SAPIENZA
UNIVERSITÀ DI ROMA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, AUTOMATICA E GESTIONALE

R3: Motion Retargeting Methods Comparison

MEDICAL ROBOTICS COURSE

Professors:
Vendittelli Marilena,
De Santis Emanuele

Students:
Alemanno Giulia, Coletta Emanuele,
Lazzaroni Lisa, Negrenti Alessandra

Academic Year 2024/2025

Contents

1 Introduction

From the late 1900s onward, medical robots and virtual simulators have played an increasingly important role in medical practice. Indeed, these devices represent a significant advancement for rehabilitation, medical assistance, and clinical routines. The objectives of this development are to improve the precision of procedures, reduce risks for patients, and enable continuous training for medical professionals. In this context, the task of grasping is a very important aspect for reproducing realistically the movement of the virtual hand and its interaction with surrounding objects.

Grasping refers to the act of gripping, holding, and manipulating an object using the hand. In the field of robotic medicine, this task presents difficulties as it requires fine motor control and accurate sensory feedback. For this reason, the precise reproduction of human hand movements and the ability to provide haptic stimuli are fundamental goals [1].

Haptic interfaces provide tactile stimuli, such as vibrations, temperature variations, or object textures. Complementing these devices, virtual reality offers the capability to construct immersive digital environments for medical professionals to engage with .

1.1 Applications in the Medical Field

The integration of Virtual Reality (VR) and haptic feedback within medical robotics and simulation has significantly broadened training opportunities beyond traditional surgical procedures. These systems provide a controlled, low-risk environment where healthcare professionals can develop procedural competence and motor coordination. For instance, VR/haptic simulators have been specifically developed and evaluated for training in blood-related procedures such as arterial blood gas analysis (ABGA), blood culture, and transfusion. The use of haptic devices in these simulations provides real-time tactile feedback (e.g., during needle insertion), which is essential for realism and the acquisition of psychomotor and precision skills [2].

Furthermore, VR technology has historically been employed for the training and evaluation of surgical skills like suturing, enabling the objective measurement of performance parameters such as tissue damage, task completion time, and accuracy. This objective quantification, often coupled with force feedback devices, has been shown to differentiate performance levels between expert surgeons and novice students, and to induce performance improvement with practice. [3]

While VR has become common practice in training for a variety of surgical procedures, haptic feedback and interaction with the environment are usually attained by using ad-hoc instrumentation for the procedure at hand. In this project, we are interested in developing a realistic simulation of the grasping task in a virtual reality environment, using both haptic interfaces and computer vision-based hand tracking. The haptic interface, specifically WeArt's TouchDriver G1, allowed to obtain data related to

human hand movement to animate the virtual hand through inverse kinematics; the VR device, specifically Meta Quest, enabled real time tracking for the hand pose in the virtual space.

The applications of the project go beyond just simple grasping, the combination of hand tracking and grasping systems have been proven be a new front in rehabilitation of post-stroke patients, and, in general, for upper-limb rehabilitation. For the former in particular, loss of hand-motor control is the most debilitating long-term effect and rehabilitation consists of intensive and repeated practice of movement to induce re-learning of the movement and neuroplasticity. In this context, virtual reality systems have shown promising results compared to conventional therapy [4] [5] [6] [7].

2 Materials and Methods

For the development of the motion retargeting system for grasping tasks, a combination of hardware and software tools is employed to capture, process, and visualize human hand movements in a virtual environment. The experimental platform consists of a haptic glove for motion capture and tactile feedback, a virtual reality headset for immersive visualization, and a host computer responsible for data acquisition, processing, and rendering.

2.1 Hardware

- **WeArt TouchDIVER G1 Haptic Glove**

  The WeArt glove is equipped with three motion-sensing clips attached to the thumb, index, and middle fingers. Movements of the ring and little fingers are coupled to those of the middle finger. Each clip measures the closure angle of its respective finger and the thumb is additionally tracked for abduction. Sensor values are normalized between 0 (fully extended) and 1 (completely folded). The glove also delivers haptic feedback in the form of cutaneous force and thermal stimulation, enabling bidirectional interaction with virtual objects.

- **Meta Quest 3 Headset and Controllers**

  The headset provides immersive first-person visualization of the virtual environment. Two distinct configurations are considered:

  1. Integration with the haptic glove, where spatial tracking of the hand is achieved by rigidly attaching one Oculus controller to the WeArt glove and using its inertial measurement unit (IMU).
  2. Vision-based hand tracking, in which the headset's integrated cameras are used as a stand-alone motion capture solution for grasping tasks, without reliance on the glove.

- **Host Computer**

  The host workstation executes all computation for simulation and rendering. The workstation has the following hardware specifications:

  - CPU: Intel Core 7 165H
  - RAM: 32GB DDR5
  - GPU (integrated): Intel Arc Pro Graphics
  - GPU (dedicated): NVIDIA RTX 4070 Laptop

2.2 Software

- **Unity Engine (v2022.3.62f1)**

  The Unity game engine serves as the development platform, providing the simulation environment and rendering of the virtual hand and surrounding objects, as well as a physics engine which supports rigid bodies dynamics and collisions.

- **WeArt Unity SDK (v2.0.0)**

  The WeART Software Development Kit (SDK) for Unity enables direct access to glove sensor data from the game engine, including closure and abduction parameters for each finger, and provides the interface for haptic feedback control. It also includes the base 3D hand model used for animation, and a playground virtual scene for the user to test the glove's capabilities.

- **Oculus/Meta SDK (v78.0.0)**

  This package allows for the integration of the Meta Quest headset and controllers within Unity, supporting both controller-based tracking and vision-based hand tracking.

2.3 Functional Overview

This work is divided in two main parts: in Section 3 we explore hand tracking using the WeART TouchDiver haptic interface. Our objective is to replace the hand animation of the WeART SDK with an animation computed by modeling each finger as a 3R planar robot manipulator and simulating it at runtime. Smooth finger movement is obtained by tracking a predefined path. In section 4 we explore the use of our solution in a VR environment and try to approach grasping tasks with our modifications.

In Section 5 we explore the same concepts, leveraging the hand tracking capabilities of the Meta Quest 3 VR headset via its integrated cameras, and we approach grasping tasks utilizing this tracking. In Section 6 we compare the two methods in a variety of tests. Conclusions are drawn in Section 7.

3 Hand Motion Acquisition and Retargeting

The SDK developed by WeArt provides a dedicated prefab that incorporates a complete virtual hand within the Unity environment, together with the control scripts required for its operation. Within the Unity Hierarchy, the prefab is comprised of two primary structures, corresponding to the right and left hands, arranged in a mirrored configuration.

The hand model itself is structured into four distinct sections, each encompassing specific fingers and the associated portion of the palm. The first section includes the thumb and index finger; the second, the middle finger; the third, the ring finger; and the fourth, the little finger. Each finger is further subdivided into three segments, reproducing the anatomical phalanges and thereby enabling more precise articulation of movement. A notable exception is the thumb: although anatomically composed of only two phalanges, in the WeArt model it is represented by three segments, the additional one accounting for the osseous structure connecting the thumb to the wrist. This modeling decision ensures structural consistency with the other fingers while simultaneously providing enhanced flexibility in the simulation of motion.

