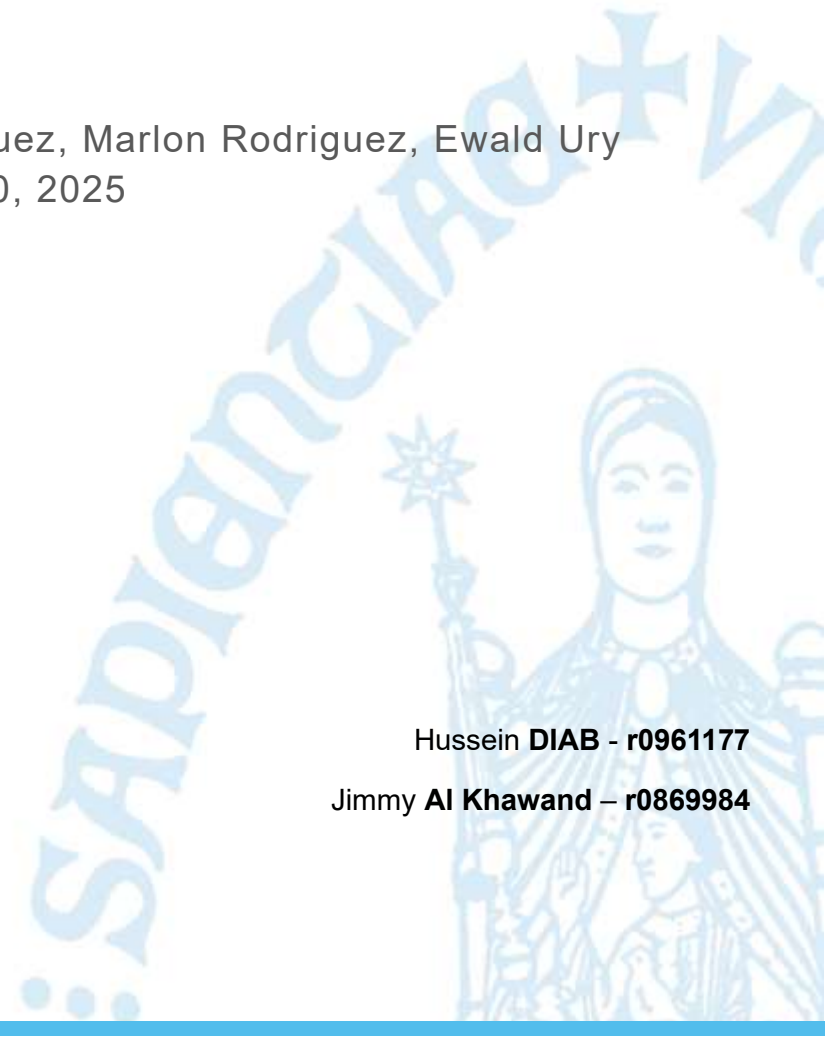
KU LEUVEN

Faculty Of Engineering Technology
Group T Leuven Campus

VR-Based Vitals Assessment Training with Haptic Feedback

Course: Medical Devices
Professor: Carlos Rodriguez, Marlon Rodriguez, Ewald Ury
Submission Date: May 30, 2025



Hussein **DIAB** - **r0961177**

Jimmy **Al Khawand** – **r0869984**

Table of Contents

List of Figures

1. Introduction

In emergency medical scenarios, the ability to accurately assess a patient's vital signs such as detecting a carotid pulse or heartbeat is critical. However, current educational tools fall short of providing realistic training environments for these tactile-dependent skills. Traditional mannequins lack the capability to simulate physiological sensations, which undermines the development of proper palpation technique, diagnostic intuition, and muscle memory. This deficiency can result in improper hand placement, misinterpretation of a patient's condition, or delays in administering life-saving interventions [1].

To address these limitations, we propose a VR-based vitals assessment training system that integrates tracking hand position via the Meta Quest 3 using the meta quest controllers with a custom-designed haptic glove outfitted with Drake TacHammer actuators. Research has shown that tactile realism is key to training physical examination skills effectively [1], and simulating haptic sensations, either physically or visually, can significantly improve users' sense of presence and accuracy in virtual tasks [2].

In our system, trainees interact with 2 virtual patients and follow standard first aid protocols to assess pulse and heartbeat. The Meta Quest 3 tracks the movement of its controllers in real time, while proximity to anatomical target zones (e.g., neck or chest) triggers tactile feedback via haptic motors embedded in the glove that simulates a heartbeat and pulse.

Trainees must interpret these cues and decide whether to initiate emergency action such as cardiopulmonary resuscitation (CPR) or the patient doesn't need further medical attention. This system combines immersive visuals with dynamic haptic feedback to offer a multi-sensory learning experience. By simulating tactile feedback in an intuitive and spatially accurate way, the platform enhances medical learners' ability to make confident decisions on the spot.

The training objectives of the system are centered on enhancing core clinical skills through immersive, tactile-based interaction. First, it aims to help learners develop proper placement for detecting vital signs such as the carotid pulse and heartbeat. By delivering location-specific haptic feedback, the system trains users to identify the tactile signatures that differentiate normal from abnormal physiological states. Additionally, the simulation encourages trainees to practice clinical decision-making in real-time emergency scenarios, where quick, informed judgments are essential. Finally, through repeated exposure and interaction, the system fosters the development of muscle memory and diagnostic proficiency, preparing users for high-pressure real-world situations.

