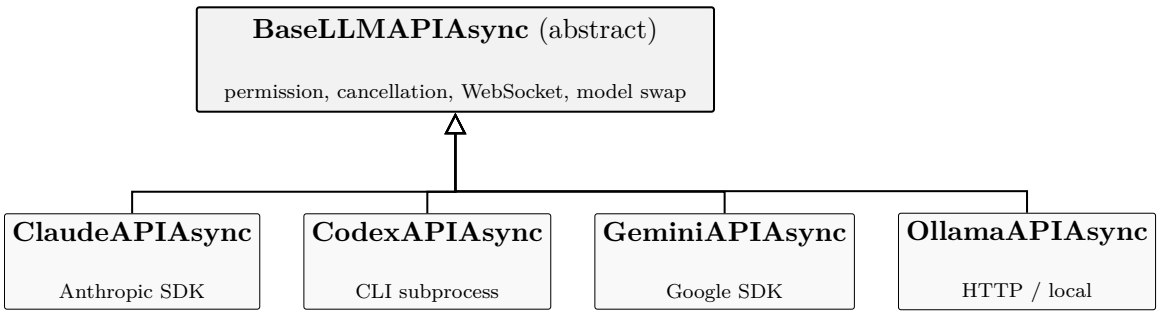


Figure 11.3: Inheritance hierarchy of the four provider adapters. Each subclass implements only the provider-specific generation method; cancellation, permission checking, WebSocket streaming, and model swapping are inherited from the abstract base.

Tool Engine

Motivation Before the tool engine was extracted as an independent module, tool execution was coupled to the Ollama adapter, and adding tools for the other providers required reimplementing the dispatch logic three more times.

A refactoring in April 2026 extracted the dispatch logic into a single `ToolEngine` class that all four providers use through dependency injection. After the refactor, adding a new tool requires writing it once; all four providers gain access automatically.

Tool Engine Architecture The tool engine is composed of two registries: a built-in tools handler exposing seven filesystem and shell operations (read, write, shell, grep, glob, ls, replace) for general-purpose agent work, and a tool registry exposing the remaining tools backed by the skill services of Section 11.5.2.

The engine provides three uniform operations (Figure 11.4):

1. return tool schemas in the format the specified provider requires;
2. execute a tool call by name with a payload and an execution context;
3. format the result into the response format the calling provider expects.

Permission Model A separate permission checker decides whether a given tool call is allowed under the current policy. Permissions are scoped per tool, per user tier, and per session, and the checker supports a dry-run mode that logs the call without executing it. The permission layer is the boundary between the agent's autonomy and the user's control, and is the same conceptual mechanism that the multi-level approval system of Chapter 10 (System 9) operates at the agent-coordination layer.

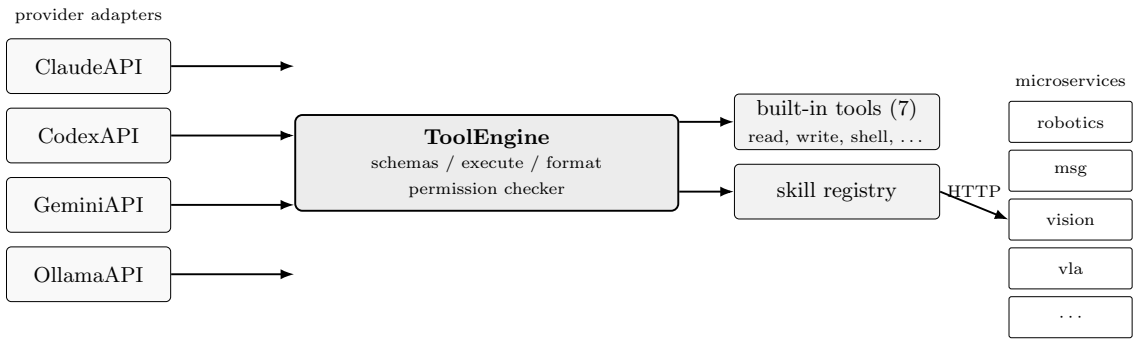


Figure 11.4: Tool engine dispatch path. Each of the four provider adapters calls the same `ToolEngine` through dependency injection; the engine owns a permission checker, a built-in tools handler (seven filesystem and shell operations), and a skill registry backed by the microservices of Section 11.5.2. Adding a new tool requires writing it once; all four providers gain access automatically.

Skills Platform

The skills platform is an extensible collection of independent microservices, each exposing an HTTP REST interface and, where relevant, a WebSocket. Each service holds its own `pyproject.toml` and dependency graph, runs in its own container or bare process, and registers a manifest with the tool engine at startup so that the engine translates each tool call into an HTTP request to the corresponding service. Independence is enforced at the build level: a broken dependency in one skill cannot affect another, and an upgrade to one skill does not require touching any other service.

Table 11.4 maps the skills consumed by other dissertation chapters to the architectural function they provide at the consumption point. The skills described below are:

1. bilateral teleoperation through a leader–follower web interface used in Chapters 13 and 12;

2. asynchronous messaging through the two-layer ZMQ + Zenoh bus that connects every laboratory device through the VPN (Corsaro, 2022);

3. persistent memory exposed through the author's Obsidian vault and consumed by the agent layer of Chapter 10;

4. multi-device mixed reality orchestration that deploys head-mounted-display clients used by Chapters 7 and 13.

The vision processing band hosts SAM2 and SAM3D segmentation with photogrammetry, vision-language-model perception routes, and HEVC encode/decode for the diffusion-interaction policy of Chapter 12 and the agent layer of Chapter 10; the vision-language-action pipeline (*See* $\rightarrow$ *Think* $\rightarrow$ *Act*) stages perception, reasoning, and action emission for the same consumers. Skills outside the demonstration critical path—office automation, research lookup, document review, browser automation, Jupyter notebooks, and diagram generation—are documented in Appendix K.

Table 11.4: Cross-chapter usage map of the skills platform. Skills outside the demonstration critical path (`office`, `research`, `review`, `browser`, `jupyter`, `diagram`) are not reused by other chapters and are listed in the platform but omitted from this map.

Skill	**Consumed by**	**Architectural function at consumption point**
`robotics`	Chapter 13, Chapter 12	Bilateral teleoperation web UI and BT-take orchestrator for the demonstration runs.
`msg`	Chapter 9, Chapter 12, Chapter 13	Voice channel for operator dispatch and the unified messaging substrate that replaces the earlier ZeroMQ broker.
`xr`	Chapter 7, Chapter 13	Multi-device deployment of the see/move/follow/owner/mapping policy tuple to head-mounted clients.
`obsidian`	Chapter 10	Persistent-memory substrate for the agent layer's long-term context.
`vision`, `perception`, `video`	Chapter 12, Chapter 10	Perception inputs to the diffusion-interaction policy and VLM routes for the agent layer.
`vla`	Chapter 12	See–Think–Act staging for the policy pipeline.

Bilateral Teleoperation Through a Leader–Follower Web Interface The robotics skill exposes a browser-based bilateral teleoperation interface with a leader–follower split view.

The leader side is configurable as a slider control, a PHANToM haptic device, or a Foxglove live feed. The follower side is a slider, a Drake simulation, an Isaac Sim instance, a MuJoCo simulation, or a real robot through Foxglove.

The user loads a YAML scenario; the interface advances through phases; force feedback is displayed in real time. State persists to JSON files keyed by site and robot pair (for example, `teleop-state-franka-kinova.json`), so an interrupted session resumes. The same web interface collected the bilateral teleoperation data of Chapter 13 and operated the Drake validation runs of Chapter 12.

A complementary host-side command-line suite supplements the in-process tool engine. Five commands cover the behavior-tree pipeline (Table 11.5); they share the Mongo event store of Section 11.5.2 as the persistence target.

Table 11.5: The five host-side behavior-tree pipeline commands and their persistence targets. Each command operates against the shared Mongo event store of Section 11.5.2.

Command	Function	Persistence target
`bt-bringup`	Driver and URCap lifecycle (start, stop, restart, status, list, logs, verify, doctor)	Mongo `host_state` events
`bt-take`	Take orchestration: BT scenario, dual data recording, and synchronized metadata (run, stop, cleanup, list, show)	Mongo `runs` + recorded data files
`bt-post`	Post-processing of recorded data pairs into video and 600 DPI publication plots (mp4, verify, probe, analyze, all)	local PNG and MP4 artifacts
`bt-events`	Mongo events query and Markdown / JSON / JSONL export (list, failed, show, timeline, export, doctor)	Mongo read-only + sovereignty exports
`bt-beacon`	Cluster discovery on the `mr_panel` channel (start, stop, clear, status, peers, test)	Mongo `peer_state` events

Messaging Capability: Voice and Zenoh Transport The msg skill replaces a previously deployed voice container with three tightly coupled functions: a Zenoh (Corsaro, 2022) pub/sub session that carries voice and tool-call topics, a Whisper-based speech-to-text pipeline, and a REST relay on port 9084 that exposes both to other skills. The persistent voice agent (*Riley*) runs on Ollama with a small model and a 1024-token budget, and reaches every robot device on the laboratory network through the VPN overlay.

A user utterance is captured locally and transcribed by Whisper. Riley receives it through the Zenoh session, expands it into a tool call via the tool engine, and publishes the call over Zenoh to the target robot. The robot executes the call under the supervision of the permission checker.

In an end-to-end dispatch test on a small open-weight backend, the agent issued the correct tool call on four of six wearer utterances directly. It recovered a fifth through an explicit confirmation turn. It missed one diagnostic utterance, requiring mitigation through prompt re-ordering or a larger backend.

The Zenoh-based transport replaces an earlier ZeroMQ XSUB/XPUB broker, unifying the messaging substrate with the cross-distribution Zenoh bridge of Chapter 9 and removing the need for a separate broker process.

On the Unity client (HoloLens 2, Quest 3, and editor builds), the same two-layer bus is hooked through a single `OwnCodeBridge` component that subscribes to the Zenoh transport for default channels and to a ZeroMQ lane for voice-specific routing; a peer registry tracks each host's beacon liveness on a fifteen-second time-to-live and emits join, update, and graceful-leave events to the rest of the application.

For XR clients that cannot run a VPN client and cannot reach the broker through static configuration—HoloLens 2 and Quest 3 head-mounted devices in particular—a UDP broadcast beacon on port 9999 announces the broker endpoints every three

seconds, and an XR-side discovery component connects the ZmqBus automatically upon receiving the announcement. This replaces a previously hardcoded ScriptableObject configuration source and removes the per-deployment-site rebuild that the static configuration required.

Persistent Memory The obsidian skill exposes the author's Obsidian vault as a REST API on port 9082. LLM agents read and write the vault directly: the Claude Code auto-memory feature uses it to persist user preferences, Riley uses it for long-term context, and the dissertation working drafts that were the source material for this chapter live inside that same vault.

XR Skills: Multi-Device Mixed Reality Orchestration The xr skill is the orchestration substrate that the multi-user mixed reality architecture of Chapter 7 and the head-mounted demonstrations of Chapter 13 are deployed through. It exposes a single HTTP and WebSocket surface that abstracts three vendor-specific control planes behind a uniform tool interface: the HoloLens 2 Windows Device Portal REST API, the Quest 3 Android Debug Bridge (ADB), and a Model Context Protocol bridge to the Unity Editor on the build host. From the agent's perspective, deploying a new mixed reality client to either head-mounted device, building a Unity scene, or rolling out a session policy across a heterogeneous fleet of clients is a single tool call regardless of which vendor's runtime is at the other end.

The skill organizes its operations along four axes:

- *Device lifecycle*: tools for device discovery (`scan`, `detect`, `device-types`), state inspection (battery, network, session health), and power management (`wake_up`, `keep_awake`) operate uniformly across HoloLens and Quest.

- *Application lifecycle*: `apps/install`, `apps/launch`, `apps/stop`, and their bulk `-all` variants deploy and orchestrate the mixed reality client across every connected device in a single call, which is the mechanism by which Exemplar E3 of Chapter 13 reaches an arbitrary number of head-mounted clients without per-device scripting.

- *Unity Editor integration*: the `unity_mcp/build_and_deploy_hl2`, `unity_mcp/uwp_msbuild`, and `unity_mcp/winappdeploy` endpoints drive the Unity Editor on the build host through the Model Context Protocol, so that an agent can rebuild and redeploy the OpenXR client without leaving the language-model conversation.

- *Mixed reality session policy*: `mr/deploy_policy`, `mr/infra_ready`, and `mr/verify_reception` carry the see/move/follow/owner/mapping policy tuple of Chapter 7 from the configuration hierarchy of Chapter 9 to the connected clients and verify that each client received the intended role assignment before the session begins.

Live media handling is integrated alongside the orchestration plane. The skill exposes Mixed Reality Capture stream URLs and a JPEG proxy stream for HoloLens, an MJPEG endpoint for Quest, and a Wi-Fi casting control pair (`quest/start_casting`, `quest/stop_casting`) that lets a supervisor monitor head-mounted views from the same dashboard that drives the agent layer of Chapter 10. Recording control is exposed through `recording/bulk` so that synchronized take recording across the fleet, used by the bilateral teleoperation demonstrations of Chapter 13, is a single call rather than a per-device coordination script.

xr-skills absorb the multi-device, multi-vendor, multi-runtime nature of head-mounted mixed reality at this layer. The agents of Chapter 10, the policy tuple of

Chapter 7, and the demonstration scripts of Chapter 13 interact with a single uniform surface, and vendor differences (HoloLens Windows vs. Quest Android), runtime differences (Unity Editor vs. deployed UWP/APK), and connection differences (USB ADB vs. Wi-Fi ADB vs. HoloLens device portal over the LAN) localize inside the skill and never leak upward.

Persistence Layer (own_store + MongoDB)

An abstraction analogous to the vision-core perception substrate exists on the persistence side: `own_store`, a 14-file package in own-code's `lib/own_store/` with its own `pyproject.toml`.

The interfaces are two abstract base classes, `Engine` and `Store`, each with synchronous and asynchronous variants. The index abstraction is a backend-agnostic `IndexSpec` dataclass. The MongoDB backend supplies concrete implementations: `MongoEngine`/`MongoEngineAsync` and `MongoStore`/`MongoStoreAsync`. The package exposes 24 public symbols and depends only on `pymongo` and `motor` at runtime.

Three application classes inherit from the MongoDB concrete classes:

1. chat persistence through `SessionStore`, `MessageStore`, and `ProjectStore`;
2. browser session recording through `RecordingStore`;
3. the task-contract framework that drives the demonstrations of Chapter 13 through `BTRunStore` and `BTEventStore`.

This inheritance is the only inter-system coupling.

A 62-test `pytest` suite covers the package against a live MongoDB container. It exercises connect/disconnect lifecycle, fail-fast and graceful-degrade modes, index idempotency, and the bulk batcher's count and time-window flush triggers.

Distributed Network Provider Substrate

The cross-site, multi-vendor mechanisms that make the skills platform usable across heterogeneous robot fleets consolidate into a single 24/7 container, `multi-hra-network-provider`, running on every cluster site (Figure 11.5). The container hosts uniform addressing across vendor drivers, endpoint resolution through the `ConnectionResolver` introduced in Chapter 7 and extended in Chapter 9, cross-site transport on the VPN overlay, per-channel admission driven by the active task-contract topology, behavior-tree task-contract orchestration, and a top-level event-history collection on MongoDB.

The substrate's design rule is that the operation wrapper is a swappable layer above the controller compiled object. The wrapper-orthogonal recording case study of Section 11.6.1 quantifies the rule.

Uniform addressing across heterogeneous drivers. The interaction policy of Chapter 12 operates on a tuple of robots indexed by identifier r. Each robot must expose the same observation and command channels regardless of vendor driver. The drivers do not provide this empirically.

A *transport-relay* layer addresses the gap by subscribing to the driver's native state channels (joint state, end-effector wrench, controller status), re-publishing them under a uniform `{r}/...` namespace, and exposing a parallel set of command channels that forward into whichever vendor-specific interface the driver expects. The relay is stateless (no filtering, estimation, or rate change), the narrow waist at which higher-level policy decouples from driver implementation.

The bilateral teleoperation demonstrations of Chapter 13 compose five manufacturer-distinct manipulators (Universal Robots, Franka Emika, Kinova,

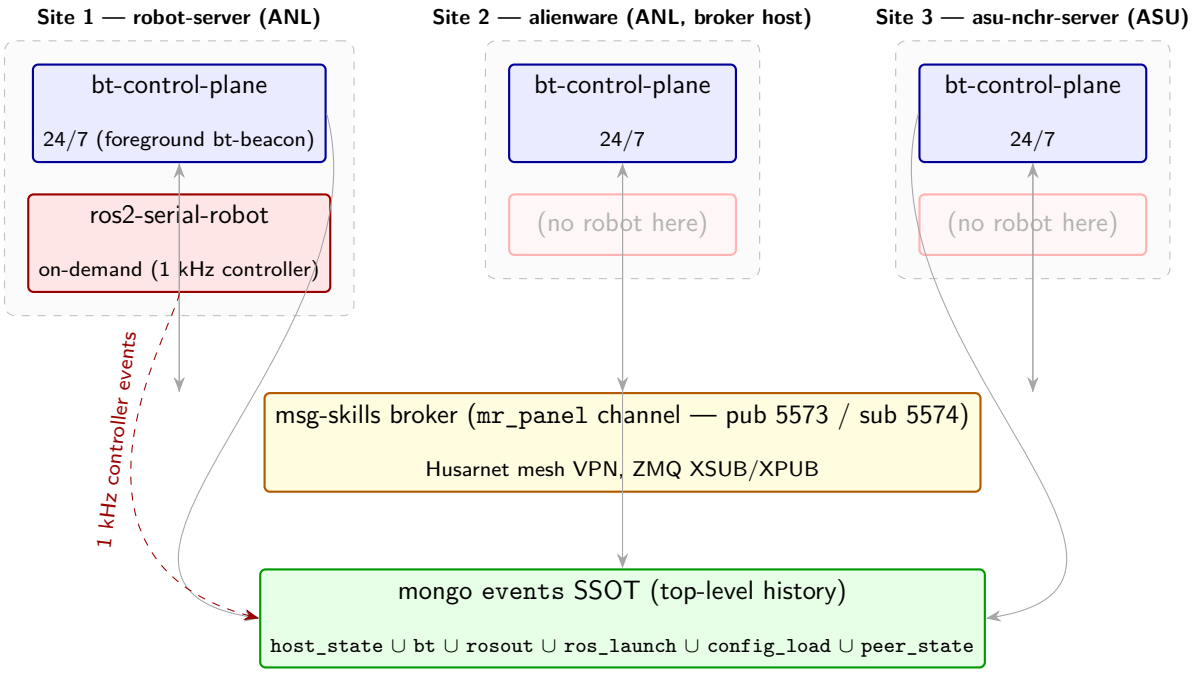


Figure 11.5: High-level architecture of the network provider substrate. Each cluster site (here three: `anl_robot`, `alienware`, `asu_nchr`) runs a 24/7 lightweight `multi-hra-network-provider` container that publishes `bt/beacon` heartbeats on the `mr_panel` channel of the msg-skills broker; only sites with a controllable robot also bring up the on-demand `ros2-serial-robot` container. All container lifecycle events accumulate in a single MongoDB `events` collection (entries tagged by source: `host_state`, `bt`, `rosout`, `ros_launch`, `config_load`, `peer_state`).

UFactory xArm, UFactory Lite) under a single behavior-tree specification. The composition is a direct consequence of the substrate, not of the behavior-tree engine.

Endpoint resolution. Skills query the `ConnectionResolver` gateway introduced in Chapter 7 (Appendix D) and extended in Chapter 9 (Section 9.5.1) for the resolution of site, robot, and service identifiers to concrete network endpoints. The substrate adapts the resolver to a cluster-aware HTTP query surface at port 9083 (`/api/cluster/peers`) on the network-provider container, where peer membership is maintained from the `bt/beacon` stream and exposed to the rest of the platform through a single REST endpoint.

Cross-site transport. For cross-site deployments where ROS2 DDS multicast cannot traverse the network overlay, the in-process ZMQ broker that the skills platform already operates additionally serves as the inter-site transport: an IPv6 overlay carries the broker traffic between the on-site robot workstation and a remote rendering host, and one hundred consecutive frames of joint-state telemetry traversed this path without loss during the demonstration integration of Chapter 13. The cross-site overlay used in the present deployment is a low-latency peer-to-peer VPN, substituted for an earlier cloud-relay design after the deployment environment exposed an outbound-blocking firewall.

Admission and staleness. Uniform addressing alone does not bound which named robots may participate in a given session. The substrate maintains a per-channel admission allowlist driven by the active task-contract topology: when a scenario is loaded, the substrate extracts the participants section, pushes the identifier set to the broker, and the broker drops any incoming message whose declared `entity_id` is outside the set before any state mutation. A monotonic `rejected_count` field on the broker health endpoint surfaces rejection traffic for operators. Distinct from admission, the substrate maintains a per-entity *staleness* field that records the age of the most recent message; staleness is a passive observation rather than an eviction policy, because robots may be deliberately silent during long deterministic operations. Combining admission and staleness into a TTL eviction policy would flap under load; keeping them separate gives the operator two precise signals at the cost of a few extra fields on the health endpoint.

Task-contract orchestration. The unit of orchestration is a named *task contract*: a topology entry that binds leaf actions, control-flow operators, and default arguments to an identifier (for example, `ur16e_bringup_bt`).

The contract identifier is the primary key throughout the system. The lifecycle wrapper indexes its state files by contract name. The runtime emits events tagged with a `run_id` of the form `<contract>_<host>_<timestamp>`. The Mongo persistence layer stores both the `runs` and `events` collections under that key. The take orchestrator (`bt-take`) produces post-processed records under the same identifier.

The orchestration plane is structured as a behavior-tree (BT) executor in the sense of Colledanchise and Ögren (2018) and Colledanchise and Ögren (2022), with the standard finite-status alphabet (RUNNING / SUCCESS / FAILURE) returned by every tick and the standard control-flow operators (sequence, parallel, fallback, retry) acting on subtree composition. Control-flow operators externalize into the configuration hierarchy of Chapter 9, so subtree composition selects a contract instance through configuration alone, and each leaf action follows an idempotent re-entry pattern in the sense of Fielding (2000), which reduces operator-driven recovery to a minimal-state restart parameterized only by the last contract identifier and its argument set. Runner phase, shell, process, launch, and log transitions are surfaced as event-carried state-transfer messages (Hohpe and Woolf, 2003) on a single broker channel, consumed by the Mongo persistence layer, the language-model partner's diagnostic context, and the head-mounted-display client.

Two-plane container architecture. The substrate partitions its runtime into two containers along the cycle-rate boundary (Figure 11.5).

The *real-time plane* is the `ros2-serial-robot` container: the heavy ROS2 stack with manipulator drivers and `poe_robotics`. It hosts the 1 kHz `ros2_control` cycle

of Section 11.5.1. It launches only on sites with a controlled robot, and only for the controlled session.

The *control plane* is the `multi-hra-network-provider` container: ∼120 MB on a `python:3.11-slim` base, running continuously on every participating host. It carries the lifecycle daemons: the beacon publisher and subscriber, the Mongo client for the event store, the bus subscriber for the `mr_panel` channel, the `ServiceBeacon` UDP discovery publisher, and the `TopologyService` HTTP endpoint at port 9083.

The two containers do not share Docker socket access. The control-plane container's smaller image and absence of robot drivers makes it cheap to run on every site, including sites that do not host a controllable robot. The container boundary prevents the lifecycle plane from perturbing the controller jitter measured in Section 11.5.1: the schedules execute in different containers.

Deployer overlay pattern. The substrate's source files contain no site label, broker hostname, or institutional identifier. The upstream `multi-hra-robots` repository is site-agnostic: each cluster site supplies its identifiers through a four-line `.env` overlay (`MULTI_HRA_ROBOTS_DIR`, `BT_BEACON_SITE`, `BT_BEACON_HOST`, `BT_MONGO_URI`) loaded by the container's entrypoint. An open-source release of the substrate is therefore possible without scrubbing site-specific identifiers from the source: a new site joins the cluster by `rsync` of `lib/cli/` together with the four-line overlay alone, with no source-code edit.

Top-level history concentrated in MongoDB events. Container lifecycle events, beacon announcements, configuration loads, and BT contract events all record into the same `events` collection through the source-tagged schema of Section 11.5.2. Lifecycle

entries take the form `host_state.started` on launch and `host_state.stopped` on graceful shutdown.

The schema reserves a top-level `source` field with values `bt`, `rosout`, `ros_launch`, `custom`, `host_state`, `config_load`, and `peer_state`. The same query surface answers questions about runtime behavior, configuration drift, and cluster-membership changes uniformly.

Ad-hoc verification commands (file-hash diffs, manual port scans) are unnecessary for cross-host sync detection. Each host's beacon payload carries a `provenance.git_sha` (the deployed `multi-hra-robots` commit) and a `provenance.config_sha` (the loaded `control_plane.yaml` content hash). A single Mongo query on the latest `bt/beacon` entries reports whether all sites agree on source revision and configuration.

Three-host cluster verification (2026-04-29). A 3-host cluster spanning Argonne (`anl_robot` on the robot host, `alienware` on the broker host) and Arizona State (`asu_nchr` on the ASU host) sustained bidirectional beacon announces (5 s heartbeat, 15 s TTL) over the mesh VPN. All three sites reported an identical `provenance.config_sha` value in their `bt/beacon` payloads. The deployer-overlay pattern carries the cluster to a new site by `rsync` of the upstream files and a four-line `.env` alone, with no source-code modification.

The two-plane container architecture meant ASU joined as a control-plane participant only. `ros2-serial-robot` was not deployed there. Cluster membership and robot ownership decouple as intended.

Bringup substrate stabilization. The bringup substrate that hosts the operation wrappers of Section 11.6.1 is stabilized by a cascade of four fixes, checked at session

start by a single `bt-bringup doctor` command. Each fix targets a failure mode that compounds with the others; none is sufficient on its own.

1. *Re-entrancy in the ROS2 launch lifecycle.* The in-process event emitter dispatches during the launch setup phase and terminates the child processes of the launch. The fix is an external launch-coordination process that separates event emission from the launch lifecycle.

2. *Process-tree orphans on stop.* The launch leaves descendant processes when the parent exits, and those descendants race the next contract's processes. The fix is an explicit reaper that claims the descendants at stop time.

3. *Zombie accumulation in the deployment container.* Terminated children are not reclaimed by the container's init process and accumulate in the kernel process table. The fix is a PID-1 reaper inside the container that performs the reclaim.

4. *Sticky failure at the URCap boundary on the vendor controller.* A stop-load-play cycle on the URCap (Universal Robots Cap, the UR controller's plugin layer) leaves the vendor-side state in an inconsistent configuration, surfacing as a persistent failure at the operator interface. The fix is an idempotent re-entry check at the URCap boundary that guards the vendor-side state.

The cascade as a whole holds the substrate stable across recurring contract restarts.

Browser-side multi-user inspection. The substrate exposes a browser-side interface (Figure 11.6) that renders the per-user policy assignment of an active scenario without instantiating a task contract. The widget queries the operation-plane container through the same `/api/sessions/list` endpoint that the contract loader consumes, and reads the see/move/follow/owner/mapping policy directly from

`cards/<scenario>/bt_topology.yaml`. The displayed assignment matches the policy yaml field-for-field, with no runtime translation between policy authoring and operation.

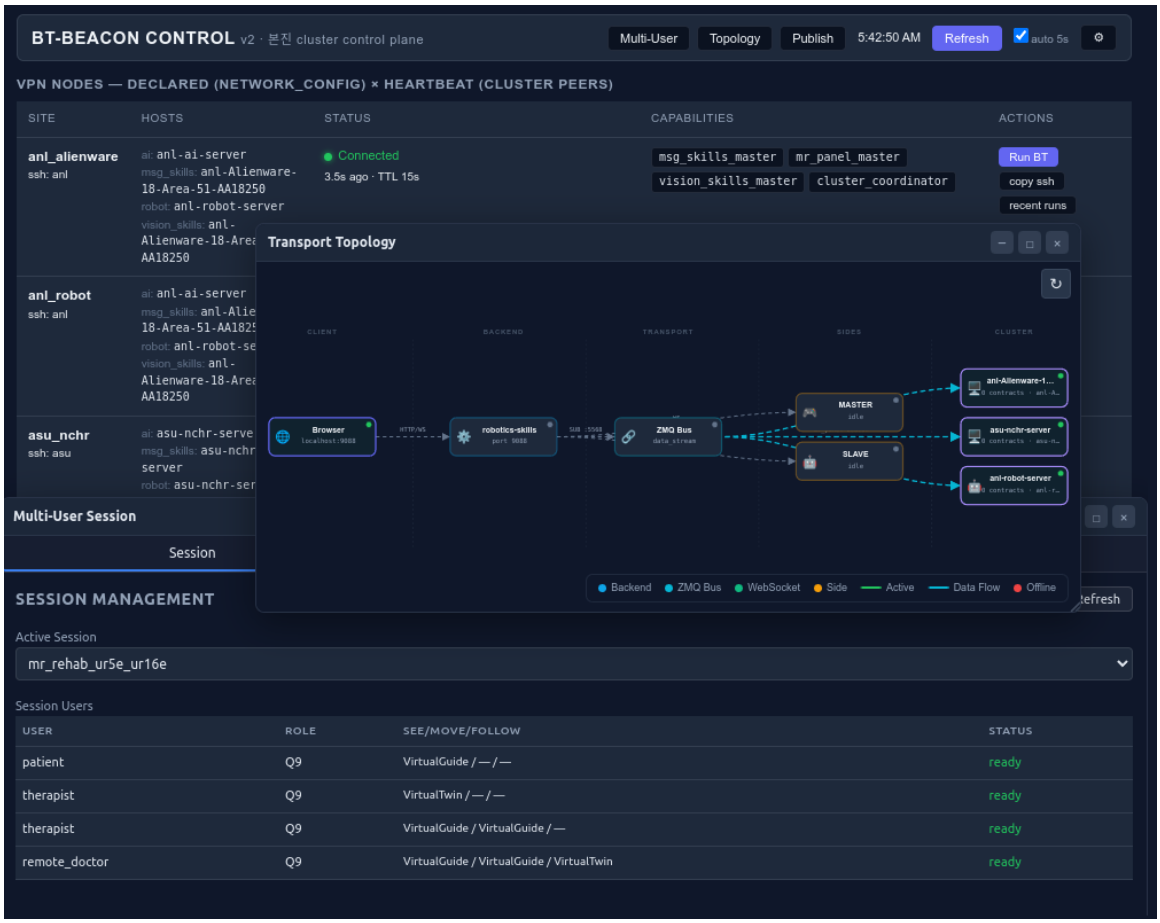


Figure 11.6: Browser-side BT control-plane interface (Multi-User Session widget) displaying the per-user policy assignment for an active multi-user scenario. The widget consumes the policy tuple from `cards/<scenario>/bt_topology.yaml` via the `/api/sessions/list` endpoint of the operation-plane container, and renders one row per assigned user with the role, robot mapping, and visualization-control split that the yaml encodes. The Korean text in the header ("*bonjin*", meaning "main camp") is an internal cluster identifier; substitute the cluster name of any deployment.

11.6 Results

This section reports two validation studies for the own-code substrate. Section 11.6.1 measures the reproducibility of the operation wrapper layer, and Section 11.6.2 maps the multi-agent decision-support application onto own-code modules.

11.6.1 Operation Wrapper Reproducibility Study

The same compiled controller object can be invoked through different operation wrappers: the legacy host shell that issues a sequence of shell commands, or the BT-aware command-line interface `bt-take` introduced in Section 11.5.2. The reproducibility test asks whether the wrapper choice perturbs the resulting end-effector motion and joint-tracking error beyond a per-platform noise floor.

The reproducibility test paired two wrappers driving the identical UR16e platform within a 96-hour window. The legacy host shell wrapper recorded the first take on 23 April 2026; the BT-aware command-line interface `bt-take` recorded the matching take on 27 April 2026. Both runs drove the same six-waypoint diamond contract under an automatic target source, a hardcoded waypoint generator that supplies target poses without operator-applied wrench. The automatic source isolates the wrapper layer as the only experimental variable: the controller, robot driver, force/torque sensor, and data recorder operate at identical real-time fidelity in both runs. The `poe_robotics` commit hash and the controller gain set were also pinned across runs. Table 11.6 reports the geometric and joint-tracking deltas.

Table 11.6: Operation wrapper reproducibility on UR16e: legacy host shell (2026-04-23) vs. BT-aware command-line interface `bt-take` (2026-04-27). Geometric metrics reproduce to within sub-millimeter; joint-trajectory controller tracking improves marginally under the BT-aware lifecycle. The wrapper-induced noise floor on this platform is below 0.05 mm in end-effector excursion and below 0.25 mrad in RMS joint-tracking error.

Metric	**Legacy shell**	**BT-aware CLI**	**Δ**
Motion duration	10.04 s	10.05 s	+0.01 s
ΔX peak excursion	198.79 mm	198.78 mm	−0.01 mm
ΔY peak excursion	198.75 mm	198.77 mm	+0.02 mm
ΔZ drift	0.51 mm	0.62 mm	+0.11 mm
sh_pan peak $\lvert v \rvert$	12.74 °/s	12.70 °/s	−0.04 °/s
JTC sh_pan RMS error	4.183 mrad	3.942 mrad	−0.24 mrad
JTC sh_pan peak error	7.578 mrad	7.126 mrad	−0.45 mrad

The corresponding raw evidence panel for one such take is shown in Figure 11.7.

One operational behavior changed with the BT-aware wrapper. The data recorder captured the session-start trigger event (one message at $t = 17.51$ s; motion window began at $t = 19.63$ s), which the legacy wrapper missed because the recorder started subscribing after the trigger message had already been published. Post-processing alignment of the trigger timestamp is therefore no longer required.

11.6.2 Multi-Agent Decision-Support Integration Cost

The Multi-Agent AI Platform of Chapter 10 is realized as a composition of own-code components. The application context is described in Chapter 10 and the cited paper (Chang *et al.*, 2025). This section reports the mapping onto own-code modules.

The agents run on Ollama through the multi-provider abstraction of Section 11.5.2. Llama-3 (Meta, 2024; Ollama, 2024) serves as the inference backend and `mxbai-embed-large` serves as the embedding model (Lee *et al.*, 2024c). The full pipeline runs on a single workstation without external cloud API access, eliminating both per-request cost and provider-side rate limits. The orchestrator is implemented

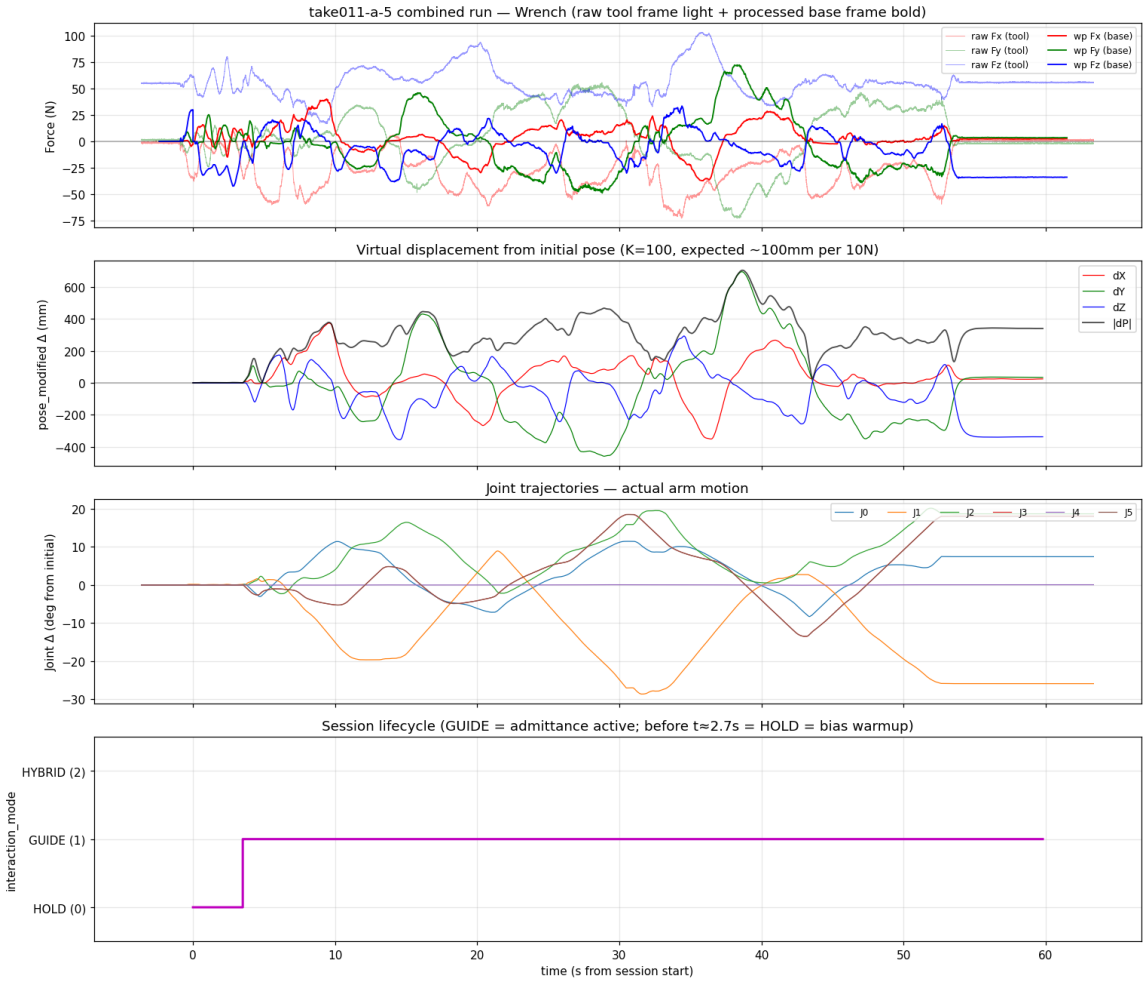


Figure 11.7: Combined evidence panel from a take used in the wrapper-reproducibility study: end-effector trajectory and the corresponding controller signals captured during the same six-waypoint diamond contract. The two wrappers produce indistinguishable kinematic plots at this scale; the quantitative deltas in Table 11.6 are below the visual resolution of this panel.

by the agent manager in `server/api/agents_api_async.py`, and Table 11.7 summarizes the component mappings. The integration cost is two domain-specific configuration artifacts (agent prompts and retrieval corpora) and zero infrastructure code lines; application-side measurements (decision throughput, retrieval recall, comparisons against cloud-API baselines) are reported in Chapter 10.

Table 11.7: Mapping from the multi-agent decision-support application of Chapter 10 onto own-code modules.

Component	**Role**	**own-code module**
Orchestrator	Query routing and report synthesis	agent manager (`agents_api_async.py`)
RAG retrieval	Document grounding for each agent	obsidian skill + research skill
Perception agent	Visual scene understanding	vision skill, screen-viewer service
LLM backend	Inference for all agents	multi-provider abstraction (Section 11.5.2)
Dashboard UI	Operator-facing visualization	robotics-skills HTML/JS pattern

11.7 Discussion

11.7.1 Empirical Validation

The same-kernel design rule (the controller compiled object is shared between the Drake validation environment of Section 11.5.1 and the real-time ROS2 controller process, with no re-implementation) extends to the operation wrapper layer: the wrapper choice does not perturb the end-effector motion beyond the noise floor reported in Table 11.6 of Section 11.6.1.

Table 11.8 summarizes the system's quantitative footprint: the structural counts (skill services, providers, MCP bridges, tools) are stable, while the latency and bandwidth values are reported from informal benchmarks on the deployment workstation, with a formal benchmark against a ROS2 DDS baseline listed as a future validation item in Chapter 14.

Table 11.8: Quantitative footprint of own-code. Latency and bandwidth numbers are reported from informal benchmarks on the deployment workstation. †Indicative line-of-code count from a one-off `cloc` run; a clean measurement is deferred to the validation step described in Chapter 14.

Metric	Value	Note
Total source lines of code	∼60,000†	Python + C++ + JavaScript
Skill services	extensible set	each independently deployed; new services added as the platform grows
MCP bridges	3	Unity, Isaac Sim, Obsidian
Provider integrations	4	Claude, Codex, Gemini, Ollama
Safety check latency	1–5 μs	C++ vs. 50–200 μs in Python
Vision transport latency	<0.2 ms	SHM vs. 30–300 ms DDS
Voice STT latency	<500 ms	local Whisper
Multi-camera bandwidth	100+ MB/s	zero-copy mmap

A failure mode encountered during humanoid platform integration validates the substrate boundary: a developer bypassed `vision-core` by linking the consumer code directly against the camera vendor library, which introduced a binary incompatibility between the `librealsense2` version expected by the consumer and the version available in kernel-space and switched the on-board filesystem to read-only before any perception code could fail in software; the substrate-mediated path centralizes the version coupling at a single library boundary and avoids this class of failure. Operationally, the system has been used by a single user (the author) for daily research workflows and has survived two episodes of provider API drift since the multi-provider abstraction was put in place, with each change requiring edits only to the affected provider adapter and no disturbance to skill services or downstream consumers.

11.7.2 Comparison with Existing Frameworks

LangChain, LlamaIndex, AutoGen (Wu *et al.*, 2023), and CrewAI (CrewAI, 2024) provide comparable provider abstractions and tool-calling layers. They are designed for application development rather than for robotics research, and do not include a

real-time C++ hot path or a zero-copy shared-memory vision transport. Conversely, robotics middleware such as ROS2 (Macenski *et al.*, 2022) provides the real-time and transport layers but does not include a multi-provider LLM abstraction or a tool-calling registry. own-code addresses the intersection of these two requirements through the operation-layer substrate of Section 11.5.2, whose behavior-tree orchestration plane (Section 11.5.2) binds the multi-provider runtime, the skill services, and the real-time controller into a single configurable session.

11.7.3 Limitations

own-code has four limitations that bound its claims:

- The deployment is single-user; the system is not engineered for fleet operation, and several services would require hardening (authentication, rate limiting, observability) for multi-user use.
- The C++ core requires manual rebuilding when a new robot is added, because each robot's safety limits and trajectory parameters live in C++ headers; a future revision will move these into the YAML configuration layer.
- LLM provider API changes still require occasional adapter updates, although the layered design contains the impact to a single file.
- The safety checker is verified informally through unit tests and not through formal methods; the implications for safety-critical deployment are discussed in Chapter 14.

