ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Robotic Arm Integrated into Wheelchair with Mixed Reality Interface

Team #12

YINUO YANG (yinuoy4@illinois.edu)
YILIN WANG (yilin14@illinois.edu)
YUNYI LIN (yunyiy3@illinois.edu)
XINGRU LU (xingrul2@illinois.edu)

Professor: Liangjing Yang
TA: Yun Long

May 27, 2025

Abstract

Wheelchair users often face substantial challenges when interacting with objects or performing tasks outside of their immediate reach. Furthermore, wheelchair users may experience limited situational awareness due to a primarily forward-facing field of view. This highlights the urgent need for innovative solutions that improve both accessibility and autonomy, allowing wheelchair users to interact more effectively and conveniently with their surroundings independently. In this project, we proposed and developed a system that offers wheelchair users an assistive, real-time robotic arm system with a mixed reality (MR) interface, as a solution to improve their independence.

Contents

1 Introduction

Wheelchair users face significant challenges when reaching for objects or performing tasks beyond their immediate reach, compounded by limited situational awareness from their fixed forward-facing perspective. Our proposed solution integrates a robotic arm with a mixed reality (MR) interface into a wheelchair platform. The system enhances users' daily living by assisting with object retrieval and button activation, while simultaneously expanding their field of view through a camera, enabling more effective and independent environmental interaction.

1.1 Function

Our solution integrates a depth camera that streams real-time visuals to a Mixed Reality (MR) interface, allowing wheelchair users to gain visual awareness of their surroundings, including blind spots behind them. Additionally, a robotic arm mounted at the back of the wheelchair can be controlled through MR, enabling users to perform assistive actions such as pressing buttons and interacting with objects beyond their physical reach. This system enhances both situational awareness and independent mobility, providing a more intuitive and convenient way for users to navigate and interact with their environment.

1.2 Block Diagram

Our solution is divided into 3 modules and the dataflow of this system are shown in Figure 1: the robotic arm module for precise object manipulation, the mixed reality module for intuitive human-robot interaction, and the mobile platform module for autonomous navigation. Together, these modules form a complete assistive system that improves wheelchair users' independent interaction with their surroundings. The dataflow of our system is demonstrated in Figure 2.

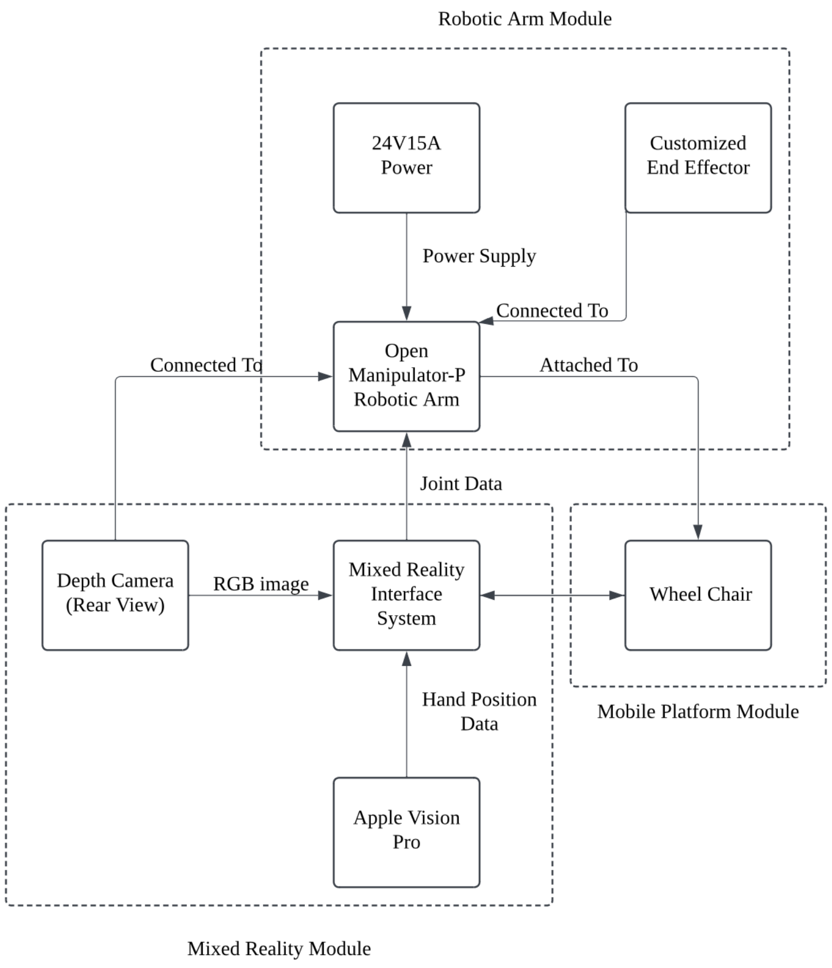

Figure 1: Block Diagram

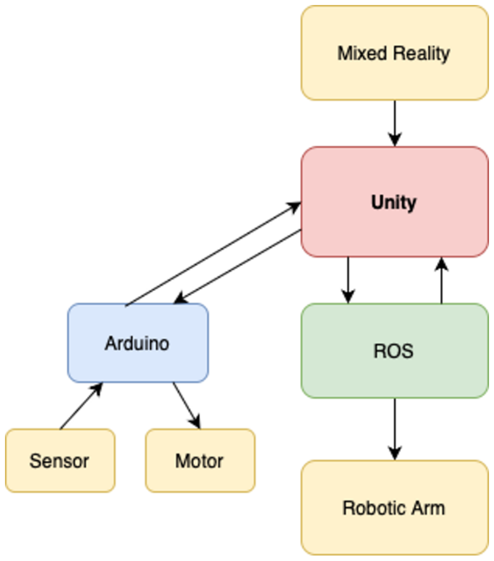

Figure 2: Dataflow Diagram

- **Robotic Arm Module:** This module consists of a customized end effector and the OpenMANIPULATOR-P robotic arm, powered by a 24 V-15 A power supply. The function of this module is to interact with the environment, such as to press buttons. It is physically mounted on the Mobile Platform Module (Wheelchair) and connected to the Mixed Reality Module for feedback and control.
- **Mixed Reality Module:** This module consists of the Apple Vision Pro, the Depth Camera, and the Mixed Reality Interface System, the core processor for interpreting inputs provided by the user and information regarding the surroundings. It sends precise commands to the Robotic Arm Module via joint data, enabling intuitive and adaptive control. The Depth Camera expands the system's field of view, and the Mixed Reality Interface System maps the motion of the user's hand for intuitive interaction with the robotic arm.
- **Mobile Platform Module:** This module consists of the wheelchair and an aluminum profile box, connected by suitable fasteners. The top of the aluminum profile box is designed with a rail structure, allowing the robotic arm to achieve a larger reach along the rail. This part completes the connection between the robotic arm and the wheelchair, forming the main physical structure of the entire system.

