

CARNEGIE MELLON UNIVERSITY

School of Architecture
College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE:

JointLab: A VR Framework for Early Robotics Education in Architectural
Fabrication

AUTHOR:

Shandon Jude Herft

ACCEPTED BY ADVISORY COMMITTEE:

Joshua Bard Principal Advisor

5.6.2025
DATE

Daragh Byrne Advisor

May 7th 2025
DATE

Vernelle A. A. Noel Advisor

May 7th, 2025
DATE

JointLab:

A VR Framework for Early Robotics Education in Architectural Fabrication

Shandon Jude Herft

Master of Science in Computational Design Thesis Proposal

Carnegie Mellon University

May 2025

Thesis Committee:

Joshua Bard

Daragh Byrne

Vernelle A. A. Noel

Table of Contents

Abstract	1
1 Introduction	2
1.1 Applications of Robotics in Architecture and Fabrication	2
1.2 The Integration of Virtual Reality with Robotics	4
1.3 Challenges and Limitations of Integrating Robotics and Virtual Reality	5
1.4 Research Question	6
1.5 Structure of the Thesis	7
2 Background	9
2.1 Learning Robotics Systems	9
2.2 Embodied Learning	11
2.3 Virtual Reality (VR) Applications	12
2.4 Educational Robotics	13
2.4.1 Challenges of the Teach Pendant Interface	14
2.5 Key Case Studies	17
2.5.1 Mixed Reality for Industrial Robot Programming	17
2.5.2 VR-Based Programming-by-Demonstration (PbD)	19
2.5.3 Augmented Reality-Assisted Robot Programming System (ARRPS)	21
3 Prior Critical Investigations	24
3.1 Unity3D Industrial Robotics: ABB IRB 120	24
3.2 Unity3D Industrial Robotics: ABB IRB 120 EGM (Externally Guided Motion)	26
3.3 Unity3D Industrial Robotics: ABB CRB 15000 EGM (Externally Guided Motion)	28
3.4 egm-for-abb-robots	30
4 JointLab	33

4.1 Digital Recreation of the dFab Lab Environment	36
4.1.1 Spatial Mapping and Virtual Environment Implementation	38
4.1.2 IRB 120 Robotic Workstation: Architecture and Control Systems	39
4.2 UI Implementation	43
4.2.1 Connection Panel	43
4.2.2 Diagnostic Panel	48
4.2.3 Camera Panel	53
4.2.4 Joint Controller	60
4.2.5 EGM Communication Framework	70
4.2.6 Waypoint Panel	72
5 User Study and Evaluation	82
5.1 Key Observations from User Interaction	87
5.1.1 Task 3 Quantitative Analysis: 3D Maze Navigation	89
5.2 Challenges and Suggestions	96
5.3 Observations on Skill Development	98
5.4 Implications for Design	100
5.5 Implications for Training and Learning	101
6 Discussion	104
6.1 The Future of Immersive Robotics Education	104
6.2 Conclusions	107
6.3 Limitations	108
6.4 Future Works	111
7 Bibliography	114
8 Appendix	118
8.1 User Transcripts	118
8.2 Task 3 Results	142

8.2.1 Target Coordinates for Maze Path	142
8.2.2 Participant Data	142
8.3 Repository Access	151

List of Figures

Figure 1: Examples of robotic fabrication in architecture. Source: Parametric Architecture.

Figure 2: ABB IRB 120 robot with FlexPendant controller.

Figure 3: Visualization of coordinate system misalignment between Unity and RobotStudio.

Figure 4: UI elements used for robot path creation and simulation in mixed reality setup. Source: Svensson.

Figure 5: ABB RobotStudio VR test environment with virtual robot work cell.

Figure 6: VR add-on in use for immersive robot programming.

Figure 7: Misaligned coordinate axes of the end effector in tilted local coordinate system.

Figure 8: Reachability and manipulability visualization in the ARRPS interface.

Figure 9: AR-based robot path programming using a handheld pointer.

Figure 10: Digital twin interface for ABB IRB 120 in Unity3D.

Figure 11: Real-time joint control of ABB IRB 120 via Unity3D EGM framework.

Figure 12: Digital twin of ABB CRB 15000 GoFa with Cartesian coordinate control in Unity3D.

Figure 13: EGM communication interface implemented in Unity using egm-for-abb-robots.

Figure 14: Physical layout of CMU's dFab Lab, showing ABB industrial robots and linear track system.

Figure 15: Wireframe view of the virtual dFab Lab, replicating the physical lab layout for simulation purposes.

Figure 16: Virtual workstation modeled in Unity based on the physical ABB IRB 120 setup.

Figure 17: ABB IRB 120 robot model visualized in ABB RobotStudio.

Figure 18: Connection Panel UI for managing IP setup and communication with RobotStudio.

Figure 19: Diagnostic Panel UI displaying real-time joint angles, position, and orientation data.

Figure 20: Camera Panel UI for switching between predefined viewpoints in the virtual environment.

Figure 21: Right-side camera view of the ABB IRB 120 in the virtual dFab Lab.

Figure 22: Left-side camera view of the ABB IRB 120 in the virtual dFab Lab.

Figure 23: Front (home) camera view of the ABB IRB 120 in the virtual dFab Lab.

Figure 24: Top-down camera view of the ABB IRB 120 in the virtual dFab Lab.

Figure 25: Joint Controller UI for real-time manipulation of individual robot joints and axis visualization.

Figure 26: Joint highlighting functionality showing selected joint with visual emphasis for teaching or debugging.

Figure 27: Axis visualization using rotation discs to indicate range and orientation for each joint.

Figure 28: Waypoint Panel UI for recording, playing, and exporting robot motion paths.

Figure 29: Waypoint markers displayed along a robot path in the virtual environment (right view).

Figure 30: Waypoint visualization showing motion sequence from a left perspective.

Figure 31: Robot model in default home position before waypoint path execution.

Figure 32: Auto-generated RAPID code representing recorded waypoints for ABB RobotStudio.

Figure 33: Robot returning to home position following waypoint playback completion.

Figure 34: A robot classroom scenario with limited access to robots.

Figure 35: Students train in a safe virtual environment.

Figure 36: Students complete structured robotic tasks in the virtual environment.

Figure 37: Teachers monitor session analytics.

Figure 38: Students transition to real robot after virtual training.

Figure 39: Students demonstrate learned skills on a real robot.

Figure 40: Task 2 interface showing sphere-based spatial navigation using JointLab's UI panels.

Figure 41: Task 3 maze navigation environment with numbered waypoints for spatial reasoning evaluation.

Figure 42: User operating JointLab in VR using Meta Quest during task-based sessions.

Figure 43: Mean distance error and task completion time trends across 10 users during Task 3.

Figure 44: User performance scores aggregated across all tasks, with lower scores indicating better performance.

Figure 45: Heatmap of point-by-point deviation from target positions for each user in Task 3.

Figure 46: Axis-specific mean absolute error per user across X, Y, and Z coordinates.

Figure 47: User clustering based on mean distance error and task completion time using K-means.

Abstract

This thesis presents JointLab, an immersive educational platform designed to support early-stage learning in robotics and architectural fabrication. Aimed at students and novices, JointLab leverages Unity, ABB RobotStudio, and the Meta Quest 3's controller to provide real-time, bi-directional communication with an ABB IRB 120 robot. Set within a digital twin of Carnegie Mellon University's dFab Lab, JointLab offers an accessible, low-risk environment for learning complex robotics concepts through simulation. Users can operate robotic joints, plan toolpaths, and monitor diagnostic data using intuitive interface panels, including modules for joint control, spatial navigation, live connection via Robot Web Services (RWS), and low-latency control using Externally Guided Motion (EGM). By removing traditional programming obstacles, JointLab sets opportunities for learners without any experience in coding or robotics to engage in spatial understanding and robotic workflow development. A structured user study evaluates the effectiveness of the system in understanding joint manipulation, motion planning intuition, and enabling user confidence with real-world robotics. Many participants with little to no prior experience developed a clearer understanding of spatial understanding and robotic movement. Feedback reinforced JointLab's value as an engaging educational entry point into robotics, demystifying complex concepts and made robotic systems feel more approachable. This work contributes to computational design, design fabrication, and robotics education by demonstrating how immersive VR environments, paired with simplified layouts and controller interactions, can help make robotics more accessible. JointLab offers a scalable model for bridging experimentation and real-world implementation. A fully documented implementation is publicly available on GitHub to support its further development.

1 Introduction

1.1 Applications of Robotics in Architecture and Fabrication

The integration of robotics in architectural processes represents a remarkable convergence of technology and design. By automating tasks that were once labor-intensive, robotics is redefining how architects and builders approach their work. For example, robotic arms can now execute precise bricklaying, allowing for more intricate and ambitious designs without the constraints of human error or fatigue. This not only accelerates construction timelines but also enhances the quality and consistency of the final structure. The growing role of robotics in architecture is underscored by its ability to tackle complex tasks with greater efficiency. As more architectural firms recognize the value of robotics, industry is witnessing a shift from traditional methods to more automated, technology-driven approaches. This transformation is not merely about replacing human effort but augmenting it to achieve higher standards of creativity and functionality [1].

Examples of robotic architecture projects: structures, materials, and installations built by robots.

- **Flight Assembled Architecture Robots**

It's the first installation to be built by flying machines. Flying robots assemble a tower from lightweight polystyrene bricks [6].

- **DFAB House/NCCR Digital Fabrication**

The project aimed at transforming the design and building process by integrating computational design and digital fabrication into architecture. A live scale-built structure using robots and modern fabrication methods [6].

- **ICD-ITKE Pavilion / University of Stuttgart**

The Institute for Computational Design and the Institute of Building Structures continue their series of research pavilions. These building prototypes explore novel computational design, simulation and fabrication processes in architecture. Experimentation on fabrication and construction processes on a six-axis robot used to create composite structures [6].

- **SOM Exhibition – Anatomy of Structures**

The exhibition by SOM sought to establish a dialogue between past and present ideas that lead to new architecture and an unexpected approach to realizing artworks. Industrial robots creating a vault of glass bricks [6].

- **The Programmed Wall – ETH Zurich**

The curtain wall project question traditional rigidity and massive appearance of brick faces through scripted design tools. An industrial robot laid a brick wall with a twisted profile [6].





Figure 1: Examples of robotic fabrication in architecture. Source: Parametric Architecture.

1.2 The Integration of Virtual Reality with Robotics

Virtual reality (VR) can help human-robot interaction (HRI) by providing a tool for successfully iterating across robot concepts. Moreover, virtual reality (VR) has been utilized in various human-robot cooperative and agreeable modern applications to facilitate human-robot interaction. VR headsets, such as the Oculus Rift or HTC Vive, allow users to see and interact with virtual representations of robots in real time, as if they were physically present in the same room. This can make it easier for humans to understand and control the movements and actions of robots, as well as provide visual feedback to the robot about its environment. Furthermore, VR also allows for remote collaboration, where multiple users can interact with robots from different locations. By providing a shared virtual environment, VR can enable humans to work together more effectively with robots, whether they are in the same room or in different parts of the world. Overall, VR has the potential to greatly enhance human-robot interaction by providing a more immersive and intuitive interface for controlling and communicating with robots. It can also facilitate remote

collaboration and enable simulation-based testing, making it a valuable tool for a wide range of applications in industry, healthcare and research [3].

The designers of a robot system want to evaluate its performance in environments that are difficult to reach or construct, or infeasible to simulate. In this case, VRR provides a hybrid approach to real and virtual worlds that is more realistic than simulation but more accessible than normal deployment. Testing is largely a white-box application: we have a model of the robot's "brain," and use VRR to ensure its accuracy in the face of challenging environments [4].

Virtual reality in education has added another dimension for students to explore and understand the core concept of a certain work application. Virtual reality was widely used in education for topics such as virtual reality robotics where they learn about automation. This style of teaching is very appealing for young students who enjoy using the latest technology. Their experience of studying turns into playing with technology and learning at the same time and that is the key to improving their experience. This is also useful for students who are slow-learners and have a particular style of understanding such as creatives who find the learning process easier with colors, symbols and textures [[5][6][7][8][9]]. Users can move a virtual robot end effector and generate a tool path using ABB RobotStudio. Virtual commissioning enabled by Virtual reality ensures safety but also helps in cost-cutting associated with errors, debugging, redundancy before physical implementation.

1.3 Challenges and Limitations of Integrating Robotics and Virtual Reality

The integration of Robotics and Virtual Reality has significant potential in fields like architectural fabrication and education but faces several challenges that must be addressed to realize its benefits.

Manyika et al. [10] reported that the level of digitalization index for the construction industry was the lowest of 22 industries. Results from a survey conducted by Davila Delgado [11] for the present study estimated a VR adoption factor in the UK construction sector of 2.5 out of 5, and of 1.5 out of 5 for AR, where 5 represents full adoption and 1 represents no adoption at all. These low levels of adoption are caused by myriad complex and interrelated factors that are very difficult to understand, and consequently, appropriate mitigating actions cannot be devised. Current AR and VR technologies are not compatible with AEC standard data-exchange formats (e.g., Industry Foundation Classes), making the integration of standard AEC software packages and AR and VR software tools difficult. Thus, AEC companies should engage with and aid standardization bodies to include support for AR and VR formats. AEC companies also should collaborate on developing open-source conversion tools between AEC file formats and AR and VR formats [12]. This lack of standards affects seamless and robust data exchange. There is also the impossibility of being able to play content developed for one VR device on another. There exists a lack of interoperability which limits adoption between VR platforms and Robotics.

1.4 Research Question

To explore these opportunities further the research poses the following question:

How can real-time interaction in virtual simulation environments support early-stage robotics education by improving students' skill development in joint manipulation, path planning, and spatial understanding?

Robotic workflows have become integral to architectural fabrication in the iterative prototyping phase. Existing tools present a significant barrier to entry due to reliance on complex programming and abstract interfaces. Virtual reality integration promises solutions by creating an immersive environment, simplifying learning. With this potential, there remains a gap in understanding how real-time methods can impact robotics workflow development and education in an architectural fabrication context.

1.5 Structure of the Thesis

This thesis is organized into several chapters that support the exploration and evaluation of a real-time, bi-directional interaction system integrating robotics and virtual reality for education in an architectural fabrication context. Each chapter builds upon the previous, to provide insights into the design and implementation of the proposed framework:

- *Chapter 2: Background* outlines the foundation necessary to understand the intersection of robotics, virtual reality and architectural workflows. It includes key concepts in learning robotic systems, embodied learning, VR applications, and educational robotics, supplemented by precedent case studies that inform this research.
- *Chapter 3: Prior Critical Investigations* discusses existing digital twin frameworks and robotic systems – particularly those using Unity, ABB IRB robots, and EGM protocols are analyzed. These

investigations help identify gaps in interaction, scalability, and bi-directional control that this thesis aims to address.

- *Chapter 4: JointLab* presents the core hypothesis and is also the implementation chapter where the proposed framework – JointLab is introduced. It includes the digital recreation of the dFab Lab, details the UI system (including connection, diagnostic, joint control, and waypoint panels), and explains the integration of Meta Quest 3, Unity3D, and ABB RobotStudio using EGM and RWS protocols.
- *Chapter 5: User Study and Evaluation* presents a structured user study that evaluates the usability and effectiveness of JointLab through 3 tasks. User impressions are analyzed to assess spatial understanding and learning outcomes.
- *Chapter 6: Discussion* synthesizes the results and discusses the strength and limitations of the framework. It also reflects on broader implications for robotics education, immersive design and collaboration.

2 Background

The field of robotics has rapidly evolved, integrating advanced learning mechanisms, virtual environments, and educational applications to enhance its systems. Understanding the foundational concepts that drive these advancements is essential for better contextualization of this research. This chapter reviews key areas relevant to this study, including learning robotic systems, embodied learning principles, virtual reality (VR) applications, and educational robotics. These topics collectively establish the theoretical and technological groundwork for research.

2.1 Learning Robotics Systems: Tools and Frameworks

ROS is a widely used framework in robotic programming due to its modularity and flexibility. Libraries like MoveIt! offer sophisticated motion planning capabilities. A core component of any autonomous system is motion planning [113],[114],[115] which finds feasible motions that satisfy task requirements (e.g., reaching the goal, satisfying some motion constraint, etc.). There are many motion planning software systems for general manipulators; a popular library for motion planning is MOVEIT [116], which is built on top of the ubiquitous Robot Operating System (ROS) framework [117]. MOVEIT has four key advantages: it is widely adopted in industry and research, it is easy to set up for new robots and over 150 robots are already available [118], it is easy to integrate with a ROS system, and it has a large and vibrant open-source community. However, due to MOVEIT's massive scope and abstract architecture, many tasks are challenging for both engineers and researchers. For

example, it can be difficult to evaluate and develop planning algorithms, extend a planner's functionality, extract low-level information from planners, or use a planner within the scope of a broader planning algorithm, e.g., task and motion planning [19],[20]. Familiarization with basic concepts and installations were described as time-consuming [21]. Even just a simple task such as installing ROS could be daunting for the uninitiated. Key limitations in using ROS include the steep learning curve and computational resources needed to run the program [22]. ROS was designed to run on Linux based systems; Streamlining these processes for instructors and students and making sure students have access to sufficient computational power to run simulations is a huge barrier for teaching and implementing ROS [23]. In his analysis of robotics in post-secondary education, Esposito identifies several major drawbacks to implementing ROS [24]. Esposito compared the use of robotics software and programming languages in the classroom like ROS, MATLAB, and C. He found that ROS adoption significantly lagged the other more mature software. ROS is also substantially more complicated than software like MATLAB. Students need to learn to navigate a Linux environment, use object-oriented programming concepts, and link various languages and libraries in their development environment [24]. Robotics, being so interdisciplinary, may draw students who are not particularly strong in or interested in developing the computer programming skills required to be successful with ROS [23].

ROS has itself established as a powerful tool for robotic programming. Its advanced libraries like MoveIt! has led to its widespread adoption in research, however its steep learning curve and reliance on coding, creates a challenge in accessibility, especially for non-experts, these limitations such as difficulties with installation, computational resource requirements and navigating Linux – underscores the need for more accessible solutions that are intuitive.

ABB's RobotStudio is also a powerful simulation and offline programming software tool that can be applied to a variety of industrial automation applications. Its offline programming capability, realistic simulations, and add-on features make it an essential tool for maximizing return on investment and optimizing robot systems without disrupting production. The software's virtual commissioning feature enables the testing of the complete logic and safety of the cell virtually before installation, saving time and reducing the risk of issues arising during production. RobotStudio is also widely used in the electronics industry, where it is applied to tasks such as soldering, pick and place, and assembly. The software's augmented reality feature allows for the visualization of robot solutions by overlaying the modeled solution over the real-life production environment as a hologram [25]. Key to accurate off-line programming with RobotStudio is VirtualRobot Technology. Exactly, the same code runs on the PC and the robot controller. So, when the code is fully developed off-line, it downloads direct to the controller, reducing time-to-market. RobotStudio Version 5 brought the capability to run several “virtual” robots at the same time, supporting the new MultiMove technology on the IRC5 controller [26].

2.2 Embodied Learning

Programming by Demonstration (PbD) with passive observation stands out as the most intuitive approach in robot programming, which enables the operator to perform the demonstration in a high Degree of Freedom (DOF) using their body and requires almost no extra training [27]. With the advent of collaborative robots, a third programming approach is now gaining attention.

Programming by demonstration (PbD) is often used to program collaborative robots. Using PbD, the user programs the robot by physically holding the robot end effector to move it along the intended path, and the computer records the tool path. Accuracy of the program can be enhanced

by revising the location of certain points afterward. While the PbD approach seems to be more intuitive, it requires a collaborative robot. Since collaborative robots are relatively new, few academic institutions have them available for students to use. Although this method is intuitive, it is not as popular due to its need of special hardware to detect human movement through camera or force sensors and optimize the robot path. Additionally, the safety of the programmer or the operator in such a closed-range, collaborative environment [[28],[29]].

PbD has proven to be an intuitive approach for robotics programing, particularly in service robots' contexts. Its ability to leverage human interactions simplifies robot programming for non-expert users. However, challenges such as precision, scalability, reliance on specialized hardware, and operator safety in close range environments limit their widespread adoption.

2.3 Virtual Reality (VR) Applications

Current HRC (Human-Robot Collaboration) interfaces primarily involve workers interacting with robots through 2D screen teaching pendants, control panels, and teaching keys, often requiring a certain level of experience to operate them proficiently. Gaining this experience necessitates extensive training and prolonged usage, enabling less experienced workers or newcomers to understand the principles of interaction and the methods for executing corresponding operations. In recent years of research, the combination of various intelligent devices has been proposed to achieve more intuitive visualization, remote control, and collaborative work. Emerging interactive devices and display terminals such as tablets, virtual reality (VR) [[30],[31]], augmented reality (AR) devices [32], sound [33], smartphones, and gestures [34] have been integrated. These devices have brought new solutions to achieve a more intuitive visualization of teleoperation issues. Additionally, in advanced

manufacturing, including digital twins [35], data analysis, and industrial big data, there is a demand for autonomy separate from the interface. Therefore, advanced HRC necessitates novel HRC to facilitate intuitive visualization, control, and autonomous collaboration. VR devices simulate the real-world environment through computer technology, providing a remote operation and immersive experience. This offers new solutions for more intuitive visualization of teleoperation challenges. The system's adaptability to diverse industrial settings and various robots stems from its openness. The Unity engine's support for different scene models and its inherent compatibility enables the system to accommodate various industrial environments [36]. The integration of VR into robotics programming has revolutionized Human-Robot Collaboration by providing visualization, control and interaction through immersive solutions. Unlike 2D interfaces, VR systems enable users to simulate robotic workflows, refine adjustments that are task-specific without any physical prototypes, and enhance user experience with intuitive hand-tracking mechanisms. The adaptability of VR platforms, such as Unity, further helps support these robotic systems, making them highly scalable.

2.4 Educational Robotics

The use of virtual environments and of remote access for the teaching/learning of Robotics and Mechatronics may help in a shared way and collaboration among the national universities that possess the infrastructure for it [37]. Virtual laboratory is the application of combined classical way of teaching and e-learning courses [38]. Any improper operation of robot arm movement may cause damage to the equipment as it may occur during the learning process. The development of robotic e-learning can face the problem mentioned above, and the student can feel virtually the robotic simulation and can decrease the financial of robot maintenance [39]. Robotics is an important area in

science and technology and with the use of virtual reality technology users can interact efficiently with 3D computerized environments in real time using their natural senses and skills [40]. In the area of computer graphics, virtual robots are realized through avatars, the captured motion data is used for directly determining the avatar behavior [41]. Robot control with Internet Technology becomes an interesting and promising research direction and in most existing teleoperation systems, 3D virtual environments are used to get better control of robots [42]. Virtual environments can be built using computer simulation technology; therefore, carry on the direct control through data gloves, head mounted displays and other interactive equipment [43]. In the field of robotics within manufacturing processes, many applications have been created [[44][45][46][47]] as virtual reality systems for e-learning training intentions, mainly via web. The main drawback in the previously mentioned papers is that they are not immersive systems [48]. Educational robotics has emerged as a transformative tool by combining traditional laboratory methods with virtual learning environments. By leveraging VR, robotics education provides students with hands-on opportunities without the risk of physical damage and high maintenance costs that would otherwise be associated with it. This allows learners to interact intuitively, fostering a deeper understanding of its applications and concepts. Despite these advancements, the main drawback to its lack of immersion is their ability to fully replicate real-world scenarios.

2.4.1 Challenges of the Teach Pendant Interface



Figure 2: ABB IRB 120 robot with FlexPendant controller.

The ABB IRB 120 is a widely used 6-axis industrial robot that is typically controlled with the FlexPendant in combination with ABB’s IRC5 Compact controller. While this configuration is useful in industrial applications, it introduces multiple barriers in an education setting.

- *Interface Complexity and Learning Curve:* A core obstacle in robotics education, especially at the beginner level, is the difficulty in engaging with the FlexPendant. These tools are powerful but not designed with novice users in mind. For learners with no prior programming experience, navigating such interfaces could be difficult. This is particularly evident in the case of the ABB FlexPendant, the default interface for the ABB IRB 120. In a recent user-centered study exploring learning curves of robotics, researchers found that new users struggled with the navigation structure of the FlexPendant and its programming. Participants reported code entry on the pendant was unintuitive and more difficult than using a computer, especially for users unfamiliar with programming [55].

- *Limited Visual Feedback:* For novice users, the absence of clear visual cues can create uncertainty. As highlighted in user interviews, beginners often struggle to predict how the robot will move, especially when trying to fine-tune speed, distance, or direction. This challenge is paired with the difficulties in understanding the currently active coordinate system, an essential detail that affects the robot's motion but is not easily visible on the interface. Additionally, the supporting documentation for the FlexPendant is described as difficult to understand, offering less clarity for users attempting troubleshoot movement behavior [55].
- *Safety and Accessibility Concerns:* Operating the robot for teaching or jogging tasks while using the FlexPendant requires the user to be near the robot being manipulated. Collaborative robots are designed to be operated alongside human operates in a collaborative manner, but working without fences increases the probability of hazardous situations. Struggling to understand the robot's movements due to its interface limitations brings up a safety concern for being physically near a powerful machine [55].
- *Barriers to Exploratory Learning:* The primary programming language is RAPID, ABB's own language, this indicates a reliance on traditional text-based programming. Novice operators experience five times higher cognitive load than needed and 50% less efficient than experts at completing the same task in a 3D environment while using the FlexPendant. The current interface makes it harder for novices to build a functional mental model [55].

2.5 Key Case Studies

To get a better understanding of the current landscape of work, I have reviewed three prior studies to partially address the research question. These works added with visuals will help create a foundation to the understanding of the current field with gaps that are presented and how this research will address it.

2.5.1 Mixed Reality for Industrial Robot Programming

This study demonstrates the integration of the Microsoft HoloLens 2 and ABB RobotStudio via a TCPIP connection that allows the user to create and execute robotic paths in a mixed-reality environments. Within this application, the user can interact with several menus and objects to program the path, which can be configured and simulated in the virtual environment with reachability checking. Paths can also be exported to the real robot, where they are conducted with accurate performance. Considering the learning process, a guide can be activated to show recommended steps, and it is possible to jog the virtual robot to comprehend robot motions. In addition, two experiments are conducted to test spatial mapping and marker detection capabilities [49].

