ECE445: Senior Design Laboratory

Project Proposal

Human-Robot Interaction for Object Grasping

with Virtual Reality and Robotic Arms

Team #

Jiayu Zhou, jiayu9

Ziming Yan, zimingy3

Yuchen Yang, yucheny8

Jingxing Hu, hu80

Professor: Gaoang Wang, Liangjing Yang

TA: Tielong Cai, Tianci Tang

March 16, 2025

1. Introduction

1.1 Objective and Background

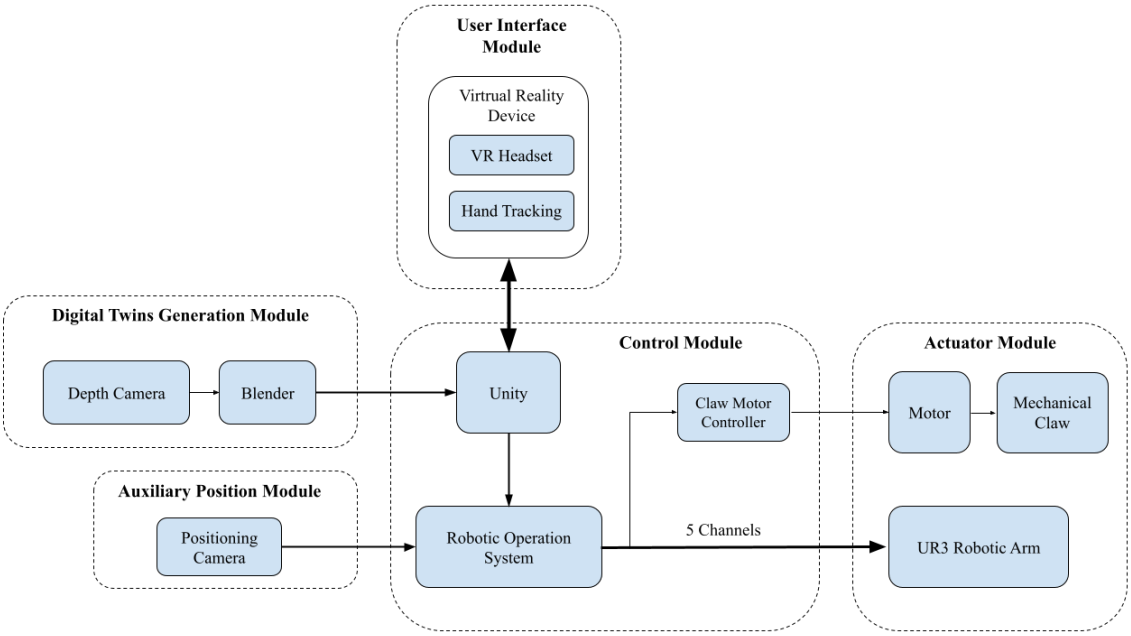
Current robotic systems lack intuitive and seamless human-robot interaction for object manipulation. Traditional teleoperation methods often require complex controllers, making it difficult for users to interact naturally. With advancements in Virtual Reality (VR) and robotic systems, it is possible to develop an intuitive interface where users can manipulate objects in a virtual space, and a robotic arm replicates these actions in real-time. This project aims to bridge the gap between human intention and robotic execution by integrating VR with robotic grasping, enabling precise and efficient remote object manipulation. Such design opens up to a wide potential market catering to various customer needs, such as remote working with precise control of streamline, dining requests of the Parkinson's, etc.

1.2 High-Level Requirements List

- Successfully generate and import at least 10 digital twin objects into Virtual Reality.
- The system should accurately map hand trajectory to robotic arm movements in real-time.
- The robotic arm should replicate the grasping motion within 2 minutes of user interaction.

2. Design

2.1 Block Diagram



2.2 Digital Twin Creation Subsystem

The digital twin creation subsystem utilizes depth camera data for image acquisition and subsequent point cloud processing.

2.2.1 Depth Image Acquisition & Point Cloud Generation

Acquisition Hardware:

A depth camera (e.g., Intel RealSense or Microsoft Kinect) is deployed to capture both RGB and depth data simultaneously.

The depth camera generates dense point cloud images, accurately representing the spatial geometry of the environment.

Point Cloud Generation:

Raw depth data is processed in real-time to produce detailed point clouds,

serving as the primary data source for 3D reconstruction.

The system ensures that the spatial resolution is sufficient for distinguishing between closely placed objects.

Requirements:

Requirement 1: The depth camera must deliver point clouds with a spatial resolution accurate to within ±0.5 mm.

2.2.2 Point Cloud Segmentation & Tabletop Recognition

Tabletop Detection:

An initial segmentation algorithm analyzes the point cloud to detect the tabletop by fitting a planar model (using techniques like RANSAC) to identify a consistent horizontal surface.

The detected plane provides the reference height for subsequent object segmentation.

Object Segmentation:

Following tabletop identification, objects are segmented based on the criterion that their boundary points extend above the detected table surface. This rule effectively differentiates individual objects from the continuous plane of the tabletop.

Requirements:

Requirement 3: The tabletop detection algorithm must reliably identify the table surface with an error margin below 2 cm.

Requirement 4: Object segmentation must accurately distinguish separate items, achieving a segmentation accuracy of at least 90% even when objects are in close proximity.

2.2.3 3D Reconstruction & Unity Integration

3D Model Reconstruction:

The segmented point clouds are converted into mesh representations using algorithms such as Poisson surface reconstruction.

Subsequent optimization with tools like Blender or Meshlab reduces polygon count by up to 70% while maintaining essential visual and structural details.