1.3 High-Level Requirements List

- **Precision:** The robotic arm should reliably press buttons with a diameter of at least **35 mm**, which is a common size of elevator buttons. The force applied must be sufficient to activate buttons without excessive pressure that could cause damage or failure.
- **Safety and Stability:** Users should be able to see both the front and rear environments through Vision Pro, while also adjusting the robotic arm's perspective to gain a broader field of view.
- **Reach:** The robotic arm should be able to reach a height from 110cm to 160 cm.

2 Design

2.1 Design Procedure

2.1.1 Customized End Effector

Form Factor Design

- **Gripper Shape Design:** Two configurations were considered:
  - A curved gripper driven by a gear mechanism, which conforms well to cylindrical objects but suffers from limited maximum opening and uneven force distribution.
  - A parallel gripper driven by a linkage transmission mechanism, which offers a wider gripping range, accommodates diverse object shapes, and ensures uniform force application.

  The parallel gripper was selected for its adaptability and stable gripping performance. Additionally, sponge padding was added to the inner surfaces of the fingers to increase friction and distribute contact force evenly. The sketches of both grippers are shown in Figure 3.

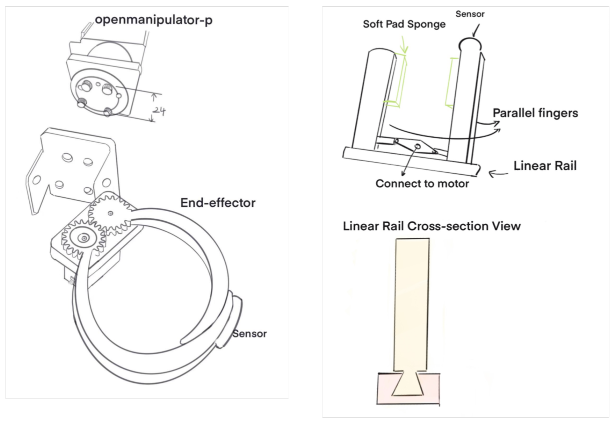

Figure 3: Curved Gripper (left) v.s. Parallel Gripper (right)

- **Actuation Selection:** Two servo motors were evaluated:
  - SG90 micro servo, featuring compact size and low cost but insufficient torque for heavier objects.
  - MG995 high-torque servo, providing higher torque output and greater durability.

  The MG995 servo was selected to meet gripping force requirements and ensure reliable performance under practical working conditions.

- **Material Selection:** Two materials were considered:

  - PLA plastic using 3D printing in the design lab, which supports fast prototyping but has low strength and poor wear resistance.
  - Professionally printed nylon, which offers high tensile strength, toughness, and fatigue resistance.

  Nylon was selected to improve structural integrity and enhance long-term durability.

- **Structural Refinement:** Two finger configurations were considered:
  - Symmetric flat-end fingers, which are simple but less suitable for precise control interactions.
  - An asymmetric design with one extended curved finger, which improves access to recessed panels and enables accurate button pressing.

  The asymmetric design with the extended curved tip was selected to support assistive use cases while maintaining gripping capability.

End Effector Control

For the end-effector gripper (including the force sensing resistor), we use an individual Arduino board to receive input from force sensing resistor and control the motor that enables the opening and closing of the gripper. There are two main pathways in the overall design of this component:

- Make the end-effector an individual component solely controlled by Arduino, creating an automated process where the gripper opens and closes for certain seconds after the FSR (force sensing resistor) detects force that generated from contact with object to be gripped.
- Transmit data between Arduino and Unity and design control procedures in Unity.

We chose the second one because the first one means hard-coding the gripper control, which is less flexible and introduces certain manipulation difficulties in real-world testing.

The force sensed by the front-end of the gripper will be converted to voltage value and sent to Unity as a variable value. We set a threshold for the voltage, halting the robotic arm once the threshold is reached to prevent potential damage to objects and the gripper. In Unity, the voltage value could be visualized, providing the user with more intuition of the robotic arm operation. More importantly, after establishing communication between Unity and Arduino, the user could control the opening and closing of the gripper via hand gesture at any time, without conforming to hard-coded procedures.

Design Tools:

- SolidWorks & Fusion 360: Models end effector and connection parts.

- Bambu Lab: Performs slicing on the modeled STL files and exports G-code for 3D printing.
- Creality 3D Printer: Prints our end effector and connection parts.
- Arduino IDE: Programs the microcontroller. It allows the user to write and upload code that reads analog data from a pressure sensor, calculates the corresponding voltage, sends the voltage over a serial port to Unity, receives commands like ″OPEN″ or ″CLOSE″ from Unity, and controls the servo motor or LED accordingly.
- Unity: Receives voltage data from the FSR and integrating it to the whole system.
- System.IO.Ports: Library used for establishing bidirectional communication between Arduino and Unity.

2.1.2 OpenMANIPULATOR-P Robotic Arm

For the selection of the robotic arm system, two primary options were evaluated based on mobility requirements, functional capabilities, and integration complexity.

- The UR3 industrial robotic arm has an external control box weighing about 15 kg, creating significant integration challenges for wheelchair mounting and movability, while the industrial-grade capabilities exceed our application requirements.
- The OpenMANIPULATOR-P provides integrated electronics and lightweight construction (5.76 kg total weight).

The final selection adopts the OpenMANIPULATOR-P system based on three key considerations: First, the self-contained design simplifies wheelchair integration by eliminating peripheral components. Second, the reduced weight minimizes impact on wheelchair maneuverability. Third, the sufficient functional performance covers essential tasks including lightweight object retrieval and interface activation. However, the official maintenance of this robot is likely outdated, and the compatibility issues between its ROS packages and our other subsystems have caused integration challenges.

Design Tools:

- Ubuntu 18.04 LTS (Virtual Machine)
- ROBOTIS OpenMANIPULATOR-P e-manual [1]
- ROS Kinetic Kame with core packages (package dependencies are shown in Figure 4)
- Official ROBOTIS repositories [2]:
  - `open_manipulator_p`
  - `open_manipulator_msgs`
- ROS-TCP-Endpoint is a ROS-side communication bridge provided by Unity's Robotics Hub.