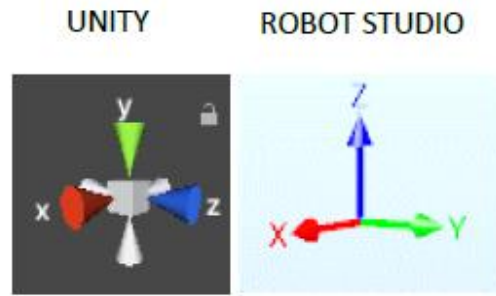


Figure 3: Visualization of coordinate system misalignment between Unity and RobotStudio.

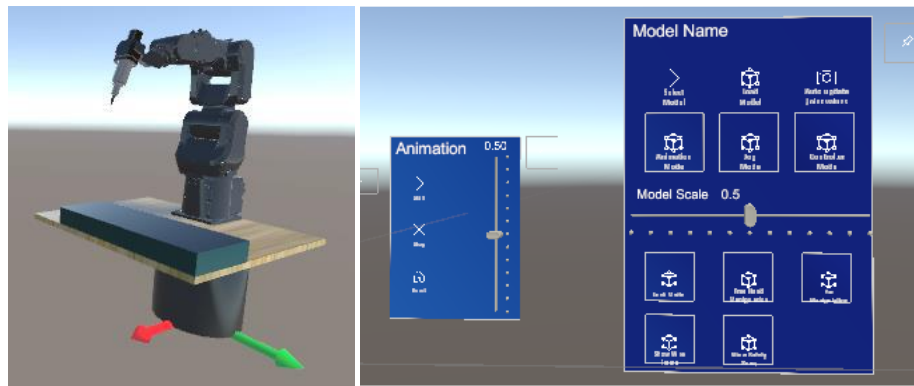


Figure 4: UI elements used for robot path creation and simulation in mixed reality setup. Source: Svensson.

This would be a critical issue for achieving something of a real-time interaction [49]. This challenge takes my aim and prioritizes it to optimize latency issues and communication errors that would usually exist between frameworks like Unity and RobotStudio.

This approach establishes that there is a possibility for integrating design and simulation workflows, but it doesn't show its scalability. It applies only to a single robot and more advanced features like collision avoidance. User fatigue and steep learning curves for new users pose challenges that follow [49]. My research also focuses on a single robot, and I intend to address inefficiencies when it comes to interaction and latency by reaching near real-time feedback and adjustment.

The mixed reality approach is an innovative approach to bridge gaps that exist between virtual and physical environments to help make robotics more accessible. However, the HoloLens hardware has scalability issues and for various use cases is hard to adapt for a single robot approach instance while multi robot approach are not considered, so the next step would naturally be to expand on that, also the issues regarding user fatigue and steep learning curves highlights the needs for more intuitive interfaces which my research addresses by incorporating the Meta Quest 3 with its hand tracking capabilities in contrast to the workflow in this research. My research explores better integration by connecting frameworks like Unity, and RobotStudio in a bi-directional feedback loop. While there exists a foundation for mixed reality integration with industrial robots, my work seeks to expand on its concepts with the focus on real-time performance and user experience.

2.5.2 VR-Based Programming-by-Demonstration (PbD)

Using ABB RobotStudio's virtual reality add-on this study introduces an immersive method for robot path planning by adjusting a virtual robot's end effector. The virtual reality test environment, and Tilted Coordinate System shows how to set up more intuitive programming and presents challenges like misaligned coordinate systems. Being part of ABB RobotStudio, the VR add-on serves as the human-computer interface and provides an immersive environment for the user to interact with virtual robots. The compatible VR headsets include HTC VIVE, Oculus Rift, or Microsoft HoloLens 1. The former two systems are tethered to the computer for rendering and require two infrared sensors (e.g. lighthouses) to determine the user's spatial location, movement,

body posture, and controller gesture. A physical space free of obstacles is needed to ensure natural yet safe navigation in the virtual space. The study also suggests the need for additional research on multi-user collaboration [29].



Figure 5: ABB RobotStudio VR test environment with virtual robot work cell.

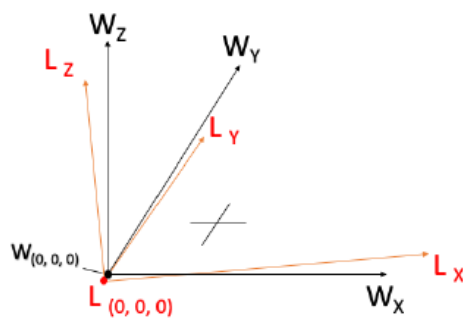


Figure 6: VR add-on in use for immersive robot programming.

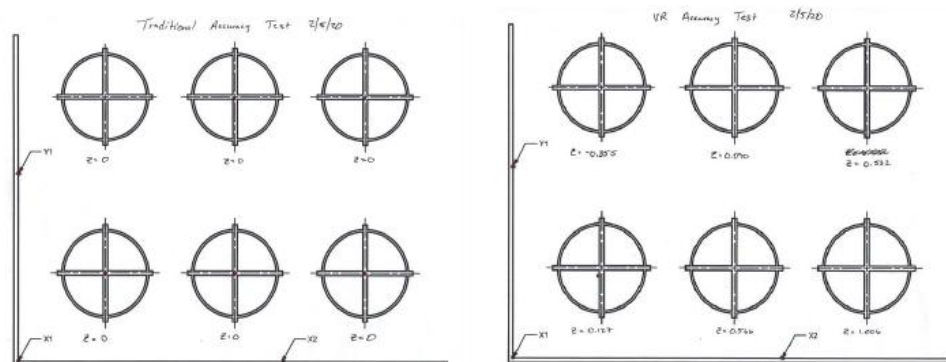


Figure 7: Misaligned coordinate axes of the end effector in tilted local coordinate system.

The test outcome image [29] demonstrates the deviations in accuracy between virtual reality and traditional approaches, making it require a more precise coordinate calibration system; This would be a key focus in my approach.

My research addresses these shortcomings by optimizing the data transfer pipeline to reduce latency and ensure alignment. My research integrates a more robust communication between Unity and RobotStudio, that aims to tackle the misalignment issues. This study also underscores the potential of virtual reality to bring robotic workflow development and addressing the full user experience. My work would fill this gap by utilizing features like the near real-time hand tracking framework for more precise and intricate interactions.

2.5.3 Augmented Reality-Assisted Robot Programming System (ARRPS)

ARRPS combines augmented reality interfaces, and motion planning algorithms to simplify the robotic programming process by offering real-time feedback and information such as reachability and manipulability. ARRPS is designed to allow users with little robot programming knowledge to program tasks for a serial robot. The system transforms the work cell of a serial industrial robot into an AR environment. With an AR user interface and a handheld pointer for interaction, users are free to move around the work cell to define 3D points and paths for the real robot to follow. Sensor data and algorithms are used for robot motion planning, collision detection and plan validation. The

proposed approach enables fast and intuitive robotic path and task programming and allows users to focus only on the definition of tasks. The color-coded feedback visualized [50] showcases how intuitive design helps the user understand better, this is a principle that is central to this research.

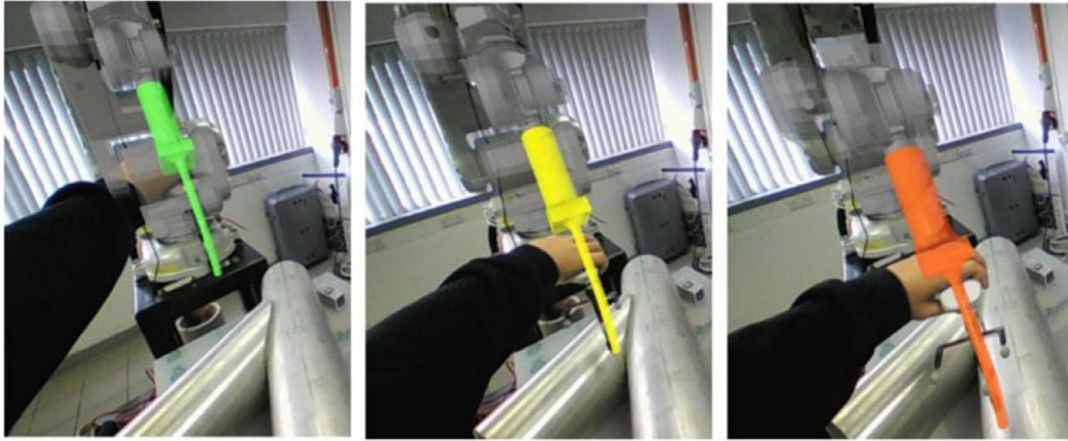


Figure 8: Reachability and manipulability visualization in the ARRPS interface.

Users define robotic paths using a handheld pointer that is tracked by a motion tracker system while system automated collision avoidance and optimized joint configuration. Although this approach is more efficient for applications like welding and pick-and-place, the study does not explore the use of general-purpose robotics or during task specific measures, additionally high precision tasks remain a challenge [50]

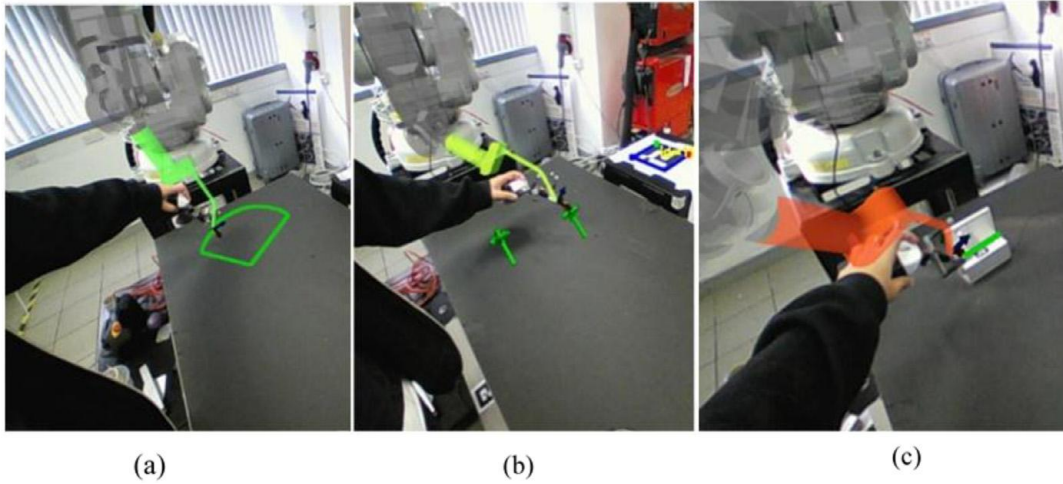


Figure 9: AR-based robot path programming using a handheld pointer.

The absence of user input mechanisms, such as hand tracking limits its potential for broader use. My research expands on these ideas by integrating task specific parameters and ensuring accessibility is considered making users of all types can seamlessly interact with the virtual environment. My approach integrates additional features, such as controller-based navigation for users with additional accessibility requirements. This research also relies on external motion capture devices which restrict their scalability, my research is more scalable in comparison as a solution.

3 Prior Critical Investigations

Recent advancements in robotic programming and simulation have improved the ability to connect virtual environments with physical robotic systems. Unity3D-based digital twins, Robot Web Services (RWS), and Externally Guided Motion (EGM) protocols, provide tools for real-time interactions and visualization. Despite their contributions, they fall short in addressing challenges related to immersion, bi-directional feedback and scalability for task workflows. This section explores existing frameworks and libraries that serve as precedents for integrating Unity3D with robotic systems. By analyzing its contributions and limitations, gaps are identified as opportunities for my research to enhance these frameworks through immersive virtual reality interfaces with the Meta Quest 3. Specifically, this section examines:

1. Digital twin frameworks for robotic visualization (Unity3D_Robotics_ABB).
2. Real-time joint control using EGM protocols (Unity3D_ABB_IRB_120_EGM).
3. Advanced Cartesian control with end-effector integration
(Unity3D_ABB_CRB_15000_GoFa_EGM).
4. Open-source implementations of EGM for Unity (egm-for-abb-robots).

By addressing its limitations, this research builds upon their foundations to develop more immersive and accessible systems for robotic programming in architectural fabrication.

3.1 Unity3D Industrial Robotics: ABB IRB 120

Digital twins have been used in recent robotic simulations to connect virtual and physical operations. The Unity3D_Robotics_ABB framework by Parak, which uses Robot Web Services (RWS) to create a client-server communication between Unity3D and the ABB IRB120 robot, which

is an important precedent. Through a 2D digital twin interface, this makes it possible to monitor joint positions, cartesian coordinates, and diagnostics in real time. It is an important tool for robotics visualization because of its modular architecture, which allows for deployment on PCs and mobile devices connected to the same network [51].

Limitations of Current Work: Although Parak's architecture has monitoring capabilities, its application for iterative workflows is limited due to its lack of immersive interfaces and bi-directional interaction. In particular:

1. Unidirectional Data Flow: Focus on monitoring rather than active control, preventing real time changes.
2. 2D Interface Constraints: Relies on traditional screens instead of VR/AR, missing spatial interaction opportunities [51].
3. No integration with Parametric Tools: Does not include platforms like Grasshopper, which would be crucial for architectural fabrication.

This research addresses these gaps by integrating Meta Quest 3's hand tracking, with a bi-directional RobotStudio communication, extending the previous digital twin application into immersive user-centric robotic workflows.

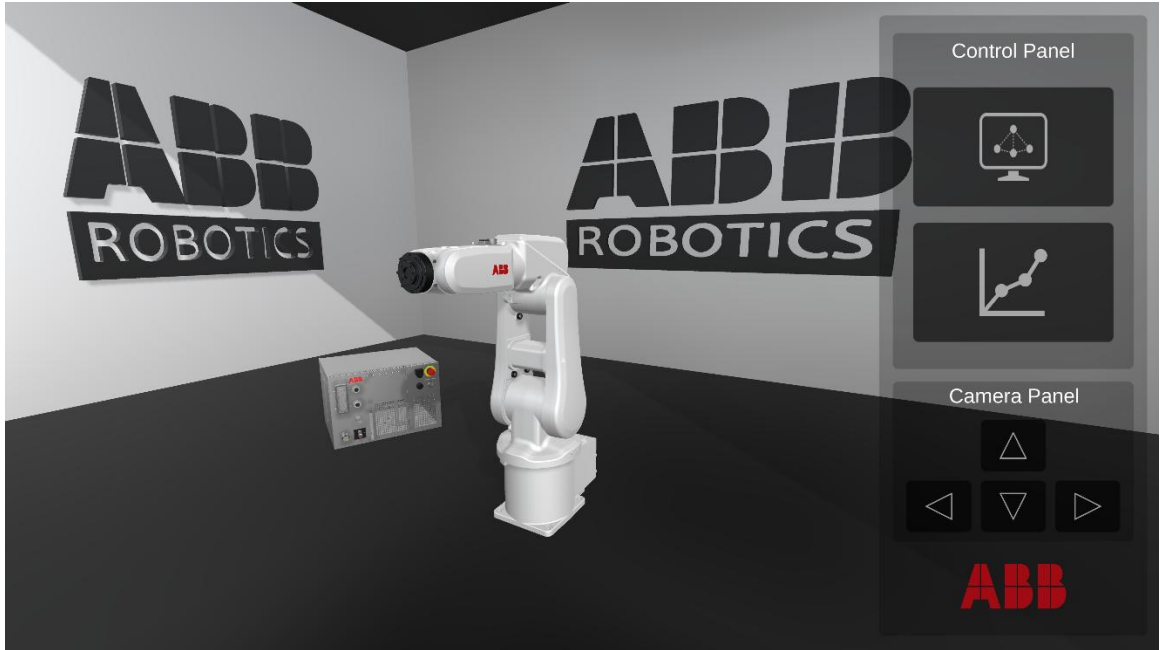


Figure 10: Digital twin interface for ABB IRB 120 in Unity3D.

3.2 Unity3D Industrial Robotics: ABB IRB 120

EGM (Externally Guided Motion)

Building on earlier digital twin frameworks, Parak developed the Unity3D_ABB_IRB_120_EGM project, which introduces Externally Guided Motion (EGM) for real time joint control of the ABB IRB120 robot via UDP communication in Unity3D. This framework allows for direct changes of robotic arm positions through a Unity based interface, supporting both simulation and real-world hardware. The use of Protocol Buffers (Protobuf) for data serialization and the High-Definition Render Pipeline (HDRP) for better visual fidelity, making it an important precedent for real-time robotic control [52].

Key Contributions of Unity3D_ABB_IRB_120_EGM:

- Real-Time EGM Control: Enables joint level adjustments through UDP, moving forward from RAPID programming.
- Cross-Platform Deployment: Deploys on mobile devices, tablets or computers on the same network.
- Protobuf Integration: Enables data exchange using egm.proto files, for compatibility with ABB RobotWare.

Limitations Addressed by This Research:

1. Lack of Immersive Interaction: Relies on 2D interfaces, excluding VR/AR capabilities for spatial manipulation [52].
2. Unidirectional Joint Control: Focus is only on joint coordinates without bi-directional feedback or integration with parametric tools like Grasshopper.
3. Latency in Real-World Testing: Testing alternative to the UDP Unity and RobotStudio pipeline.

By addressing these gaps, the research advances EGM application into architectural fabrication, focusing on accessibility for non-experts and for educational purposes.

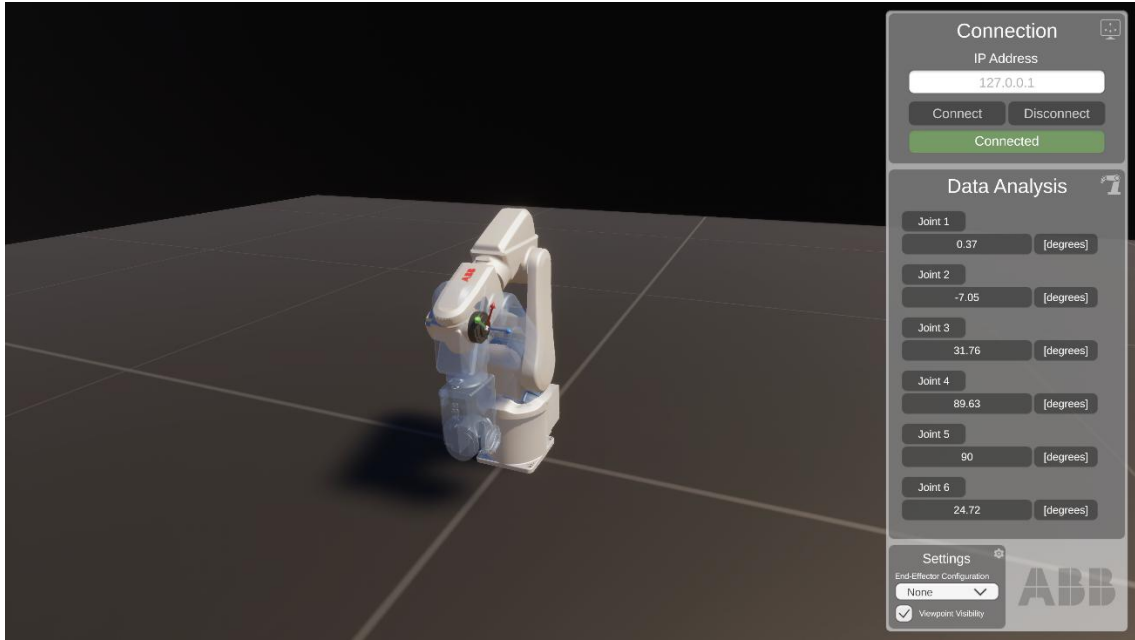


Figure 11: Real-time joint control of ABB IRB 120 via Unity3D EGM framework.

3.3 Unity3D Industrial Robotics: ABB CRB 15000 EGM (Externally Guided Motion)

The progression of digital twins for industrial robots has led to Parak's Unity3D_ABB_CRB_15000_GoFa_EGM project, which expands Externally Guided Motion (EGM) to Cartesian coordinate control of the ABB CRB 15000 GoFa robot and its SCHUNK end-effector via Unity3D. This framework introduces bi-directional UDP/TCP-IP communication allowing for control of the robot's TCP and end-effector while visualizing its motion in Unity. Advancements include Protocol Buffers (Protobuf) for data exchange and integration with HDRP, setting a benchmark for collaborative robotics [53].

Key Contributions of Unity3D_ABB_CRB_15000_GoFa_EGM:

- Cartesian Coordinate Control: Enables precise changes of TCP translation and rotation (Euler/Quaternion) in Unity.
- End-Effector Integration: Synchronizes SCHUNK Co-act EGP-C 40-N-N-GoFa control via TCP/IP.
- DH Parameter Visualization: Provided Blender 3D models for forward kinematics calculations.

Limitations Addressed by This Research:

1. Non-Immersive Interface: Relies on 2D screens, reduces the need for VR/AR for spatial interaction [53].
2. Limited Parametric Integration: Does not include computational design tools like Grasshopper, affecting architectural fabrication.
3. Latency in Multi-Device Communication: Requires firewall adjustments.

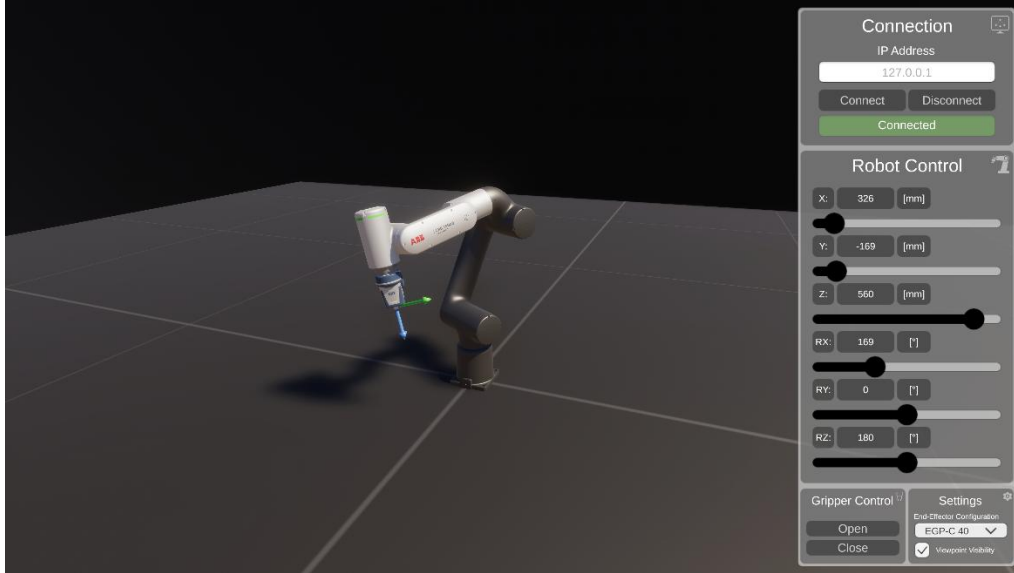


Figure 12: Digital twin of ABB CRB 15000 GoFa with Cartesian coordinate control in Unity3D.

3.4 egm-for-abb-robots

Building on previous digital twin frameworks, the `egm-for-abb-robots` project by Fronchetti provides an implementation in an open-source implementation of EGM for ABB robots in C#. This allows for real-time joint control of ABB robots using a UDP communication, allowing for changes of robotic axes from Unity without needing the ABB PC SDK, which Unity does not support otherwise. The GitHub repository included components like `EgmCommunication.cs` and `Egm.cs`, which facilitate bi-directional communication between Unity and the ABB IRB120 robot. These scripts process joint position updates and take in robotic control for interactivity. This system is essential for this research as it ensures a responsive, low latency connection between virtual and physical, allowing for precise motion tracking [54].

Key Contributions of egm-for-abb-robots:

- Real-Time EGM Control: Implements UDP-based communication to adjust joint configurations.
- Unity Integration: Provides a framework that works with Unity's C# scripting, supporting virtual reality interaction.
- Simplified EGM Implementation: Eliminates the need for RAPID programming by allowing Unity to directly make changes to the robot's movement.

Limitations Addressed by This Research

1. Lack of VR Integration: It focuses on 2D Unity interfaces. This research extends it by incorporating Meta Quest 3's hand tracking for a more immersive robotic control system.
2. Unidirectional Data Flow: The existing framework takes joint movement commands without direct interaction with the robot.
3. Scalability for Architectural Fabrication: The EGM functionality is functional but is not explored in parametric workflows. This research fits right into the architecture pipeline, allowing for geometry driven robotics motion.