11.7.4 Generalization and Cross-Chapter Reuse

own-code uses four design patterns reusable in other research settings: an async base class for provider abstraction, a tool engine decoupled from any single provider,

microservice skills behind a uniform manifest interface, and a zero-copy shared-memory vision transport. The substrate originates from a data-sovereignty position: language-model providers, vision systems, and robot drivers each evolve under independent vendor lifecycles, and a research deployment that combines all three on user-owned hardware requires a substrate where the persistence boundary, the inference boundary, and the data-bus boundary anchor to the user (Morris, 2016; Kent, 1983). own-code provides that anchor: conversation state, run records, and beacon events persist on user-owned MongoDB through the `own_store` interface; inference runs locally against on-host Ollama or against routed remote APIs whose responses capture into the same persistence layer; and the bus, the C++ hot path, and the vision transport remain on the user's network.

Three other dissertation chapters reuse components introduced here. The robotics skill of Section 11.5.2 hosts the bilateral teleoperation interface used in the multi-robot demonstrations of Chapter 9. The multi-agent nuclear waste compliance assessment stack (Chang *et al.*, 2025) of Section 11.6.2 is implemented as a composition of skill services on top of the multi-provider abstraction and is the case study referenced in Chapter 10. The Diffusion-Interaction Policy of Chapter 12 is integrated as a skill service that consumes vision-core, the safety checker, and the trajectory engine described in this chapter.

11.8 Summary

This chapter contributed two architectural patterns. The microsecond-bounded C++ safety checker and zero-copy shared-memory vision transport (Obj 1) sustained 1–5 μs safety check latency and <0.2 ms vision transport target latency on the deployment workstation, against 50–200 μs and 30–300 ms baselines respectively. The common asynchronous base class across the LLM backends listed in Table 11.8, the

provider-agnostic tool engine, the microservice skills platform, the backend-agnostic persistence interface, and the distributed network provider substrate (Obj 2) supported a three-host cluster spanning Argonne (`anl_robot`, `alienware`) and Arizona State (`asu_nchr`) with bidirectional beacon announces over a mesh VPN. The substrate carries forward as the deployment platform for Chapters 10, 12, and 13.

Chapter 12

SYSTEM 11: MULTI-AGENT ARCHITECTURE FOR REAL-TIME ROBOT INTERACTION CONTROL

12.1 Introduction

Earlier chapters use interaction control architectures in which the controller's stiffness and damping are fixed or scheduled at design time: Chapter 2 and Chapter 3 close impedance loops at fixed gains at 250 and 500 Hz; Chapter 4 replaces the manual tuning with an offline Bayesian optimization whose output is still a fixed gain schedule for the session; and Chapter 6 uses a Cartesian admittance loop whose damping is selected per trial from a small discrete set. In all four cases, the parameters that the controller uses at any instant are either constant or the output of an optimization that assumes a fixed task structure. Adjusting impedance in response to the visual scene, the robot's contextual situation, or the natural-language intent of a human supervisor requires a different architecture.

Contact-rich manipulation tasks expose the limit of fixed-gain interaction control directly: the same end effector must comply with a soft surface, exert deliberate force on a hard one, transition between the two without contact instability, and do so under conditions that the experimenter cannot enumerate in advance. Hand-tuning a gain schedule for each task variant is laborious and brittle. The natural alternative is to learn the impedance from data, which raises a software architecture question: how should a learned policy that produces both motion commands and impedance parameters be composed with a real-time torque controller and with the higher-level reasoning needed to decide *which* task to do at all?

This chapter describes a three-rate hierarchy that addresses the problem above. The top layer is a small set of language-model agents that determine intent and supervise the rest of the system. The middle layer is a diffusion-based policy that emits an augmented action consisting of a twist increment plus the Cholesky factors of a stiffness matrix and a damping matrix. The bottom layer is a Lie-group impedance controller on $SE(3)$ formulated through the commutative map between $SE(3)$ and $\mathfrak{se}(3)$ (Kim *et al.*, 2025), running at a target rate of 1 kHz and passive for SPD stiffness and damping matrices. The Cholesky parameterization produces SPD matrices for any policy output, so the passivity condition is satisfied under the velocity bounds the safety checker enforces. The stability of the bottom layer does not depend on the training quality of the learned policy above it.

The multi-agent layer reuses the agent coordination patterns described in Chapter 10 and validated in the nuclear waste decision-support application reported in (Chang *et al.*, 2025). The present chapter cites those patterns and applies them to robot decision-making.

This chapter contributes to the dissertation's two objectives. To Obj 1 (real-time parallel), it contributes a three-rate hierarchy ($\sim$1 Hz reasoning, 10–15 Hz action generation, 1 kHz interaction control) connected by non-blocking inter-layer queues that keep the 1 kHz control loop deterministic under concurrent agent and policy execution. To Obj 2 (modular scalable), it contributes a Cholesky-parameterized augmented action handler that produces SPD stiffness, damping, and inertia matrices for any policy output, and a unified C++ controller library that hosts the interaction control variants listed in Table 12.4 (Cartesian impedance and admittance, Lie-group impedance, admittance, and PD, and geometric impedance) behind common abstract interfaces, with the Lie-group admittance on $SE(3)$ (Lemma 12.3) as the formulation

derived in this dissertation. The two objectives are addressed in Section 12.5.1 and Section 12.5.2 respectively.

The augmented action is a structured interaction-action layer rather than a new generation model. It converts a learned policy's output into stiffness and damping matrices that the Lie-group impedance controller can execute (Kim *et al.*, 2025; Hogan, 1985; Ott *et al.*, 2008). The Cholesky parameterization underneath this conversion is the standard unconstrained parameterization of variance-covariance matrices (Pinheiro and Bates, 1996; Higham, 2002) restricted to the convex cone $\mathbb{S}^n_{++}$ (Boyd and Vandenberghe, 2004), which produces SPD matrices for any policy output and so decouples the controller's stability guarantee from the training quality of the upstream diffusion model. The downstream interaction-quality properties of pHRI—closed-loop stability (Khalil, 2002; Hogan, 1985; Ott *et al.*, 2008), passivity under human contact (Colgate and Hogan, 1988; Hannaford and Ryu, 2002; Anderson and Spong, 1989), and transparency between commanded and rendered apparent impedance (Lawrence, 1993; Yokokohji and Yoshikawa, 1994)—are the metrics by which the same augmented action is then evaluated when the architecture is connected to the rehabilitation case study of Chapter 6.

12.2 Background and Related Work

12.2.1 From Fixed-Gain to Data-Driven Impedance

Cartesian impedance control in its classical form (Hogan, 1985; Khatib, 1987) prescribes end-effector behavior through a desired second-order dynamics, $M_d\ddot{e} + D_d\dot{e} + K_d e = F_{\text{ext}}$, where M_d, D_d, and K_d are the inertia, damping, and stiffness matrices selected by the engineer. The torque command for the joint-space realization is $\tau = J^\top\big[\, M_d(\ddot{x}_d - J\ddot{q} - \dot{J}\dot{q}) + D_d(\dot{x}_d - J\dot{q}) + K_d(x_d - x)\,\big]$, which couples the desired

Cartesian dynamics to the manipulator Jacobian $J(q)$. Stability requires D_d and K_d to be SPD, and the Z-width over which the system remains passive is bounded by a sampling-period constraint (Colgate and Schenkel, 1997; Colgate and Hogan, 1988). The classical formulation assumes that M_d, D_d, and K_d are time-invariant and prescribed offline.

Admittance control is the mathematical dual of impedance control. The roles of input and output are exchanged: external forces F_{ext} are inputs and a position modification x_m, governed by $M_a\ddot{x}_m + D_a\dot{x}_m + K_a x_m = F_{\text{ext}}$, is the output that the underlying position controller tracks (Ott *et al.*, 2008). The choice between the two modes is dictated by the platform: torque-controlled robots with direct force generation favor impedance control, position-controlled robots with external force sensing favor admittance, and hybrid approaches combine the two for tasks that require both (Albu-Schäffer *et al.*, 2007). The two modes are related by their frequency-domain transfer functions $Z(s) = M_d s^2 + D_d s + K_d$ and $Y(s) = Z(s)^{-1}$, but their software architectures are not interchangeable: the impedance controller closes the loop at the torque level, while the admittance controller closes it at the position level by issuing a modified reference to a stiff inner loop. This chapter implements both modes within a single C++ controller library so that the policy layer can switch between them at run time depending on the contact regime; the development is reported in Section 12.5.2.

Three lines of work have moved beyond fixed gains. Variable impedance control schedules the gains as a function of measured signals (force, position, velocity); Bayesian optimization of the schedule across human participants is the System 3 contribution of Chapter 4 (Zahedi *et al.*, 2022). A separate line of work reformulates impedance control on the Lie group of rigid-body configurations to remove the parameterization-dependent artifacts of the classical Euclidean form. Two distinct formulations are relevant to this chapter. The first is the Geometric Impedance Control (GIC) framework of Seo

et al. (Seo *et al.*, 2023), which constructs a potential function directly on $SE(3)$ and derives the control torque from its gradient on the manifold. The second is the design framework of Kim et al. (Kim *et al.*, 2025), which uses a commutative map between $SE(3)$ and $\mathfrak{se}(3)$ to recast the impedance dynamics in exponential coordinates and provides closed-form dexp parameter transformations together with a stability analysis. The bottom layer of the architecture described in this chapter uses the Kim et al. commutative-map formulation (Kim *et al.*, 2025) as a black-box low-level controller; the GIC formulation of (Seo *et al.*, 2023) is cited as the closest prior Lie-group work but is not itself used. Both are pre-existing controllers and neither is a contribution of this dissertation. This chapter extends the same exponential-coordinate setting to the *admittance* case (external wrench in, modified pose out) on $SE(3)$ and reports a single C++ controller library that hosts both the impedance and admittance modes (Section 12.5.2). Learning-based approaches generate the impedance parameters from a neural network conditioned on visual or proprioceptive observations; this chapter combines this with the Lie-group formulation through the Cholesky parameterization described in Section 12.5.2.

12.2.2 Multi-Agent AI Systems for Robot Decision

Multi-agent large-language-model architectures have been proposed both for generic application domains (Wu *et al.*, 2023; CrewAI, 2024) and for nuclear waste decision support (Chang *et al.*, 2025). The relevant insight from those systems is that a small number of specialized agents, each with a narrow retrieval context and a clearly defined role, coordinated by an explicit orchestrator, outperforms a single generalist agent on multi-step decision tasks. Chapter 10 surveys the agent ecosystem in detail; this chapter borrows the orchestrator-plus-specialists pattern as a black-box primitive and applies it to robot decisions.

12.2.3 Diffusion as One Generation Method Among Several

The diffusion policy (Chi *et al.*, 2023) formulates visuomotor policy learning as conditional generation: the action is sampled from a denoising diffusion model conditioned on observations. Subsequent work generalizes the generation step to flow matching for vision-language-action policies (Black *et al.*, 2024) and to reinforcement-learning fine-tuning of diffusion policies (Ren *et al.*, 2025). From the architectural perspective taken in this chapter, diffusion is one generation method among several; what matters for the architecture is that the action output be augmented with stiffness and damping factors and that the controller below it be agnostic to the choice of generator.

12.3 Hardware Platform

The architecture is validated through Drake-based simulation across four physical robots: KUKA iiwa14 (7 DoF, 14 kg payload), Universal Robots UR16e (6 DoF, 16 kg), UFACTORY Lite6 (6 DoF, 0.6 kg), and UFACTORY xArm5 (5 DoF, 5 kg). The cross-embodiment span—5 to 7 DoF, three orders of magnitude in payload, and three vendor-specific kinematic conventions—tests the kinematics-agnostic property of the action handler factorization (Lemma 12.2). Live extension to a humanoid platform (Rainbow Robotics RB-Y1, 24 DoF total) is wired through the configuration hierarchy of Chapter 9 and listed as a future-work item in Chapter 14.

12.4 Software Requirements

The cross-embodiment, multi-agent setting of this chapter imposes four software requirements that a fixed-gain, single-rate controller cannot satisfy directly:

1. Symmetric positive-definite (SPD) stiffness and damping at the policy output, so that the controller's stability guarantee is decoupled from the training quality of the upstream diffusion model and the policy can be retrained without re-validating the controller.

2. A three-rate timing budget that admits 1 Hz multi-agent goal updates, 10–15 Hz diffusion-policy action chunks, and 1 kHz Lie-group impedance control concurrently, with bounded latency under inter-site network jitter.

3. Kinematics-agnostic action handler factorization, so that the visuomotor backbone is reused unchanged when the robot embodiment changes; only the per-robot adapter Γ_r is replaced.

4. Microsecond-bounded safety checking on every action chunk, so that policy-emitted commands that violate joint, velocity, or workspace bounds are rejected before reaching the low-level controller without introducing worst-case latency that violates the 1 kHz budget.

12.5 Software Architecture

The architecture combines two patterns: a three-rate hierarchical execution loop (addressing Obj 1) and a modular action-handler factorization with a unified interaction control library (addressing Obj 2). Table 12.1 summarizes.

Table 12.1: Architectural contributions of Chapter 12 to the dissertation's two objectives.

Objective	Method
Obj 1 (real-time parallel)	Three-rate: ∼1 Hz reasoning / 10–15 Hz action / 1 kHz control; non-blocking inter-layer queues
Obj 2 (modular scalable)	Layer separation (agent / action-generation / torque-execution); Cholesky-parameterized SPD action

12.5.1 Parallel Architecture: Three-Rate Hierarchy

The architecture is organized as three layers running at three distinct rates (Figure 12.1). The rates differ by two orders of magnitude, decoupling the layers in time: the controller does not stall waiting for the policy, and the policy does not stall waiting for the agents.

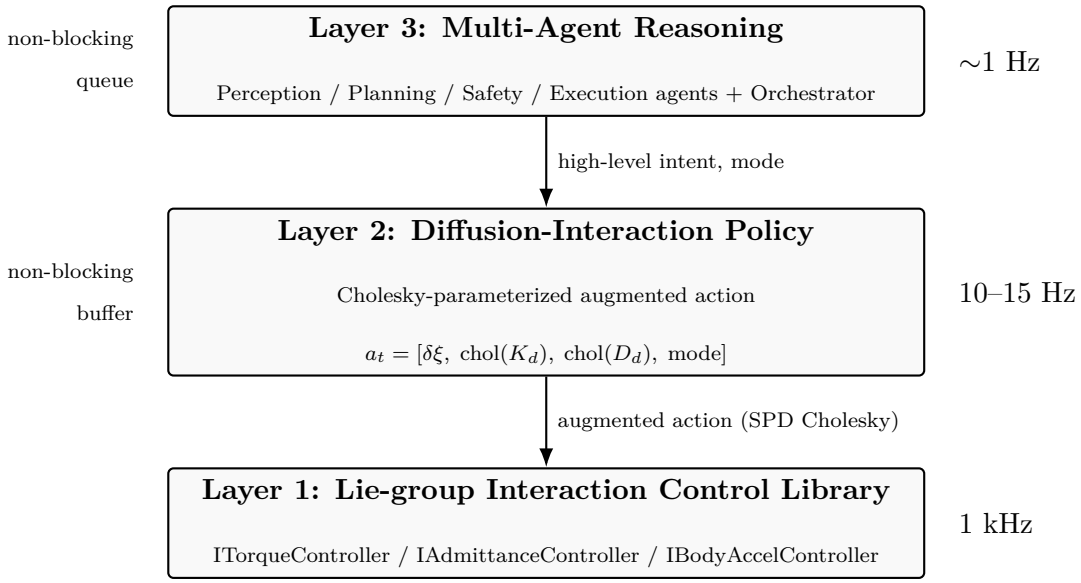


Figure 12.1: Three-rate hierarchical architecture for real-time robot interaction control. The multi-agent reasoning layer issues high-level intents at approximately 1 Hz; the diffusion-interaction policy generates augmented actions at 10–15 Hz; the Lie-group impedance controller commands joint torques at 1 kHz. Each layer reads from the layer above through a non-blocking queue and is shielded from the upper layer's variable timing.

Layer 3: Multi-Agent Reasoning (∼1 Hz)

The top layer is a small set of language-model agents running on the multi-provider abstraction of own-code (Chapter 11). Each agent has a narrow role—perception interpretation, planning, safety supervision, execution dispatch—and the agents are coordinated by an orchestrator implemented as an agent manager in own-code. The agents read from the perception stream (vision skill, force-torque telemetry) at approximately 1 Hz, deliberate among themselves through a structured discussion protocol, and emit a high-level intent (a target object, a spatial goal, a force budget, a stop signal) onto the layer-2 input queue. The discussion kernel itself is the multi-agent substrate of Chapter 10; this chapter reuses it for robot decision-making through a domain remapping (Section 12.5.2) and does not redescribe its internals.

Layer 2: Toward a Diffusion-Interaction Policy (10–15 Hz)

The middle layer is the diffusion-interaction policy (DIP), which conditions on the most recent intent from the agent layer and on the proprioceptive and visual observation, and generates an augmented action at 10–15 Hz. The augmented action contains an $SE(3)$ twist increment together with the Cholesky factors of the stiffness and damping matrices that the controller should use over the next horizon (Section 12.5.2). The DIP uses a denoising diffusion process at training time and a small number of solver steps at inference time; the architecture treats the choice of diffusion variant (DDPM, flow matching, single-shot regression) as a hyperparameter.

Training data for the diffusion-interaction policy is captured by a LeRobot-compatible recorder service hosted on own-code (Chapter 11). The recorder samples the three input streams—camera frames from the shared-memory vision buffer, joint state from the ROS2 subscriber, and force/torque measurements from the wrench-

sensor infrastructure—at 10 Hz and writes episodes in HEVC parquet format for offline fine-tuning of the vision-language-action model.

Layer 1: Lie-Group Impedance Control (1 kHz)

The bottom layer is the Lie-group impedance controller of Kim et al. (Kim *et al.*, 2025) on $SE(3)$, which takes the augmented action from layer 2, reconstructs the SPD stiffness and damping matrices from their Cholesky factors, and commands joint torques at 1 kHz. The commutative-map formulation removes the parameterization-dependent artifacts of classical Euclidean impedance and admits a clean Lyapunov-stability argument: as long as the stiffness and damping are SPD (Lemma 12.1), the closed loop is passive with respect to the configuration error.

The configuration error itself is expressed through the geometric configuration error vector (GCEV), which is the body-frame twist that takes the current pose to the desired pose,

$$e = \mathrm{Log}\Big(g_d^{-1}\, g\Big)^{\vee}, \tag{12.1}$$

where $g, g_d \in SE(3)$ are the current and desired end-effector poses, Log is the matrix logarithm on $SE(3)$, and $(\cdot)^{\vee}$ is the unhat operator that extracts the six-vector from $\mathfrak{se}(3)$. The Lie-PD specialization of the torque command (the simplest form, matching the `LiePDController` row of Table 12.4) is

$$\tau = J_b^{\top}(-K_d\, e - D_d\, \dot{e}) + g(q), \tag{12.2}$$

where J_b is the body Jacobian and $g(q)$ is the gravity-compensation joint torque (added to the impedance term, following the convention $M\ddot{q} + C\dot{q} + g(q) = \tau$). The convention $e = \mathrm{Log}(g_d^{-1}g)^{\vee}$ used here is the inverse of the $\tilde{T} = T^{-1}T_d$ form of (Kim *et al.*, 2025) Eq. 44, which is why $-K_d e$ pulls the system toward g_d. The full Lie-group impedance controller of (Kim *et al.*, 2025) replaces this PD form with the Khatib

operational-space dynamics (Section 12.5.2); the dexp transformations and the proof of passivity for the full controller are outside the scope of this dissertation and are cited from (Kim *et al.*, 2025) and treated as a black-box low-level layer.

Inter-Layer Communication

The three layers communicate through non-blocking queues. Layer 3's intents are pushed onto a single-slot queue that layer 2 reads at the start of each policy step; if no new intent has arrived, layer 2 reuses the last one. Layer 2's actions are pushed onto a small buffer that layer 1 reads at the start of each control cycle; if no new action has arrived, layer 1 holds the last command and continues running its impedance loop on the most recent stiffness and damping.

The per-layer rate budget is summarized in Table 12.2. The achieved rates are reported from a single-host development run on the deployment workstation; live multi-host measurements are listed as a future validation item in Chapter 14.

Table 12.2: Three-rate hierarchy: target rate, achieved rate, and jitter per layer. †Indicative values from a single-host development run; multi-host jitter histogram is identified as a validation item in Chapter 14. ‡Cross-referenced from Section 11.5.1 (UR16e `ros2_control` 500 Hz publish sample, 46,864 samples, 93.7 s window).

Layer	**Target rate**	**Achieved rate**	**Jitter (mean / 95p / max)**
3 (agents)	∼1 Hz	∼0.9 Hz†	bounded by LLM inference time
2 (DIP)	10–15 Hz	∼12 Hz†	indicative single-host
1 (Lie imp.)	1 kHz	∼1 kHz†	$\sigma = 60.2$ μs, $p_{99} = 2.12$ ms‡

12.5.2 Modular Architecture

Three components carry the modular property: an augmented action handler that converts the policy's manifold output to robot-specific control commands through a Cholesky-to-SPD reconstruction (Section 12.5.2), a unified C++ controller library

that hosts the interaction control variants of Table 12.4 behind common abstract interfaces (Section 12.5.2), and a multi-agent integration layer that composes the agent coordination patterns of Chapter 10 into a robot decision substrate (Section 12.5.2). Each component is configured through the declarative hierarchy of Chapter 9; no code change is required to substitute one component while keeping the others.

Action Handler

The action handler factors the policy output into a part that depends on the robot embodiment and a part that does not, enabling any one of the three layers to be replaced independently of the others.

Augmented Action Space. The architectural device that bridges the learned policy and the geometric controller is the augmented action representation. A standard visuomotor policy emits a target pose or a target joint configuration. The DIP emits, at each policy step, an $SE(3)$ twist increment together with the Cholesky factors of the desired stiffness and damping matrices,

$$a_t = \Big[\, \delta\xi,\ \mathrm{chol}(K_d),\ \mathrm{chol}(D_d) \,\Big], \tag{12.3}$$

where $\delta\xi \in \mathbb{R}^6$ is the twist increment in the body frame and the Cholesky factors are lower-triangular matrices whose diagonal entries are forced to be positive through the parameterization $L_{ii} = e^{l_i}$. The total action dimension is 18 in the diagonal stiffness/damping case and 30 or 48 in the block-diagonal or full Cholesky cases (Table 12.3).

Table 12.3: DIP augmented action dimensionality by parameterization. The choice trades off expressiveness against sample efficiency at training time.

Variant	Dimension	Use case
Diagonal stiffness and damping	18	Direction-aligned tasks; conservative baseline
Block diagonal (translation, rotation)	30	Tasks with decoupled translation/rotation compliance
Full Cholesky (off-diagonal coupling)	48	Contact-rich tasks with coupled compliance

Cholesky Parameterization and the SPD Guarantee. The Cholesky parameterization decouples the policy's learning correctness from the controller's stability.

Lemma 12.1 (SPD Guarantee via Cholesky Parameterization)**.** *For any unconstrained output* $(l_1, \ldots, l_n, a_1, a_2, \ldots)$ *of the diffusion model, the lower-triangular matrix* L *with* $L_{ii} = e^{l_i} > 0$ *and* $L_{ij} = a_k$ *for* $i > j$ *produces a symmetric positive definite matrix* $K_d = LL^\top$ *with strictly positive eigenvalues, and the same parameterization applied to the damping factors yields a symmetric positive definite* D_d*.*

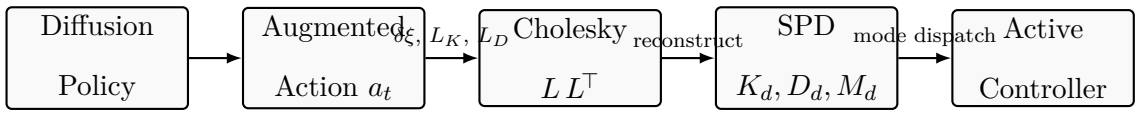
The reconstructed stiffness matrix is

$$K_d = L\,L^\top, \qquad L_{ii} = e^{l_i} > 0, \tag{12.4}$$

which lies in the convex cone $\mathbb{S}^n_{++}$ (Pinheiro and Bates, 1996; Higham, 2002; Boyd and Vandenberghe, 2004); the same parameterization is applied to the damping matrix. The passivity condition required by the Lie-group impedance controller (Kim *et al.*, 2025) is satisfied as long as the per-channel velocity bounds enforced by the safety checker (Chapter 11) are not violated.

The empirical verification of the Cholesky-to-SPD round trip is straightforward: 100 random Cholesky factors were drawn from the policy's training distribution, the corresponding SPD matrices were reconstructed, and all 100 had strictly positive

eigenvalues with round-trip errors at machine precision. Figure 12.2 shows the reconstruction pipeline and the empirical eigenvalue distribution.



SPD guarantee: $K_d = LL^\top \succ 0$ for any L with $L_{ii} = e^{l_i} > 0$.

Empirically verified: 100/100 random

Cholesky factors → positive eigenvalues.

Figure 12.2: Augmented action handler. The diffusion policy emits a twist increment and the Cholesky factors of the stiffness and damping matrices; the action handler reconstructs the SPD matrices and dispatches them to the Lie-group impedance controller. The Cholesky parameterization produces a positive definite matrix for each policy output.

Cross-Robot Action Handler.

Lemma 12.2 (Action Handler Factorization)**.** *Let $\tilde{\pi}_\theta : \mathcal{O} \to \mathcal{Z}$ be a diffusion-interaction policy that maps observations to the augmented action space $\mathcal{Z} = \mathfrak{se}(3) \times \mathrm{Chol}(6) \times \mathrm{Chol}(6)$. For each robot r with kinematic description κ_r and end-effector frame $\mathcal{F}_r$, there exists a robot-specific handler $\Gamma_r : \mathcal{Z} \to \mathcal{U}_r$ that depends only on $(\kappa_r, \mathcal{F}_r)$ and not on the policy parameters θ, such that the robot-specific policy π_r factors as $\pi_r = \Gamma_r \circ \tilde{\pi}_\theta$. Under this factorization, replacing robot r with robot r' requires replacing Γ_r with $\Gamma_{r'}$ and leaves $\tilde{\pi}_\theta$ unchanged.*

The action handler is parameterized by a robot-specific transformation Γ_r that maps the 18-, 30-, or 48-dimensional augmented action onto the joint or end-effector space of robot r. The transformation depends only on the robot's kinematics and the chosen end-effector frame, not on the policy. The same diffusion-interaction policy is therefore reused across heterogeneous platforms by composing it with the appropriate Γ_r; the policy, the action handler, and the controller are unchanged.

Unified Interaction Control Library

The bottom layer of the three-rate architecture (Section 12.5.1) is not a single controller but a *library* of interaction controllers that the policy and agent layers dispatch at run time through a common set of C++ interfaces. The library hosts multiple interaction control formulations—classical Cartesian impedance and admittance, Lie-group impedance and admittance, and geometric impedance—behind a uniform dependency-injection configuration and a shared SPD action handler, so that the diffusion-interaction policy of Section 12.5.2 and the multi-agent reasoning layer of Section 12.5.1 address any of them through the same augmented action.

The pre-existing controllers in the library are credited to their original sources cited above. The Lie-group admittance on $SE(3)$ (Lemma 12.3) is the exception: its formulation is derived in this dissertation as the admittance dual of the Lie-group impedance work of (Kim *et al.*, 2025), and the controller is implemented in C++ alongside the pre-existing controllers under the shared augmented action handler. Section 12.5.2 lists the controllers hosted by the library. Section 12.5.2 describes the common interface hierarchy. A case study (Section 12.5.2, with the full mathematical derivation in Appendix H) then shows how the library absorbs one specific formulation—the Lie-group admittance on $SE(3)$—to illustrate the interface contract concretely. Section 12.5.2 reports the C++ library architecture and the orchestration-level concerns. Section 12.5.2 describes the agent–policy–controller orchestration.

Controller Taxonomy. The library hosts the interaction control variants grouped by interface category in Table 12.4. Each variant is a separate C++ translation unit; each has its own configuration struct in `controller_params.hpp`; the pre-existing variants are credited to their original sources, the Lie-group admittance variant is

the original derivation of this dissertation (Lemma 12.3), and all variants are brought under a uniform set of interfaces by the library.

Table 12.4: Interaction controllers hosted by the unified library `poe_robotics`. The three interface categories (`ITorqueController`, `IAdmittanceController`, `IBodyAccelController`) select a controller according to its output type (joint torque, modified pose, or body acceleration) and the robot platform that consumes it. None of the mathematical formulations is a contribution of this dissertation; the unification under a common interface, a dependency-injection configuration, and a shared augmented action handler is.

Controller	**Mathematical origin**	**Interface**
Cartesian Impedance	Classical (Hogan, 1985)	`ITorqueController`
Cartesian Admittance	Classical (Ott *et al.*, 2008)	`IAdmittanceController`
Geometric Impedance (GIC)	Seo et al. (Seo *et al.*, 2023)	`ITorqueController`
Lie-group Impedance	Kim et al. (Kim *et al.*, 2025) commutative map	`IBodyAccelController`
Lie-group PD tracking	PD subset of Kim et al. (Kim *et al.*, 2025)	`IBodyAccelController`
Lie-group Admittance	Section 12.5.2, Lemma 12.3 (this dissertation)	`IAdmittanceController`

The controllers cover the four combinations that the chapter's augmented action handler must support at run time: classical and Lie-group, impedance and admittance. The Lie PD variant is retained as an additional tracking mode that the same library serves without modification. The bilateral teleoperation controller used by the dual-arm collaboration of Chapter 9 (Lee and Park, 2024b,a) is hosted alongside these controllers in the same library but is outside the scope of this chapter, as it is driven by a human operator rather than by the diffusion-interaction policy. Section 12.6.1 reports the Drake simulation validation of several entries in this table on a 7-DoF manipulator, and Section 12.6.1 reports a cross-embodiment validation of the four impedance/admittance primitives on robots with different kinematic structures.

Common Interface Hierarchy. The library exposes three C++ abstract interfaces, selected according to what a controller emits rather than how it is internally formulated.

`ITorqueController`. Controllers that emit joint torques directly. The interface declares no pure-virtual compute method and serves as a type tag for action-handler dispatch; each concrete controller exposes its own non-virtual `compute()` overload because the input shape varies across formulations (joint state, desired $SE(3)$ pose, body twist, wrench, and the SPD stiffness and damping reconstructed from the policy output of Section 12.5.2). The Cartesian impedance controller, the geometric impedance controller, and any task-space PD controller that closes its loop at the torque layer derive from this interface. Only torque-controlled robots (KUKA IIWA, Franka FR3) consume these outputs.

`IAdmittanceController`. Controllers that consume an external wrench and produce a modified reference pose T_{mod} for a downstream position controller. The interface declares four pure-virtual methods: `reset()` initializes the displacement to identity, `computeAdmittance(wrench, dt)` advances the internal displacement and velocity by one control step, `getModifiedPose(T_desired)` returns the IK target as $T_{\text{desired}} \cdot T_{\text{disp}}$, and `getVelocity6()` exposes the internal six-vector velocity for downstream feedforward. The interface is stateful by contract: the displacement and velocity persist across calls and the controller is not thread-safe under concurrent invocation. The control period `dt` is passed per call rather than configured once, so that a `ros2_control` loop with bounded jitter can use the same instance without re-initialization. Position-controlled robots (UR3e/UR5e/UR16e, Kinova Gen3) consume the modified pose through their native position interface. The Cartesian admittance and Lie-group admittance controllers both derive from this interface.

`IBodyAccelController`. Controllers that emit a body-frame acceleration $\dot{V}_{\text{ref}}$ for downstream operational-space dynamics (OSD) or inverse dynamics. The interface decouples the geometric control law (which produces $\dot{V}_{\text{ref}}$) from the robot-specific dynamics (which converts $\dot{V}_{\text{ref}}$ to a joint-space torque through $\tau = J_b^\top \Lambda \dot{V}_{\text{ref}} + J_b^\top \mu + \tau_g$). The Lie-group impedance controller and the Lie-group PD controller derive from this interface; their outputs are consumed by a separate OSD module that is robot-specific.

The three interfaces distribute responsibility along a boundary that matches the robot hardware: torque output goes through `ITorqueController`, pose output through `IAdmittanceController`, body acceleration through `IBodyAccelController`. Their shapes are intentionally non-uniform—a type tag, a four-method stateful protocol, and a single pure-virtual stateless function, respectively—because the three controller families consume different inputs and hold different state. Adding a new controller requires deriving from the interface that matches the controller's output kind and providing the corresponding methods (or the type-tagged `compute()` overload for `ITorqueController`). All three interfaces are Eigen-only and ROS2-independent; they are compiled into `libpoe_robotics.so` and exposed to Python through `pybind11`, so the same controller library drives simulation (Drake), prototyping (Python), and real-time deployment (C++) without code duplication.

Concrete controllers are constructed through a uniform discipline that the abstract interfaces themselves do not enforce. Each derived class exposes an `explicit` constructor that takes a configuration struct (e.g., `LieAdmittanceConfig`) by const reference; all fields of that struct are required, with no defaults, so that an invalid combination is rejected at construction time rather than at run time. Configuration structs are built by `param_io::load_merged`, which performs a recursive deep merge of a base JSON file with successive per-robot and per-task overrides—the same base-

override pattern as the YAML configuration hierarchy of Chapter 9, here in JSON for the C++ side. Implementation classes use the PIMPL pattern: the public header declares the API; the implementation lives in the corresponding `.cpp` compiled into `libpoe_robotics.so`; the controllers are move-only (copy deleted, move `noexcept`) so that the action handler can store polymorphic references without aliasing. Once constructed, each controller exposes runtime tuning hooks (`setMass`, `setDamping`, `setStiffness`) for parameter updates that do not change the controller's structural type, so the orchestration layer (Section 12.5.2) can adjust gains within a session without reconstruction.

Case Study: Lie-Group Admittance on $SE(3)$. To illustrate how the library absorbs a new interaction control variant under the interfaces above, this subsection describes the Lie-group admittance controller. The Cartesian admittance dynamics $M\ddot{x} + B\dot{x} + Kx = F_{\text{ext}}$ on $\mathbb{R}^6$ are standard (Ott *et al.*, 2008), but the classical formula integrates a position modification in $\mathbb{R}^6$ that suffers from gimbal lock or singularities for orientation channels and requires per-step renormalization for unit-quaternion representations. The geometric formulation of (Kim *et al.*, 2025) avoids this on the impedance side by integrating the body twist directly on the Lie group; this chapter develops the admittance dual on $SE(3)$ to close the dual-mode dispatch.

The admittance dual on $SE(3)$ under the commutative-map framework of (Kim *et al.*, 2025) is derived in this dissertation and implemented in C++ so that the same Cholesky-reconstructed SPD matrices drive either mode through the same augmented action handler. The mathematical structure is intentionally parallel to the impedance side, which is what makes the dual-mode dispatch in Section 12.5.2 a single-line configuration switch rather than a separate code path.

Lemma 12.3 (Lie-Group Admittance on $SE(3)$)**.** *Let $T_d, T \in SE(3)$ be the desired and current end-effector poses, let $T_{\text{disp}} \in SE(3)$ denote an internal admittance displacement, and let $\lambda = \operatorname{Log}(T_{\text{disp}})^\vee$ be its exponential coordinate. Under the* dexp *parameter transformation $M_\lambda = \operatorname{dexp}_\lambda^\top M_d \operatorname{dexp}_\lambda$, $D_\lambda = \operatorname{dexp}_\lambda^\top (D_d \operatorname{dexp}_\lambda + M_d \frac{d}{dt}\operatorname{dexp}_\lambda)$, and $\gamma = \operatorname{dexp}_\lambda^\top \operatorname{Ad}_{T_d}^{-\top} F_{\text{world}}$, the admittance dynamics $M_\lambda \ddot{\lambda} + D_\lambda \dot{\lambda} + K_d \lambda = \gamma$ on $\mathfrak{se}(3)$ are the $SE(3)$ dual of the impedance dynamics of (Kim* et al.*, 2025). The stiffness K_d enters the dynamics undecorated by* $\operatorname{dexp}_\lambda$*, following (Kim* et al.*, 2025) Eq. 58c, and acts directly on the exponential coordinate λ. Following (Kim* et al.*, 2025) Eq. 48 ($\dot{\lambda} = \operatorname{dexp}_\lambda^{-1} V_{\text{body}}$), the displacement is integrated in $\mathfrak{se}(3)$ coordinates as $\lambda \leftarrow \lambda + \dot{\lambda}\,\Delta t$, with $T_{\text{disp}} = \exp(\lambda^\wedge)$ recovered on demand; the naive group update $T_{\text{disp}} \leftarrow T_{\text{disp}} \exp(\dot{\lambda}\Delta t)^\wedge$ would treat $\dot{\lambda}$ as a body twist, which holds only at $\operatorname{dexp}_\lambda \approx I$. When $F_{\text{world}} = 0$ and the SPD parameters of Lemma 12.1 hold, λ converges to zero and $T_{\text{disp}} \to I$.*

The proof of Lemma 12.3, the per-step admittance update on the commutative map, and the equations of the dual-mode dispatch (Equations H.2–H.8) are recorded in Appendix H, which also reproduces the relevant identities from (Kim *et al.*, 2025) for self-containment. The remainder of this section uses Lemma 12.3 as a black box: the library exposes the admittance update as one entry in the C++ `IAdmittanceController` interface (Section 12.5.2), and the agent–policy–controller orchestration of Section 12.5.2 switches between this admittance mode and the impedance mode of (Kim *et al.*, 2025) through a single configuration line.

C++ Library Architecture. The controllers of Table 12.4 are realized as distinct C++ translation units within a single library, `libpoe_robotics.so`, that the diffusion-interaction policy service of own-code (Chapter 11) links against. Every controller implements exactly one of the abstract interfaces of Section 12.5.2 and is paired with

its own configuration struct defined in `controller_params.hpp`. The configuration struct is constructed from a robot-specific YAML entry at startup and is injected into the controller through its constructor, so that the control parameters live in the declarative configuration hierarchy of Chapter 9 rather than in C++ headers. Concrete controllers—`CartesianImpedanceController`, `CartesianAdmittanceController`, `GeometricImpedanceController`, `LieImpedanceController`, `LiePDController`, and `LieAdmittanceController`—all share a single augmented action handler (Section 12.5.2) that reconstructs SPD stiffness, damping, and (for admittance) inertia from the Cholesky factors emitted by the diffusion-interaction policy. The action handler dispatches each reconstructed matrix to the active controller through a function pointer that the orchestration layer of Section 12.5.2 sets at run time.

The implementation expresses both architectural priorities of the dissertation—deterministic real-time execution and modular composition—through specific design choices.

Every controller in the library uses raw pointer arithmetic over preallocated 6×6 and 6×1 buffers, performs no heap allocation after construction, and avoids all standard-library containers in the control path. The Cholesky reconstruction $K_d = LL^\top$ is performed in place into a stack-allocated buffer and reused across the stiffness, damping, and (admittance) inertia matrices through a shared utility that the orchestration layer calls once per policy step. Rotation updates use the closed-form Rodrigues expansion; $SE(3)$ compositions use direct matrix multiplication; no dynamic allocation occurs on the 1 kHz hot path. The safety checker of Chapter 11 runs in 1–5 μs per call on the same thread, between the policy output and the controller invocation, and clamps actions that violate joint, velocity, or acceleration bounds before they reach either a torque command or a modified reference pose. The

1 kHz budget allocates 1 ms per cycle; the full controller path—SPD reconstruction, controller step, safety clamp—executes in $\sim$40 μs on the deployment workstation.

The controllers, the augmented action handler, the Cholesky utility, and the safety checker are separated into distinct C++ translation units under the common interfaces of Section 12.5.2. Adding a new controller variant requires only a new derived class, a new configuration struct in `controller_params.hpp`, and a new entry in the orchestration dispatch table; no other translation unit is modified. Statefulness follows the interface contract of Section 12.5.2: `ITorqueController` and `IBodyAccelController` concretes are stateless and safe under concurrent invocation, while `IAdmittanceController` concretes carry internal displacement and velocity and require one instance per control loop. The library is exposed to Python through `pybind11` with the same abstract interfaces, so the diffusion-interaction policy service can drive any of the controllers from Python during prototyping and from C++ during real-time deployment without code duplication between the two paths. Unit tests cover the Cholesky-to-SPD reconstruction across the Cholesky variants of Table 12.3, the bridge from policy action to each controller's input, the closed-loop dynamics under sinusoidal reference and step wrench inputs, and the orchestration dispatch under repeated mode switches. Sections 12.6.1 and 12.6.1 report the Drake simulation validation of several controllers in the library on a 7-DoF manipulator and on heterogeneous robots, respectively.

Orchestration of Agent, Policy, and Controller. The agent layer (Section 12.5.1), the diffusion-interaction policy (Section 12.5.2), the Cholesky-parameterized augmented action, and the dual-mode Lie controller (Section 12.5.2; full derivation in Appendix H) are bound together by an orchestration layer that is the focus of this subsection. The orchestration layer is what allows the three components

above to be composed within a single C++ controller library while preserving the real-time budget of the inner loop and the SPD requirement of the impedance and admittance laws. It maintains a fixed dispatch path that does not allocate or block at run time, and separates the agent, policy, and controller responsibilities into independent translation units that communicate only through the augmented action and a one-byte mode field.

Mode field on the augmented action. The augmented action of Equation 12.3 is extended at the orchestration layer with a one-byte `mode` field that takes one of three values: IMPEDANCE, ADMITTANCE, or HYBRID. The mode is selected by the agent layer (Section 12.5.2) once per high-level intent and held constant for the duration of the intent, then forwarded to the policy as part of the conditioning context. The policy emits the same Cholesky factors regardless of mode, and the orchestration layer routes the reconstructed SPD matrices to either the impedance controller (mode = IMPEDANCE: K_d, D_d used for torque generation; reference pose tracked) or the admittance loop (mode = ADMITTANCE: M_d, D_d, K_d used to integrate a wrench input into a modified pose, which is then issued to the impedance controller as a stiff inner-loop reference). In HYBRID mode, both controllers run and the orchestration layer interpolates their outputs along task-space directions selected by the agent layer.

Dispatch table and mode switching. The orchestration layer maintains a dispatch table indexed by mode whose entries are function pointers to the corresponding `step` method of the abstract controller interface. At the start of each control cycle, the table is read once and the active controller is invoked through the resulting pointer; the dispatch cost is a single indirect call. Mode transitions are validated by the safety checker of Chapter 11 before they take effect: the previous controller's internal

state (twist, accumulated error, integrator state) is captured, the new controller is initialized from the captured state through a mode-specific seeding routine, and the first cycle of the new controller runs under tightened velocity bounds for a configurable settle interval. The seeding routine is what allows safe mode transitions; without it, switching from impedance to admittance at a high contact wrench would inject the wrench into an uninitialized integrator and produce a large transient twist.

