Enhancing Two-Handed Controller Interaction in Virtual Reality with Arm Tapping

Author: Minh Hoang NGUYEN

Student ID: 22103433

Supervisor: Dr Huawei TU

Department of Computer Science and Information Technology

La Trobe University
Victoria, Australia

October 2025

Enhancing Two-Handed Controller Interaction in Virtual Reality with Arm Tapping, © October 2025

Author:
Minh Hoang Nguyen

Supervisors:
Dr Huawei Tu

Institute:
La Trobe University, Victoria, Australia

CONTENTS

List of Figures . . . vi

List of Tables . . . x

Abstract . . . xi

Declaration of Authorship . . . xiii

Acknowledgments . . . xv

1 INTRODUCTION . . . 1
- 1.1 Background . . . 1
- 1.2 Motivation . . . 2
- 1.3 Summary of Completed Work . . . 3
- 1.4 Empirical Evidence from Completed Work . . . 4

2 LITERATURE REVIEW . . . 7
- 2.1 Two-Handed VR Controllers Interactions . . . 7
  - 2.1.1 Built-In Controller Components . . . 7
  - 2.1.2 External Hardware Augmentations . . . 9
  - 2.1.3 On-Body Interaction . . . 10
- 2.2 On-Arm Interaction . . . 13
  - 2.2.1 Traditional Sensor-Based On-Arm Interaction . . . 13
  - 2.2.2 Electrical Signal-Based On-Arm Interaction . . . 15
  - 2.2.3 Photo-Reflective Sensing for On-Body Interaction . . . 17
  - 2.2.4 Wearable skin sensors . . . 18
- 2.3 Bimanual Action Theory . . . 20
  - 2.3.1 Asymmetrical Bimanual Actions . . . 20
  - 2.3.2 Symmetric Bimanual Activities . . . 21
- 2.4 Conclusion . . . 23

3 EXPERIMENT ONE: DATA COLLECTION AND MODEL TRAINING . . . 25
- 3.1 Data Collection . . . 25
  - 3.1.1 Apparatus . . . 25
  - 3.1.2 Participants . . . 26
  - 3.1.3 Experiment Procedure . . . 26
  - 3.1.4 Data Logging and Feature Schema . . . 28
  - 3.1.5 Labeling . . . 29
- 3.2 Tapping Identification . . . 30
  - 3.2.1 Sliding Windows . . . 30
  - 3.2.2 Candidate Classifiers . . . 32
  - 3.2.3 Tapping Detector . . . 32
  - 3.2.4 Tapping Location Detector . . . 33
  - 3.2.5 Hand Use Detector . . . 34
  - 3.2.6 Force Level Detector . . . 34
- 3.3 Result Analysis and Discussion . . . 35
  - 3.3.1 Tapping Detection . . . 35

3.3.2 Tapping Location Detection . . . 36
3.3.3 Hand Use Detection . . . 37
3.3.4 Force Detection . . . 37
3.4 Conclusion . . . 38

4 EXPERIMENT TWO: CLASSIFICATION OF TAPPING AND NON-TAPPING ACTIONS 41
4.1 Data Collection . . . 41
4.1.1 Task and Procedure . . . 41
4.2 Results Analysis and Discussion . . . 42
4.3 Conclusion . . . 43

5 EXPERIMENT THREE: APPLICATION-LEVEL INTEGRATION OF TAPPING IN VR 45
5.1 Apparatus . . . 45
5.2 Participants . . . 46
5.3 Application Implementation in Unity . . . 46
5.3.1 System Overview and Data Flow . . . 46
5.3.2 Telemetry, Data Schema, and Timing . . . 46
5.3.3 Streaming, Inference, and Reliability . . . 47
5.3.4 Binding Tapping to Game Features . . . 48
5.3.5 Game Scenario . . . 49
5.4 Experiment Procedure . . . 50
5.4.1 Interaction Mapping (Taps → Game Actions) . . . 50
5.4.2 Participant Briefing and Practice . . . 51
5.4.3 Runtime Detection Pipeline . . . 52
5.4.4 Gameplay Task and Rules . . . 52
5.5 Results Analysis and Discussion . . . 53
5.5.1 Overall Impression and Learnability . . . 53
5.5.2 Ease of Use and Control . . . 53
5.5.3 Engagement and Creativity . . . 53
5.5.4 Implications and Next Steps . . . 54