The WeART SDK animates the hand by using the closure and abduction parameters obtained by the motion-sensing clips to interpolate between three fixed animation states: hand open, fist closed, and hand open with abducted thumb.

Our objective is to allow a smooth rendering of the animation, by instead modeling each finger as a 3R planar manipulator, where each joint corresponds to the natural articulation joint of the hand, and to obtain a realistic animation via path tracking in the cartesian space. The three links of the manipulator represent the three phalanxes of the finger, and each joint represents the relative articulation.

For a natural animation of the hand, it was also necessary to model the anatomical limitations of the articulations as joint saturation limits. We constrained each joint to only assume values $q_i \in [0, \frac{\pi}{2}], i = 1, 2, 3$.

The two limit configurations, $\mathbf{q} = \mathbf{0}$ and $\mathbf{q} = \begin{bmatrix} \frac{\pi}{2} & \frac{\pi}{2} & \frac{\pi}{2} \end{bmatrix}^T$ correspond, respectively, to the fingers being completely extended (along the same plane of the palm of the hand) and completely folded (in a fist configuration) as seen in Figure 1.

The length of each link, as well as the initial rotation and other useful parameters are extracted at runtime from the hand model present in the prefab.

For the implementation of this part of the project the *WeArtHandController.cs* script was modified to fetch the necessary data directly from the hand model and later expanded to support modeling and simulation of the manipulators.

3.1 Deriving information from the hand model

Retrieving the closure and abduction data given by the WeArt SDK was a necessary step in order to make it available for the simulation of the manipulators associated to

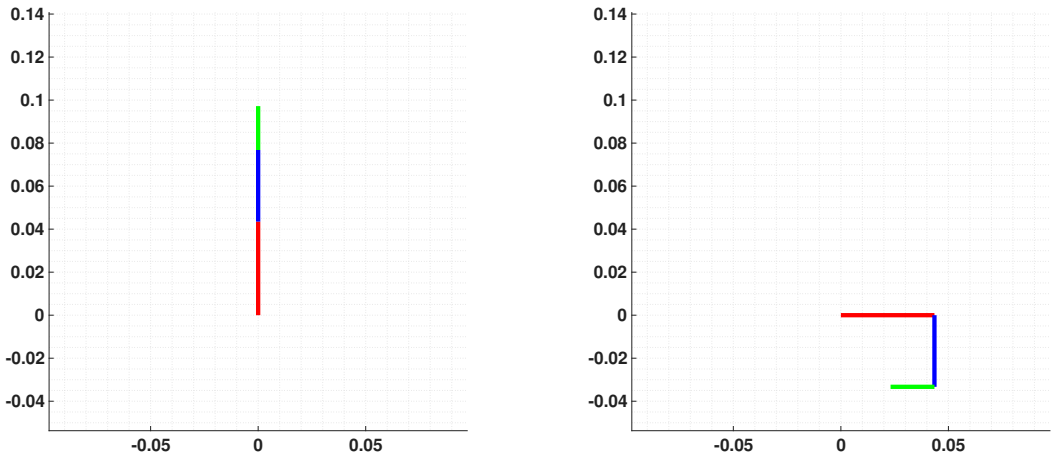


Figure 1: MATLAB plot of the 3R planar manipulator in the (a) fully extended and (b) closed limit configurations.

each finger.

This part is divided into three main steps:

- **Initialization of the sensor's references**: Inside the *Awake()* function the references to the WeArtThimbleTrackingObject components for the thumb, pointer finger, middle finger, ring finger, and pinky were instantiated, adding them to the struct *_thimbles*. Simultaneously the haptic actuators of the class WeArtHapticObject were abilitated to guarantee compatibility with the actuation provided by the glove. The transforms of each thimble are also saved by walking down the hierarchy of the model.

- **Management of the mapping**: It was then necessary to handle the device's mapping to the fingers: this was implemented through an index constructed in accordance to the hardware's characteristics, where the middle, ring and pinky finger all map to the middle finger. This allows the retrieval of the appropriate closure value for each finger.

- **Retrieval of the signals during runtime**: Inside the *FixedUpdate()* function the closure value for each finger is registered and additionally, the abduction value for the thumb is applied as a rotation on the Z-axis of the first joint. These signals belong to the chain of kinematic control and constitute the instantaneous input received from the hardware.

The length of the first two links of each finger is computed by taking the difference between initial positions of the respective thimbles. The length of the last link is computed against the *ExplorationOrigin* point, which corresponds to a collider used for grasping tasks, with origin exactly at the fingertip. Initial rotations are also saved for later use.

3.2 Computing the direct and inverse kinematics

Each finger is modeled as a 3R planar manipulator, moving in the XY-plane, as shown in Figure 2. Thus, each finger has direct kinematics.

$$\begin{cases} x = l_1 \cos q_1 + l_2 \cos(q_1 + q_2) + l_3 \cos(q_1 + q_2 + q_3) \\ y = l_1 \sin q_1 + l_2 \sin(q_1 + q_2) + l_3 \sin(q_1 + q_2 + q_3) \\ \alpha = q_1 + q_2 + q_3 \end{cases} \tag{1}$$

where l_i are the link lengths, as derived in section 3.1.

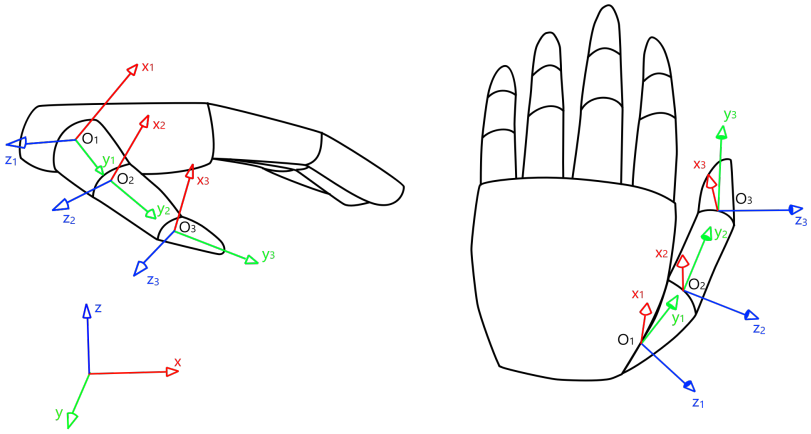


Figure 2: Reference frame associated to each joint in the hand model.