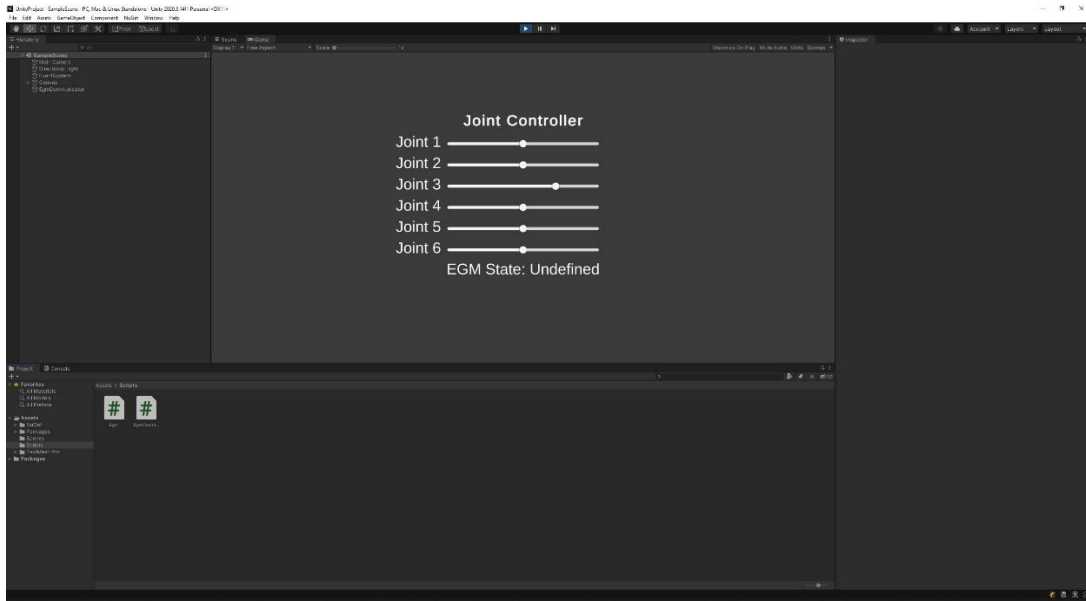
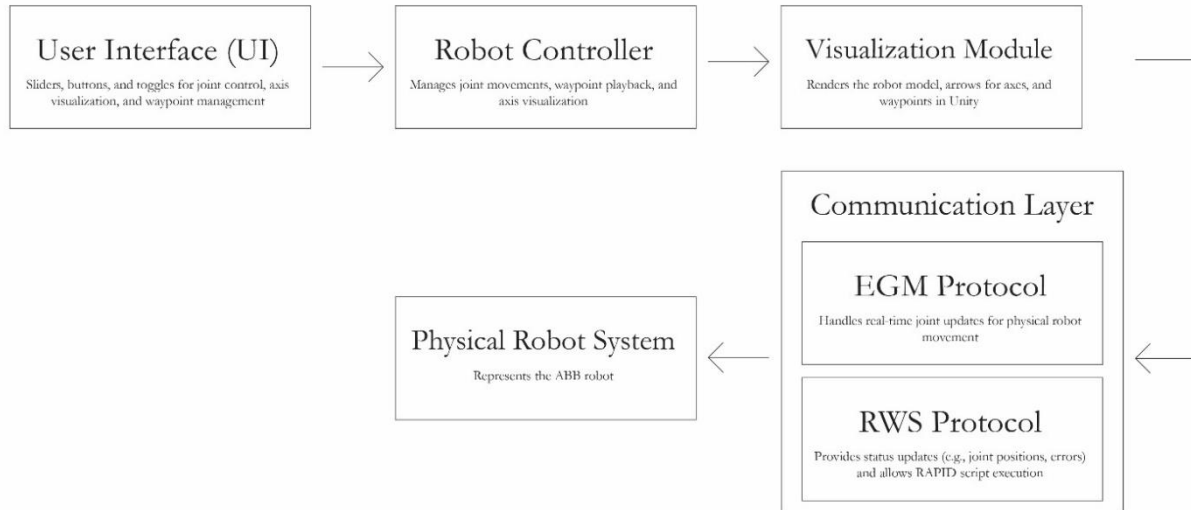


Figure 13: EGM communication interface implemented in Unity using egm-for-abb-robots.

By using egm-for-abb-robots, this research strengthens the bi-directional virtual reality robot interaction, allowing for robotic control being more precise and allowing for its expansion in architecture fabrication.

4 JointLab

The integration of real-time bi-directional interaction methods between Unity, ABB RobotStudio, and Meta Quest 3 with hand and controller tracking features, will add to robotics education and enable users to develop robotic workflows for architectural fabrication by creating immersive user interaction, real-time diagnostics and providing spatially accurate environments.

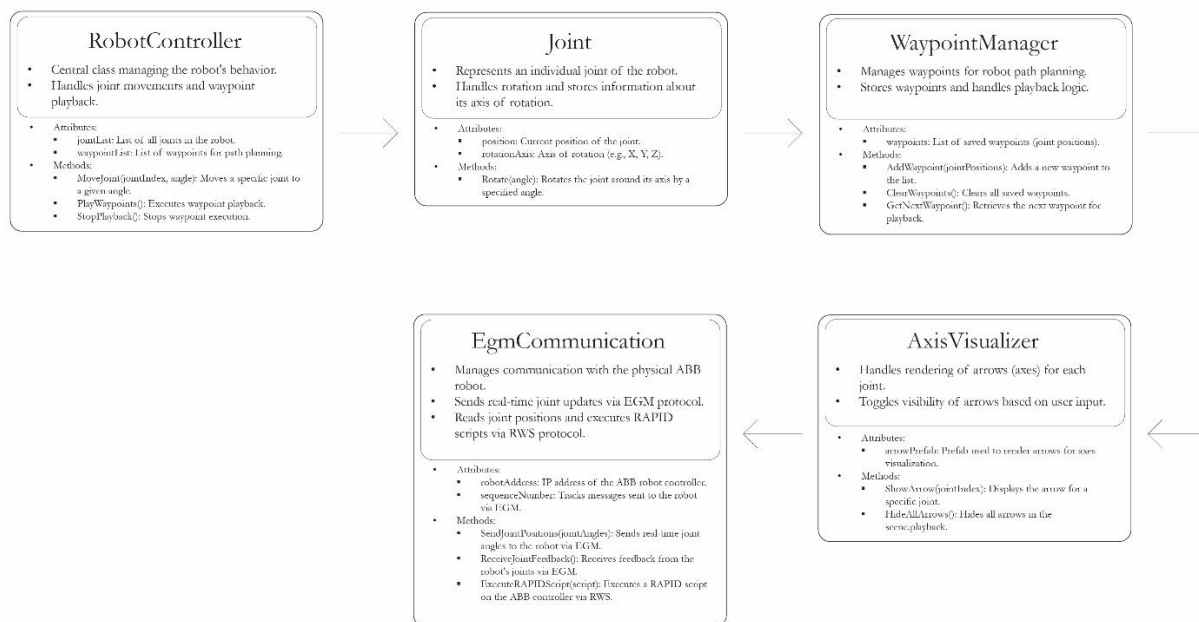


1. Educational Context:

Robotics education often relies on tools like RobotStudio or Grasshopper that are inaccessible to non-experts. The virtual dFab lab provides an environment that is interactive where students can learn robotics without having the risk of damaging physical equipment or requiring technical expertise; By using the Meta Quest 3 with its tracking features, this research provides an accessible starting point for students to explore these robotic workflows interactively.

2. Diagnostic Capabilities:

The pipeline includes a Diagnostic Panel that provides real-time joint position data (J1-J6), Cartesian coordinates (X, Y, Z) and Quaternion orientations (Q1-Q4). This ensures users can monitor the robot during operations, allowing for troubleshooting and adjustments to be made immediately; This is made possible with the RWS component. The EgmCommunication component facilitates live feedback on joint movements which is critical for synchronization between the virtual tool and physical robot. The EGM script facilitates commands like EGMActJoint and EGMRunJoint in RobotStudio for enabling a real-time connection with Unity for virtual simulations.



3. Connection Panel:

This panel simplifies communication by managing IP connections between Unity and RobotStudio.

This ensures reliable data exchange for EGM and RWS commands to execute properly.

4. Camera Panel:

This panel enhances spatial visualization by allowing users to toggle between predefined views. This helps in quick shifting between different views, debugging and refining robotic paths within the virtual environment.

5. Immersive Interaction:

The Meta Quest 3 with its controller and hand tracking capabilities provides a natural way to interact with interfaces for controlling robotic joints and axes. Combined with EGM, this creates a better experience for experts and novices.

6. Spatial Alignment

Misaligned coordinate systems are addressed by using EGM's script to handle joint-level corrections ensuring accurate alignment between Unity and RobotStudio.

This research not only builds upon existing frameworks but also introduces solutions to optimize real-time interaction and scalability. By incorporating EGM and RWS it sets the stage for robotics education and architectural fabrication workflows. The methodology section will outline detailed steps taken to implement this framework and evaluate its effectiveness.

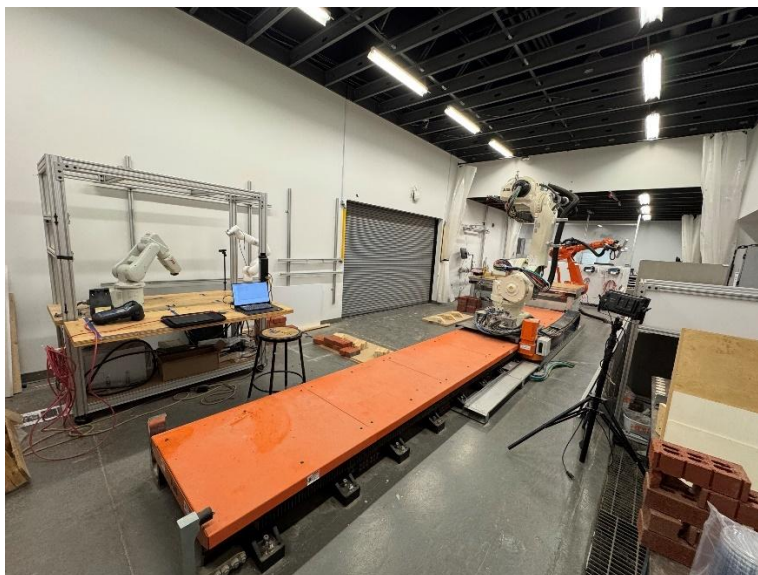




Figure 14: Physical layout of CMU's dFab Lab, showing ABB industrial robots and linear track system.

4.1 Digital Recreation of the dFab Lab

Environment

The virtual recreation of Carnegie Mellon University's dFab Lab serves as the foundational environment for testing and developing robotic workflows based on real-time feedback. This section highlights the implementation process focused on spatial accuracy, interactivity, and the integration of robotic systems within Unity.

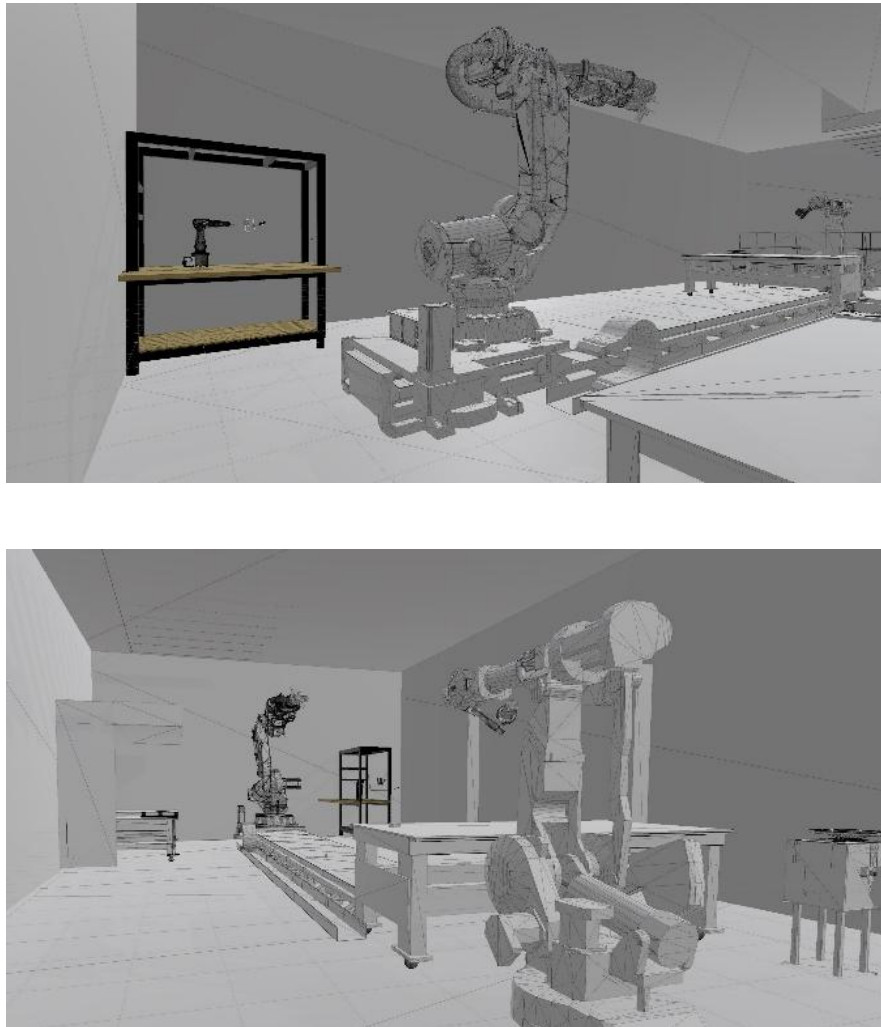


Figure 15: Wireframe view of the virtual dFab Lab, replicating the physical lab layout for simulation purposes.

4.1.1 Spatial Mapping and Virtual Environment Implementation

The virtual dFab Lab environment was created in Unity to provide a realistic and immersive setting for robotic workflow simulations. The implementation involved critical steps:

1. Spatial Mapping and Accuracy:

The virtual environment uses an existing Rhino CAD model of the dFab Lab imported into Unity. Physical measurements were taken to ensure the CAD model is accurately scaled where 1 Unity Unit equals 1 meter. The precise mapping allows users to align virtual elements with their physical counterparts, keeping headset tracking accurate.

2. Visual Elements:

PBR (Physically based rendering) techniques were used to simulate realistic lighting and materiality. Environmental elements such as planes, sky and sun were added to create a complete visual experience.

3. Navigation and Interaction:

A camera system is established with predefined viewpoints and allows users to observe the environment from multiple angles. The Meta Quest 3 with its hand tracking and controller-based capabilities are used for an intuitive interaction with the lab. Custom UI elements discussed later will provide contextual information and control within the virtual space.



Figure 16: Virtual workstation modeled in Unity based on the physical ABB IRB 120 setup.

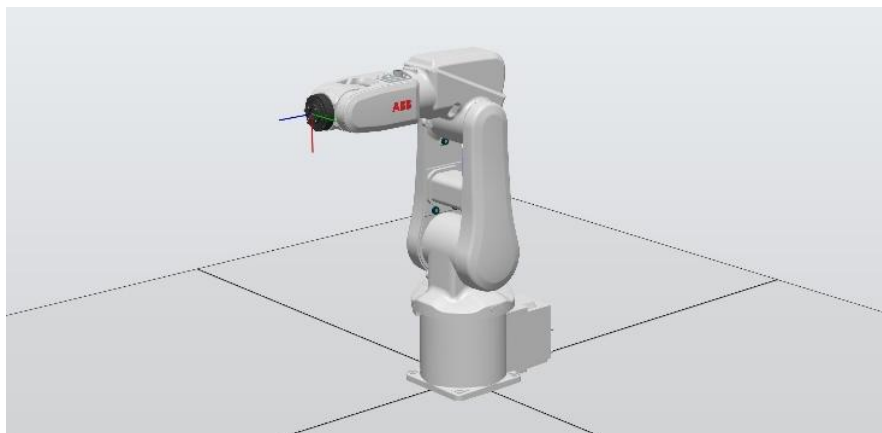


Figure 17: ABB IRB 120 robot model visualized in ABB RobotStudio.

4.1.2 IRB 120 Robotic Workstation: Architecture and Control Systems

The workstation area represents the primary interaction zone within the virtual dFab lab, centered around the ABB IRB 120 robot. This section was developed with particular attention to functionality and user accessibility:

1. Workstation components:

Accurately positioned work surface matching physical lab's layout. Interactive control panels discussed later providing robot diagnostic data and manipulation interfaces. Virtual tools and materials are positioned for easy access during robotic workflow simulations.

2. Interface Design:

Includes Connection Panel, Diagnostic Panel, Joint Control Panel, Camera Panel, and Waypoint Panel.

3. ABB IRB 120 robot

The centerpiece of the workstation is the ABB IRB 120 robot, a compact 6-axis industrial robot with high precision capabilities. In our virtual implementation:

- Robot Specifications:

6-axis articulated robot with 3 kg payload capacity, 50 mm reach making it ideal for small scale fabrication tasks, and a compact footprint that's suitable for educational environments.

- Virtual Model Structure:

The robot hierarchy consists of:

Control

```

└ irb_120_object
  └ IRB_Link1
    └ IRB_Link2
      └ IRB_Link3
        └ IRB_Link4
          └ IRB_Link5
            └ IRB_Link6
              └ Viewpoint_EE_0
                └ AX_X
                  └ AX_Y
                    └ AX_Z
                      └ Viewpoint

```

- Joint Control Implementation:

The robot's movement in Unity is controlled through a set of scripts attached to each link, mirroring the movements of either a virtual or physical ABB IRB 120 robot in RobotStudio. Each link script (irb120_link1.cs through irb120_link6.cs) handles one specific robot joint:

```

// Example from irb120_link1.cs

void FixedUpdate()
{
    try
    {
        transform.localEulerAngles = new Vector3(0f, 0f, (float)(
EgmCommunication.GetJointPosition(0)));
    }
}

```

```

        catch (Exception e)
        {
            Debug.Log("Exception:" + e);
        }
    }
}

```

Each joint script rotates its respective link based on data received via the EgmCommunication system, with specific rotation axes configured according to the physical robot's joint movement capabilities.

- Data Processing Script:

Attached to the “Control” object, the abb_data_processing.cs script serves as the communication between Unity and the ABB robot. This script:

Establishes a two-way communication using ABB's Robot Web Services (RWS). Processes both XML and JSON data streams for different data types: XML for joint target positions (J1-J6), JSON for Cartesian coordinates and quaternion orientations. Handles connection, disconnection, and thread management. Provides real-time data at configurable update dates (adjustable time steps).

This implementation allows for

```

// Connection initialization
ABB_Stream_Data_XML.ip_address = "127.0.0.1";
ABB_Stream_Data_XML.xml_target = "jointtarget";
ABB_Stream_Data_XML.time_step = 2;

// Data access from other scripts
ABB_Stream_Data_XML.J_Orientation[0]; // Access Joint 1 position

```

This complete recreation of the dFab lab workstation, centered around the ABB IRB 120, provides an immersive and functional environment for testing robotic workflows and educational purposes.

The bi-directional communication system ensures that movements started in the virtual environment can be mirrored on the physical robot, and vice versa, creating a seamless integration between virtual and physical fabrication.

4.2 UI Implementation

A key component of JointLab is its user interface, which serves as the primary point for interaction between the user, the virtual environment and the robotic system. Designed for both experts and novices, the UI prioritizes clarity, and functionality-bridging the gap between complex robotic programming and real-time control. This section details the structure and implementation of the various UI elements developed within Unity, to streamline specific aspects of the user experience.

From establishing and managing robot communications to visualizing spatial data and controlling robotic joints. These panels form an integrated interface that supports immersive learning, manipulation, and transitions between simulation and physical execution. The following subsections outline each panel's purpose, philosophy and execution, and its role within the overall system architecture.

4.2.1 Connection Panel

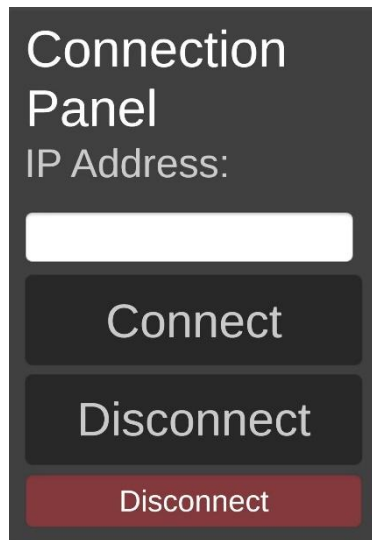


Figure 18: Connection Panel UI for managing IP setup and communication with RobotStudio.

The Connection Panel forms the foundation for the human-robot interaction pipeline, enabling users to establish and monitor communication between Unity and the ABB RobotStudio environment. This panel was designed to provide clear visual feedback and an easy-to-understand connection management system.

- Design Philosophy and Structure

The connection panel was developed with three principles:

1. Simplicity: minimizing complexity for new users.
2. Visual feedback: providing clear indicators for connection status.

3. Accessibility: easy access between desktop and VR.

The panel consists of hierarchically organized UI elements:

Title section displaying “Connection Panel”, IP address input field for specifying robot controller address, Connect/Disconnect button for managing connection state, and Visual indicator for showing connection status with color.

- Technical Implementation

The Connection Panel using Unity’s Canvas system with custom C# scripts, based on the hierarchy, the panel contains nested elements:

```
Connection_Panel
├─ Title
├─ IP
├─ ip_adr_input
├─ Connect_btn
├─ disconnect_btn
└─ connection_info_img
```

The implementation handles connection management through the main_ui_control.cs script, which initializes the panel as follows:

```
// Connection information {image} -> Connect/Disconnect
connection_info_img.GetComponent().color = new Color32(255, 0, 48,
50);
connectionInfo_txt.text = "Disconnect";
```

```
// Robot IP Address
```

```
ip_address_txt.text = "127.0.0.1";
```

- Connection State Management

The panel uses a state management system with visuals that indicate its connection status:

Disconnected State displays a red color with Disconnect text.

```
connection_info_img.GetComponent().color = new Color32(255, 0, 48, 50);
```

```
connectionInfo_txt.text = "Disconnect";
```

Connected State displays a green color with Connect text.

```
connection_info_img.GetComponent().color = new Color32(135, 255, 0, 50);
```

```
connectionInfo_txt.text = "Connect";
```

- IP Address Configuration

The panel includes an input field for specifying the address of the ABB robot controller:

```
// Robot IP Address (Read) -> XML thread function
```

```
abb_data_processing.ABB_Stream_Data_XML.ip_address =  
ip_address_txt.text;
```

```
// Robot IP Address (Write) -> JSON thread function
```

```
abb_data_processing.ABB_Stream_Data_JSON.ip_address =  
abb_data_processing.ABB_Stream_Data_XML.ip_address;
```

This implementation allows the input IP address to be used for both XML-based communication (for joint target positions) and JSON-based communication (for Cartesian coordinates and quaternion orientations).

- Communication Protocol Selection

The Connection Panel also manages selected the communication protocol between RWS (Robot Web Services) and EGM (Externally Guided Motion) modes:

```
public void TaskOnClick_ConnectBTN()
{
    abb_data_processing.GlobalVariables_Main_Control.connect = true;
    abb_data_processing.GlobalVariables_Main_Control.disconnect =
false;
    isRwsActive = true; // Activate RWS mode
    isEgmActive = false; // Deactivate EGM mode
}

public void SwitchToEgmMode()
{
    isRwsActive = false; // Deactivate RWS mode
    isEgmActive = true; // Activate EGM mode
}
```

This dual approach allows for flexibility between different interactions:

RWS for status monitoring and non-real time control, and EGM for low latency real time joint and position control.

- Accessibility Considerations

To enhance accessibility in the VR environment, the connection panel includes:

Visual positioning with proper spacing for interaction with the Quest 3, persistent visuals display connection status always, and toggle visuals show/hide the panel when needed.

```
public void TaskOnClick_ConnectionBTN()
{
    // visible on
    connection_panel_img.transform.localPosition = new Vector3(0f, 0f,
0f);
    // visible off
    diagnostic_panel_img.transform.localPosition = new Vector3(780f +
(ex_param), 0f, 0f);
}
```

The connection panel represents the first step in establishing the bi-directional pipeline between virtual and physical robotic systems. Its real time control and feedback are essential for JointLab's framework.

4.2.2 Diagnostic Panel

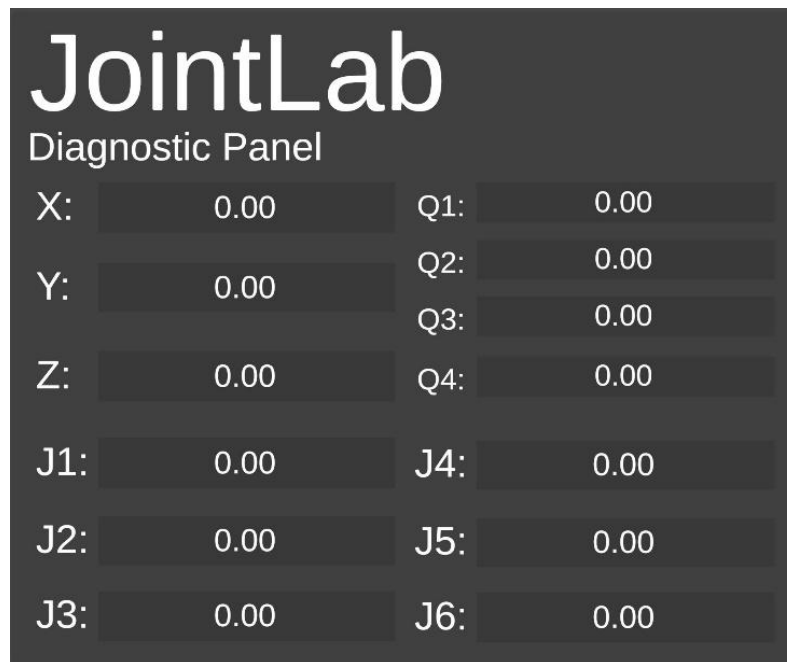


Figure 19: Diagnostic Panel UI displaying real-time joint angles, position, and orientation data.

The diagnostic panel represents a critical component of the JointLab interface, providing users with real-time feedback on robot positioning, orientation and joint configurations. This monitoring system enables precise tracking of robot states during operation, supporting educational and workflow optimization.

- Design Philosophy and Structure

The diagnostic panel was designed with three primary objectives:

1. Comprehensive Data Visualization: all essential robot parameters presented in a readable format.
2. Real-Time Feedback: minimal latency between robotic movements and UI.

3. Contextual Organization: grouping related parameters for intuitive understanding.

The panel's hierarchical structure includes:

```
Diag_Panel
├─ Title
├─ Title
├─ x_pos
├─ y_pos
├─ z_pos
├─ q1_pos
├─ q2_pos
├─ q3_pos
├─ q4_pos
├─ j1_pos
├─ j2_pos
├─ j3_pos
├─ j4_pos
├─ j5_pos
└─ j6_pos
```

This structure is organized into logical groups:

Cartesian Position (x_pos, y_pos, z_pos): current end-effector position in world coordinates,

Quaternion Orientation (q1_pos, q2_pos, q3_pos, q4_pos): end-effector orientation using

quaternion representation, and Joint Values (j1_pos through j6_pos): angular position of each robot joint.