6 GENERAL DISCUSSION . . . 55
6.1 Identification of Tapping with VR Controllers . . . 55
6.2 Tapping Performance with VR Controllers . . . 56
6.2.1 Hand Roles and Performance . . . 56
6.2.2 Tap Position . . . 56
6.2.3 Robsutness to non-tapping actions . . . 56
6.2.4 Real-time Consideration . . . 57
6.2.5 Summary . . . 57
6.3 Interactive Experience with Tapping . . . 57
6.3.1 Learnability and Onboarding . . . 57
6.3.2 Predictability and Control . . . 57
6.3.3 Ease of Use . . . 58
6.3.4 Engagement and Creativity . . . 58
6.3.5 Summary . . . 58
6.4 User Experience and Workload . . . 58
6.4.1 Subjective Experience . . . 59
6.4.2 Implications . . . 59
6.5 Design Implications . . . 59
6.5.1 Recognition and Mapping . . . 59
6.5.2 Modeling and Data . . . 60
6.5.3 Feedback and UX . . . 60

6.6 Limitations and Future Work 60
6.6.1 Data and Generalisation 60
6.6.2 Sensing and Features 61
6.6.3 Model and Runtime 61
6.6.4 UX and Evaluation 61
6.6.5 Summary 61

7 CONCLUSION 63

BIBLIOGRAPHY 64

LIST OF FIGURES

Figure 1.1 Examples of commercially available VR headsets 1
Figure 1.2 Illustration of the Arm Tapping technique. (a) Tapping right arm with the controller held by left hand; (b) Tapping left arm with the controller held by right hand. 3
Figure 2.1 System concept of touchpad-based interaction in AR environments. (a) Overall system appearance combining an HMD and touchpad. (b) Video-see-through AR created using a monocular head-mounted camera (HMC) attached to the HMD, with AR objects superimposed on real-time camera images. (c) Robust touchpad tracking achieved by attaching multiple visual markers, supporting accurate manipulation even under occlusion. (d) Miniature AR objects displayed on the touchpad enable quick and precise manipulation of distant AR objects. A pointer and scope assist in object selection and navigation [33]. 8
Figure 2.2 Selection-based text entry methods in VR, including Head Pointing (HP), Controller Pointing (CP), Controller Tapping (CT), Freehand (FH), Discrete Cursor (DC), and Continuous Cursor (CC) [31]. 8
Figure 2.3 Illustration of variable grip adjustment: (A) a firm grip holds the glass upright; (B) loosening the trigger allows the glass to swing downward under gravity while the controller position remains unchanged [7]. 9
Figure 2.4 Interaction gestures for distant manipulation using the trigger and joystick. Users can translate, rotate, and uniformly scale objects while holding the trigger; transformations can be combined or locked individually for precise control [32]. 9
Figure 2.5 Two users simultaneously interact with a 3D object while sharing the same physical space regardless of the interface. This study investigates the effectiveness of cooperative object manipulation in asymmetric Mixed Reality (MR) environments, where one user interacts in immersive Virtual Reality (VR) and the other in mobile Augmented Reality (AR), as shown in (c). Comparisons are also made with symmetric setups, where both participants use the same interface, either AR (a) or VR (b). The techniques support manipulations beyond arm's reach, where rays drawn from the hands to the object represent selection, and icons indicate the performed actions. Dashed outlines visualize user avatars in the virtual environment, and all virtual elements are synchronized among participants [32]. . 10
Figure 2.6 Three Haptic Links prototypes attached to HTC VIVE controllers: (Left to Right) Layer-Hinge, Chain, and Ratchet-Hinge. These prototypes demonstrate variable stiffness mechanisms for bimanual haptic feedback [17]. 10
Figure 2.7 GamesBond creates the illusion of physical connection between controllers using grip deformation. It simulates rigid, soft, and dynamic bonds for immersive VR experiences [34]. 11