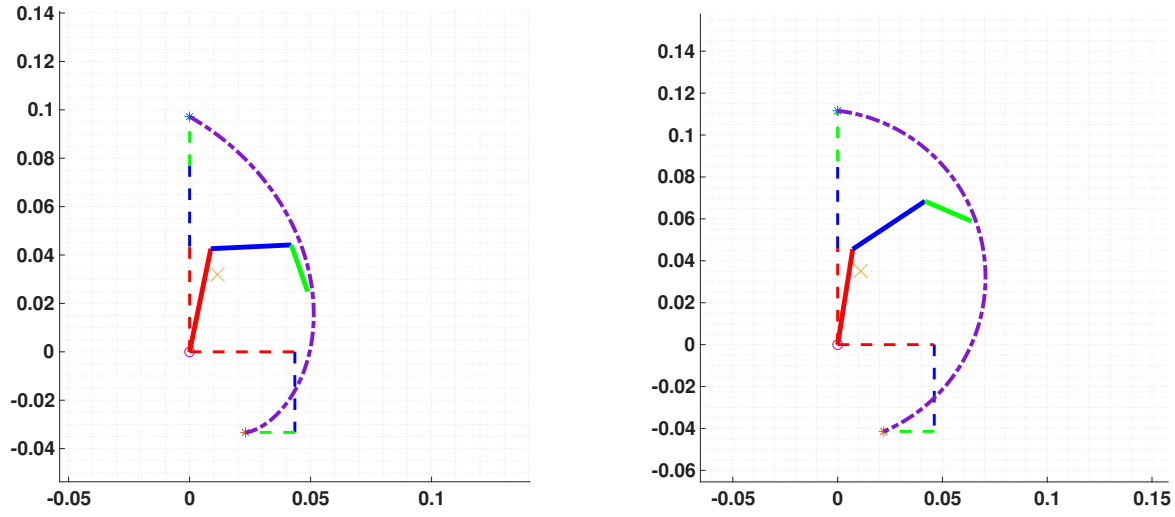
The control approach used is that of path tracking in cartesian space, via differential inverse kinematics. The controller features a term proportional to the error in cartesian space and, most importantly, a Jacobian null-space term to keep the joints within the $[0, \frac{\pi}{2}]$ limit.

$$\dot{q} = J^{-1}\big[K_p(p_d - f(q))\big] + (I - J^{\#}J)\dot{q}_o \tag{2}$$

where $K_p > 0$, p_d is the desired position, and $f(q)$ is the cartesian position of the end effector corresponding to current configuration q.

The control and the dynamics of the manipulators are computed during Unity's *FixedUpdate()* function, which happens at regular intervals. At each update, the control is computed and the evolution and the robot joint values are found by numerical integration of the system dynamics. We found that simple backwards Euler integration resulted in choppy and visually unpleasant movement, thus the robot dynamics is integrated using a second order Runge-Kutta method, provided by the Math.NET Numerics library.

However, the null space term alone does not ensure that the joints remain within the limits. In particular, we noticed that very sudden movements of the fingers result in impulsive jumps of the closure parameter from 0 to 1. When no saturation limits



(a) (b)

Figure 3: MATLAB plot of the 3R planar manipulator following (a) a cardiod path (index finger) and (b) an arc of circle path (thumb).

are considered on the manipulators (neither in position nor in velocity), this results in the joint values quickly reaching very high values outside the natural limit. Visually, this results in the fingers flexing in an unnatural way, though still correctly tracking the path.

To solve this issue, the joint values are hard-constrained in $[0, \pi/2]$ after each simulation step and before rendering.

The simulation of the manipulator system dynamics is completely abstracted from the actual hand behaviour. During rendering the pose of base joint of each manipulator is transformed to have a one to one correspondence to the knuckle of the respective finger, using the data obtained in Section 3.1.

3.3 Path configuration and experimentations

It was then necessary to find a path connecting the extended and folded position of the fingers, parametrized by the closure parameter. Two different path definitions were explored:

- **Arc of circle of set radius**: This mimics the already configured WeArt's animation. The radius is set empirically for a pleasant visual behavior. It has the drawback that each finger responds differently to different radii, with smaller radii creating a more pronounced arc between the two limit positions. Thus the radius had to be calibrated manually for each finger. This was implemented through the assist of some MATLAB simulations.

- **Cardiod curve**: This specific path configuration resulted in a more natural movement, especially when the finger is folded. The main advantage over the

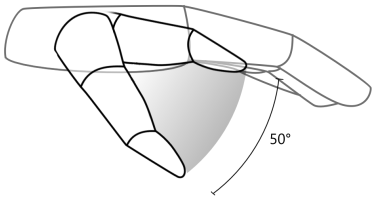


Figure 4: Degree of motion of thumb through abduction.

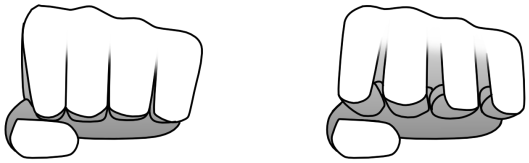


Figure 5: Fist position, with and without splaying correction.

arc of circle lies in the definition of a cardiod curve: given the orientation one and only one cardiod will be defined through the two selected points, so there is no need of manually calibrating a radius for each finger.

The cardiod delivered the best visual results for all the fingers except for the thumb. In the final product, fingers from the index to the pinky trace a cardiod path, while the thumb is the only one tracking an arc of circle. Figure 3 shows the index and the thumb following the two paths. Notice in particular how the curve of the cardioid (Figure 3a) is more accentuated near the closed configuration.

3.3.1 Abduction

The thumb is modeled as a 3R planar manipulator just like the other fingers. The abduction parameter is then mapped to a further rotation about the Z-axis ranging from 0 degrees (abduction=0) to 50 degrees (abduction=1). This rotation happens only at render time, and it is not considered in the simulation of the manipulators.

3.3.2 Splaying

Within the WeART hand model, there is significant space between the fingers even with the fist completely closed. For a more pleasant visualization, we added a splaying movement to the fingers as they close, by adding a slight rotation on the Z-axis for each finger, excluding the thumb. The direction and the maximum angle of this rotation varies per finger and has been calibrated empirically. This rotation only happens at render time, and it is not considered in the simulation of the manipulators.

4 Integration in Virtual Reality

The next phase of the project involved using the haptic glove in a VR environment, using the headset and controllers for global position and orientation in space, with local data on finger closure and abduction obtained from the TouchDiver device.

4.1 Unity configuration

For the implementation, Unity was configured to support virtual reality by installing and activating the *XR Plugin Management* package, enabling the specific plugin for the Meta Quest 3 device. Subsequently, the *XR Interaction Toolkit* was imported, which provides tools and prefabs specifically made for managing interactions in immersive environments. An *XR Origin* object was created within the scene, responsible for managing the camera and controllers, which constitutes the spatial reference for the headset and allows the position of the virtual hand to be synchronized with the real hand detected by the device's controllers.

4.2 Information provided by Meta Quest 3

The tracking provided by the headset was used to determine the position and orientation of the hand in three-dimensional space, while the TouchDiver data continued to govern the dynamics of finger closure. The integration of these two informations made it possible to develop a more complete system: on the one hand, the headset ensures accurate and consistent hand positioning; on the other, the haptic interface provides kinematic parameters on finger flexion and abduction, mapped onto joint angles using the inverse kinematics functions developed in the initial phase of the project.