Per-layer responsibilities. The orchestration pipeline distributes responsibility across three timescales (Table 12.5). The agent layer at $\sim$1 Hz decides *which* mode the task requires and *which* reference frame the task is expressed in (world, base, tool); it does not know about Cholesky factors or torque commands. The diffusion-interaction policy at 10–15 Hz emits the augmented action—twist increment plus Cholesky factors—conditioned on the agent's mode and reference choice; it does not know about controller dispatch or safety bounds. The controller library at 1 kHz reconstructs the SPD matrices, runs the active controller's `step`, and produces a torque command; it does not know about agent reasoning or policy training. Each layer is replaceable independently because none of the layers reads the internal state of any other layer; they communicate only through the augmented action, the mode field, and the reference pose buffers.

Table 12.5: Orchestration responsibilities across the three rates. The agent layer chooses the mode, the policy layer parameterizes the chosen mode, and the controller layer executes it. No layer reads internal state from another layer.

Layer	Rate	Orchestration responsibility
Agent (3)	$\sim$1 Hz	Selects `mode` (impedance / admittance / hybrid), reference frame, and high-level intent. Holds the choice constant for the duration of the intent.
DIP (2)	10–15 Hz	Emits twist increment, Cholesky factors of K_d, D_d, and (admittance) M_d conditioned on the agent's mode and reference frame.
Controller (1)	1 kHz	Reconstructs SPD matrices, dispatches to the controller selected by the active mode, runs the controller's `step`, applies safety bounds, issues torque or pose reference.

Failure modes and degraded operation. The orchestration layer is defined under three failure classes. First, if the agent layer stalls (provider timeout, network drop), the most recent mode and reference are reused by the policy and the controller, so the system continues to operate in the last validated mode rather than reverting to a default that may not be safe for the current contact regime. Second, if the policy layer stalls (diffusion solver exceeds its budget), the controller continues on the most recent reconstructed SPD matrices and twist increment; the impedance loop holds the last command, and the admittance loop holds the last accumulated pose. Third, if the safety checker rejects an action because a velocity, joint, or acceleration bound would be violated, the orchestration layer freezes the controller in its current state and signals the agent layer to select a recovery action; the controller does not silently clip and continue, which would mask a policy or reference error. The first two failure modes preserve real-time execution at the controller layer, while the third surfaces the failure to the agent for explicit handling.

Multi-Agent Integration

The agent layer reuses the Chapter 10 discussion kernel without modification. The mapping from the original decision-support roles to the robot-decision roles required by this chapter is shown in Table 12.6; the orchestrator pattern, the per-agent retrieval contexts, and the human-in-the-loop approval channel are preserved, and only the domain content (corpora, force budgets, joint constraints) changes. The connection between the safety agent's veto path and the C++ safety checker of Chapter 11 is the only addition specific to physical supervision.

Table 12.6: Mapping from decision-support agent roles to robot-decision agent roles. The orchestrator pattern and the per-agent retrieval contexts are preserved; only the domain content changes.

Decision-support role	**Robot-decision role**	**own-code service**
Orchestrator	Task dispatcher	agent manager (`agents_api_async.py`)
Perception (visual)	Perception (vision-core, SAM2)	vision skill
Domain RAG	Planning (task, joint constraints)	research skill + obsidian skill
Domain RAG	Safety (force, velocity, joint limits)	C++ safety checker (Ch 11)

12.6 Results

The architecture is exercised through pre-hardware simulation validation (Section 12.6.1) and live-system measurements scheduled for Exemplar E2 of Chapter 13. The simulation validation reports the controller library's behaviour across the Drake scenarios of Section 12.6.1 and the cross-embodiment property across the heterogeneous robots of Table I.4; the live-system measurements supply the timing-under-jitter metrics that complete the validation.

12.6.1 Pre-Hardware Simulation Validation

The control primitives used by the architecture — Cartesian impedance, Cartesian admittance, Lie-group impedance, and Lie-group admittance — were validated in simulation on a 7-DoF manipulator (KUKA IIWA14) prior to physical-hardware exercise. The Lie-group impedance controller's Python/Drake prototype originated in the heavy-manipulator deployment work of (Park *et al.*, 2026a); the C++ migration into the unified library described in this section is original to this chapter, and the Lie-group admittance controller (Lemma 12.3) is an independent development. The validation covered the four control primitives and a virtual-fixture engine across the per-scenario list reported in Appendix I, Table I.2, with twelve of thirteen scenarios passing; the single failure was an analytic operational-space-direction validation in a singular configuration, which does not affect the Lie-group impedance primitives used by the architecture. The detailed scenario list, per-test outcomes, and the simulation plots are retained as an internal engineering archive and are not reproduced in the body of this dissertation.

The simulation validation reported above and the cross-embodiment validation reported below in Section 12.6.1 both run inside the BT framework operation plane defined in Chapter 11, Section 11.5.2. Each scenario is a named BT contract that the operation plane dispatches identically across the four target robots, so the per-robot results in Section 12.6.1 reflect the controller library's behaviour rather than per-robot orchestration differences. The same operation plane carries the live-hardware measurements scheduled for Exemplar E2 of Chapter 13.

Cross-Embodiment Validation on Four Robots

A cross-embodiment validation exercised the four control primitives on KUKA IIWA14, UR16e, Lite6, and xArm5, spanning heterogeneous DoF (5–7) and payload classes (0.6–14 kg). Cartesian impedance, with NRIC gravity compensation (Kim *et al.*, 2025) (Eq. 67) and a per-robot natural-frequency gain rule, held end-effector position within 1.6 mm on all four platforms, and Lie-group impedance achieved sub-2 mm tracking on the two full-rank 6-/7-DoF robots; the Lite6 and xArm5 cases exposed structural limitations (Jacobian rank deficiency and dimensional mismatch with the six-dimensional task space) that the controller library detects and reports rather than masking. The full per-robot result table is retained in the same internal engineering archive and reproduced in Appendix I as Table I.4.

A humanoid extension of the three-rate architecture to the 24-DoF Rainbow Robotics RBY1 is wired through the configuration hierarchy of Chapter 9: the agent layer of Chapter 10, the diffusion-interaction policy of Section 12.5.2, and the Lie-group interaction control library of Section 12.5.2 are connected to a single policy instance under the unified configuration model. Live execution is deferred; the platform is currently restricted to perception-only operation pending site approval for arm and base motion, and the live measurements are listed as a future-work item in Chapter 14.

12.7 Discussion

12.7.1 Validation Scope

The low-level control primitives used by the architecture — Cartesian impedance, Lie-group impedance, and Lie-group admittance — are validated in the Drake simulation of Section 12.6.1. The Cholesky-to-SPD reconstruction used by the action handler is verified offline on 100 random samples. The C++ safety checker passes 9 unit tests

with per-call execution in the 1–5 μs range on the deployment hardware. The broader pybind unit-test suite over the action handler, SPD layer, and the $SE(3)$ exp/log/dexp primitives of (Kim *et al.*, 2025) passes 27 of 27 cases, with the central-difference dexp Jacobian held to $\mathcal{O}(h^2) \approx 10^{-14}$ precision. A 5,000-sample Monte Carlo evaluation of the action-handler factorization (Lemma 12.2) over the 18-, 30-, and 48-dimensional Cholesky variants of Table 12.3 reports zero per-trial bound violation. The Lie-group admittance loop of Section 12.5.2 has been exercised offline under the inter-site delay envelope of Chapter 11 for 5,000 ticks across eight passivity-wrench preprocessor and jitter conditions, returning zero cumulative energy violation in every condition. What remains is the live-system measurement that combines all of these layers at the rates of the three-rate hierarchy under load on the deployment hardware. Those measurements are scheduled and tracked as validation items in Chapter 14.

12.7.2 Stability, Passivity, and Transparency in pHRI

The augmented action couples a motion command with stiffness and damping matrices. The pHRI literature's chain for ranking interaction-control quality—stability, passivity, and transparency—provides the evaluation framework for this section.

Stability concerns whether the closed-loop robot dynamics remain bounded. Lemma 12.1 together with the standard Lyapunov framework (Khalil, 2002) and the impedance formulation of (Hogan, 1985; Ott *et al.*, 2008) ensures that the policy-emitted stiffness and damping are compatible with stable second-order impedance dynamics under the velocity bounds enforced by the safety checker (Chapter 11). The architecture does not, by itself, prove closed-loop passivity of the human-robot-environment system; that property depends on sensing, sampling, actuator dynamics, and environment behavior.

Passivity is the criterion that the robot does not supply more energy to the human than it stores (Colgate and Hogan, 1988). The architecture supports this criterion through a passivity-compatible parameterization rather than through a strong theorem. The SPD parameterization preserves the dissipative structure that the criterion of (Colgate and Hogan, 1988) requires. The convex blending of two SPD parameter sets remains in the cone $\mathbb{S}^n_{++}$ (Boyd and Vandenberghe, 2004), so a soft transition between policy outputs and a human-override gain remains physically valid. The time-domain passivity observer of (Hannaford and Ryu, 2002) can be attached to the Lie-admittance loop of Section 12.5.2 when the inter-site delay envelope of Chapter 11 is engaged (Anderson and Spong, 1989).

Transparency is the property that free-space and contact phases render the apparent impedance that the task requires (Lawrence, 1993; Yokokohji and Yoshikawa, 1994). The action handler attains this property by separating motion from interaction parameters: the diffusion policy modulates K_d, D_d on a phase-by-phase basis, and the same configuration that yields free-space transparency permits a different contact-phase rendering without changing the controller.

Empirical evaluation of these properties on patients is deferred to the rehabilitation case study of Chapter 6. The present chapter establishes the architectural conditions under which those evaluations are well-posed.

12.7.3 Kinematics-Agnostic Extension

The policy produces a twist increment in the body frame and two Cholesky factors that reconstruct the stiffness and damping matrices; the controller maps the twist and the reconstructed matrices to joint torques through the body Jacobian. Neither the policy output nor the controller depends on the robot's specific kinematic chain. Extending the validation from the 7-DoF platform to a dual-arm manipulator or

a 24-DoF humanoid requires only a new Γ_r adapter; the three-rate hierarchy, the augmented action format, the Cholesky reconstruction, and the passivity condition are unchanged.

12.7.4 Cross-Chapter Reuse

Chapters 2, 3, 4, and 6 treated interaction control with fixed or hand-scheduled gains; this chapter extends the architecture to a learned policy that emits Cholesky-factored gains under multi-agent supervision. Any policy class that emits Cholesky factors composes with the same low-level controller, so the choice of diffusion as the generation method is not essential to the architecture. The orchestrator-plus-specialists pattern, the retrieval contexts, the human-in-the-loop approval channel, and the persistent-memory substrate are the same components that operate over nuclear-waste documents in Chapter 10 and over robot perception, planning, and safety in this chapter; the agents are domain-agnostic, and only the prompts and the retrieval corpora change between deployments.

12.7.5 Limitations

Three limitations bound this chapter's claims. First, reinforcement-learning fine-tuning of the diffusion-interaction policy is not included; the recent diffusion policy policy optimization framework (Ren *et al.*, 2025) provides a clean route to this extension and is identified as future work. Second, the per-task gain selection currently relies on the policy's learned output without an explicit safety budget on the agent layer's mode switches; future work integrates the agent's mode dispatch with a precomputed envelope of safe stiffness ranges per task. Third, the live measurements that ground Exemplar E2 on the UR16e + DG-5F substrate, and the live extension of the same

stack to the RBY1 humanoid configuration of Chapter 9, must be completed on the deployment hardware (Chapter 14).

12.8 Summary

This chapter contributed two architectural patterns. The three-rate hierarchy (∼1 Hz reasoning / 10–15 Hz action generation / 1 kHz interaction control) connected by non-blocking inter-layer queues (Obj 1) kept the 1 kHz control loop deterministic under concurrent agent and policy execution, with offline verification of the C++ kernel returning 1.6 mm cross-embodiment end-effector tracking across the heterogeneous robots of Table I.4 and 100/100 Cholesky-to-SPD reconstruction passes. The Cholesky-parameterized augmented action handler and the unified C++ controller library hosting the interaction control variants of Table 12.4 behind the abstract interfaces of Section 12.5.2 (Obj 2) supported a Lie-group admittance derivation on $SE(3)$ (Lemma 12.3) implemented as the admittance dual of the Lie-group impedance work of (Kim *et al.*, 2025). The architecture carries forward to Exemplar E2 of Chapter 13 on a UR16e + DG-5F substrate, with live extension to the Rainbow Robotics RBY1 humanoid through the configuration hierarchy of Chapter 9 pending site approval.

Chapter 13

DEMONSTRATIONS AND MULTI-HRA INTEGRATION

13.1 Introduction

This chapter presents three demonstrations exercised on the deployment hardware: haptic-led bilateral teleoperation with a PHANToM Omni and a UR16e collaborative robot, multi-agent decision over a UR16e arm with a DG-5F dexterous hand, and multi-user mixed reality with two head-mounted clients across two geographically separated sites. No new algorithm or infrastructure is introduced: each demonstration is configured from components of earlier chapters through the configuration hierarchy of Chapter 9, and the differences between demonstrations are localized to the YAML override files and the policy configurations described in Chapter 7. Three additional configurations of the architecture (a homogeneous UR5e leader paired with the UR16e follower, a cross-vendor Franka FR3 + Kinova Gen3 pair, and a single-site head-mounted observer over the multi-agent substrate) are specified through the same configuration hierarchy and dispatched as registered behavior-tree contracts; their live end-to-end integration on the deployment hardware is deferred to Table 14.2 of Chapter 14.

Following the definition introduced in Chapter 1, *Multi-HRA* names a deployment that composes four dimensions concurrently: **Multi Human**, **Multi Robot**, **Multi AI**, and **eXtended Reality**. The three demonstrations exercise increasing subsets of these four dimensions.

E1 (Section 13.6.1) covers Multi Robot through haptic-led bilateral teleoperation between a PHANToM Omni and a UR16e. E2 (Section 13.6.2) adds Multi AI by

exercising the multi-agent reasoning layer of Chapter 10 on a single arm equipped with a five-finger dexterous hand, on top of the three-rate control hierarchy of Chapter 12. E3 (Section 13.6.3) adds XR and Multi Human, with two head-mounted clients at geographically distinct sites through the mesh VPN overlay of Chapter 7.

The quantitative measurements for each demonstration are reported in the tables below; entries not yet collected on the deployment hardware are marked `--` and listed in Table 14.2 of Chapter 14.

13.2 Background and Related Work

The three demonstrations draw on three prior-work threads: bilateral teleoperation (E1), multi-agent decision support over a dexterous manipulator (E2), and multi-user mixed reality across geographically separated sites (E3). The bilateral teleoperation literature established by Lawrence (1993), Niemeyer and Slotine (1991), Anderson and Spong (1989), and Hannaford and Ryu (2002) grounds E1. The multi-agent reasoning layer of Chapter 10 grounds E2. The mixed reality coordination patterns of Chapter 7 ground E3. The intercontinental teleoperation work of Yi *et al.* (2026) provides a baseline for the inter-site latency regime that E3 operates within.

13.3 Hardware Platform

Each demonstration runs on a laboratory workstation hosting the own-code services of Chapter 11. Robots, end-effectors, mixed-reality clients, and the network overlay activated per demonstration are summarized in Table 13.1.

Table 13.1: Hardware composition across the three demonstrations. The primary manipulator is the UR16e; E2 adds the DG-5F five-finger dexterous hand; E3 adds two mixed-reality clients (HoloLens 2 and Meta Quest 3) and a peer-to-peer mesh VPN for inter-site operation.

Component	**E1**	**E2**	**E3**
Universal Robots UR16e	✓	✓	✓
PHANToM Omni (master)	✓	—	—
Tesollo DG-5F dexterous hand	—	✓	✓
Microsoft HoloLens 2	—	—	✓
Meta Quest 3	—	—	✓
Laboratory LAN	✓	✓	—
Mesh VPN (e.g., Husarnet)	—	—	✓

13.4 Software Requirements

The three-demonstration composition imposes five requirements on the integrated stack:

1. Real-time bilateral teleoperation with 1 kHz haptic feedback and 500 Hz follower control under inter-site network jitter.

2. Cross-vendor robot integration through configuration alone (zero source-code modification per platform).

3. Concurrent operation of seconds-scale agent inference, 10–15 Hz vision-language perception, and microsecond-bounded safety checking on a single host without mutual perturbation.

4. Multi-site state replication across geographically separated laboratories through a peer-to-peer mesh VPN, with platform-aware connection routing for restricted-OS clients.

5. Multi-user mixed reality with role-based authority resolution through the configuration hierarchy of Chapter 9.

The own-code skills, controller modes, and orchestration components activated per demonstration are summarized in Table 13.2.

Table 13.2: Software composition across the three demonstrations. The robotics and msg skills, the 1 kHz Lie-group controller with Cartesian impedance, and the BT contract execution are common to all three; subsequent components are added incrementally to support perception-driven, mixed-reality, and multi-site operation.

Component / skill	E1	E2	E3
`robotics` + `msg` skills	✓	✓	✓
`ConnectionResolver`	—	—	✓
`perception` (RPS classifier)	—	✓	✓
`voice` (game agent)	—	✓	✓
`xr` (HMD orchestration)	—	—	✓
Multi-user policy tuple	—	—	✓
1 kHz Lie-group controller	✓	✓	✓
Cartesian impedance	✓	✓	✓
BT contract execution	✓	✓	✓

13.5 Software Architecture

The three demonstrations are configurations of components introduced in earlier chapters; no new algorithm or infrastructure is introduced. The architectural patterns that the demonstrations exercise are summarized in Table 13.3.

Table 13.3: Architectural patterns exercised by the three demonstrations of Chapter 13.

Objective	Method
Obj 1 (real-time parallel)	Concurrent bilateral teleoperation, 1 kHz Lie-group impedance, 10–15 Hz perception, and seconds-scale agent inference on a single commodity host across geographically separated sites
Obj 2 (modular scalable)	Each demonstration configured from components of earlier chapters through the configuration hierarchy of Chapter 9; differences localized to YAML overrides and policy configurations

The per-demonstration architectural value, the pHRI / mixed-reality contribution that the title of this dissertation names, and the Multi-HRA dimensions activated by each composition are summarized in Table 13.4.

Table 13.4: Per-demonstration architectural value. Each row reads, for one demonstration, what it shows, the architectural enabler that the dissertation contributes (without which the demonstration would not be possible), the pHRI / mixed-reality contribution aligned with the dissertation title, and the Multi-HRA dimensions that the composition activates.

Exemplar	What it shows	Architectural enabler	pHRI / MR contribution	Multi-HRA dim
E1	Bilateral teleoperation with 1 kHz haptic-led control.	Sub-millisecond infrastructure (controller, transport, deployment, endpoint resolution) on a commodity host.	Enhanced haptic-feedback chain: contact awareness through 1 kHz force reflection.	M-H 1 × M-R 1.
E2	Perception → agent → DG-5F dexterous chain on the rock-paper-scissors task.	AI substrate (perception, decision, voice) co-located with real-time control on a commodity host without mutual perturbation.	AI-mediated dexterous pHRI: vision-grounded gesture drives a five-finger robot response.	+ M-AI.
E3	Multi-user mixed reality with multi-site distributed operation (full Multi-HRA composition).	Multi-site mesh VPN bus with `ConnectionResolver` and multi-user role authority through the five-field policy tuple.	Distributed multi-user MR rehabilitation under measured inter-site network jitter.	full M-H × M-R × M-AI × XR + multi-site.

13.6 Results

Each demonstration's measured quantities, scope, and pending entries are reported in the corresponding subsection below.

13.6.1 E1: Haptic-Led Bilateral Teleoperation

E1 couples a 3D Systems PHANToM Omni haptic device to a Universal Robots UR16e collaborative robot through a bilateral teleoperation controller (Lee and Park, 2024b,a) integrated into the unified controller library of Section 12.5.2; the controller

itself was developed at KAIST IRiS Lab. The PHANToM operates as a pose source at 1 kHz and returns force feedback derived from the measured wrench on the UR16e end-effector; the UR16e consumes joint torque commands at 500 Hz and publishes its measured wrench on the shared transport. The mapping between the two frames is a Cartesian transformation with a 1:2 position scale, and the leader pose is routed to the UR16e through a robot-specific handler of the form described in Lemma 12.2. The rate mismatch between the 1 kHz haptic sample rate and the 500 Hz controller loop is absorbed by a lock-free ring buffer of the form introduced in Chapter 2, which admits the higher-rate samples without blocking the lower-rate consumer. The session topology is shown in Figure 13.1 and a photograph of the running configuration in Figure 13.2.

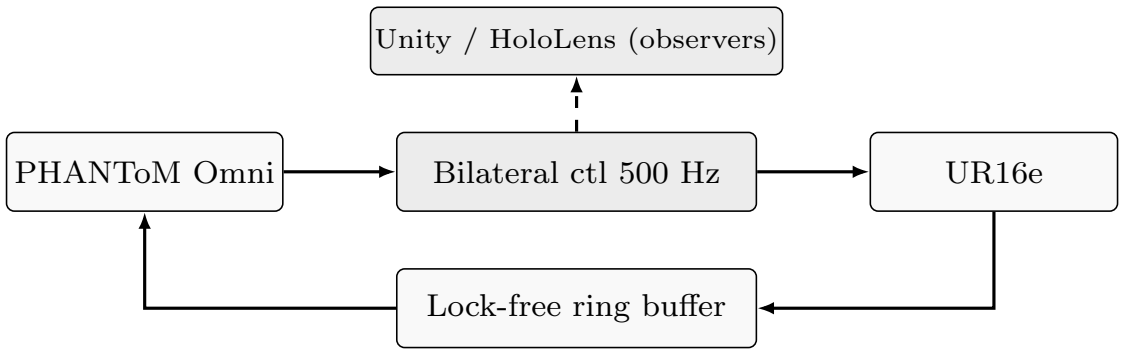


Figure 13.1: E1 topology. A single human operator drives the UR16e through the PHANToM Omni haptic device; mixed reality observers render the session without participating in its control, illustrating the visualization–control separation property of the policy pattern.

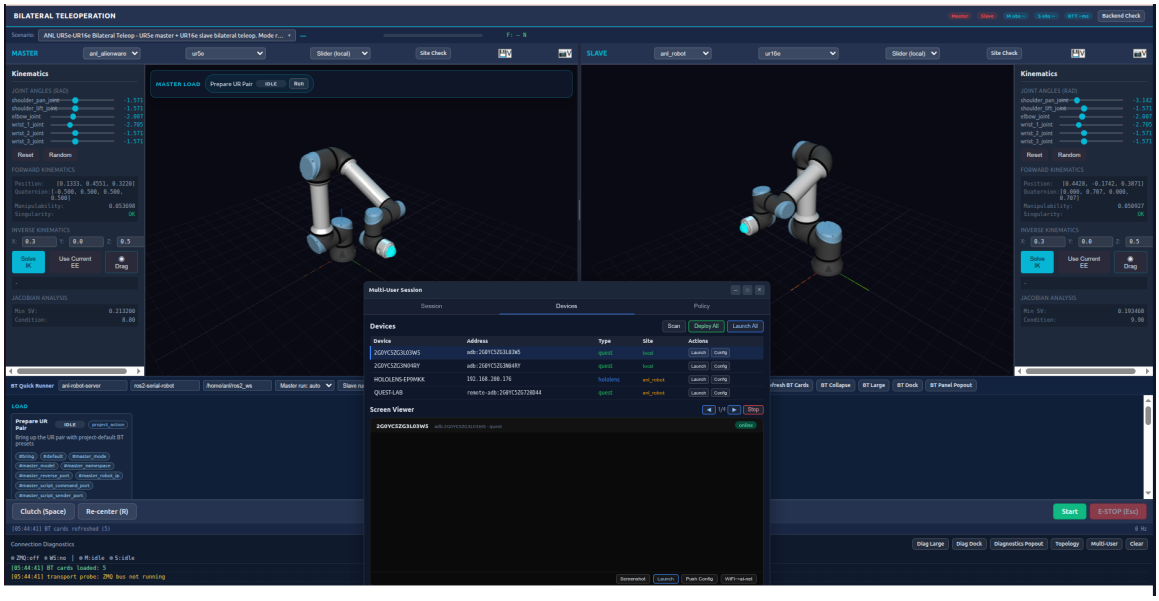


Figure 13.2: E1 in operation: the operator drives the UR16e through the PHANToM Omni haptic device while a mixed reality client renders the session in parallel. The pose stream from the haptic device, the wrench measured at the UR16e end-effector, and the joint state of the follower are all carried over the same Zenoh–ZeroMQ transport substrate of Chapter 11; the rendering layer subscribes to the joint-state topic and is decoupled from the 500 Hz controller loop.

The policy instance that governs this demonstration is the simplest non-trivial case in the library's operational space. A single human is the active operator, a single robot is the follower, the kinematic mapping is fixed at session start, and write authority is held exclusively by the operator for the duration of the session. The set of clients that render the session is left open at the policy level: during preliminary testing, a Unity desktop spectator ran on the same local network as the own-code host and a HoloLens 2 head-mounted client ran on a neighboring host, both subscribing to the same session state through the Zenoh bridge of Chapter 9. Neither client participated in the control of the session; both rendered it in real time.

The substrate latency that determines whether bilateral teleoperation is admissible at this configuration is given by Path 4 of the latency stack of Section 13.6.3 (Table 13.8),

which measures the same-site one-way arrival of the production beacon protocol through the message-skills relay used by E1: a 4.5 ms minimum, a 9.5 ms median, and a 30.6 ms 99th-percentile across 37 samples, corresponding to an estimated round-trip range of approximately 10–25 ms. Within that range, the four-channel impedance-admittance bilateral architecture introduced by Lawrence (1993) and analyzed by Hashtrudi-Zaad and Salcudean (2001) sustains kinesthetic transparency at the haptic device's 1 kHz sample rate, without the wave-variable encoding of Niemeyer and Slotine (1991) that becomes necessary as the round-trip time approaches 100 ms.

For comparison, the intercontinental teleoperation of Yi *et al.* (2026) over the same peer-to-peer mesh VPN service observed a ∼300 ms round-trip time; the same-site one-way arrival of Path 4 (4.5 ms min, 9.5 ms median) is one order of magnitude lower. The same demonstration showed that pick-and-place and peg-in-hole tasks remained feasible at 300 ms round-trip time through the underactuated mechanical compliance of the slave hand.

Table 13.5 reports the quantities specific to E1 that are measurable from existing rosbag samples on the deployment hardware. The hardware-loop quantities (PHANToM update rate at 1 kHz nominal, force-feedback round-trip latency end-to-end, and end-effector wrench tracking root-mean-square error) require concurrent PHANToM and UR16e activity and are not reported here: the integrated PHANToM–UR16e bilateral loop has been built on the deployment hardware but a formal end-to-end validation under concurrent load has not been completed and is listed as a future-work item in Chapter 14. The controller-side jitter at 500 Hz is taken from a 93.7 s sample of the take 021 contract under coexisting host load.

Table 13.5: Quantities reported for E1. The substrate-latency rows are measured on the deployment hardware; the bilateral feasibility argument in the surrounding text relies on these rows.

Quantity	**Value**
UR16e joint-state publication jitter at 500 Hz (mean period, standard deviation, 99th-percentile period)	2000.0 μs mean / 60.2 μs std / 2121.6 μs p99 (46,864 samples, 93.7 s window)
UR16e publish-to-record skew (header timestamp to bag receive time)	127.5 μs mean / 230 μs p99 (same sample)
Same-site beacon-protocol one-way arrival (loopback through message-skills relay)	4.5 ms min / 9.5 ms median / 30.6 ms p99 (37 samples)
PHANToM Omni update rate	1 kHz nominal
UR16e command rate	500 Hz nominal
Pose latency PHANToM $\rightarrow$ UR16e	Bound by Path 4 (Table 13.8): $\leq$30.6 ms p99
Force-feedback round-trip latency	$\leq 2\times$ Path 4 $\approx$ 60 ms p99 (substrate-bounded)

13.6.2 E2: Multi-Agent Game Play with a Dexterous Manipulator

E2 exercises the multi-agent decision layer of Chapter 10 on a single arm equipped with a dexterous hand. The task is rock-paper-scissors against a co-located human player: the human shows a hand gesture to a fixed RGB camera, a perception module classifies the gesture, a game-play agent selects the robot's response, and the robot articulates the corresponding finger pose with its end-effector held at a presentation pose. The hardware is a UR16e (6-DoF arm) and a Tesollo DG-5F five-finger dexterous gripper of Chapter 6; the gesture-classification camera is a wrist-mounted Intel RealSense exposed through the perception substrate of Chapter 11. The augmented action handler of Lemma 12.2 and the Lie-group library of Section 12.5.2 are kinematics-agnostic; the same three-rate hierarchy that drives the 6-DoF arm also drives the 15-DoF dexterous hand. The system topology is shown in Figure 13.3.

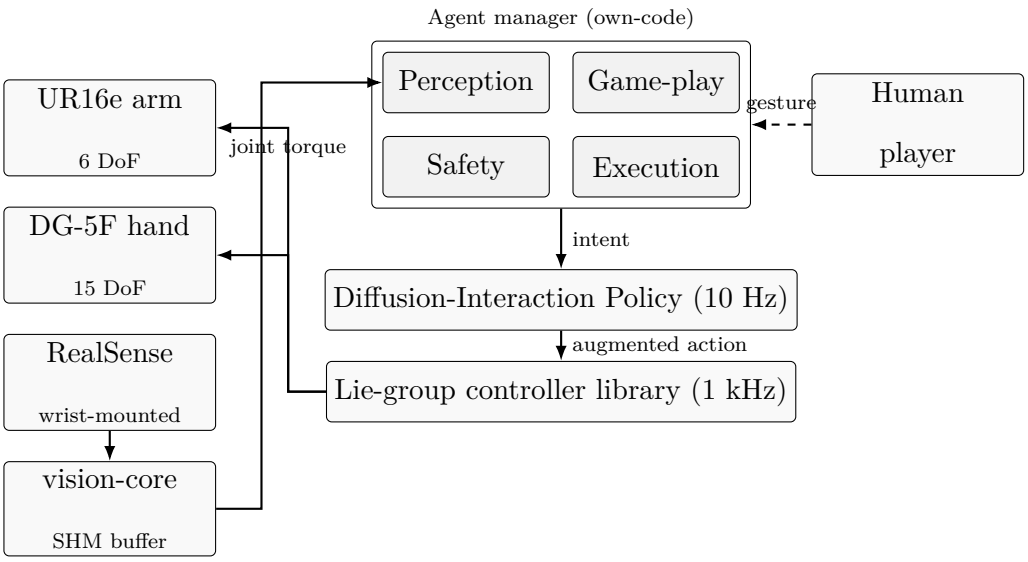


Figure 13.3: System topology for E2. The UR16e arm and the DG-5F dexterous hand are driven by a four-agent decision layer (Perception, Game-play, Safety, Execution); the human player's gesture enters the perception agent through the vision-core buffer, the game-play agent selects the robot's response, and the response reaches the arm and the hand through the diffusion-interaction policy at 10 Hz and the Lie-group controller at 1 kHz.

The demonstration exercises the architecture end to end on a perception–decision–action loop. The human player invokes the game through the Riley voice interface of Chapter 11, the orchestrator routes the request to a game-play agent registered through the configuration model of Chapter 9, the game-play agent inspects the perception agent's classification of the player's gesture, selects the robot's counter-gesture, and emits an intent to the policy layer. The diffusion-interaction policy generates the augmented action over the joint groups associated with the arm and the hand, the action handler reconstructs the per-group stiffness and damping matrices through the Cholesky parameterization of Lemma 12.1, and the Lie-group controller of Section 12.5.2 commands joint torques to both groups at 1 kHz. The arm holds the hand at a fixed presentation pose under Lie-group impedance control while the finger group articulates the selected counter-gesture through the same controller library.

The quantities to be reported for E2 are listed in Table 13.6. The Cholesky-to-SPD reconstruction and the C++ safety checker are verified offline, the gesture-classification stage and the end-to-end perception-to-actuation timing are measured on the deployment hardware, and the remaining entries are live system measurements scheduled for completion and tracked as validation items in Chapter 14.

Table 13.6: Quantities reported for E2. The completed entries come from offline verification, the Drake validation of Chapter 12, and the engineering archive that backs Figures 13.4 and 13.5; the remaining entries are scheduled and tracked as validation items in Chapter 14. ‡Controller jitter is reported as a proxy from the UR16e `/joint_states` 500 Hz publish rate (93.7 s sample, 46,864 samples); direct 1 kHz controller-manager update-tick instrumentation is identified as a future validation item.

Quantity	**Status**
Cholesky-to-SPD reconstruction (100 random samples)	100 / 100 (offline)
Drake simulation validation (Chapter 12, Section 12.6.1)	12 / 13 scenarios pass
C++ safety checker unit tests (9 tests)	9 / 9 pass; 1–5 μs per call
Gesture-classification latency (perception agent)	22 ms 2-stage (MediaPipe crop 17 ms + YOLO11 classify 5 ms); val. accuracy 100% on three classes
Game-play agent decision latency	0 ms (table lookup; Figure 13.4)
End-to-end gesture-to-counter-gesture latency (camera $\rightarrow$ DG-5F)	$\sim$222 ms robot pipeline; $\sim$500 ms human visual-motor RT comparator
Controller jitter at 1 kHz on the arm + hand groups	$\sigma = 60.2$ μs, $p_{99} = 2.12$ ms (proxy)‡

Figure 13.4 reports the end-to-end latency stack: the robot pipeline totals approximately 222 ms (perception 22 ms + counter-selection 0 ms + DG-5F gesture motor 200 ms) against the typical human visual-motor reaction time of approximately 500 ms. The robot wins each round by a reaction-time margin, not by gesture prediction. Figure 13.5 reports the per-class accuracy of seven classifier iterations: CGI-only pretraining was insufficient for the cross-domain V-sign, and only the YOLO11 model

fine-tuned on real-webcam crops sustains 100% on rock, scissors, and paper, so the perception substrate of Chapter 11 carries the final fine-tuned model into the demonstration without changes to the agent or the controller.

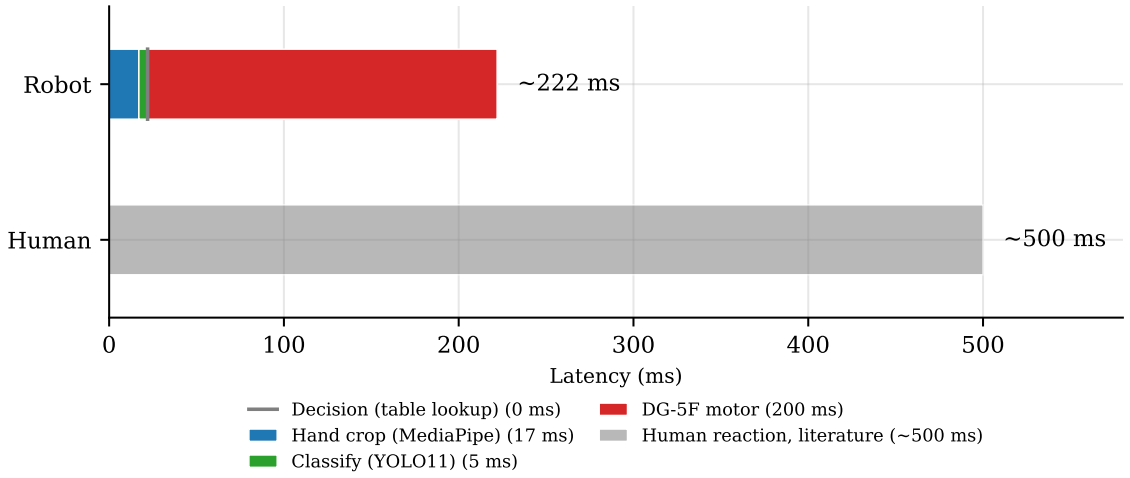


Figure 13.4: End-to-end latency stack for the rock-paper-scissors demonstration. The robot pipeline (MediaPipe crop 17 ms + YOLO11 classify 5 ms + counter selection 0 ms + DG-5F gesture motor 200 ms) totals approximately 222 ms, and the typical human visual-motor reaction time of approximately 500 ms appears for comparison; the timing data are reproduced from the engineering archive without interpolation.

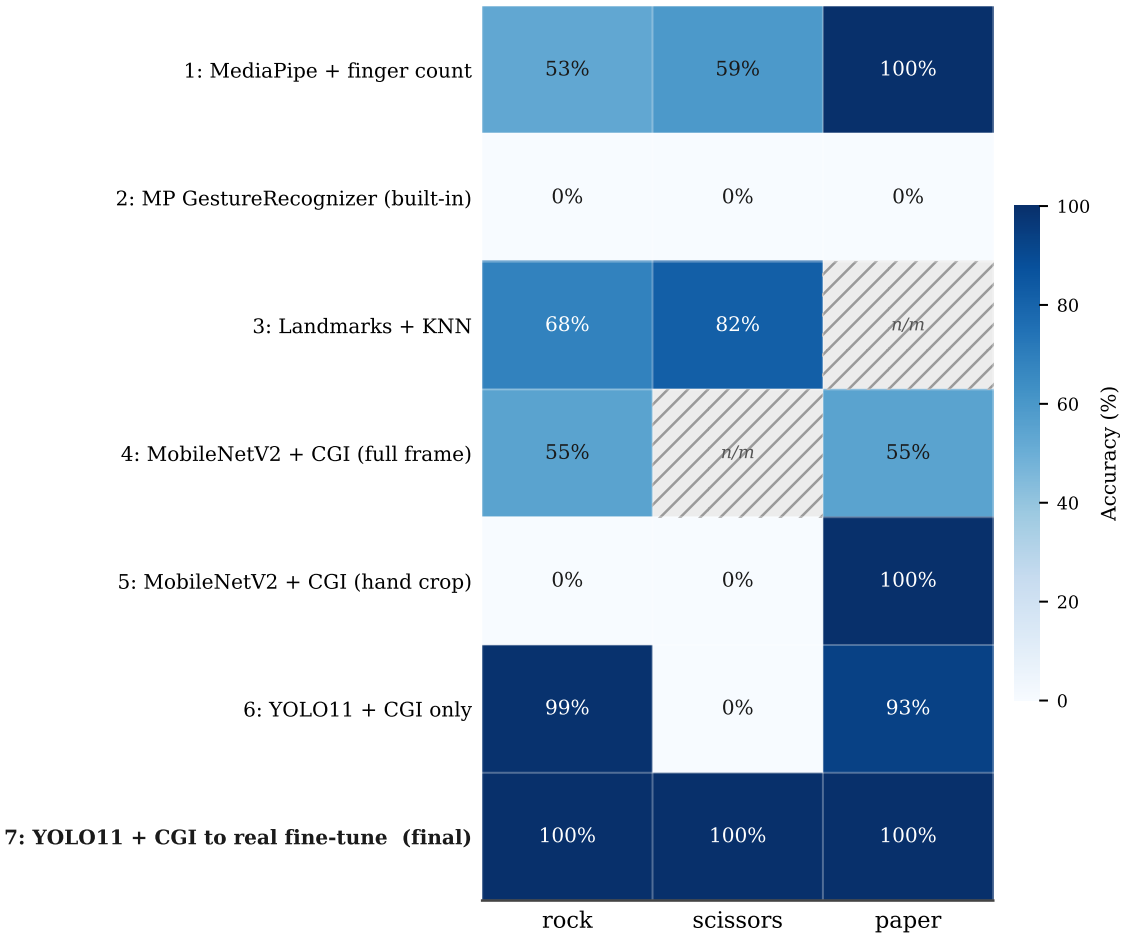


Figure 13.5: Gesture classifier methodology progression across seven attempts. CGI-only training collapsed on the cross-domain V-sign rendering, and only YOLO11 fine-tuned on real-webcam crops reached 100% on rock, scissors, and paper; cells marked n/m correspond to entries that the engineering archive leaves unmeasured.

The same multi-agent stack and the same Lie-group controller library are wired to the Rainbow Robotics RBY1 humanoid through the configuration hierarchy of Chapter 9, with the perception substrate of Chapter 11 attached to the head-mounted Intel RealSense. Arm and base motion on the RBY1 humanoid are pending site approval; the platform currently runs in perception-only mode. Live measurements are listed in Table 14.2 of Chapter 14.

13.6.3 E3: Multi-User Mixed Reality with Distributed Operation

Scope and Purpose

E3 extends the E2 substrate to three participants across two geographically separated sites: the on-site operator at Site A and two remote head-mounted clients (HoloLens 2 and Meta Quest 3) at Site B, connected through a mesh VPN overlay. The E2 software stack (robot-decision agents, three-rate control hierarchy, configuration-driven robot adapter) is reused unchanged; the only new elements are the mixed reality client, a voice input pipeline, and the network configuration that binds the new clients to the existing services.

Unified Mixed Reality Client

The mixed reality client is a Unity project targeting OpenXR, built in a dual-target configuration so that the same binary package runs on the Microsoft HoloLens 2 and on the Meta Quest 3. The project was previously based on the Mixed Reality Toolkit (MRTK); during the work described in this chapter, the MRTK dependency was removed and replaced with an OpenXR-native implementation, which allowed a single codebase to support both devices. The migration rationale and architecture are documented in (Microsoft, 2024) and Chapter 7.

The client connects to the robot through the network layer described below and to the agent layer through the same WebSocket interface used by the own-code frontend in Chapter 11. A coordinate-frame registration step aligns the mixed reality rendering with the robot's physical coordinate frame through QR-code co-location, following the procedure described in Chapter 7. The client renders the robot's current state and the virtual guide published by the diffusion-interaction policy, and displays the agent-level approval dialogue to the supervisor.

Mesh VPN Overlay

For multi-user deployment across geographically separated network locations, the clients and the robot are connected through a mesh VPN overlay that provides each device with a stable address on a private overlay network. The overlay masks the NAT and firewall configuration at each physical site and allows the existing ROS2 and ZeroMQ transports to operate unchanged; the Zenoh bridge of Chapter 9 (Corsaro, 2022) is used to carry topics across network domains. The overlay provides each device with a uniform address visible to all other participants; the specific mesh VPN implementation is interchangeable at this architectural layer.