Figure 2.8 System overview and hardware configuration of the SPIDAR-G&G system. The system consists of two 6-DOF string-based haptic devices connected to a PC via USB controllers for real-time feedback (left). The developed prototype (right) positions the devices approximately 40 cm apart, separated by a 17-inch LCD display where virtual objects are projected. Users interact by grasping dedicated grips attached to each haptic device, enabling precise bimanual manipulation within the defined workspace. The setup is designed to ensure a comfortable working posture and effective coordination for bimanual tasks [35]. 11
Figure 2.9 Forearm-based virtual menu interaction using Leap Motion. This system supports four intuitive gestures: (a) *Touch* for menu selection, (b) *Drag* for scrolling content, (c) *Slide* for switching menu layers by sliding the hand along the forearm, and (d) *Rotation* for changing menu modes or adjusting parameters [36]. 12
Figure 2.10 Tap-Tap Menu concept illustrating interaction on various body parts. This approach provides tactile feedback and reduces display area limitations [37]. 12
Figure 2.11 Blended interaction system using inside-out tracking of Oculus Quest 2. Combines on-body touch interactions with mid-air gestures for enhanced immersion. (A) A user is scaling a vase towards a specific direction by performing thumb-on-finger gestures and mid-air movements. (B) A user is rotating a blue planet around and/or moving it towards a red planet by combining bimanual thumb-on-finger gestures with mid-air input. (C) Finger-on-arm gestures and mid-air input enable users to translate a fox with one degree of freedom. (D) Users can teleport to different locations by manipulating an on-body minimap display [38]. 13
Figure 2.12 OmniTouch wearable system projects interactive interfaces onto everyday surfaces, including the forearm. It enables multi-touch input and finger tracking using depth-sensing technology [39]. 13
Figure 2.13 Armura system combining real-time sensing and projection to enable complex on-body interactions, including cursor control and multitouch gestures [24]. 14
Figure 2.14 Skinput system demonstration, where an armband augmented with a pico-projector allows interactive elements to be rendered directly on the skin. This system transforms the arm and hand into interactive input surfaces using projected virtual interfaces and bio-acoustic sensing [25]. 14
Figure 2.15 A user utilizing PUB to bind music controls to the forearm through a mobile device, enabling eyes-free control of the music player via an ultrasonic sensor attached to the wrist [27]. 15
Figure 2.16 The properties of ultrasound signal propagation along the skin. A transducer placed perpendicular to the skin results in surface wave propagation [40]. 15
Figure 2.17 SkinTrack system schematic. A finger's location on the skin produces varying phase differences observed by electrode pairs, enabling accurate touch localization. Note that the waveform "ripple" is illustrative and not drawn to scale [41]. 16

Figure 2.18 Dominant RF signal paths before (left) and after touch (middle). The two electrodes on the headset receiver capacitively couple to P2 for touch detection (right) [42]. 17
Figure 2.19 SenSkin system: the proposed input interface using skin deformation, enabling recognition of both normal and tangential forces applied to the forearm surface [43]. 18
Figure 2.20 SkinWatch hardware prototype: backside of the device. Twelve sensors are installed on the backside of the watch, enabling accurate gesture detection around the wrist area [44]. 18
Figure 2.21 The touch-enabled textile used for creating the GestureSleeve. Two layers of striped electrodes are placed perpendicular to each other with a force-sensitive fabric sandwiched in between. The processing electronics is shown in the top right [45]. 19
Figure 2.22 Two hands can create a book interaction metaphor. “Pages” can be turned by flipping the appropriate hand, which transitions the interface. Time advances from left to right. This interaction also visually demonstrates the application of KCM principles, where the non-dominant hand provides a spatial reference and the dominant hand performs the dynamic flipping action [24]. 21
Figure 2.23 Experimental setup for the symmetric bimanual curve manipulation study. The experimental setup allowed participants to control curve parameters using a stylus in their dominant hand, while simultaneously adjusting supporting variables with a small puck held in the non-dominant hand. This configuration leveraged the Wacom tablet’s capability to sense both input devices simultaneously, facilitating seamless bimanual interaction. [51]. 22
Figure 2.24 symSpline interface concept. symSpline utilizes two cursors to control the ends of the tangent at an edit point. By translating the entire tangent, users can adjust both the position of the edit point and the curve’s shape in a continuous and intuitive manner [52]. 23
Figure 3.1 Experiment 1 trial timeline and hand poses. Panels (a–e): (a) baseline posture (both controllers at chest height); (b) start cue—press X (Quest) or circular disc (HTC); (c) execute the instructed arm tap (zone and force on screen); (d) return to baseline; (e) end cue—press Y (Quest) or trigger (HTC). 27
Figure 3.2 Active-hand acceleration around a single tap. The red markers indicate the dominant peak and trough that characterize the impact. The label interval is taken from the point where the signal departs the baseline before the peak to the point where it returns to baseline after the trough; low-amplitude oscillations on both sides reflect normal hand motion outside the tap. 30
Figure 3.3 Window labeling with size $= 3$, stride $= 1$. The red window contains only one tap frame (majority = 0) and is labeled *negative*; the green window contains two tap frames (majority = 1) and is labeled *positive*. Using window blocks provides temporal context for training and mitigates boundary jitter from frame-wise labels. 31
Figure 4.1 Seven representative non-tapping controller actions used to challenge the tap detector. They cover common VR interactions with varied directions, ranges, and bilateral roles. 42