Once Unity was configured, the grasping of rigid virtual objects present in the scene was implemented.
During grasping, finger closure is regulated by TouchDiver, while global positioning and hand rotations are governed by headset tracking. The grasping mechanism used is the one provided by WeArt in the SDK.

4.3 Grasping system

Within the WeArt SDK package for Unity, the grasping mechanism is achieved through the cooperation of two components: *WeArtHandGraspingSystem* and *WeArtHandController*. Grasp detection is based on a network of colliders associated with the fingers and palm. Each finger has a *haptic collider*, linked to the physical contact area of the haptic thimbles, which represents the actual point of contact, and a *proximity collider*, which extends the detection area by anticipating the possibility of interaction and therefore has a predictive function with respect to possible imminent interactions. Collision events provided by the Unity engine update internal data structures that collect, for

each finger, the objects currently in contact or in proximity. This mechanism allows the system to maintain a dynamic mapping of the state of interaction between the hand and the environment.

As mentioned in Section 3, the unmodified WeArt SDK animates the hand by using the Unity animation system and interpolating between three states (hand open, closed, open with abducted thumb) based on animation *weights*. The weights are also filtered depending on a variety of conditions, which stop or slow down the animation of the hand when it is in contact (or is approaching contact) with other rigid body objects. The original *WeArtGraspingSystem* tracks two different hand models: a ghost hand that uses the closure values directly and a visible animated hand that uses filtered animation weights. The latter is used to prevent visual hand-object interpenetration, similarly to [8]. The system provides two grasping modes: SnapGrasp, which snaps the hand to predefined closure values when grasping starts, and PhysicalGrasp, in which the hand pose is evaluated from physical collisions with the grasped object.

Both methods use similar conditions for detecting grasping start and stop events. We are particularly concerned with the latter, which is explained in next paragraph. For SnapGrasp, the predefined closure values are defined on a per-object and per-hand basis in the Unity editor.

The grasping system directly uses and manipulates the animation weights to evaluate grasping conditions and to avoid visual penetration of the hand with the object. Thus, directly using closure values to animate the hand as explained in Section 3 breaks most preexisting functionality of the grasping system. However, the team wanted to keep any modifications compatible with the rest of the SDK as much as possible. To do so, all the components of the hand were directly disabled using the unity animation system and only kept the *FingerMixers* active, which take care of interpolating and filtering the animation weights based both on the closure values of each thimble and on the grasping system. This way, no hand animations are used, and the *FingerMixers* component practically behaves only as an array of variables. Then, the filtered weights are used as the geometric variable to track the paths for the manipulators (Section 3.3) instead of using the thimble closure data directly. This ultimately results in keeping complete compatibility with the rest of the WeArt SDK while only animating the hand with the method explained in Section 3.

4.3.1 Physical Grasping Conditions

The *WeArtHandGraspingSystem* update cycle, performed at each frame, integrates several sequential procedures. First, collision detection functions are used to check for actual collisions between the finger colliders and objects in the scene.
Contact is determined if the thumb or the palm are in contact with the object and another additional finger is involved in the collision [9]. If contact is confirmed, the system may decide to temporarily lock the finger, visually preventing it from closing

beyond the point of contact (the ghost hand keeps tracking the closure of the real hand despite the collision). This mechanism avoids unrealistic interpenetration and ensures that the fingers adapt to the surface of the object. Once the states of the individual fingers have been updated, the system evaluates the overall conditions necessary to declare a valid grip.

The grasp is recognized when the thumb and at least one other finger have reached a level of closure compatible with contact with an object. In this case, the global state variable is updated from *Released* to *Grabbed*, and a grasping event is emitted.

5 Visual Hand Tracking with Meta Quest

The Meta Interaction SDK provides Unity functions and Prefabs to track the pose of both hands in real time. The SDK provides a computer vision model to estimate the pose (position and rotation) of different bones and joints of the hand, as shown in Figure 6. Detailed information is available in [10].

The SDK is developed for VR gaming, hence grasping tasks like in the scope of this project are not fully supported. Two different hand tracking methods are provided by Meta: OpenXR hands and OculusVR (OVR) hands. In both cases, the mesh of the hand is constructed at runtime based on the tracking data from the vision system.

OpenXR hands are mainly used in games and, while they support complex grasping of objects, this is obtained by snapping the hand model to predefined poses when a grasping action starts. Furthermore, the supported poses have to be defined manually for each graspable object and the hands do not support physics interactions with other rigid bodies.

OVR hands, on the other hand, support physics interactions by embedding rigid bodies and capsule colliders in the mesh for each finger, the palm and the wrist. This allows seamless interaction with other rigid bodies via the standard Unity Physics engine. We decided to use OVR hands in order to construct a collision-based grasping system that would also allow physics interactions.

5.1 OVR Hands Architecture

The main component of the architecture is OVRHand which controls all the other components responsible for hand rendering.

During runtime, upon detection of the hand, all the child components of OVRHand are built in order to return an accurate rendering. Each child component is composed of a script which main purpose is to handle the data flow.

The child components are:

- **OVRSkeleton**: responsible for building the skeleton and tracking the hand movement.
- **OVRMesh**: builds and holds the data necessary for the mesh rendering of the hand.
- **OVRSkeletonRenderer / OVRMeshRenderer / SkinnedMeshRenderer**: handle the rendering of the different components.

The main component updates the hand state, including but not limited to the pose of each bone, using the *FixedUpdate()* and *Update()* methods on OVRSkeleton.

The OVRSkeleton script tracks all the hand movements, therefore disabling the script causes the hand tracking to stop and the mesh to freeze in the last known state.

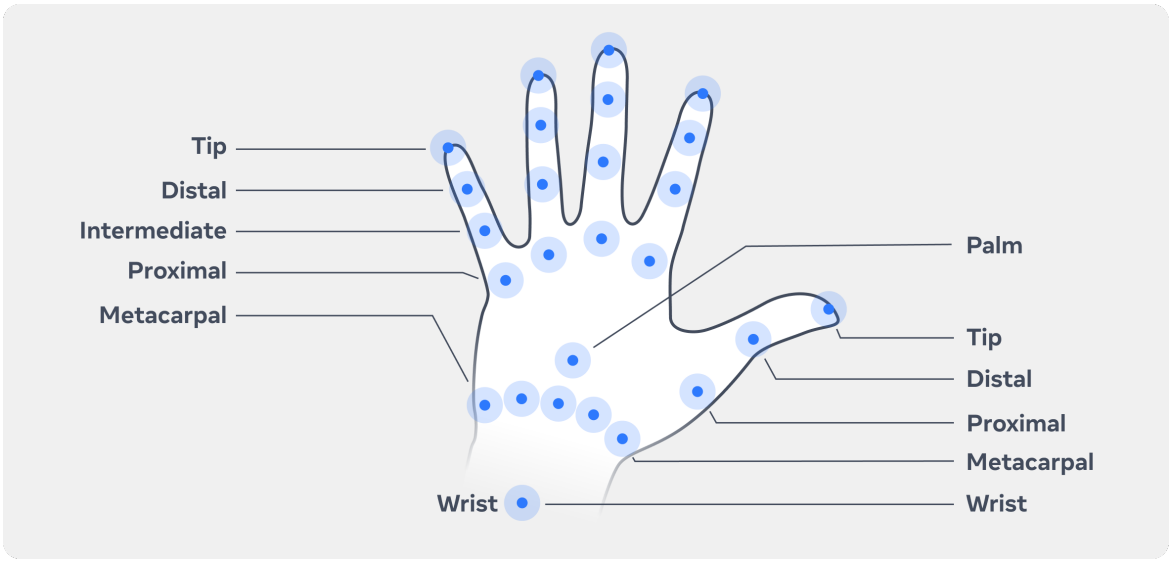


Figure 6: Bones and joints tracked by the Meta SDK

Meanwhile the renderers only handle the visual rendering so disabling the scripts will only cause the hand to disappear visually from the scene while still allowing tracking data to be computed.