2. Supplies

To replicate our VR-based vitals assessment training system, the components and materials required are presented in table 1 below. All items were sourced from the KU Leuven Haptic Interfaces Lab inventory [3].

Table 1: Components needed for this project

Component	Description
Meta Quest 3	Standalone VR headset with full hand tracking using the Meta controllers.
Unity Engine	Used for developing the 3D virtual environment, tracking logic, and interaction logic.
Arduino Micro	Microcontroller board that receives serial commands from Unity to drive the haptic actuators.
Custom Haptic Glove	A glove with haptic motors positioned one on the index fingertip and the other on the palm.
Drake TacHammer Actuators	Simulate heartbeat and pulse feedback, use the multiplexer
TCA9548A Multiplexer	Allows control of multiple DRV2605 motor drivers using a shared I²C bus from the Arduino.
Jumper Wires, Breadboard, USB Cable	Used for electrical connectivity, signal routing, and power distribution across system components.

Figure 2.1: Meta Quest 3s with the controllers

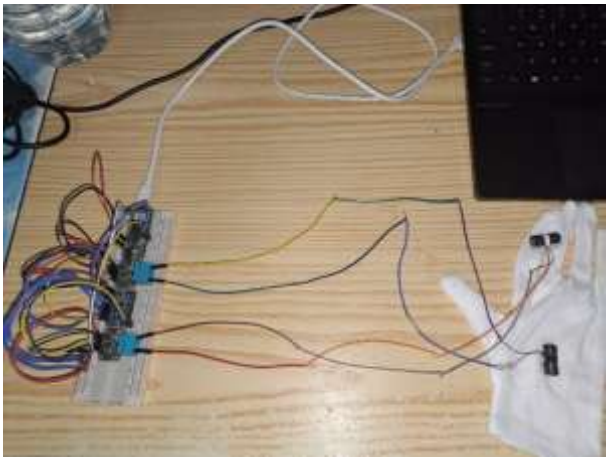


Figure 2.2: The Arduino micro with the rest of the components mounted

All wiring and construction followed recommended best practices from the lab documentation to ensure consistent actuator control and stable integration between hardware and software components [3].

3. Method

This section outlines the technical process followed to build the VR-based vitals assessment training system, from conceptual design to hardware-software integration. It is structured in progressive steps aligned with component layers: virtual simulation, communication logic, haptic glove design, and evaluation feedback.

3.1 Designing the Virtual Environment (Unity + Meta Quest 3)

The simulation begins with a custom-designed 3D environment in Unity, featuring two virtual patients that reflect contrasting clinical conditions: one exhibiting stable vital signs, and another presenting critical indicators such as a faint heartbeat or absent pulse. The scene also includes clear on-screen instructions guiding the trainee through the assessment process. After the trainee makes their decision, the system provides immediate feedback to indicate whether their diagnosis was accurate. In this section we will discuss the process of creating the Unity-based environment and configuring Meta Quest integration for interactive VR deployment.

3.1.1 Unity Environment Setup and Simulation Configuration

To build this environment, development started with a new Unity project. The API compatibility was configured to use the .NET Framework to ensure proper functionality of all C# scripts. This is done by going to Edit tab in unity, Project Settings, Player, then you find the Api compatibility Level and change it to the .NET Framework as can be shown in figure 3.1 to the right.

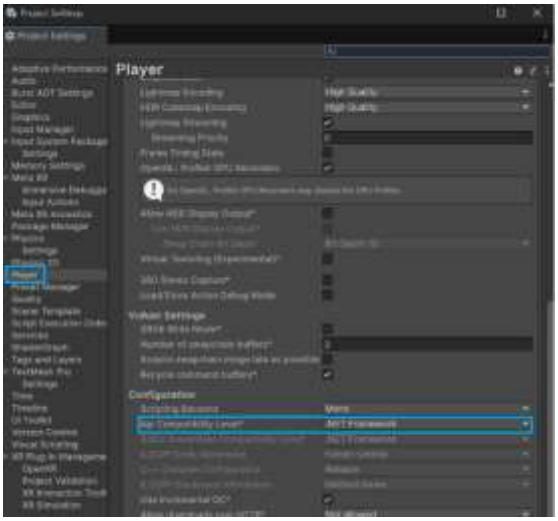


Figure 3.1: Setting up the Api compatibility level

Key assets were then added from the Unity Asset Store [4], including the Meta XR All-in-One SDK, which provides the necessary tools for Meta Quest integration, and the “Banana Man” character model, which serves as the virtual patient.

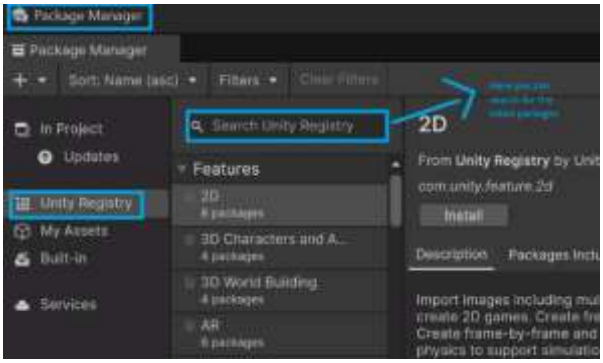


Figure 3.2: How to add more Packages

Using Unity’s Package Manager, both the Meta XR SDK and the Banana Man asset were imported into the project. The Package Manager can be reached by going to the window tab at the top and then Package Manager. There, additional essential packages were installed from the Unity Registry: the XR Interaction Toolkit and Unity OpenXR Meta, as we can see in figure 3.2.