Figure 5.1 Unity device view (OpenXR) showing headset/controller signals used for capture: pose, linear/angular velocity, button states, and tracking. 47
Figure 5.2 Firebase batch container. Each batch spans one grip-press→release episode and includes `samples` plus `sessionId/tappingHand`. 48
Figure 5.3 Example per-frame entry for `LeftController`: acceleration/velocity magnitudes, 3D position, and `timestampMs`. 48
Figure 5.4 Example per-frame entry for `RightController`: acceleration/velocity magnitudes, 3D position, and `timestampMs`. 49
Figure 5.5 Participant posture used for runtime capture (Sec. 5.3.2). The left hand is positioned to perform a tap on the right hand. 49
Figure 5.6 Firebase prediction schema. The Python listener writes the most recent decision to `/predictions/latest` (fields: `tapping`, `hand`, `position`, `ts`), which Unity reads to trigger actions. Historical predictions are stored under `/predictions/{timestamp}` for analysis. . 50
Figure 5.7 Unity inspector for `TappingInput`. Detected tap events (e.g., left/right wrist, left elbow) are bound to game actions such as `MiniMap.ToggleMiniMap`, `VRWeaponSwitcher.NextWeapon`, and `PickUpManager.ActivateMostRecentHeld/ResetCycle`. 51
Figure 5.8 In-game evidence of tap-driven utilities: healing, wrist mini-map, and combat interactions. 52
Figure 5.9 Runtime data flow. Unity streams sensor data to Firebase; a local ML process subscribes, performs windowing and normalisation, runs the detector and (if positive) the locator, then writes prediction results back. Unity listens for these events to drive gameplay logic. . . . 52
Figure 5.10 User–study ratings (Mean $\pm$ SD; 1–10 scale, n=6) for all metrics: Overall impression, Learnability, Effort/Efficiency, Control/Predictability, Exciting/Motivating, Creative/Interesting. 53

LIST OF TABLES

Table 1.1 Experiment 2: Tap detector on non-tapping actions (held-out sequences). 5
Table 3.1 Features used in the CSV files (Experiment 1). 29
Table 3.2 Detector performance on the test set (window size = 3, stride = 1). . 36
Table 3.3 Tap location (left-hand taps): test-set metrics. 36
Table 3.4 Tap location (right-hand taps): test-set metrics. 37
Table 3.5 Hand-use classifier performance on the test set (window size = 3, stride = 1). 37
Table 3.6 Force detection (left-hand, wrist taps): test-set metrics. 38
Table 4.1 Experiment 2 — Performance by non-tapping action type (tapping vs. non-tapping). 43

ABSTRACT

This thesis investigates a controller-only method for recognising arm tapping in virtual reality (VR). Grounded in asymmetric bimanual interaction (Guiard's kinematic chain), the non-dominant forearm serves as an always-available target surface while the dominant hand taps with a standard VR controller. We operationalise this as a lightweight, staged recogniser: first detect a tap event from short spans of commodity controller kinematics, then infer the acting hand, and finally classify the tap position (wrist, mid-forearm, elbow)—all without extra hardware or heavy models.

We collected synchronized signals from both controllers, labelled tap moments via an acceleration peak–trough procedure, and trained subject-separated models with min–max normalised features. Across offline evaluations, the tap detector achieved near-perfect performance (overall accuracy 100%, macro F1 0.99). The hand-use classifier cleanly separated acting versus target arms (100% accuracy). Tap-position classification was high for both sides (0.97 for left-hand taps; 0.94 for right-hand taps), with residual confusion concentrated at mid-forearm. Robustness tests on seven common non-tapping actions sustained perfect precision (1.000) and high accuracy/recall ($\sim$0.97), indicating strong resistance to false triggers.

We integrated the models in a real-time Unity–Firebase–Python loop and bound taps to practical utilities (mini-map, weapon swap, heal) in a shooter scenario. With six participants, post-task ratings indicated that tapping was learnable and appealing (overall impression median 8; learnability 8.5) and consistently engaging (medians 9/9), while perceived effort (median 6) and control (median 7) revealed occasional friction under fast gameplay. Runtime logs showed perfect agreement between detected acting hand and Unity controller metadata.