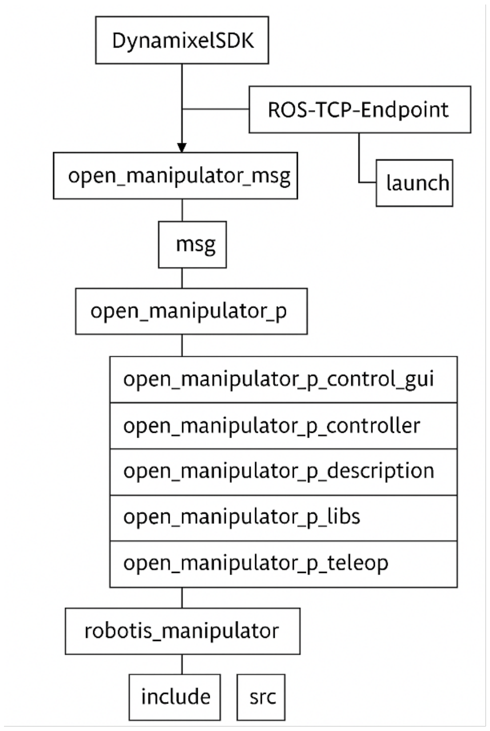

Figure 4: ROS Package Dependencies for OpenMANIPULATOR-P System

2.1.3 Depth Camera

We selected the Intel RealSense D435i depth camera due to its excellent balance of performance, compact size, and advanced sensing capabilities. Specifically, the D435i provides accurate depth perception with a wide field of view, enabling reliable spatial awareness in complex environments.

2.1.4 Mixed Reality Interface System

Pose Smoothing and Hybrid Interpolation To achieve smooth yet responsive end-effector motion, we first evaluated applying a Kalman filter to the full 6-DOF pose (position + quaternion). Although the filter effectively reduced noise, directly smoothing the quaternion components (w, x, y, z) disrupted the unit-norm constraint and geometric consistency of rotations. Consequently, we adopted a two-stage approach:

- **Position Smoothing:** Apply a standard Kalman filter to the 3D position measurement z_k:
$$\hat{x}_k = K_k\, z_k + (1 - K_k)\, \hat{x}_{k|k-1}, \tag{1}$$
yielding a jitter-reduced position p_{filtered}.
- **Hybrid Interpolation:**
  - *Position Command:* Blend the filtered position with the previous command p_{prev}

using linear interpolation with weight $\alpha = 0.6$:

$$p_{\text{cmd}} = 0.6\, p_{\text{filtered}} + 0.4\, p_{\text{prev}}. \tag{2}$$

– *Orientation Command:* Preserve quaternion integrity by performing spherical-linear interpolation between the previous orientation q_{prev} and the newly measured orientation q_{meas} with weight $\beta = 0.3$:

$$q_{\text{cmd}} = \text{slerp}\big(q_{\text{prev}},\ q_{\text{meas}},\ 0.3\big). \tag{3}$$

Reset-Position Function Initially, we did not include a reset-position feature, requiring users to manually guide the robotic arm back via hand-gesture control after each grasp. Based on our instructor's suggestion, we added a dedicated reset-position command that automatically returns the arm to a predefined pose, which significantly reduces user operation time and improves workflow efficiency.

2.1.5 Mobile Platform

For the integration of the robotic arm with the wheelchair, multiple design options were evaluated to balance safety, flexibility, and usability.

- One option was directly mounting the robotic arm onto the wheelchair frame. While this approach minimizes additional structures, it compromises safety by introducing mechanical loads directly onto the wheelchair, potentially affecting stability.
- Another option was using a fixed mounting platform without sliding capability. Without the ability to reposition the base, the arm cannot effectively interact with objects located at various positions relative to the user.

The final design adopts a modular aluminum-profile platform with a sliding mechanism. This platform is constructed using standard 20x20 mm aluminum profiles for high structural strength and ease of assembly. Four wheels are installed at the bottom, providing mobility and enabling the system to be positioned alongside or detached from the wheelchair as needed.

Sliding blocks are mounted on the top aluminum frame. The robotic arm is fixed on the sliding blocks. This design significantly increases the effective reach of the robotic arm without the need for a larger or more complex arm structure.

2.2 Design Details

2.2.1 Customized End Effector

Form Factor Design

As shown in Figure 5, The final gripper adopts a parallel gripper structure driven by a linkage transmission mechanism. This design provides a wide and adjustable gripping range, accommodating objects of various sizes and shapes. Sponge padding is applied

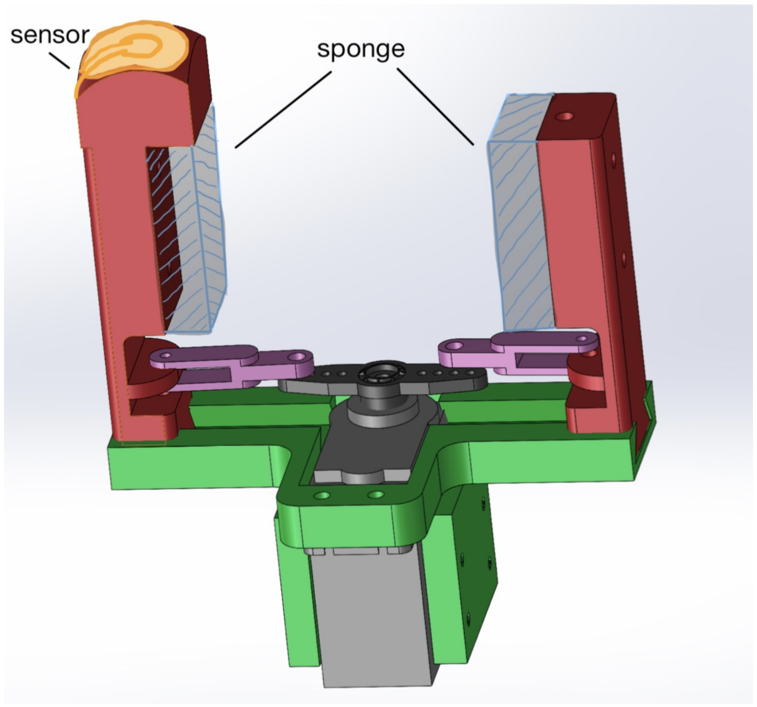

Figure 5: Final Gripper Design with Extended Curved Finger

to the inner surfaces of the gripper fingers to ensure uniform contact force distribution, increase friction, and improve grip stability, while also protecting fragile items from damage.