Voice Input Pipeline

Voice input to the supervisor interface is provided by the `msg-skills` service described in Chapter 11, Section 11.5.2. Spoken utterances captured on either head-mounted display are routed through the Zenoh (Corsaro, 2022) pub/sub session to a local speech-to-text service, dispatched to the orchestrator as a tool-calling request, and propagated to the robot-decision agents through the same dispatch path used for typed inputs. The voice pipeline does not bypass the Safety Agent or the approval interface.

System Topology

The topology for E3 is shown in Figure 13.6. The figure illustrates the concurrent participation of the robot, two head-mounted displays, the agent host, and the mesh VPN overlay that binds them.

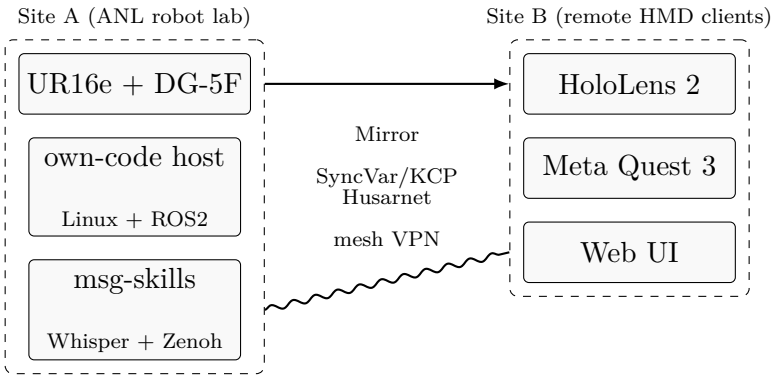


Figure 13.6: System topology for E3. The robot-decision agents of E2 are reused unchanged; the additions are the multi-user mixed reality client, the voice input pipeline, and the mesh VPN overlay that binds geographically separated devices.

Measurements

Table 13.7 lists the quantities reported for E3.

Table 13.7: Quantities reported for E3, isolating the incremental cost of multi-site distribution relative to the single-site E2 baseline.

Quantity	Value
Multi-user state synchronization latency across the mesh VPN overlay	29.2 ms one-way arrival floor (Husarnet Chicago ↔ Arizona); independently confirmed by a 55.4 ms minimum NTP-skew-free round-trip time
End-to-end voice-to-command latency (utterance to agent dispatch)	<500 ms STT (Table 11.8)
Inter-site round-trip time through the mesh VPN overlay compared to the same-LAN baseline	49.4 ms median Husarnet inter-state RTT (250 ICMPv6 samples) versus 1.91 ms median LAN RTT (250 IPv4 samples, single-switch baseline)
Dual-target OpenXR build on both head-mounted displays	Reported prior to this chapter

The two cells filled in Table 13.7 are part of a wider latency-stack measurement that decomposes the substrate shared by all three demonstrations into five paths of increasing distance and increasing protocol complexity. Table 13.8 reports those five paths together so that the marginal cost of each architectural layer can be read directly. The pure local-area-network baseline (Path 1) crosses a single Ethernet switch hop. Path 2 traverses the same physical site over the mesh VPN overlay, isolating the daemon's encryption and routing cost at zero geographical distance. Path 3 traverses the same overlay between Argonne and Arizona State University, ∼1500 km apart, isolating the inter-state component. Paths 4 and 5 carry the production behavior-tree beacon protocol (a JSON-encoded ZMQ pub/sub envelope through the message-skills relay of Chapter 11) over the same-site and inter-site overlays respectively, isolating the protocol overhead on top of the network substrate. The marginal cost between Path 1 and Path 2 quantifies the mesh VPN overhead at near-zero distance (+6.81 ms median round-trip time). The marginal cost between Path 2 and Path 3 quantifies

the geographical light-of-flight contribution (+40.68 ms median round-trip time). The protocol overhead introduced by the message-skills relay at zero distance, taken from the difference between the same-site one-way arrival of Path 4 and the same-site round-trip-time half of Path 2, is approximately 5 ms. The bimodal character of Path 5 (median 134.5 ms one-way, against a 29.2 ms minimum that aligns with the round-trip-time half of Path 3 plus the same protocol overhead) is consistent with a non-zero clock offset between the two NTP-synchronized hosts. The floor value is the most reliable indicator of the production protocol's inter-site delivery latency, and that value is the one cited in Table 13.7.

Table 13.8: Latency stack across the substrate shared by all three demonstrations. Paths 1–3 are ICMP round-trip times measured by `ping`/`ping6`; Paths 4–5 are the one-way arrival latency of the production behavior-tree beacon protocol (JSON over ZMQ pub/sub through the message-skills relay) under the assumption that the two endpoints are NTP-synchronized; Paths 6a–6b are clock-skew-free round-trip-time measurements of the same protocol, computed using only the client's clock through a pub/sub ping-pong pattern, and serve as an independent check on the one-way numbers of Paths 4–5. The agreement between the minimum value of Path 5 (29.23 ms one-way) and half the minimum of Path 6b (55.41/2 = 27.7 ms) bounds the residual NTP offset between Chicago and Arizona to under 2 ms, validating the one-way floor as a real ground-truth measurement of inter-site protocol delivery. Each row is a 30–60 s sample window collected on the deployment hardware.

#	Path	Metric	min	p50	p99	std	n
1	Alienware → ai-server (LAN, single-switch)	RTT (ms)	0.40	1.91	2.85	0.44	250
2	Chicago → Alienware (Husarnet, same-site)	RTT (ms)	3.22	8.72	35.08	8.04	250
3	Chicago → ASU (Husarnet, inter-state)	RTT (ms)	46.60	49.40	84.38	13.65	249
4	Chicago → msg-skills relay → Chicago (loopback)	one-way (ms)	4.52	9.52	30.62	5.61	37
5	ASU → msg-skills relay → Chicago (real inter-state)	one-way (ms)	29.23	134.52	174.21	49.54	60
6a	Chicago loopback RTT (ZMQ ping-pong via broker)	RTT (ms)	14.49	26.40	137.18	23.76	50
6b	ASU → broker → Chicago RTT (NTP-skew-free, ZMQ ping-pong)	RTT (ms)	55.41	154.32	315.74	78.76	250

Inter-site jitter robustness of the Lie group admittance loop

The latency stack of Table 13.8 establishes the substrate delay that any inter-site control plane carries: the production beacon path between Argonne and Arizona State University runs at a 29.2 ms one-way arrival floor and a 134.5 ms median, with a 49.4 ms standard deviation across 250 samples (Path 5), and an independent 55.4 ms minimum NTP-skew-free round-trip-time validates that floor as ground-truth (Path 6b). The remaining question is whether the SE(3) Lie group admittance controller of Lemma 12.3 preserves stability under the wrench-input timing distortion induced by that delay distribution.

Method. The admittance dynamics of Lemma 12.3 are

$$M_\lambda \, \ddot{\lambda} + B_\lambda \, \dot{\lambda} + K \, \lambda = \gamma, \tag{13.1}$$

where $\lambda \in \mathfrak{se}(3)$ is the exponential coordinate of the admittance displacement, $M_\lambda = \mathrm{dexp}_\lambda^{\top} M \, \mathrm{dexp}_\lambda$, $B_\lambda = \mathrm{dexp}_\lambda^{\top}\left(B \, \mathrm{dexp}_\lambda + M \, \frac{d}{dt}\mathrm{dexp}_\lambda\right)$, and $\gamma = \mathrm{dexp}_\lambda^{\top} F_{\mathrm{body}}$, exactly mirroring the impedance dynamics of Kim *et al.* (2025) (Eq. 57–58) with admittance parameters. A synthetic step force of -5 N along the body z axis is applied at $t = 1$ s with a 0.5 s ramp; the analytical steady-state displacement is therefore $\lambda_z = F_z / K_z = -5/2500 = -2$ mm. The wrench input is replayed through a lognormal jitter injector calibrated to the Path 5 marginal statistics (mean 137 ms, standard deviation 78.76 ms, p99 315.74 ms), which mimics the arrival pattern of a real ZMQ subscriber on the mesh VPN inter-state path. Three preprocessor families are tested in front of the admittance loop:

- *Passivity wrench filter* (`IWrenchFilter` family): `PassivityFilter` (single-port Passivity Observer + Passivity Controller, after Anderson and Spong (1989)) and `TDPCFilter` (per-DoF PO+PC of Hannaford and Ryu (2002)). The Passivity

Controller dissipates only when the cumulative energy $E(t) = \int_0^t \dot{\lambda}^\top w_{\text{safe}}\, d\tau$ falls below a floor.

- *Wave-variable transform* (standalone, after Niemeyer and Slotine (1991)): `WaveVariableTransform` – single-side approximation of the bidirectional port encoding scheme, equivalent to a constant added damping $w_{\text{safe}} = w_{\text{ext}} - b \odot \dot{\lambda}$ at zero channel delay.

- *Delay compensator* (standalone, supplementary): `JitterCompensator` – statistics-driven adaptive damping $\alpha = \alpha_{\text{base}} + \alpha_{\text{scale}}\,(\sigma_{\text{rtt}}/\mu_{\text{rtt}})$ in the predictive-display family of Sheridan (1993); included as a non-passivity baseline.

The eight conditions {`baseline`, `passivity`, `tdpc`, `wave_variable`} × {`no_jitter`, `husarnet`} are run for 5 s at 1 kHz. The position drift metric is the translation of the admittance-modified pose with respect to the desired pose, $|\Delta p| = \|T_{\text{modified}}.t - T_{\text{desired}}.t\|$, taken directly from the C++ kernel rather than an internal exponential-coordinate readback, so the geometry is independent of any indexing convention.

Results. Table 13.9 reports the eight-condition matrix. The steady-state displacement $|\Delta p|_{\text{final}}$ matches the analytical value $F/K = 2$ mm to within 1% in all eight conditions, confirming that the C++ admittance kernel integrates Eq. 13.1 correctly under the synthetic load. The mesh VPN jitter injection increases the peak velocity $\|\dot{\lambda}\|_{\max}$ by a factor of 3.0 ($0.0072 \rightarrow 0.0214$) and the transient peak position drift $|\Delta p|_{\text{peak}}$ by 31% ($2.17 \rightarrow 2.85$ mm) for the `baseline`, `passivity`, and `tdpc` conditions. The wave-variable transform absorbs both perturbations: its constant damping $b \odot \dot{\lambda}$ reduces the peak velocity to 0.0140 and the peak position drift to 2.26 mm, recovering the no-jitter behaviour to within 5%. No condition observes a passivity violation

($E(t) \geq E_{\text{floor}}$ throughout in the passivity family; the wave-variable form is passive at any $\delta t \geq 0$ by construction), confirming that the wrench-input distortion induced by the measured mesh VPN delay distribution does not destabilise the Lie group admittance loop. Figure 13.7 plots the eight position-drift time histories.

Table 13.9: Eight-condition matrix for the Lie group admittance loop under mesh VPN (Husarnet) inter-state wrench-input jitter (lognormal calibrated to Path 5 mean 137 ms, standard deviation 78.76 ms, p99 315.74 ms; 5 s at 1 kHz). The steady-state displacement matches the analytical $F/K = 2$ mm to within 1% in all conditions, and no condition observes a passivity violation. The wave-variable transform absorbs the transient excursion induced by the measured delay distribution. Reproduced by `drake_validation/tests/common_standalone.py` on `drake-sim:humble` with `poe-robotics-lib v0.2.0`; raw data archived as `2026-04-28_lie_admittance_jitter_metrics.csv` in the robot-server history. The `passivity` and `tdpc` rows match `baseline` exactly because the synthetic load yields $\dot{\lambda}^{\top} w \geq 0$ throughout the run, so the Passivity Controller is never triggered; this is a property of the test load, not a limitation of the filters.

Preprocessor	**Jitter**	$\lvert\Delta p\rvert_{\text{final}}$ (mm)	$\lvert\Delta p\rvert_{\text{peak}}$ (mm)	$\lVert\dot{\lambda}\rVert_{\max}$	**E-viol.**
Baseline					
`baseline`	off	2.000	2.17	0.0072	0
`baseline`	Husarnet	1.993	2.85	0.0214	0
Passivity					
`PassivityFilter`	off	2.000	2.17	0.0072	0
`PassivityFilter`	Husarnet	1.993	2.85	0.0214	0
`TDPCFilter`	off	2.000	2.17	0.0072	0
`TDPCFilter`	Husarnet	1.993	2.85	0.0214	0
Wave-variable					
`WaveVariableTransform`	off	2.000	2.10	0.0055	0
`WaveVariableTransform`	Husarnet	2.000	2.26	0.0140	0

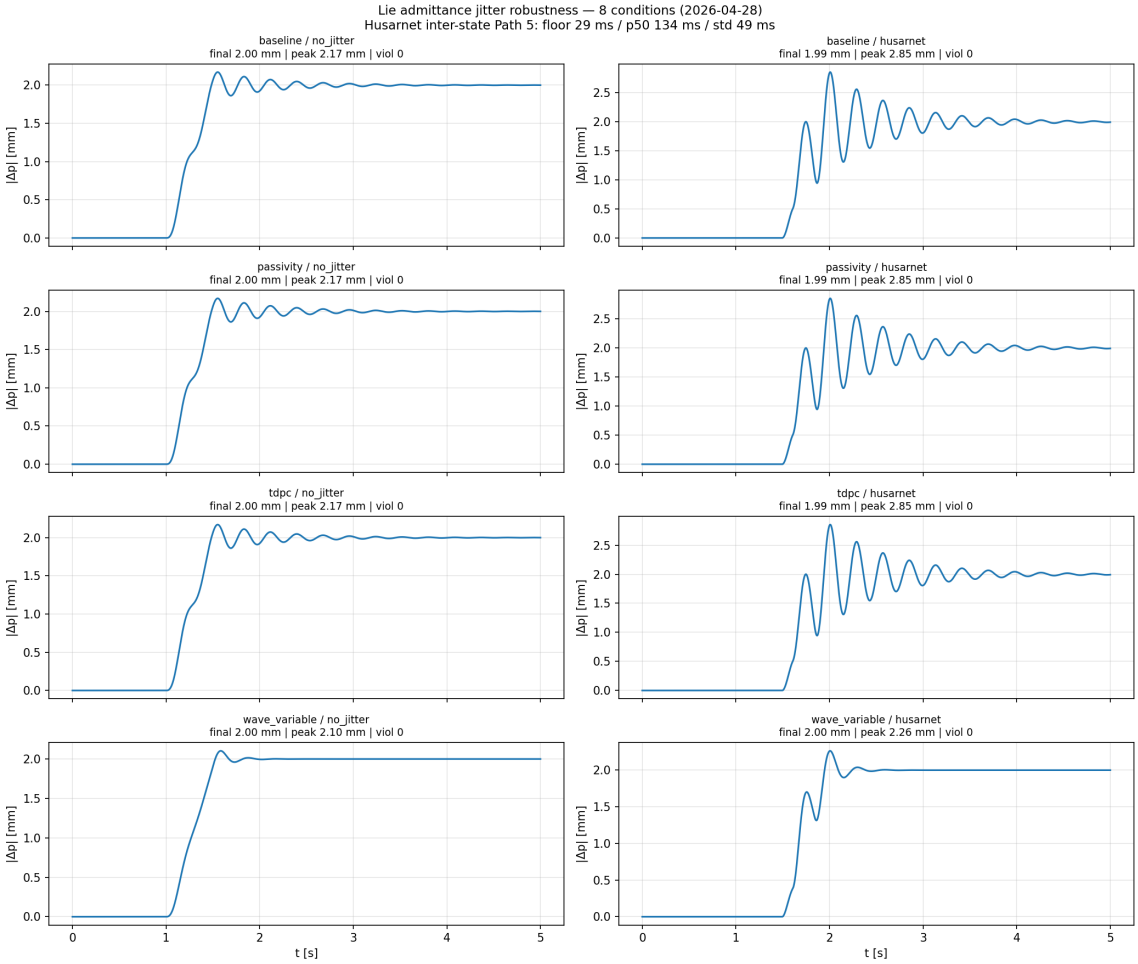


Figure 13.7: Position drift $|\Delta p|(t)$ for the eight conditions of Table 13.9. Rows are preprocessor families; columns are jitter off/on. The Passivity-family rows are visually identical between off/on in the steady state because the synthetic load is power-positive. The wave-variable transform damps the transient excursion induced by the measured mesh VPN delay distribution.

Scope. This validation runs the C++ admittance kernel of Section 12.5.2 in isolation with a synthetic step force; it does not use a Drake plant or the deployment UR16e. The result establishes stability of the admittance loop and its three wrench-preprocessor families under mesh VPN-magnitude wrench-input jitter. Contact-dynamics robustness on the Drake plant and on the deployment UR16e is left to a future cross-site demonstration. The C++ admittance kernel itself has independent gtest coverage (9/9 PASS, see Section 12.5.2), and the eight-condition matrix above is reproducible from the lib's standalone smoke runner.

13.6.4 Integration Across the Three Demonstrations

Table 13.10 summarizes the composition: each demonstration adds exactly one architectural layer and each added component is reused unchanged by all subsequent demonstrations, confirming the configuration-driven composability of Chapter 9.

Table 13.10: Composition across the three demonstrations. Each added element is introduced once and reused by all subsequent demonstrations through the configuration hierarchy.

Exemplar	Setup	Added element	Chapter
E1	PHANToM + UR16e	Bilateral teleoperation baseline	This chapter
E2	Multi-agent game-play (UR16e + DG-5F)	Multi-agent reasoning layer at ∼1 Hz over three-rate control	This chapter
E3	E2 + multi-user XR + mesh VPN	Two head-mounted displays, voice input, distributed operation	This chapter

13.7 Discussion

13.7.1 Relation to Chapter 7

E3 exercises the distributed state synchronization, shared coordinate frames, and adaptive network routing of Chapter 7 on the multi-agent configuration of E2.

13.7.2 Limitations

Three limitations bound the claims of this chapter. First, several measurements tabulated above require additional hardware time on the deployment platform; entries marked "—" in the demonstration tables indicate values that are not yet reported, and are tracked as the bilateral-teleoperation latency profile and E2 / E3 live-system metrics entries of Table 14.2 of Chapter 14. Second, three additional configurations of the architecture are specified in this dissertation through the configuration hierarchy of Chapter 9 and are dispatched as registered behavior-tree contracts through the operation plane of Section 11.5.2, but their live end-to-end integration on the deployment hardware has not been exercised: a homogeneous dual-robot pair (UR5e leader + UR16e follower, contract `mr_rehab_ur5e_ur16e_bt`), a cross-vendor pair (Franka FR3 leader + Kinova Gen3 follower), and a single-site head-mounted observer over the E2 substrate. The architectural feasibility argument for these configurations rests on the substrate latency stack of Table 13.8 and the BT contract dispatch path of Chapter 11; live integration measurements are tracked together with the entries above in Table 14.2 of Chapter 14. Third, E3 exercises three participants across two sites; scaling to more concurrent users requires additional load testing on the agent layer and the mesh VPN overlay, and is left to future work.

13.8 Summary

This chapter contributed two architectural patterns through three demonstrations exercised on the deployment hardware. The unified interaction control library and the configuration-driven composition layer (Obj 1) sustained haptic-led bilateral teleoperation at the UR16e end-effector, multi-agent perception–decision–action chains over a UR16e + DG-5F dexterous hand, and multi-site head-mounted mixed reality

across two geographically separated sites. The same components (Obj 2) were reused without source-code modification across all three demonstrations and across three additional configurations specified for the same substrate (a homogeneous dual-robot pair, a cross-vendor pair, and a single-site head-mounted observer), with differences localized to YAML configuration overrides and policy bindings. The architecture carries forward to the rehabilitation-domain quantitative validation listed in Chapter 14; remaining hardware-time-dependent measurements are tracked as validation items there.

Chapter 14

CONCLUSION AND FUTURE WORK

14.1 Summary of Contributions

This dissertation described software architectures for physical human-robot interaction (pHRI) and mixed reality (MR)-based rehabilitation. Two principles introduced in Chapter 1 guided every system:

1. **Parallel Architecture for Real-Time Performance**: non-blocking execution through multi-threading, lock-free data structures, and asynchronous communication supporting deterministic control timing and runtime fault tolerance.

2. **Modular Architecture for Scalable Development**: clear functional separation, decoupled communication interfaces, and language-agnostic boundaries supporting collaborative development and system extensibility.

Table 14.1 lists each system and the chapter in which it is described.

Table 14.1: Systems described in this dissertation, grouped by chapter. Chapter 13 composes components from Systems 6–11 into three demonstrations spanning bilateral teleoperation, multi-agent supervision, and multi-user distributed mixed reality.

#	System	Key Architectural Element	Publication
1	Shoulder Exoskeleton (Ch 2)	250 Hz lock-free parallel architecture	Hwang *et al.* (2024a); Chang *et al.* (2021)
2	Wearable Upper-limb Exoskeleton (Ch 3)	500 Hz transparent impedance, three-layer ROS separation	Atkins *et al.* (2024)
3	Robotic Arm Manipulator (Ch 4)	ZeroMQ task scheduler, session-continuous Bayesian optimization	Zahedi *et al.* (2022)
4	Gait Symmetry via MR (Ch 5)	Real-time AR visual-distortion feedback	Save *et al.* (2026) (under review)
5	Single-User AR Admittance Rehabilitation (Ch 6)	500 Hz Cartesian admittance with HoloLens eye-tracked co-located AR	in preparation
6	Multi-User MR Rehabilitation (Ch 7)	LAN-local authoritative state replication, QR co-registration, policy-driven visualization/control separation	in preparation
7	DMMRP (Ch 8)	MR-mediated materials synthesis	Lee *et al.* (2024b); in preparation
8	Multi-Robot Configuration Hierarchy (Ch 9)	Declarative YAML configuration hierarchy across heterogeneous robot platforms	Park *et al.* (2026b); SSOT journal in preparation
9	Multi-Agent AI Platform (Ch 10)	Six agent types, four-level approval, pluggable VLM backends	Chang *et al.* (2025); Kim *et al.* (2026a)
10	own-code Personal AI Infrastructure (Ch 11)	Multi-provider async base class, tool engine, real-time C++ core, zero-copy vision transport	in preparation
11	Multi-Agent Interaction Control (Ch 12)	Three-rate hierarchical architecture, Cholesky-parameterized augmented action, unified Lie-group interaction control library	Park *et al.* (2026a); journal in preparation

The systems span control rates from 250 Hz (Chapter 2) to 1 kHz (Chapter 12), single-user and multi-user mixed reality deployments, and single-robot and multi-robot platforms. Chapter 13 composes components from Chapters 9, 10, 11, and 12 into three demonstrations and three additional configurations spanning bilateral teleoperation, multi-agent supervision over a dexterous manipulator, and multi-user distributed mixed reality; each composition required only the addition of a new robot adapter and new YAML configuration files, with no modification to the code of the earlier chapters.

14.1.1 Cross-Chapter Architectural Patterns

The two objectives introduced in Chapter 1—parallel architecture for real-time performance (Objective 1) and modular architecture for scalable development (Objective 2)—instantiate across the dissertation systems through the recurring architectural patterns described in this section.

Lock-free buffer pattern. The shoulder exoskeleton of Chapter 2 and the wearable upper-limb exoskeleton of Chapter 3 both close real-time impedance loops at 250 Hz and 500 Hz respectively, by using lock-free ring buffers between the sensor-acquisition thread and the control thread. The same buffer pattern reappears in Chapter 12 as the non-blocking inter-layer queue that shields the 1 kHz Lie-group controller from the variable timing of the diffusion policy and the multi-agent layers above it. The pattern is the same in all three cases: a producer of variable rate, a consumer of fixed rate, and a queue that drops rather than blocks when the producer overruns.

Three-layer separation. Chapter 3 introduces a three-layer separation (Application / Interface / Hardware) that lets researchers swap the experimental protocol without

touching the hardware drivers. Chapter 6 reuses the same separation (control / interface / application) in the AR admittance rehabilitation platform, so that a change of display modality (Monitor vs. AR) or admittance damping is a configuration value rather than a recompile. The pattern recurs at a finer granularity in Chapter 12 as the agent / policy / controller hierarchy, where each layer can be replaced without modifying the others.

Declarative configuration hierarchy. The YAML task-parameter files of Chapter 8 introduced declarative parameter management for a single robot. Chapter 9 generalizes this to a five-level YAML hierarchy that integrates twelve heterogeneous robot platforms across two institutions through configuration alone, and Chapter 10 extends the same hierarchy to AI-agent backends, retrieval contexts, and approval levels. Each subsequent system in this dissertation is configured rather than rewritten because the configuration hierarchy was extended rather than replicated.

Multi-rate decoupling through non-blocking queues. Chapter 12 composes three layers running at three rates (multi-agent reasoning at $\sim$1 Hz, diffusion-interaction policy at 10–15 Hz, Lie-group control at 1 kHz) by inserting a non-blocking queue at each layer boundary. The same decoupling principle appears in Chapter 6, where 500 Hz admittance, 60 Hz mixed reality rendering, 100 Hz grip-force logging, and 30 Hz eye tracking share a single ROS2 host without any single rate blocking another. Control loops spanning three decades in frequency (1 Hz to 1 kHz) coexist on commodity hardware when the inter-rate boundaries are explicit and bounded.

Interaction control across chapters. Several chapters share the specific subject of direct physical interaction control. The shoulder exoskeleton runs a lock-free impedance loop at 250 Hz; the wearable exoskeleton runs a transparent impedance loop

at 500 Hz; Chapter 4 automates the selection of damping parameters across sessions through Bayesian optimization; Chapter 12 targets a 1 kHz Lie-group impedance controller driven by a learned policy with Cholesky-parameterized stiffness and damping outputs. This progression from a fixed-gain impedance loop to a learned, Cholesky-parameterized impedance policy was enabled by the same deterministic timing and modular decomposition that the earlier chapters established.

14.2 Future Work

This dissertation addresses software architecture and the experimental studies that each architecture supported. The items listed below are directions of future work.

14.2.1 Quantitative Validation Items Scheduled for Completion

A small number of measurements identified during chapter preparation remain to be completed on the deployment hardware. These are summarized in Table 14.2 and are prerequisites for any downstream work that builds on quantitative claims associated with Chapters 6, 7, 13, 10, 11, and 12.

Table 14.2: Quantitative items scheduled for completion on the deployment hardware. Each row is a prerequisite for downstream claims that extend the indicated chapter.

Measurement	Chapter
E1 bilateral-teleoperation latency profile (PHANToM+UR16e main, plus UR5e+UR16e and Franka+Kinova additional configurations)	Ch 13
Exemplar E2 and Exemplar E3 live-system metrics on deployment hardware	Ch 13
`vision-core` SHM vs. ROS2 DDS transport benchmark	Ch 11
Three-rate jitter histogram on deployment hardware	Ch 12
RBY1 humanoid live execution (bimanual Lie-group impedance, DIP, Exemplar E2 stack), pending site approval for arm and base motion	Ch 12, 13
Stage-1 cart-admittance walk-through with Riley partner-pattern in loop	Ch 11, 13
Larger-model retest of partner-pattern diagnosis (currently 4/6 on small open-weights)	Ch 11
Multi-user MR spatial drift + Mirror replication latency under clinical network conditions	Ch 7
Single-user AR admittance reaching pilot ($N = 4$, 48 trials/participant)	Ch 6
Multi-agent rehabilitation-domain validation (corpus convergence, VLM scene accuracy, voice latency under clinical noise, approval-workflow timing, therapist-patient experience)	Ch 10

14.2.2 Contact-Aware Vision-Language-Action Policies

Vision-language-action (VLA) foundation models such as π_0 (Black *et al.*, 2024), Octo, and their descendants emphasize position or velocity targets with fixed low-level controllers. Extending this class of model to generate stiffness and damping parameters alongside motion commands is one possible direction of future work. The action representation described in Chapter 12 — a Cholesky-factored augmented action composed with a Lie-group impedance controller — is compatible with such an extension at the output-layer level. The data-collection, training-runtime, and evaluation components needed for this direction (the bilateral teleoperation contract abstraction of Chapter 13, the configuration hierarchy of Chapter 9, the `vision-core` transport and tool engine of Chapter 11, and the three demonstrations of Chapter 13) are present in the infrastructure described in this dissertation. Open questions include which stiffness parametrization generalizes across embodiments, how contact behaviors

transfer between robots with matching abstract kinematics but different dynamics, and how hard safety constraints compose with the implicit preferences of a learned policy.

14.2.3 Application to Additional Domains

The architectural contributions of this dissertation — parallel non-blocking execution, modular decoupling, multi-user MR coordination, multi-agent AI supervision, and interaction control — were developed in a rehabilitation robotics context. Chapter 8 applied the same principles to autonomous materials synthesis, a non-rehabilitation application. Empirical validation in clinical rehabilitation and autonomous synthesis domains is deferred to IRB-approved protocols and domain-specific collaborations.

14.2.4 Clinical Validation of Multi-Site Operation

Chapter 9 deployed heterogeneous robot platforms across two institutions through the declarative configuration hierarchy, and Chapter 13 exercised multi-user operation through a mesh VPN overlay in the laboratory. What remains is clinical validation: characterizing communication latency under real clinical network conditions, evaluating therapist situational awareness when supervising remote sessions through avatar visualization, and comparing rehabilitation outcomes between co-located and remote-guided sessions under Institutional Review Board (IRB)-approved protocols. The infrastructure described in this dissertation supports all three studies without architectural modification; IRB-approved protocols and clinical-site agreements are the remaining prerequisites.

14.2.5 Rehabilitation-Specific AI Validation

The multi-agent AI platform of Chapter 10 was validated in a nuclear waste decision-support domain (Chang *et al.*, 2025). Domain-specific validation for rehabilitation is still open: VLM-based posture assessment against motion-capture ground truth, retrieval accuracy on rehabilitation-protocol corpora, and supervisory prioritization across concurrent multi-patient sessions. The approval mechanism of Chapter 10 and the voice interface of Chapter 11 provide the interaction substrate for these studies; the domain content (protocols, corpora, metrics) remains to be developed with clinical collaborators.

14.3 Closing Remarks

Chapter 1 asked: *what software architecture patterns allow pHRI and MR rehabilitation systems to achieve deterministic real-time control, multi-modal sensor integration, and modular team development on commodity hardware?* The systems documented in the preceding chapters answer that question by instantiating two objectives—parallel architecture for real-time performance and modular architecture for scalable development—across rehabilitation, materials synthesis, and nuclear-cleanup deployments. Both objectives draw on prior literature; their consistent co-application is what this dissertation documents, across the robot fleet of Table 9.2 at two institutions, control rates from 250 Hz to 1 kHz, single-user and multi-user mixed reality, and single-robot and dexterous-manipulator embodiments, on commodity hardware and without a real-time operating system.

Evidence for this transferability comes from three sources: the cross-platform validation of Chapter 9, the cross-domain reuse of Chapter 10, and the multi-system composition of Chapter 13.

3D Systems / Geomagic, "PHANToM Omni (geomagic touch) haptic device specification", `https://www.3dsystems.com/haptics-devices/touch`, 6-DoF impedance-type haptic input device; workspace 160×120×70 mm; max continuous force 3.3 N; 1 kHz update rate; IEEE 1394 FireWire interface (2023).

Ahn, M., A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances", in "Proceedings of the 6th Conference on Robot Learning (CoRL)", (PMLR, 2022).

Al-Issa, H., H. Regenbrecht and L. Hale, "Augmented reality applications in rehabilitation to improve physical outcomes", Physical Therapy Reviews **17**, 1, 16–28, URL `https://doi.org/10.1179/1743288X11Y.0000000051` (2012).

Albu-Schäffer, A., C. Ott and G. Hirzinger, "A unified passivity-based control framework for position, torque and impedance control of flexible joint robots", The International Journal of Robotics Research **26**, 1, 23–39 (2007).

Amershi, S., D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen *et al.*, "Guidelines for human-AI interaction", in "Proceedings of the CHI Conference on Human Factors in Computing Systems", (2019).

Anderson, R. J. and M. W. Spong, "Bilateral control of teleoperators with time delay", IEEE Transactions on Automatic Control **34**, 5, 494–501 (1989).

Assis, G. A. d., A. G. D. Corrêa, M. B. R. Martins, W. G. Pedrozo and R. d. D. Lopes, "An augmented reality system for upper-limb post-stroke motor rehabilitation: a feasibility study", Disability and Rehabilitation: Assistive Technology **11**, 6, 521–528, URL `https://www.tandfonline.com/doi/abs/10.3109/17483107.2014.979330` (2016).

Atkins, J., D. Chang and H. Lee, "Design of a wearable shoulder exoskeleton robot with dual-purpose gravity compensation and a compliant misalignment compensation mechanism", Wearable Technologies **5**, e25 (2024).

Balasubramanian, S., A. Melendez-Calderon, A. Roby-Brami and E. Burdet, "On the analysis of movement smoothness", Journal of NeuroEngineering and Rehabilitation **12**, 1, 112 (2015).

Black, K. *et al.*, "π_0: A vision-language-action flow model for general robot control", Physical Intelligence Technical Report (2024).

Bohren, J. and S. Cousins, "SMACH—ROS state machine library", in "IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation", (2011).

Boyd, S. and L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, UK, 2004).

Brennan, D. M., L. Tindall, D. Theodoros, J. Brown, M. Campbell, D. Christiana, D. Smith, J. Cason and A. Lee, "A blueprint for telerehabilitation guidelines", International Journal of Telerehabilitation **2**, 2, 31–34 (2010).

Brochu, E., V. M. Cora and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions", arXiv preprint arXiv:1012.2599 (2010).

Brohan, A., N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control", arXiv preprint arXiv:2307.15818 (2023).

Bruyninckx, H., "Open robot control software: the OROCOS project", in "Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)", pp. 2523–2528 (2001).

Burns, A. and R. I. Davis, "A survey of research into mixed criticality systems", ACM Computing Surveys **50**, 6, 82:1–82:37 (2017).

Calinon, S., "A tutorial on task-parameterized movement learning and retrieval", Intelligent Service Robotics **9**, 1, 1–29 (2016).

Carmigniani, J., B. Furht, M. Anisetti, P. Ceravolo, E. Damiani and M. Ivkovic, "Augmented reality technologies, systems and applications", Multimedia Tools and Applications **51**, 1, 341–377, URL `https://doi.org/10.1007/s11042-010-0660-6` (2011).

Casini, D., T. Blaß, I. Lütkebohle and B. B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", in "31st Euromicro Conference on Real-Time Systems (ECRTS 2019)", vol. 133 of *LIPIcs*, pp. 6:1–6:23 (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019).

Cervin, A., D. Henriksson, B. Lincoln, J. Eker and K.-E. Årzén, "How does control timing affect performance? analysis and simulation of timing using Jitterbug and TrueTime", IEEE Control Systems Magazine **23**, 3, 16–30 (2003).

Chang, D., J. Hunt, J. Atkins and H. Lee, "Validation of 4b-spm robot for shoulder rehabilitation", in "IEEE International Conference on Robotics and Automation (ICRA)", (2021).

Chang, D., S. Kim and Y. S. Park, "AI-supported platform for system monitoring and decision-making in nuclear waste management with large language models", in "WM2025 Conference", (Phoenix, AZ, 2025), superior Paper Award. Supported by ASU GPSA JumpStart Grant Program.

Chi, C., S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion", in "Robotics: Science and Systems (RSS)", (2023).

Chinchali, S., A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti and M. Pavone, "Network offloading policies for cloud robotics: a learning-based approach", Autonomous Robots **45**, 7, 997–1012 (2021).

Colgate, J. E. and J. M. Brown, "Factors affecting the Z-width of a haptic display", in "Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)", pp. 3205–3210 (1994).

Colgate, J. E. and N. Hogan, "Robust control of dynamically interacting systems", International Journal of Control **48**, 1, 65–88 (1988).

Colgate, J. E. and G. G. Schenkel, "Passivity of a class of sampled-data systems: Application to haptic interfaces", Journal of Robotic Systems **14**, 1, 37–47 (1997).

Colledanchise, M. and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*, Chapman & Hall/CRC Artificial Intelligence and Robotics Series (CRC Press, 2018).

Colledanchise, M. and P. Ögren, "Behavior trees in robot control systems", Annual Review of Control, Robotics, and Autonomous Systems **5**, 81–107 (2022).

Corsaro, A., "Zenoh: Zero overhead pub/sub, store/query and compute", arXiv preprint arXiv:2208.09793 (2022).

Costa, G. M., M. R. Petry, J. G. Martins and A. P. G. M. Moreira, "Assessment of multiple fiducial marker trackers on HoloLens 2", IEEE Access (2024).

CrewAI, "CrewAI: Framework for orchestrating role-playing autonomous AI agents", `https://github.com/crewAIInc/crewAI` (2024).

Crick, C., G. Jay, S. Osentoski, B. Pitzer and O. C. Jenkins, "Rosbridge: Providing web-based access to robot middleware", in "IEEE International Conference on Robotics and Automation (ICRA)", (2017).

Dragan, A. D., K. C. T. Lee and S. S. Srinivasa, "Legibility and predictability of robot motion", in "ACM/IEEE International Conference on Human-Robot Interaction (HRI)", (2013).

Drake Development Team, "Drake: Model-based design and verification for robotics", `https://drake.mit.edu` (2024).

Fielding, R. T., *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. thesis, University of California, Irvine (2000).

Franka Robotics GmbH, "Franka Research 3 (FR3) robot arm specification", `https://franka.de/research`, 7-DoF research robot, payload 3 kg, reach 855 mm, pose repeatability ±0.1 mm (2024).

Garcia-Hernandez, N., S. Buccelli, A. De Angelis, E. Taglione, M. Laffranchi and L. deMichieli, "Assessment of Gamified Mixed Reality Environments for Upper Limb Robotic Rehabilitation: Pilot Study on Healthy Adults", URL `https://www.researchsquare.com/article/rs-4535965/v1` (2024).

Garcia Hernandez, N. V., S. Buccelli, M. Laffranchi and L. de Michieli, "Mixed Reality-based Exergames for Upper Limb Robotic Rehabilitation", in "Companion of the 2023 ACM/IEEE International Conference on Human-Robot Interaction", HRI '23, pp. 447–451 (Association for Computing Machinery, New York, NY, USA, 2023), URL https://doi.org/10.1145/3568294.3580124.

Garrido-Jurado, S., R. Muñoz-Salinas, F. J. Madrid-Cuevas and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", Pattern Recognition **47**, 6, 2280–2292 (2014).

Gassert, R. and V. Dietz, "Rehabilitation Robots for the Treatment of Sensorimotor Deficits: A Neurophysiological Perspective", Journal of Neuroengineering and Rehabilitation (2018).

Giang, C., E. Pirondini, N. Kinany, C. Pierella, A. Panarese, M. Coscia, J. Miehlbradt, C. Magnin, P. Nicolo, A. Guggisberg and others, "Motor improvement estimation and task adaptation for personalized robot-aided therapy: a feasibility study", Biomedical engineering online **19**, 1–25 (2020).

GPy Authors, "GPy: A Gaussian process framework in Python", https://sheffieldml.github.io/GPy/ (2014).

Hannaford, B. and J.-H. Ryu, "Time-domain passivity control of haptic interfaces", IEEE Transactions on Robotics and Automation **18**, 1, 1–10 (2002).

Hashtrudi-Zaad, K. and S. E. Salcudean, "Analysis of control architectures for teleoperation systems with impedance/admittance master and slave manipulators", The International Journal of Robotics Research **20**, 6, 419–445 (2001).

Higgins, P., R. Barron, D. Engel and C. Matuszek, "A Comparative Analysis of VR-Based and Real-World Human-Robot Collaboration for Small-Scale Joining", in "2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)", pp. 845–846 (2023), URL https://ieeexplore.ieee.org/document/10108789.

Higham, N. J., *Accuracy and Stability of Numerical Algorithms* (SIAM, Philadelphia, PA, 2002), 2nd edn.

Hintjens, P., *ZeroMQ: Messaging for Many Applications* (O'Reilly Media, 2013).

Hogan, N., "Impedance control: An approach to manipulation: Part i—theory", Journal of Dynamic Systems, Measurement, and Control **107**, 1, 1–7 (1985).

Hohpe, G. and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions* (Addison-Wesley, 2003).

Howard, M. C. and M. M. Davis, "A meta-analysis and systematic literature review of mixed reality rehabilitation programs: Investigating design characteristics of augmented reality and augmented virtuality", Computers in Human Behavior **130**, 107197, URL https://linkinghub.elsevier.com/retrieve/pii/S074756322200019X (2022).

Hung, L., J. A. Yager, D. Monteverde, D. Baiocchi, H.-K. Kwon and S. Sun, "Autonomous laboratories for accelerated materials discovery: A community survey and practical insights", Digital Discovery **3**, 7, 1273–1279 (2024).

Hunt, J. and H. Lee, "A new parallel actuated architecture for exoskeleton applications involving multiple degree-of-freedom biological joints", ASME Journal of Mechanisms and Robotics **10**, 5, 051017 (2018).

Hunt, J. and H. Lee, "Development of a low inertia parallel actuated shoulder exoskeleton robot for the characterization of neuromuscular property during static posture and dynamic movement", in "IEEE International Conference on Robotics and Automation (ICRA)", pp. 556–562 (2019).

Hwang, S., D. Chang, A. Saxena, E. Oleen, S. L. Paing, J. Atkins and H. Lee, "Characterization of human shoulder joint stiffness across 3d arm postures and its sex differences", IEEE Transactions on Biomedical Engineering **71**, 10, 2833–2841, co-first authors: Hwang and Chang (2024a).

Hwang, S., D. Chang, A. Saxena, E. Oleen, S. L. Paing, J. Atkins and H. Lee, "Characterization of human shoulder joint stiffness across various arm postures and its sex differences", in "Proceedings of the 48th Annual Meeting of the American Society of Biomechanics (ASB)", (Madison, Wisconsin, USA, 2024b), poster P1-281.

Hönig, W., C. Milanes, L. Scaria, T. Phan, M. Bolas and N. Ayanian, "Mixed reality for robotics", in "2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 5382–5387 (2015), URL `https://ieeexplore.ieee.org/document/7354138`.

Jennewein, D. M. *et al.*, "The sol supercomputer at Arizona State University", in "Practice and Experience in Advanced Research Computing (PEARC '23)", pp. 296–301 (ACM, New York, NY, 2023).

Just, F., Ö. Özen, P. Bösch, H. Bobrovsky, V. Klamroth-Marganska, R. Riener and G. Rauter, "Exoskeleton transparency: Feed-forward compensation vs. disturbance observer", at-Automatisierungstechnik **66**, 12, 1014–1026 (2018).