The primary deployment gap stems from data mismatch—scripted training poses versus varied gameplay postures, speeds, and contexts—rather than model capacity. We outline straightforward remedies: broaden in-situ data, add light augmentation or brief per-user calibration, and reinforce feedback/timing guards (ready/detected indicator, haptic tick, short cooldown). Overall, accurate tap detection, perfect acting-hand identification, and strong position classification are achievable today with off-the-shelf controllers and interpretable, low-latency models, keeping the technique portable across devices and applications.

STATEMENT OF AUTHORSHIP

STATEMENT OF AUTHORSHIP AND AUTHORITY TO SUBMIT

I, Minh Hoang Nguyen, born on April 14, 2001 in VietNam, declare that the thesis titled *Enhancing Two-Handed Controller Interaction in Virtual Reality with Arm Tapping* and the work presented in it are my own. This work was undertaken primarily during my candidature for a research degree at La Trobe University. It has not been submitted, in whole or in part, for any other degree or diploma at any tertiary institution.

I further confirm that my Principal Supervisor has authorised the submission of this thesis. The duly signed *Authority to Submit* form accompanies this document as required by the University.



Minh Hoang Nguyen October 29, 2025

ACKNOWLEDGMENTS

This thesis marks the culmination of a challenging yet incredibly rewarding journey, one that would not have been possible without the unwavering support and encouragement of many wonderful individuals. To all the friends, colleagues, and mentors I have had the privilege to meet along the way—thank you. Your kindness, insights, and shared moments made this experience memorable and meaningful.

First and foremost, I would like to express my deepest gratitude to my supervisor, **Dr Huawei Tu**, for their invaluable guidance, constructive feedback, and constant encouragement throughout my research journey. His insightful perspectives and tireless dedication have played a crucial role in shaping not only this thesis but also my approach to academic inquiry. I will always remember our engaging discussions that sparked new ideas and kept me motivated during difficult times.

I am also sincerely thankful to **Prof. Eric Pardede**, for his thoughtful advice, detailed reviews, and invaluable mentorship. His dedication and willingness to provide constructive feedback, especially in reviewing the thesis structure and ensuring its overall coherence, have greatly contributed to the quality of this work.

I extend my thanks to all members of the *Lab for Immersive Interaction Design*, whose collaborative spirit and open discussions provided valuable insights and fresh perspectives on my work. A special thanks goes to **Jianqiao Zhang** for helping me refine my writing and improve the clarity of my ideas.

Finally, I would like to express my heartfelt gratitude to my family for their unwavering support and encouragement throughout this journey. Your patience, understanding, and belief in me gave me the strength to overcome challenges and stay focused on my goals. Without your love and support, this thesis would not have been possible.

This thesis stands as a testament not only to my academic endeavors but also to the collective support of those who believed in me. For that, I am forever grateful.

1 INTRODUCTION

1.1 BACKGROUND

In contemporary virtual reality (VR) systems, head-mounted displays are typically paired with two handheld controllers that serve as primary input devices for user interaction within virtual environments. Prominent examples include HTC Vive [1], Meta Quest [2], and Sony PlayStation VR [3] (Fig. 1.1), all of which feature standardized controllers equipped with a range of input modalities. These controllers generally include built-in motion sensors such as accelerometers [4] and gyroscopes [5] for tracking hand movement and orientation in 3D space, as well as physical input components like buttons, triggers, grip sensors, and thumbsticks (e.g., [6–8]). Such input schemes support both discrete and continuous interactions, enabling a wide variety of VR tasks ranging from object manipulation and navigation to menu selection and gesture-based commands. The integration of these components forms the baseline hardware for many VR interaction techniques proposed in recent research.



(a) Meta Quest VR [9]



(b) HTC Vive VR [10]



(c) Sony PlayStation VR [11]

Figure 1.1: Examples of commercially available VR headsets

Building on these capabilities, VR controllers play a vital role in enabling intuitive and context-sensitive interaction across a range of virtual reality applications. In gaming, they allow users to aim, grab, throw, or operate tools with precision [12]. In training simulations—such as medical, military, or industrial environments—controllers replicate hand-based tasks like surgery, equipment handling, or assembly procedures [13, 14]. Educational VR apps further leverage these inputs to let learners explore 3D content by rotating, scaling, or selecting virtual objects [15]. Ultimately, these controllers serve as the user's virtual hands, translating physical actions into digital outcomes to support immersion, realism, and effective task performance.