Gripper components are fabricated using nylon material through professional 3D printing. Nylon offers high tensile strength, toughness, and fatigue resistance, ensuring structural integrity and durability under repeated operational stresses.

To expand practical functionality, one gripper finger is extended and designed with a curved tip. This modification improves accessibility to recessed or hard-to-reach control panels and enhances precision when pressing small or closely spaced buttons, supporting assistive tasks such as elevator and door control operation.

End-effector Control

Force-to-Voltage Conversion (FSR Circuit)

The force-sensing resistor (FSR) operates as a variable resistor whose resistance R_{FSR} decreases non-linearly with applied force F. A voltage divider circuit (Figure 1) converts this resistance change into a measurable analog voltage V_{out}:

$$V_{out} = V_{in} \cdot \frac{R_{fixed}}{R_{FSR} + R_{fixed}} \tag{4}$$

Where:

- V_{in}: Supply voltage (5 V from Arduino)
- R_{fixed}: Fixed pull-down resistor
- R_{FSR}: FSR resistance

The circuit scales the FSR's non-linear response into a 0 - 3.3 V range voltage output.

Servo Motor Control Logic

The gripper actuation system implements position control through a servo motor with the Arduino Uno with `Servo` library.

Table 1: Robotic Arm Control Parameters

`Parameter`	**Value**	**Description**
`SERVO_PIN`	9	PWM output pin
`OPEN_ANGLE`	0°	Fully open position
`CLOSE_ANGLE`	120°	Fully closed position

Arduino Pin Connection Table

All the pin connections on the Arduino board are shown in Table 2. And Figure 6 is the corresponding schematic diagram.

Table 2: Pin Connection Table

Component	**Arduino Pin**	**Description**
Pressure Sensor	A0	Analog input for voltage measurement
Servo Motor	D9	PWM signal to control motor angle
LED (optional)	D13	Digital output for status display
GND	GND	Common ground for all components
VCC (Sensor)	5 V	Power supply for the pressure sensor

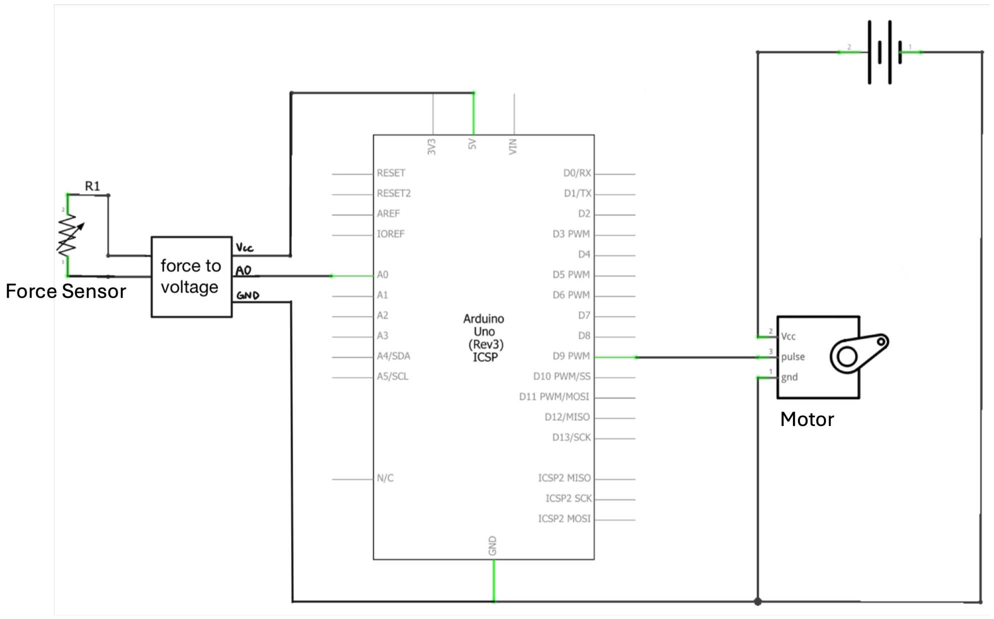

Figure 6: Pin Conncetion of the End-effector Control System

Unity Integration

On the Unity side, we use a C# script to:

- Open a serial port connection to the Arduino
- Read sensor data (voltage) from the Arduino
- Send string commands such as `"OPEN"` and `"CLOSE"` via serial

Key Unity Functions

- `SerialPort.Open()` — Opens communication.
- `serialPort.WriteLine("OPEN")` — Sends the open signal.
- `serialPort.ReadLine()` — Reads voltage data.

In Unity, we define two hand gestures to signal close and open. Upon detection of gesture, Unity sends commands via serial port to Arduino and the gripper responds accordingly.

2.2.2 OpenMANIPULATOR-P Robotic Arm

Robot Operation

A controller ROS program is provided to control each joint of OpenMANIPULATOR-P and check states of OpenMANIPULATOR-P through messages. Some key parameters for launching the controller are listed in Table 3:

Table 3: Robotic Arm Control Parameters

Parameter	Value	Description
use_robot_name	open_manipulator_p	• Defines the ROS namespace for message routing • Determines the manipulator's identifier
dynamixel_usb_port	/dev/ttyUSB*	• U2D2 board: /dev/ttyUSB* • OpenCR: /dev/ttyACM* • (* = port number, e.g. /dev/ttyUSB0)
dynamixel_baud_rate	1 000 000 (default)	• Default: 1 MBd (optimal) • Range: 9600–3 000 000 baud
control_period	0.010 s (default)	• Standard: 10 ms • Lower values increase CPU load
use_platform	true/false	• true: Physical hardware mode • false: Gazebo simulation
use_moveit	false	• Enable MoveIt! planning when true • Requires additional configuration
planning_group_name	arm	Defines kinematic chain for MoveIt! trajectories
moveit_sample_duration	0.050 s	Trajectory point interpolation interval

Topics & Services

The open_manipulator_p_controller has a list of published topics for checking the status of the robot regardless of the robot's motion. For us, the important point is to control the robot and several most widely used services are listed in Table 4.