Kaszuba, S., F. Leotta and D. Nardi, "A Preliminary Study on Virtual Reality Tools in Human-Robot Interaction", in "Augmented Reality, Virtual Reality, and Computer Graphics", pp. 81–90 (Springer International Publishing, Cham, 2021).

Kehoe, B., S. Patil, P. Abbeel and K. Goldberg, "A survey of research on cloud robotics and automation", IEEE Transactions on Automation Science and Engineering **12**, 2, 398–409 (2015).

Kent, W., "A simple guide to five normal forms in relational database theory", Communications of the ACM **26**, 2, 120–125 (1983).

Khalil, H. K., *Nonlinear Systems* (Prentice Hall, Upper Saddle River, NJ, 2002), 3rd edn.

Khatib, O., "A unified approach for motion and force control of robot manipulators: The operational space formulation", IEEE Journal on Robotics and Automation **3**, 1, 43–53 (1987).

Kim, J., M. Sung, Y. Choi, J. Park and W. K. Chung, "Impedance control design framework using commutative map between SE(3) and se(3)", IEEE Transactions on Robotics (2025).

Kim, S., D. Chang, B. Archambault and Y. S. Park, "Eyes, ears, and evidence: A multi-agent AI system for nuclear waste decision support", in "WM2026 Conference", (Phoenix, AZ, 2026a), paper 26521.

Kim, S., D. Chang and J. Wang, "Persona alchemy: Designing, evaluating, and implementing psychologically-grounded LLM agents for diverse stakeholder representation", in "ICLR 2026 Workshop on Algorithmic Fairness Across Alignment Procedures and Agentic Systems (AFAA)", (2026b), URL `https://openreview.net/forum?id=ciWvxWWUQ6`, accepted; to appear.

Kim, S.-J. and H. I. Krebs, "Effects of implicit visual feedback distortion on human gait", Experimental Brain Research **218**, 3, 495–502 (2012).

Kim, S.-J., O. Save, E. Tanner, A. Marquez and H. Lee, "Gait symmetric adaptation and aftereffect through concurrent split-belt treadmill walking and explicit visual feedback distortion", IEEE Transactions on Biomedical Engineering **72**, 3, 1170–1177, early Access; final issue Mar 2025 (2024).

Kinova Robotics, "Gen3 7-Dof spherical wrist robotic arm user guide and specification", `https://www.kinovarobotics.com/product/gen3-robots`, 7-DoF, payload 4 kg, reach 902 mm, pose repeatability ±0.1 mm; software-imposed joint velocity limit 57°/s (2023).

Koenig, N. and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2004).

Krevelen, D. W. F. v. and R. Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations", International Journal of Virtual Reality **9**, 2, 1–20, URL `https://ijvr.eu/article/view/2767` (2010).

KUKA Deutschland GmbH, "Sensitive lightweight articulated 7-axis robots: LBR iiwa 7 R800, LBR iiwa 14 R820 specification", `https://www.oir.caltech.edu/twiki_oir/pub/Palomar/ZTF/KUKARoboticArmMaterial/Spez_LBR_iiwa_en.pdf`, 7-axis collaborative robot; payload 7 / 14 kg; reach 800 / 820 mm; pose repeatability ±0.1 mm (2019).

KUKA Deutschland GmbH, "LBR Med 7 R800 and LBR Med 14 R820: Medical-certified lightweight robots specification", `https://www.kuka.com/en-de/industries/health-care/lbr-med`, medical-certified variants of LBR iiwa 7/14; ISO 13485 and IEC 60601-1 certifications; pose repeatability ±0.15 mm (2020).

KUKA Deutschland GmbH, "KR AGILUS robot series (kr 6, kr 10, kr 20) specification", https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/kr-agilus, 6-DoF compact industrial robot family, KR10 R1100-2 payload 10 kg, reach 1101 mm, pose repeatability ±0.04 mm (ISO 9283), KR C5 micro controller (2024a).

KUKA Deutschland GmbH, "KR QUANTEC heavy-payload robot series specification", https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/kr-quantec, 6-DoF heavy-payload industrial robot family, KR210 R2700-2 payload 210 kg, reach 2700 mm, pose repeatability ±0.06 mm (ISO 9283), weight 1064 kg (2024b).

Lawrence, D. A., "Stability and transparency in bilateral teleoperation", IEEE Transactions on Robotics and Automation **9**, 5, 624–637 (1993).

Lee, J.-K., H. Choi, Y. S. Park and J.-H. Ryu, "Teaching robots to handle nuclear waste: A teleoperation-based learning approach", in "Waste Management Symposia (WM2025)", (Phoenix, AZ, 2025).

Lee, J.-K., D. Kim, S.-S. Park, J. Lee and J.-H. Ryu, "Development of a collaborative robotic arm-based bimanual haptic display", in "AsiaHaptics", (2024a).

Lee, J.-K. and Y. S. Park, "Contact tooling manipulation control for robotic repair platform", in "Waste Management Symposia (WM2024)", (Phoenix, AZ, 2024a).

Lee, J.-K. and Y. S. Park, "Dual-arm telerobotic platform for robotic hotbox operations for nuclear waste disposition in EM sites", in "Waste Management Symposia (WM2024)", (Phoenix, AZ, 2024b).

Lee, S., J. I. Kim, Y. Baek, D. Chang, J. Lee, Y. S. Park, D. Lee and Y.-L. Park, "Fiber-optic force sensing of modular robotic skin for remote and autonomous robot control", IEEE Transactions on Robotics **40**, 2373–2385 (2024b).

Lee, S., A. Shakir, D. Koenig and J. Lipp, "Open source strikes bread—new fluffy embedding model", https://www.mixedbread.ai/blog/mxbai-embed-large-v1 (2024c).

Liang, J., W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence and A. Zeng, "Code as policies: Language model programs for embodied control", in "IEEE International Conference on Robotics and Automation (ICRA)", (2023).

Ljung, L., *System Identification: Theory for the User* (Prentice Hall, 1999), 2nd edn.

Lynch, K. M. and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control* (Cambridge University Press, Cambridge, UK, 2017).

Macenski, S., T. Foote, B. Gerkey, C. Lalancette and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild", Science Robotics **7**, 66, eabm6074 (2022).

Merkel, D., "Docker: Lightweight Linux containers for consistent development and deployment", Linux Journal **2014**, 239, 2 (2014).

Meta, "Llama 3.2", `https://www.llama.com/`, accessed: Sep. 30, 2024 (2024).

Metta, G., P. Fitzpatrick and L. Natale, "YARP: Yet another robot platform", International Journal of Advanced Robotic Systems **3**, 1, 43–48 (2006).

Microsoft, "Mixed Reality Toolkit (MRTK) for Unity", `https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/`, accessed: 2025-01-15 (2024).

Mirror Networking Contributors, "Mirror networking: High-level API for unity multiplayer networking", `https://github.com/MirrorNetworking/Mirror`, open-source authoritative-server networking framework for Unity; supports SyncVar state replication and KCP reliable UDP transport for LAN-local multi-user sessions (2024).

Morris, K., *Infrastructure as Code: Managing Servers in the Cloud* (O'Reilly Media, 2016).

Murnane, M., P. Higgins, M. Saraf, F. Ferraro, C. Matuszek and D. Engel, "A Simulator for Human-Robot Interaction in Virtual Reality", in "2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)", pp. 470–471 (2021), URL `https://ieeexplore.ieee.org/document/9419217`.

Mussa-Ivaldi, F. A., N. Hogan and E. Bizzi, "Neural, mechanical, and geometric factors subserving arm posture in humans", Journal of Neuroscience **5**, 10, 2732–2743 (1985).

Niemeyer, G. and J.-J. E. Slotine, "Stable adaptive teleoperation", IEEE Journal of Oceanic Engineering **16**, 1, 152–162 (1991).

NVIDIA, "Isaac Sim: Robotics simulation and synthetic data generation", `https://developer.nvidia.com/isaac-sim` (2024).

Ocampo, R. and M. Tavakoli, "Visual-Haptic Colocation in Robotic Rehabilitation Exercises Using a 2D Augmented-Reality Display", in "2019 International Symposium on Medical Robotics (ISMR)", pp. 1–7 (2019).

Ollama, "Ollama: Run large language models locally", `https://github.com/ollama/ollama`, accessed: Jul. 7, 2024 (2024).

Ott, C., A. Albu-Schäffer, A. Kugi and G. Hirzinger, "On the passivity-based impedance control of flexible joint robots", IEEE Transactions on Robotics **24**, 2, 416–429 (2008).

Park, S.-S., D. Chang and Y. S. Park, "Heavy manipulator control", in "WM2026 Conference", (Phoenix, AZ, 2026a), wM submission ID 26525.

Park, S.-S., D. Chang and Y. S. Park, "Non-spherical wrist robot control", in "WM2026 Conference", (Phoenix, AZ, 2026b), wM submission ID 26524.

Parker, L. E., D. Rus and G. S. Sukhatme, "Multiple mobile robot systems", in "Springer Handbook of Robotics", edited by B. Siciliano and O. Khatib, pp. 1335–1384 (Springer, Berlin, Heidelberg, 2016), 2nd edn.

Patterson, K. K., W. H. Gage, D. Brooks, S. E. Black and W. E. McIlroy, "Evaluation of gait symmetry after stroke: A comparison of current methods and recommendations for standardization", Gait & Posture **31**, 2, 241–246 (2010).

Perreault, E. J., R. F. Kirsch and P. E. Crago, "Voluntary control of static endpoint stiffness during force regulation tasks", Journal of Neurophysiology **85**, 6, 2879–2890 (2001).

Pezzera, M., E. Chitti and N. A. Borghese, "MIRARTS: A mixed reality application to support postural rehabilitation", in "2020 IEEE 8th International Conference on Serious Games and Applications for Health (SeGAH)", pp. 1–7 (2020), URL `https://ieeexplore.ieee.org/document/9201694`.

Phan, H. L., T. H. Le, J. M. Lim, C. H. Hwang and K.-i. Koo, "Effectiveness of Augmented Reality in Stroke Rehabilitation: A Meta-Analysis", Applied Sciences **12**, 4, 1848, URL `https://www.mdpi.com/2076-3417/12/4/1848` (2022).

Pinheiro, J. C. and D. M. Bates, "Unconstrained parametrizations for variance-covariance matrices", Statistics and Computing **6**, 3, 289–296 (1996).

Quesada, R. C. and Y. Demiris, "Proactive Robot Assistance: Affordance-Aware Augmented Reality User Interfaces", IEEE Robotics & Automation Magazine **29**, 1, 22–34, URL `https://ieeexplore.ieee.org/document/9680771/` (2022).

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, "Ros: An open-source robot operating system", in "ICRA Workshop on Open Source Software", (2009).

Rainbow Robotics, "RB-Y1 wheeled dual-arm humanoid robot specification", `https://www.rainbow-robotics.com/`, 22-DoF wheeled humanoid: 2 wheels + 6 torso + 7 left arm + 7 right arm; per-arm payload 3 kg, per-arm reach 730 mm; ROS 2 Humble + MoveIt2; `rby1-sdk` Python interface (2024).

Reisman, D. S., H. J. Block and A. J. Bastian, "Interlimb coordination during locomotion: What can be adapted and stored?", Journal of Neurophysiology **94**, 4, 2403–2415 (2005).

Reisman, D. S., R. Wityk, K. Silver and A. J. Bastian, "Locomotor adaptation on a split-belt treadmill can improve walking symmetry post-stroke", Brain **130**, 7, 1861–1872 (2007).

Reisman, D. S., R. Wityk, K. Silver and A. J. Bastian, "Split-belt treadmill adaptation transfers to overground walking in persons poststroke", Neurorehabilitation and Neural Repair **23**, 7, 735–744 (2009).

Ren, A. Z. *et al.*, "Diffusion policy policy optimization", in "International Conference on Learning Representations (ICLR)", (2025), arXiv:2409.00588.

ROS Community, “Rep 103: Standard units of measure and coordinate conventions”, https://www.ros.org/reps/rep-0103.html, accessed: 2024-12-28 (2010).

Salvucci, D. D. and J. H. Goldberg, “Identifying fixations and saccades in eye-tracking protocols”, in “Proceedings of the 2000 Symposium on Eye Tracking Research & Applications (ETRA)”, pp. 71–78 (2000).

Save, O. M., D. Chang, S. P. Joglekar, V. Choudhary and H. Lee, “Feasibility of gait symmetric learning through augmented reality-based visual distortion”, IEEE Transactions on Biomedical Engineering Co-equal first authorship (O. M. Save and D. Chang); submitted, under review at IEEE TBME (2026).

Seo, J., N. P. S. Prakash, A. Rose, J. Choi and R. Horowitz, “Geometric impedance control on SE(3) for robotic manipulators”, IFAC-PapersOnLine **56**, 2, 276–283 (2023).

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams and N. de Freitas, “Taking the human out of the loop: A review of Bayesian optimization”, Proceedings of the IEEE **104**, 1, 148–175 (2016).

Sheridan, T. B., “Space teleoperation through time delay: review and prognosis”, IEEE Transactions on Robotics and Automation **9**, 5, 592–606 (1993).

Szczurek, K. A., R. Marin, E. Matheson, J. Rodriguez-Nogueira and M. D. Castro, “Multimodal Multi-User Mixed Reality Human–Robot Interface for Remote Operations in Hazardous Environments”, IEEE Access (2023).

Tsumugiwa, T., R. Yokogawa and K. Hara, “Variable impedance control based on estimation of human arm stiffness for human-robot cooperative calligraphic task”, in “Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)”, pp. 644–650 (IEEE, Washington, DC, USA, 2002).

UFACTORY, “Lite 6: 6-dof lightweight cobot specification”, https://www.ufactory.cc/lite-6-collaborative-robot/, 6-DoF, payload 0.6 kg, reach 440 mm, repeatability ±0.5 mm (2024a).

UFACTORY, “xArm 5 lite: 5-dof industrial cobot specification”, https://www.ufactory.cc/xarm-collaborative-robot/, 5-DoF, payload 3 kg, reach 700 mm, repeatability ±0.1 mm (2024b).

Unity Technologies, “Unity real-time development platform”, https://unity.com/ (2024).

Universal Robots and ROS-Industrial Consortium, “Universal_Robots_ROS_Driver: ROS driver for universal robots manipulators”, https://github.com/UniversalRobots/Universal_Robots_ROS_Driver, open-source ROS/ROS2 driver supporting position, velocity, and torque interfaces (2024).

Universal Robots A/S, "Universal robots e-Series: UR3e, UR5e, UR10e, UR16e robot arm specifications (collective data sheet)", https://www.universal-robots.com/media/1827367/05_2023_collective_data-sheet.pdf, joint range ±360° on all six axes; pose repeatability ±0.03 mm (ISO 9283); base/shoulder max velocity 120°/s, others 180°/s (2023).

Vestal, S., "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance", in "28th IEEE International Real-Time Systems Symposium (RTSS 2007)", pp. 239–243 (IEEE Computer Society, 2007).

Wang, F., Y. Wen, J. Bi, H. Li and J. Sun, "A portable SSVEP-BCI system for rehabilitation exoskeleton in augmented reality environment", Biomedical Signal Processing and Control **83**, 104664, URL https://www.sciencedirect.com/science/article/pii/S1746809423000976 (2023).

Wenk, N., K. A. Buetler, J. Penalver-Andres, R. M. Müri and L. Marchal-Crespo, "Naturalistic Visualization of Reaching Movements Using Head-Mounted Displays Improves Movement Quality Compared to Conventional Computer Screens and Proves High Usability", Journal of NeuroEngineering and Rehabilitation (2022).

Williams, T., D. Szafr, T. Chakraborti and H. B. Amor, "Report on the First International Workshop on Virtual, Augmented, and Mixed Reality for Human-Robot Interaction", AI Magazine **39**, 4, 64–66, URL https://onlinelibrary.wiley.com/doi/10.1609/aimag.v39i4.2822 (2018).

Wolpert, D. M., R. C. Miall and M. Kawato, "Internal models in the cerebellum", Trends in Cognitive Sciences **2**, 9, 338–347 (1998).

Wu, Q., G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu *et al.*, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation", arXiv preprint arXiv:2308.08155 (2023).

Yan, Z., N. Jouandeau and A. A. Cherif, "A survey and analysis of multi-robot coordination", International Journal of Advanced Robotic Systems **10**, 12, 399 (2013).

Yi, J., W. J. Chung, J. Lee, H. Muzammal, J. Park, Y. S. Park and Y.-L. Park, "A multifingered robotic hand with fiber-optic force and tactile sensing for remote manipulation", IEEE Transactions on Robotics **42**, 509–521 (2026).

Yokokohji, Y. and T. Yoshikawa, "Bilateral control of master-slave manipulators for ideal kinesthetic coupling—formulation and experiment", IEEE Transactions on Robotics and Automation **10**, 5, 605–620 (1994).

Yoshikawa, T., "Manipulability of robotic mechanisms", The International Journal of Robotics Research **4**, 2, 3–9 (1985).

Zahedi, F., D. Chang and H. Lee, "User-adaptive variable damping control using bayesian optimization to enhance physical human-robot interaction", IEEE Robotics and Automation Letters **7**, 2, 2724–2731 (2022).

APPENDIX A

ROBOT SPECIFICATIONS AND KINEMATIC PARAMETERS

This appendix lists the hardware specifications and kinematic parameters for the robotic platforms integrated into the multi-HRI framework. Robots are presented in monotone DoF order: single-arm manipulators (5–7 DoF) first, then multi-arm composite systems (14 DoF), the humanoid platform (22 DoF), and finally the haptic input device. All robots in the framework adhere to the ROS Enhancement Proposal 103 (REP-103) coordinate convention (ROS Community, 2010): X forward, Y left, Z up, right-handed. Kinematic chains use the Product of Exponentials (POE) formulation (Lynch and Park, 2017); the analytical POE solver implementation is documented in Appendix B.

A.1 Multi-HRI System Robot Inventory

Table A.1 presents the complete inventory of robotic systems integrated into the multi-HRI framework, organized by manufacturer and assigned QR codes for spatial tracking and identification. Single-arm robots occupy QR identifiers Q1–Q13, the humanoid platform sits at Q20, multi-arm composite systems at Q111–Q113, and the haptic input device is registered without a QR (it is connected via FireWire/USB rather than spatially anchored).

Table A.1: Complete robot inventory in the multi-HRI framework. Specifications cross-verified against manufacturer datasheets and the configuration files in `multi-hra-robots/robots/overrides/`.

QR ID	Manufacturer	Model	DoF	Payload [kg]	Reach [mm]
UFACTORY (China) – Lightweight Arms					
Q1	UFACTORY	xArm5	5	3	700
Q2	UFACTORY	Lite6	6	0.6	440
Universal Robots (Denmark) – 6DOF Cobots					
Q7	Universal Robots	UR3e	6	3	500
Q8	Universal Robots	UR5e	6	5	850
Q9	Universal Robots	UR16e	6	16	900
KUKA Industrial RSI (Germany) – 6DOF Heavy-Duty					
Q12	KUKA	KR10 R1100-2 (Agilus)	6	10	1101
Q13[†]	KUKA	KR210 R2700-2 (Quantec)	6	210	2700
KUKA iiwa (Germany) – 7DOF Cobots					
Q3	KUKA	iiwa7	7	7	800
Q4	KUKA	iiwa14	7	14	820
Q5	KUKA	MED7	7	7	800
Q6	KUKA	MED14	7	14	820
Research & Specialty 7DOF Robots					
Q10	Franka Emika	FR3	7	3	855
Q11	Kinova	Gen3	7	4	902
Multi-Arm Composite Systems					
Q111	Kinova	Dual Gen3 (Gen3×2)	14	4/arm	902/arm
Q112	Franka	Dual FR3 (FR3×2)	14	3/arm	855/arm
Q113	Kinova	Gantry + Dual Gen3	14+4	4/arm	extended
Humanoid Platform					
Q20	Rainbow Robotics	RB-Y1	22[‡]	3/arm	730/arm
Haptic Input Device (no QR)					
—	3D Systems	Phantom Omni	6	—	160×120×70[§]

[†]Q13 reserved; URDF prefab integration in progress as of writing.
[‡]RB-Y1 22 DoF = 2 (mecanum wheels) + 6 (torso) + 7 (left arm) + 7 (right arm).
[§]Phantom workspace reported in mm (X×Y×Z), not reach.

The inventory spans 18 distinct robotic platforms from 7 manufacturers, with DoF configurations ranging from 5 (xArm5) to 22 (RB-Y1) and serial-arm payload capacities from 0.6 kg (Lite6) to 210 kg (KR210 heavy manipulator). The QR identifiers reflect the order in which platforms were registered in the spatial-anchor system (Section A.10 below); the dissertation chapters that exercise these platforms refer to the QR identifiers consistently. Section ordering in the remainder of this appendix is by DoF and morphology (single arm → multi-arm → humanoid → haptic), not by QR order.

A.2 UFACTORY Series

A.2.1 Lite6 Lightweight Robot (6-DoF)

Table A.2: Lite6 basic specifications (UFACTORY, 2024a).

Parameter	**Value**
Degrees of Freedom	6
Payload	0.6 kg
Reach	440 mm
Repeatability	±0.5 mm
Weight	7.2 kg
Mounting	Floor, table

Table A.3: Lite6 joint limits and home position (UFACTORY, 2024a).

Joint	**Range** [°]	**Max Velocity** [°/s]	**Home Position** [°]
joint1	±360	180	0
joint2	[−118, 120]	180	35
joint3	[−225, 11]	180	60
joint4	±360	360	0
joint5	±360	360	25
joint6	±360	360	−20

A.2.2 xArm5 Robot Arm (5-DoF Under-Actuated)

The xArm5 is the only 5-DoF platform in the framework; it cannot satisfy the six-dimensional task-space requirement of the Lie-group impedance controller, and is restricted to position-only control modes (see Appendix C, Section C.4.5).

Table A.4: xArm5 basic specifications (UFACTORY, 2024b).

Parameter	Value
Degrees of Freedom	5
Payload	3 kg
Reach	700 mm
Repeatability	±0.1 mm
Weight	12 kg
Mounting	Floor, table

Table A.5: xArm5 joint limits and home position (UFACTORY, 2024b).

Joint	Range [°]	Max Velocity [°/s]	Home Position [°]
joint1	±360	180	0
joint2	[−118, 120]	180	0
joint3	[−225, 11]	180	0
joint4	±360	360	0
joint5	±360	360	0

A.3 Universal Robots Series

Universal Robots employs a standardized 6-DoF architecture across their collaborative robot (cobot) series, using consistent kinematic conventions and joint naming schemes. Subsections are ordered by payload class (UR3e → UR5e → UR16e).

A.3.1 UR3e Collaborative Robot

Table A.6: UR3e basic specifications (Universal Robots A/S, 2023).

Parameter	Value
Degrees of Freedom	6
Payload	3 kg
Reach	500 mm
Repeatability	±0.03 mm (ISO 9283)
Weight	11 kg
Mounting	Floor, ceiling, wall

Table A.7: UR3e joint limits and home position (Universal Robots A/S, 2023).

Joint	Range [°]	Max Velocity [°/s]	Home Position [°]
shoulder_pan_joint	±360	180	0
shoulder_lift_joint	±360	180	−90
elbow_joint	±360	180	90
wrist_1_joint	±360	180	−90
wrist_2_joint	±360	180	−90
wrist_3_joint	±360	180	0

A.3.2 UR5e Collaborative Robot

Table A.8: UR5e basic specifications (Universal Robots A/S, 2023).

Parameter	Value
Degrees of Freedom	6
Payload	5 kg
Reach	850 mm
Repeatability	±0.03 mm (ISO 9283)
Weight	20.6 kg
Mounting	Floor, ceiling, wall

Table A.9: UR5e joint limits and home position (Universal Robots A/S, 2023).

Joint	Range [°]	Max Velocity [°/s]	Home Position [°]
shoulder_pan_joint	±360	180	0
shoulder_lift_joint	±360	180	−90
elbow_joint	±360	180	90
wrist_1_joint	±360	180	−90
wrist_2_joint	±360	180	−90
wrist_3_joint	±360	180	0

A.3.3 UR16e Heavy-Duty Collaborative Robot

Table A.10: UR16e basic specifications (Universal Robots A/S, 2023).

Parameter	**Value**
Degrees of Freedom	6
Payload	16 kg
Reach	900 mm
Repeatability	±0.05 mm (ISO 9283)
Weight	33.1 kg
Mounting	Floor, ceiling, wall

Table A.11: UR16e joint limits and home position (Universal Robots A/S, 2023).

Joint	**Range** [°]	**Max Velocity** [°/s]	**Home Position** [°]
shoulder_pan_joint	±360	120	0
shoulder_lift_joint	±360	120	−90
elbow_joint	±360	180	90
wrist_1_joint	±360	180	−90
wrist_2_joint	±360	180	−90
wrist_3_joint	±360	180	0

A.4 KUKA Industrial Robots (RSI Driver)

The KUKA Agilus and Quantec series are 6-DoF industrial manipulators that use the KUKA Robot Sensor Interface (RSI) for real-time external control through the KR C5 micro / KR C5 controller running KUKA System Software (KSS) version 8.7 UL. The framework integrates these arms via the open-source `kroshu/kuka_drivers` ROS2 driver (`kuka_rsi_driver`) at a 4 ms RSI cycle time. The Agilus KR10 R1100-2 is used for compact-cell pattern-execution scenarios; the Quantec KR210 R2700-2 is the target heavy-manipulator platform for the heavy-manipulator follow-on listed as a future-work item in Chapter 14.

A.4.1 KR10 R1100-2 (Agilus, 10 kg payload)

Table A.12: KR10 R1100-2 basic specifications (KUKA Deutschland GmbH, 2024a).

Parameter	Value
Degrees of Freedom	6
Payload	10 kg
Reach	1101 mm
Repeatability	±0.04 mm (ISO 9283)
Weight	55 kg (approximate)
Mounting	Floor, ceiling, wall
Controller	KR C5 micro (KSS 8.7 UL)
External interface	KUKA RSI (`eki_rsi`, 4 ms cycle)

Table A.13: KR10 R1100-2 joint limits and velocity limits (from MoveIt configuration).

Joint	Range [°]	Max Velocity [rad/s]	Home Position [°]
joint_1	[−170, 170]	5.236	0
joint_2	[−190, 45]	3.927	−90
joint_3	[−120, 156]	5.760	90
joint_4	[−185, 185]	6.283	0
joint_5	[−120, 120]	6.283	0
joint_6	[−350, 350]	7.557	0

A.4.2 KR210 R2700-2 (Quantec, 210 kg payload)

Table A.14: KR210 R2700-2 basic specifications (KUKA Deutschland GmbH, 2024b).

Parameter	Value
Degrees of Freedom	6
Payload	210 kg
Reach	2700 mm
Repeatability	±0.06 mm (ISO 9283)
Weight	1064 kg
Mounting	Floor
Controller	KR C5 (KSS 8.7 UL)
External interface	KUKA RSI (`eki_rsi`, 4 ms cycle)

Table A.15: KR210 R2700-2 joint limits and velocity limits (from MoveIt configuration).

Joint	Range [°]	Max Velocity [rad/s]	Home Position [°]
joint_1	[−185, 185]	2.038	0
joint_2	[−140, −5]	1.945	−90
joint_3	[−120, 155]	1.955	90
joint_4	[−350, 350]	3.124	0
joint_5	[−122.5, 122.5]	3.002	0
joint_6	[−350, 350]	3.840	0

A.5 KUKA iiwa Series

The KUKA iiwa (intelligent industrial work assistant) series uses a 7-DoF kinematic chain. The redundant joint provides a nullspace degree of freedom for singularity avoidance. The industrial iiwa7 and iiwa14 are presented first; the medical-certified LBR Med variants share the same kinematic chain and are documented as a single combined subsection.

A.5.1 iiwa7 Collaborative Robot

Table A.16: iiwa7 basic specifications (KUKA Deutschland GmbH, 2019).

Parameter	Value
Degrees of Freedom	7
Payload	7 kg
Reach	800 mm
Repeatability	±0.1 mm
Weight	23.9 kg
Mounting	Floor, ceiling

Table A.17: iiwa7 joint limits and home position (KUKA Deutschland GmbH, 2019).

Joint	Range [°]	Max Velocity [°/s]	Home Position [°]
iiwa7_A1	±170	85	0
iiwa7_A2	±120	85	0
iiwa7_A3	±170	100	0
iiwa7_A4	±120	75	−90
iiwa7_A5	±170	130	0
iiwa7_A6	±120	135	90
iiwa7_A7	±175	135	0

A.5.2 iiwa14 Heavy-Duty Collaborative Robot

Table A.18: iiwa14 basic specifications (KUKA Deutschland GmbH, 2019).

Parameter	Value
Degrees of Freedom	7
Payload	14 kg
Reach	820 mm
Repeatability	±0.1 mm
Weight	29.5 kg
Mounting	Floor, ceiling

Table A.19: iiwa14 joint limits and home position (KUKA Deutschland GmbH, 2019).

Joint	Range [°]	Max Velocity [°/s]	Home Position [°]
iiwa14_A1	±170	85	0
iiwa14_A2	±120	85	0
iiwa14_A3	±170	100	0
iiwa14_A4	±120	75	−90
iiwa14_A5	±170	130	0
iiwa14_A6	±120	135	90
iiwa14_A7	±175	135	0

A.5.3 LBR Med 7/14 (Medical-Certified Variants)

The KUKA LBR Med 7 R800 and LBR Med 14 R820 are the medical-certified variants of the LBR iiwa 7 R800 and iiwa 14 R820 respectively, sharing the 7-axis

kinematic chain and joint limits of the corresponding industrial models while adding ISO 13485-compliant materials and surfaces, IEC 60601-1 electrical safety, and a sterilizable end-flange interface for medical applications (KUKA Deutschland GmbH, 2020). The Multi-HRI framework treats the LBR Med variants (QR IDs Q5 and Q6) as kinematic equivalents of the iiwa7 R800 (Q3) and iiwa14 R820 (Q4); the medical-grade mechanical and software certifications are out of scope for the present dissertation.

Table A.20: LBR Med 7/14 basic specifications (KUKA Deutschland GmbH, 2020). Kinematic chain and joint limits are identical to the iiwa7/iiwa14 tables above.

Parameter	**LBR Med 7 R800**	**LBR Med 14 R820**
Degrees of Freedom	7	7
Payload	7 kg	14 kg
Reach	800 mm	820 mm
Repeatability	$\pm$0.15 mm	$\pm$0.15 mm
Weight	25.5 kg	32.3 kg
Mounting	Floor, ceiling, wall	Floor, ceiling, wall
Certification	ISO 13485, IEC 60601-1	ISO 13485, IEC 60601-1

A.6.1 Franka Emika FR3 Research Robot

Table A.21: FR3 basic specifications (Franka Robotics GmbH, 2024).

Parameter	**Value**
Degrees of Freedom	7
Payload	3 kg
Reach	855 mm
Repeatability	±0.1 mm
Weight	18 kg
Mounting	Floor, table

Table A.22: FR3 joint limits and home position (Franka Robotics GmbH, 2024).

Joint	**Range** [°]	**Max Velocity** [°/s]	**Home Position** [°]
fr3_joint1	±166	150	0
fr3_joint2	±101	150	−45
fr3_joint3	±166	150	0
fr3_joint4	[−176, −4]	150	−135
fr3_joint5	±166	180	0
fr3_joint6	[−1, 215]	180	90
fr3_joint7	±166	180	45

A.6.2 Kinova Gen3 7DOF Collaborative Robot

Table A.23: Kinova Gen3 7DOF basic specifications (Kinova Robotics, 2023).

Parameter	**Value**
Degrees of Freedom	7
Payload	4 kg
Reach	902 mm
Repeatability	±0.1 mm
Weight	19 kg
Mounting	Floor, table, ceiling

Table A.24: Kinova Gen3 joint limits and home position (Kinova Robotics, 2023).

Joint	Range [°]	Max Velocity [°/s]	Home Position [°]
joint_1	±360	57	0
joint_2	±138	57	15
joint_3	±360	57	180
joint_4	±152	57	−130
joint_5	±360	57	0
joint_6	±128	57	55
joint_7	±360	57	90

A.7 Multi-Arm Composite Systems

The framework integrates two dual-arm composite systems used as leader and follower platforms for bilateral teleoperation. Both compose two of the 7-DoF research arms documented in Section A.6; the dual-arm *system* differs from the underlying single-arm in mounting frame, end-effector, and total DoF. The mount geometry, gantry kinematics (where present), and combined Jacobian are documented in Appendix C, Section "Dual-Arm Robot Configuration".

A.7.1 Franka Dual Arm (Q112, 14-DoF Leader)

The Franka Dual Arm system pairs two Franka FR3 research arms (Section A.6) on a fixed LeaderMount structure with ±45° X-axis tilt, used as the leader side of the bimanual bilateral teleoperation pair (follower: Kinova Dual Arm, Q111). Each arm retains its native 7-DoF kinematic chain and integrated F/T sensing.

Table A.25: Franka Dual Arm system specifications (Franka Robotics GmbH, 2024). Per-arm specs match the FR3 single-arm tables above; the dual-arm composite parameters are listed below.

Parameter	Value
Total Degrees of Freedom	14 (7×2)
Payload (per arm)	3 kg
Reach (per arm)	855 mm
End effector	Franka Hand (1-DoF parallel gripper) per arm
Mount type	Fixed LeaderMount, world-frame anchored
Left arm offset (XYZ)	$(0.049,\ +0.121,\ 0.872)$ m, RPY $(-\pi/4,\ 0,\ 0)$
Right arm offset (XYZ)	$(0.049,\ -0.121,\ 0.872)$ m, RPY $(+\pi/4,\ 0,\ 0)$
F/T sensing	ATI NetFT (RDT/UDP) per arm
Control rate	1 kHz (libfranka)
SDK	`franka_ros2`

A.7.2 Kinova Dual Arm (Q111, 14-DoF Follower; Q113 with Gantry)

The Kinova Dual Arm system pairs two Kinova Gen3 7DOF arms (Section A.6) on a triangular connection plate. Two QR variants exist: Q111 mounts the arms on a fixed plate (no gantry), and Q113 mounts the same dual-arm assembly on a 4-DoF Cartesian gantry that adds three prismatic axes plus one revolute axis for workspace extension (gantry joint specifications in Appendix C, Table C.6).

Table A.26: Kinova Dual Arm system specifications (Kinova Robotics, 2023). Per-arm specs match the Kinova Gen3 single-arm tables above.

Parameter	**Value**
Total Degrees of Freedom	14 (7×2); 18 with gantry (Q113)
Payload (per arm)	4 kg
Reach (per arm)	902 mm
End effector	Robotiq 2F-85 parallel gripper per arm
Mount type	Triangular connection plate (Q111 fixed, Q113 gantry-mounted)
Left arm offset (XYZ)	$(0.030,\ +0.140,\ 0.585)$ m, RPY $(-\pi/4,\ 0,\ 0)$
Right arm offset (XYZ)	$(0.030,\ -0.140,\ 0.585)$ m, RPY $(+\pi/4,\ 0,\ 0)$
Eye-in-hand cameras	Intel RealSense D415 per arm
SDK	`ros2_kortex`

A.8 Humanoid Platform

A.8.1 Rainbow Robotics RB-Y1 (Q20, 22-DoF Wheeled Humanoid)

The RB-Y1 is a wheel-based dual-arm humanoid platform integrated as the central hardware target for Exemplar E2 (humanoid extension; see Chapter 13, Section 13.6.2). Its 22 degrees of freedom are partitioned into four enabled kinematic sub-chains plus a (currently disabled) head: 2 mecanum-wheel joints providing omnidirectional locomotion, a 6-DoF torso, a 7-DoF left arm, and a 7-DoF right arm. The framework's POE solver was validated on each sub-chain and on the full `base` → `ee_left` 13-DoF chain in January 2026.

Table A.27: RB-Y1 basic specifications (Rainbow Robotics, 2024).

Parameter	Value
Total Degrees of Freedom	22 (2 wheels + 6 torso + 7 left arm + 7 right arm)
Payload (per arm)	3 kg
Reach (per arm)	730 mm
Locomotion	Omnidirectional (mecanum wheels)
Mounting	Floor (mobile)
Sensors	2D LiDAR + IMU + 3×RealSense (2 eye-in-hand D405 + 1 head D435)
Wrist F/T	Both wrists (left, right)
ROS distribution	ROS2 Humble + MoveIt2
SDK	`rby1-sdk` (Python)
Control rate	100 Hz (publish rate)

Table A.28: RB-Y1 sub-chain decomposition (POE-validated 2026-01-06).

Sub-chain	DoF	Base link	Tip link	Status
Mobile base (omni)	2	—	—	enabled
Torso	6	`base`	`link_torso_5`	POE ✓
Left arm	7	`link_torso_5`	`ee_left`	POE ✓
Right arm	7	`link_torso_5`	`ee_right`	POE ✓
Full chain (L)	13	`base`	`ee_left`	POE ✓

The MoveIt 2 configuration exposes four move groups (`left_arm`, `right_arm`, `dual_arm`, `torso`) that the unified motion interface (Appendix D.9) selects based on the requested task DoF. Joint limits are estimated values pending final tuning on the physical platform; the values used by the planner are listed in the override YAML (`multi-hra-robots/robots/overrides/rby1.yaml`).

A.9 Haptic Input Device

A.9.1 3D Systems Phantom Omni (6-DoF Haptic Leader)

The Phantom Omni (now sold as the Geomagic Touch by 3D Systems) is a 6-DoF impedance-type haptic input device used as the master side for bilateral teleoperation in the PHANToM–UR16e single-arm scenario (Exemplar E1 of Chapter 13). Unlike the serial-arm manipulators above, the Phantom is connected via FireWire/USB rather than registered through a QR-anchored base offset; it is documented here for completeness of the hardware inventory.

Table A.29: Phantom Omni basic specifications (3D Systems / Geomagic, 2023).

Parameter	Value
Degrees of Freedom	6 (3 position-sensed + 3 orientation-sensed; 3 force-feedback axes)
Workspace	160×120×70 mm (X×Y×Z)
Maximum continuous force	3.3 N
Position resolution	~ 0.023 mm
Update rate	1 kHz
Connection	IEEE 1394 FireWire
Joint scales	$[-1.0,\ +1.0,\ +1.0,\ -1.0,\ -1.0,\ -1.0]$

Table A.30: Phantom Omni joint limits

Joint	Range [°]
waist	± 56
shoulder	$[0,\ 100]$
elbow	$[-46,\ 72]$
yaw	± 360
pitch	± 360
roll	± 360

A.10 QR-Spatial-Anchor Mapping

The robot QR identifiers used throughout this appendix are defined in the SSOT mapping file `multi-hra-config/config/robot_qr_mapping.yaml`, which also stores a 6-DoF base-pose offset (ENU position + quaternion rotation) for each QR. The full mapping table and the offset format are documented in Appendix D, where they appear alongside the rest of the configuration hierarchy. The kinematic analysis (singularity classification, HRI home positions, 20 cm-cube workspace results, and dual-arm configurations) for the platforms above is reported in Appendix C.

APPENDIX B

ANALYTICAL KINEMATICS TOOLKIT

This appendix documents the POE-based kinematics toolkit used as the default solver in the unified motion interface (Appendix D.9). The toolkit is organized as a four-stage flow: (i) the POE formulation of forward kinematics, (ii) the URDF-to-POE conversion that supplies the screw axes and home configuration, (iii) the analytical Jacobian, and (iv) the iterative inverse-kinematics solver that consumes the Jacobian, with a separate gravity-compensation translation unit (NRIC) that the inverse-dynamics path uses on top of the kinematic chain. Two software artifacts—a REST API server and a workspace-analysis utility—wrap these computations for non-ROS2 clients. The toolkit ships as a single C++ kernel exposed to Python through a pybind11 binding module; the C++/Python coexistence is documented as part of the POE formulation section below.

B.1 Product of Exponentials Formulation

B.1.1 Mathematical Foundation

The POE formula represents the forward kinematics of an n-joint serial manipulator as a product of matrix exponentials:

$$T(\theta) = e^{[\mathcal{S}_1]\theta_1}\, e^{[\mathcal{S}_2]\theta_2}\, \cdots\, e^{[\mathcal{S}_n]\theta_n}\, M \tag{B.1}$$

where $T(\theta) \in SE(3)$ is the end-effector pose, $\mathcal{S}_i = (\omega_i, v_i) \in \mathbb{R}^6$ is the screw axis of joint i expressed in the space frame, θ_i is the joint angle, and $M \in SE(3)$ is the end-effector pose at the home configuration ($\theta = 0$).

The matrix exponential $e^{[\mathcal{S}]\theta}$ for a revolute joint with screw axis $\mathcal{S} = (\omega, v)$ is computed as:

$$e^{[\mathcal{S}]\theta} = \begin{bmatrix} e^{[\omega]\theta} & (I\theta + (1-\cos\theta)[\omega] + (\theta - \sin\theta)[\omega]^2)v \\ 0 & 1 \end{bmatrix} \tag{B.2}$$

where $[\omega] \in so(3)$ is the skew-symmetric matrix of $\omega \in \mathbb{R}^3$, and $e^{[\omega]\theta}$ is computed using Rodrigues' formula:

$$e^{[\omega]\theta} = I + \sin\theta\,[\omega] + (1-\cos\theta)\,[\omega]^2 \tag{B.3}$$

Equation B.3 assumes $\|\omega\| = 1$ (a unit revolute axis); the URDF-to-POE conversion of Section B.2 normalizes every axis to $\|\omega_i\| = 1$ for revolute joints (or $\|v_i\| = 1$ for prismatic joints) at conversion time, so the formula applies without the general $\|\omega\|$-dependent rescaling throughout this appendix.

B.1.2 Advantages Over DH Parameters

The POE formulation offers four advantages over the Denavit-Hartenberg (DH) convention for the multi-robot configuration system:

1. **No frame assignment ambiguity**: The POE formula uses the space frame and the home configuration, eliminating the non-unique frame assignment problem inherent in DH parameters.

2. **Direct geometric interpretation**: Each screw axis $\mathcal{S}_i$ has a direct physical meaning—the axis of rotation and its location in space.

3. **Uniform representation**: Both revolute and prismatic joints use the same screw axis formulation, simplifying the implementation for heterogeneous robots.

4. **Numerical stability**: The matrix exponential computation avoids the singularities that can arise from certain DH parameter combinations.