1.2 MOTIVATION

Designing intuitive and efficient interaction methods remains a critical focus in the development of virtual reality (VR) systems and has been extensively explored in prior research. Existing studies broadly categorize interaction techniques into three primary approaches: enhancing controllers with auxiliary hardware, utilizing built-in controller components, and implementing on-body interaction designs.

Enhancing controllers with auxiliary hardware is the first approach, which involves integrating additional sensors, actuators, or mechanical structures into standard VR controllers (e.g., [16–18]). These augmentations are designed to provide richer haptic feedback, simulate physical forces, or deliver nuanced tactile sensations. By bridging the gap between virtual actions and natural physical interactions, such enhancements significantly improve realism, immersion, and the overall user experience.

The second approach focuses on leveraging intrinsic components of existing VR controllers, such as touchpads [19], buttons [20], and thumbsticks [8]. This line of research aims to maximize the potential of built-in hardware through innovative interaction techniques. For instance, touchpads have been adapted to support advanced pointing methods that enhance target selection accuracy, especially for distant or small objects. Buttons have been assessed for their effectiveness and usability in a range of VR tasks, such as selection, manipulation, and control within immersive analytics environments. Similarly, thumbsticks have been reimagined for more complex inputs, such as circular keyboard layouts for efficient text entry, improving both speed and learnability even for novice users. These improvements demonstrate that refining the interaction logic of existing controller elements can yield significant gains in usability and interaction quality without requiring external hardware additions.

The third approach involves developing on-body interaction techniques (e.g., [21–23]) that utilize the user's body as an interactive surface and spatial reference. These techniques enhance the capabilities of handheld controllers by extending the input space beyond the physical device itself. By anchoring virtual elements—such as menus, commands, or gesture triggers—to specific body parts (e.g., arms, hands, or torso), users can perform interactions more naturally and intuitively. On-body techniques also leverage kinesthetic memory and spatial awareness, improving gesture recall, input precision, and overall interaction efficiency. Especially in bimanual or asymmetric interactions, such designs offer enhanced comfort and task performance by aligning virtual tasks with familiar physical references on the user's body.

This thesis specifically focuses on on-body interactions, wherein the human body serves as an input/output medium for virtual interaction. Employing the body in this capacity offers several advantages in human-computer interaction contexts. First, the body acts as a readily available and always-accessible interface surface [24, 25], significantly enhancing usability in VR applications. Second, on-body interfaces naturally support eye-free targeting mechanisms [26–28], allowing users to interact precisely and intuitively without needing to visually locate or aim at the target area. Lastly, on-body interaction methods

inherently provide rich haptic feedback [24] due to direct physical contact, thereby enhancing the immersive experience and user engagement.

A popular way to design on-body interaction is tapping on an arm position to trigger a command [23]. Previous studies generally relied on fingers or hands [23, 29] to perform on-arm interactions with the need to use auxiliary devices to track the interactions (e.g., a Vicon motion tracking system in [23] or visual trackers from a VR headset [29]). However, when two hands are holding controllers, it is not natural and convenient to carry out interactions on the forearm with fingers or hands; instead, the use of VR controllers could be a more practical and straightforward manner. Further, using auxiliary hardware limits the applicability of on-arm interaction techniques. As an alternative, the sensors provided by VR controllers (e.g., accelerometers and gyroscopes) may provide the ability to detect on-arm interaction actions, which can leverage their applicability.

Inspired by previous works [27, 29], this study proposes arm tapping with two-handed controllers to enhance VR interaction. To perform the action, users employ a controller with one hand to tap on the forearm of the other hand holding another controller (Fig. 1.2). This action is capable of invoking commands for VR interaction in a situation where two hands are holding controllers.

1.3 SUMMARY OF COMPLETED WORK

This thesis progressed through four concrete stages spanning data collection, model development, validation, and in-game evaluation.

1. **Sensor data collection.** We implemented a Unity-based logger that records both hand-controller streams to CSV. Participants performed a predefined set of *tapping* gestures (as illustrated in Fig. 1.2) and a matched set of *non-tapping* movements. The non-tapping set serves as negative samples to assess robustness against false activations. Each trial includes synchronized timestamps to align events and motion windows.