Table 4: Robotic Arm Control Services

Service	**Msg Type**	**Description**
`/goal_js_path`	SetJointPosition	• FK: Direct joint control • Input: Target angles + time • Controls all 6 joints
`/goal_js_to_pose`	SetKinematicsPose	• IK solution • Input: EE pose + time • Auto-calculates joints
`/goal_ts_path`	SetKinematicsPose	• Cartesian control • Input: Target pose + time • Straight-line motion
`/goal_js_from_present`	SetJointPosition	• Relative FK • Input: Angle Δ + time • From current position
`/goal_ts_from_present`	SetKinematicsPose	• Relative Cartesian • Input: Pose Δ + time • Offset from current

ROS-Unity Integration

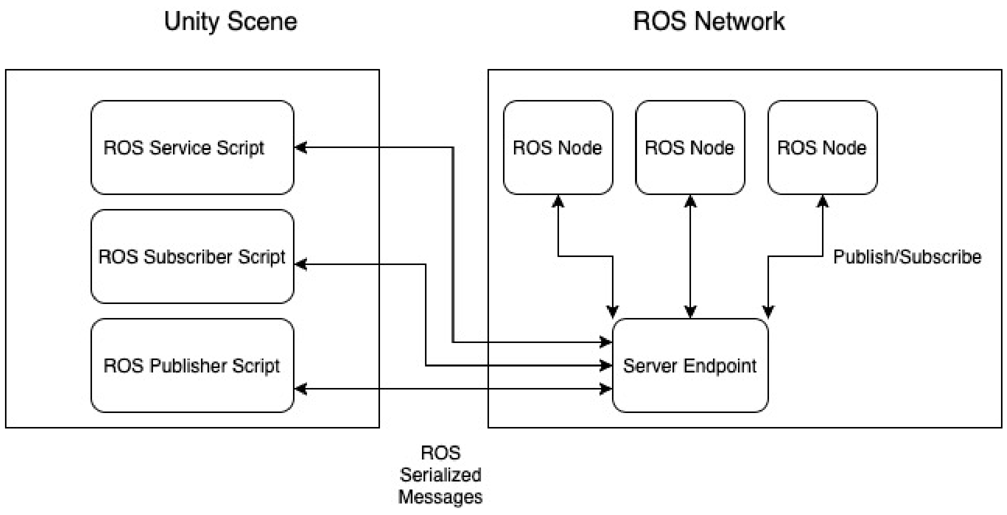

Figure 7: ROS-Unity Communication Structure[3]

The ROS–Unity communication bridge plays a critical role in our system because the pose information used to control the robotic arm is generated within the Unity environment. However, the services that execute these pose commands and control the physical robotic arm reside on the ROS side, running on an Ubuntu system. Therefore, real-time communication is necessary to ensure that Unity can request and receive robotic actions from the ROS backend.

The structure of this bridge is shown in Figure 7. Communication between ROS nodes follows the traditional ROS **Publish/Subscribe** model, allowing for decoupled and scalable message passing. Unity interacts with this system in real time, sending commands and receiving feedback, making it possible to simulate, visualize, and control physical robotic systems directly from a Unity-based application.

This architecture enables effective and synchronized bidirectional communication between Unity and ROS, essential for applications involving real-time robotic control, simulation, and user interaction through immersive interfaces like VisionPro. The communication between Unity and ROS is facilitated through a structured interface composed of the publisher, subscriber, and service scripts on the Unity side, and corresponding nodes and a server endpoint on the ROS side.

Within the **Unity Scene**, three types of scripts are responsible for exchanging data with the ROS system:

- **ROS Publisher Script:** Sends messages from Unity to ROS, such as robot control commands or object positions.
- **ROS Subscriber Script:** Receives messages from ROS, including sensor data and robot status, and makes them available to the Unity environment for visualization or feedback control.

- **ROS Service Script:** Initiates service requests (e.g., `SetKinematicsPoseRequest`) and processes responses from ROS, enabling more structured and request-response-type communication.

These scripts communicate with a centralized **Server Endpoint** in the **ROS Network**, which acts as a gateway, handling the serialization and deserialization of ROS messages. The server endpoint further interfaces with various **ROS Nodes**, each responsible for specific robotic tasks such as inverse kinematics, motion planning, or sensor integration.

2.2.3 Depth Camera

Real-time visual feedback from the robotic arm's end-effector significantly improves user perception and enhances precision during manipulation tasks. To achieve this functionality, we mounted a high-resolution camera on the robotic arm's end-effector and connected it directly to the control computer using a USB 3.0 cable, ensuring reliable and low-latency video transmission.

Within Unity, we leveraged its built-in Webcam functionality to seamlessly capture and integrate the live video stream from the end-effector camera. The video feed was projected onto the MR interface's canvas, allowing users wearing the Apple Vision Pro headset to observe real-time images directly within their Mixed Reality view.

This integrated visual feedback provided users with a precise understanding of the robotic arm's environment and task conditions. Consequently, it significantly enhanced the system's usability, accuracy, and the overall immersion experience during robotic manipulation tasks.

2.2.4 Mixed Reality Interface System

To enable intuitive human–robot interaction, we designed a Mixed Reality (MR) interface utilizing Unity along with Apple Vision Pro's XR hand-tracking capabilities.

- **Hand Skeleton Coordinate**
  In the Mixed Reality (MR) interface, accurately capturing the user's hand posture is crucial. We utilized Unity's XR Hand Subsystem plugin, integrated with the Apple Vision Pro headset, to achieve real-time hand skeleton tracking based on visual feedback from the MR device. This subsystem accurately identifies and tracks detailed skeletal joint positions and orientations of the user's hand. Throughout our development process, we specifically extracted and utilized the coordinates of critical anatomical landmarks, including the wrist joint and the tips of all five fingers: thumb, index, middle, ring, and little fingers. These skeletal joint coordinates serve as the fundamental input to our MR interface, enabling precise gesture recognition and subsequent robotic manipulation. In Figure 8, we demonstrate the real view of the recognized skeleton in the Apple Vision Pro.