- Technical Implementation

The diagnostic panel leverages Unity's UI system with TextMeshPro components for high quality rendering. This implementation is managed by main_ui_control.cs script, which handles data and visual updates:

```
// Position {Cartesian} -> X..Z

position_x_txt.text =
((float)Math.Round(abb_data_processing.ABB_Stream_Data_JSON.C_Position
[0], 2)).ToString();

position_y_txt.text =
((float)Math.Round(abb_data_processing.ABB_Stream_Data_JSON.C_Position
[1], 2)).ToString();

position_z_txt.text =
((float)Math.Round(abb_data_processing.ABB_Stream_Data_JSON.C_Position
[2], 2)).ToString();

// Position {Rotation} -> Quaternion(q1..q4)

position_q1_txt.text =
((float)Math.Round(abb_data_processing.ABB_Stream_Data_JSON.C_Orientat
ion[0], 6)).ToString();

// ...additional quaternion values...

// Joint positions updated through dedicated method
public void UpdateDiagnosticPanel(double[] jointValues)
{
```

```

        position_j1_txt.text = ((float)Math.Round(jointValues[0],
2)).ToString();

        // ...remaining joint values...
    }

```

Key features of the implementation:

1. Precise Control: values rounded to decimal places (2 for positions, 6 for quaternions).
 2. Real-time updates: values are updated in the FixedUpdate() method for consistent refresh rates.
 3. Bi-directional Data Flow: panel displays values from both RWS and EGM communication system.
- Visual Design and User Experience

Users can show/hide the panel as needed using UI controls.

```

public void TaskOnClick_DiagnosticBTN()

{

    // visible on

    diagnostic_panel_img.transform.localPosition = new Vector3(0f, 0f,
0f);

    // visible off

    connection_panel_img.transform.localPosition = new Vector3(1215f +
(ex_param), 0f, 0f);

}

```

- Integration with Robotic Systems

The diagnostic panel is an essential interface for users:

1. Facilitates debugging: users can identify discrepancies between intended and actual robot positions.
2. Supporting Learning provides immediate feedback on control inputs affecting the robot.
3. Validating Operations: confirms the communication Unity and the robot is functioning.

The panel receives data through two main channels:

1. ABB_Stream_Data_JSON: processes Cartesian positions and quaternion orientations from RobotStudio.
2. Direct joint value updates: receives joint angles updated through the UpdateDiagnosticPanel() method.

By providing comprehensive real-time feedback on all robot parameters the diagnostic panel enables users to understand the complex relationship between joint configurations and end-effector positioning, supporting both their educational goals and precision required for architectural fabrication tasks.

4.2.3 Camera Panel

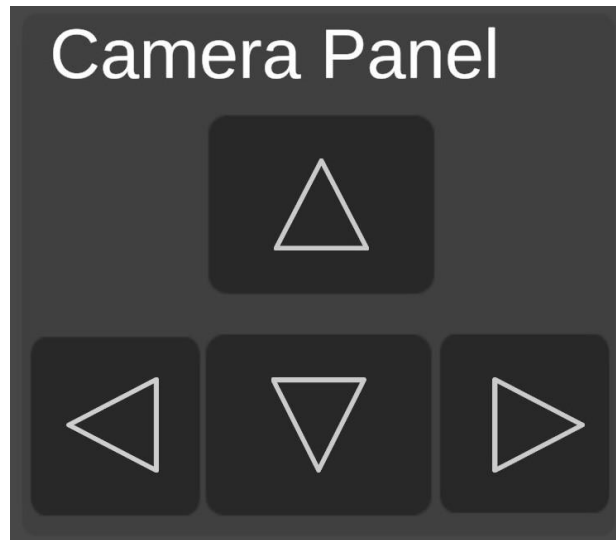


Figure 20: Camera Panel UI for switching between predefined viewpoints in the virtual environment.

The Camera Panel represents a navigation component for the interface, enabling users to observe and interact with the environment from multiple perspectives.

- Design Philosophy and Structure

The Camera Panel is designed with three principles in mind:

1. Spatial Awareness: provides users with quick access to key viewpoints for spatial understanding.
2. Simplified Navigation: cognitive load reduced through intuitive direction control.
3. Workflow Enhancement: debugging and precision is achieved through strategic camera positioning.

The panel's structure consists of:

Camera_Panel

└ Title

└ Button_L

└ Button_T

└ Button_H

└ Button_R

The structure employs directional controls:

Left View (Button_L): perspective from the left side, Top View (Button_T): overhead perspective for workflow planning, Home View (Button_H): returns to the front-facing position, Right View (Button_R): shows the robot from the right side.

- Technical Implementation

The camera panel uses Unity's UI system with custom scripts to manage camera positioning. The implementation is controlled using the `main_ui_control.cs` script with specific functions for each viewpoint:



Figure 21: Right-side camera view of the ABB IRB 120 in the virtual dFab Lab.

```
// Right view positioning  
public void TaskOnClick_CamViewRBTN()  
{  
    camera_obj.transform.localPosition = new Vector3(0.114f, 2.64f, -  
2.564f);  
    camera_obj.transform.localEulerAngles = new Vector3(10f, -30f,  
0f);  
}
```



Figure 22: Left-side camera view of the ABB IRB 120 in the virtual dFab Lab.

```
// Left view positioning  
public void TaskOnClick_CamViewLBTN()  
{  
    camera_obj.transform.localPosition = new Vector3(-3.114f, 2.64f, -  
2.564f);  
    camera_obj.transform.localEulerAngles = new Vector3(10f, 30f, 0f);  
}
```



Figure 23: Front (home) camera view of the ABB IRB 120 in the virtual dFab Lab.

```
// Home (front) view positioning  
public void TaskOnClick_CamViewHBTN()  
{  
    camera_obj.transform.localPosition = new Vector3(-1.5f, 2.2f, -  
3.5f);  
    camera_obj.transform.localEulerAngles = new Vector3(0f, 0f, 0f);  
}
```

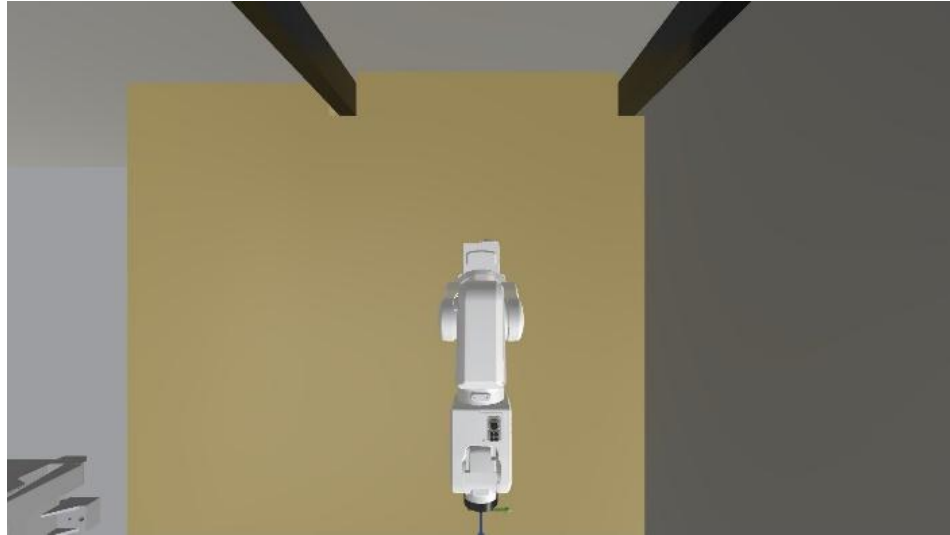


Figure 24: Top-down camera view of the ABB IRB 120 in the virtual dFab Lab.

```
// Top view positioning
public void TaskOnClick_CamViewTBTN()
{
    camera_obj.transform.localPosition = new Vector3(-1.5f, 4.5f, 0f);
    camera_obj.transform.localEulerAngles = new Vector3(90f, 0f, 0f);
}
```

Each function positions the camera at predetermined coordinates with specific rotation angles for maximum visibility and awareness of its surroundings.

- Integration with Robotics Workflow

The camera panel serves functions within the broader workflow:

1. Task verification: users can quickly verify robot positioning from multiple angles.
2. Error Identification: multiple viewpoints identify potential collisions or issues in planned paths beforehand.

3. Teaching Enhancement: supports better understanding of robot kinematics for educational perspectives.
4. Workflow Documentation: standard views provide consistency for documentation and comparison.

By providing intuitive navigation through the virtual dFab lab environment, the camera panel enhances the educational and practical aspects of the JointLab framework.

4.2.4 Joint Controller

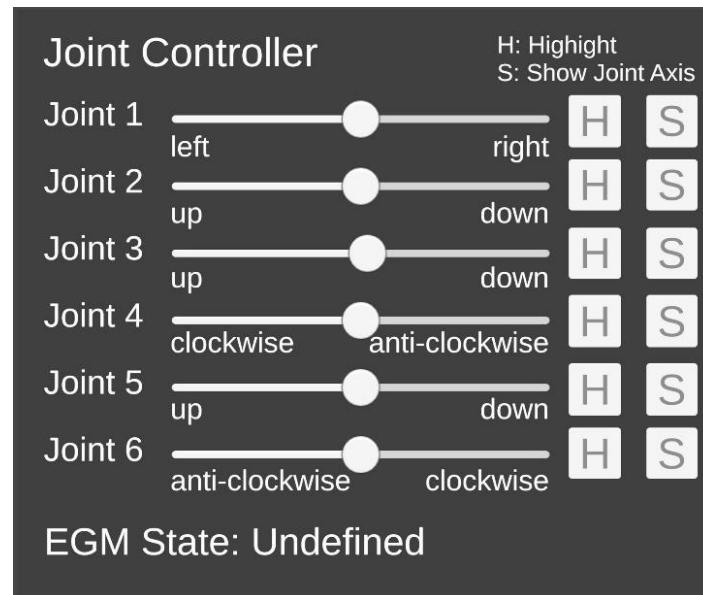


Figure 25: Joint Controller UI for real-time manipulation of individual robot joints and axis visualization.

The Joint controller represents the primary interaction mechanism for direct manipulation of the ABB IRB 120 robot within the virtual environment. This specialized interface component provides intuitive control of each robot joint through interactive sliders, visual feedback systems and real-time state monitoring features.

- Design Philosophy and Structure

The joint controller was designed with three fundamental principles:

1. Direct Manipulation: enabling precise control of individual robot joints using sliders.
2. Visual Feedback: clear directional cues and highlighting to enhance user understanding.
3. Educational Value: visualizing movement axes and joint relationships to support learning.

The hierarchical structure of the Joint Controller comprises:

JointController

- ├ J1Slider through J6Slider
- ├ Joint 1 (Text) through Joint 6 (Text)
- ├ Highlight1 through Highlight6 (H buttons)
- ├ Show1 through Show6 (S buttons)
- ├ Joint direction text elements

└ EGM State (Text)

└ Title

- Technical Implementation

The joint controller integrates with the EgmCommunication.cs script to establish bi-directional communication between the UI sliders and the robot's joint positions. The implementations with sliders for direct manipulation are as follows:

```
// Slider event listeners for joint control
j1Slider.onValueChanged.AddListener(delegate {
    SendJointsMessageToRobot(j1Slider.value, j2, j3, j4, j5, j6);
});
j2Slider.onValueChanged.AddListener(delegate {
    SendJointsMessageToRobot(j1, j2Slider.value, j3, j4, j5, j6);
});
// Additional slider listeners for joints 3-6
```

Each slider represents one of the six joints of the ABB IRB 120 robot, with the value range corresponding to the physical limitations of each joint:

1. Joint 1: Rotation around vertical axis (-165° to $+165^{\circ}$) with "left/right" direction indicators
2. Joint 2: Shoulder movement (-110° to $+110^{\circ}$) with "up/down" direction indicators
3. Joint 3: Elbow movement (-110° to $+70^{\circ}$) with "up/down" direction indicators
4. Joint 4: Wrist rotation (-160° to $+160^{\circ}$) with "clockwise/anticlockwise" direction indicators
5. Joint 5: Wrist bend (-120° to $+120^{\circ}$) with "up/down" direction indicators
6. Joint 6: Tool rotation (-400° to $+400^{\circ}$) with "clockwise/anticlockwise" direction indicators

The directional text under each slider provides contextual guidance for each user, indicating the expected movement the robot would make when manipulating each joint., enhancing the intuitiveness of the interface for novice users.

- Joint Visualization Features

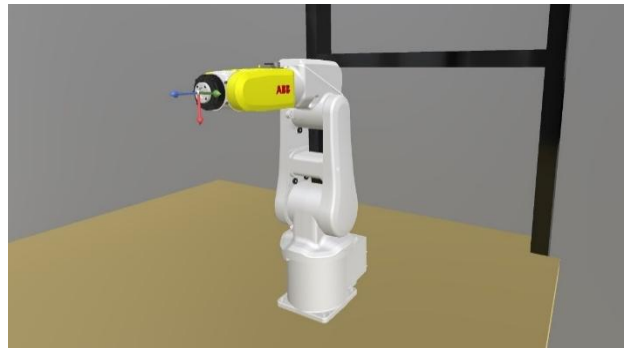
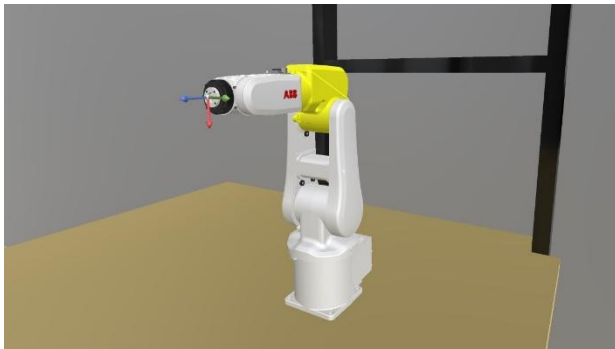
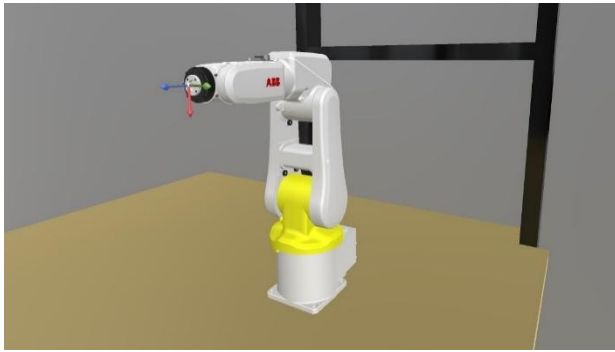


Figure 26: Joint highlighting functionality showing selected joint with visual emphasis for teaching or debugging.

Highlight Functionality (H Buttons)

The Joint controller implements a visual highlighting feature that allows users to identify specific joints with a single click, when activated through the “H” buttons, the corresponding joint changes material to provide clear visual understanding:

```
public void HighlightJoint(GameObject joint)
{
    // Cancel any pending resets
    if (resetHighlightCoroutine != null)
    {
        StopCoroutine(resetHighlightCoroutine);
    }

    ResetAllJoints();

    Renderer renderer = joint.GetComponentInChildren<Renderer>();
    if (renderer != null)
    {
        Material[] materials = renderer.materials;
        for (int i = 0; i < materials.Length; i++)
        {
            if (materials[i].name.Contains("Mat4"))
```

```

    {
        materials[i] = highlightMaterial;
    }
}

renderer.materials = materials;
}
}

```

The highlighting feature serves multiple purposes:

Isolating specific joints for focused attention during teaching/learning, and supports troubleshooting allowing users to verify which joint is being controlled.



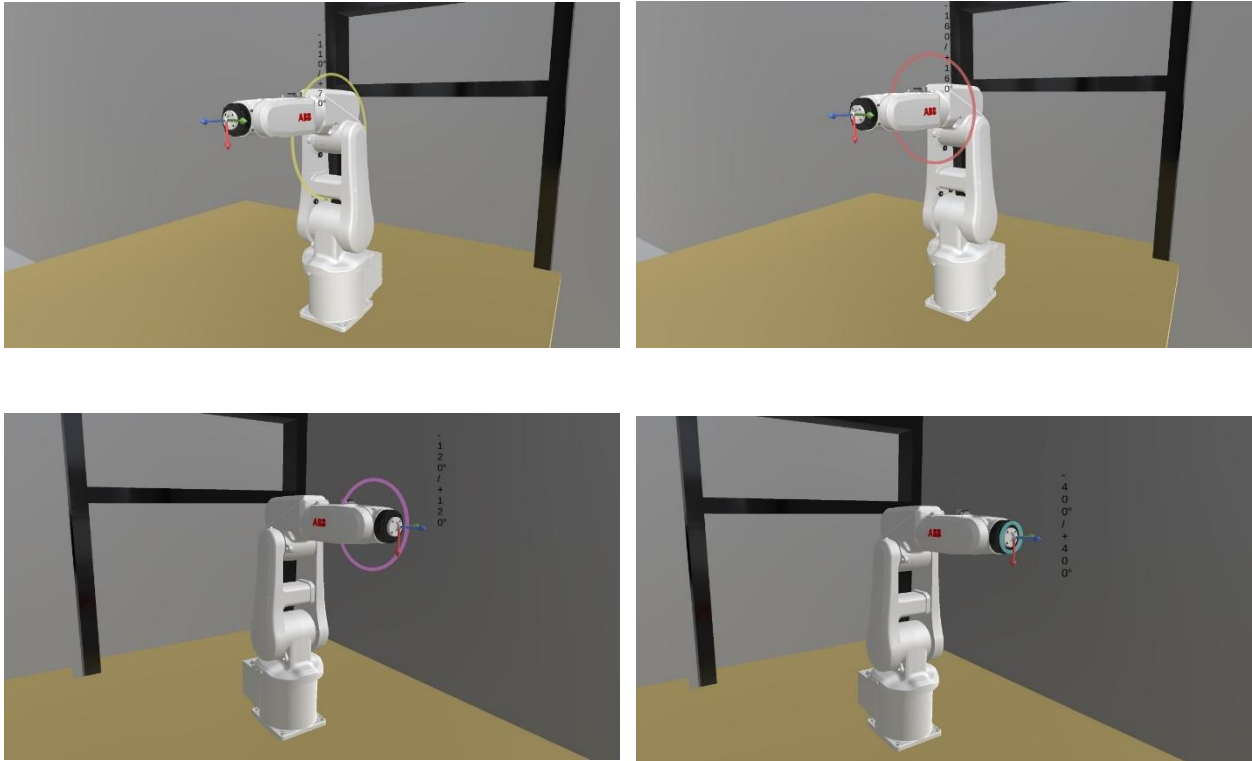


Figure 27: Axis visualization using rotation discs to indicate range and orientation for each joint.

Show Joint Axis Functionality (S Buttons)

The "S" buttons activate a visualization system that shows the rotation axis of each joint using custom-rendered discs. This feature provides critical information about each rotation in space:

```
private void CreateJointDiscs()
{
    jointDiscVisuals = new GameObject[joints.Length];
    for (int i = 0; i < joints.Length; i++)
    {
        GameObject discParent = new GameObject($"Joint{i + 1}_Disc");
```

```

        discParent.transform.SetParent(joints[i].transform, false);

        discParent.transform.localPosition = Vector3.zero;

        discParent.transform.localRotation = GetDiscRotation(i);

        string rangeText = GetRotationRangeText(i);

        float dynamicRadius = CalculateDiscRadius(joints[i]);

        CreateRotationDisc(discParent.transform, dynamicRadius,
        GetDiscColor(i), rangeText);

        jointDiscVisuals[i] = discParent;

        discParent.SetActive(false);
    }
}

```

The joint axis visualization includes key features:

Color coded unique discs identifying each joint (soft blue for Joint 1, sage green for Joint 2, etc.), Range text displaying minimum and maximum rotation values for each joint, proper orientation of each disc to represent the joint's rotation axis, and dynamic sizing based on the joint's dimensions for optimal visibility.

- EGM State Monitoring

The joint controller features an EGM State indicator that provides real-time feedback on the communication status between Unity and the ABB robot controller:

```

void Update()
{
    egmStateText.text = "EGM State: " + egmState;

    if (server.Available > 0) // Check if new data is available

```

```

{
    UpdateSlidersWithJointValues();
}
}

```

The EGM states include:

1. Undefined: initial state before connection.
2. Running: Active communication with the controller.
3. Stopped: Communication is stopped.
4. Error: Communication has an error.

This real-time status indicator is critical for users to understand whether manipulations are being successfully communicated to the physical robot or RobotStudio.

- Bi-directional Data Flow

A key feature of the joint controller is its bi-directional data flow. When users manipulate sliders, the sliders changes are sent to the robot through the `SendJointsMessageToRobot` method:

```

private void SendJointsMessageToRobot(double j1, double j2, double j3,
double j4, double j5, double j6)
{
    using (MemoryStream memoryStream = new MemoryStream())
    {
        EgmSensor message = new EgmSensor();
        CreateJointsMessage(message, j1, j2, j3, j4, j5, j6);
        message.WriteTo(memoryStream);
    }
}

```

```

        int bytesSent = server.Send(memoryStream.ToArray(),
(int)memoryStream.Length, robotAddress);
    }
}

```

Conversely, when the robot moves physically or in simulation, the joint positions are received, and the sliders are updated for the current state:

```

private void UpdateSlidersWithJointValues()
{
    var bytes = server.Receive(ref robotAddress);

    if (bytes != null)
    {
        EgmRobot message = EgmRobot.Parser.ParseFrom(bytes);

        ParseJointValuesFromMessage(message);

        j1Slider.value = (float)j1;
        j2Slider.value = (float)j2;
        j3Slider.value = (float)j3;
        j4Slider.value = (float)j4;
        j5Slider.value = (float)j5;
        j6Slider.value = (float)j6;
    }
}

```

The bi-directional flow ensures the virtual representation remains in sync with the physical robot, allowing for real-time feedback and adjustments.

- Integration with Educational Objectives

The joint controller serves as a fundamental educational tool within the JointLab framework by:

1. Visualizing Abstract Concepts: making rotation axes and joint relationships visual.
2. Providing Tactile Learning: enabling hands-on manipulations of robotic joints.
3. Supporting Experimentation: allowing safe experimentation of joint limits and movement.
4. Facilitating Troubleshooting: help identify issues in robotic workflows through visuals.

By combining intuitive controls with visual feedback and real-time status, the joint controller bridges the gap between abstract robotic concepts and practical understanding.

4.2.5 EGM Communication Framework

The EGM communicator is a core component of the framework, enabling bi-directional communication between Unity and ABB robots.

- Technical Implementation

The `EgmCommunication.cs` script establishes low-latency UDP communication between Unity and ABB robots:

```
/* UDP port where EGM communication should happen (specified in RobotStudio) */
```

```
public static int port = 6511;
```

This script implements essential features:

1. Protocol Buffer Integration

The system uses Google's Protocol Buffers (defined in Egm.cs) for data serialization, enabling message exchange between Unity and the robot:

```
using Abb.Egm;
```

```
using Google.Protobuf;
```

2. Bi-directional Data Flow

The communicator handles both:

Outbound data: sending joint position commands from Unity to the robot and Inbound data: receiving real-time joint positions and state information from the robot.

```
private void SendJointsMessageToRobot(double j1, double j2, double j3, double j4, double j5, double j6)
```

```
{
```

```
    // Code that formats and sends joint positions via UDP
```

```
}
```

```
private void UpdateSlidersWithJointValues()
```

```
{
```

```
    // Code that receives and parses robot feedback
```

```
}
```

3. Integration with RobotStudio

To enable this communication, a RAPID script runs on the robot controller that:

Initializes EGM communication, configures parameters for joint-based control, establishes error handling and loop management.

```
PROC EGM_JOINT_CONTROL()  
  
    EGMSetupUC ROB_1, egm_id, "default", "ROB_1", \Joint;  
  
    WHILE TRUE DO  
        EGMActJoint egm_id, [...parameters...]  
        EGMRunJoint egm_id, EGM_STOP_HOLD, [...parameters...]  
        WaitTime 1.0;  
    ENDWHILE  
ENDPROC
```

4. Joint Control System

The script connects UI sliders to robot movements allowing manipulation:

```
j1Slider.onValueChanged.AddListener(delegate {  
    SendJointsMessageToRobot(j1Slider.value, j2, j3, j4, j5, j6);  
});
```

4.2.6 Waypoint Panel

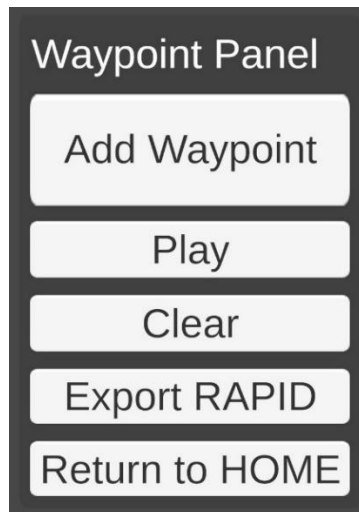


Figure 28: Waypoint Panel UI for recording, playing, and exporting robot motion paths.

The waypoint panel enables users to create, test and export complex robotic movement sequences.

This interface bridges the gap between intuitive VR-based manipulation and production ready robotic programming, serving as both an educational tool and practical workflow development system.

- Design Philosophy and Structure

The waypoint panel has three fundamental objects:

1. Path creation: enabling non-experts to develop complex trajectories.
2. Workflow testing: immediate visual and physical feedback on created paths.
3. Production Integration: seamlessly translate virtual movements into executable robot code.

The panel's hierarchical structure in Unity consists of:

```
Waypoint_Panel
├── Title (1)
├── Button (Add Waypoint)
├── Button (1) (Play)
├── Button (2) (Clear)
├── Button (3) (Export RAPID)
```

└ Button (4) (Reset Robot)

This design focuses on core path creation functioning through clear labelled buttons with distinct purposes, accessibility being ensured for novices and experts.