Once these components were in place, configuration proceeded through the Project Settings under XR Plugin Management. There the Open XR feature should be enabled along with the Meta Quest and Meta XR feature groups as represented in figure 3.3.

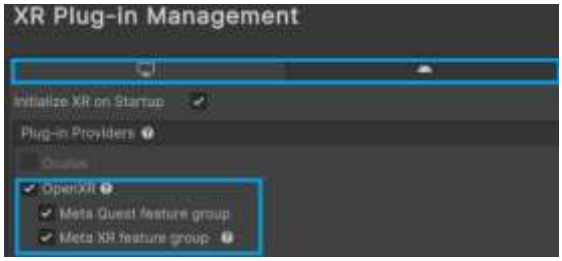


Figure 3.3: Enabling Open XR

Within the OpenXR section, the following features were enabled: Meta XR, Meta Quest, hand interaction poses. In addition to that, the features Hand interaction Prodile and Meta Quest Touch Profiles were added under “Enable Interaction Profile”. Make sure to add these features for both Android and PC as shown in figure 3.4 below.

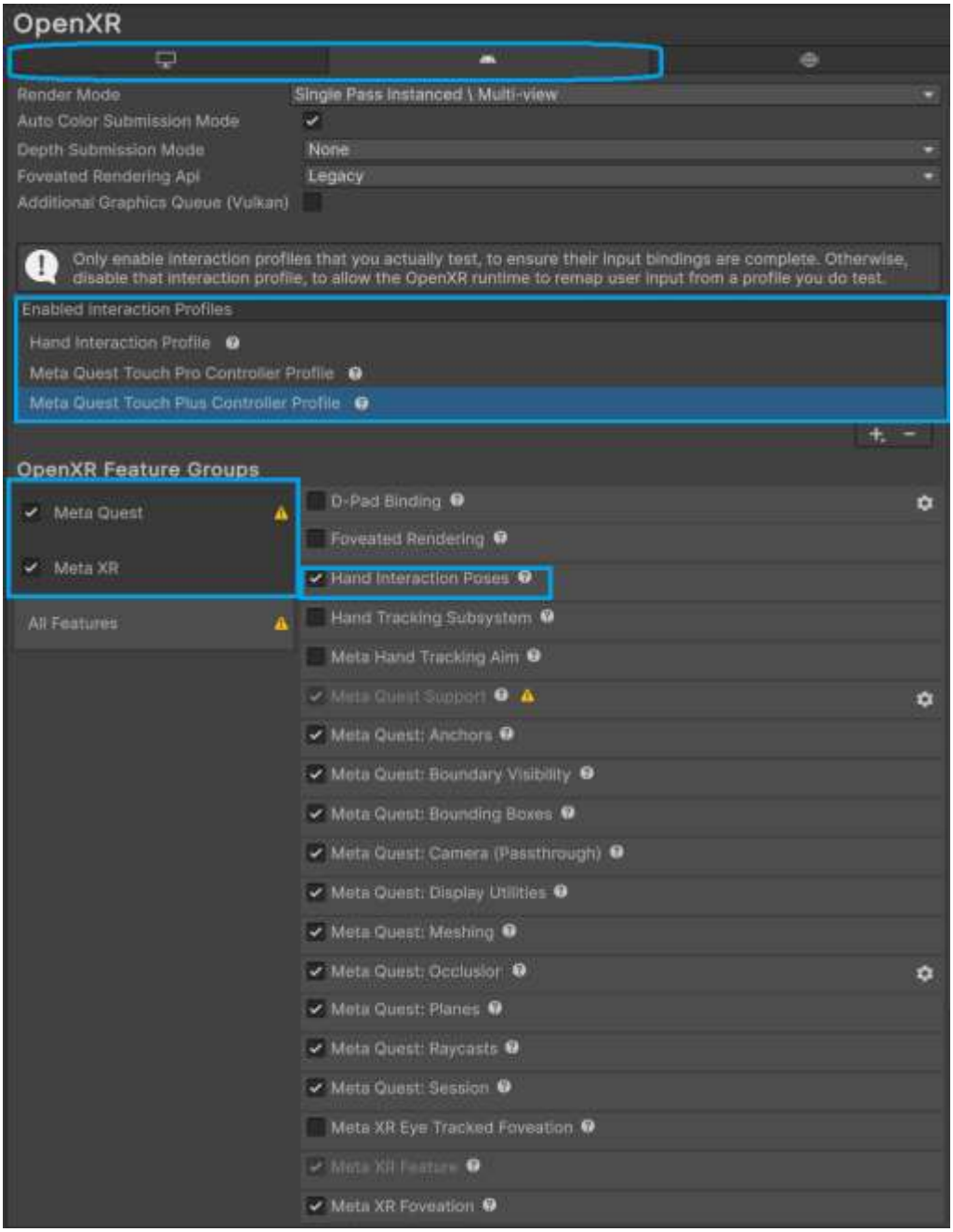


Figure 3.4: Enabling more features

With these configurations complete, the foundational setup for the interactive VR environment was established, enabling the development of realistic training interactions within the scene.