5.2 Grasping Implementation

To allow the implementation of a secure grasping system many of the main hand components had to be involved and utilized. The main objective was to implement a feature that was coherent with the physics interactions between rigid bodies, therefore the grasping action is triggered by the fingers colliding with other rigid bodies.

5.2.1 Component setup

In order to handle each collision a new script *CollisionHandler* was added to each distal collider for each finger and to the palm. This script handles the collisions incoming and outgoing and calls custom scripts attached to the hand prefab.

Attaching a new script to the capsule colliders provided by Meta SDK was not an easy task since all the GameObjects that compose the hand are created at runtime. To work around this issue a new script *SkeletonCapsuleHook* was introduced which recognizes the capsule colliders in the fingers and attaches the script to only the relevant ones.

Additionally another custom script *PhysicsGraspController* was added to the hand prefab which is triggered by the *CollisionHandler* script once a new collision is detected starting or ending.

5.2.2 Grasping event conditions

To guarantee stability, the grasping event starts if and only if a collision including the thumb or the palm and at least another finger is detected towards the same rigidbody, and if the open palm is facing towards the object within a threshold distance of the hand.

When these conditions are met, the object pose is attached to that of the hand, and the object velocities and acceleration are set to zero. Furthermore, the minimum distance d_i between each distal and the object is saved, $d_{i,0} = d_i$.

A grasping event stops if $d_i > d_{i,0} + \epsilon$, for each finger initially in contact with the object. The variable ϵ is a positive scalar used as a threshold for robustness, to avoid situations in which small numerical errors cause a grasping event to oscillate between start and stop states [8].
In order to allow a smoother transition between grasping and not grasping states, we decided to introduce another condition in the *EndGrab()* function: if the hand is open the object is automatically released. The hand is considered open when all the fingers are extended. A finger is extended if the dot product between the finger's pointing direction and the palm normal is less than or equal to zero and their cross product has a positive X-component. The dot product condition is true when the finger is either completely extended or completely open, with the cross product condition distinguishing between the two cases. All computations are carried out in the palm's local reference frame. Therefore a finger is considered open if it is pointing roughly along the palm plane and it is oriented in the expected direction relative to the palm's local frame. Each parameter involved in the grasping event, such as closing parameters, opening threshold, etc., are configurable via sliders in Unity's workspace editor.

5.2.3 Avoidance of hand-object interpenetration

Although the core concept works and provides visually a natural grasping motion, it has the annoying side effect of the hand being able to pass through the object when it is blocked in the grasping state. To avoid this, the mesh needed to be blocked to its position at the start of the grasping action.

Unfortunately, the Meta SDK does not support this feature, and it was impossible to block the mesh state in the SDK, even with extensive modifications.

We worked around this issue by creating two copies of the hand. Originally there is only one OVRHand prefab for each actual hand (one for the user's left hand and one for the right hand), however, when a grasping action starts, a new copy of the hand is created and it is attached to the original hand itself as a child.

All the physics components are then disabled from the copy, including Meta's physics capsules and our own *PhysicsGraspController*, and any remaining children used for the grasping task are destroyed. This procedure is necessary in order to avoid

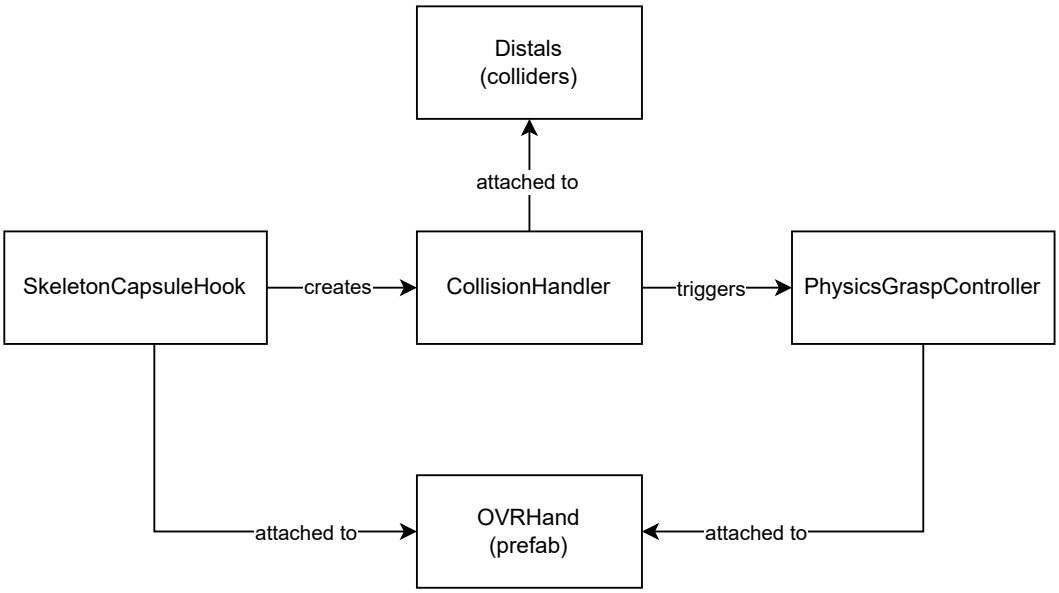


Figure 7: Block scheme of the scripts used

a (possibly, infinite) cascade of copies being created if the grasping start condition is met again. We then asynchronously wait for the copy OVRHand to finish one update cycle, so that it can populate the hand state structure with tracking data generated by the corresponding OVRSkeleton instance. After the first update is done:

- The OVRSkeleton component is disabled for the hand copy. This stops hand tracking and blocks the mesh in the last known state, i.e. that at the start of the grasping action. All rendering-related components for the hand are left active. This hand is then made to be a child of the original hand, so that its pose is bound to it.

- All rendering-related components of the original hand are disabled, but the hand tracking components are kept active, so that a grasping end event can still be detected

- When grasping ends, the copy hand is destroyed and all the rendering components of the original hand are enabled again.