- Technical Implementation

The waypoint panel leverages the EgmCommunication.cs script to manage waypoint creation, playback, and export. This implementation provides elegant solutions translating joint manipulations into reusable robotic paths:

```
public void AddWaypoint()
{
    double[] currentWaypoint = { j1, j2, j3, j4, j5, j6 };
    waypoints.Add(currentWaypoint);
    // Use the end effector's position as the waypoint position
    Vector3 waypointPosition = endEffectorTransform.position;
    // Instantiate a visual marker for the waypoint
    GameObject waypointInstance = Instantiate(waypointPrefab,
    waypointPosition, Quaternion.identity);
    waypointInstance.tag = "Waypoint";
    // Update line renderer positions
    UpdateLineRenderer();
}
```

The panel features five key features that form a complete path creation workflow:

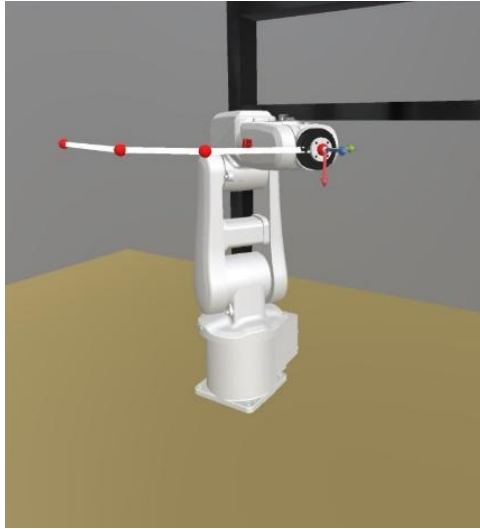


Figure 29: Waypoint markers displayed along a robot path in the virtual environment (right view).

1. Add Waypoint

This function captures the current robot configuration as a waypoint in the path sequence. When a user moves the robot into a desired position using the joint controller and uses this function:

The current joint values ($J1-J6$) are recorded and stored in the waypoints list, a visual marker is initiated at the current end-effector position, a line render is updated to visually connect waypoints providing clear visuals.



Figure 30: Waypoint visualization showing motion sequence from a left perspective.

2. Play

The play function executes the waypoint sequence, allowing users to verify their path immediately:

```
private IEnumerator PlayWaypointsCoroutine()
{
    for (int i = 0; i < waypoints.Count - 1; i++)
    {
        double[] startPoint = waypoints[i];
        double[] endPoint = waypoints[i + 1];
        float duration = 2f; // Time to move between waypoints
        float elapsedTime = 0f;
        while (elapsedTime < duration)
        {
            elapsedTime += Time.deltaTime;
            // Interpolate joint values
            double[] interpolatedJoints = new
double[startPoint.Length];
            for (int j = 0; j < startPoint.Length; j++)
            {
                interpolatedJoints[j] = Mathf.Lerp(
                    (float)startPoint[j],
                    (float)endPoint[j],
                    elapsedTime / duration
                );
            }
            // Send interpolated joint values to the robot
        }
    }
}
```

```

        SendJointsMessageToRobot(
            interpolatedJoints[0],
            interpolatedJoints[1],
            interpolatedJoints[2],
            interpolatedJoints[3],
            interpolatedJoints[4],
            interpolatedJoints[5]
        );
        yield return null;
    }
}

```

This implementation features:

Smooth interpolation between waypoints, configurable duration parameters for speed control, real-time feedback as the robot follows the path, automatic addition of the current position as first waypoint if need be.



Figure 31: Robot model in default home position before waypoint path execution.

3. Clear

The Clear function resets the waypoint to allow for new path creation:

Stops any ongoing waypoint playback, destroys all waypoint markers in scene, clears the internal waypoint list, resets the line renderer visuals.

```
MODULE Module1
VAR jointtarget waypoint;
PROC Main()
  waypoint := [[57.99,0.18,-0.31,-0.03,0.00,0.00], [9E9,9E9,9E9,9E9,9E9,9E9]];
  MoveAbsJ waypoint, v100, fine, tool0;
  waypoint := [[-39.42,-0.01,0.02,0.00,0.00,0.00], [9E9,9E9,9E9,9E9,9E9,9E9]];
  MoveAbsJ waypoint, v100, fine, tool0;
  waypoint := [[4.42,-0.01,0.02,0.00,0.00,0.00], [9E9,9E9,9E9,9E9,9E9,9E9]];
  MoveAbsJ waypoint, v100, fine, tool0;
  waypoint := [[27.59,0.01,-0.02,0.00,0.00,0.00], [9E9,9E9,9E9,9E9,9E9,9E9]];
  MoveAbsJ waypoint, v100, fine, tool0;
  waypoint := [[57.59,0.02,-0.04,0.00,0.00,0.00], [9E9,9E9,9E9,9E9,9E9,9E9]];
  MoveAbsJ waypoint, v100, fine, tool0;
ENDPROC
ENDMODULE
```

Figure 32: Auto-generated RAPID code representing recorded waypoints for ABB RobotStudio.

4. Export RAPID

This function translates the created waypoint sequence into ABB RAPID programming:

```
public void ExportWaypointsToRAPID()
{
    if (waypoints.Count == 0)
    {
        Debug.LogWarning("No waypoints available for export.");
        return;
    }

    // Define RAPID script header
    string rapidScript = "MODULE Module1\n";
    rapidScript += "VAR jointtarget waypoint;\n";
    rapidScript += "PROC Main()\n";

    // Add waypoints as RAPID commands
    for (int i = 0; i < waypoints.Count; i++)
    {
        double[] wp = waypoints[i];
```

```

        rapidScript += $" waypoint :=
[[{wp[0]:F2},{wp[1]:F2},{wp[2]:F2},{wp[3]:F2},{wp[4]:F2},{wp[5]:F2}],
[9E9,9E9,9E9,9E9,9E9,9E9]];\\n";
        rapidScript += $" MoveAbsJ waypoint, v100, fine, tool0;\\n";
    }

    // Close RAPID script
    rapidScript += "ENDPROC\\n";
    rapidScript += "ENDMODULE";

    // Save to file
    string filePath = Application.dataPath + "/Waypoints.mod";
    File.WriteAllText(filePath, rapidScript);
}

```

This function:

Formats each waypoint as a RAPID jointtarget variable, includes MoveAbsJ commands for joint based movement, configures appropriate speed and zone parameters (v100, fine), generates a complete RAPID module ready for deployment to a physical robot, saves the code to a .mod file in the directory of the application.



Figure 33: Robot returning to home position following waypoint playback completion.

5. Reset Robot

The reset robot function returns the robot to a predefined home position:

Using a smooth interpolation between current and home positions, visual feedback shown during transition ensures consistent starting condition for new path creation.

- Visual Representation and Feedback

The implementation includes visual feedback to enhance user understanding:

1. Waypoint Markers: 3D spheres placed at recorded positions.
2. Path Visualization: line renderers connecting sequential waypoints.
3. Smooth Transitions: Interpolated movement between waypoint playback.
4. Progress Feedback: visual indicator of executed waypoint segment.

- Integration with Robotics Workflow

The waypoint panel serves a bridge between VR based manipulation and production robotics:

Educational Value: demonstrates core robotic concepts like joint-space motion planning, immediate visual feedback on planned paths, illustrates connection between manual teaching and program execution, Workflow Enhancement: reduces the need for complex programming knowledge, enables rapid prototyping of movement sequence, iterative testing is made possible to refine robotic paths, Production Integration: RAPID code export provides a direct path to physical robot implementation, maintains exact joint configurations for precise task jobs, joint space movement ensures the feasibility of the created robot paths.

By combining intuitive waypoint creation with code generation, the waypoint panel embodies the goal of making the development of robotic workflows more accessible.

In summary, the development of JointLab presents an approach to early-stage robotics education within architectural fabrication by merging virtual reality with real-time, bi-directional control of industrial robots. Through the integration of Unity, ABB RobotStudio, and the Meta Quest 3, the system allows users to safely explore robotic workflows, joint manipulation, and spatial coordination in a simulated dFab environment. The UI design which includes the connection panel, camera panel, waypoint panel, diagnostic panel, and joint controller panel, helps support beginners in engagement. While the platform addresses accessibility, it reduces the technical barrier to entry, there does exist limitations such as its lack of automated path validation, inverse kinematic planning, and collaborative multi-user/robot modes. These gaps provide a clear direction for future iterations of the system and helps position JointLab as a functional prototype for more advanced research in robotics education.

5 User Study and Evaluation

To understand the educational impact of JointLab, a structured user study was conducted with learners. This chapter evaluates the effectiveness of the system in developing robotic skills such as joint manipulation, motion planning, and spatial understanding.

As part of the user study, participants completed three structured tasks to evaluate the usability of JointLab and its key features. These tasks allowed participants to familiarize themselves with the virtual environment and test their skills in robotic manipulation and navigation.

Task 1: Joint Adjustment to Predefined Positions

Participants were instructed to set the robot's joints (J1-J6) to exact predefined positions using the joint controller. An example set of joint positions provided was:

J1: -68.9 | J2: 36.51 | J3: -0.71 | J4: 120.96 | J5: 69.07 | J6: -167.4

Objective: Assess participants' ability to interpret and achieve specific joint configurations.

Metrics Observed:

1. Accuracy of reaching the exact joint values.
2. Feedback on the responsiveness and precision of the joint sliders.
3. Participants' impressions on the Highlight and Show Joint Axis feature for understanding joint movement.

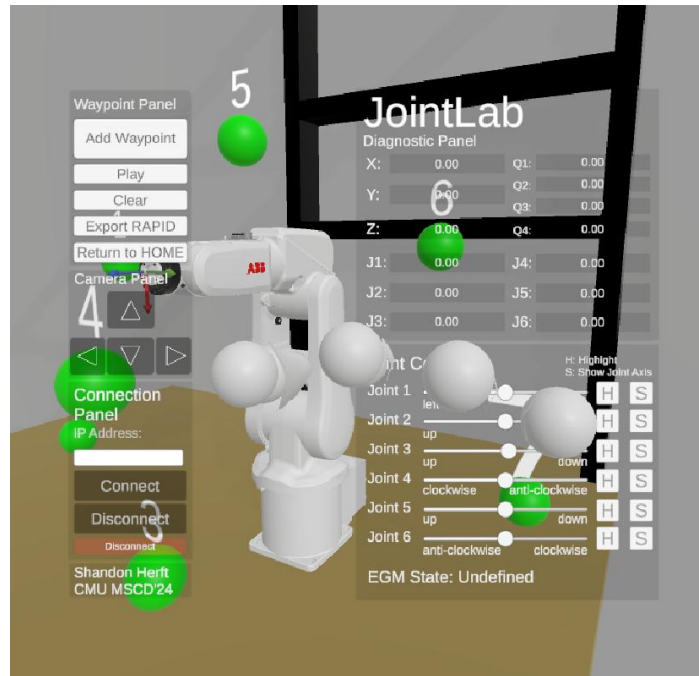


Figure 40: Task 2 interface showing sphere-based spatial navigation using JointLab's UI panels.

Task 2: Spatial Navigation and Accuracy

Participants were asked to use the Joint Controller to move the robot's end effector to seven pre-placed spheres in the virtual workspace. The spheres were numbered (1-7) and served as points for navigation. Participants were also encouraged to use the Camera Panel to switch between views to refine movements.

Objective: tests participants' ability to navigate the robot's joints within the virtual environment.

Metrics Observed:

1. Time taken to reach each sphere.
2. Number of slider adjustments made for precise positioning.
3. Accuracy of alignment with the sphere coordinates.

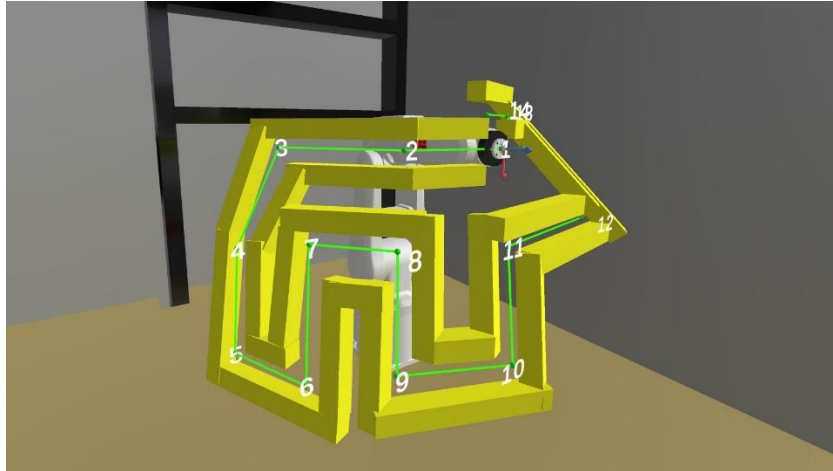


Figure 41: Task 3 maze navigation environment with numbered waypoints for spatial reasoning evaluation.

Task 3: Precision Path-Following Task in 3D Maze

The third task in the user study evaluated users' ability to plan and execute a complex robotic path through a 3D maze specifically designed for this task. Unlike the previous tasks that emphasized single-joint manipulation or basic spatial alignment, Task 3 required participants to coordinate movements over multiple axes and maintain spatial awareness. Using the waypoint panel in the VR interface, users were instructed to manually place 14 waypoints that guided the ABB IRB 120 robot's end effector through a predefined path. This simulated real-world motion planning within a virtual fabrication context.

The task was designed to measure both speed and accuracy. Completion time was tracked from the first waypoint placement until the path was complete, while accuracy was quantified by calculating the mean spatial deviation of each user defined waypoint from the ideal path. This evaluation provided insight into the trade-offs users made between efficiency and precision. The task surfaced individual strategies and control styles – some users favored quick movement, while others were more cautious minimizing errors.

This challenge also tested users' depth perception and axis coordination in an immersive environment. The maze layout emphasized vertical displacement making navigation in the Z-axis particularly difficult. The task offered a comprehensive assessment of users' ability to coordinate in 3D space and demonstrated the impact of VR interface design on robotic path planning.

These tasks captured a wide range of user interactions, including spatial understanding, joint accuracy and overall system usability.





Figure 42: User operating JointLab in VR using Meta Quest during task-based sessions.

The user study was conducted to assess the usability and effectiveness of JointLab in skill development in robotics education, particularly in joint manipulation, path planning and spatial understanding within virtual reality. Ten participants with diverse backgrounds in robotics and VR were testing the application. To preserve anonymity, all participants will be referred to by user numbers:

User #	Robotics Experience	VR Experience

1	Familiar, not frequent robotics user	Experienced
2	Highly experienced, works with robots regularly	Familiar, limited hands-on
3	No experience	Skilled in VR development
4	Intermediate	Limited
5	Little to no experience	Experienced
6	No experience with robotic tools/systems	Proficient
7	Highly experienced	Highly experienced
8	Experienced	Some knowledge, limited hands-on
9	No experience; non-technical background	No experience
10	Experienced	Knowledgeable, low usage

5.1 Key Observations from User Interaction

By including ten participants in the study, deeper insights were gained into how users engaged with JointLab's VR interface. These observations highlight interaction strengths, recurring challenges, and the interface's educational value. For a complete record of participant responses referenced in this section please refer to the full transcripts in Appendix 8.1.

1. Joint Manipulation and Feature Use

All participants highly benefited from the Highlight (H) and Show Joint Axis (S) features, especially in the earlier tasks. These visual aids helped new users understand which joints were being moved and the direction they were being moved in.

- User 1 noted that once they felt the joint movement was intuitive, they didn't need to rely on these features as much, but they did acknowledge its importance early in the testing.
- User 7 found the features useful but relied more on experience and instinct when navigating through the maze.

2. Waypoint Panel and Path Planning

The waypoint panel emerged as a core feature. Users praised its ability to create, test and revise robotic paths in real-time with visualization.

- User 3 commented on the ability to see the robot follow paths immediately, reinforcing the learning loop.
- User 6 appreciated the ability to review and adjust placements dynamically, especially under spatial constraints.

3. Spatial Understanding via Camera Controls

The camera panel played an essential role in spatial understanding. Participants toggled between different views (top, left, right, front) to better judge joint orientation.

- User 8 found switching views helpful for understanding spatial layout in 3D space, especially noting it improved over RobotStudio.
- User 9 relied heavily on the camera panel to navigate tight spaces and corners during task 9.

4. Learning Curve and Interface Adaptability

Despite different levels of experience, most users adapted to the tool quickly. Many users expressed a sense of improvement between tasks, adding to the intuitiveness of the interface.

- User 2 performed well despite having limited experience using VR, showing how the tool levels the learning field.
- User 4 expressed an increase in confidence over time, linking understanding to hands-on interaction.

5. Reflections from Task 3 (3D Maze)

Task 3 revealed difference in user strategies and control. Users with stronger spatial orienting skills performed better under time constraints and in managing joint strategies.

- User 5 demonstrated precision but slower speed, suggesting a more cautious strategy.
- User 10 struggled with slider sensitivity but praised the overall clarity and utility of the UI.

While participant feedback showed how users experienced and interacted with the system in their own way, it was important to quantify their performance in a structured comparative manner. To achieve this, task 3 focused on assessing users' spatial skills and tool usage to capture precision, efficiency, and control in a measured space.

5.1.1 Task 3 Quantitative Analysis: 3D Maze

Navigation

To further evaluate spatial reasoning and fine control of users interacting with the system, the third task was introduced to navigate the robot end effector through a 3D maze using joint-level manipulation.

Methodology

Participants competed Task 3 in the simulation where the ABB IRB 120 was present in the dFab lab space. Metrics were captured between Unity and RobotStudio, including:

- Mean Distance to Target – the average offset from the waypoint placed by the user to the predefined target.
- Completion Time – the total time taken by the user to complete the maze task.
- Performance Score – calculated as Mean Distance x Time, combining precision and efficiency.
- Axis-specific Errors – breakdown of the deviation across X, Y, and Z axes.

Results Overview

Across ten participants, mean distances ranged from 0.615 to 0.695 units, with completion times between 205 and 355 seconds.

User	Mean Distance	Max Distance	Min Distance	Standard Deviation
User 1	0.656	0.742	0.000	0.183
User 2	0.658	0.726	0.000	0.183
User 3	0.647	0.707	0.000	0.180
User 4	0.651	0.729	0.000	0.181

User 5	0.615	0.699	0.000	0.172
User 6	0.662	0.768	0.000	0.187
User 7	0.659	0.753	0.000	0.184
User 8	0.666	0.757	0.000	0.186
User 9	0.695	0.768	0.000	0.193
User 10	0.638	0.750	0.000	0.180

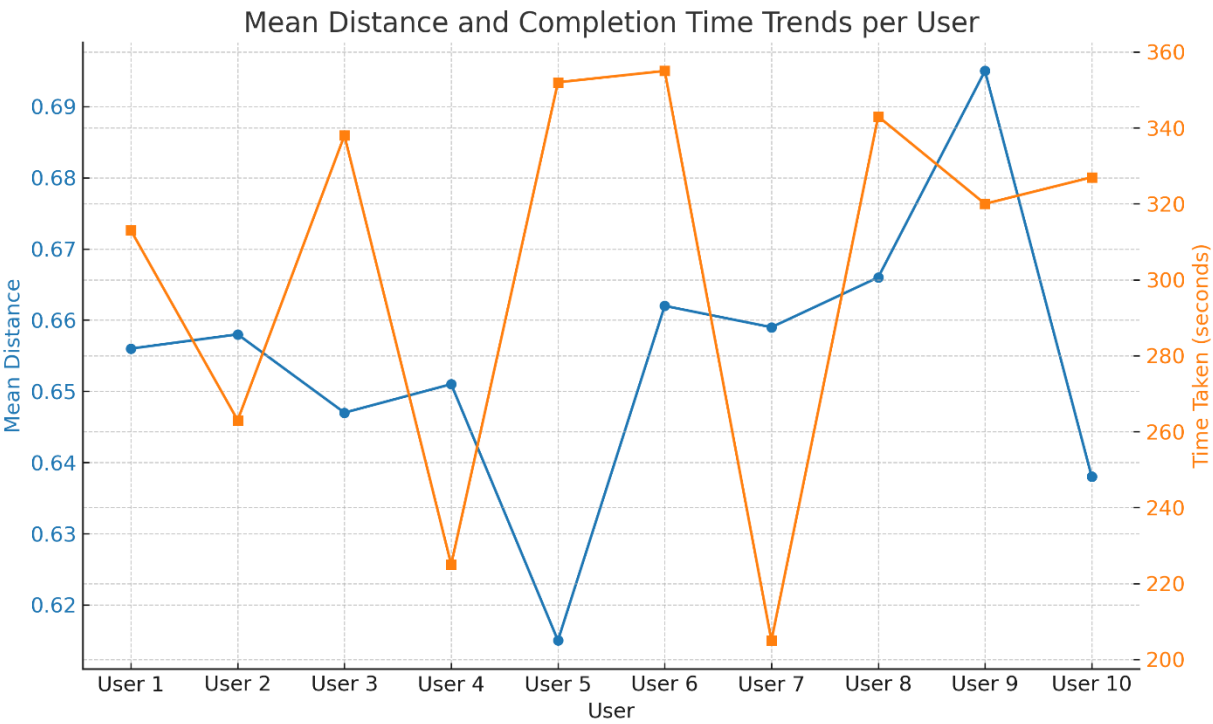


Figure 43: Mean distance error and task completion time trends across 10 users during Task 3.

Compares mean distances, highlighting User 5 as the most precise and User 9 as an outlier with the highest deviation.

User	Mean Distance	Time (seconds)	Performance Score
User 7	0.659	205	135.16
User 4	0.651	225	146.57
User 2	0.658	263	173.05
User 1	0.656	313	205.41
User 10	0.638	327	208.63
User 5	0.615	352	216.49
User 3	0.647	338	218.62
User 9	0.695	320	222.42
User 8	0.666	343	228.43
User 6	0.662	355	235.01

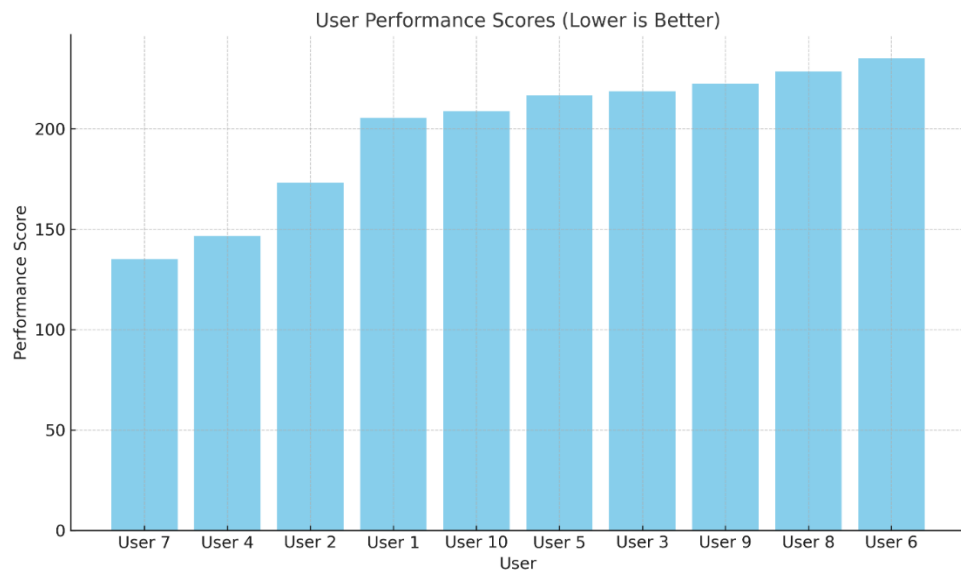


Figure 44: User performance scores aggregated across all tasks, with lower scores indicating better performance.

Illustrates the performance scores, identifying User 7 as the top performer based on speed and precision.

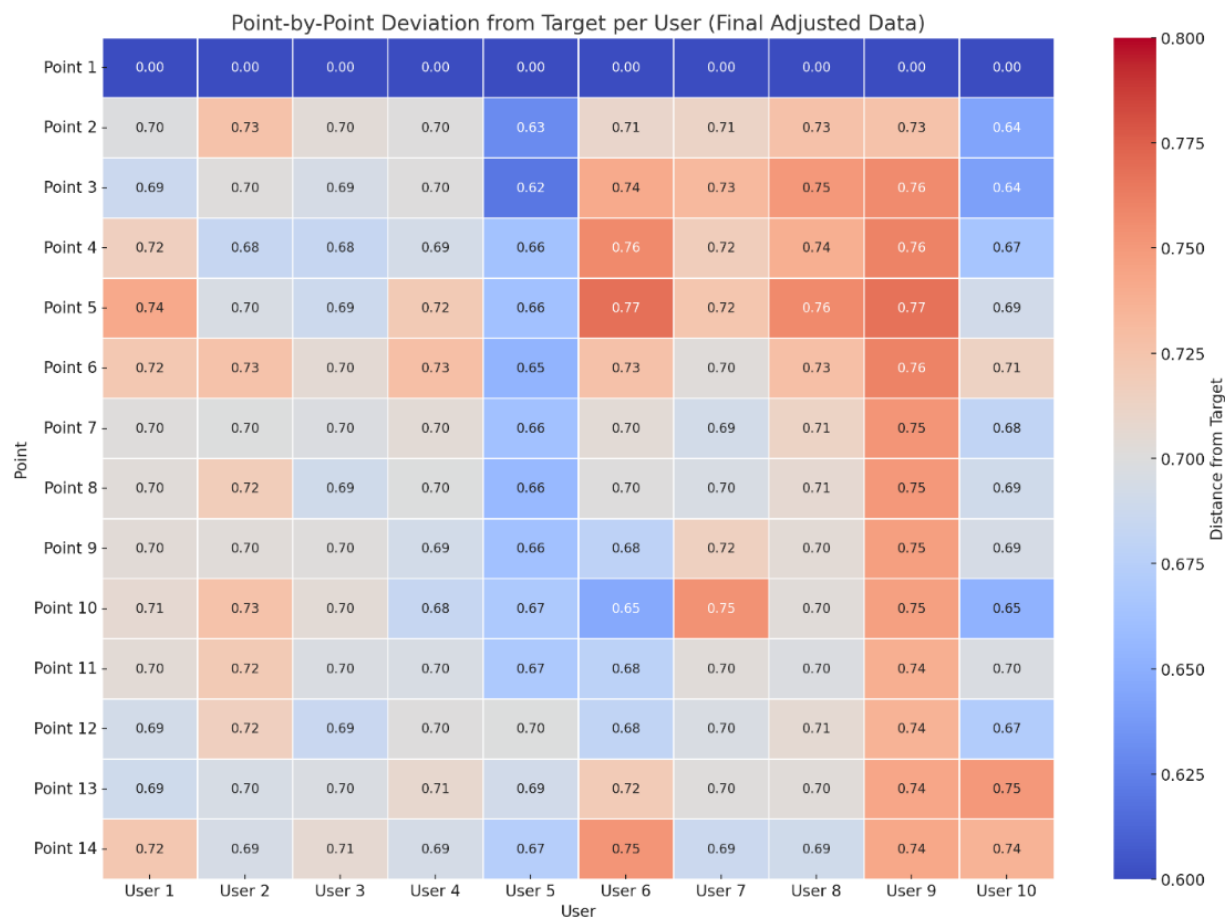


Figure 45: Heatmap of point-by-point deviation from target positions for each user in Task 3.