The XR Interaction Toolkit includes a sample scene titled ‘HandsDemoScene’, which provides a preconfigured environment for hand-based interactions. Although not all components of the demo were necessary, the scene was used as a base due to its compatibility with Meta Quest devices. The environment was then customized and adapted to fit the desired simulation layout and interaction requirements. The figure 3.5 below illustrates how to reach the HandsDemoScene

Figure 3.5: Adding the HandsDemoScene

To populate the scene, two Banana Man models were placed into the environment, each representing a different patient scenario. Each patient model was enhanced with interaction zones to simulate the physical assessment of vital signs. These zones were created by placing a spherical collider at the chest to represent the heart location, and a capsule collider at the neck to simulate the carotid pulse. The colliders were configured with the "Is Trigger" option enabled and made child objects (figure 3.6) of the respective Banana Man model, specifically under the left shoulder for the heart and under the head for the pulse, thus ensuring spatial consistency during movement or animation.

Figure 3.6: Placing the Heart 1 and Pulse 1 objects in the Hierarchy for patient 1

To enable user interaction via Meta Quest controllers, a new tag named "Hand" was created. This tag was assigned to both left and right controller objects found under the XR Origin (Rig) hierarchy at the "Camera Offset" level. Each controller was augmented with a collider and a rigidbody component, allowing Unity’s physics engine to detect interactions with trigger zones.

For signal handling, a custom C# script named CollisyionTrigger2.0 was attached to each trigger zone. This script monitored collisions and, upon detecting an object tagged "Hand," issued a corresponding command such as Pulse, Heart, FaintPulse, or FaintHeart, based on a predefined Trigger Type string configured via the Unity Inspector as seen in figure 3.7.

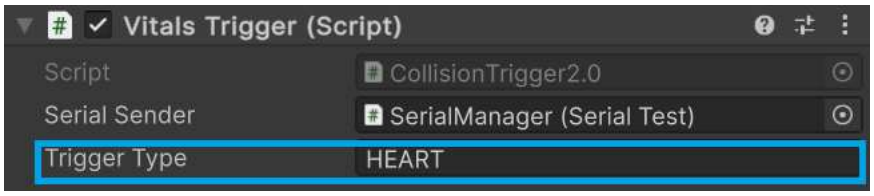


Figure 3.7: Assigning the Trigger Type for the objects

Each command was routed through a reference to a central communication script, SerialTest, responsible for managing the serial connection to the Arduino board. Refer to appendix 1 and appendix 2 at the end for the actual codes used.

A dedicated empty GameObject labeled SerialManager was added to the scene and assigned the SerialTest script. This script initialized a serial port connection (e.g., COM4, 9600 baud) and exposed public methods to send specific serial messages upon request. When a trigger zone was activated, SerialTest dispatched the appropriate message through the serial interface, prompting Arduino to generate the desired haptic feedback using the TacHammer actuators on the glove.

The virtual training environment was further customized to present two patients, each representing a distinct medical condition. Beside each patient, a dedicated decision panel was placed, prompting the trainee to assess whether the patient requires cardiopulmonary resuscitation (CPR) and immediate medical attention. These panels simulate a real-time decision-making interface, encouraging the trainee to apply clinical judgment based on the tactile and visual cues provided during the assessment.

Above each decision panel, a text area was allocated for feedback. This space remains hidden until the trainee interacts with the corresponding "reveal" button. Once triggered, the answer is displayed which indicates whether the chosen action was correct. This functionality is managed by the AnswerRevealUI script (Appendix 3), which is attached to a central UI controller object in the scene.

The script references two TextMeshPro Text elements: answerText1 and answerText2, corresponding to patient 1 and patient 2 respectively. When the trainee presses the button assigned to Patient 1 or Patient 2, the script updates the respective text field with predefined feedback. Specifically, it reveals that Patient 1 does not require CPR, while Patient 2 is in critical condition and requires immediate intervention. This setup reinforces learning by providing immediate, scenario-specific feedback upon completion of the assessment task.

As an additional enhancement to increase realism and immersion, simple animations were incorporated for both Banana Man models. Free character animations were sourced from the Mixamo platform [5]: one depicting normal, steady breathing for the stable patient, and another showing shallow or labored breathing for the critical patient. These animations were imported into Unity and integrated using basic Animator Controllers. Since we had limited experience in 3D animation, publicly available YouTube tutorials were followed to guide the implementation.

The animations were intentionally kept simple to ensure stability within the VR environment while still effectively conveying each patient's condition. In addition to the animations, a breathing sound was assigned to each patient, allowing the trainee to hear the respiratory patterns during the simulation. This added an additional sensory layer to the experience, further enhancing realism and engagement.

The final Unity environment, shown in figure 3.8 below, represents the primary VR setting in which the trainee carries out the simulation.