It is important to note that disabling physics capsules at runtime from another class and tracking the order of the updates requires modification of the Meta SDK, such as changing the visibility and adding some parameters. The project's repository [11] contains all the modified scripts appropriately commented.

6 Comparison of the two methods

Precision is a fundamental part of grasping and it is the basis of most medical robotics systems. In order to truly assest the precision of the two different approaches followed, we decided to implement a much more complex Unity scene to test both grasping systems.

6.1 Testing Scene

The scene is organized in different tests, each designed to target a different aspect of grasping. In order to implement a diverse testing environment, different shapes were introduced for the objects.

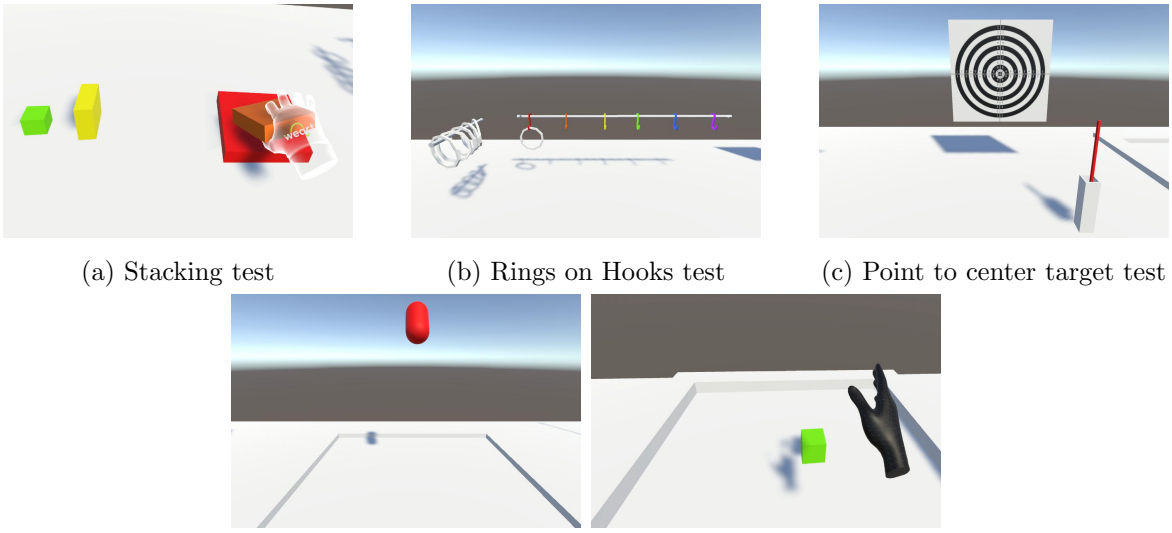
Each test is executed by three different users, and each user repeats the same test ten times. Each test uses different metrics for performance evaluation.

The devised tests were:

- **Stacking Test**: multiple boxes of different sizes are laid out and have to be arranged in a tower construction. This test is designed to target dexterous grasping which is essential for fine manipulation tasks in robotics.
- **Rings on Hooks**: a hook stand construction is placed in front of the user and multiple ring-shaped objects are hanging on a bar to the side. The objective is to hang each ring on the hooks, testing precision and aiming tasks.
- **Point to Center Target**: the user is faced with a target in front of them and a thin stick-shaped object. The user has to handle the object as a pointer, aiming for the center of the target.
- **Moving Target**: a free falling object has to be grasped mid-air. The experiment targets precision and fast reflexes.
- **Blind Test**: a cube is placed in a random position that is shown to the user for one second. The objective is to try and grasp the object without being able to see it at the time of grasping. This simulates, for instance, connection problems in a remote-operation procedure.

6.2 General findings

Both of the hand tracking methods presented some peculiarities that need to be taken into consideration when evaluating the results: The the TouchDiver G1 has bulky thimbles, which physically prevent the execution of fine pinching movements, especially on smaller objects, but has fine control of the closure value of each finger. The vision-based system, on the other hand, leaves the hand free, allowing for finer



(a) Stacking test (b) Rings on Hooks test (c) Point to center target test

(d) Moving object test (e) Blind test

Figure 8: The five tests

pinching movements, but the tracking is heavily dependent on the visibility of each finger. For example, depending on the pose of the hand, the thumb might obstruct the view of the pinky and ring fingers, causing both rendering and collision detection to be imprecise; other times the index could obstruct the view of the middle and successive fingers. The hand pose tracking also appears to feature heavy filtering, resulting in very small movements of the real hand barely having an effect on the virtual one. Similarly, the the TouchDiver G1 is dependent on the VR headset controller for tracking the pose of the hand, meaning that when the headset has no clear view of the controller (e.g. when the palm of the hand is pointing up) the hand pose tracking is imprecise, stutters, and oftentimes completely stops.

In general, it was observed that users used the first half of their attempts on each test to get accustomed with the tracking method used and understanding how to complete the test with good results, by taking advantage -or working around the disadvantages- of the peculiarities of each system, getting better results in the latter half of their attempts.

6.3 Stacking test

In this test users are required to arrange boxes of different dimensions one on top of the other, by decreasing side length. Time required from start to a complete tower is tracked, as well as the number of failed attempts to grasp an object. A failed attempt is defined as the user completing the movement to attempt a grasp, but not resulting in an effective grasp of the object.

Table 1 shows the results using the the TouchDiver, and Table 2 shows the results using the vision-based hand tracking. For the former, grasping has been reported very

	User 1		User 2		User 3	
Attempt	Time	Failed grasps	Time	Failed grasps	Time	Failed grasps
1	00:36:41	10	00:43,43	12	00:37,90	10
2	00:20,41	1	00:36,68	8	00:43,32	12
3	00:39,50	7	00:13,68	1	00:20,20	1
4	00:31,72	9	00:24,20	6	00:15,16	0
5	00:16,21	1	00:22,07	6	00:55,38	16
6	00:36,59	7	00:28,04	7	00:30,26	6
7	00:24,41	4	00:20,69	4	00:21,77	2
8	00:31,38	6	00:21,17	5	00:33,82	6
9	00:28,26	4	00:14,60	4	00:28,17	6
10	00:25,46	5	00:37,27	8	00:30,37	6

Table 1: Stacking test results using the TouchDiver G1

	User 1		User 2		User 3	
Attempt	Time	Failed grasps	Time	Failed grasps	Time	Failed grasps
1	00:00:32	3	00:26.0	3	00:17,43	3
2	00:19.0	3	00:31.0	6	00:16:75	2
3	00:20.0	1	00:26.0	4	00:17,11	2
4	00:18.0	2	00:21.0	2	00:11,46	0
5	00:18.0	3	00:20.0	3	00:15,36	0
6	00:30.0	5	00:33.0	9	00:09,90	0
7	00:24.0	3	00:16.0	1	00:17,84	5
8	00:14.0	0	01:05.0	13	00:10.0	3
9	00:13.0	0	00:15.0	1	00:14,88	2
10	00:16.0	2	00:21.0	5	00:10,19	2

Table 2: Stacking test results using vision-based tracking

much dependent of the closure of the thumb, which incorrect of wearing the thimble may prevent to be closed completely, resulting in a failed grasp. For the latter, users reported difficulties in releasing an already grasped object. This is likely due to some fingers obscuring the view of others, as users started easily releasing objects when the back of the hand pointed towards the headset, giving a full view of each finger.