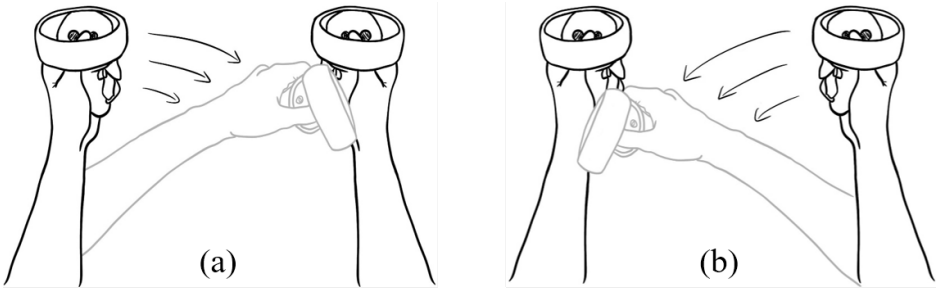


Figure 1.2: Illustration of the Arm Tapping technique. (a) Tapping right arm with the controller held by left hand; (b) Tapping left arm with the controller held by right hand.

2. **Model development.** Using the collected dataset, we trained three supervised models: (i) a *tap detection* model (binary: tap vs. non-tap); (ii) a *left-hand tap location* model (multi-class over target regions on the left arm); and (iii) a *right-hand tap location*

model (multi-class over target regions on the right arm). We also explored a *tap-force estimation* model; however, limited sensor fidelity and insufficient force-related signal prevented obtaining a reliable regressor. This limitation informs our future hardware/data strategy.

3. **Robustness checks against non-tapping actions.** We evaluated the tap-detection model specifically on the non-tapping set to quantify *false positives*. This analysis estimates practical reliability in real gameplay, where incidental controller motion is common.

4. **Real-time VR integration and user study.** We built a Unity VR zombie-shooter prototype and an end-to-end inference loop using Firebase as a lightweight message bus: Unity streams sensor features $\rightarrow$ the model produces detections $\rightarrow$ detections are written back to Firebase $\rightarrow$ Unity consumes detections to trigger in-game actions (e.g., weapon swap, minimap). We conducted a formative user study ($n = 6$), in which participants used tapping interactions during gameplay and then completed a short questionnaire on usability and experience. Collected telemetry and subjective responses provide initial evidence for feasibility and areas for refinement.

1.4 EMPIRICAL EVIDENCE FROM COMPLETED WORK

This section reports measurable, evidence-based outcomes from the completed phases of the project. We summarize end-to-end system behavior (including latency), recognition performance, robustness to incidental motion, and initial user-centered evidence obtained in a formative gameplay study. Each claim is tied to a verifiable metric and will be cross-referenced to detailed methods and analyses in later chapters.

1. **Real-time detection using controller-only sensing.** We implemented a Unity (C#) pipeline that ingests raw VR controller streams and performs on-arm tap detection without extra hardware. Measured end-to-end median latency is **300 ms** [IQR: 750–1300 ms], computed from stimulus onset in Unity to detection availability for gameplay logic. Latency instrumentation uses synchronized timestamps across the logging layer and detection callbacks, and results will be reported with distributional summaries and *[95% CI]*.

2. **Recognition performance on tapping, hand-role, and tap-location.** On the Experiment 2 dataset, the *tapping detector* achieves **Accuracy 0.999**, **AUC 0.989**, **Precision 0.992**, **Recall 0.989**, **F1 0.991**, and **TNR 0.992**. Tap-location is evaluated separately with the left/right models (reported alongside confusion profiles). In addition, on held-out windows from Experiment 1, the *hand-role (acting-hand) detector* attained **100%** accuracy, cleanly separating the moving arm from the stabilising arm. All metrics are accompanied by per-class breakdowns, macro/micro averages, and 95% confidence intervals.

3. **Robustness to incidental non-tapping actions.** To estimate false activations during realistic movement, we evaluated the tap detector on **seven** non-tapping actions: (1) swing arms forward; (2) place controllers on a table; (3) zoom; (4) collide; (5) tap touchpads; (6) tap in the air with a controller; (7) draw a circle. Macro performance across these actions is **Accuracy 0.971**, **Precision 1.000**, **Recall 0.971**, and **F1 0.985**, with a false-positive rate of ≈ 0. Per-action results are› reported in Table 1.1, alongside class-conditional error analyses and 95% confidence intervals.