B.1.3 C++ Kernel and Python Bindings

The toolkit is implemented as a single C++ translation unit (the "same-kernel" pattern) that contains the POE formulation, the analytical Jacobian, the IK solver, and the NRIC gravity-compensation block. Eigen provides the linear-algebra primitives. The C++ kernel is exposed to Python through a pybind11 binding module (`poe_robotics_cpp`); the bindings use `Eigen::Ref<const T>` on the input side to avoid copies, and the storage-order shim around the binding entry points keeps NumPy row-major arrays compatible with Eigen's column-major default. The Python REST API server (Section B.6) and the ROS2 motion interface (Appendix D.9) both call into the same compiled kernel, so the same screw axes, the same Jacobian, and the same IK iteration are exercised regardless of which interface initiated the request.

B.2 URDF to POE Conversion

An automated conversion tool was developed that extracts joint axes and link poses from URDF (Unified Robot Description Format) files and computes the corresponding POE parameters ($\mathcal{S}_i$, M). The conversion proceeds as follows:

1. Parse the URDF XML to extract the kinematic chain (joints and links).

2. For each joint, compute the axis direction ω_i and a point q_i on the axis in the space frame.

3. Compute the screw axis: $\mathcal{S}_i = (\omega_i, -\omega_i \times q_i)$ for revolute joints, or $\mathcal{S}_i = (0, v_i)$ for prismatic joints.

4. Compute the home configuration M by evaluating the forward kinematics at $\theta = 0$ using the URDF link transforms.

The output is a JSON file containing the screw axes, home configuration, joint limits, and link mesh references. This file is loaded by the configuration system at runtime and supplies the $\mathcal{S}_i$ and M used in Eq. B.1.

The following listing shows the actual POE output for the UFACTORY xArm5, generated by the URDF-to-POE conversion tool. The screw axes matrix $\mathcal{S} \in \mathbb{R}^{6\times5}$ is stored in column-per-joint format, and the home configuration $M \in SE(3)$ is the 4×4 end-effector pose at zero joint angles.


```
{
  "name": "xarm5",
  "base_link": "world",
  "tip_link": "link5",
  "n_joints": 5,
  "joint_names": ["joint1","joint2","joint3","joint4","joint5"],
  "Slist": [
    [ 0.000,  0.000,  0.000,  0.000,  0.000],
    [ 0.000,  1.000,  1.000,  1.000,  0.000],
    [ 1.000,  0.000,  0.000,  0.000, -1.000],
    [ 0.000, -0.267, -0.551, -0.209,  0.000],
    [ 0.000,  0.000,  0.000,  0.000,  0.207],
    [ 0.000,  0.000,  0.053,  0.131,  0.000]
  ],
  "M": [
    [ 1.000,  0.000,  0.000,  0.207],
    [ 0.000, -1.000,  0.000,  0.000],
    [ 0.000,  0.000, -1.000,  0.112],
    [ 0.000,  0.000,  0.000,  1.000]
  ]
}
```


The screw axes reveal the xArm5's kinematic structure: joint 1 rotates about the space-frame Z-axis (vertical), joints 2–4 rotate about axes parallel to the Y-axis (horizontal, with the Y-component ≈ 1.0), and joint 5 rotates about the negative Z-axis at a position offset of 0.207 m along X. The home configuration M places the end-effector at $(0.207, 0, 0.112)$ m with the tool frame pointing downward (180° rotation about X). The values -0.267, -0.551, and -0.209 in the fourth row of $\mathcal{S}$ correspond to the joint axis heights (Z-coordinates) of joints 2, 3, and 4 in the space frame, reflecting the xArm5's link lengths.

The actual POE parameters for each supported robot are generated by the URDF-to-POE conversion tool and stored as JSON files in the shared kinematics data directory.

B.3 Jacobian Computation

The spatial Jacobian $J_s(\theta) \in \mathbb{R}^{6\times n}$ is computed analytically from the screw axes:

$$J_s(\theta) = [\mathcal{S}_1 \quad \mathrm{Ad}_{e^{[\mathcal{S}_1]\theta_1}}\mathcal{S}_2 \quad \cdots \quad \mathrm{Ad}_{T_1\cdots T_{n-1}}\mathcal{S}_n] \tag{B.4}$$

where Ad_T is the adjoint representation of $T \in SE(3)$:

$$\mathrm{Ad}_T = \begin{bmatrix} R & 0 \\ [p]R & R \end{bmatrix} \tag{B.5}$$

where $T \in SE(3)$ has rotation R and translation $p \in \mathbb{R}^3$, and $[p] \in so(3)$ is the 3×3 skew-symmetric matrix associated with p, defined so that $[p]\, v = p \times v$ for all $v \in \mathbb{R}^3$, in the same bracket convention as Equation B.3. The block placement is consistent with the screw-axis ordering $\mathcal{S} = (\omega, v)$ of Equation B.1 (Lynch–Park MR §3.3).

The body Jacobian $J_b(\theta)$ is obtained by transforming the spatial Jacobian to the end-effector frame:

$$J_b(\theta) = \mathrm{Ad}_{T(\theta)^{-1}}\, J_s(\theta) \tag{B.6}$$

B.4 Inverse Kinematics

An iterative numerical IK solver based on the Newton-Raphson method with damped least squares (DLS) regularization is implemented, using the Jacobian defined in the previous section.

B.4.1 Algorithm

Given a target pose $T_d \in SE(3)$ and an initial guess θ_0, the solver iterates:

$$\theta_{k+1} = \theta_k + J^{\dagger}(\theta_k)\, \xi_k \tag{B.7}$$

where $J^{\dagger}$ is the damped pseudoinverse of the spatial Jacobian:

$$J^{\dagger} = J^T (J J^T + \lambda^2 I)^{-1} \tag{B.8}$$

and $\xi_k \in \mathbb{R}^6$ is the twist error computed from T_d and the current pose $T(\theta_k)$:

$$\xi_k = \log(T_d\, T(\theta_k)^{-1})^{\vee} \tag{B.9}$$

The damping factor λ prevents large joint corrections near kinematic singularities. The solver terminates when $\|\xi_k\| < \epsilon$ or after a maximum number of iterations. Table B.1 lists the default parameters from the implementation.

Table B.1: IK solver default parameters from the codebase.

Parameter	Symbol	IKConfig	POE function
Angular error tolerance	ϵ_ω	10^{-3} rad	10^{-3} rad
Linear error tolerance	ϵ_v	10^{-4} m	10^{-4} m
Damping factor	λ	0.01	—
Maximum iterations		50	20

The differential IK solver (`DiffIKConfig`) also specifies a maximum joint velocity limit of 2.0 rad/s, joint limit clamping enabled by default, and a singularity detection threshold of 0.01 (on the minimum singular value of the Jacobian). Adaptive damping, when enabled, increases λ near singular configurations proportional to the reciprocal of the condition number.

B.4.2 Convergence

For redundant robots (7+ DoF), the nullspace component is used to optimize a secondary objective (e.g., joint limit avoidance or manipulability maximization).

B.5 Gravity Compensation: Nominal Reference Inverse Compensator (NRIC)

The toolkit includes a nominal reference inverse compensator (NRIC) that computes a feed-forward gravity torque from the same kinematic chain that the FK and IK use. The NRIC follows the form of Eq. 67 in (Kim *et al.*, 2025): the joint-space gravity vector $g(\theta)$ is evaluated from a nominal mass and centre-of-mass model loaded alongside the screw axes, and the controller adds it to the impedance or admittance command before the joint-space command is sent to the driver. The NRIC sits in a separate translation unit from the kinematic chain so that the kinematic computations remain pure (no controller-side coupling), and the impedance/admittance controllers compose the chain-level FK/Jacobian with the NRIC torque at the controller boundary.

The cross-embodiment table reported in the engineering archive (multi-platform impedance/admittance validation on the KUKA IIWA14, UR16e, Lite6, and xArm5) was produced with NRIC enabled on the four platforms; the NRIC parameters per platform are loaded from the same per-robot YAML files that supply the kinematic chain.

B.6 REST API Server

The kinematics computations are exposed through a REST API server for non-ROS2 clients such as web interfaces and Unity applications. Table B.2 summarizes the available endpoints.

Table B.2: POE Robotics REST API endpoints.

Method	Endpoint	Description
POST	`/api/fk`	Forward kinematics: joint angles → end-effector pose
POST	`/api/ik`	Inverse kinematics: target pose → joint angles
POST	`/api/jacobian`	Jacobian matrix at given configuration
GET	`/api/robots`	List available robot configurations
GET	`/api/workspace`	Compute reachable workspace boundaries

The API server loads robot configurations from the same YAML files used by the ROS2-based motion interface, maintaining consistency between the two access paths.

B.7 Workspace Analysis

The workspace analysis tool computes the reachable workspace of a robot by sampling the joint space and evaluating the forward kinematics at each sample. Uniform grid sampling with configurable resolution is used. For each sampled configuration, the end-effector position is recorded and the convex hull of all reachable positions is computed. This analysis is used to verify that planned rehabilitation patterns fit within the robot's workspace and to compare workspace coverage across different robot platforms in the multi-robot system. The cube-localized variant of this analysis at the HRI home position is reported in Appendix C.

APPENDIX C

KINEMATIC ANALYSIS AND WORKSPACE CHARACTERIZATION

This appendix presents kinematic analysis results for the robots in the multi-HRI framework. The flow is: (i) pose-convention vocabulary, (ii) zero-configuration singularity analysis (results first, mathematical background last), (iii) the HRI home positions designed to avoid those singularities, (iv) a 20 cm-cube workspace analysis at each HRI home, and (v) the dual-arm hardware configurations and their gantry-mounted kinematics. The POE formulation and tooling that underlie the computations are documented in Appendix B.

C.1 Manufacturer Pose Conventions

Industrial robotics distinguishes several pose types that are often conflated. Table C.1 fixes the vocabulary used throughout this appendix and across the framework.

Table C.1: Pose convention definitions.

Pose Type	**Purpose**
Zero Pose	Coordinate reference, calibration, joint encoder zeroing
Home Pose	Safe position for control initialization, singularity-free
Ready Pose	Task entry position, optimized for workspace access
Control Pose	Optimal manipulability for specific task execution

All manufacturers (UR, KUKA, Franka, Kinova) distinguish between zero and home configurations in their documentation and control software. The remainder of this appendix uses "zero" and "home" in the senses defined above.

C.2 Zero Configuration Singularity Analysis

C.2.1 Motivation

A common misconception in robotics practice is that the URDF zero configuration (all joint angles equal to zero) represents a "safe" or operational pose. This section demonstrates that zero configurations typically contain kinematic singularities due to axis alignment, and should be distinguished from operational home positions in the sense of Table C.1.

C.2.2 Experimental Results

Table C.2 presents the Jacobian rank, minimum singular value, and condition number at the URDF zero configuration for all robots in the multi-HRI framework. Every tested platform exhibits rank deficiency at zero, classified by the underlying

cause. The mathematical definitions of $\sigma_{\min}$, $\kappa(J)$, and the manipulability index are given in Section C.2.6 below; this subsection reports the result first.

Table C.2: Zero configuration singularity analysis.

Robot	**DoF**	**Rank**	$\sigma_{\min}$	$\kappa(J)$	**Classification**
Universal Robots (Wrist Singularity)					
ur3e	6	5/6	0.0000	∞	Wrist Alignment
ur5e	6	5/6	0.0000	∞	Wrist Alignment
ur16e	6	5/6	0.0000	∞	Wrist Alignment
UFACTORY					
xarm5	5	2/3	0.0000	∞	Under-actuated
lite6	6	5/6	0.0000	∞	Axis Alignment
KUKA (Intentional Calibration Pose)					
iiwa7	7	3/6	0.0000	∞	Symmetric Design
iiwa14	7	3/6	0.0000	∞	Symmetric Design
med7	7	3/6	0.0000	∞	Symmetric Design
med14	7	3/6	0.0000	∞	Symmetric Design
Research Robots					
fr3	7	5/6	0.0000	∞	Task-space Alignment
kinova_gen3	7	4/6	0.0000	∞	Symmetric Alignment

C.2.3 Universal Robots Wrist Singularity

The UR series exhibits the classical wrist singularity at zero configuration. Analysis of the UR5e Jacobian reveals:

$$J_s(0) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -0.16 & -0.16 & -0.16 & -0.13 & -0.06 \\ 0 & 0 & 0 & 0 & 0.82 & 0 \\ 0 & 0 & 0.43 & 0.82 & 0 & 0.82 \end{bmatrix} \tag{C.1}$$

The angular velocity components show that joints 2, 3, 4, and 6 all rotate about parallel Y-axes ($\omega_y = 1$), creating linear dependence among four columns. This results in rank deficiency (rank 5 instead of 6) and loss of one rotational degree of freedom.

C.2.4 KUKA iiwa Series

The KUKA iiwa series exhibits rank reduction at zero configuration (rank 3/6). This is not a design defect but an intentional choice:

- Zero pose serves as a **calibration reference**, not an operational pose
- Symmetric geometry simplifies kinematic calibration
- Gravity compensation is simplified at symmetric configurations

KUKA officially documents that the zero pose is not intended for motion control, with separate home and control poses defined for operation.

C.2.5 Interpretation

> *"Most commercial robots define URDF zero configuration for calibration convenience rather than operational safety. The zero pose typically contains kinematic singularities due to intentional axis alignment for manufacturing and calibration purposes."*

C.2.6 Mathematical Background

This subsection collects the singularity definitions and metrics that underlie the result in Table C.2. Kinematic singularities occur when the manipulator Jacobian matrix $J(q)$ loses rank:

$$\text{rank}(J(q)) < \min(m, n) \tag{C.2}$$

where m is the task space dimension (typically 6) and n is the number of joints. At singularity, certain end-effector velocities become impossible to achieve with finite joint velocities. The proximity to singularity is quantified through singular value decomposition:

$$J = U\Sigma V^T, \quad \Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) \tag{C.3}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthonormal, $r = \min(m, n)$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$.

Singularity Metrics

Minimum Singular Value: Direct indicator of singularity

$$\sigma_{min} \to 0 \implies \text{singularity} \tag{C.4}$$

Condition Number: Ratio of velocity amplification extremes

$$\kappa(J) = \frac{\sigma_{max}}{\sigma_{min}}, \quad \kappa \to \infty \implies \text{ill-conditioned} \tag{C.5}$$

Manipulability Index (Yoshikawa, 1985): Volume of the velocity ellipsoid

$$w(q) = \sqrt{\det(J(q)J^T(q))} = \prod_{i=1}^{r} \sigma_i \tag{C.6}$$

Equation C.6 assumes $m \leq n$. For under-actuated arms with $m > n$ (e.g., the xArm5 with $n = 5$ joints and $m = 6$ task-space dimensions), $JJ^\top \in \mathbb{R}^{m\times m}$ is rank-deficient by construction and $\det(JJ^\top) = 0$ identically; the position-restricted form of Section C.4.5 (Equation C.10) is used in that case. The equivalent right-form $w(q) = \sqrt{\det(J^\top J)}$ (Yoshikawa, 1985) reduces to the same product of nonzero singular values and generalizes both regimes.

Manipulability Ellipsoid

The manipulability ellipsoid visualizes the set of end-effector velocities achievable with unit joint velocity:

$$\mathcal{E} = \{\dot{x} \in \mathbb{R}^m : \dot{x} = J\dot{q}, \|\dot{q}\| \leq 1\} \tag{C.7}$$

The ellipsoid axes are defined by the columns of U, with semi-axis lengths σ_i. At singularity, one or more $\sigma_i \to 0$, collapsing the ellipsoid.

Velocity-Force Duality

The relationship between velocity transmission and force transmission is dual:

$$\dot{x} = J\dot{q} \quad \Leftrightarrow \quad \tau = J^T F \tag{C.8}$$

Directions of poor velocity transmission correspond to directions of high force capability, and vice versa. At singularity:

- Lost velocity DoF becomes a direction of infinite force capability
- Requires infinite joint torque to generate velocity in the singular direction

Position-Only Jacobian for Under-Actuated Arms

For robots with DoF < 6 (e.g., xArm5 with 5 joints), the position-only Jacobian extracts the linear-velocity rows. Under the screw-axis ordering $\mathcal{S}_i = (\omega_i, v_i)$ used in Appendix B (rows 1–3 angular, rows 4–6 linear), this is

$$J_p = J_{4:6,:} \in \mathbb{R}^{3\times n}. \tag{C.9}$$

Manipulability is then computed on the position subspace:

$$w_p(q) = \sqrt{\det(J_p J_p^T)} \tag{C.10}$$

The implications of this restriction for the xArm5 are discussed in Section C.4.5 below.

C.3 HRI Home Position Design

C.3.1 Design Requirements

For human-robot interaction applications, the home position must satisfy:

1. End-effector positioned for human interaction (forward reach ∼50-70 cm)
2. Singularity-free configuration ($\sigma_{min} > 0.1$)
3. Low condition number ($\kappa(J) < 20$) for stable force control
4. Sufficient workspace coverage around the interaction point

C.3.2 Recommended HRI Home Positions

Table C.3 presents optimized home configurations for HRI applications.

Table C.3: HRI-optimized home positions.

Robot	**DoF**	**Home Configuration [deg]**
ur3e	6	[0, −45, 90, −45, 90, 0]
ur5e	6	[0, −45, 90, −45, 90, 0]
ur16e	6	[0, −45, 90, −45, 90, 0]
lite6	6	[0, −30, 60, 0, 90, 0]
xarm5	5	[0, 0, 0, 0, 0] + work offset
fr3	7	[0, −30, 0, −120, 0, 90, 45]
iiwa7/14	7	[0, 30, 0, −60, 0, 60, 0]
med7/14	7	[0, 30, 0, −60, 0, 60, 0]
kinova_gen3	7	[0, 30, 180, −60, 0, −60, 90]

C.4 20 cm Cube Workspace Analysis

C.4.1 Motivation

HRI applications typically involve a localized workspace where human and robot hands interact. A 20 cm × 20 cm × 20 cm cube centered at the end-effector position provides a practical characterization of the manipulable region for handshake-style interactions.

C.4.2 Mathematical Formulation

Workspace Definition

The HRI workspace is defined as a cube $\mathcal{W}$ centered at the home end-effector position:

$$\mathcal{W} = \{p \in \mathbb{R}^3 : \|p - p_{home}\|_\infty \leq 0.1 \text{ m}\} \tag{C.11}$$

where $p_{home} = T_{home}[0:3, 3]$ is the end-effector position at the home configuration.

Reachability Criterion

A target pose $T_d \in SE(3)$ is *reachable* if inverse kinematics converges:

$$\exists q \in \mathcal{Q} : T_{sb}(q) = T_d, \quad \|q - q_{home}\| < q_{max} \tag{C.12}$$

where $\mathcal{Q}$ is the joint limit manifold and q_{max} constrains the solution to remain near the home configuration.

Workspace Quality Metrics

For the reachable set $\mathcal{W}_r \subseteq \mathcal{W}$:

Reachability Ratio:

$$R = \frac{|\mathcal{W}_r|}{|\mathcal{W}|} \times 100\% \tag{C.13}$$

Singularity-Free Ratio (points with $\kappa(J) < 50$):

$$S_f = \frac{|\{p \in \mathcal{W}_r : \kappa(J(q_p)) < 50\}|}{|\mathcal{W}_r|} \times 100\% \tag{C.14}$$

Average Manipulability:

$$\bar{w} = \frac{1}{|\mathcal{W}_r|} \sum_{p \in \mathcal{W}_r} w(q_p) \tag{C.15}$$

C.4.3 Analysis Methodology

For each robot at its HRI home position:

1. Compute forward kinematics to determine end-effector position p_{home}
2. Sample 300 random points within a 20 cm cube centered at p_{home}
3. Attempt inverse kinematics for each target point (maintaining home orientation)
4. Evaluate Jacobian condition at successful IK solutions
5. Compute reachability percentage and singularity occurrence rate

C.4.4 Results

Table C.4 presents the 20 cm cube workspace analysis results.

Table C.4: 20 cm cube workspace analysis at HRI home position.

Robot	**EE Position [m]**	**Reach%**	**Sing%**	$\kappa(J)$
ur5e	[0.68, 0.13, 0.09]	100%	0%	8.5
ur16e	[0.71, 0.17, 0.14]	100%	0%	8.9
fr3	[0.39, 0.00, 0.62]	100%	0%	10.2
ur3e	[0.42, 0.13, 0.09]	99%	0%	15.5
iiwa7	[0.66, 0.00, 0.58]	97%	0%	13.4
iiwa14	[0.67, 0.00, 0.61]	97%	0%	13.4
med7	[0.66, 0.00, 0.58]	98%	0%	13.4
med14	[0.67, 0.00, 0.61]	97%	0.3%	13.4
kinova_gen3	[0.61, 0.00, 0.50]	96%	0.7%	14.4
lite6	[0.13, 0.00, 0.44]	100%	0%	23.1
xarm5*	[0.21, 0.10, 0.21]	100%	0%	13.5

*xarm5: Position-only mode with work offset [0, +10, +10] cm from zero configuration.

C.4.5 5-DoF Robot Considerations

The xArm5 has only 5 degrees of freedom, structurally precluding full SE(3) control (the position-only Jacobian and manipulability metric were defined in Eqs. C.9 and C.10). For HRI applications:

- Position control (3-DoF) is fully achievable
- Orientation control limited to 2-DoF (roll axis constrained)
- Work offset from zero configuration provides optimal position-only workspace

This restriction is also why the xArm5 row of Table C.4 reports a position-only result with a work offset, and why the cross-embodiment validation in the engineering archive marks the 5-DoF Lie-group case as DoF<6 rather than as a tracking failure.

C.5 Dual-Arm Robot Configuration

This section documents the three dual-arm hardware configurations and the gantry-mounted forward kinematics that compose their reachable sets. The bilateral teleoperation control parameters that drive these configurations (leader–follower dynamics, position and force scaling, impedance pairing, passivity condition) are documented in the cited works of the KAIST IRiS Lab (Lee and Park, 2024b,a; Lee *et al.*, 2025, 2024a), which developed the controller; the dissertation's contribution is the architectural integration of the controller into the configuration hierarchy of Chapter 9.

C.5.1 System Overview

The multi-HRI framework supports three dual-arm configurations for bilateral teleoperation, each with a distinct mounting frame and kinematic chain. Table C.5 summarizes the configurations.

Table C.5: Dual-arm robot system configurations.

System	Role	Mount Type	Mobile	QR Code
UR5e + UR16e	Leader/Follower	Independent bases	No	Q8, Q9
Kinova Gen3 × 2	Follower	Gantry + Triangle	Yes (4-DoF)	Q111
Franka FR3 × 2	Leader	Fixed LeaderMount	No	Q112

C.5.2 Kinova Gantry Dual-Arm System (Follower)

The Kinova Gen3 dual-arm system is mounted on a 4-DoF gantry with a triangular connection hardware, serving as the follower side for bilateral teleoperation.

Gantry Structure

The gantry provides workspace extension through 4 degrees of freedom:

Table C.6: Kinova gantry joint specifications.

Joint	Type	Axis	Range	Parent → Child
joint_1	Prismatic	X	$[-1.8,\ 1.7]$ m	base_link → link_1
joint_2	Prismatic	Y	$[-0.45,\ 0.45]$ m	link_1 → link_2
joint_3	Prismatic	Z	$[0,\ 0.25]$ m	link_2 → link_3
joint_4	Revolute	Z	$[0,\ \pi]$ rad	link_3 → link_4

Kinematic Chain

The complete kinematic chain from world to each arm end-effector is:

$$\text{World} \to \text{base\_link} \xrightarrow{\text{4-DoF}} \text{link\_4} \xrightarrow{\text{fixed}} \text{connecting\_hardware} \xrightarrow{\text{fixed}} \text{arm\_base} \tag{C.16}$$

Triangle Mount Attachment

The triangular connection hardware (connecting_hardware) attaches to gantry link_4 with a 90° X-axis rotation:

$$T_{\text{link4}\to\text{triangle}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{C.17}$$

Arm Attachment Parameters

Table C.7 presents the arm attachment offsets from the triangular mount.

Table C.7: Kinova dual-arm attachment parameters (from triangle mount).

Arm	**XYZ [m]**	**RPY [rad]**	**Description**
Left Kinova	$(0.03,\ -0.585,\ 0.14)$	$(-\frac{\pi}{4},\ 0,\ -\pi)$	Y- direction, 45° tilt
Right Kinova	$(0.03,\ -0.585,\ -0.14)$	$(-\frac{3\pi}{4},\ 0,\ -\pi)$	Y- direction, 135° tilt

Combined Degrees of Freedom

The total system DoF is 4 (gantry) $+ 7 \times 2$ (arms) $+ 1 \times 2$ (grippers) $= 20$ DoF, where each arm uses a Robotiq 2F-140 gripper.

C.5.3 Franka Leader Dual-Arm System (Leader)

The Franka FR3 dual-arm system serves as the leader side for teleoperation, mounted on a fixed LeaderMount structure.

Mount Configuration

Unlike the Kinova follower system, the Franka leader uses a **fixed** mount (no gantry):

- **LeaderMount**: Fixed structure attached to world frame
- **No mobile base**: Position is fixed in the operator workspace
- **Symmetric arm placement**: Arms at ±45° X-axis rotation

Arm Attachment Parameters

Table C.8 presents the Franka arm attachment from world frame.

Table C.8: Franka dual-arm attachment parameters (from world).

Arm	XYZ [m]	RPY [rad]	Description
Left FR3	$(0.049,\ 0.121,\ 0.872)$	$(-\frac{\pi}{4},\ 0,\ 0)$	Y+ side, $-45°$ X
Right FR3	$(0.049,\ -0.121,\ 0.872)$	$(\frac{\pi}{4},\ 0,\ 0)$	Y- side, $+45°$ X

Force/Torque Sensing

Each Franka arm is equipped with an ATI NetFT force/torque sensor integrated into the custom end-effector:

- **Sensor**: ATI NetFT (6-DoF F/T)
- **Protocol**: RDT over UDP
- **Integration**: Sensor mesh integrated into end-effector STL

C.5.4 UR5e + UR16e Teleoperation System

The UR5e and UR16e operate as independent fixed-base robots for single-arm bilateral teleoperation:

- **UR5e**: Leader robot (operator side)
- **UR16e**: Follower robot (remote side)
- **Mount**: Each robot has independent fixed base (no shared mount)
- **Gripper**: Robotiq 2F-85 on both robots
- **Total DoF**: $6 + 6 = 12$ DoF

This configuration is suitable for tasks where the operator controls a single remote arm, rather than bimanual manipulation, and is the platform configured for the dual-arm bilateral teleoperation collaboration of Chapter 9, Section 9.6 (Integration Case 3).

C.5.5 Gantry-Mounted Dual-Arm Kinematics

For the Kinova dual-arm system mounted on the gantry via triangular hardware, the forward kinematics of each arm is:

$$T_{left}(q_g, q_L) = T_g(q_g) \cdot T_{tri} \cdot T_{offset,L} \cdot T_L(q_L) \tag{C.18}$$

$$T_{right}(q_g, q_R) = T_g(q_g) \cdot T_{tri} \cdot T_{offset,R} \cdot T_R(q_R) \tag{C.19}$$

where:

- $q_g \in \mathbb{R}^4$: Gantry joint configuration (3 prismatic + 1 revolute)

- $q_L, q_R \in \mathbb{R}^7$: Left/right arm joint configurations
- $T_g(q_g)$: Gantry forward kinematics (Eq. C.16)
- T_{tri}: Fixed transformation from link_4 to triangle mount (Eq. C.17)
- $T_{offset,L/R}$: Arm attachment offset (Table C.7)

The combined body Jacobian at the arm end-effector frame, in the convention $J_b = \mathrm{Ad}_{T^{-1}} J_s$ of Appendix B (Section B.1.3), is

$$J_{combined} = \begin{bmatrix} \mathrm{Ad}_{(T_{tri}T_{offset}T_{arm})^{-1}} J_g & J_{arm} \end{bmatrix} \in \mathbb{R}^{6\times 11}, \tag{C.20}$$

where J_g is the gantry's body Jacobian at the gantry tip (link_4), J_{arm} is the arm's body Jacobian at its own end-effector, and the inverse adjoint transports the gantry columns from the gantry tip frame to the arm-tip frame through the fixed transforms T_{tri} and T_{offset} together with the arm's instantaneous configuration T_{arm}.

C.5.6 Mobile Base Advantage

The 4-DoF gantry provides workspace extension for dual-arm manipulation through the union of the arm workspace over the gantry pose set:

$$\mathcal{W}_{total} = \bigcup_{q_g \in \mathcal{Q}_g} T_g(q_g) \cdot \mathcal{W}_{arms} \tag{C.21}$$

where $\mathcal{Q}_g$ is the gantry joint manifold and $T_g(q_g) \in SE(3)$ is the gantry forward kinematics. When the gantry is restricted to the three prismatic joints, the union reduces to the Minkowski sum $\mathcal{W}_{gantry,\mathrm{trans}} \oplus \mathcal{W}_{arms}$; the revolute joint about Z extends the union further by sweeping the arm workspace through $[0, \pi]$.

C.5.7 Workspace Coordination

For dual-arm manipulation, the combined workspace must account for:

- **Collision avoidance**: Minimum 10 cm separation between arm links
- **Shared workspace**: Overlapping region for bimanual tasks
- **Singularity-free zone**: Both arms maintain $\kappa(J) < 20$

C.6 Additional Platforms (Analysis Pending)

The platforms documented in the previous sections are the eleven single-arm 5/6/7-DoF cobots that were available at the time of the framework's first round of kinematic profiling. Three platform classes were added to the framework after that round and are listed here with the analysis state declared explicitly, so that the absence of these platforms from Tables C.2 and C.4 is not interpreted as an oversight.

C.6.1 KUKA Industrial RSI Arms (KR10 R1100-2, KR210 R2700-2)

The KUKA Agilus and Quantec arms documented in Appendix A, Section A.4, are 6-DoF serial manipulators of standard structure (offset wrist with three intersecting wrist axes); their zero-configuration singularity classification is expected to fall in the same "wrist alignment" family as the UR series in Table C.2, and their HRI home position is the canonical industrial pose $[0, -90, 90, 0, 0, 0]^\circ$ recorded in the override YAML. The 20 cm-cube workspace metric is sensitive to base mounting and to the operator's task-anchor frame, both of which differ from the cobot deployments in Table C.4. The cube metric is therefore deferred until the heavy-manipulator follow-on listed in Chapter 14 matures.

C.6.2 Multi-Arm Composite Systems (Franka Dual, Kinova Dual, Kinova Gantry)

The dual-arm composite systems documented in Appendix A, Section A.7, compose two 7-DoF arms (FR3 or Gen3) on a fixed mount or on a 4-DoF gantry. Per-arm singularity at zero matches the Franka FR3 and Kinova Gen3 entries in Table C.2, since the dual-arm system inherits the kinematic chain of the single arm with a fixed base offset. The new objects of analysis are the bimanual workspace overlap, the gantry-mounted reachability, and the collision-free zone between the two arms; the gantry-mounted forward kinematics (combined Jacobian, Eq. C.20) is documented in Section C.5 above. The workspace overlap and the collision-free zone are pending; they will be reported alongside the deferred bilateral-teleoperation measurement set listed in Table 14.2 of Chapter 14.

C.6.3 Rainbow Robotics RB-Y1 Humanoid (Q20)

The RB-Y1 wheeled humanoid documented in Appendix A, Section A.8, is the framework's first 22-DoF platform. Its kinematic chain decomposes into four sub-chains (mobile base, torso, left arm, right arm); the POE solver was validated on each sub-chain and on the full `base` → `ee_left` 13-DoF chain in January 2026. Singularity analysis on the humanoid differs from the serial-arm case in two ways: the torso-and-arm composition admits multiple sub-task formulations (left-arm-only, right-arm-only, dual-arm, torso-as-base), and the mobile base adds two omnidirectional wheel joints whose Jacobian column structure is rank-1 each. The tables of this appendix do not yet include RB-Y1 entries because the per-sub-chain singularity classification, the bimanual workspace overlap, and the body-frame HRI home position are scheduled to be reported as part of the Exemplar E2 measurement set on the physical platform. The override YAML lists an estimated home configuration that the planner uses in the meantime.

C.7 Summary

Key results:

1. **Zero configuration singularity is universal**: All tested robots exhibit singularities at URDF zero pose, by design rather than defect.

2. **HRI home positions achieve full workspace**: Properly designed home configurations provide 95–100% reachability within a 20 cm interaction cube with no singularities.

3. **Condition number predicts control quality**: Robots with $\kappa(J) < 15$ (ur5e, ur16e, fr3) provide the most isotropic force transmission for impedance-controlled HRI.

4. **5-DoF robots require position-only mode**: The xArm5 achieves full position-space reachability when orientation constraints are relaxed.

The POE formulation, the URDF-to-POE conversion that supplies $\mathcal{S}_i$ and M, the analytical Jacobian, and the IK solver used throughout this analysis are documented in Appendix B.

APPENDIX D

CONFIGURATION REFERENCE

This appendix provides the implementation details of the parameterized configuration system described in Chapter 9. The structure is: (i) the basic YAML configuration hierarchy (robot, task, motion pattern), (ii) the QR-coded spatial-anchor mapping that ties each robot to a 6-DoF base offset, (iii) the behavior-tree (BT) topology and the multi-user policy tuple that resolve what each participant sees and drives, (iv) experiment-level scenario configuration, (v) per-user profile and device portal configuration that bind individual humans and headsets into the multi-user session, (vi) the network and communication layer that carries the runtime traffic, (vii) the Docker service architecture that packages everything for deployment, and (viii) the unified motion interface that consumes the above. The AI agent profiles that occupy a parallel slot in the YAML hierarchy are documented in Appendix E.

D.1 YAML Configuration Hierarchy

The configuration system is organized into hierarchical levels, each managing a distinct aspect of the multi-robot platform. All configurations use YAML format with environment variable expansion, allowing deployment-specific customization without code modification. Network and AI-agent settings, which sit at the same level in the directory structure, are documented separately—network in Section D.7 below and AI agents in Appendix E.

D.1.1 Robot Configuration

Each robot platform is defined by a YAML file specifying its kinematic properties, control parameters, and communication endpoints. The configuration follows a base-override pattern: a default template (`_defaults.yaml`) provides common settings, and robot-specific files override only the parameters that differ. The following listing shows the xArm5 configuration as an example.

```
# xArm5 Robot Override Configuration
# Override values supersede _defaults.yaml

controllers:
  position_command_topic: "/xarm5_traj_controller/joint_trajectory"
  joint_state_topic: "/joint_states"
  trajectory_action: "/xarm5_traj_controller/follow_joint_trajectory"
  control_rate: 250.0  # Hz

robot:
  model: "xarm5"
  manufacturer: "UFACTORY"
  dof: 5
  prefab_name: "xarm5_1305"
```

```
joints:
  names: ["joint1", "joint2", "joint3", "joint4", "joint5"]
  positions:
    zero: [0, 0, 0, 0, 0]
    home: [0, 0, 0, 0, 0]
  limits:
    joint1: [-360, 360]
    joint2: [-118, 120]
    joint3: [-225, 11]
    joint4: [-360, 360]
    joint5: [-360, 360]

frame_hierarchy:
  network_frames:
    "{robot_id}_base":
      ros_frame: "link_base"
    "{robot_id}_eef":
      ros_frame: "link_eef"

control:
  type: "moveit"
  relay:
    actions:
      follow_joint_trajectory:
        from: "/{robot_id}/xarm5_traj_controller/..."
        to: "/xarm5_traj_controller/..."
```

The `{robot_id}` placeholder is resolved at runtime, so that multiple instances of the same robot type to coexist on the same network. Adding a new robot platform to the system requires creating a YAML file following this schema; no source code modification is necessary.

D.1.2 Task Configuration

High-level rehabilitation tasks are defined as sequences of typed operations. Each operation specifies a type (waypoint, trajectory, pattern, gripper action, or vision query) and its parameters. The following listing shows the default task definition structure.

```
tasks:
  - id: 1
    name: "Inspection Task A"
    description: "Visual inspection sequence"
    operations:
      - name: "Position Check"
        type: "waypoint"
      - name: "Pick Tool"
```

```
      type: "gripper"
    - name: "Moving Path"
      type: "trajectory"
    - name: "Execute Operation"
      type: "action"
    - name: "Return Tool"
      type: "gripper"

# Operation types:
#   waypoint    - single position move
#   trajectory  - path following
#   pattern     - parameterized motion (circle, diamond, etc.)
#   gripper     - gripper open/close/pick/place
#   action      - general action (service call)
#   vision      - vision-based operation
```

D.1.3 Unit Pattern Configuration

Six parameterized motion patterns (circle, square, diamond, spiral, hemisphere, triangle) are defined as unit patterns with a normalized radius of 1.0. At runtime, the system scales and transforms these patterns to match the robot's workspace.

```
# Circle Pattern (unit-based, generated at runtime)
coordinate_system: ENU
pattern_type: circle

defaults:
  radius: 0.20            # meters (actual radius)
  height: 0.0             # meters (z offset)
  num_points: 16           # points on circle
  duration_per_point: 0.5  # seconds

# Generator creates points on unit circle (radius=1)
# then multiplies by defaults.radius
```

D.2 Robot QR Mapping and Base-Pose Offsets

The framework uses QR-coded fiducials to register each robot's base frame in the shared mixed-reality coordinate system. The mapping file `config/robot_qr_mapping.yaml` associates each QR identifier with a robot type (matched to the override YAML in Section D.1) and a 6-DoF base-pose offset expressed as an East-North-Up position vector and a unit quaternion. The QR identifier ranges are organized by morphology: Q1–Q11 single-arm cobots, Q12–Q19 KUKA RSI heavy-duty arms, Q20–Q29 humanoid platforms, Q101–Q102 general-purpose markers (calibration, reference points), and Q111–Q113 multi-arm composite systems. The QR identifiers and the per-robot specifications are documented in Appendix A, Table A.1.

```yaml
# robot_qr_mapping.yaml -- single-arm range Q1-Q11
robot_qr_codes:
  Q9:
    robot_type: "ur16e"
    robot_base_offset:
      position: [0, -0.381, 0.01]   # ENU: 0 East, 381 mm South, 10 mm Up
      rotation: [0, 0, 0, 1]        # identity quaternion (xyzw)

# Humanoid range Q20-Q29 -- wheel/bipedal locomotion
humanoid_qr_codes:
  Q20:
    robot_type: "rby1"
    locomotion: "wheel"
    description: "Rainbow Robotics RB-Y1 (Wheel + Dual Arm, 22 DoF)"
    robot_base_offset:
      position: [0, 0, 0]           # wheel center == QR position
      rotation: [0, 0, 0, 1]

# Composite-system range Q111-Q113 -- dual-arm and gantry
robot_system_qr_codes:
  Q112:
    system_type: "franka_dual_arm"
    description: "Franka FR3 x2 Dual-Arm on MasterMount"
    robots:
      - { model: "fr3", role: "left" }
      - { model: "fr3", role: "right" }
```

The base-pose offset is consumed by the Unity QR-anchor subsystem to place the virtual twin of each robot at the correct position in the operator's headset coordinate frame, and by the planner to express end-effector targets in the world frame. Because the offset is part of the configuration rather than the URDF, the same robot model can be re-deployed at a different location simply by re-printing the QR fiducial or by editing the offset entry; no robot code or ROS launch file changes.

D.3 Behavior Tree Topology and Multi-User Policy

The behavior-tree (BT) topology configuration encodes the visualization-vs-control decoupling described in Chapter 9. Each BT topology YAML defines (a) the deployment *targets* (which container at which site runs which subsystem), (b) one or more *contracts* (named scenarios with a contract name, a description, and a multi-user policy), (c) the *participants* (human users and robots) with a per-participant role, and (d) a list of *cards* (load-time bringup steps, execute-time actions, and unload steps).

D.3.1 Topology and Targets

The `targets` block declares the deployment endpoints of each subsystem. Each target specifies a site identifier (resolved to a host through the network configuration

of Section D.7), a Docker container, a workspace directory, and the ROS distribution and domain ID at which the subsystem runs. Skill-service targets are resolved per-site through the anl-overlay network configuration, so that the same BT contract can be re-deployed at a different site by changing only the site identifier.

```
topology:
  kind: "bt_connectivity"
  version: 2
  project_id: "asu_ur16e_pattern_humble"

targets:
  phantom:
    site: "anl_ai"
    host: "192.168.100.2"
    container: "phantom-ros2"
    ros_distro: "humble"
    ros_domain_id: 1
  ros2:
    site: "anl_robot"
    host: "anl-robot-server"
    container: "ros2-serial-robot"
    ros_distro: "humble"
    ros_domain_id: 1
  skills:
    msg_skills:
      site: "anl_alienware"
```

D.3.2 Multi-User Policy Tuple

The multi-user policy tuple is a five-field block that resolves what each participant sees, what they move, what they passively follow, who they own as a session role, and which spatial QR they map to. The five fields are arrays of equal length whose index corresponds to the participant ordering in the `participants` list. The tuple encodes the visualization-vs-control separation described in Chapter 9.

Table D.1: Multi-user policy tuple — five fields per participant. The same tuple format is used in BT topology contracts and in experiment-level configuration (Section D.4); the framework's server applies the resolution rule consistently across both layers.

Field	Meaning
`see`	Which visualization layer the participant observes (e.g., `VirtualGuide`, `VirtualTwin`, real).
`move`	Which visualization layer the participant actively drives (e.g., `VirtualGuide` for the therapist, blank for read-only roles).
`follow`	Which visualization layer the participant passively follows when not driving (e.g., `VirtualTwin` mirrors the real robot to a remote observer).
`owner`	Session-level role taxonomy: `therapist`, `patient`, `remote_therapist`, etc. The owner field disambiguates duplicates of the same robot mapping across users.
`mapping`	QR identifier or device identifier the participant is bound to (e.g., `Q9` for the UR16e, `phantom` for the haptic master).

```
# bt_phantom_ur16e.yaml -- Exemplar E1 (PHANToM master, UR16e slave)
contracts:
  phantom_ur16e_bt:
    description: "PHANToM master + UR16e slave buffered teleop"
    multi_user_policy:
      see:     ["VirtualGuide", "VirtualTwin"]
      move:    ["VirtualGuide", ""]
      follow:  ["",             "VirtualTwin"]
      owner:   ["therapist",    "patient"]
      mapping: ["phantom",      "Q9"]
    participants:
      - id: "phantom"
        model: "phantom_omni"
        role: ["master", "input_device", "leader"]
        target: "phantom"
      - id: "ur16e_Q9"
        model: "ur16e"
        role: ["slave", "robot", "follower"]
        target: "ros2"
```

In the example above, the therapist (participant index 0) sees the virtual guide and actively drives it through the PHANToM haptic master; the patient (participant index 1) sees the virtual twin (which mirrors the real UR16e), does not drive it, and passively follows it. The same contract format scales to a three-user remote-rehabilitation scenario simply by extending the arrays to length three; the experiment configuration in Section D.4 below shows the corresponding three-participant tuple.