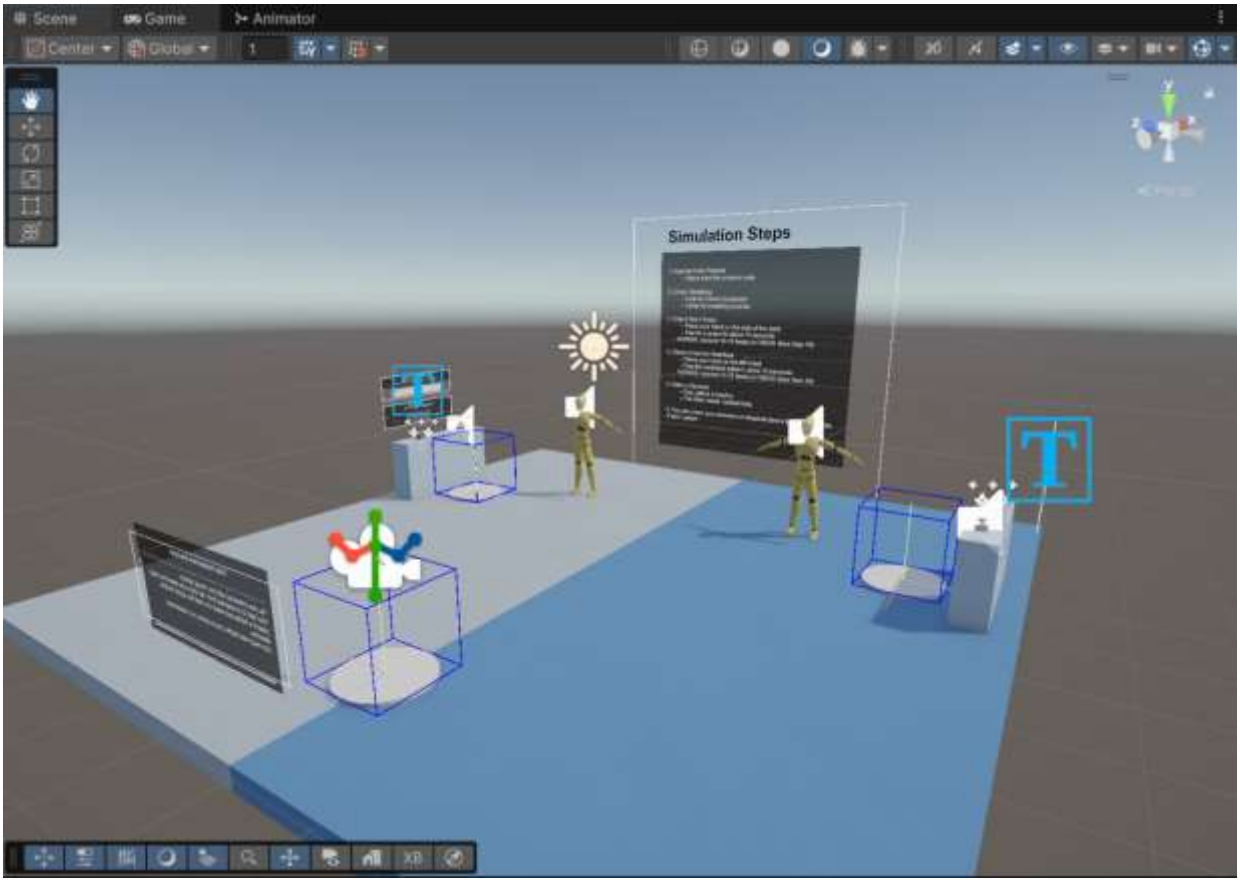


Figure 3.8: Simulation environment

3.1.2 Meta Quest configuration

To enable VR deployment and testing, the Meta Quest 3 headset was connected to the development laptop using the Meta Quest Link application, which was downloaded directly from the official Meta website. Once installed, the Link app provided a stable bridge between the headset and the PC, allowing for seamless communication with the Unity Editor. After establishing the Air link connection and enabling Link mode on the headset, the VR scene could be tested and interacted in real time within Unity. This setup enabled rapid iteration during development and ensured accurate alignment between virtual hand movements and in-scene interactions.

Using the Meta Quest 3's built-in controller tracking, hand proximity to the predefined vital sign zones is continuously monitored. Detection is achieved through the use of 3D colliders combined with C# logic scripts that evaluate spatial overlap between the user's virtual hand and the trigger zones attached to the patient model. As detailed previously, when the hand

intersects a target collider, a signal is sent to activate the corresponding haptic feedback through the Arduino system. The Meta Quest 3's wireless, high-precision tracking without the need for external sensors makes it well-suited for creating portable and replicable standalone VR training environments.

3.2 Integrating Unity and Arduino via Serial Communication

Integration between Unity and Arduino was established through serial communication using a USB connection. In Unity, each anatomical zone is mapped to a unique identifier. When the user's hand enters a detection zone, the system sends a corresponding command via USB serial to the Arduino ('Pulse', 'Heart', 'FaintPulse', or 'FaintHeart')

On the Arduino side, a serial listener parses the incoming command and activates the correct DRV2605 output channel through the multiplexer. This ensures location-specific and condition-specific vibration.

The Arduino code defines two main functions for this purpose. When it receives the commands "Pulse" or "Heart," it executes a function that generates a normal heartbeat pattern using the TacHammer actuator using a dual-pulse sequence that mimics a steady cardiovascular signal. In contrast, when it receives "FaintPulse" or "FaintHeart," it calls an alternative function designed to simulate a weaker, irregular heartbeat. This faint pattern is characterized by reduced intensity and longer pauses, effectively conveying a critical physiological condition to the user through haptic feedback.

3.3 Constructing the Haptic Glove

The haptic glove was constructed using only available lab materials and a standard fabric glove. The wires of the TacHammer actuators were extended to allow flexible placement, and two motors were affixed directly to the glove: one on the index fingertip to simulate the carotid pulse and another on the palm to simulate the heartbeat. Both actuators were secured using adhesive, ensuring reliable skin contact for effective tactile feedback during interaction with the virtual patient.