Integration into Unity:

Optimized digital twin models are imported into Unity, where they are aligned spatially using the Mixed Reality Toolkit (MRTK).

Unity's physics engine is leveraged to assign appropriate physical properties to the models, ensuring dynamic interaction fidelity.

Requirements:

Requirement 5: Reconstructed models must be fully compatible with Unity

and support real-time rendering at a minimum of 60 fps.

2.3 Robotic Arm Execution

This subsystem translates user interactions into real-world robotic grasping by integrating three core modules: the **UR3 robotic arm, the robotic gripper, and the depth camera**. These components work together to enable accurate, real-time grasping through motion planning, visual perception, and controlled actuation.

The system receives user interaction data, processes it through motion planning algorithms, and executes robotic grasping actions. The **UR3 robotic arm** provides the primary actuation, while the **robotic gripper** ensures a stable grip on objects. The **depth camera** supplies crucial spatial data for object localization and trajectory optimization.

2.3.1 Robotic Arm

- **Function**: Executes planned motion trajectories to approach and manipulate objects.
- **Connection**: Receives trajectory commands from **ROS MoveIt**, integrating with the depth camera for dynamic adjustments.
- **Role in System**: Acts as the primary actuator, responsible for positioning the gripper to execute precise grasping tasks.

2.3.2 Robotic Gripper

- **Function**: Clamps and releases objects based on control signals, ensuring a stable grasp.
- **Connection**: Directly mounted on the **UR3 robotic arm**, following grasping commands from the system.
- **Role in System**: Ensures secure object manipulation, preventing slip or excessive force application.

2.3.3 Depth Camera

- **Function**: Captures **3D spatial information** to improve grasping accuracy through object localization.
- **Connection**: Interfaces with **ROS perception modules** to generate real-time depth data for motion planning.
- **Role in System**: Provides essential visual feedback, allowing adaptive adjustments to grasping trajectories.

2.3.4 Communication Module

- **Function**: Facilitates real-time data transfer between **HoloLens and ROS** via **WebSocket or ROS Bridge**.
- **Connection**: Ensures that user commands from **HoloLens** are translated into executable robotic actions.
- **Role in System**: Serves as the **interface** between the user and the robotic

system, enabling intuitive operation.

2.3.5 Grasping Algorithm Modue

- **Function**: Optimizes grasping execution by ensuring stability and preventing object damage.
- **Connection**: Uses force feedback and environmental constraints to refine grasping strategies.
- **Role in System**: Enhances the **success rate of grasping operations** by dynamically adjusting grip force and trajectory.

2.4 Meta Quest App

The Meta Quest (or Oculus Quest VR headset) App block enables Meta Quest users to interact with the digital twin scene. In the runtime, it receives user hand trajectory from real world input. Then it processes the information for object recognition and finally sends the object index as well as the hand trajectory to the unity digital twin.

2.4.1 Scene

The predefined scene serves as a VR counterpart to our Digital Twin in Unity. It incorporates physical models of the following elements: 10 objects, table and robotic arm.

Requirements: The elements in the scene must have less than 5% precision tolerance in scales. The 10 objects must be viewed as separable entities from

the table. The robotic arm could be simplified as a rigid body, but the end-point gripper must manifest the same level of detail with other elements.

2.4.2 Hands Tracking

This module captures user hands trajectories and sends them to the Digital Twin in Unity via proprietary API databus1 and object recognition module in real time. This is the key element to enable human robot interaction and is implemented by Unity First-Hand Dependencies.

Requirements: Must capture hand trajectory every 0.2 seconds. Must convert the trajectory data to unity-readable format.

2.4.3 Object Recognition

This module predicts which object the user is targeting based on hand trajectories and records the object is grasped. It constantly receives data from Hands Tracking module and reads the real-time object locations in the scene as input, and after processing it sends the “predicted_object” data packet to the Digital Twin Unity via proprietary API databus 2. The prediction task is a classification problem on the 10 objects with input of a string of hand trajectory in the past 10 seconds. The code is written into Meta Quest App Build.

Requirement: The classification task should be performed in a limited time, e.g. 0.2 seconds. The prediction result must be accurate as the user hand

approaches close (within 10 cm) to the target object.

2.5 Risk Analysis

The component at highest stake is the Unity control unit. It not only recalculates the calibration of the two different frames of VR and reality but should also perform object recognition as a classification task in very short delay. The component also constantly renders the scene and the digital twin. Such parallel computing places the response time at risk and therefore adds difficulty to our high-level requirement 3.

3. Ethics and Safety

3.1 Ethics

We should constrain the use of our VR + Robotic Arm design to beneficial but not destructive use. (Ethics Code 1)

We will improve the community's understanding of VR-Robotic Arm Interaction by writing technical reports with substantial details. (Ethics Code 2)

We should be honest with all kinds of technical references. (Ethics Code 5)

We should treat all people alike and not let our target users be restricted to certain race or gender. (Ethics Code 7)

We should avoid injuries caused by any unexpected activities of our design especially by the robotic arm. (Ethics Code 9)

3.2 Safety

Electrical Safety: Ensure our voltage is within the safety range.

Mechanical Safety: Set halt conditions for the robotic arm. Unit test and simulate every module before finally running in the lab. End users should stay away far from the robotic arm.

Lab Safety: Always experiment under the supervision of a TA or instructor.

References

Universal Robots, "UR3 Robot User Manual," Version 3.4, May 2016. [Online]. Available: https://www.universal-robots.com/media/240787/ur3_us.pdf. [Accessed: 17-Mar-2025].