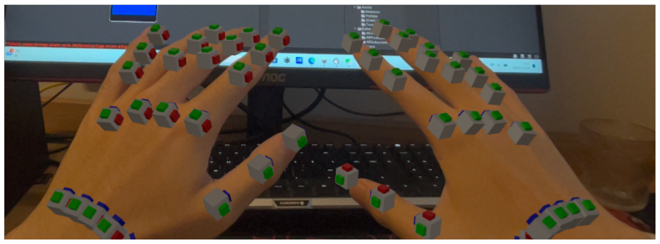

Figure 8: Hand Skeleton Tracking Visualization in VisionPro Interface

- **Gesture Recognition**
  In order to achieve intuitive control of the robotic system, our MR interface focuses on recognizing two primary hand gestures from the user: pinch and fist.
  - **Pinch Gesture Recognition**
    To accurately detect the pinch gesture, we extracted the positions of the thumb and index fingertips obtained from the XR Hand Subsystem. Using Unity's built-in function, `Vector3.Distance`, we calculated the real-time distance between these two fingertip coordinates. A pinch gesture is recognized only when this calculated distance falls below a predefined threshold value, termed `PinchDistanceThreshold`. This method ensures robust and responsive gesture detection, enabling precise interaction such as object selection or fine manipulation in our robotic control interface.
  - **Fist Gesture Recognition**
    The fist gesture is identified by examining the positions of all five fingertips relative to the wrist position. We computed the individual distances between each fingertip (thumb, index, middle, ring, and little fingers) and the wrist joint position using `Vector3.Distance`. The system recognizes the fist gesture only when all these fingertip-to-wrist distances are simultaneously below a predefined threshold. This approach ensures reliable fist gesture detection, suitable for commands such as grasping or executing a reset function within the robotic manipulation process.
- **Pose Mapping Between User's Hand and End-Effector**
  A critical challenge in developing the Mixed Reality interface was ensuring accurate pose mapping between the user's hand and the robotic end-effector, especially considering the differences between coordinate systems used in Unity and robotic control. We encapsulated this logic in a `HandEyeMapper` class with the following steps:
  - *Initialization*:

$$h_0 = \text{initialHandRot}, \quad r_0 = \text{robotInitialRot}, \quad h_0^{-1} = \left(h_0\right)^{-1}. \tag{5}$$

  - *Relative Hand Rotation:* Given the current hand quaternion h_{cur}, compute

$$h_{\text{rel}} = h_{\text{cur}} \otimes h_0^{-1}. \tag{6}$$

– *Target End-Effector Rotation:* Apply this relative rotation to the robot's base orientation:

$$r_{\text{target}} = h_{\text{rel}} \otimes r_0. \tag{7}$$

– *Normalization:* Finally normalize to avoid drift:

$$q_{\text{cmd}} = \frac{r_{\text{target}}}{\|r_{\text{target}}\|}. \tag{8}$$

In addition, to reconcile left-handed Unity coordinates with the robot's right-handed system, we apply a component swap:

$$\begin{cases} w_u = w_r, \\ x_u = y_r, \\ y_u = -z_r, \\ z_u = -x_r, \end{cases} \tag{9}$$

ensuring that both orientation and position mappings remain consistent across coordinate conventions. This transformation is shown in Figure 9.

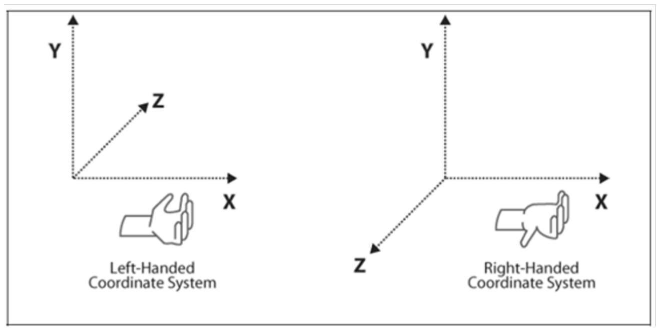

Figure 9: Left-Handed vs. Right-Handed Coordinate Systems

- **Arm Control Algorithm**
  To ensure smooth and responsive robot motions, we first apply a Kalman filter to the raw 3D hand-position measurements, reducing jitter and noise. The filtered position is then blended with the previous command via linear interpolation:

$$p_{\text{cmd}} = 0.6\, p_{\text{filtered}} + 0.4\, p_{\text{prev}}. \tag{10}$$

For orientation, we perform a spherical-linear interpolation (slerp) between the previous end-effector quaternion and the newly filtered quaternion with a weight of 0.3:

$$q_{\text{cmd}} = \mathrm{slerp}\big(q_{\text{prev}},\, q_{\text{filtered}},\, 0.3\big). \tag{11}$$

This combination of Kalman smoothing and weighted interpolation strikes a balance between stability and responsiveness, yielding smooth yet timely arm movements.

- **Gripper Control**
  To realize intuitive grasping and manipulation capabilities in our Mixed Reality interface, we established direct communication between Unity and the robotic gripper's control system through an Arduino microcontroller. Specifically, Unity was configured to communicate with the Arduino via serial port COM16.

  The control logic for the robotic gripper was implemented based on real-time detection of the user's left-hand gestures. When a pinch gesture was detected by the Unity MR interface (indicating a grasping intent), Unity transmitted the command `"CLOSE"` to the Arduino over the established serial connection, triggering the robotic gripper to close and securely grasp objects. In contrast, when the pinch gesture ended (the user released the grasp), Unity immediately sent the `"OPEN"` command, prompting the Arduino to open the gripper and release the object.

- **Reset Position Function**
  To enhance the usability and safety of the Mixed Reality robotic manipulation system, we implemented a dedicated reset function triggered by a specific user gesture. When the user's right hand forms a fist gesture - detected through our Unity-based gesture recognition system - the robotic arm automatically returns to a predefined ”reset” position.

  This functionality is particularly beneficial upon completion of a grasping task, as it swiftly moves the robotic arm back into a safe, easily reachable area for the user. As a result, users can conveniently retrieve objects held by the robot without needing complex manual repositioning or multiple gestures. By incorporating this intuitive reset capability, we improved overall efficiency, user-friendliness, and operational safety within our MR-based robotic manipulation system.

- **Safety**
  Ensuring safe operation of the robotic system within a Mixed Reality interface was a fundamental priority in our design. To achieve this, we implemented multiple comprehensive safety measures, specifically targeting potential risks associated with robotic movements and human-robot interactions:

  - **Workspace Constraints:** The robot's workspace was carefully restricted through software-defined boundaries, explicitly preventing the robotic arm's end-effector from entering any area occupied or reachable by the user. This ensured a clear separation between robot operations and user activity, significantly reducing collision risk.

  - **Joint Angle Movement Limits:** We imposed strict limits on the allowed change of robotic joint angles within each single command cycle. By constraining rapid joint movements, the risk of abrupt, unsafe motions was substantially mitigated.