6.4 Rings-on-hooks test

	User 1		User 2		User 3	
Attempt	Time	Failed attempts /5	Time	Failed attempts /5	Time	Failed attempts /5
1	00:59,03	0	00:44,25	4	00:42,85	2
2	00:56,10	0	00:58,22	3	00:47,29	1
3	00:32,96	1	01:08,12	4	00:41,28	3
4	00:27,96	1	00:46,92	5	00:40,50	2
5	00:30,66	0	01:21,97	4	00:25,16	4
6	00:42,69	1	00:55,61	2	00:34,75	1
7	00:41,51	0	00:47,81	3	00:48,25	0
8	00:33,42	0	00:50,55	3	00:35,14	0
9	00:30,24	0	00:39,10	2	00:37,24	0
10	00:26,56	1	00:41,14	5	00:37,71	0

Table 3: Rings-on-hooks test results using the TouchDiver G1

	User 1		User 2		User 3	
Attempt	Time	Failed attempts /5	Time	Failed attempts /5	Time	Failed attempts /5
1	00:42.0	3	00:40.0	4	00:21,71	4
2	00:47.0	0	00:31.0	5	00:20,52	2
3	00:33.0	4	00:34.0	4	00:29,48	1
4	00:42.0	3	00:29.0	4	00:29,43	2
5	00:38.0	1	00:34.0	4	00:22,46	1
6	00:43.0	2	00:41.0	4	00:23,43	1
7	00:33.0	1	00:26.0	4	00:21,94	2
8	00:31.0	0	00:40.0	3	00:16,41	3
9	00:33.0	2	00:39.0	1	00:33,62	1
10	00:41.0	3	00:21.0	4	00:25,22	1

Table 4: Rings-on-hooks test results using vision-based tracking

In this test users are required to move some rings from their initial position to a hook, testing precision and aiming. Five rings in total are present. Time required to

move all five rings and failed attempts are tracked. A failed attempt is defined as a ring falling on the table; users are not allowed to pick rings up from the table once they fall.

In this case, the the TouchDiver presents a variety of problems: first are the aforementioned problems with hand pose tracking and pinching of small objects. Furthermore, the WeArt Grasping System blocks the hand movement when colliding with other rigid bodies, giving misleading visual feedback. The grasping problems resulting from the ring model being too thin resulted in users avoiding full grasping maneuvers and instead using the finger or thumb as hooks and leveraging the (realistic) physics interaction with the object. Tables 3 and 4 show results using the TouchDiver and the vision-based tracking, respectively.

6.5 Moving object test

	User 1	User 2	User 3
Attempt	Successful attempt?	Successful attempt?	Successful attempt?
1	N	N	N
2	N	N	N
3	N	N	N
4	N	N	N
5	N	N	N
6	N	N	N
7	N	N	N
8	Y	N	N
9	N	Y	N
10	Y	N	N

Table 5: Moving object test results using the TouchDiver G1

In this test users are required to catch a free-falling capsule-shaped object in mid-air, testing precision and response speed. Each fall of the object is considered as an attempt, and successful catches are counted. A successful catch is defined as the object completely stopping in the hand, with the user voluntarily releasing it afterwards.

As can be observed in Tables 5 and 6, the TouchDiver performs much worse than vision-based tracking. This is because the preferred way to catch a falling object is either from the bottom or from the side, the former resulting in the line of sight

	User 1	User 2	User 3
Attempt	Successful attempt?	Successful attempt?	Successful attempt?
1	Y	N	Y
2	Y	N	Y
3	Y	N	N
4	Y	N	N
5	Y	N	N
6	N	N	Y
7	Y	Y	Y
8	Y	Y	N
9	Y	Y	Y
10	Y	Y	Y

Table 6: Moving object test results using vision-based tracking

between headset and controller being obstructed, hence hand pose tracking failing and the hand getting stuck in the last know position; the latter resulting in the object being pushed away by the hand itself before it can be successfully grasped.

6.6 Point to Center test

In this test users are required to handle a thin stick and point it at a dartboard, measuring grasping and aiming precision. Time required and score on the dartboard are measured. The score is counted as the first ring the stick touches.

As for the glove, users avoided problems with pinching caused by the bulky thimbles by using instead a grasping technique that only involves the palm and the other fingers, as one would for a cylindrical object. This also results in the tactile feedback of the glove being a misleading indicator of a successful grasp, with the visual feedback being much more indicative. Incidentally, this means that the stick is practically impossible to pick up again once it falls on the table. For the glove, this results in a score of zero points. For the vision-based tracking, the object can easily be picked up again. In this case, the score is not immediately counted as zero, but as a failed grasping attempt.

6.7 Blind test

In this test users are shown an object for one second, then they are required to grasp the object while their vision is made completely black, simulating connections problems in a remote-operating situation. An attempt is considered successful if the user can

	User 1		User 2		User 3	
Attempt	Time	Score	Time	Score	Time	Score
1	00:37,29	11	00:31,62	8	00:43,16	9
2	00:44,74	0	00:17,48	0	00:17,04	5
3	00:34,61	11	00:14,33	10	00:10,0	0
4	00:20,01	11	00:12,33	**7**	00:06,09	10
5	00:27,62	10	00:04,25	0	00:22,49	9
6	00:41,36	11	00:10,90	0	00:24,23	10
7	00:13,93	11	00:14,81	**9**	00:04,92	11
8	00:12,20	11	00:11,22	0	00:25,37	10
9	00:10,38	11	00:04,09	0	00:11,32	11
10	00:10,06	11	00:07,59	0	00:08,49	0

Table 7: Point-to-center test results using the TouchDiver G1. Scores in bold indicate successful but unnatural grasps, for example with the stick penetrating the hand.

	User 1			User 2			User 3		
Attempt	Time	Score	Failed grasps	Time	Score	Failed grasps	Time	Score	Failed grasps
1	00:18.0	10	2	00:10.0	7	3	00:08,34	8	0
2	00:05.0	11	1	00:12.0	10	2	00:11,26	11	1
3	00:08.0	11	1	00:05.0	10	0	00:06,04	11	0
4	00:10.0	11	1	00:12.0	9	4	00:04,12	11	0
5	00:11.0	10	2	00:10.0	9	1	00:03,84	10	0
6	00:11.0	10	1	00:10.0	10	1	00:06,05	11	0
7	00:22.0	11	4	00:18.0	9	4	00:05,12	11	0
8	00:12.0	11	3	00:10.0	9	1	00:04,99	10	0
9	00:09.0	10	1	00:09.0	10	1	00:11,71	10	1
10	00:09.0	11	0	00:17.0	10	5	00:08,35	11	0

Table 8: Point-to-center test results using vision-based tracking

grasp the object within one minute of being blinded, otherwhise the test is considered failed (N). Users can only start to moving their hands after being blinded.