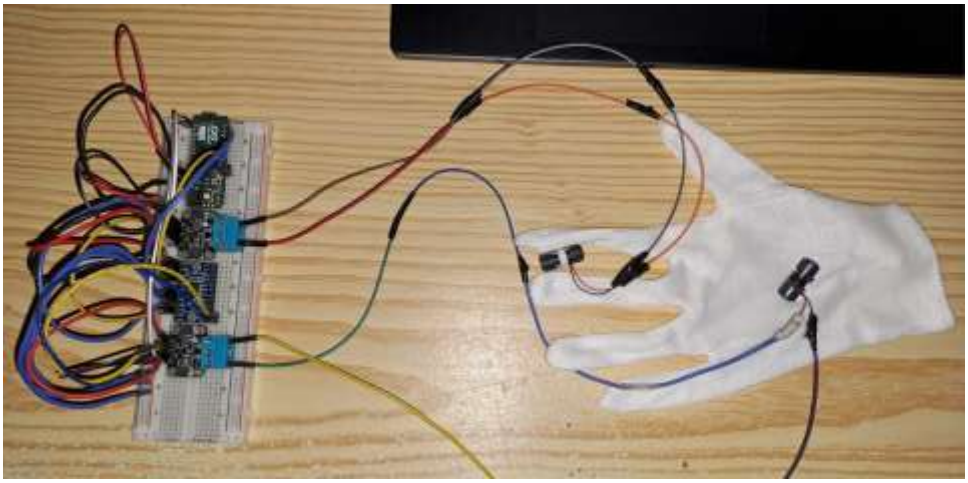


Figure 3.9: The fabric glove with the actuators

4. Discussion

The primary goal of our system was to bridge the gap between theoretical first aid training and realistic tactile experiences in vital sign assessment. Through integrating VR interaction with location-specific haptic feedback, we aimed to provide learners with a training platform that builds intuition, spatial awareness, and clinical decision-making skills.

4.1 Performance Evaluation

Overall, the prototype performed reliably in simulating both heartbeat and pulse sensations via the haptic glove. The system successfully tracked the controller position in 3D space to feedback zones on the virtual patient, enabling:

- Clear distinction between normal and absent vitals.
- Accurate vibration triggering at the chest and neck locations.
- Realistic simulation of weak or strong vitals through modulated feedback intensity and waveform patterns.

Users were able to identify correct anatomical zones with tactile guidance and make scenario-appropriate decisions such as whether or not to initiate CPR.

To gather preliminary feedback, the system was tested by a small group of five participants, all of whom were close friends. These users interacted with the VR environment using the haptic setup and were asked to assess the usability, clarity of the interface, and realism of the feedback on a scale from 1 to 5. The results are displayed in table 2 below. The overall response was positive, with participants highlighting the intuitiveness of the user interface and the educational value of the tactile cues. Most users reported that the system provided a clear and engaging way to simulate vital sign assessment and decision-making in a virtual clinical scenario. Despite its simplicity, the setup was seen as effective and immersive.

Though the feedback was generally positive, influenced by the participants being friends of ours, it nonetheless suggests that the system is clear, intuitive, and easy to use. Participants were able to navigate the interface and understand the objectives without additional guidance, indicating strong usability fundamentals. However, some limitations were also reported during the sessions, which will be discussed in detail in the following section.

Table 2: Summary of User Feedback

Participant	UI Clarity	Realism of Haptics	Overall Usefulness
P1	5	4	5
P2	4	4	5
P3	5	5	5
P4	5	4	4
P5	4	5	5

4.2 Technical and User-Experience Constraints

Despite the overall positive reception, several technical and practical limitations were identified during testing.

The most prominent issue was the complexity of the wiring system. The integration of multiple DRV2605 haptic drivers in combination with an I²C multiplexer increased the likelihood of loose or unstable connections. Additionally, this setup restricted the fluidity of hand movement, particularly due to the reliance on physical controllers instead of full hand-tracking. The bulk and stiffness of the wiring, combined with the need to grip the controllers, reduced the natural range of motion and hindered the realism of the interaction.

Additionally, the glove's comfort and weight distribution require improvement, particularly in extended actuator wiring. Also, one notable limitation observed was the rapid heating of the TacHammer actuators during extended use, which may affect user comfort and long-term device reliability.

From a scenario design perspective, it was observed that while the tactile cues were effective, the system's ability to simulate clinical urgency could be significantly enhanced by integrating time-based decision pressure, prompting trainees to respond more realistically in high-stress conditions.

4.3 Alignment with the Medical Challenge

Our system directly addresses the core challenge presented in the introduction, which is the absence of tactile realism in conventional first aid training tools. Traditional mannequins, while useful for procedural demonstrations, often fail to provide the nuanced physical feedback necessary for developing intuition in vital sign assessment. In contrast, the proposed VR-based simulation integrates real-time vibrotactile feedback, creating a more immersive and responsive training environment. This approach allows learners to go beyond visual cues and engage their sense of touch in a way that mirrors real-world clinical interactions.

Specifically, the system enables trainees to:

- Practice correct hand placement based on feedback: By detecting proximity to anatomical zones and responding with vibration cues, the system guides users toward accurate palpation techniques. Trainees learn to identify precise locations on the virtual patient's body, reinforcing spatial awareness and muscle memory.