This heatmap visualization shows spatial error distributions across the maze, with there being an increase in error density in curved or regions with sharp curves, this suggests that depth approximation is a recurring challenge.

Axis Based Deviation

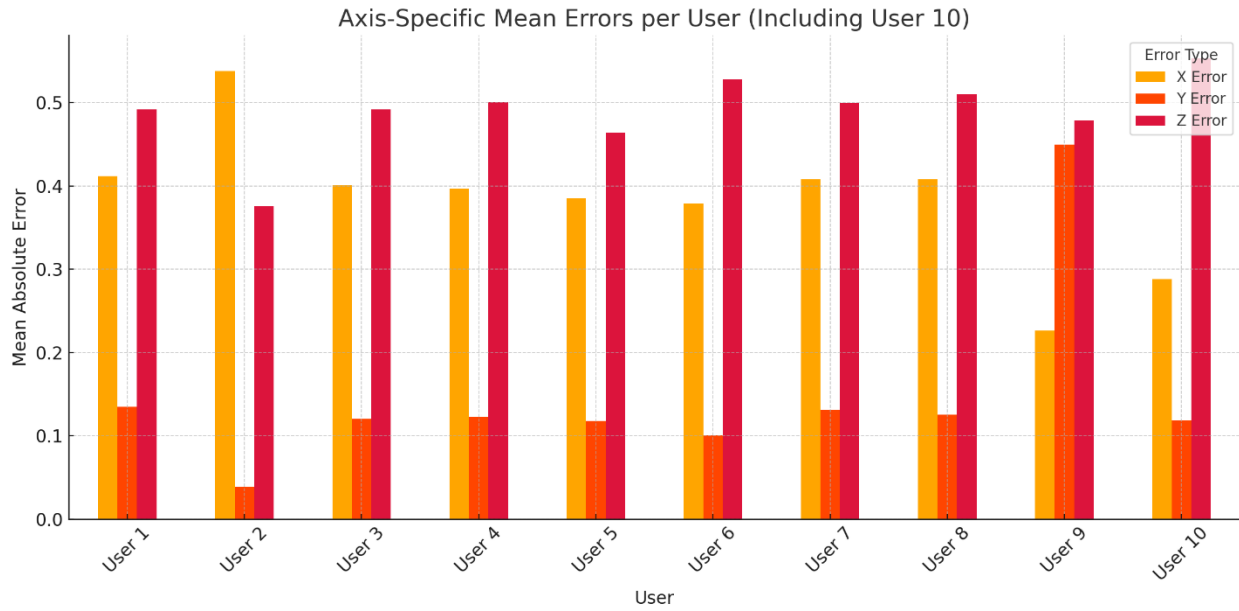


Figure 46: Axis-specific mean absolute error per user across X, Y, and Z coordinates.

	X Error	Y Error	Z Error
User 1	0.411907221	0.134990064	0.49186724
User 2	0.537887043	0.038625271	0.375907059
User 3	0.401366586	0.120561943	0.492040844
User 4	0.396426303	0.122715386	0.500594802
User 5	0.38553847	0.117579229	0.464025754
User 6	0.379267698	0.100441714	0.527907244
User 7	0.408148911	0.130760364	0.499385733
User 8	0.40784811	0.125285314	0.510444298
User 9	0.2263917	0.449469666	0.478968233
User 10	0.2880197	0.118443942	0.553587915

To explore this further, each suers errors were aggregated by axis (X, Y, Z). As shown above, most users have had a greater deviation along the Z-axis, while Y-axis errors remain minimal.

Axis	Error Trend
X	Lateral drifts were common
Y	Low overall deviation
Z	Most affected due to lack of depth cues.

User 8 said that it looked like the end-effector was aligned, but it was always too far forward.

Cluster Analysis

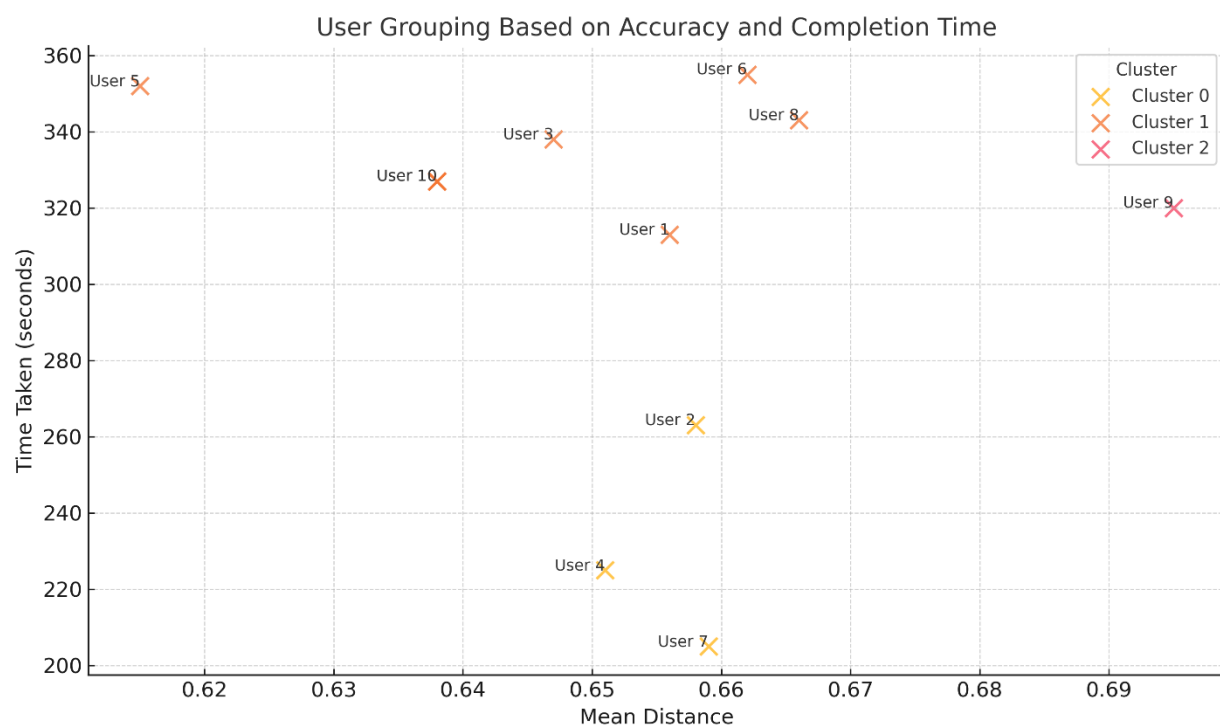


Figure 47: User clustering based on mean distance error and task completion time using K-means.

Users were grouped into performance clusters based on distance and time. These clusters reflect:

- Efficient/Accurate Users (e.g. User 7)
- Precise but Slower Users (e.g. User 5)
- Inconsistent Users (e.g. User 9)

This categorization provides an insight into user behavior under cognitive and spatial difficulties, informing where future interface optimizations could be made.

Insights and Implications

- The 3D maze differentiated between users' spatial coordination and planning.
- Z-axis drift reinforced the importance of depth cues in future iterations.
- Heatmap and deviation analysis offer feedback for tailoring curriculum and UI refinements.

5.2 Challenges and Suggestions

Following the user study with ten participants with varying experience levels with robotics and VR, several challenges were identified. These insights informed interface improvements and directions for future work.

1. Slider Responsiveness and Feedback Sensitivity:

Several participants noted difficulty in making precise adjustments to the joints to input lag and inconsistent feedback from the sliders. This issue was partially mitigated by changing the `LpFilter` and `SampleRate` in the RAPID script. It still surfaced under high system load.

- User 4: “I tried to nudge it a bit, but it jumped more than I expected.”
- User 7: “Fine-tuning is tricky, especially when you’re near limits.”

Suggested Improvement: introduce dynamic sensitivity option mode that adapts based on proximity to joint limit, with optional haptic feedback.

2. Diagnostic Panel Usage:

While advanced users appreciated having the diagnostic panel for real-time insights, beginners ignored it while interacting with the tool.

- User 1: “I saw the panel there, but I wasn’t sure what to do with the data.”
- User 3: “I only used it when something went wrong — maybe it needs a tutorial.”

Suggested Improvement: implement a guided overlay or tooltip walkthrough for first time users, and simplified terminology for novices.

3. VR Headset Comfort:

The Meta Quest 3 offered high immersion, but several users reported discomfort during longer tasks, especially around the nose bridge and forehead area.

- User 5: “After a while I just wanted to take it off. It gets heavy.”
- User 8: “I had to pause halfway just to reset — maybe a standing mode?”

Suggested Improvement: provide a headset-fit prompt before starting, include a standing vs seated mode selection.

4. Spatial Misperception and Depth Cueing:

Users showed higher deviations along the Z-Axis, indicating depth estimation challenges. This misalignment influenced path accuracy and confidence.

- User 6: “I thought the robot was aligned, but it was way off when I replayed the path.”

Suggested Improvement: add real-time shadows, grid snapping, and ghost previews of the robot's path for better depth perception.

5. Task Flow Interruptions and Limited Undo Options:

Several participants were frustrated about not being able to undo or modify waypoints without restarting the task.

- User 2: “I just needed to fix one step but had to delete everything.”
- User 10: “An undo or back button would’ve saved a lot of time.”

Suggested Improvement: introduce an edit-history stack and a confirmation before adding waypoints, to support and error-tolerant workflow.

While these challenges highlight areas for improvement in the interface and fidelity, users also revealed how they adapted and grew experience throughout. Despite initial friction, significant learning gains were observed, particularly in spatial reasoning, toolpath planning, and control. These patterns are explored in the next section.

5.3 Observations on Skill Development

Throughout this study, users demonstrated clear progression in robotic control skills, spatial awareness and confidence. This development was observable across all three tasks and was reinforced by participants in their own reflections during the sessions.

1. Early Familiarization and Growing Confidence

Most users began with minimal familiarity with joint-level robot control but quickly adapted after more exposure to the tool. Several users spoke about the intuitive feeling of the tool early on.

“The interface feels like a game. I’m already thinking how this would be great for beginners.” —

User 5

“I just added a waypoint—there’s a red dot now. That’s really intuitive.” — User 3

By task 2 joint manipulation was more deliberate and confidence expressed to understand and predict robot motion was understood.

“Okay, playing it now... wow. This is really cool—definitely better than coding it in RAPID first and hoping the robot can follow it.” — User 3

2. Strategic Thinking and Adaptation

The 3D maze prompted more advanced planning and problem solving. Some participants shifted from reactionary adjustments to proactive movements.

“I want to beat the current record time, but also stay accurate.” — User 5

“This task is more demanding... but it was useful for learning robot motion planning.” — User 2

Participants learned from earlier mistakes and made intentional movements by the end of this task.

View switching, positioning panels dynamically, and adjusting interaction strategies became common optimizations.

3. Refined Spatial Understanding

Repeated use of the camera panel and highlight/show joint axis tools helped participants internalize the robot’s layout. Users relied on visual aids, but over time they were less dependent on them.

“Vertical (up/down) movement feels less sensitive than left/right. My hands are shaking, so it’s tricky to stay precise near the end.” — User 5

“I had better luck earlier standing slightly off-center... That helped with finding points.” — User 3

These user reflections provide qualitative evidence of growing competence in the core aspects of learning through the system.

5.4 Implications for Design

The insights gathered from JointLab user study highlight key design principles that could inform future development of educational robotics tools that are immersive. These implications support different levels of experience focusing on accessibility and responsiveness.

1. Design for Fine-Grained Control

Several users struggled with joint slider sensitivity, during tasks that demanded high precision. This highlights the importance of integrating different control modes, such as coarse or fine adjustments, or gesture based micro controls.

Design Implication: interface should provide adjustable control, allowing users to adjust and tune as need be.

2. Embed Feedback Through Spatial and Visual Cues

Participants found the camera panel and highlight/axis tools helpful for understanding spatial joint qualities. However, users still had difficulty with Z-axis alignment.

Design Implication: future interfaces should include spatial cues, ghosting, shadows and previews for aiding in depth perception.

3. Support Progressive Disclosure and Feature Layering

Novices tended to ignore the Diagnostic Panel until later in the session. This suggests that the interface should grow with the user, revealing advanced features with increased confidence.

Design Implication: adaptive UI layers that start with core tools and introduce diagnostic data, toolpath parameters and export functionality progressively.

4. Facilitate Reversibility and Iteration

Multiple users expressed the inability to undo or revise waypoints without clearing the whole sequence. This broke flow and experimentation.

Design Implication: include history stacks, per waypoint editing access, and an undo/redo system to support iterative design learning.

5. Consider Physical Comfort and Task Duration

Discomfort with longer use of the VR headset was a recurring issue. Some users paused mid-task to reorient themselves.

Design Implication: optimize for comfort with seated/standing modes, shorter breaker into the UI, and have desktop VR alternatives.

Together, these reinforce the idea that good design in education robotics interfaces is not just about functionality but also learning, reducing friction and adapting to users over time. These have a direct impact on educational outcomes and set the stage for the next section in training.

5.5 Implications for Training and Learning

The JointLab study reveals how users interact with robotic systems, but also how they learn through hands-on immersion. These findings provide insights into designing robotics training experiences, for students and beginners in fields like architecture, design or fabrication.

1. Learning-by-Doing Builds Intuition Faster Than Instruction

Participants demonstrated clear learning gains while interacting with the system, by discovering concepts by experimenting rather than explicit tutorials.

“I didn’t really read anything—I just clicked around and figured it out.” — User 1

Training Insight: immersive training should prioritize open-ended exploration and have a reduced cognitive load from the manuals or lessons available. Self-discovery by visual feedback can offer a deeper understanding.

2. Progressive Task Design Encourages Transferable Skills

Tasks were designed to build on one another, from basic joint understanding to complex spatial navigation. Users referenced previous tasks while attempting the new challenges.

Training Insight: design learning that helps introduce core concepts early, then scaffold difficulty to allow for knowledge transfer and train in skills like planning, path optimization, and orientation correction.

3. Embodied Learning Enhances Spatial Reasoning

The use of full body VR, hand-tracking and camera toggling allow users to “feel” the robot’s movement and see a cause-and-effect relationship in real-time.

“It was easier to understand once I could move around and see the robot from different sides.” — User 6

Training Insight: embodied spatial interaction in 3D environments should be a component of robotics training to help users' intuition for kinematics and spatial constraints.

4. Confidence and Competence Are Built Through Feedback Loops

Immediate feedback via real-time movements, visual cues, and replaying features have enabled users to course-correct and gain confidence in using the system quickly. Users expressed “getting it” after watching the robot respond to their inputs.

Training Insight: tools should reinforce feedback loops to close the gap between action and outcome. Features like path replay, joint previews and live diagnostic data help learners internalize the abstract robotics principles.

5. Adaptability for Non-Experts Broadens Access

Participants from non-engineering backgrounds (e.g. architecture, art) complete all tasks with little instruction. The interface's visual and low-risk nature helped reduce the learning curve.

Training Insight: training systems should be designed with accessibility in mind, not just language and visuals, but also in cognitive load, styles of interaction, and tolerance of failure. This democratizes access to robotics education.

These insights demonstrate that immersive VR systems like JointLab can be more than just a control interface; They can function as a powerful learning environment. When intentionally designed, they allow users to grasp complex robotic behaviors, preparing them for real-world applications in architecture, design and others.

6 Discussion

6.1 The Future of Immersive Robotics Education

JointLab has a broader vision for how immersive, accessible environments can transform early-stage robotics education, especially in the context where access to physical equipment is an issue. The framework enables learners to engage in robotic workflows through interactions in a safe and structured environment.

The following storyboard illustrates a typical user journey, from first exposure to the tool to hands-on execution with the real-world robot. It highlights how JointLab brings confidence in robotics through an immersive tutorial interaction.

Storyboard: VR Based Robotics Education



Figure 34: A robot classroom scenario with limited access to robots.



Figure 35: Students train in a safe virtual environment.



Figure 36: Students complete structured robotic tasks in the virtual environment.



Figure 37: Teachers monitor session analytics.

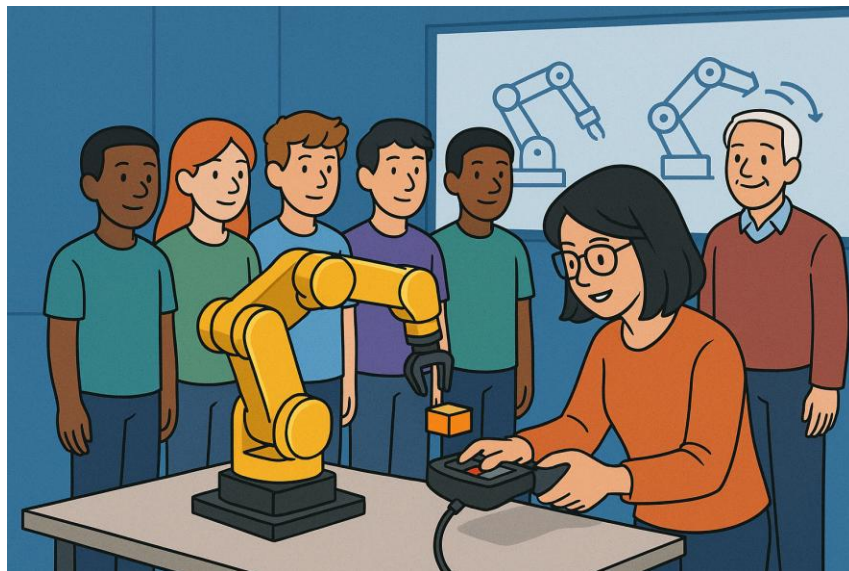


Figure 38: Students transition to real robots after virtual training.

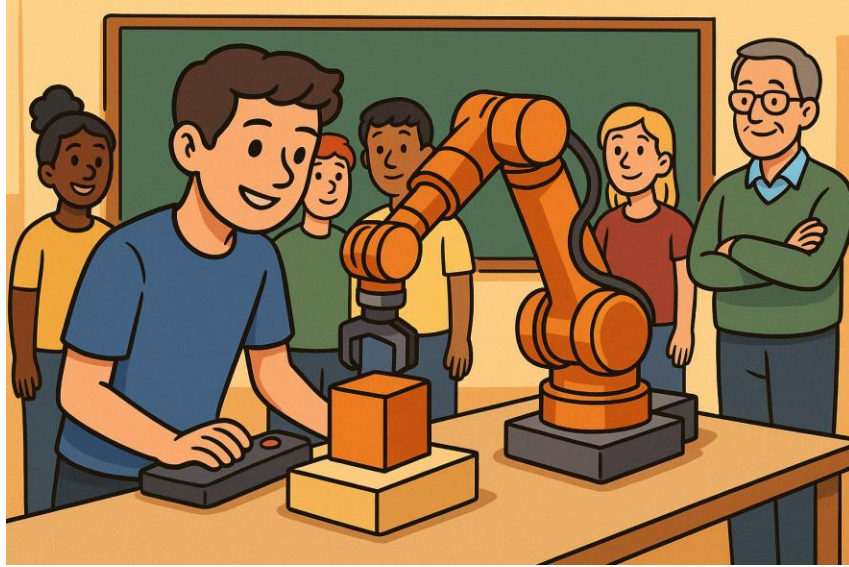


Figure 39: Students demonstrate learned skills on a real robot.

This vision shows how platforms like JointLab can lower the barrier of entry to robotics education. With immersive VR based learning, teachers can extend robotics to broader student populations, without the risk typically associated with industrial systems. As a scalable framework, JointLab not only simplifies robotics instruction but also lays the groundwork for future models of learning in design, engineering and more.

6.2 Conclusions

This thesis presented JointLab, a virtual framework designed to integrate real-time interaction methods between Unity, ABB RobotStudio, and the Meta Quest 3, with a focus on improving robotics education and skill development in architectural fabrication workflows. The research explored how immersive and intuitive interfaces can lower the barrier of entry for robotics, enabling users to engage with complex systems through spatially interactive environments.

The following key contributions were demonstrated:

1. Accessibility: By incorporating Meta Quest 3's hand tracking and controller capabilities, JointLab provided an immersive, user-centric interface suitable for participants from non-technical backgrounds (e.g. architecture and art) to complete robotic tasks. This suggests that JointLab democratizes access to robotics education by lowering the cognitive load threshold.
2. Skill Development: features like joint manipulation, waypoint creation, and spatial navigation enabled users to learn and interact intuitively with robotic systems, leveraging VR for better conceptual understanding.
3. Real-Time Interaction: bi-directional communication protocols using EGM ensured seamless control and synchronization between virtual and physical robots, enhancing the user experience and precision.
4. Design and Training Implications: insights from task 3 and user interviews showcased the importance of Z-axis cues and feedback in better understanding, allowing them to pave the way for confident transitions from virtual to real robots.

While JointLab demonstrated promising outcomes and revealed new opportunities for an immersive robotics education experience, the system also faced constraints during its development and evaluation stages. The next section outlines these limitations and offers context for interpreting results for future refinement.

6.3 Limitations

While JointLab demonstrates the integration of virtual reality and robotics for architectural fabrication workflows, it is subject to several limitations. These constraints shape the current implementation and inform the next steps for refinement and scaling.

1. Depth Perception Challenges and Interface Clarity for Beginners.

Many users struggled with Z-axis accuracy during task 3. This reflects a broader issue of spatial misperception in VR, which affects performance and confidence. While the diagnostic panel allowed monitoring, it lacked higher level analytics or visual summaries. Novices found it irrelevant during task execution.

Limitation Impact: spatial errors limit task efficiency and highlight the need for visual cues such as ghost previews or shadows that provide depth hints. The current design favors experienced users, new users could benefit from a more layered, guided display or tutorial integrated into the tool.

2. VR Headset Constraints and Latency Optimization

Although Meta Quest 3 provides immersive interaction, extended use raised comfort issues during testing. This could limit the framework's usability for long duration tasks or continuous learning sessions. Although the system supports real-time interaction, potential latency issues in the communication pipeline were noted during testing and addressed due to issues in the RAPID code for EGM communication and system hardware having a slower graphics card. Adjustments to parameters like LpFilter and SampleRate improved performance.

Limitation Impact: ergonomic constraints may reduce training session durations and exclude participants from being sensitive to physical strain. While latency was reduced in later tests, high-performance hardware requirements for smooth interaction, limits the portability of the framework and accessibility.

3. Dependency on Specific Hardware and Software

The framework relies exclusively on the Meta Quest 3 for VR interaction and ABB RobotStudio for robot control. This restricts its usability for organizations or users with access to such technologies and limits the framework.

Limitation Impact: by relying on proprietary systems, its adoption in academic or cross-disciplinary settings could vary.

4. Limited Sample Size in User Studies and Participant Diversity

The testing involved only a small number of participants, representing a varied skill level in robotics and VR. While the feedback was valuable, broader testing across a larger sample size is required to arrive at a universally applicable conclusion.

Limitation Impact: broader testing with more diverse participants (e.g. educators, industry professionals) which is necessary for generalizing conclusions.

5. Focus on Single-Robot Systems

The framework only supports one robot (ABB IRB 120). Multi-robot coordination remains unexplored, limiting its scalability for larger, more complex applications in fabrication.

Limitation Impact: the absence of multi-robot support limits the platform's application in full scale architectural fabrication setups.

By addressing these limitations in future iterations, JointLab can be further refined to expand its scalability, accessibility, and educational value. The next section outlines specific strategies to evolve the system for broader use.

6.4 Future Works

Building on the findings from this research, several areas for future development can enhance the scope of JointLab. These proposals aim to enhance the scalability, accessibility, intelligence and educational relevance in both architecture and robotics domains.

1. Multi-Robot Integration and Multi-User Collaborative Environments

Expanding the framework to support multiple robots would enable more comprehensive simulations and workflows. This would introduce challenges in coordinating collisions, task sharing, and synchronization, which would require advanced algorithms and spatial visualization. Introducing multi-user functionality would allow multiple participants to engage simultaneously in the virtual environment. This could help promote collaborative design and learning in education.

Future Enhancement: implement support for multiple robots, along with visual tools for shared workspaces and path coordination. Implement networked sessions with role-based views (e.g. instructor, peer) and collaborative toolpath editing.

2. Integration with Artificial Intelligence (AI) and Advanced Data Visualization Learning Analytics

Incorporating AI driven tasks planning and optimization could streamline waypoint creation, joint manipulation, and collision avoidance. This would help users program more complex robotic workflows requiring minimal intervention. The addition of learning analytics, such as progress

tracking, performance heatmaps, and error trends would provide insights to instructors and learners enhances the effectiveness of the system. Graphs tracking joint movement over time, visual overlays showing joint limits, reachable workspace, and potential collision zones.

Future Enhancement: integrate machine learning for dynamic waypoint suggestion, path optimization and real-time error correction. Include real-time charts, collision zones, tasks success heatmaps as part of the educational dashboard.

3. Enhanced Accessibility Features, Multimodal Interaction, and Cross-Platform Support

Supporting voice commands could help for controlling, expanding interaction beyond VR for desktop, adding adaptive scaling for users with limited fine motor control. Future work could expand compatibility to include other VR devices, motion-tracking systems, and robot brands to increase adoption. Incorporating multimodal feedback such as sound and haptics can also enhance the user experience, especially in reinforcing safety cues or responses.

Future Enhancement: add support for voice commands, desktop interaction, adaptive control scaling based on user, and screen-reader friendly interfaces. Integrate sound/haptic feedback for key events. Develop plug-in architecture for other brands (e.g. UR, KUKA) and devices (e.g. Vive, Leap Motion).

4. Integration with Parametric Design Tools

Further development of workflows connecting JointLab to parametric tools like Grasshopper could strengthen the relevance of the tool to architectural fabrication. Expanding the framework to include inverse kinematics and motion constraints could further support planning tasks.