D.3.3 Cards: Load, Execute, Unload

Each contract declares a list of cards in three categories: `load` cards bring up subsystems (drivers, controllers, teleop bridges) and verify readiness through topic, action, and service probes; `execute` cards run the actual scenario logic (forward-trajectory dispatch, pattern execution); and `unload` cards tear down subsystems in reverse order. Each card declares a target and an optional `ready` block listing the topics, actions, and timeouts that must be satisfied before the card is considered loaded. The card runlist is dispatched by the framework backend through a single `card_runlist` mode; the same backend handles the auto-resolution between fake/real modes that previous versions of the framework split into separate scenarios.

D.4 Experiment Configuration

The experiment configuration is the highest-level scenario specification in the framework: it declares the experimental design (factors, trials per condition, counterbalance order), the recording preset (which ROS topics to capture into a rosbag for a given trial), the metric definitions, the participant inclusion/exclusion criteria, and a multi-user policy tuple of the same form as Section D.3.2. Two experiment configurations exist as of this writing: `mr_single.yaml` (attention-shifting measurement, two-user variant) and `mr_multi.yaml` (the three-user multi-MR rehabilitation experiment, which is the journal-paper target).

```
# mr_multi.yaml -- three-user multi-MR rehabilitation
experiment: mr_multi
title: "MR Multi --- Distributed rehabilitation with PHANToM
        teleoperation and multi-user AR"

design:
  factors:
    role:    [Patient, OnSiteClinician, RemoteClinician]
    network: [LAN, VPN]
  trials_per_condition: 6
  counterbalance: random_block_shuffle

recording:
  robot_id: ur16e_Q9
  preset:  mr_multi_pilot         # expands to a topic list
  output_prefix_template: "{experiment}_{participant}_t{trial:03d}_
  {role}_{network}"

timing:
  soft_max_trial_duration_sec: 20
  hard_max_trial_duration_sec: 90
  vpn_latency_budget_ms:       50

metrics:
  primary:   [patient_task_completion_time,
```

```
            clinician_intervention_count, coordination_error_rms]
  secondary: [N_shift, F_rms, network_rtt_ms]

multi_user_policy:
  see:     [VirtualGuide,  VirtualTwin,  VirtualTwin]
  move:    [VirtualGuide,  "",           ""]
  follow:  ["",            VirtualTwin,  VirtualTwin]
  owner:   [therapist,     patient,      remote_therapist]
  mapping: [phantom,       Q9,           Q9]
```

The three-participant policy resolves to: a therapist who drives the virtual guide via the PHANToM master, an on-site patient who sees the virtual twin of the real UR16e in their headset, and a remote clinician who also sees the virtual twin (but at a remote site, with the network factor switching between LAN-local and mesh VPN (e.g., Husarnet)). The recording preset `mr_multi_pilot` expands at runtime into the explicit topic list (`/ur16e_Q9/joint_states`, `/ur16e_Q9/current_pose`, `/ur16e_Q9/wrench`, `/phantom/pose`, `/phantom/state`); the expansion lives outside the experiment YAML so that the same preset can be reused across experiments.

D.5 User Profile and Permission Configuration

Each human participant in a session is described by a per-user profile YAML that specifies a stable user identifier, a role from a small fixed taxonomy, the assigned ROS server and Riley voice server, the streaming mode (local ZMQ or VPN-relayed), per-permission flags for safety-critical capabilities, and the QR identifiers the user is allowed to access. The profile binds individual humans to the multi-user policy tuple of Section D.3.2: when the policy tuple resolves `owner: therapist`, the framework looks up the profile of the participant who owns the `therapist` role and applies that profile's `can_remote_manipulation` permission before allowing the corresponding control path.

```
# users/_defaults.yaml (template overridden per user)
user_id: ""
display_name: ""
role: "user"       # admin | professor | researcher | user | developer
streaming_mode: "zmq_local"     # zmq_local | zmq_vpn
server:
  ros_server: ""
riley:
  server: "localhost"
permissions:
  can_remote_manipulation: false
  can_modify_config:       false
accessible_qr: []                # whitelist of QR identifiers
```

The framework's connection resolver consumes the profile to decide which Riley voice server, which streaming mode, and which permission set to apply; skills do not look up user information directly.

D.6 Device Portal Configuration

The device portal configuration documents the deployment endpoints used to communicate with HoloLens 2 and Desktop clients. The portal exposes a small set of REST endpoints (live MP4 stream, battery status, device identity) that the framework polls or subscribes to during a session, plus a per-device entry that maps a hardware identifier to its owner profile, IP address, and on-site versus remote location.

```
# device_config.yaml (representative excerpt)
device_portal:
  port: 443
  endpoints:
    live_stream:    "/api/holographic/stream/live.mp4"
    battery_status: "/api/power/battery"
unity_app:
  package_full_name: "com.MultiHRA.UnityClient_..."

streaming_presets:
  hololens2_default: {bitrate: 6Mbps, resolution: "1280x720", fps: 30}

devices:
  "EXAMPLE-HOLOLENS-1":
    owner: "example_admin"
    device_ip: "192.168.1.100"
    user_location: "on_site"
```

The device entry's `user_location` field (`on_site` or `remote`) is consumed alongside the multi-user policy tuple at session resolution time: if a participant is marked remote and the experiment's network factor is `VPN`, the framework routes their stream through the mesh VPN rather than through the LAN-local Mirror server.

D.7 Network and Communication Layer

This section consolidates the network configuration with the communication protocols that ride on top of it: ports, topic structures, the Zenoh cross-distribution bridge, the ROS TCP connector for Unity clients, and the ZeroMQ messaging used by the voice pipeline.

D.7.1 Port Assignments and Site Offsets

Network settings are centralized in a single YAML file that defines port assignments, VPN mesh hostnames, and SSH tunnel configurations for each deployment site. Table D.2 lists the internal port assignments. Every service container references these ports through environment variable expansion.

Table D.2: Internal port assignments defined in the centralized network configuration.

Port	Service	Protocol
10000	ROS2 TCP Endpoint (Unity ↔ ROS2)	TCP
8765	Foxglove WebSocket (visualization)	WS
9090	ROSBridge WebSocket	WS
5565	ZMQ Publish (Unity → Python)	ZMQ
5566	ZMQ Subscribe (Python → Unity)	ZMQ
5010	AI Server (VLM + Agent API)	HTTP
5011	RAG Server (document retrieval)	HTTP
5012	ChromaDB Vector Database	HTTP
5057	WebGUI Backend (ROS + POE + Gateway)	HTTP
5058	Scanner (SAM2/SAM3D, GPU)	HTTP
5059	POE Kinematics REST API	HTTP
5060	VLM Action Pipeline	HTTP
7000	Drake Meshcat Visualization	HTTP
7446	Zenoh Router (WAN)	TCP
7447	Zenoh Bridge Jazzy (Drake)	TCP
7448	Zenoh Bridge Humble (Robot)	TCP
11434	Ollama LLM API	HTTP

The site configuration maps each deployment location to a port offset and host aliases. For example, the ANL robot server uses offset 1 (adding 1 to each internal port for the nginx reverse proxy), while the ASU NCHR server uses offset 2. The localhost site uses offset 0 with direct connections. Each site entry specifies SSH usernames, IPv4 overrides (for environments where IPv6 is unavailable), and host aliases for robot and AI servers.

D.7.2 ROS2 Topic and Service Structure

A namespace-based topic structure is adopted where each robot instance publishes and subscribes under its own `/{robot_id}/` prefix. Table D.3 lists the standard topics.

Table D.3: Standard ROS2 topic structure per robot instance.

Topic	Description
`/{id}/joint_states`	Current joint positions (sensor)
`/{id}/joint_trajectory_controller/joint_trajectory`	Joint commands
`/{id}/target_pose`	Cartesian target from planner
`/{id}/planned_joint_states`	IK-computed joints (VirtualGuide)
`/{id}/current_pose`	Current end-effector pose
`/{id}/wrench`	Force/torque sensor readings

D.7.3 Zenoh Cross-Distribution Bridge

The Zenoh bridge is used to enable communication between ROS2 Jazzy (used by Drake simulation, domain ID 0) and ROS2 Humble (used by robot controllers,

domain ID 1). The two bridge instances operate in peer mode: the Jazzy bridge listens on port 7447 and connects to the Humble bridge on port 7448, and vice versa. An optional Zenoh router on port 7446 provides WAN-level connectivity for multi-site deployment.

The bridge configuration uses explicit allow lists to control which ROS2 interfaces are bridged. The Jazzy-side allows all Drake services (`/drake/.*`) and standard parameter services. The Humble-side allows the motion interface services (`execute_motion`, `execute_pattern`, `move_to_named_position`) defined in the `hri_msgs` package. This selective bridging reduces unnecessary cross-distribution traffic to only the interfaces required for simulation–hardware synchronization.

D.7.4 ROS TCP Connector

Unity applications on HoloLens 2 communicate with ROS2 through the ROS TCP Connector, which serializes ROS2 messages for transmission over TCP sockets on port 10000. This bridge supports bidirectional topic publishing and service calls.

D.7.5 ZeroMQ Messaging

The RILEY voice system and Unity application communicate through ZeroMQ PUB/SUB sockets (ports 5565/5566). This decoupled messaging pattern supports asynchronous communication without the overhead of ROS2's DDS middleware.

D.7.6 Adaptive Network Routing

Head-mounted displays such as Microsoft HoloLens run restricted operating systems that cannot install VPN software, yet they must connect to ROS2 services on protected networks (Figures D.1 and D.2); the multi-site nature of the system—spanning geographically separated laboratories—further complicates network configuration.

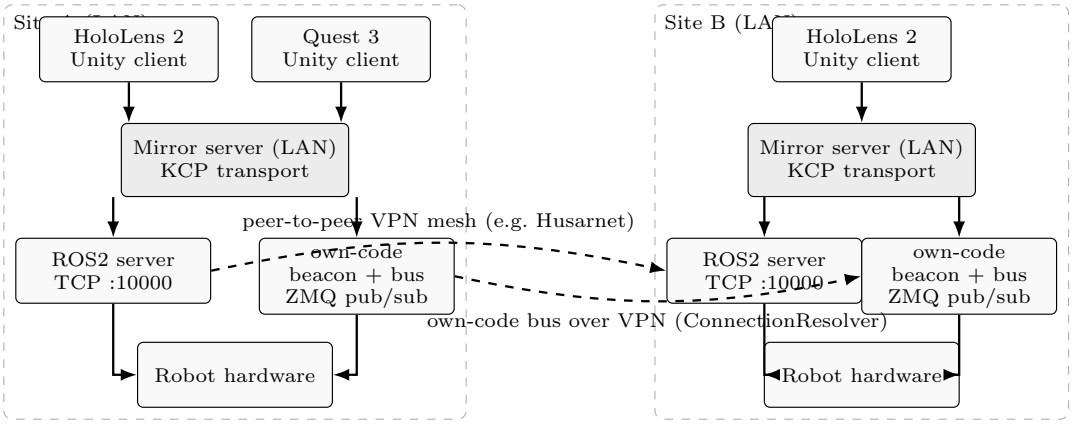


Figure D.1: Multi-site network architecture. Clients connect to a LAN-local Mirror server providing authoritative state replication over KCP transport. A peer-to-peer VPN mesh connects on-site infrastructure across laboratories; ROS2 servers, AI services, and robot hardware communicate through ROS TCP and ZMQ protocols while HoloLens and Quest clients join the Mirror session directly on the local subnet.

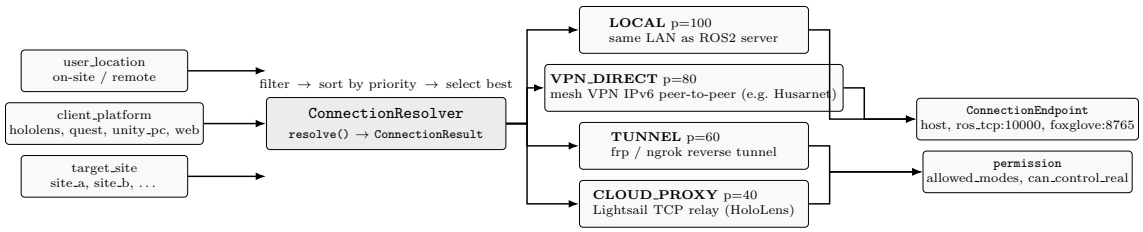


Figure D.2: ConnectionResolver architecture: adaptive routing based on user location, client platform, and target site.

The ConnectionResolver component selects a connection method from three inputs: user location (on-site or remote), client platform (HoloLens, Unity PC, or web browser), and target site. Four methods are evaluated in priority order: local connection (priority 100, minimal latency on the same network), VPN direct (priority 80, peer-to-peer VPN for VPN-capable devices), tunnel (priority 60, SSH tunnel for VPN-blocked environments), and cloud proxy (priority 40, relay through a cloud server for devices without VPN or tunnel). The resolver returns the endpoint and the permissions associated with that method, with cloud proxy connections carrying restricted permissions due to their additional latency and security considerations. Platform-aware routing matches the constraints: Unity desktop applications establish direct VPN connections; web browsers use WebSocket through the cloud proxy; HoloLens devices use the cloud proxy due to operating-system restrictions but benefit from optimized binary protocols. The single resolution endpoint lets clients report platform and target while the server selects the method, so the network infrastructure evolves without client updates. The resolver is also the centre of the multi-site

architecture of Chapter 11: it resolves abstract site/robot identifiers to concrete VPN-mesh peer IPs, so the own-code beacon publishers and the cross-site message bus address peers by logical name rather than by VPN-assigned IP.

D.7.7 SSH Gateway and Device Discovery

The SSH Gateway container provides automatic device discovery and secure remote access to ROS2 servers across multiple sites (Figures D.3 and D.4). The HoloLens Scanner reads the Address Resolution Protocol (ARP) table (`/proc/net/arp`) to identify active devices on the local network, attempts HTTPS connection to port 443 for each candidate IP, checks for the Microsoft-HTTPAPI signature that identifies HoloLens Device Portal, authenticates with pre-configured credentials, and queries `/api/os/machinename` to retrieve the device hostname; the procedure eliminates manual IP configuration when HoloLens devices connect to the network. The VPN Hostname Resolver translates human-readable hostnames (e.g., `site-a-robot-server`) to network addresses within the VPN mesh, which lets client applications reference servers by logical name rather than numeric address. The SSH Client executes commands on remote ROS2 servers, and per-site key files let the gateway respect site-specific security policies through a unified API.

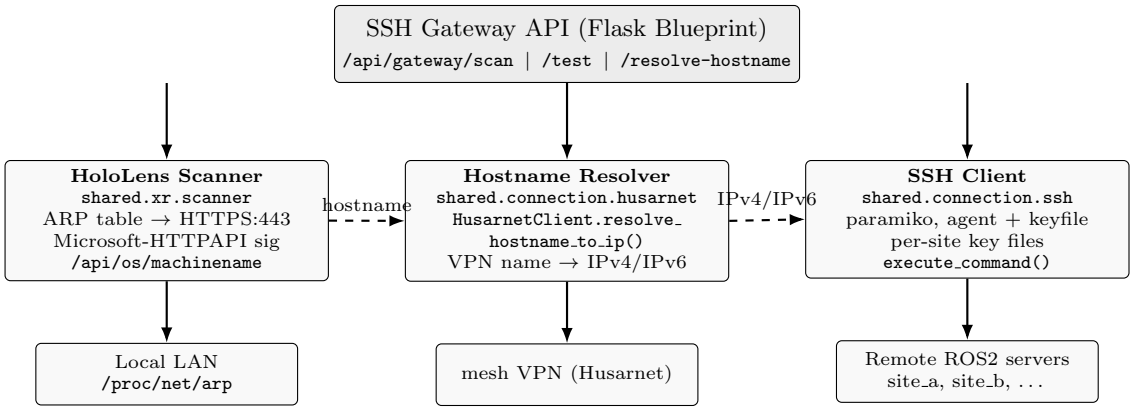


Figure D.3: SSH Gateway architecture. The HoloLens scanner discovers local network devices, the VPN hostname resolver translates logical names to addresses, and the SSH client executes secure remote commands across distributed sites.

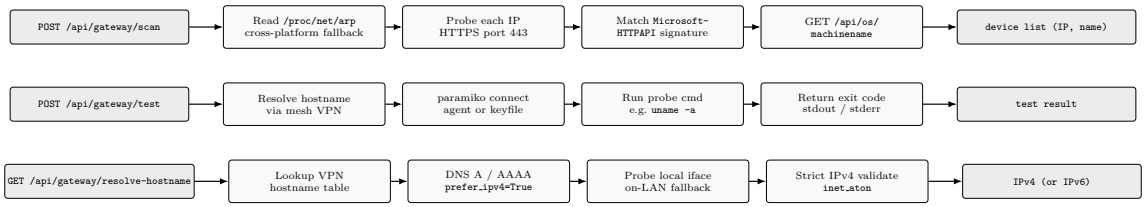


Figure D.4: SSH Gateway API flow: HoloLens device scanning, SSH connection testing, and VPN hostname resolution.

D.7.8 Device Authentication

Secure API access for cloud-deployed services is constrained on head-mounted displays where text entry is cumbersome. The system uses device-based JSON Web Token (JWT) authentication that grants access without a user login interface (Figure D.5). Each device is pre-registered with a unique device identifier and device secret; on connection, the device presents these credentials to the authentication endpoint, the server validates them, and a 24-hour JWT is issued. Subsequent API requests carry the JWT in the Authorization header, and the server validates the signature and expiration before processing the request with the device's associated permissions. Device permissions are configured in the database to express fine-grained control over which devices access which functions. Unauthenticated requests receive filtered responses rather than outright rejection: sensitive fields such as VPN addresses and internal network configurations are stripped, while basic functionality remains available for development and testing without full authentication. Two-factor authentication via Time-based One-Time Passwords (TOTP) is optionally available for high-security deployments where device theft is a concern.

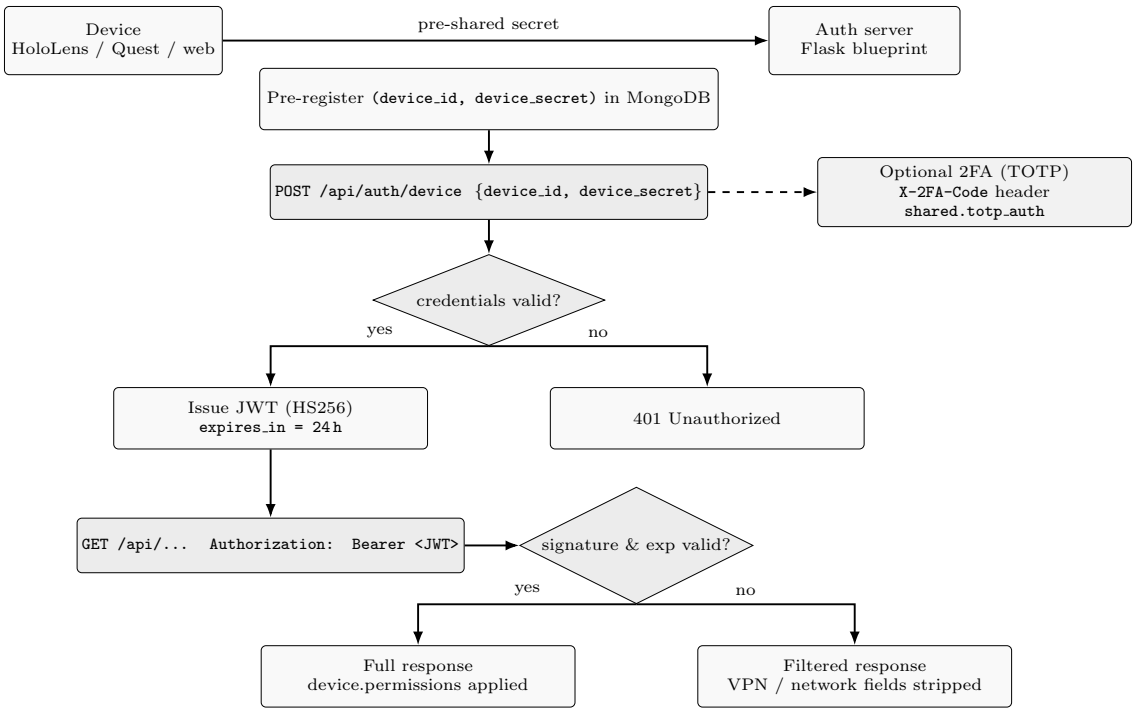


Figure D.5: Device authentication flow: JWT token acquisition and API access with graceful degradation for unauthenticated requests.

D.7.9 Mirror Webhook for Web Integration

Clinical deployment requires monitoring and management interfaces accessible from standard web browsers (Figure D.6). The Mirror server forwards room events to a configured webhook endpoint on the Flask server: Join and Leave events track player presence in sessions, RoomCreate and RoomClose events manage session lifecycle, and custom events convey application-specific information such as robot state changes, error conditions, and milestone completions. The web panel displays real-time session information including active rooms, connected players, and recent events, which lets therapists monitor multiple patients without context-switching between head-mounted displays. Webhook delivery is not guaranteed—network issues or server downtime may cause missed events—so Unity and HoloLens clients continue operating normally regardless of webhook status, and the web panel displays stale-data indicators when webhook events have not been received recently.

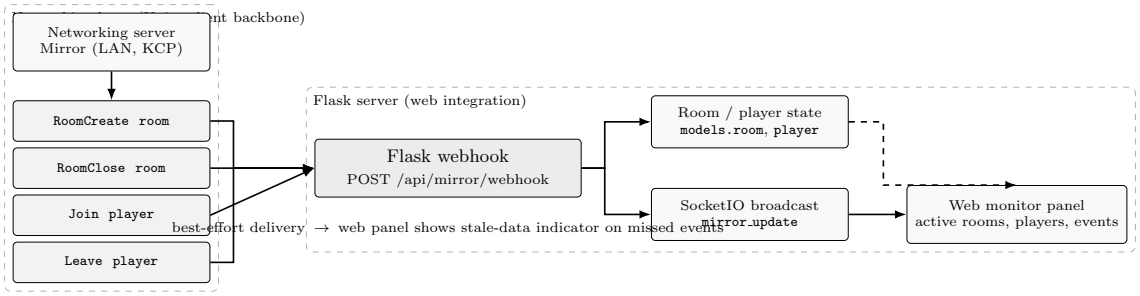


Figure D.6: Mirror server event integration for real-time web monitoring of MR session events.

D.8 Docker Service Architecture

The system is deployed as a collection of Docker containers, each encapsulating a specific functional domain. Table D.4 lists the service groups and their constituent containers.

Table D.4: Docker service architecture organized by functional domain.

Domain	Service	Function
AI	ai-server	VLM + Multi-Agent API (port 5010)
	rag-server	RAG document retrieval (port 5011)
	chromadb	Vector database (port 5012)
	ollama	Local LLM inference (port 11434)
Voice	stt-wakeword	Wake word detection + STT (ZMQ 5555)
	nlu-router	Intent classification (ZMQ 5556)
Scanner	scanner	SAM2/SAM3D GPU pipeline (port 5058)
ROS2	zenoh-bridge-jazzy	Drake-side Zenoh (port 7447)
	zenoh-bridge-humble	Robot-side Zenoh (port 7448)
	zenoh-router	WAN router (port 7446)

All containers share configuration through volume mounts to the centralized YAML directory, maintaining the single source of truth principle. Service-specific environment variables are defined in deployment-specific `.env` files, and port assignments reference the values listed in Table D.2.

D.9 Unified Motion Interface

A set of ROS2 message, service, and action definitions provides a robot-agnostic interface for motion planning and control. These definitions are collected in the

`hri_msgs` package and are used consistently across all robot platforms; the analytical solver behind the kinematics services is documented in Appendix B.

D.9.1 Services

Table D.5: Motion interface services.

Service	Description
MotionPlan	Unified planning with pluggable backends (Linear, Cubic, Quintic, Trapezoidal, Drake, MoveIt, OMPL, STOMP, DMP)
ValidateMotion	Trajectory validation against joint/velocity/acceleration limits and collisions (bitmask flags)
ExecuteMotion	Trajectory execution on robot controller
ExecutePattern	Parameterized pattern execution (circle, diamond, spiral, square) with preview mode
MoveToNamedPosition	Named position move (home, work, extended) with velocity scaling
GenerateTrajectory	Trajectory interpolation between configurations with optional collision checking
StartTeleoperation	Bilateral teleoperation session with configurable mode and force scaling

D.9.2 Actions

Table D.6: Motion interface actions (long-running operations with feedback).

Action	Description
ImpedanceControl	Cartesian/joint impedance control with stiffness and damping parameters
AdmittanceControl	Force-input position-output compliant control
BilateralTeleop	Leader-follower teleoperation with force reflection
ExecutePattern	Pattern execution with real-time progress feedback
ReachingTask	Goal-directed reaching with configurable constraints
GeneralTask	Generic task execution from YAML task definitions

The `MotionPlan` service supports goal specification by joint angles, Cartesian pose, or named configuration. The planner type is selected at request time from basic interpolators (linear, cubic, quintic, trapezoidal), physics-based planners (Drake, MoveIt, OMPL, STOMP), or learning-based planners (DMP, imitation, policy). The service returns the trajectory along with the actual planner used, which may differ from the requested planner if the preferred backend is unavailable.

The `hri_msgs` package also defines specialized message types for compliant control:

- `AdmittanceParams`: virtual mass, damping, and stiffness parameters (both diagonal and full 6×6 matrix forms) with velocity and force safety limits.

- `TeleopState`: bilateral teleoperation state including pose increments, wrench feedback, and three-level clutch states.
- `VirtualFixture`: geometric constraints (plane, cylinder, box, sphere, or mesh) with configurable compliance and constraint modes (do-not-penetrate, do-not-escape, or guidance).
- `HapticFeedback`: combined force/torque feedback from multiple sources (F/T sensor, virtual fixtures, joint limits, singularity, collision) with per-source gain control.

The solver backend for kinematics services is selected through the configuration system. Available backends include the analytical Product of Exponentials solver (Appendix B), KDL, TRAC-IK, and Drake's built-in solver.

APPENDIX E

AI COMPONENT REFERENCE

This appendix documents the implementation details of the multi-agent AI framework described in Chapter 10. The framework is organized as six layers in data-flow order: (i) the vision-language model (VLM) backend that produces grounded perception, (ii) the voice pipeline that converts spoken input into structured commands, (iii) the registered voice command set that the MR application accepts at the user-facing layer (a 50-command vocabulary spanning robot setup, demo session control, BT contract execution, partner-pattern diagnosis/recovery/sovereignty markers, and PHANToM teleoperation), (iv) the message-skills broker (a two-layer Zenoh + ZMQ bus) that carries voice traffic, agent responses, and skill RPC across services, (v) the multi-provider language-model abstraction that lets each agent select from local Ollama or cloud-hosted Claude, Codex, or Gemini at configuration time, and (vi) the retrieval-augmented generation and dual memory subsystem. The eight agent profiles (Section E.7) bind these layers through YAML configuration, defining the role, services, language-model backend, and memory store of each agent.

E.1 VLM Backend Configuration

A multi-backend vision-language model (VLM) handler is implemented that supports runtime switching between inference providers. Table E.1 summarizes the available backends and their resource requirements.

Table E.1: VLM backend comparison from deployment configuration.

Backend	Model	Deployment	Grounding
Ollama	(environment variable)	Local, port 11434	No
HF API	nvidia/Eagle2-9B	Cloud (HuggingFace)	Yes (InternVL)
HF Local	(environment variable)	Local GPU	Yes (InternVL)

All backends share common inference parameters: temperature 0.1, max tokens 512, and timeout 120 s (60 s for the HuggingFace API). The active backend is selected through the `VLM_BACKEND` environment variable, allowing runtime switching without configuration file changes. Grounding-capable backends return object bounding boxes in a normalized $[0, 1000]$ coordinate range using the InternVL format.

```
# VLM configuration (from vlm_config.yaml)
vlm:
  active_backend: ${VLM_BACKEND}
  backends:
    ollama:
      endpoint: ${OLLAMA_ENDPOINT}
      model: ${OLLAMA_MODEL}
      port: 11434
```

```
    temperature: 0.1
    max_tokens: 512
    supports_grounding: false
  huggingface_api:
    endpoint: https://api-inference.huggingface.co
    model: nvidia/Eagle2-9B
    supports_grounding: true
grounding:
  coordinate_range: 1000
```

E.2 Voice Pipeline Specifications

The RILEY voice system processes audio input through four sequential stages: wake word detection, speech-to-text (STT), natural language understanding (NLU), and action dispatch. The full structured command catalog (both the commands produced by this pipeline and those registered directly in the MR application) is documented in Section E.3.

E.2.1 Wake Word Detection

Mozilla Precise serves as the wake word engine with the trigger word "RILEY." The detector runs continuously on the audio input stream and activates the STT stage upon detection.

E.2.2 Speech-to-Text

Vosk is used for offline speech recognition, supporting both English and Korean language models. The Riley agent configuration specifies Whisper (base model) as an alternative STT backend with automatic language detection.

E.2.3 Natural Language Understanding

The transcribed text is processed through a local Ollama LLM (`llama3.2`) to extract structured commands. The NLU router configuration specifies a temperature of 0.3, top_p of 0.9, and top_k of 40, with a 5-second timeout and 2 retry attempts. The model is constrained to produce JSON output matching a predefined action schema, with schema validation enforced by the router. A confidence threshold of 0.7 filters low-confidence intent classifications. The conversation history is limited to the most recent 5 exchanges to maintain context without exceeding the model's context window.

E.3 Voice Command Reference

The Riley voice system dispatches a structured 50-command vocabulary registered in the MR application's voice recognition subsystem. Each command is encoded as a `!command:<token>` marker that the LLM appends at the end of its response; Unity's `LLMManager` parses the marker and triggers the corresponding handler. Commands

are organized by functional category in Table E.2; the 50-command count is the value reported in the SSOT file `config/ai/agents/types/riley_unity_commands.yaml`.

Table E.2: Riley voice command reference, organized by functional category. The right-hand column lists the `!command:<token>` identifiers as they appear in the YAML SSOT.

Category (count)	**Commands**
Robot operation	
Robot Setup (5)	spawn_robot, test_robot, setup_robot_base, stop_robot, start_robot
Robot Navigation (2)	go_to_work, go_to_home
Demo Sessions (5)	start_demo_1, start_demo_2, start_demo_3, start_experiment, stop_experiment
Pattern Execution (5)	run_diamond, run_circle, run_square, run_figure8, run_spiral
Behavior-tree contracts	
BT Contract Execution (8)	list_bt_scenarios, run_bt_cart_mr, run_bt_cart_mr_take, run_bt_move_to_work, run_bt_rps_demo, run_bt_pattern_viz, run_bt_bag_to_mp4, cancel_bt_run
Partner pattern	
Diagnosis (2)	diagnose_current_run, bt_progress_summary
Recovery (3)	restart_bt_run, run_bt_urcap_restart, bt_doctor
Sovereignty (2)	mark_outcome_good, mark_outcome_discard
Meta (1)	graceful_shutdown_session
Mixed-reality & spatial	
PHANToM Haptic (3)	enable_phantom_teleop, disable_phantom_teleop, phantom_check
QR & Spatial (5)	start_qr_code, stop_qr_code, share_qr, rescan_qr, simulate_qr
Visualization (4)	show_virtual_twin, hide_virtual_twin, show_riley, hide_riley
Riley Avatar (3)	spawn_riley, despawn_riley, riley_wake
Session control	
Dialog Confirmation (2)	"yes," "no"
General (5)	save_logs, toggle_ui, enable_zeromq, disable_zeromq

The dispatch protocol distinguishes two execution modes. *Two-turn confirmation* commands—`spawn_robot`, `start_qr_code`, `setup_robot_base`, `start_experiment`, `stop_experiment`, `despawn_riley`, `enable_phantom_teleop`, `disable_phantom_teleop`, and the three partner-pattern destructive recoveries (`restart_bt_run`, `run_bt_urcap_restart`, `graceful_shutdown_session`)—require explicit confirmation before the LLM emits the `!command:` marker. Affirmative responses are accepted in either supported language (English or Korean). *Direct*

execution applies to all other commands and to any direct restatement of the command name.

The partner-pattern category enforces a diagnosis-first recovery flow through the prompt template: when the wearer reports a problem, the LLM is instructed to issue `!command:diagnose_current_run` *before* proposing a recovery action, so that the wearer hears the root cause in plain language (e.g., "URCap process died 8 seconds ago, likely pendant e-stop") before deciding whether to restart. Outcome markers (`mark_outcome_good`, `mark_outcome_discard`) are immediate: the wearer's voice itself is the decision, captured the moment it is spoken without a second-turn confirmation. Each Partner-Pattern voice command pairs with a CLI verb on the robot-server side (`bt-bringup`, `bt-take`, `bt-events`, `bt-beacon`); the voice-CLI pairing is documented in the engineering archive of the multi-HRA project.

E.4 Message-Skills Broker

The message-skills broker is the runtime substrate that carries voice traffic, agent responses, and skill RPC across the framework's services. It is implemented as a two-layer bus: a Zenoh layer that handles default discovery, lifecycle, and metadata broadcast across all channels, and a ZeroMQ XSUB/XPUB broker layer that carries low-latency voice and Riley response data on dedicated channels. The two layers are bridged unidirectionally where lifecycle events from Zenoh inform ZMQ-resident services; ZMQ remains active for the voice pipeline (rather than deprecated) because it provides the deterministic latency that Whisper STT and TTS playback require. Table E.3 summarizes the responsibilities of the two layers.

Table E.3: Two-layer architecture of the message-skills broker. Both layers run concurrently; the lifecycle-only Zenoh-to-ZMQ bridge keeps cross-transport channels in sync.

Layer	Role	Channels and Notes
Zenoh	Default bus driver, discovery, lifecycle, metadata broadcast	All non-voice channels by default; cross-host federation.
ZMQ XSUB/XPUB	Specific data lane for voice and Riley response	`riley-voice`, `own-code-relay`; deterministic latency.

The broker exposes a small REST API for service discovery and out-of-band publish: `GET /health` (broker liveness), `GET /channels` (channel topology), `GET /channels/status` (per-channel subscriber count and last-publish timestamp), `POST /publish` (publish to a channel), `POST /publish/riley-response` (specialized Riley response path with TTS metadata), and `POST /voice/transcribe` (Whisper STT relay, used by the voice pipeline of Section E.2 above). The channel topology is configured through `config/channels.yaml` (in the message-skills service directory), where each entry specifies the channel name, the default transport, and the consumer service URLs. The broker thereby serves as the single integration point for the voice pipeline (Riley client → STT relay → NLU router), the agent platform (own-code

agent → Riley response), and the skill services (vision, robotics, Jupyter, etc.) that publish status into the bus.

E.5 Multi-Provider Language Model Backend

In addition to the four VLM backends documented in Section 1, the framework includes a hand-built language-model client hierarchy in own-code (`lib/clients/`) that lets each agent profile select from six providers at runtime. The hierarchy is rooted at the abstract base class `BaseLLMClient`, with six concrete subclasses: `OllamaClient`, `OpenAIClient`, `ClaudeAgentClient` (Claude Python SDK, multi-turn session), `ClaudeCliClient` (Claude CLI subprocess, one-shot), and two CLI subclasses that extend `ClaudeCliClient`—`CodexCliClient` and `GeminiCliClient`. The selection is performed by an `agent_manager` module that instantiates the provider-specific subclass; agents are otherwise written against the provider-agnostic `BaseLLMClient` interface. Each subclass implements an asynchronous `from_config()` factory that consumes the provider block of the agent's YAML profile and returns a configured client. Table E.4 summarizes the providers.

Table E.4: Language-model providers in the own-code `BaseLLMClient` hierarchy. The `provider` field in the agent's YAML profile selects the subclass at construction time. `CodexCliClient` and `GeminiCliClient` inherit subprocess-handling from `ClaudeCliClient`.

Subclass	Deployment	Use case
`OllamaClient`	Local Ollama daemon (port 11434)	Default offline backend for Riley, Coordinator, Robot Operator, GR00T.
`OpenAIClient`	OpenAI API (HTTP)	Direct API access.
`ClaudeAgentClient`	Claude Python SDK (multi-turn session)	Long-running orchestration with tool use.
`ClaudeCliClient`	Claude CLI subprocess (one-shot)	Single-turn analytical prompts.
`CodexCliClient`	Codex CLI subprocess (extends ClaudeCli)	Code-generation tasks.
`GeminiCliClient`	Gemini CLI subprocess (extends ClaudeCli)	Long-context analysis.

The same dependency-injection pattern is used for tool registration: a separate `ToolRegistry` loads tools from a list of skill directories merged with the agent's profile-level `extra_tools` list. The Riley agent uses a delegation pattern: it loads only the message-skills tools directly and delegates office, robotics, vision, and Jupyter tool calls to the corresponding specialized agents through the message-skills bus. The pattern keeps the Riley prompt small while preserving access to the full tool catalog.

E.6 RAG and Memory System

E.6.1 Retrieval-Augmented Generation

ChromaDB is integrated as the vector database for document retrieval. Clinical protocols, exercise guidelines, and patient history documents are embedded using

the sentence-transformers library and stored as vector collections. Documents are split into chunks of 512 tokens with 50 tokens of overlap. At query time, the system retrieves the top-5 most similar chunks (similarity threshold ≥ 0.7) and prepends them to the LLM context. The RAG server uses `llama3.1:8b-instruct-q8_0` through the Ollama backend (timeout 300 s), with Gemini 2.0 Flash available as a secondary provider (timeout 180 s).

E.6.2 Dual Memory Architecture

A two-tier memory system is implemented. Short-term memory stores session-specific context (detected objects, planned trajectories, approval states) in ChromaDB. Long-term memory stores cross-session knowledge (patient preferences, successful protocols) in a Neo4j graph database. A transfer mechanism promotes frequently accessed short-term memories to long-term storage based on a configurable access frequency threshold.

E.7 Agent Configuration Profiles

Each AI agent operates according to a YAML profile that specifies its role, available services, language-model provider, and memory backend, binding the perception (Section 1), voice (Sections 2–3), broker (Section E.4), language-model (Section E.5), and memory (Section 6) layers above into a deployable unit. Table E.5 summarizes the eight agent profiles. The numbers in parentheses under Memory indicate the short-term memory limit (maximum stored items).

Table E.5: Multi-agent profiles from the YAML configuration directory.

Agent	Role	Services	Memory
VLM Agent	Visual Perception Expert	memory, rag	ChromaDB (100)
Riley	Voice AI Assistant	voice, conversation, memory	ChromaDB (50)
Robot Operator	Robot Execution Expert	memory, action	ChromaDB (50)
Discussion Agent	Multi-Agent Facilitator	memory, rag, conversation, text_analyzer	Neo4j (200)
Coordinator	Multi-Agent Orchestrator	memory	ChromaDB (100)
Simple Agent	Basic Assistant	memory	ChromaDB (20)
GR00T Agent	VLA Pipeline Operator	memory, tools (groot bridge), msg_skills	ChromaDB (50)
PlayMaker Agent	MR Game Companion (delegation target)	memory, tools, msg_skills	ChromaDB (50)

The following listing shows representative agent profiles:

```
# Riley Agent profile (from riley_agent.yaml)
agent:
  name: Riley
  role: "Voice AI Assistant"
  services: [voice, conversation, memory]
```

```
  stt:
    backend: whisper
    model: base
    language: auto
  llm:
    backend: ollama
    model: "llama3.2:3b"
    temperature: 0.7
    max_tokens: 256
    timeout: 30
  memory:
    backend: chromadb
    short_term_limit: 50

# Robot Operator profile (from robot_operator.yaml)
agent:
  name: Robot Operator
  role: "Robot Execution Expert"
  services: [memory, action]
  approval_required: true
  default_approval_level: 2  # APPROVAL
  memory:
    backend: chromadb
    short_term_limit: 50

# Discussion Agent profile (from discussion_agent.yaml)
agent:
  name: Discussion Agent
  role: "Multi-Agent Discussion Facilitator"
  services: [memory, rag, conversation, text_analyzer]
  memory:
    backend: neo4j
    short_term_limit: 200
    long_term_threshold: 0.7
  rag:
    top_k: 10
  conversation:
    strategy: "all"
    max_rounds: 10
    summarize_frequency: 5

# GR00T Agent profile (from groot_agent.yaml)
groot_agent:
  name: "GR00T"
  role: "VLA Pipeline Operator"
  services:
    - memory    # ChromaDB short-term recall
    - tools     # groot bridge API (analyze, validate, pipeline_step)
```

```
  - msg_skills    # ZMQ messaging (Unity to own-code)
service_config:
  memory:     {backend: chromadb, short_term_limit: 50}
  msg_skills: {channel: groot_pipeline,
               response_type: GROOT_RESPONSE}
llm:
  backend: ollama
  model: gemma3:12b
  temperature: 0.3        # safety-related: low temperature
  max_tokens: 2048        # detailed execution plans
  timeout: 60
collaboration:
  can_initiate: true
  can_respond_to: [vlm_agent, robot_operator, coordinator, riley]
  primary_outputs: [task_type, safety_result, execution_plan,
                    action_chunk]
```

The GR00T agent is the VLA (Vision-Language-Action) pipeline operator. It classifies natural-language commands into task types (inspect, scan, move, manipulate, report), coordinates VLM scene analysis with action-policy generation, and emits structured execution plans through the GR00T-bridge tool API. A four-level safety shield (V1–V4) is enforced before plan emission, with the agent profile pinning the shield to "approval required" for all manipulation tasks. The C++ core that backs the safety check, trajectory smoothing, and image preprocessing is exposed through pybind11 bindings. The GR00T agent is therefore one of the framework's two paths from natural-language input to physical robot motion, the other being the partner-pattern voice commands of Section E.3 above.

The Human-in-the-Loop level is configurable per agent and per session. The four levels—AUTO (0), NOTIFY (1), APPROVAL (2), and MANUAL (3)—determine the degree of human oversight required before the robot operator (or GR00T agent, when acting in execution mode) executes a planned action.

APPENDIX F

3D RECONSTRUCTION PIPELINE DETAILS

This appendix documents the implementation of the click-to-mesh 3D reconstruction pipeline described in Chapter 8. The flow follows the data path of the pipeline itself: (i) SAM2 segmentation produces a mask from user clicks, (ii) SAM3D consumes the masked image and produces a textured GLB mesh together with a Gaussian Splatting PLY, (iii) the web-based Gaussian Splatting viewer renders that PLY, (iv) a global coordinate convention maps reconstructed geometry into the Unity scene, and (v) ICP registration aligns the reconstructed mesh against a known CAD model when one is available.