- Develop diagnostic reasoning by interpreting vibration patterns: Different haptic signals are used to simulate varying physiological states, such as a strong, steady pulse versus a faint or absent one. This requires users to analyze the nature of the feedback and make informed decisions thereby fostering critical thinking in high-pressure scenarios.

Overall, the integration of tactile feedback not only bridges the gap between theory and practice but also supports the development of both procedural skill and clinical judgment.

5. Conclusion and Future Work

This project set out to create a more realistic and engaging training system for assessing vital signs by merging VR simulation, real-time hand tracking, and location-specific haptic feedback. Our prototype demonstrates that simulating heartbeat and pulse through a customized glove can effectively replicate essential clinical cues that are typically absent from traditional mannequins.

5.1 Key Findings

- The system successfully provides distinct tactile feedback for vital sign detection at key anatomical locations (chest and neck), enabling learners to make decisions such as initiating CPR.
- Vibration intensity and waveform patterns proved effective for distinguishing between normal and critical physiological states.
- Real-time spatial interaction via the Meta Quest 3 allows for intuitive making the training more immersive.

These results support the conclusion introduced in the beginning that multi-sensory feedback improves medical training quality, particularly for building palpation skills and response confidence.

5.2 Future Improvements

To enhance realism, usability, and scalability, several recommendations were identified that could significantly improve the overall performance and user experience of the system:

- Wireless glove system: Replace USB and jumper wires with some sort of low-latency wireless microcontroller to improve comfort and mobility.
- Expanded physiological feedback: Add additional cues such as respiratory simulation or temperature variation to train broader diagnostic skills.
- Advanced haptic modulation: Implement more nuanced waveforms to differentiate bradycardia, tachycardia, or arrhythmia patterns.
- Performance logging and analytics: Integrate back-end systems to automatically record user decisions, timing, and positioning accuracy for instructor review.

5.3 Lessons Learned

In summary, this prototype demonstrates the potential of integrating spatially-aware VR interaction with targeted haptic feedback to enhance the realism and effectiveness of medical training simulations. While the system introduces certain hardware complexities, it provides valuable insights into how tactile realism can support the development of essential clinical skills. Careful scenario design remains crucial to ensure that educational goals are met without overwhelming novice learners. Ultimately, this work lays the groundwork for future advancements in immersive, accessible, and cost-effective haptic VR systems aimed at improving clinical preparedness and response confidence.

6. References

[1] D. A. Nicholls and L. Sweet, “Developing palpation skills in physiotherapy education,” *Physiotherapy Research International*, vol. 17, no. 4, pp. 214–223, 2012.

[2] A. Lécuyer, “Simulating haptic feedback using vision: A survey of research and applications of pseudo-haptic feedback,” *Presence: Teleoperators and Virtual Environments*, vol. 18, no. 1, pp. 39–53, 2009.

[3] KU Leuven Haptic Interfaces Lab Documents, “Lab Hardware Manuals & Unity Integration Guides,” Internal Documentation, 2025.

[4] Unity Asset Store. [Online]. Available: https://assetstore.unity.com/

[5] Mixamo – 3D character animation. [Online]. Available: https://www.mixamo.com/

[6] Meta Quest Support – Set up Meta Quest Link. [Online]. Available: https://www.meta.com/en-gb/help/quest/1517439565442928/

7. Appendix

Appendix 1: VitalsTrigger script

```
using UnityEngine;

public class VitalsTrigger : MonoBehaviour
{
    public SerialTest serialSender;
    public string triggerType = "PULSE"; // Set this in Unity inspector:
      PULSE, HEART, FAINTPULSE, FAINTHEART

    private void OnTriggerEnter(Collider other)
    {
        if (!enabled || serialSender == null)
        {
            Debug.LogWarning($"[VitalsTrigger] Trigger ignored.
              SerialSender is null or script is disabled on
              {gameObject.name}");
            return;
        }

        if (other.CompareTag("Hand"))
        {
            Debug.Log($"[VitalsTrigger] Trigger '{triggerType}' activated
              by {other.name}");

            switch (triggerType.ToUpper())
            {
                case "PULSE":
                    serialSender.SendPulse();
                    break;
                case "HEART":
                    serialSender.SendHeart();
                    break;
                case "FAINTPULSE":
                    serialSender.SendFaintPulse();
                    break;
                case "FAINTHEART":
                    serialSender.SendFaintHeart();
                    break;
                default:
                    Debug.LogWarning($"[VitalsTrigger] Unknown trigger
                      type: {triggerType}");
                    break;
            }
        }
    }
}

```

Appendix 2: SerialTest script

```
using System.IO.Ports;
using UnityEngine;

public class SerialTest : MonoBehaviour
{
    private SerialPort serial;
    public string portName = "COM4";
    public int baudRate = 9600;

    void Start()
    {
        try
        {
            serial = new SerialPort(portName, baudRate);
            serial.Open();
            Debug.Log($"[SerialTest] Serial port {portName} opened at
              {baudRate} baud.");
        }
        catch (System.Exception ex)
        {
            Debug.LogError($"[SerialTest] Failed to open serial port
              {portName}: {ex.Message}");
        }
    }

    public void SendPulse() => SendSerialCommand("Pulse");
    public void SendHeart() => SendSerialCommand("Heart");
    public void SendFaintPulse() => SendSerialCommand("FaintPulse");
    public void SendFaintHeart() => SendSerialCommand("FaintHeart");

    private void SendSerialCommand(string message)
    {
        if (serial != null && serial.IsOpen)
        {
            serial.WriteLine(message);
            Debug.Log($"[SerialTest] Sent: {message}");
        }
        else
        {
            Debug.LogWarning($"[SerialTest] Tried to send '{message}', but
              serial port is not open.");
        }
    }

    void OnApplicationQuit()
    {
        if (serial != null && serial.IsOpen)
        {
            serial.Close();
```

```
47              Debug.Log("[SerialTest] Serial port closed.");
48          }
49      }
50  }
51
```