Future Enhancement: extend API connections or develop custom plug ins for geometry to motion workflows. Add inverse kinematic functionality and motion validation.

5. Application to Broader Industries

While this thesis focuses on architectural fabrication, its framework could be adapted for applications in other industries such as manufacturing, healthcare, and education, extending its impact to a wider range of workflows.

Future Enhancement: explore partnerships with domain experts to adapt JointLab for training, simulation, or prototyping in other industries.

By addressing these areas in future research, the capabilities of JointLab could transcend its current limitations, making it more versatile and powerful of a tool for robotics education. These future directions position the system as a bridge between human centered design and advanced robotic control, relevant to not only architecture but to the broader trajectory of Industry 4.0 learning environments.

7 Bibliography

- [1] Meegle. (2025, January 5). Robotics in architecture. Meegle. https://www.meegle.com/en_us/topics/robotics/robotics-in-architecture
- [2] Arpitha, S. (2023, May 14). Robots in architecture: Transforming architecture with precision and efficiency. PA | Architecture & Technology. <https://parametric-architecture.com/robots-in-architecture-transforming-architecture-with-precision-and-efficiency/>
- [3] Lei, Y., Su, Z., & Cheng, C. (2023). Virtual reality in human-robot interaction: Challenges and benefits. *Electronic Research Archive*, 31(5), 2374–2408. <https://doi.org/10.3934/era.2023121>
- [4] Suomalainen, M., Nilles, A. Q., & LaValle, S. M. (2020). Virtual reality for robots. *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1–8. <https://ras.papercept.net/images/temp/IROS/files/0304.pdf>
- [5] Hu, E.-A., & Lee, J. J. (2017). Virtual reality in education: A tool for learning in the experience age. *International Journal of Innovation in Education*, 4(4), 215–226.
- [6] Whitman, L., et al. (2004). Virtual reality case study throughout the curriculum to address competency gaps. *International Journal of Engineering Education*, 20(5), 690–702.
- [7] Bhattacharjee, D., et al. (2018). An immersive learning model using evolutionary learning. *Computers & Electrical Engineering*, 65, 236–249.
- [8] Shin, D.-H. (2017). The role of affordance in the experience of virtual reality learning: Technological and affective affordances in virtual reality. *Telematics and Informatics*, 34(8), 1826–1836.
- [9] Zhao, R., Aqlan, F., Elliott, L. J., Baxter, E. J. (2020). Multiplayer physical and virtual reality games for team-based manufacturing simulation. *ASEE Annual Conference Proceedings*.
- [10] Manyika, J., Ramaswamy, S., Khanna, S., Sarrazin, H., Pinkus, G., Sethupathy, G., & Yaffe, A. (2015). *Digital America: A tale of haves and have-mores*. McKinsey Global Institute.
- [11] Davila Delgado, J. M. (2019). *Vision network: Augmented reality and virtual reality for digital built Britain (Final report)*. University of Cambridge.
- [12] Davila Delgado, J. M., Oyedele, L., Beach, T., & Demian, P. (2020). Augmented and virtual reality in construction: Drivers and limitations for industry adoption. *Journal of Construction Engineering and Management*, 146(5), 04020026. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0001844](https://doi.org/10.1061/(ASCE)CO.1943-7862.0001844).
- [13] Choset, H. M., Hutchinson, S., Lynch, K. M., Kantor, G., Burgard, W., Kavraki, L. E., & Thrun, S. (2005). *Principles of robot motion: Theory, algorithms, and implementation*. MIT Press.
- [14] LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.
- [15] Kavraki, L. E., & LaValle, S. M. (2016). Motion planning. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics* (2nd ed., pp. 139–162). Springer.
- [16] Şucan, I. A., & Chitta, S. (2011). MoveIt! Retrieved from <http://moveit.ros.org>
- [17] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: An open-source robot operating system. In *Workshop on Open Source Software at ICRA* (Vol. 3).

- [18] Coleman, D., Şucan, I. A., Chitta, S., & Correll, N. (2014). Reducing the barrier to entry of complex robotic software: A MoveIt! case study. *Journal of Software Engineering for Robotics*, 5(1), 3–16.
- [19] Dantam, N. T., Kingston, Z., Chaudhuri, S., & Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *International Journal of Robotics Research*, 37(10), 1134–1151.
- [20] Kingston, Z., & Kavraki, L. E. (2022). Robowflex: Robot motion planning with MoveIt made easy. *IEEE Robotics and Automation Letters*.
- [21] Cooney, M., Yang, C., Siva, A. P., Arunesh, S., & David, J. Teaching robotics with Robot Operating System (ROS): A behavior model perspective.
- [22] Esposito, J. M. (2017). The state of robotics education: Proposed goals for positively transforming robotics education at postsecondary institutions. *IEEE Robotics & Automation Magazine*, 24(3), 157–164.
- [23] Oca, P. S. R. (2023). Implementing and Using ROS in Undergraduate Robotics Curricula [Paper ID #44190].
- [24] Esposito, J. M. (2017). The state of robotics education: Proposed goals for positively transforming robotics education at postsecondary institutions. *IEEE Robotics & Automation Magazine*, 24(3), 157–164.
- [25] Qviro. (n.d.). ABB RobotStudio® use-cases. Qviro. Retrieved [03/22/25], from <https://qviro.com/product/abb/robotstudio-abb/use-cases>
- [26] Connolly, C. (2009). Technology and applications of ABB RobotStudio. *The Industrial Robot*, 36(6), 540-545. <https://doi.org/10.1108/01439910910994605>
- [27] Ravichandar, H., Polydoros, A. S., Chernova, S., & Billard, A. (2020). Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1), 297–330.
- [28] Matthias, B., Kock, S., Jerregard, H., Kallman, M., Lundberg, I., & Mellander, R. (2011). Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept. 2011 IEEE International Symposium on Assembly and Manufacturing (ISAM), 1–6.
- [29] Chang, Y.-h. I., Devine, K. L., & Klitzing, G. K. (Date unknown). Exploring the VR-based PBD programming approach to teach industrial robotics in manufacturing education (Paper ID #29244).
- [30] Huang, J., Dudley, J. J., Uzor, S., Wu, D., Kristensson, P. O., & Tian, F. (2022). Understanding user performance of acquiring targets with motion-in-depth in virtual reality. *International Journal of Human-Computer Studies*, 163, 102817. <https://doi.org/10.1016/j.ijhcs.2022.102817>
- [31] Adami, P., Rodrigues, P. B., Woods, P. J., Becerik-Gerber, B., Soibelman, L., Copur-Gencurk, Y., & Lucas, G. (2022). Impact of VR-based training on human–robot interaction for remote operating construction robots. *Journal of Computing in Civil Engineering*, 36(4), 04022006. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0001012](https://doi.org/10.1061/(ASCE)CP.1943-5487.0001012)
- [32] Fang, B., Ding, W., Sun, F., Shan, J., Wang, X., Wang, C., & Zhang, X. (2022). Brain-computer interface integrated with augmented reality for human-robot interaction. *IEEE Transactions on Cognitive and Developmental Systems*. <https://doi.org/10.1109/TCDS.2022.3147896>
- [33] Dou, X., Yan, L., Wu, K., & Niu, J. (2022). Effects of voice and lighting color on the social perception of home healthcare robots. *Applied Sciences*, 12(24), 12191. <https://doi.org/10.3390/app122412191>
- [34] Peral, M., Sanfeliu, A., & Garrell, A. (2022). Efficient hand gesture recognition for human-robot interaction. *IEEE Robotics and Automation Letters*, 7, 10272–10279. <https://doi.org/10.1109/LRA.2022.3206854>
- [35] Ramasubramanian, A. K., Mathew, R., Kelly, M., Hargaden, V., & Papakostas, N. (2022). Digital twin for human–robot collaboration in manufacturing: Review and outlook. *Applied Sciences*, 12(9), 4811. <https://doi.org/10.3390/app12094811>
- [36] Zhang, Q., Liu, Q., Duan, J., & Qin, J. (2023). Research on Teleoperated Virtual Reality Human–Robot Five-Dimensional Collaboration System. *Biomimetics*, 8(8), 605. <https://doi.org/10.3390/biomimetics8080605>

- [37] Familia, R. (2005). A virtual laboratory for cooperative learning of robotics and mechatronics. In 6th International Conference on Information Technology Based Higher Education and Training (ITHET) (pp. 17–20). IEEE. <https://doi.org/10.1109/ITHET.2005.1560277>
- [38] Kostal, P., & Mudrikova, A. (2010). Virtual laboratory and e-learning. In 2nd International Conference on Computer and Automation Engineering (ICCAE) (pp. 647–650). IEEE. <https://doi.org/10.1109/ICCAE.2010.5451568>
- [39] Hamid, N., Haron, H., Jambak, M., & Sukimin, Z. (2009). An overview of robotic simulation e-learning. In Third Asia International Conference on Modelling & Simulation (pp. 566–571). IEEE. <https://doi.org/10.1109/AMS.2009.66>
- [40] Xiaoli, Y., Yaqi, Z., Wei, W., & Hui, W. (2008). Virtual reality-based robotics learning system. In Proceedings of the IEEE International Conference on Automation and Logistics (pp. 859–864). IEEE. <https://doi.org/10.1109/ICAL.2008.4636359>
- [41] Xianhua, J., & Yuichi, M. (2005). Learning by observation of robotic tasks using on-line PCA-based eigen behavior. In Proceedings IEEE International Symposium on Computational Intelligence in Robotics and Automation (pp. 391–396). IEEE. <https://doi.org/10.1109/CIRA.2005.1560537>
- [42] Xiaoli, Y., & Qing, C. (2004). Virtual reality tools for internet-based robotic teleoperation. In Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (pp. 236–239). IEEE. <https://doi.org/10.1109/DS-RT.2004.41>
- [43] Tang, J.-L., & Wu, F.-S. (2010). VRML-based laboratory system for material mechanical performance testing. In VRML-based laboratory system for material mechanical performance testing. IEEE.
- [44] Duarte, N., Costa, S., Tyska, J., Marcos, P., & Maffei, R. (2009). An automated platform for immersive and collaborative visualization of industrial models. In 14th IEEE International Conference on Engineering of Complex Computer Systems (pp. 258–264). IEEE. <https://doi.org/10.1109/ICECCS.2009.35>
- [45] Ou, S.-C., Sung, W.-T., & Hsia, S.-J. (2002). Development of intelligent virtual reality web-based robotics for manufacturing applications. In IEEE International Conference on Industrial Technology (ICIT) (pp. 348–353). IEEE. <https://doi.org/10.1109/ICIT.2002.1189195>
- [46] Song, J.-S., Seo, M.-S., & Hae-Ryung, K. (2004). Open modes between real and virtual manufacturing devices. In SICE Annual Conference (pp. 2588–2591). IEEE. <https://doi.org/10.1109/SICE.2004.1378892>
- [47] Tzafestas, C., Palaiologou, N., & Alifragis, M. (2006). Virtual and remote robotic laboratory: Comparative experimental evaluation. IEEE Transactions on Education, 49(3), 360–369. <https://doi.org/10.1109/TE.2006.879253>
- [48] Vázquez Hurtado, C., Rojo Valerio, A., & Ruvalcaba Sánchez, L. (2010). Virtual Reality Robotics System for Education and Training. In 2010 Electronics, Robotics and Automotive Mechanics Conference. IEEE.
- [49] Redondo Verdú, C., & Sempere Maciá, N. (2022). Mixed reality for industrial robot programming (Dissertation). Retrieved from <https://www.diva-portal.org/smash/get/diva2:1679151/FULLTEXT02>
- [50] Ong, S. K., Yew, A. W. W., Thanigaivel, N. K., & Nee, A. Y. C. (2020). Augmented reality-assisted robot programming system for industrial applications. Robotics and Computer-Integrated Manufacturing, 61, 101820. <https://doi.org/10.1016/j.rcim.2019.101820>
- [51] Parak, R. (2020–2021). A digital-twins in the field of industrial robotics integrated into the Unity3D development platform [GitHub repository]. GitHub. https://github.com/rparak/Unity3D_Robotics_Overview
- [52] Parak, R. (2020–2023). A digital-twins in the field of industrial robotics integrated into the Unity3D development platform [GitHub repository]. GitHub. https://github.com/rparak/Unity3D_Robotics_Overview
- [53] Parak, R. (2020–2023). A digital-twins in the field of industrial robotics integrated into the Unity3D development platform [GitHub repository]. GitHub. https://github.com/rparak/Unity3D_Robotics_Overview

[54] Franchetti, F. (2024). EGM-for-ABB-robots: Using Externally Guided Motion (EGM) to control ABB robots in C# [GitHub repository].

GitHub. <https://github.com/lurobotics/egm-for-abb-robots>

[55] Engström, A. (2021). *Designing an interface for teach pendants, with focus on novice robot users: Investigating users experience in early usage of a teach pendant*

(Master's thesis). Retrieved from <https://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-185069>

8 Appendix

8.1 User Transcripts

Note: This transcript was reconstructed with the assistance of ChatGPT based on the original iPhone audio recordings. All transcript content was reviewed and approved by the researcher.

Portions of the recordings that were outside the scope of the study, such as unrelated conversations or background chatter were excluded during the transcription process to maintain clarity and relevance.

- User 1

Facilitator:

You'll start by placing waypoints through a series of spheres numbered 1 through 7. Try to follow the wave path as closely as possible. Use the joint controller, highlight, and axis features if helpful.

User 1:

Working in 3D space was a bit of a challenge. But once I figured out it was mostly joint one and two, it became manageable.

There was noticeable lag at first—it was very jerky. After restarting and turning off the screen recording, the performance improved significantly.

Facilitator:

If it helps, you can switch camera angles or get physically closer to improve tracking.

User 1:

Yeah, I noticed the tracking got better when I moved closer.

Also, being left-handed made it harder to use the controller—it might help to have an option to switch hands in the UI.

I avoided using my left hand because of the setup.

Facilitator:

You're doing well. Keep adding the waypoints through each sphere.

User 1:

I found myself trying to be very precise, but the system still felt jittery. The perfectionist in me wanted it to be exact.

I didn't use the highlight or joint axis display—once I understood the joint behavior, it felt intuitive enough.

In hindsight, I should have adjusted joint three to follow the top slope better, but I skipped it toward the end—I was getting a bit tired.

Facilitator:

You can now play it back and review the path.

User 1:

Watching it replay is satisfying.

It actually followed the line pretty well. I could've been more accurate, but slowing down would've helped.

It would be really helpful to export the waypoint data and compare it to the ideal path. That would make evaluating performance more objective.

Facilitator:

Great. The next part is a maze task. Same idea—place waypoints, but this time there are 14 points, and you need to keep the robot's path within the wall constraints.

User 1:

Alright. This feels like a solid challenge—it really exercises your spatial reasoning.

It would be cool to have variations of this to test complexity limits.

After a while, though, the joint manipulation becomes tiring—especially with the interface responsiveness.

Facilitator:

Totally understandable. You can take off the headset when you're done.

User 1:

Thanks. This was a really interesting task. Slight dizziness aside, I think it's a good interface overall—it just needs a few quality-of-life improvements.

- User 2

Facilitator:

Let's begin by testing the interface. Try using the joint controller, and explore the “highlight” and “show joint axis” features. You can also use the camera controls for better viewing.

User 2:

Got it. I'm adjusting the camera now. Using the controllers together feels manageable. I've pulled the panel closer.

Facilitator:

Now for the main task: you'll move the robot's end effector through spheres numbered 1 to 7. At each point, place a waypoint. When you're done, play back the motion path.

User 2:

Alright. The robot's moving really fast. The velocity is difficult to control—it overshoots when I make slight changes.

Facilitator:

Try moving more slowly. The system responds in real time, so high-speed inputs can trigger velocity errors or cause RobotStudio to lag.

User 2:

I see. Yeah, joint 5 seems really sensitive. When I rotate it too quickly, the robot freezes or doesn't respond.

Facilitator:

Joint 5 rotates the wrist. Just ease into the movement, especially with joints that affect tool orientation.

User 2:

This UI panel moves with the world—it's a little disorienting. I lose track of it when I move my head.

Facilitator:

Yes, it's world-anchored. Try holding it in place or repositioning yourself to stay centered.

User 2:

The interface is tough to use. The sliders are too sensitive. I've had to redo a few points already—it's frustrating.

Facilitator:

Let me reset the system. We'll continue after that.

(brief system reset)

Facilitator:

Go ahead and finish the sphere task, then play it back when ready.

User 2:

Okay. Playback looks better now. The overall motion is smoother, though I still had some issues with control accuracy.

Facilitator:

We'll now move to the second task—a 3D maze. Similar to before, but with tighter constraints. Move through points 1 to 14, keeping all waypoints within the maze walls.

User 2:

Understood. Do I start from where the robot is now?

Facilitator:

Yes, add a waypoint at the starting position and continue from there.

User 2:

Okay. The axis indicator popped up again. I'll toggle it off—too much visual clutter.

Facilitator:

That's joint 6's visualization—you can disable it anytime by clicking its button again.

User 2:

I accidentally cleared the path. I was trying to hit "Play" but pressed the wrong button.

Facilitator:

No problem. I'll save what we have. The hotkeys can be sensitive, so we can add a lockout or confirmation in future versions.

User 2:

This task is more demanding. The joint control is tedious, and the velocity issue keeps coming back. I feel like I discovered a bug—some sliders become unresponsive after fast movements.

Facilitator:

It's possible. If velocity input exceeds a threshold, RobotStudio might suppress updates. That's a known limitation we're tracking.

User 2:

Got it. I'll continue slowly then. Okay, that's the last point. The playback path stayed within the maze, but it took a lot of effort.

Facilitator:

Great job. You can remove the headset now.

User 2:

Thanks. The task was useful for learning robot motion planning, but the controls definitely need fine-tuning.

- User 3

Pre-Task Familiarization & User Introduction

Facilitator:

You can also use your hands if you want—though most people stick to the controller.

User 3:

Oh, I can? I'll just stick with the controller for now.

Facilitator:

Cool. So first, click the Meta button on your right controller and reset the view—look in the direction you want your screen to be, then click anywhere.

User 3:

Done. Okay, I think I've got the UI grabbed now—yeah, you have to hold it, right?

Facilitator:

Exactly. Keep it held and move around the scene using the camera panel. Try going to a different view, then come back to the home position.

User 3:

All right—cool. That works well.

Facilitator:

Great. Now try moving a joint using the joint controller panel. Just be gentle—you want to avoid going too fast or too far out of range, or it'll trigger a guard stop.

User 3:

Okay. I'll be slow with it. I see that the sliders control the joint rotations.

Facilitator:

Yep. You can also use the highlight feature to visualize which joint you're about to move, and the “S” button to show the joint's rotation axis.

User 3:

Nice. I just added a waypoint—there's a red dot now. That's really intuitive.

Facilitator:

Great! You can add more and play them back. It'll move the robot through those positions. If you want, clear them and try again.

User 3:

Okay, playing it now... wow. This is really cool—definitely better than coding it in RAPID first and hoping the robot can follow it.

Facilitator:

Exactly. It gives immediate feedback. Now let's move into a task to see how you handle structured motion planning.

Task 1: Maze Navigation (Joint-Based)

Facilitator:

You're already positioned at waypoint 1. Go ahead and add the first waypoint.

As a reminder, don't clear the waypoints when you finish—I'll be saving your path for evaluation.

User 3:

Okay, added. It's a bit laggy at the moment.

Facilitator:

That's likely due to system load. Try to continue. Just a heads-up: there's no collision system here, so the robot won't stop even if it passes through maze walls.

This task focuses on joints 1 to 3—joints 4 through 6 are mainly for fine-tuned movement.

User 3:

Got it. Yeah, I can't see the highlight or joint axis stuff clearly—the maze blocks the view.

This definitely feels more realistic though—like an actual obstacle's in the way.

Facilitator:

That's intentional. The goal is to simulate real-world spatial constraints where full visibility isn't always possible.

User 3:

I'm having some bounce-back on the panel—it's not staying stable in my view.

Facilitator:

You can try pulling the panel closer, or adjusting your view. Moving with the panel helps maintain control.

User 3:

I'm rotating now... Wait, where's the next point? It's tough to reach—it's too close to me.

This isn't the best tracking spot. I had better luck earlier standing slightly off-center.

Facilitator:

You're doing fine. Keep adjusting—remember, you only need to use joints 1 through 3 to complete this maze.

User 3:

Okay, added the next one. Playing it back now... cool, it's moving smoothly.

Facilitator:

Perfect. Give me a moment—I'll pause here and save your waypoint data. Don't clear anything.

User 3:

That wasn't too bad. I noticed I haven't really used joints 4–6 much.

Facilitator:

That's expected. For this task, they're not really needed. They're more useful in tool orientation or precision manufacturing.

User 3:

Makes sense. I just jogged through the maze with the first few joints.

This reminds me a lot of how I'd approach joystick controls in a game, honestly.

Facilitator:

That's a good analogy. The whole idea is to build muscle memory and spatial intuition.

User 3:

Yeah, it's surprisingly natural once you get used to the movement limitations. I'm curious—this is actually controlling a real robot, right?

Facilitator:

Yes—this is synced with RobotStudio and an actual ABB IRB via RWS and EGM. Movements reflect on the physical robot as well.

User 3:

Whoa. That's amazing. It makes sense now why the guard stops matter so much.

Post-Task Reflections

Facilitator:

You mentioned a few times the slider felt a bit jittery—was it difficult to make precise movements?

User 3:

Yeah, especially with the lag. Sometimes it jumps further than I intend, and I'd prefer clearer feedback on range limits. Maybe even a visual cue for slider boundaries?

Facilitator:

That's great feedback. Would it help if, instead of using sliders, the controller's orientation was used—like tilting for joint rotation?

User 3:

Oh yeah! Like if I just select a joint and then tilt the controller to move it? That'd feel super natural—like a physical extension of your hand.

Facilitator:

Would you say the waypoint system was intuitive?

User 3:

Yes, though the add button could be a little bigger. I hit delete once by mistake.

It'd be nice to map those to controller buttons too—like “A” to add, “B” to delete. Less UI clicking.

Facilitator:

What about the joint highlighting—did it help?

User 3:

Definitely. I'd even go further—when I move a slider, it should automatically highlight that joint. I shouldn't need to manually toggle it.

Facilitator:

And camera navigation—was it easy to position yourself?

User 3:

Pretty smooth. Though joystick-based navigation might be better than predefined views. Or maybe teleportation to fixed points?

Facilitator:

Noted! Lastly—how did the VR experience compare to your usual robotics workflows?

User 3:

Honestly, it's a game-changer. In RobotStudio, you can't feel space. Here, I can intuit how far a joint moves and where things collide. It's immersive and great for learning.

Facilitator:

That's exactly what I hoped to hear. Thanks for all the thoughtful feedback.

User 3:

No problem. This was fun—and fast, too. I think I finished in under 7 minutes.

- User 4

Pre-Task Discussion (Context & Reflections)

Facilitator:

So how has your experience been working with the robot so far?

User 4:

It's been okay. I started doing it on the sidelines during IT²TR. I wasn't fully paying attention back then—I just wanted to get it done.

But now with the independent study, I'm doing everything myself, and I finally understand how it all works. I know how to fix my scripts and simulations now.

Facilitator:

So it's just you now?

User 4:

Yes—just me and the robot. It's challenging, but it's also rewarding. Back in IT²TR, the FlexPendant felt intimidating and confusing.

It didn't really guide you, and I often moved the robot incorrectly. But now, with more experience, I feel much more comfortable.

Facilitator:

That's exactly what I'm trying to solve with this system—making the first experience easier and more intuitive.

User 4:

Totally. When you're new, you don't even know what each joint is. It's not until you start moving them that it begins to make sense.

Facilitator:

Have you used the Quest headset before?

User 4:

Only in a previous research project related to ceramics. I had issues with cable length back then—it was hard for others to use during demos.

Also, I get nervous presenting, even in student settings, so I just want everything to work smoothly when I press play.

Facilitator:

Totally understandable. What other classes are you taking right now?

User 4:

I'm taking a machine learning class that focuses on text-based data, and a seminar on Human-Robot Interaction (HRI). That one's really cool—we're combining HRI and reinforcement learning. Nick is also in that seminar.

Facilitator:

Do you think you might do a robotics-based thesis?

User 4:

Maybe. I'm still figuring it out. I don't like picking just one thing—I'm more about the range of work I do.

But my mom pointed out that I seemed happiest when working in the lab with the robot, so that might say something.

Maze Task (Timed)

Facilitator:

Okay, you're already at position one. First, add a waypoint there using the Waypoint Panel. Then continue through the maze, placing a waypoint at each target.

User 4:

How do I add a waypoint again?

Facilitator:

At the top—it's the Waypoint Panel. Let me know when you're ready.

User 4:

Okay, I'm ready. Adding the first one now.

Facilitator:

Great. Be gentle with the joint movements—quick changes can trigger velocity issues.

User 4:

Got it. It's only using joints one through three, right?

Facilitator:

Yes, joints four through six are for fine-tuned movements, not needed here.

User 4:

Okay, I'll just focus on the main ones. This layout is actually pretty simple—it guides you.

Facilitator:
Exactly. The goal is to test both speed and accuracy under mild constraints.

User 4:
Tracking is okay so far. A little tricky at points, but manageable.
I almost missed one waypoint—it was a bit far, but I adjusted.

Facilitator:
You're doing great. You're under three minutes so far.

User 4:
Nice. This is fast-paced, but easier than expected once you figure out the movement flow.

Facilitator:
Be sure to hold onto the panel—if you let go, it may drift out of view.

User 4:
Yeah, I've got it. Just added another waypoint.
I think I missed one earlier, but added it now.

Facilitator:
Perfect. That's the full sequence. I'll pause the session to save your data—please don't clear the path.

User 4:
Understood.

Facilitator:
You're done! Just saving your motion path and waypoint data now.

User 4:
Was I supposed to use negative or positive values? I think I mixed them up.

Facilitator:
No worries—all right-side values are positive, left-side values negative. It's just a coordinate reading convention.

User 4:
Makes sense. Thanks—this task was smooth once I got into it.

- User 5

Pre-Task Familiarization and Contextual Feedback

Facilitator:
So I'm aiming for this tool to become something useful for robotics education—especially for people who are just getting into it. It's designed for someone like you: familiar with robots, but maybe newer to VR-based control.