F.1 SAM2 Segmentation Server

The Meta Segment Anything Model 2 (SAM2) is deployed as an HTTP server with GPU acceleration. The server accepts RGB images and user-specified click points, and returns binary segmentation masks.

F.1.1 API Reference

Table F.1: SAM2 server API endpoints.

Method	Endpoint	Description
POST	`/api/segment`	Submit image with click coordinates, returns mask
GET	`/api/images`	List available images in the session
GET	`/api/masks`	Retrieve generated masks for a session
POST	`/api/session`	Create or reset a segmentation session

The segmentation request includes the image, a list of positive click points (foreground), and optional negative click points (background). The server processes these inputs through the SAM2 encoder and mask decoder, returning a binary mask at the original image resolution.

```
# Segmentation request format
{
  "image_id": "capture_001",
  "points": [
    {"x": 320, "y": 240, "label": 1},   # positive (foreground)
    {"x": 100, "y": 100, "label": 0}    # negative (background)
  ],
  "multimask_output": false
}
```

F.1.2 GPU Configuration

The SAM2 server requires a CUDA-capable GPU. The Docker container is deployed on port 5058 with GPU passthrough. Actual inference latency and VRAM consumption depend on the SAM2 model checkpoint size and input image resolution; these should be profiled for each deployment GPU.

F.2 SAM3D Mesh Generation

The SAM3D module generates 3D meshes from 2D images through the HuggingFace Spaces API (HorizonRobotics/EmbodiedGen-Image-to-3D). When local depth data is available, the system also generates meshes from RGB-D point clouds through a local reconstruction pipeline. Both paths emit two output formats: a textured GLB mesh consumed by Unity and Blender, and a Gaussian Splatting PLY consumed by the Rerun viewer of the next section.

F.2.1 Processing Pipeline

The HuggingFace API pipeline proceeds in three stages:

1. **Preprocessing**: The input image is segmented using the SAM2 mask (if available) or the built-in background removal (`rembg`). If a mask is provided, it is applied to produce an RGBA image with the target object isolated.

2. **3D Generation**: The preprocessed image is submitted to the EmbodiedGen model with the following parameters: structured sampling steps = 25, spatially-latent sampling steps = 25, structured guidance strength = 7.5, and spatially-latent guidance strength = 3.0. A fixed random seed (default 42) ensures reproducibility.

3. **Representation Extraction**: The model outputs are extracted as two formats: a textured GLB mesh (texture size 2048×2048 with delighting enabled) and a Gaussian Splatting PLY file.

For the local RGB-D pipeline, the processing follows four stages:

1. **Mask application**: The binary mask from SAM2 is applied to the depth image, isolating the target object's depth values.

2. **Point cloud generation**: The masked depth pixels are projected into 3D space using the camera intrinsic matrix. Points with invalid or out-of-range depth values are discarded.

3. **Surface reconstruction**: Poisson surface reconstruction (Open3D implementation) is applied to generate a watertight mesh from the oriented point cloud.

4. **Texture mapping**: The original RGB colors are projected onto the mesh vertices using the camera model.

F.2.2 Output Formats

The system generates two output formats:

- **GLB (GL Transmission Format Binary)**: A textured triangle mesh suitable for import into Unity, Blender, or web-based 3D viewers. The GLB file includes vertex positions, normals, UV coordinates, and a texture atlas.
- **PLY (Polygon File Format)**: A point-based representation used for Gaussian Splatting rendering. Each point stores position, spherical harmonics coefficients (`f_dc`), opacity, scale (3D), and rotation (quaternion).

F.3 Rerun-Based 3D Visualization

The reconstructed meshes, point clouds, and Gaussian-splat outputs of the previous section are inspected through a Rerun viewer embedded in the WebGUI scanning panel. Rerun is run as a separate service exposing a gRPC ingestion endpoint on port 9876 and a web-rendered viewer on port 9877; the WebGUI's `panel-scanning.js` embeds the web viewer through an iframe so that the operator sees RGB, depth, point clouds, and registered meshes in a single 3D scene. The reconstruction handlers stream their outputs into Rerun through the gRPC endpoint as they produce them, so the scanning workflow is incrementally rendered rather than reloaded at the end.

Among the reconstruction outputs, the Gaussian-splat PLY file stores the following per-splat attributes:

Table F.2: Gaussian Splatting PLY attributes per splat.

Attribute	Type	Description
x, y, z	float32	Splat center position
f_dc_0/1/2	float32	Spherical harmonics (DC band, RGB)
opacity	float32	Splat opacity (sigmoid-activated)
scale_0/1/2	float32	Splat scale in 3 axes (log-space)
rot_0/1/2/3	float32	Splat rotation quaternion

F.4 Coordinate Conventions

The reconstruction pipeline operates in the camera coordinate system (Z-forward, Y-down), while Unity uses a left-handed Y-up convention. Both the GLB mesh path and the ICP alignment of the next section apply the following coordinate transformation to convert from camera space to Unity space, with $M = \mathrm{diag}(1, -1, 1)$:

$$\mathbf{p}_{\text{Unity}} = M\,\mathbf{p}_{\text{camera}}, \qquad R_{\text{Unity}} = M\,R_{\text{camera}}\,M^{-1}. \tag{F.1}$$

The point form on the left applies to vertex coordinates (meshes, point clouds, Gaussian-splat centres). The pose form on the right is the similarity transform required for the rotation block of any 4×4 pose, including the ICP output of the next section, so that orientation transforms consistently with handedness.

F.5 ICP Alignment

The Iterative Closest Point (ICP) algorithm registers reconstructed meshes against known CAD models. This alignment determines the 6-DoF pose of known objects in the robot's workspace.

F.5.1 Two-Stage Registration

The ICP handler is configured with a voxel size of 0.01 m for downsampling, 10,000 sampled points per mesh, and a maximum of 200 iterations. The two-stage process operates as follows:

1. **Global registration**: RANSAC-based global registration is first applied using FPFH (Fast Point Feature Histograms) descriptors. The FPFH features are computed with a search radius of 5× voxel size and up to 100 nearest neighbors. The RANSAC correspondence distance threshold is 1.5× voxel size (0.015 m), with edge length consistency checking (ratio 0.9) and a convergence criterion of 100,000 iterations at confidence 0.999.

2. **Local refinement**: The global alignment is then refined using point-to-plane ICP with a correspondence distance threshold of 0.4× voxel size (0.004 m). Normals are estimated with a search radius of 2× voxel size and 30 nearest neighbors.

F.5.2 Alignment Output

The alignment result is stored as a 4×4 homogeneous transformation matrix in JSON format, expressed in the Unity coordinate convention defined in Section F.4 (Eq. F.1). The Unity application loads this matrix to position the CAD model at the estimated object pose.

APPENDIX G

ROBOTIC SKIN COLLABORATION WITH SEOUL NATIONAL UNIVERSITY

This appendix describes the SNU robotic skin collaboration that reused the same Sawyer platform and ROS infrastructure as the DMMRP of Chapter 8. Lee et al. at Seoul National University developed the modular fiber-optic robotic skin and the Fiber Bragg Grating (FBG) sensing principle (Lee *et al.*, 2024b); the author's contribution was the software architecture that integrated the skin into the Sawyer ROS stack, the joint-space admittance controller that consumed the skin's force estimate, and the obstacle-avoidance demonstration that the integrated system supported. The skin module characterization, the FBG demodulation, and the materials and beam structure are reported in (Lee *et al.*, 2024b) and are not repeated here.

G.1 Skin Module and Sensing

The robotic skin module embeds Fiber Bragg Grating sensors in a custom triangular beam structure (Lee *et al.*, 2024b). Each module provides three degrees of freedom in sensing: contact force magnitude and two-axis contact localization. Reported characterization results from (Lee *et al.*, 2024b) are 1.45 N for force estimation and 1.85/1.91 mm for horizontal/vertical contact localization.

The skin's ROS interface streams the FBG-derived contact force as a wrench topic at 100 Hz, which the controller below consumes alongside the Sawyer's joint state at 1 kHz.

G.2 Admittance Control with the SNU Force Sensor

The author derived a joint-space admittance controller for the Sawyer that consumes the skin's contact wrench as the external input (Lee *et al.*, 2024b). The desired second-order joint dynamics are

$$I_d\,(\ddot{\Theta} - \ddot{\Theta}_e) + B_d\,(\dot{\Theta} - \dot{\Theta}_e) + K_d\,(\Theta - \Theta_e) \;=\; \tau_{\text{ext}}, \tag{G.1}$$

where I_d, B_d, K_d are the desired inertia, damping, and stiffness matrices, Θ is the joint configuration, Θ_e is the equilibrium configuration, and τ_{ext} is the joint-space external torque mapped from the skin's measured wrench through the body Jacobian. Assuming $\ddot{\Theta}_e = \dot{\Theta}_e = 0$ and applying second-order finite differences,

$$\begin{aligned}\Theta(t) = &\left(\tfrac{I_d}{\Delta t^2} + \tfrac{B_d}{\Delta t} + K_d\right)^{-1} \\ &\times \left[\, \tfrac{I_d}{\Delta t^2}\left(2\Theta(t-1) - \Theta(t-2)\right) + \tfrac{B_d}{\Delta t}\,\Theta(t-1) + K_d\,\Theta_e + \tau_{\text{ext}}(t) \,\right]. \end{aligned} \tag{G.2}$$

The implementation runs inside the same materialOS ROS workspace described in Chapter 8; the controller is dispatched by the SMACH state machine and shares the joint-state and trajectory-generation pipeline with the trajGMM motion primitives of `pbdlib_manager`.

G.3 Obstacle-Avoidance Demonstration

The integrated system was exercised in two application configurations on the Sawyer platform (Lee *et al.*, 2024b).

Remote manipulation with haptic feedback. A PHANToM Omni operator drives the Sawyer through the bilateral teleoperation pipeline of `phantom_interface` while the skin's contact estimate is rendered as force feedback to the operator. The operator performs shape tracing of unknown objects and path finding through multi-obstacle environments using contact information alone.

Autonomous obstacle-avoidance navigation. The Sawyer follows straight-line paths between waypoints under joint-space admittance control (Eq. G.2) and reactively avoids obstacles through the contact-detection signal from the skin. A virtual-fixture engine biases the admittance reference away from detected contacts; representative trajectories with and without the virtual fixture were recorded as `trajectory_w_vf.txt` and `trajectory_wo_vf.txt` in the `phantom_interface` package. A separate launch entry, `demo_exp_smart_sensor_obstacle_avoidance_01.launch` in the `teleop` package, brings up the full obstacle-avoidance configuration (Sawyer controller, PHANToM cursor visualizer, RViz overlay).

For object manipulation, the autonomous mode maintained a target contact force of 6 N with RMSE 0.64 N while rotating an object around a fixed reference. The same admittance loop handles both reactive avoidance and contact-rich manipulation through a configuration change.

G.4 Architectural Distinction from DMMRP

The skin collaboration shared the Sawyer hardware and the ROS communication stack with the DMMRP of Chapter 8, but treated physical sensing as the contribution rather than MR-mediated task programming. Both projects shared the same architectural limitation that motivated the parameterized multi-robot platform in Chapter 9: robot-specific control parameters and sensor configurations were embedded in launch files and source headers, so adding a second robot or porting to a different institution required rewriting these components. The declarative configuration hierarchy of Chapter 9 eliminates this coupling by extracting the skin-related impedance gains and the smart-sensor topic remappings into per-robot YAML overrides, the same pattern that integrates the robot platforms listed in Table 9.2 across two institutions.

APPENDIX H

MATHEMATICAL DERIVATION OF LIE-GROUP ADMITTANCE ON $SE(3)$

This appendix provides the full derivation of Lemma 12.3 (Lie-Group Admittance on $SE(3)$) introduced in Chapter 12. The Lie-group impedance framework on which the derivation rests is established in (Kim *et al.*, 2025); the admittance dual on $SE(3)$ is derived below as the admittance counterpart of that framework. The notation follows (Kim *et al.*, 2025): $T \in SE(3)$ denotes a rigid-body pose, $V \in \mathfrak{se}(3)$ denotes a body twist, $\mathrm{Log}(\cdot)^\vee : SE(3) \to \mathbb{R}^6$ denotes the logarithm map composed with the canonical isomorphism between $\mathfrak{se}(3)$ and $\mathbb{R}^6$, and dexp_λ denotes the closed-form left Jacobian of the exponential map evaluated at $\lambda \in \mathbb{R}^6$.

H.1 Admittance Dynamics on $SE(3)$

Let $g_d \in SE(3)$ be the desired pose issued by the upper-layer policy, let $T_{\mathrm{disp}} \in SE(3)$ be an internal admittance displacement maintained by the controller, and let

$$\lambda \;=\; \mathrm{Log}(T_{\mathrm{disp}})^\vee \;\in\; \mathbb{R}^6 \tag{H.1}$$

be the exponential coordinate of that displacement. The same Log operator is used by (Kim *et al.*, 2025) for the impedance error of Equation 12.1, which keeps the impedance and admittance variables in the same geometric setting.

The externally measured wrench F_{world} is expressed in the body frame of the desired pose through the adjoint transpose, with $T_d \equiv T_{sd}$ taken as the desired body in the space frame,

$$F_{\mathrm{body}} \;=\; \mathrm{Ad}_{T_d}^{-\top} F_{\mathrm{world}}, \tag{H.2}$$

which is the dual of the velocity transformation between the same frames. Applying the dexp parameter transformation that (Kim *et al.*, 2025) derives for the impedance case (their Equations 58a–c) to the admittance inertia and damping yields the $\mathfrak{se}(3)$ admittance dynamics

$$\boxed{\; M_\lambda\, \ddot{\lambda} \;+\; D_\lambda\, \dot{\lambda} \;+\; K_d\, \lambda \;=\; \gamma \;} \tag{H.3}$$

with

$$\begin{aligned} M_\lambda &= \mathrm{dexp}_\lambda^{\top}\, M_d\, \mathrm{dexp}_\lambda, \\ D_\lambda &= \mathrm{dexp}_\lambda^{\top}\Big(\, D_d\, \mathrm{dexp}_\lambda \;+\; M_d\, \tfrac{d}{dt}\mathrm{dexp}_\lambda \,\Big), \\ \gamma &= \mathrm{dexp}_\lambda^{\top}\, F_{\mathrm{body}}, \end{aligned} \tag{H.4}$$

where M_d, D_d, and K_d are SPD matrices reconstructed from the Cholesky factors of the policy output (Section 12.5.2) and dexp_λ is the closed-form left Jacobian of the exponential map on $SE(3)$ derived in (Kim *et al.*, 2025). The stiffness K_d enters Equation H.3 undecorated by dexp_λ, following (Kim *et al.*, 2025) Equation 58c: K_d is defined in the body frame at the desired pose, and its action on the exponential coordinate λ is direct rather than transformed. Near the desired pose $\mathrm{dexp}_\lambda \to I$, so this is consistent with the dexp_λ-transformed inertia and damping to first order in $\|\lambda\|$ and matches the impedance-side stiffness convention of (Kim *et al.*, 2025).

Equation H.3 is the $SE(3)$ dual of the impedance dynamics in Equation 57 of (Kim *et al.*, 2025): the external wrench F_{world} enters as input through Equation H.2 and Equation H.4, the damping term dissipates the resulting twist, and the stiffness term pulls λ back toward zero (which corresponds to $T_{\text{disp}} = I$). When $F_{\text{world}} = 0$, λ converges to zero and $T_{\text{disp}} \to I$, so the modified pose returns to the desired pose. The Cholesky parameterization of Section 12.5.2 preserves the SPD structure of M_d, D_d, and K_d, and therefore of M_λ and D_λ, for any policy output.

H.2 Motivation for the Lie-Group Form

The classical Cartesian admittance equation $\Lambda_a \ddot{x}_m + D_a \dot{x}_m + K_a x_m = F_{\text{ext}}$ integrates a position modification in $\mathbb{R}^6$ and adds it to a nominal trajectory. For translation, this is unproblematic; for orientation, it is not. Three-parameter representations (Euler angles, axis-angle, rotation vector) all exhibit either gimbal lock or coordinate singularities, and four-parameter representations (unit quaternions) require a renormalization step at every integration that breaks the additive update assumed by the classical formula. Practical implementations therefore restrict the admittance to translation only, or treat orientation through a separate parallel update with empirically tuned coupling, and accept the resulting drift between the two channels.

The geometric formulation of (Kim *et al.*, 2025) avoids this by integrating the body twist $V \in \mathfrak{se}(3)$ directly on the Lie group. The exponential map composes the integrated twist with the current pose, and the resulting pose remains in $SE(3)$ without renormalization. The same property that motivates the Lie impedance controller also motivates the admittance extension reported here: the end-effector contact behavior depends on orientation as much as position, and a controller that drifts in orientation is not safe under a wrench with both linear and angular components. The admittance dual is therefore derived on $SE(3)$ rather than translated from the classical Euclidean form.

A second motivation is the unification with the diffusion-interaction policy. The augmented action of Equation 12.3 produces a single set of Cholesky factors whose reconstructed SPD matrices play the role of K_d and D_d for the impedance controller. The same SPD matrices play the role of K_d and D_d in the admittance dynamics, with the addition of an inertia term Λ_d that is also Cholesky-parameterized. A single policy output therefore parameterizes both modes, and the controller library can switch between them without retraining or reconfiguration.

H.3 Admittance Update on the Commutative Map

The $\mathfrak{se}(3)$ admittance dynamics of Equation H.3 are integrated one control step at a time. At step k, given the current exponential coordinate λ_k, its rate $\dot{\lambda}_k$, the measured wrench $F_{\text{world},k}$, and the policy-supplied SPD matrices M_d, D_d, K_d, the controller (i) transforms the wrench into the body frame through Equation H.2; (ii) assembles M_λ, D_λ, and γ from Equation H.4 using the closed-form dexp_λ machinery of (Kim *et al.*, 2025); (iii) solves Equation H.3 for the exponential-coordinate acceleration,

$$\ddot{\lambda}_k \;=\; M_\lambda^{-1}\Big(\,\gamma \;-\; D_\lambda\,\dot{\lambda}_k \;-\; K_d\,\lambda_k\,\Big); \tag{H.5}$$

and (iv) advances the rate and the exponential coordinate by Euler integration in the Lie algebra,

$$\dot{\lambda}_{k+1} \;=\; \dot{\lambda}_k + \ddot{\lambda}_k\,\Delta t, \qquad \lambda_{k+1} \;=\; \lambda_k + \dot{\lambda}_{k+1}\,\Delta t. \tag{H.6}$$

The internal displacement is recovered on the $SE(3)$ manifold through the exponential map on demand,

$$T_{\mathrm{disp},\,k+1} \;=\; \exp\!\Big(\,\lambda^{\wedge}_{k+1}\,\Big). \tag{H.7}$$

This coordinate-then-exponentiate update follows (Kim *et al.*, 2025) Equation 48, $\dot{\lambda} = \mathrm{dexp}_{\lambda}^{-1} V_{\mathrm{body}}$: $\dot{\lambda}$ is not the body twist except at $\mathrm{dexp}_{\lambda} \approx I$, so the naive group composition $T_{\mathrm{disp},\,k+1} = T_{\mathrm{disp},\,k}\,\exp((\dot{\lambda}_{k+1}\,\Delta t)^{\wedge})$ would treat $\dot{\lambda}$ as a body twist and drift from the dynamics of Equation H.3 for non-trivial displacements. Integrating in $\mathfrak{se}(3)$ coordinates and re-exponentiating preserves the variational consistency that the dexp_{λ}-transformed dynamics encode: the displacement remains in $SE(3)$ for any rate of change, the rotation and translation channels are coupled through the group structure rather than through an empirically tuned linear coupling, and there is no per-step renormalization. The C++ implementation of Section 12.5.2 uses this coordinate update directly.

The modified reference pose issued to the impedance controller of Equation 12.2 is then

$$T_{\mathrm{mod},\,k+1} \;=\; T_{d,\,k+1}\,T_{\mathrm{disp},\,k+1}, \tag{H.8}$$

which the impedance loop tracks as a stiff inner-loop reference. The admittance loop therefore translates a measured wrench into a modified reference pose for the existing impedance controller, without itself producing a torque command. This completes the derivation of Lemma 12.3.

APPENDIX I

PRELIMINARY VALIDATION OF THE DIFFUSION-INTERACTION POLICY

Section 12.6 of Chapter 12 reports the cardinal outcomes of the pre-hardware validation. The per-scenario, per-cell, and per-component evidence behind those outcomes is collected here. The four target robots are listed in Table I.1.

Robot	**DoF**	**Payload**	**Vendor (country)**
KUKA IIWA14	7	14 kg	KUKA AG (Germany)
Universal Robots UR16e	6	16 kg	Universal Robots A/S (Denmark)
UFACTORY Lite6	6	0.6 kg	UFACTORY (China)
UFACTORY xArm5	5	5 kg	UFACTORY (China)

Table I.1: Target robots for the pre-hardware validation. The four platforms span 5–7 DoF and three orders of magnitude in payload.

I.1 Single-Platform Drake Validation on KUKA IIWA14

The thirteen control scenarios that exercise every primitive of the controller library on KUKA IIWA14 are listed in Table I.2. Twelve scenarios are verified against the analytic or trajectory expectation of the scenario; the analytic operational-space-direction check halts at a singular initial body Jacobian and is treated separately in Section I.1.1.

#	**Scenario**	**Runtime (s)**
1	Cartesian admittance	2.7
2	Cartesian impedance	1.6
3	Lie-group impedance	2.2
4	Lie-group PD tracking	1.4
5	Lie-group admittance	1.8
6	Remote pattern execution	3.0
7	Digital-twin synchronization	3.6
8	Bilateral teleoperation scenario	3.8
9	Operational-space direction check†	0.3
10	Controller benchmark	5.1
11	Tracking under disturbance	2.4
12	Bilateral loop, C++ via Python binding	3.4
13	Virtual-fixture engine	7.4

Table I.2: Single-platform Drake scenarios on KUKA IIWA14 (Table I.1). Aggregate runtime 38.7 s; twelve scenarios match their analytic or trajectory expectation, and scenario 9 (†) halts at a singular initial body Jacobian (Section I.1.1). Recorded on 2026-03-28.

I.1.1 Singular Configuration in the Analytic Direction Check

The analytic check forms the operational-space inertia matrix $\Lambda = (J_b\, M^{-1}\, J_b^{\top})^{-1}$ at the initial joint configuration of the test, then inverts it to recover the principal task directions. At that configuration the body Jacobian J_b loses full rank: the smallest singular value reaches $\sigma_{\min} = 6 \times 10^{-6}$, and the inversion that defines Λ becomes ill-conditioned. The library returns a diagnostic exception that names the rank deficiency and the smallest singular value, and the test harness halts the scenario rather than substituting a clamped value or a pseudoinverse, so the singular configuration is exposed rather than masked. The bilateral teleoperation mode and the Lie-group impedance primitives used elsewhere in the architecture do not pass through this analytic path and are not affected by the halt.

I.1.2 Operational-Space Gravity Cancellation

The operational-space dynamics formula subtracts the gravity feedforward from the controller output, so the static balance holds without a joint-space integrator. The four legal combinations of gravity-feedforward sign and joint-space integrator are run on KUKA IIWA14 and Universal Robots UR16e under a 5 N, 0.5 Hz sinusoidal disturbance applied along the world $\hat{x}$ axis; Table I.3 reports the end-effector deviation at steady state.

Combination	**KUKA IIWA14**	**UR16e**	**Diagnostic**
$+\tau_g$ feedforward, integrator off	58 mm	49 mm	gravity offset uncovered
$+\tau_g$ feedforward, integrator on	< 1 mm	< 1 mm	integrator absorbs offset
$-\tau_g$ feedforward, integrator off	0.41 mm	0.41 mm	present formulation
$-\tau_g$ feedforward, integrator on	< 1 mm	< 1 mm	integrator inactive

Table I.3: End-effector deviation at steady state on KUKA IIWA14 and UR16e under a 5 N, 0.5 Hz sinusoidal disturbance along the world $\hat{x}$ axis. The $-\tau_g$ row with the integrator off is the present formulation of the controller library; the $+\tau_g$ row with the integrator off records the deviation when the prior gravity-feedforward sign is held without the integrator hiding it. The two integrator-on rows are reported as < 1 mm because the per-cell numeric was not separately recorded for those combinations.

I.1.3 Per-Step Gravity Probes

Two probes test alternative explanations for the residual in Section I.1.2. The first compares the gravity torque the controller applies against the gravity torque that Drake re-evaluates within the same 1 ms control step, and records a difference of 1×10^{-7} N·m, which is inconsistent with a per-step gravity-cache lag as the cause of the deviations in Table I.3. The second probes the smallest singular value of the body Jacobian at the home configuration of UFACTORY Lite6, and records $\sigma_{\min} = 0.217$,

which is inconsistent with a rank-deficient home configuration as an explanation for the Lite6 row in Section I.2.

I.2 Cross-Embodiment Regulation Residuals

The Cartesian and Lie-group impedance and admittance primitives are run on the four target robots under a regulation scenario family that is distinct from the disturbance scenario family of Section I.1.2. The peak end-effector deviation across the recorded scenarios is reported in Table I.4, with the column controller chosen on each robot to match the kinematic structure: a 5-DoF arm cannot satisfy a 6-D task-space target with the Lie-group controller and is recorded under the Cartesian-impedance specialisation only.

Robot	**Controller**	**Peak deviation**
KUKA IIWA14	Lie-group impedance	0.09 mm
UFACTORY Lite6	Lie-group impedance	0.01 mm
UFACTORY xArm5	Cartesian impedance	0.13 mm
Universal Robots UR16e	Lie-group $SE(3)$ admittance	not separately recorded

Table I.4: Cross-embodiment regulation residuals on the four target robots (Table I.1) under a regulation scenario family separate from the disturbance scenario of Table I.3. The three full-rank robots reach a peak deviation between 0.01 mm and 0.13 mm on the controller chosen for that platform. The UR16e row is left at not separately recorded because the corresponding measurement is scheduled together with the hardware-bench pilot in Chapter 13.

The full scenario registry covers thirty-one scenarios across the four robots, the two target families (hold-pose and circular trajectory), the four control primitives, and a contact-stiffness sweep on KUKA IIWA14. The registry file and per-scenario sidecar records are retained in the internal engineering archive.

I.3 Augmented Action Handler

The augmented action emitted by the diffusion-interaction policy carries two Cholesky factors that the controller reconstructs into the stiffness and damping matrices required by the impedance and admittance laws. The reconstruction returns symmetric positive-definite matrices on every draw of a reference distribution (Table I.5, row 1); the residual between the augmented action and the reachable subset of the controller's input space stays inside the bound of the factorisation theorem of the chapter (row 2); the map from the augmented action to the commanded torque has a finite Lipschitz constant (row 3); and the per-function unit suite over the C++ bindings the policy and controller use at runtime returns the expected value on every case (row 4). Figure I.1 shows the eigenvalue scatter that backs the reconstruction outcome of row 1.

I.4 Per-Component Timing

The 1 kHz budget that Section 12.5.1 assigns to the inner control loop is supported by the per-component timings in Table I.6. The torque kernel is benchmarked at

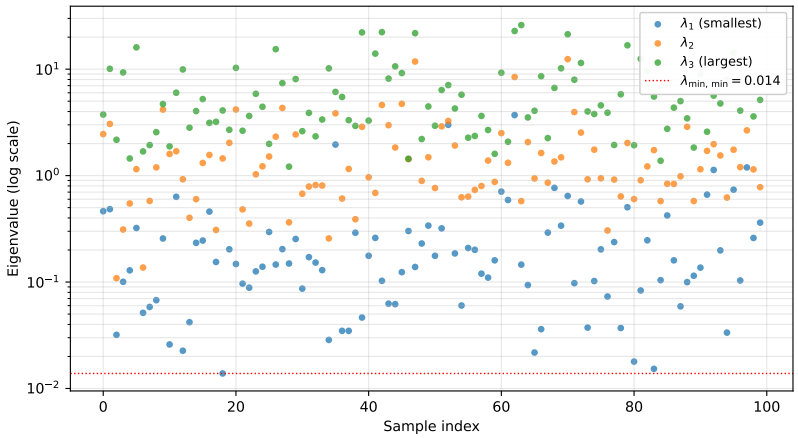


Figure I.1: Eigenvalue scatter of the Cholesky-to-SPD reconstruction for the 100 random draws of Table I.5 (seed 42, 3-D). Every drawn matrix has all three eigenvalues strictly positive; the smallest eigenvalue across the 100 samples remains above 0.013.

14.07 μs per call with a maximum repeat rate of 71 kHz; the safety checker that gates the dispatch path returns the expected verdict on every unit case at 1–5 μs per call; the per-step gravity-evaluation residual stays below 1×10^{-7} N·m.

Quantity	Recorded value
Cholesky-to-SPD reconstruction (100 random samples, seed 42, 3-D)	100/100 returned positive-definite; smallest eigenvalue mean $\overline{\lambda_{\min}} = 0.311$, range starts at $\lambda_{\min} = 0.013$; symmetry residual at machine precision.
Action-handler factorisation residual bound (5,000 Monte-Carlo samples; 18-D, 30-D, and 48-D Cholesky variants spanning 6-DoF through 23-DoF action spaces)	residual bound $B = 0.22$ in the three-term decomposition; 0/5,000 violations.
Action-to-torque Lipschitz constant	$L_z = 3.3$.
Pybind unit suite (`spd_from_flat`, `spd6_from_diag`, `validate_spd`, workspace projection, gain projection, delegation, torque, `ActionHandler.compute` (7-DoF and 6-DoF), `ActionHandler.compute_blended` ($\alpha = 0, 0.5, 1$), `TransferBound`, $SE(3)$ exp/log/dexp/adjoint round-trip; central-difference dexp Jacobian held to $\mathcal{O}(h^2) \approx 10^{-14}$)	27/27 unit cases returned the expected value.

Table I.5: Augmented action handler measurements. The reconstruction, factorisation residual, and Lipschitz constant are recorded against the references in Section 12.5.2; the pybind unit suite is the per-function check of the C++ bindings.

Component	Recorded value
C++ controller torque kernel (Lie-group impedance, body-frame formula)	14.07 μs per call; maximum repeat rate 71 kHz.
C++ safety checker (joint-limit and stiffness-budget gates)	9/9 unit cases returned the expected verdict; 1–5 μs per call.
Per-step gravity-torque residual within a 1 ms control step	1×10^{-7} N·m.

Table I.6: Per-component unit and timing measurements that back the 1 kHz inner-loop budget; recorded on the deployment workstation.

APPENDIX J

CO-AUTHOR PERMISSION STATEMENT

This appendix is provided in accordance with the ASU Graduate College *Policy on using Previously Published or Publishable Work in a Culminating Experience Document*, which requires either signed letters from co-authors or a statement by the student that all co-authors have granted their permissions.

The author hereby states that all co-authors of the previously published, publishable, and submitted collaborative works cited in this dissertation have granted their permission for the use of the cited material as supporting evidence for the architectural contributions described in this document.

J.1 Itemized Attribution and Permissions

The collaborative works cited in this dissertation are listed below by chapter. For each work, the venue, author list, and the author's contribution relative to co-authors are recorded; the chapter abstract that introduces each work also names the relevant co-authors and their contributions.

J.1.1 System 1 — Shoulder Rehabilitation Exoskeleton (Chapter 2)

- **Chang *et al.* (2021)** — IEEE ICRA 2021. Author list: Chang (first), with co-authors at ASU NeuRRo Lab. Author's contribution: complete software rewrite of the 4B-SPM platform from a 166 Hz monolithic prototype to the 250 Hz lock-free multi-threaded architecture documented in Chapter 2; pilot validation with five human subjects. Co-authors granted permission.
- **Hwang *et al.* (2024a)** — IEEE TBME 2024. Author list: Hwang and Chang (co-first authors), with senior-author supervision. Author's contribution: software architecture extension to the multi-language layered design that sustained the 40-participant biomechanics study (1,200+ trials over months), including parallel development support for four researchers. Hwang's contribution: biomechanics study design, IRB protocol, data collection lead, and the sex-difference analysis that constitutes the chapter's biomechanical claim. Co-authors granted permission.
- **Hwang *et al.* (2024b)** — 48th Annual Meeting of the American Society of Biomechanics (ASB), Madison, WI, August 2024 (Poster P1-281). Author list: Hwang (first), Chang, Saxena, Oleen, Paing, Atkins, Lee. Companion conference dissemination of the IEEE TBME 2024 work; author's contribution: software architecture support for the data collection campaign on the lock-free multi-threaded 4B-SPM platform documented in Chapter 2. Co-authors granted permission.

J.1.2 System 2 — Wearable Upper-limb Exoskeleton (Chapter 3)

- **Atkins *et al.* (2024)** — Wearable Technologies 2024. Author list: Atkins (first, mechanical hardware lead), Chang, with co-authors. Atkins's contribu-

tion: hybrid four-bar mechanical design, dual-purpose gravity compensation, and 6-DoF Spatial Compliant Alignment Mechanism. Author's contribution: ROS-based three-layer software architecture (Application/Interface/Hardware), 500 Hz transparent impedance loop with <2 ms jitter, and the protocol-switch infrastructure. Co-authors granted permission.

J.1.3 System 3 — Variable Damping Control (Chapter 4)

- **Zahedi *et al.* (2022)** — IEEE Robotics and Automation Letters 2022. Author list: Zahedi (first), with co-authors. Zahedi's contribution: Bayesian optimization framework for user-adaptive damping selection. Author's contribution: ZeroMQ ROUTER–ROUTER asynchronous task scheduler, session-continuous state machine, and the architectural integration that allowed the BO loop to coexist with the real-time controller. Co-authors granted permission.

J.1.4 System 4 — Gait Symmetry via MR (Chapter 5)

- **Save *et al.* (2026)** — IEEE TBME, under review (2026). Author list: Save and Chang (co-first authors), with senior-author supervision. Save's contribution: $N = 12$ participant recruitment, IRB study execution, and clinical interpretation. Author's contribution: Python real-time pipeline, ZeroMQ communication layer, ScriptableObject configuration layer, and Unity HoloLens 2 visualization that supported the controlled AR-vs-CD comparison. Co-authors granted permission.

J.1.5 System 5 — Single-User AR Admittance Rehabilitation (Chapter 6)

- Single-user AR admittance rehabilitation paper, in preparation. Author list: Chang (first), with co-authors. Author's contribution: 500 Hz Cartesian admittance loop, Unity–ROS2 data pipeline, eye-tracking integration on the same ROS2 clock, and the reaching-task experimental substrate documented in Chapter 6.

J.1.6 System 7 — DMMRP and Robotic Skin (Chapter 8)

- **Lee *et al.* (2024b)** — IEEE Transactions on Robotics 2024. Author list: Lee et al. (Seoul National University), with collaboration. Lee et al.'s contribution: robotic skin and fiber-optic sensing. Author's contribution: software architecture integration on the shared Sawyer platform. Co-authors granted permission.

J.1.7 System 8 — Multi-Robot Configuration Hierarchy (Chapter 9)

- The bilateral teleoperation controller used in the dual-arm collaboration is from KAIST IRiS Lab under Prof. Jee-Hwan Ryu and was a pre-existing software asset (Lee and Park, 2024b,a; Lee *et al.*, 2025, 2024a); the author's contribution is the architectural integration into the configuration hierarchy of Chapter 9. Co-authors granted permission.
- **Lee and Park (2024b,a); Lee *et al.* (2025, 2024a)** — KAIST IRiS Lab works cited as the controller substrate; the author's contribution to these papers is integration only. Co-authors granted permission.

J.1.8 System 9 — Multi-Agent AI Platform (Chapter 10)

- **Chang *et al.* (2025)** — WM2025 Symposia, Phoenix, AZ. Author list: Chang (first), Sola Kim, Young Soo Park. Author's contribution: multi-agent discussion framework, three-agent retrieval-augmented protocol, ten-round convergence study on consumer-grade hardware. Co-authors' contributions: domain expertise (Park, ANL Principal Investigator) and complementary corpus engineering (Kim). Co-authors granted permission.

- **Kim *et al.* (2026a)** — WM2026 Symposia, Phoenix, AZ. Author list: Sola Kim (first), Chang, Brian Archambault, Young Soo Park. Kim's contribution: four-agent extension with Orchestrator and heterogeneous data sources. Author's contribution: knowledge-graph-backed retrieval subsystem and the substrate reuse for physical supervision (Section 10.6.2). Archambault's contribution: domain expertise. Park's contribution: senior-author supervision and domain validation. Co-authors granted permission.

- **Kim *et al.* (2026b)** — ICLR 2026 Workshop on Algorithmic Fairness Across Alignment Procedures and Agentic Systems (AFAA), accepted (to appear). Author list: Sola Kim (first), Chang (second), Jieshu Wang. Kim's contribution: psychologically-grounded persona design framework and evaluation methodology. Author's contribution: multi-agent substrate integration and retrieval-augmented memory architecture support. Wang's contribution: stakeholder representation domain expertise. Co-authors granted permission.

J.1.9 System 10 — own-code Personal AI Infrastructure (Chapter 11)

- own-code multi-timescale runtime paper (R7 — Multi-Timescale Runtime; R8 — SHM Vision Transport), in preparation for IEEE RA-L. Author list: Chang (first), with co-authors. Author's contribution: complete software architecture — multi-provider asynchronous base class, provider-agnostic tool engine, microservice skill platform, real-time C++ core (safety checker, trajectory engine), zero-copy shared-memory vision transport, and operation-layer services (cascade-verified bringup, declarative task-contract orchestrator, voice-dispatchable tool surface) documented in Chapter 11.

J.1.10 Lie-Group Controller Collaboration (Chapter 12)

- **Park *et al.* (2026b)** — WM2026 Symposia, Phoenix, AZ. Author list: Seong-Su Park (first), Chang, Young Soo Park. Seongsu Park is a KAIST IRiS Lab researcher hosted at ANL as a visiting researcher during the collaboration period; subsequently returned to KAIST. Park's contribution: non-spherical-wrist control formulation. Author's contribution: software architecture integration on the multi-robot platform of Chapter 9. Co-authors granted permission.

- **Park *et al.* (2026a)** — WM2026 Symposia, Phoenix, AZ. Author list: Seong-Su Park (first), Chang, Young Soo Park. Park's contribution: heavy-manipulator control formulation, together with the Python/Drake prototype that served as the starting reference for the Lie-group impedance controller of Chapter 12.

Author's contribution: C++ migration into the unified `poe_robotics` library (Sections 12.5.2 and 12.5.2), the cross-embodiment validation across four robots, the Cholesky-parameterized augmented action handler, and the Lie-group admittance extension on $SE(3)$ (Lemma 12.3). Co-authors granted permission.

APPENDIX K

IMPLEMENTATION REFERENCE AND AUTHORSHIP STATEMENT

This appendix lists the repositories that hold the implementations described in Chapter 11 (`own-code`, `own-store`) and the operation-plane substrate referenced in Chapter 12 and Chapter 13 (`multi-hra-robots`, `multi-hra-dockers`), together with authorship and licensing terms.

K.1 Repositories

Table K.1: Repositories supporting the architectural contributions of this dissertation. Submission commits are the heads at the time the dissertation was filed; later commits are not part of the dissertation record.

Repository	**URL**	**Submission commit**
`own-code`	github.com/DongjuneChang/own-code	01cd33e
`own-store`	github.com/DongjuneChang/own-store	76c4ca6
`multi-hra-robots`	github.com/DongjuneChang/multi-hra-robots	8f5ca81
`multi-hra-dockers`	github.com/DongjuneChang/multi-hra-dockers	0be9c1b

K.2 Reproducibility

Validation results in Chapter 11 (the safety checker hot-path measurement, the cross-site latency stack, the controller jitter sample, and the operation-plane verification across multiple sites) are reproducible from the commit hashes listed in Section K above. Researchers seeking access for academic replication may contact the author.

BIOGRAPHICAL SKETCH

Dongjune Chang received a Bachelor of Science in Mechanical Engineering and Mathematical Science (double major) from the Korea Advanced Institute of Science and Technology (KAIST) in 2010, and a Master of Science in Mechanical Engineering from KAIST in 2012, where he worked in the Biorobotics Laboratory on parallel robot design and control. He subsequently earned a Master of Science in Nuclear Engineering from the University of New Mexico in 2019.

Between his degrees he served as a researcher in the Applied System Development Department of Dongbu Robot Co., Ltd. in Bucheon, South Korea, from 2012 to 2014, and as an Associate Research Engineer (later renamed Specialist) in the Product Engineering Research Institute of LG Electronics Inc. in Pyeongtaek, South Korea, from 2014 to 2017, working on laser processing equipment and battery welding monitoring systems. He completed his mandatory military service obligation through Korea's technical research personnel program during this period.

In 2019 he joined the Neuromuscular Control and Human Robotics Laboratory at Arizona State University under the supervision of Prof. Hyunglae Lee to pursue his doctoral studies in Mechanical Engineering with specialization in artificial intelligence, robotics, and mixed reality. During his doctoral studies he developed and deployed the software architectures for physical human-robot interaction and mixed reality rehabilitation described in this dissertation, spanning rehabilitation exoskeletons, augmented reality feedback systems, multi-user mixed reality platforms, and distributed multi-robot configuration hierarchies. He also served as a student research aide at Argonne National Laboratory, where he contributed to the Digital Modular Materials Robotic Platform, a U.S. Department of Energy Office of Environmental Management nuclear robotics project, and a collaboration with Seoul National University on fiber-optic

robotic skin integration. He additionally collaborated with the Interactive Robotic Systems Laboratory at KAIST on bilateral teleoperation control.

His research interests include real-time software architectures for physical human-robot interaction, mixed reality rehabilitation, multi-agent AI platforms for decision support, and interaction control of compliant robot systems. Publications from his doctoral work appear in IEEE Transactions on Biomedical Engineering, IEEE Robotics and Automation Letters, IEEE Transactions on Robotics, the IEEE International Conference on Robotics and Automation, Wearable Technologies, and the Waste Management Symposia, where his sole-authored paper on a multi-agent AI platform for nuclear waste decision support received the "Paper of Note" and "Superior Paper" awards at WM2025.

Upon completion of his doctorate, he joined Argonne National Laboratory as a Postdoctoral Appointee in the Applied Materials Division's Robotics and Remote Systems organization, under the supervision of Dr. Young Soo Park.