Appendix 3: AnswerRevealUI script

```
1  using UnityEngine;
2  using TMPro;
3
4  public class AnswerRevealUI : MonoBehaviour
5  {
6      public TMP_Text answerText1;
7      public TMP_Text answerText2;
8
9      public void RevealPatient1Status()
10     {
11         answerText1.text = "Patient 1 is stable and does NOT require CPR.";
12     }
13
14     public void RevealPatient2Status()
15     {
16         answerText2.text = "Patient 2 requires CPR and IMMEDIATE medical
             attention.";
17     }
18 }
19
```

Appendix 4: Arduino script

```
#include <Wire.h>

#define TCAADDR 0x77  // I2C address of the multiplexer
#define PWM13   OCR4A
#define ModeReg 0x01

#define CHANNEL_MOTOR1 0  // Channel 0 for Motor 1 (Heart)
#define CHANNEL_MOTOR2 2  // Channel 2 for Motor 2 (Pulse)

byte DRV1 = 0x5A;  // DRV2605 address for both motors
byte DRV2 = 0x5A;

// === Multiplexer selector ===
void tcaSelect(uint8_t channel) {
  if (channel > 7) return;
  Wire.beginTransmission(TCAADDR);
  Wire.write(1 << channel);
  Wire.endTransmission();
  delay(10);
}

// === DRV2605 motor initializer ===
void initializeMotor(byte drvAddr) {
  Wire.beginTransmission(drvAddr);
  Wire.write(ModeReg); Wire.write(0x00); Wire.endTransmission();  //
    Clear standby

  Wire.beginTransmission(drvAddr);
  Wire.write(0x1D); Wire.write(0xA8); Wire.endTransmission();     // Set
    RTP unsigned

  Wire.beginTransmission(drvAddr);
  Wire.write(0x03); Wire.write(0x02); Wire.endTransmission();     //
    Library B

  Wire.beginTransmission(drvAddr);
  Wire.write(0x17); Wire.write(0xFF); Wire.endTransmission();     // Full
    scale

  Wire.beginTransmission(drvAddr);
  Wire.write(ModeReg); Wire.write(0x03); Wire.endTransmission();  // PWM
    mode

  delay(100);
}

// === Setup ===
void setup() {
  Serial.begin(115200);
```

```
  Wire.begin();

  // Init both motors through the multiplexer
  Serial.println("Initializing Motor 1 (Heart)...");
  tcaSelect(CHANNEL_MOTOR1);
  initializeMotor(DRV1);

  Serial.println("Initializing Motor 2 (Pulse)...");
  tcaSelect(CHANNEL_MOTOR2);
  initializeMotor(DRV2);
}

// === Loop: handle Unity serial input ===
void loop() {
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    input.trim();

    if (input == "Heart") {
      Serial.println("❤️ Received HEART → Motor 1");
      tcaSelect(CHANNEL_MOTOR1);
      heartbeat();
    } else if (input == "Pulse") {
      Serial.println("💢 Received PULSE → Motor 2");
      tcaSelect(CHANNEL_MOTOR2);
      heartbeat();
    } else if (input == "FaintHeart") {
      Serial.println("❤️‍🩹 Received Faint HEART → Motor 1");
      tcaSelect(CHANNEL_MOTOR1);
      faintHeartbeat();
    } else if (input == "FaintPulse") {
      Serial.println("💫 Received Faint PULSE → Motor 2");
      tcaSelect(CHANNEL_MOTOR2);
      faintHeartbeat();
    }
  }
}

void faintHeartbeat() {
  Serial.println("Faint Heartbeat");

  for (int i = 0; i < 7; i++) {
    // "Lub" – barely perceptible primary beat
    pulse(0.025, 20);     // very weak and short
    pause(15);
    pulse(0.001, 2);      // soft rebound

    pause(350);           // longer pause before "dub"

```

```
    // "Dub" - weaker secondary beat
    pulse(0.012, 15);     // ultra-low intensity
    pause(10);
    pulse(0.001, 2);      // minimal echo pulse

    pause(1100);          // extended rest to mimic bradycardia
  }
}
//                                                                              ↵
  ============================================================================ ↵
  ========
// simulates a quickening heartbeat ("lub"-"dub")
void heartbeat() {
  Serial.println("□ Heartbeat");
  for (int i=0;i<10;i++){
    pulse(0.3, 45);        // "lub"
    pause(12);
    pulse(0.3, 3);          // short pulse to sharpen pulse haptic
    pause(250);

    pulse(0.1, 45);        // "dub"
    pause(12);
    pulse(0.3, 3);          // short pulse to sharpen pulse haptic
    pause(600);
  }
}


void pulse(double intensity, double milliseconds) {
  int minimumint = 140;
  int maximumint = 255;
  int pwmintensity = (intensity * (maximumint - minimumint)) + minimumint;
  standbyOffB();
  PWM13 = pwmintensity;
  usdelay(milliseconds);
  standbyOnB();

```