- **Sequential Path Planning:** Motion planning was implemented incrementally. Only after the robotic arm closely approached its current motion planning goal would the system compute the subsequent motion trajectory. This approach reduced the accumulation of positional errors, thereby preventing large corrective movements and enhancing overall operational stability.

Together, these multi-layered safety measures created a robust operational framework, effectively minimizing risks and ensuring user safety during interactive robotic manipulation tasks in the Mixed Reality environment.

2.2.5 Mobile Platform

The final mounting platform is constructed using 20×20 mm aluminum profiles, forming a rigid rectangular frame with overall dimensions of 80 mm × 760 mm × 610 mm. Four M6 caster wheels are installed at the base, providing both mobility and stability as needed.

The horizontal sliding mechanism utilizes the T-slots of the aluminum profiles along the 760 mm length. Four small sliding blocks are inserted into the T-slots, and the robotic arm's base is fixed onto these blocks. The platform is connected to the wheelchair using clamps and L-shaped aluminum brackets, ensuring secure attachment while allowing for quick and convenient removal.

3 Verification

3.1 Verification of Robotic Arm Functions under MR Gesture Control

To verify the functionality of the robotic arm under Mixed Reality (MR) gesture control, we conducted a series of controlled experiments simulating typical user interactions. The goal was to ensure that all intended functions of the robotic arm could be successfully triggered and executed through MR hand gestures in real time.

Verification Scenarios

- **Basic Positioning:** Verified that the robotic arm can accurately move to target positions in 3D space based on the user's hand movement tracked via MR.
- **Grasping and Releasing:** Tested the gripper's response to pinch gestures for object grasping and automatic release upon pinch termination.
- **Orientation Adjustment:** Assessed the accuracy of end-effector orientation control through natural wrist rotations.
- **Reset Function:** Verified that performing a fist gesture reliably triggered the reset behavior, returning the robotic arm to its predefined safe position.
- **Safety Constraints:** Confirmed that workspace boundaries and joint movement limits effectively prevented unsafe motions during rapid hand movements.

Verification Procedure

- The user performed predefined gestures (pinch, fist, open hand) while wearing the MR headset.
- Corresponding commands were transmitted to the robotic system in real time.
- Robotic arm movements, gripper actions, and system safety mechanisms were monitored and logged for correctness.

Verification Results

- The robotic arm accurately followed hand trajectories with minimal delay and acceptable positional accuracy.
- The gripper reliably closed and opened upon detecting pinch and release gestures, successfully performing object manipulation.
- Orientation mapping remained stable, and no abrupt or unsafe movements were observed.
- The reset function consistently returned the arm to a safe position without manual intervention.

- Safety measures such as workspace constraints and joint limits were successfully enforced throughout the tests.

These results demonstrate that the robotic arm responds accurately and reliably to MR-based gesture controls, meeting the design requirements for intuitive and safe human-robot interaction.

3.2 Verification of Profile Box and Wheelchair Integration

To validate the effectiveness and stability of the aluminum profile box integrated with the wheelchair, we performed dedicated structural and functional tests, focusing on enhancing the robotic arm's operational reach and ensuring safe, stable interactions during real-world use.

Verification Scenarios

- **Reach Extension Test:** Verified that the sliding mechanism on the profile box allows the robotic arm base to smoothly move along the 80 cm rail, effectively extending the arm's reachable workspace without requiring the wheelchair to reposition.
- **Locking and Stability Test:** Assessed the ability to securely fix the robotic arm base at various positions along the rail to maintain stability during operation.
- **Structural Stability Test:** Tested the overall stability of the profile box under static and dynamic loading conditions while attached to the wheelchair, ensuring no significant vibration or displacement occurs during robotic arm movements.
- **Integration with Wheelchair Test:** Verified that the combined system allows for seamless mobility of the wheelchair without interfering with the profile box structure or the robotic arm's positioning.

Verification Procedure

- The robotic arm base was manually moved along the 80 cm sliding rail to various positions, and its maximum reachable workspace was recorded.
- The locking mechanism was engaged at multiple positions to test its effectiveness under the full load of the robotic arm, which weighs 5.76 kg.
- Static load tests were performed by operating the robotic arm to reach various angles and extensions while monitoring structural stability.
- Dynamic tests involved moving the wheelchair over different floor surfaces with the robotic arm positioned at various locations along the rail to assess stability during mobility.

Verification Results

- The sliding mechanism operated smoothly with low resistance, and the robotic arm's effective workspace was significantly extended when repositioned along the rail.
- The locking system reliably held the base in place during robotic arm operations, with no observable slippage, even when the arm was fully extended or positioned at challenging angles.
- The profile box exhibited strong structural integrity, remaining stable and vibration-free under the full 5.76 kg load during both static and dynamic tests.
- Wheelchair mobility was not hindered by the additional profile box structure, and the integrated system maintained excellent stability during normal movement and terrain changes.

These tests confirm that the profile box effectively expands the operational workspace of the robotic arm while maintaining structural safety and seamless integration with the wheelchair platform.

3.3 Verification of Pressure Sensor and Communication Functionality

To validate the correct functionality of the pressure sensor and ensure reliable communication between Unity and Arduino, we conducted focused experiments covering both sensor signal acquisition and two-way serial communication.

Verification Scenarios

- **Voltage Reading from Pressure Sensor:** Verified that analog pressure sensor readings are correctly captured by Arduino and transmitted to Unity for real-time display.
- **Unity-Arduino Serial Communication:** Verified reliable two-way serial communication for both sensor data transmission and gripper control commands.

Verification Procedure

- The pressure sensor was connected to the Arduino analog pin A0.
- Arduino continuously read analog voltage values, converted them to digital form, and transmitted the data to Unity over the serial port.
- Unity received and displayed the real-time voltage readings in the user interface.
- For command verification, Unity sent `"OPEN"` and `"CLOSE"` commands to Arduino based on user interactions or threshold triggers.
- Arduino responses were verified by both toggling onboard LEDs and controlling the robotic gripper accordingly.

Verification Results

- Unity successfully displayed real-time voltage values corresponding to the pressure applied on the sensor.
- Applying pressure resulted in clear and immediate changes in the displayed voltage readings.
- Unity successfully sent control commands to Arduino, and the Arduino reliably executed the `"OPEN"` and `"CLOSE"` commands.
- All communication functions were validated to be stable and responsive without noticeable delays or data loss.

These tests confirm that the pressure sensor reliably captures voltage data, and the communication between Unity and Arduino supports both real-time data transfer and precise gripper control commands, meeting the design requirements of the system.