Intuition would suggest that the tactile feedback capabilities of the the TouchDiver G1 would help in absence of visual feedback. In practice, the absence of actual force

	User 1	User 2	User 3
Attempt	Time	Time	Time
1	N	00:04.0	00:03,98
2	N	N	00:05,61
3	00:24	N	N
4	N	N	00:09,67
5	N	N	N
6	N	N	00:08,39
7	00:10	N	00:09,50
8	00:08	N	00:05,81
9	N	N	00:03,10
10	N	00:08.0	N

Table 9: Blind test results using the TouchDiver G1

	User 1	User 2	User 3
Attempt	Time	Time	Time
1	00:02,41	00:00:03,0	00:02,35
2	00:02,54	00:00:02,0	00:05,56
3	00:02,42	00:00:03,0	00:05,55
4	00:02,29	00:00:03,0	00:03,08
5	00:02,25	00:00:02,0	00:03,10
6	00:03,55	00:00:02,0	00:03,64
7	00:02,82	00:00:03,0	00:05,10
8	00:02,56	00:00:03,0	00:03,32
9	00:02,95	00:00:03,0	00:04,02
10	00:02,57	00:00:02,0	00:03,32

Table 10: Blind test results using vision-based tracking

feedback and poor tactile resolution of the glove, combined with the aforementioned problems with hand tracking and the grasping system getting easily stuck in other rigid bodies such at the table, means that the tactile feedback is often misleading, or not activating at all even if the hand is in contact with the table, but the fingers are not, making surface exploration not a useful (nor usable) indicator. All these factors combined result in worse performance when using the glove compared with the vision-based tracking. Results are shown in Tables 9 and 10.

7 Conclusions

This project explored and compared different methods to interact with a virtual reality environment and perform grasping tasks on virtual objects. In particular, the use of a haptic interface and vision-based hand tracking were explored. The implementation of both methods uses pre-existing Software Development Kits (SDKs) to communicate with the hardware (the WeArt glove and the vision system of the VR headset).

On one hand, using pre-existing SDKs simplified part of the development process; on the other, it introduced several limitations which needed to be taken into account during the design of our own grasping system.

This is true for both tracking methods: The WeArt SDK is heavily realiant on the Unity animation system, and our modifications had to be designed so no to break compatiblity with the rest of the SDK. We also specifically found the Meta SDK to be extremely rigid, with poorly documented code and an obscure architecture. The API reference offered by Meta is superficial and the documentation features mostly introductory paragraphs to explain the architecture. This made any modification a much more onerous labor compared to the WeArt SDK.

Two notable example of this are the deep copy of the original hand explained in Section 5.2.3 and the fact that, with the Meta SDK, hands can fully penetrate rigid body objects when those are constrained to a fixed pose, like the table in Section 6.

The motivation behind the former is that the Meta SDK does not in any way allow direct modification of the mesh of the hand, nor does it expose the joint values shown in Figure 6. The proposed workaround is the simplest solution that can be implemented in Unity. Other possible implementations could make extensive use of the Reflection API.

As for the latter, the Meta SDK continuously moves the hands to the position tracked by the headset, ignoring collisions with both kinematic objects and non-kinematic objects whose position is fixed. For example, the hand could be forced to be inside another rigid body object, resulting in interactions the Unity physics SDK is not able to cope with. We were not able to work around this issue, as the hand pose tracking is abstracted away from the user in the Meta SDK.

Finally, when using the Meta SDK, a gap is sometimes present between the hand and the grasped object. This is because colliders in Unity are slightly bigger than the object they are associated with, to give a robustness threshold to prevent fast-moving objects from penetrating the objects they collide with. In fact, the gap is not present when rapidly closing the hand around the object, but it is more pronounced when the hand movement is slower, as in most of the tests presented in this report. While this phenomenon is accounted for in the WeArt SDK, it is not in the Meta SDK, and the aforementioned limitations prevent the implementation of a fix.

Despite the added constraints and limitations, it was still possible to implement two

fully working grasping systems, one for each hand tracking method, with acceptable accuracy and realistic physics interactions with other rigid bodies.

References

[1] Mingzhao Zhou and Nadine Aburumman. Grasping objects in immersive virtual reality environments: Challenges and current techniques. In *2024 10th International Conference on Virtual Reality (ICVR)*, pages 190–197, 2024.

[2] Kun-Woo Kim, Jun-Seong Kim, Hyo-Joon Kim, and Seong-Yong Moon. Development and evaluation of a virtual reality and haptic-integrated clinical simulation system for osce-based blood-related procedures. *IEEE Access*, 13:192947–192957, 2025.

[3] Robert V. O'Toole, Robert R. Playter, Thomas M. Krummel, William C. Blank, Nancy H. Cornelius, Webb R. Roberts, Whitney J. Bell, and Marc Raibert. Measuring and developing suturing technique with a virtual reality surgical simulator. *Journal of the American College of Surgeons*, 189(1):114–127, 1999.

[4] Andrea Turolla, Mauro Dam, Laura Ventura, Paolo Tonin, Michela Agostini, Carla Zucconi, Pawel Kiper, Annachiara Cagnin, and Lamberto Piron. Virtual reality for the rehabilitation of the upper limb motor function after stroke: a prospective controlled trial. *Journal of NeuroEngineering and Rehabilitation*, 10(1):85, Aug 2013.

[5] Mindy F. Levin, Patrice L. Weiss, and Emily A. Keshner. Emergence of virtual reality as a tool for upper limb rehabilitation: Incorporation of motor control and motor learning principles. *Physical Therapy*, 95(3):415–425, 03 2015.

[6] Gustavo Saposnik, Mindy Levin, and for the Stroke Outcome Research Canada (SORCan) Working Group. Virtual reality in stroke rehabilitation. *Stroke*, 42(5):1380–1386, 2011.

[7] Amal Bouatrous, Nadia Zenati, Abdelkrim Meziane, and Chafiaa Hamitouche. A virtual reality-based serious game designed for personalized hand motor rehabilitation. In *2023 International Conference on Networking and Advanced Systems (ICNAS)*, pages 1–5, 2023.

[8] T. Ullmann and J. Sauer. Intuitive virtual grasping for non haptic environments. In *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*, pages 373–457, 2000.

[9] Hiroo Iwata. Artificial reality with force-feedback: development of desktop virtual space with compact master manipulator. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, page 165–170, New York, NY, USA, 1990. Association for Computing Machinery.

[10] Facebook Meta. Meta Interaction SDK. https://developers.meta.com/horizon/documentation/unity/unity-isdk-interaction-sdk-overview/, 2025.

[11] Medical Robotics Project GitHub Repository. https://github.com/EmaMaker/uni-medical-robotics-project/.

[12] WeArt. WeArt SDK for TouchDriver G1. https://weart.it/developer-guide-td-g1/, 2024.