User 5:

I really resonate with that. I've used real robot controllers a lot, so I know what I'm doing now—but that came after a lot of practice. The early stages were tough.

Facilitator:

Exactly! I'm trying to skip that initial learning curve—make it easier to understand joint movements and system feedback from the start.

Okay, you should see the interface now. The left controller pulls the UI panel toward you, and the right controller is for selecting and interacting.

User 5:

Got it. The grabbing motion with the left controller is really intuitive.

Facilitator:

You'll start by connecting to RobotStudio via the connect panel—once it's green, it's live. Then you've got camera views, joint sliders, waypoint controls, and highlight features to show which joint you're adjusting.

User 5:

Okay, cool. I like that you can visualize the joint rotation axis. That's usually something that takes time to figure out with a physical robot.

Facilitator:

Exactly. You can highlight any joint with the “H” button and show its axis with “S”. When you move joints to a position, you can save it as a waypoint, then play it back or export it.

User 5:

This is so much more intuitive than Grasshopper or the FlexPendant—especially seeing waypoints appear in 3D.

Facilitator:

Try playing with it and see how it feels. You can clear waypoints and return home too.

User 5:

Got it. Honestly, the interface feels like a game. I'm already thinking how this would be great for beginners.

Facilitator:

That's the goal—making robotics approachable through interaction and feedback. Ready to start the actual task?

Maze Task (Timed Motion Planning)

Facilitator:

Before starting, click the Meta button on the right controller and reset your view to a comfortable position. Let me know when you're ready.

User 5:

All good now. Just getting familiar with the controls again. I'd like to be precise with this.

Facilitator:

Great. You'll use joints 1 to 3 only—this task mimics a real-world environment with partial visual obstruction, so some features like highlight and joint axes might be harder to see due to the maze structure.

User 5:

Got it. I want to beat the current record time, but also stay accurate.

Facilitator:

Sounds good. Remember to add a waypoint at position one—looks like you didn't do that yet. Also, try to stay within tracking range; if you see the white orb, you're drifting out of bounds.

User 5:

Okay. That makes sense. I'll try to be more stable—my hands are a bit shaky.

Facilitator:

No worries. Just move slowly and gently; the system is sensitive to high input velocity. There's no collision detection, so it won't stop automatically if the robot passes through the maze walls.

User 5:

Understood. There's some lag, and joint control feels really sensitive. I'll try repositioning the panel for better accuracy.

Facilitator:

Good idea—pull the panel closer before you adjust your view. That should help reduce large movements.

User 5:

That helped. I also noticed that vertical (up/down) movement feels less sensitive than left/right. My hands are shaking, so it's tricky to stay precise near the end.

Facilitator:

Take your time. You're doing well. Did you add a waypoint at position six?

User 5:

I don't think I did—but I've added it now.

Facilitator:

Great. You're almost done. Go ahead and add the last waypoint.

User 5:

Okay—done now. This reminded me a bit of game controls. It's challenging, especially with small movements, but satisfying to complete.

Post-Task Reflections

Facilitator:

That was excellent! How did you find the joint sliders overall?

User 5:

They're intuitive, especially with the highlight feature. Without that, I'd be constantly guessing which joint I'm controlling.

But yeah, the sliders feel a bit jaggy—it's hard to fine-tune positions. There's a delay between my input and the movement.

Facilitator:

That's something I've been working on optimizing. What about the labels under each joint slider—left, right, up, down?

User 5:

Super helpful. It makes the directionality of each joint easy to understand, even without prior experience. I also like your idea of triggering highlight automatically when adjusting a joint.

Facilitator:

Thanks! What did you think of the diagnostic panel?

User 5:

I didn't use it much. It might be more useful when doing precision positioning, but for general motion, I focused more on the visual feedback.

Facilitator:

Fair point. How did you find using this in VR with the Quest headset?

User 5:

It's more immersive, for sure. But honestly, it could be tiring for longer sessions. It's a great demo or training tool, though.

Facilitator:

And overall, would you say this lowers the barrier for new users?

User 5:

Absolutely. This makes robotics approachable and even fun. It's the kind of tool I wish I had when I was first starting out.

- User 6

Familiarization & Sphere Task

Facilitator:

Let's start by testing the joint sliders. Move them gently—fast movements can cause issues.

Now use the highlight feature and try showing the joint axes.

Add a waypoint, then move one joint and add another. Do this a few times, then play back the path.

User 6:

Done. Playback worked well.

Facilitator:

Great. Now clear the waypoints and return to home position.

I'll load your next task—navigate to spheres 1 through 7 and add a waypoint at each.

User 6:

I can see them. The UI doesn't follow my view unless I hold it.

Facilitator:

Yes, the UI is world-anchored—you need to pull it along using the trigger. We've considered making it face the user's view by default.

User 6:

That might help. I've done that in Unity with text objects. But sometimes you don't want it to follow you—it could block your view.

Facilitator:

Exactly. That's why the UI can be manually placed.

User 6:

The highlight and show features help. I find joint 4 the hardest—it takes a few adjustments.

It might help to use Cartesian movement for up/down and left/right. The sliders feel too sensitive.

Facilitator:

Yes, Cartesian control is an alternative. For this task, we're sticking with joint-based control.

Try clicking the middle of the slider to avoid angular glitches.

User 6:

That worked better. Finished the path. The task helped me get used to the system.

Maze Task

Facilitator:

Now we'll move on to the maze task. Don't clear the waypoints—I'll be recording your performance.

You'll follow the same logic as before: place waypoints at each numbered point (1–14).

Use only joints 1 to 3. Let me know when you're ready, and I'll start the timer.

User 6:

Ready.

Facilitator:

3... 2... 1... Go.

User 6:

Movement feels smoother this time. I'm trying not to rush to avoid triggering RobotStudio errors.

Facilitator:

You're doing well. Remember to avoid colliding with the maze walls and to hold the UI when adjusting your view.

User 6:

Got it. Oh, I accidentally clicked Play—it reset the position.

Facilitator:

That's okay. Just continue. Avoid clicking Play mid-task—it resets everything.

User 6:

I think I missed point 12—it's embedded in the floor.

Facilitator:

Try going back and adding it. Make sure you're in position before clicking Add Waypoint. Let me know if it's not responding.

User 6:

Fixed it. Point 12 added.

Facilitator:

Great. You're finished. You can take off the headset now.

User 6:

Thanks!

Post-Task Feedback

Facilitator:

What did you think of the joint controller?

User 6:

The slider could be longer or the control marker bigger. Sometimes I thought I moved it, but it didn't register.

Left/right movement feels inconsistent. I often forgot to hold the UI when changing perspective.

Facilitator:

Did the view-switching work well for you?

User 6:

I avoided switching views—I got used to a particular camera angle that made the joint directions intuitive.

Left and right views were most informative for me.

Facilitator:

Were the sphere positions reachable?

User 6:

Yes, they were well-spaced.

Facilitator:

Did the VR environment provide a useful experience compared to traditional tools?

User 6:

Yes—I especially liked seeing the full structure in 3D. In software like RobotStudio, I usually can't visualize some elements as clearly.

I'd be interested in comparing this with my previous paths.

Facilitator:

Was it easy to add, play, and clear waypoints?

User 6:

Mostly. Sometimes the added point wasn't visible—it might have been hidden behind geometry. A color change or confirmation would help.

Facilitator:

Did the headset or controls cause discomfort?

User 6:

No issues with the controller. The headset was fine overall, just mildly uncomfortable from prolonged use.

- User 7

Familiarization Task

Facilitator:

Let's begin by setting up your view. Click the Meta button and reset your position. Use the left controller to pull the UI panel toward you, then place it where it's comfortable.

Use the right controller to interact with the panel.

User 7:

Got it. The system feels a bit laggy, probably due to the GPU, but it's manageable. I've used the Quest and robots before, including various ABB models.

Facilitator:

Perfect. Let's begin. Move any joint slowly, and try using the highlight and show rotation axis features. Add a waypoint at your current position, move a joint, then add another waypoint. Finally, play it back.

User 7:

Done. It's working well. Highlight and axis features are helpful, but I'm familiar with which joint does what, so I mostly relied on intuition.

Facilitator:

Now clear your waypoints and return to the home position. We'll begin the sphere task.

Sphere Task

Facilitator:

Go through spheres 1 to 7, adding a waypoint at each. You can change the camera angle, but remember to pull the UI panel with you.

User 7:

The fourth sphere was tricky—required a few joint adjustments. Joints 4–6 are definitely better for finer movements.

The highlight helped for checking directions, though I mostly didn't need it. I noticed some occasional Unity lag, especially with joint 3's slider.

Facilitator:

Yes, that's a known performance issue with the system. Good job reaching each point.

User 7:

Done with the last point. Played it back—it looks smooth.

Maze Task (Timed)

Facilitator:

This next task is similar, but timed. You'll follow a predefined path in the maze, adding waypoints at all marked points (1–14). Try to stay accurate and inside the maze walls. Let me know when you're ready.

User 7:

Ready.

Facilitator:

3... 2... 1... Go.

User 7:

Movement is going well. The controls feel responsive enough now. I'm adjusting the camera when needed to make sure the waypoints are accurate.

Facilitator:

Good technique. Accuracy matters more than speed here, since we'll compare your path to the original reference path.

User 7:

I'm getting through it. Being careful now—gentle slider movements seem to help with performance.

Facilitator:

Almost there. And... done!

User 7:

Great. I feel like I hit most points accurately.

Post-Task Feedback

Facilitator:

Were you able to reach all waypoints inside the spheres and the maze?

User 7:

Yes. It required multiple joint adjustments at times, but I could reach them once I knew which joints to use.

Facilitator:

Did you use or benefit from the highlight and axis features?

User 7:

They were useful, but I didn't rely on them much. I already knew joint behavior from experience.

Facilitator:

Did you notice the joint direction indicators under the sliders?

User 7:

No, I didn't see those. I just used instinct.

Facilitator:

How was your experience with the controller?

User 7:

Fine overall. Some joints didn't respond perfectly due to lag, but that's probably a performance issue, not a UI flaw.

Facilitator:

Did the VR headset help in understanding spatial layout?

User 7:

Yes. It really helped visualize 3D structure and made spatial planning more intuitive.

Facilitator:

What did you think of the exercise?

User 7:

It was fun—reminded me of a game I played before. It could work well as a tutorial for robot control.

- User 8

Familiarization

Facilitator:

Use the left controller trigger to pull the UI panel toward you and position it where it's comfortable. Use the right controller to interact with the UI and click Connect. Try switching the camera angle while holding the UI panel.

User 8:

Okay, done. Positioning is comfortable.

Facilitator:

Great. Now move joint 1 slowly. Click the Highlight (H) and Show Axis (S) buttons to visualize motion direction.

Add a waypoint, then move joint 1 again and add another. Play it back.

Now clear the path and return to home.

User 8:

All clear. Ready to proceed.

Task 1 – Sphere Navigation

Facilitator:

Now complete the first task: go to sphere 1, add a waypoint, then repeat for spheres 2 to 7. The goal is to assess joint control and spatial coordination. Use the camera panel to adjust your view as needed.

User 8:

Got it. Yes, I've used this ABB model downstairs during ITTR. I usually work with Grasshopper and RobotStudio—less so with the teach pendant.

Facilitator:

How does it compare?

User 8:

I prefer Grasshopper—it gives more parametric control over paths. The teach pendant requires switching between coordinate spaces, which can get complex.

Facilitator:

You're doing well—sphere 4 is the trickiest due to joint orientation.

User 8:

Yes, but I'm getting used to it. Controller feels fine overall.

The only issue is occasional jitter or flashing—it's probably due to RWS and EGMs running in parallel.

Facilitator:

Yes, that's a known connection issue. Play it back when you're ready.

User 8:

Done. Everything seems to have worked.

Facilitator:

Were you able to reach all spheres and use the joint sliders comfortably?

User 8:

Yes. Camera switching helped me understand the space better. It was manageable overall.

Task 2 – Maze Navigation (Timed)

Facilitator:

Now for the timed maze task. Same idea—follow the green path from point 1 to the end, placing waypoints at each stop. I'll compare your path to a predefined one to evaluate accuracy.

User 8:

Understood.

Facilitator:

3... 2... 1... Go.

User 8:

Moving through... joints 1 to 3 only, correct?

Facilitator:

Yes—only the main joints for this task.

User 8:

I noticed that fast slider input triggers a robot stop in RobotStudio. Probably due to sudden high-velocity movement.

Facilitator:

Yes, let me reset RobotStudio for you. Don't worry, your waypoints are still saved. Resume from point 9 once it reloads.

User 8:

Okay—resumed. I'll be more careful with movement now.

Facilitator:

You're doing well—just two more points.

User 8:

Done.

Facilitator:

Great. I'll exclude the time taken for the reset in your final score. Your path will be saved for analysis.

Post-Task Feedback

Facilitator:

Did the camera panel and 3D view help with spatial understanding?

User 8:

Yes, it helped a lot. The immersive space gave a clearer idea of spatial relationships than RobotStudio alone.

Facilitator:

Did you find the joint controller responsive?

User 8:

Yes, but sometimes I had to anticipate lag or overshoot—so I'd stop dragging the slider before hitting the final point.

The motion would “catch up” afterward.

Facilitator:

Interesting. Did you use the teach pendant often before?

User 8:

Rarely. I mainly use Grasshopper because it gives me geometric control over paths. Teach pendant requires too many step-by-step changes in space and coordinate frames.

This VR system is more intuitive for visual feedback, though real-world applications still need careful alignment.

Facilitator:

Was this your first time working in VR for robotics?

User 8:

Yes. Compared to watching remote tutorials or Zoom explanations, this feels much more natural. Especially for understanding things like work objects and reference frames.

- User 9

Maze Task (Timed)

Facilitator:

Okay, so this is the final task. You'll follow a green path through the maze, adding waypoints from point 1 to point 14. The task is timed, and I'll also compare your path accuracy against a predefined reference.

Use joints 1 to 3 only, and remember: gentle, slow movements work best. Let me know when you're ready.

User 9:

Ready. I've got the panel where I want it.

Facilitator:

Great. 3... 2... 1... Go.

User 9:

Okay, adding the first waypoint... The joint movement feels a little sensitive—gotta go slow. Moving to point 2 now.

Facilitator:

Nice. You can use the camera panel if the next points aren't clearly visible.

User 9:

Yeah, just switched to a side view. That helps with alignment.

Point 4 is a bit tight. I had to rotate the base a bit more to get around the corner.

Facilitator:

You're doing well. Make sure the end effector stays within the maze walls.

User 9:

Yeah, I almost clipped the wall on that last one. Adjusting slightly...

Slider's a bit touchy—I tried to do a small correction, but it jumped more than expected.

Facilitator:

That's common—try clicking along the slider bar instead of dragging the dot for finer adjustments.

User 9:

Good tip. That helped.

Now at point 10... moving toward the last stretch.

Facilitator:

Looking good. You're still within the five-minute mark.

User 9:

Okay, last point... Got it.

Facilitator:

Great. That's the full path. You can take off the headset now.

Post-Task Feedback

Facilitator:

Were you able to reach all the points comfortably?

User 9:

Yeah, though a few tight corners required multiple small joint adjustments.

Facilitator:

Did the camera panel help?

User 9:

Definitely. Switching views made it much easier to judge distances and avoid errors.

Facilitator:

How did you find the joint sliders?

User 9:

Mostly okay, but they were a bit sensitive. Sometimes a slight hand movement caused a bigger-than-expected motion. A fine-tune or plus-minus control could be helpful.

Facilitator:
Was the UI layout and VR interface intuitive for you?

User 9:
Yes, I liked being able to position the panel and see everything in 3D. That made the task more understandable than using just a flat screen.

Facilitator:
Any discomfort with the headset?

User 9:
Nope, it was fine throughout. Everything felt stable.

Facilitator:
Awesome. I'll save your waypoints and timing data for the final analysis. Thank you!

User 9:
Cool—thanks, this was fun!

- User 10

Setup & Interaction

Facilitator:
The system is running with both EGM and RWS connections to RobotStudio. This allows real-time control but may introduce lag. Be gentle with joint movements to avoid triggering faults.

User 10:
Noted. If I move my controller away from the UI, does it get worse?

Facilitator:
It helps to keep the UI closer. Also, instead of dragging the joint sliders, you can try clicking along the bar for finer adjustments.

User 10:
Okay, I'll add a waypoint now.
If I move the joint slightly, does that signal to RobotStudio?

Facilitator:
Yes. I've limited the joint range to avoid over-travel. Once a position is reached, it gets stored as a waypoint along with the robot's orientation.

User 10:
Understood. It feels a bit laggy—it's jumping around. Joint control is not very precise.

System Lag & Technical Issues

Facilitator:

Let me pause the timer—something seems off. Try moving another joint.

User 10:

Still not responsive. It says “MCI Stopped.” Nothing's working.

Facilitator:

That’s an EGM-related issue. I’ll restart the system.

User 10:

Okay, thanks. It reset, and the robot’s back to home. But I still feel like the system is slow—probably the laptop GPU. It’s from 2018 and can’t handle smooth VR tracking.

Facilitator:

Yes, the GPU is likely the bottleneck here.

User 10:

Joint control feels jerky. Even when I click near the slider dot, it doesn’t nudge properly. Dragging the dot leads to overcorrection. It's hard to get fine precision.

Facilitator:

Some users found that breathing steadily and using slow, stiff motions helped. The system is very sensitive to velocity input.

User 10:

Yeah, even when I hit the right position, letting go shifts it again. A plus/minus nudge option would really help for small adjustments.

The camera and UI panel themselves work great—very intuitive. But if the sliders were steadier, this would be a much smoother experience.

Task Completion

Facilitator:

Are you able to see point 11?

User 10:

No, I think I missed it. I’ll go back and add it manually.

Facilitator:

That’s fine. As long as it’s added in the right area, it’ll count.

User 10:

Thanks. Yeah, I definitely want to try this again on a better machine.

This system is promising, but the main issue I’ve faced is laggy, overly sensitive joint sliders.

Facilitator:

Got it. That’s extremely useful feedback. You’re done for now—thank you!

8.2 Task 3 Results

8.2.1 Target Coordinates for Maze Path

Point #	Target X	Target Y	Target Z
1	2.67E-05	0	0
2	-0.487	8.08E-05	0.1360965
3	-0.898	1.47E-04	0.6691297
4	-1.16	-0.483	0.5874476
5	-1.200001	-1.028648	0.5754696
6	-0.9866219	-1.034121	0.163699
7	-0.9362202	-0.3978307	0.2125907
8	-0.6548386	-0.4018176	-0.05184698
9	-0.6963539	-0.9158731	-0.1209126
10	-0.2102451	-0.916471	-0.3108597
11	-0.1944523	-0.4019921	-0.2315655
12	0.6005764	-0.3989997	-0.08077264
13	0.3786488	0.1668746	0.2899644
14	0.1088104	0.170537	0.2000451

8.2.2 Participant Data

- User 1

Time Taken to Complete Task: 5 minutes 13 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.904	0.127	-0.4098081
3	-1.34	0.127	0.1583484
4	-1.57	-0.311	0.0264681
5	-1.636063	-0.8360348	0.006681025
6	-1.442896	-0.8362554	-0.3596088
7	-1.383564	-0.2823379	-0.3146202
8	-1.106256	-0.2826948	-0.5773905
9	-1.150164	-0.7842033	-0.6419346
10	-0.6591586	-0.7845802	-0.8405761
11	-0.6433905	-0.2549546	-0.7541799
12	0.1683618	-0.2553618	-0.6011016
13	-0.06335819	0.3110778	-0.2186747
14	-0.37191	0.3110921	-0.3212592

- User 2

Time Taken to Complete Task: 4 minutes 23 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-1.079906	0.029077	0.554658
3	-1.474086	0.029171	1.068118
4	-1.723514	-0.43266	0.971321
5	-1.779345	-0.953598	0.952877
6	-1.579169	-0.956697	0.576626
7	-1.505551	-0.35897	0.617545
8	-1.235588	-0.358774	0.369018
9	-1.266301	-0.86964	0.288092
10	-0.802118	-0.869604	0.106659
11	-0.782259	-0.371715	0.18439
12	0.015794	-0.364461	0.331498
13	-0.192694	0.1857	0.685252
14	-0.461379	0.191811	0.59513

- User 3

Time Taken to Complete Task: 5 minutes 38 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.936	0.127	-0.3903283

3	-1.34	0.127	0.1494732
4	-1.60	-0.365	0.07721746
5	-1.640209	-0.8870021	0.0659647
6	-1.409773	-0.8868017	-0.3804044
7	-1.351744	-0.2682621	-0.3333127
8	-1.083082	-0.2675967	-0.5764795
9	-1.129846	-0.7530476	-0.6477663
10	-0.6514186	-0.7530944	-0.8351132
11	-0.6378783	-0.2787592	-0.7552235
12	0.1700784	-0.2786415	-0.600156
13	-0.03707111	0.2654002	-0.2610074
14	-0.307886	0.2655585	-0.3629507

- User 4

Time Taken to Complete Task: 3 minutes 45 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.916	0.127	-0.4032134
3	-1.34	0.127	0.141119
4	-1.58	-0.296	0.06994921
5	-1.651369	-0.8304188	0.04969174

6	-1.432112	-0.8301736	-0.3759186
7	-1.371244	-0.275934	-0.3282149
8	-1.077974	-0.2759142	-0.5940761
9	-1.119905	-0.8219993	-0.6597497
10	-0.6276835	-0.8220077	-0.8440978
11	-0.6167506	-0.2939634	-0.7737491
12	0.1611692	-0.2610967	-0.6084821
13	-0.03723514	0.2651825	-0.2745442
14	-0.2765616	0.2652268	-0.3685561

- User 5

Time Taken to Complete Task: 5 minutes 52 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.889	0.131	-0.3315784
3	-1.27	0.131	0.1904216
4	-1.57	-0.381	0.07654297
5	-1.603168	-0.8981543	0.06633216
6	-1.387733	-0.9337571	-0.3412907
7	-1.337086	-0.2817392	-0.3015346
8	-1.061771	-0.2817509	-0.5526223

9	-1.105907	-0.7734803	-0.6212868
10	-0.6405543	-0.7733061	-0.8014941
11	-0.6287364	-0.3078001	-0.730688
12	0.1529123	-0.2444763	-0.595525
13	-0.07443988	0.3091998	-0.2123752
14	-0.3177527	0.3092599	-0.302777

- User 6

Time Taken to Complete Task: 5 minutes 55 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.970	0.126	-0.3688185
3	-1.31	0.126	0.06660104
4	-1.58	-0.376	-0.03399408
5	-1.613823	-0.8583857	-0.04892409
6	-1.40374	-0.8583386	-0.4052228
7	-1.329676	-0.2220306	-0.3442277
8	-1.140207	-0.2209027	-0.5233127
9	-1.018537	-0.8520198	-0.714847
10	-0.563156	-0.8519871	-0.8490548
11	-0.5585212	-0.3557868	-0.8014137

12	0.1980726	-0.3680341	-0.6288645
13	0.01272213	0.2027457	-0.3288481
14	-0.3685797	0.2738099	-0.3712893

- User 7

Time Taken to Complete Task: 3 minutes 25 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.8922385	0.1029031	-0.440416
3	-1.35	0.103	0.1025044
4	-1.58	-0.336	0.02492893
5	-1.630167	-0.893206	0.009608686
6	-1.438545	-0.8931832	-0.3545486
7	-1.368368	-0.2316887	-0.3013777
8	-1.104866	-0.2317286	-0.5538581
9	-1.15394	-0.7359712	-0.6406712
10	-0.7274169	-0.7359593	-0.8278112
11	-0.6418322	-0.2552218	-0.7536435
12	0.1633149	-0.2551837	-0.6030847
13	-0.04224575	0.2758782	-0.2599754
14	-0.2834777	0.2758925	-0.3545707

- User 8

Time Taken to Complete Task: 5 minutes 43 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.962	0.101	-0.4044141
3	-1.35	0.102	0.07903552
4	-1.58	-0.348	-0.004548728
5	-1.627922	-0.8448241	-0.02181619
6	-1.421455	-0.8446417	-0.3893053
7	-1.367407	-0.2931459	-0.3458759
8	-1.103737	-0.2929356	-0.58671
9	-1.14039	-0.7736394	-0.6487176
10	-0.6471747	-0.7733123	-0.8427603
11	-0.632368	-0.251579	-0.7535211
12	0.1537305	-0.2515142	-0.6068899
13	-0.05937564	0.2917655	-0.2448721
14	-0.3074721	0.2917073	-0.3373393

- User 9

Time Taken to Complete Task: 5 minutes 20 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.267	0.483	-0.3593669
3	-0.651	0.483	0.1415388
4	-0.911	0.00363	0.05883962
5	-0.9540892	-0.5306799	0.0451287
6	-0.7431192	-0.530688	-0.3514549
7	-0.6857004	0.08517313	-0.3061554
8	-0.4024134	0.08500099	-0.5642684
9	-0.4472208	-0.43223	-0.633177
10	0.03955173	-0.4322666	-0.8205764
11	0.05155277	0.06693769	-0.7462242
12	0.8445539	0.06697011	-0.5959036
13	0.622056	0.6518161	-0.2185723
14	0.337615	0.651798	-0.3168781

- User 10

Time Taken to Complete Task: 5 minutes 27 seconds

Final End-Effector Coordinates by Waypoint

Point #	Final X	Final Y	Final Z
1	2.67E-05	0	0
2	-0.202109	-0.113094	0.701577

3	-0.62328	-0.14224	1.230417
4	-0.854031	-0.618572	1.162694
5	-0.901794	-1.156479	1.185563
6	-0.630729	-1.150155	0.771602
7	-0.565154	-0.512064	0.772296
8	-0.448097	-0.530203	0.59399
9	-0.357338	-1.130015	0.445808
10	0.06467	-1.040173	0.266862
11	0.173022	-0.513155	0.350067
12	0.947651	-0.480954	0.488064
13	0.637972	0.020996	0.978846
14	0.455796	0.066779	0.84093

8.3 Repository Access

The full implementation codebase, including Unity, and RobotStudio project files used in JointLab are available at: <https://github.com/sherft2607/JointLab.git>