4 Costs

Table 5: Labor Cost Analysis

Item Description	**Rate/Price (RMB)**	**Hours/Qty**	**Subtotal (RMB)**
Yinuo Yang	50 (per hour)	250	12,500
Yunyi Lin	50 (per hour)	250	12,500
Xingru Lu	50 (per hour)	250	12,500
Yilin Wang	50 (per hour)	250	12,500
Total Labor Cost			**50,000**

Table 6: Parts Cost Analysis

Item Description	**Rate/Price (RMB)**	**Hours/Qty**	**Subtotal (RMB)**
Wheel Chair	250	1	250
24 V, 15 A power supply	73	1	73
Thin Film Pressure Sensor	300	1	300
Single-Channel Module	300	1	300
Robotic Arm Bracket	300	1	300
End Effector	200	1	200
USB 3.2 Data Cable	30	1	30
Total Parts Cost			**1,453**

5 Schedule

Table 7: Project Schedule

Week	Task
3.15 – 3.21	Wrote a program to detect hand gestures and drag virtual objects in Unity. Installed ROS and the OpenManipulator package, and achieved basic control over the robotic arm.
3.22 – 3.28	Connected Unity with Isaac Sim using ROS2 and simulated a virtual environment in Isaac Sim. Established bi-directional communication between Unity and ROS (Ubuntu).
3.29 – 4.04	Successfully controlled the arm in Unity and used VisionPro to control the robotic arm.
4.05 – 4.11	Started designing the mounting system to attach the robotic arm to the wheelchair.
4.12 – 4.18	3D printed and installed a gripper that met the requirements. Installed a pressure sensor on the gripper.
4.19 – 4.25	Wrote code on the development board to control the opening and closing of the gripper and display pressure readings, while establishing a connection with Unity so that VisionPro's hand gesture recognition could control the gripper's operation.
4.26 – 5.02	Integrated the full control system, including VisionPro and pressure sensor feedback. Completed mounting with the wheelchair.
5.03 – 5.09	Improved the functionality of individual modules and conducted system-level testing for the designated tasks.
5.10 – 5.14	Made final design and code adjustments based on test results, and wrapped up the project.

6 Conclusions

6.1 Accomplishments

We successfully completed a workflow that enables wheelchair users to perform simple tasks such as grabbing lightweight objects and pressing buttons. The camera provides views from different angles through the VisionPro, while the VisionPro's built-in camera captures the user's hand gestures. Based on the detected gesture types, the robotic arm can move in real time to reach the correct end-effector pose, and the gripper opens and closes accordingly to complete the grasping task.

6.2 Uncertainties

- ROBOTIS has stopped support of OpenMANIPULATOR-P, this could risk our system's long-term maintainability.
- The robotic arm may exhibit unstable behavior when encountering kinematic singularity configurations.

6.3 Future Work

- **Integration of Electrically Actuated Sliding Rail Controlled via Unity Hand Gestures**
  To further enhance the system's usability and intuitiveness, the current manual sliding rail could be upgraded to an electrically actuated version. This would enable seamless control of the manipulator's horizontal movement directly through hand gestures recognized by the Unity interface, providing a more natural and accessible user experience, especially for users with limited mobility.
- **Development of a More Versatile Soft Gripper**
  The existing rigid parallel gripper could be replaced with a soft robotic gripper to improve adaptability when handling objects of various shapes and materials. With this upgrade, the manipulator could be capable of safely grasping a wider range of irregular and fragile objects, expanding its functional applications in everyday tasks and enhancing user autonomy.

6.4 Safety

Robotic Arm Operation Safety: Our system includes a robotic arm extending from the rear of the wheelchair, which introduces potential risks if not properly designed. To avoid these risks, we implement the following safeguards:

Hardware/Software Safety: The arm will remain folded when inactive, ensuring it does not occupy additional space beyond the wheelchair and cause potential collision.

Speed Constraints: Arm motion speed will be limited to prevent high-impact collisions.

Safe Operation Limits: The Open Manipulator-P arm will be programmed to operate within predefined safety thresholds for users and bystanders. Specifically, it will avoid the space that the user occupies. Furthermore, Apple Vision Pro's depth and spatial awareness capabilities will be utilized to enhance situational awareness and prevent unintended interactions.

Privacy and Data Protection: User privacy is a critical consideration in our system, particularly given the use of real-time cameras and Mixed Reality (MR) technology. Our system does not store or transmit user data to any external servers. All video processing and interaction tracking occur locally. The rear-facing camera feed is processed in real time solely for user awareness and robotic arm control. Similarly, Apple Vision Pro's hand-tracking data is processed locally, without transmitting biometric or movement data beyond the device[4].

6.5 Ethics

6.5.1 Privacy and Data Protection

Aligning with IEEE/ACM principles, our system prioritizes user privacy:

Local Processing: No user data (camera feeds, hand-tracking biometrics) is stored or transmitted externally.

Real-Time Use Only: Rear-facing camera data is processed locally solely for arm control and user awareness [3].

6.5.2 User Autonomy and Accessibility

Inclusive Design: The MR interface offers intuitive controls, respecting human dignity[5], [6].

Safety in Design: The MR interface ensures frontal visibility is never obstructed during use.

Transparency: Users will be informed of system capabilities/limitations to manage expectations.

References

[1] ROBOTIS. “OpenManipulator-P-Specification,” ROBOTIS e-Manual. (2020), [Online]. Available: https://emanual.robotis.com/docs/en/platform/openmanipulator_p/specification (Accessed Apr. 14, 2025).
[2] ROBOTIS. “ROS Packages for OpenMANIPULATOR-P.” (2020), [Online]. Available: https://github.com/ROBOTIS-GIT/open_manipulator_p/tree (Accessed Apr. 22, 2025).
[3] U. Technologies. “ROS-Unity Integration Tutorial.” (2023), [Online]. Available: https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/ros_unity_integration (Accessed Apr. 22, 2025).
[4] Apple. “Apple Vision Pro Privacy.” (2023), [Online]. Available: https://www.apple.com/privacy/ (Accessed Mar. 22, 2025).
[5] IEEE. “IEEE Code of Ethics.” (2020), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (Accessed Apr. 14, 2025).
[6] ACM. “ACM Code of Ethics and Professional Conduct.” (2018), [Online]. Available: https://www.acm.org/code-of-ethics (Accessed Apr. 14, 2025).