Table 1.1: Experiment 2: Tap detector on non-tapping actions (held-out sequences).

Type	Accuracy	Precision	Recall	F1-score
1	0.9668	1.0000	0.9668	0.9831
2	0.9776	1.0000	0.9776	0.9887
3	0.9695	1.0000	0.9695	0.9845
4	0.9707	1.0000	0.9707	0.9851
5	0.9686	1.0000	0.9686	0.9840
6	0.9711	1.0000	0.9711	0.9853
7	0.9743	1.0000	0.9743	0.9870
Macro mean	**0.9712**	**1.0000**	**0.9712**	**0.9854**

4. **Usability and workload in VR gameplay.** In a formative study with **6** participants, tap-based interactions were evaluated in a zombie-shooter scenario. Post-task ratings indicated that the technique was *learnable and appealing*: **overall impression** median **8**, **learnability** median **8.5**. Perceived *workload* showed room for improvement under time pressure: **effort/efficiency** median **6** and **control/predictability** median **7**. Engagement was consistently high (**exciting/motivating** and **creative/interesting** medians **9**/**9**). Thematic feedback highlighted fast learnability, satisfying "game-like" contact, and occasional predictability issues during rapid movement—informing design implications such as clearer on-wrist feedback, brief haptic ticks, and short cooldowns (or a two-detections rule) during locomotion. Runtime logs further showed that the *acting-hand* detector aligned with Unity's controller metadata on triggered events, suggesting workload is driven more by boundary and tempo factors than by hand-assignment errors.

2

LITERATURE REVIEW

This literature review is organized into three key sections: two-handed VR controllers interactions, which explores current interaction techniques using dual controllers; on-arm Interaction, which examines how the forearm serves as an interactive medium in VR; and bimanual action theory, which provides theoretical foundations for understanding and designing effective two-handed interactions.

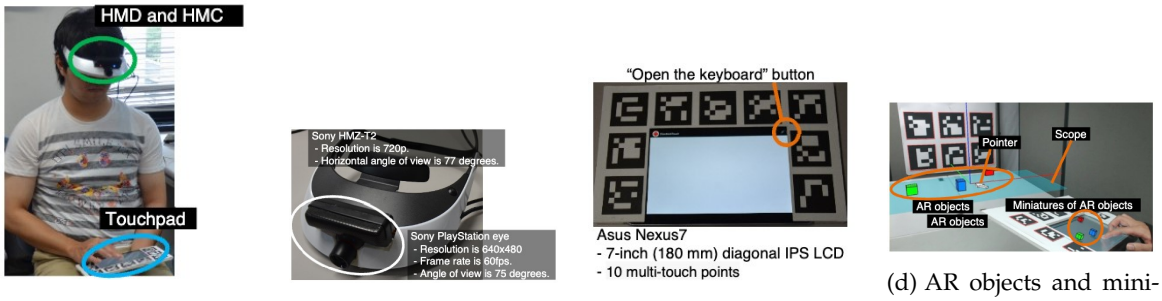
2.1 TWO-HANDED VR CONTROLLERS INTERACTIONS

Handheld controllers are widely utilized in virtual environments, primarily for tasks such as text input [30, 31] and object manipulation [32]. To improve interaction effectiveness, researchers have explored three main directions: leveraging built-in controller components, augmenting controllers with external devices, and incorporating on-body interactions.

2.1.1 *Built-In Controller Components*

VR controllers typically offer a variety of input components, including touchpads (e.g., [33]), buttons (e.g., [7, 31]), triggers (e.g., [7, 32]), and thumbsticks (e.g., [32]). These components play critical roles in enhancing interaction within virtual environments.

• **Touchpads**: Touchpads support both coarse and precise interactions by providing flexible input modalities. Nagashima et al. [33] explored the integration of touchpads with head-mounted displays (HMDs) to manipulate virtual objects using two input methods: *relative input*, which calculates pointer movement based on finger displacement for precise adjustments, and *absolute input*, which directly maps touch locations to virtual object positions, enabling faster selection but with reduced precision. Their study highlighted that the relative input method offers better control for fine interactions, while the absolute input method improves task completion speed, demonstrating the versatility of touchpads in immersive environments. As shown in (Fig. 2.1), the system setup integrates an HMD and touchpad for AR object manipulation. The touchpad, enhanced with visual markers, enables both precise and rapid interactions through miniature object representations and pointer-based control.
• **Buttons**: Buttons are primarily used to perform selection tasks, initiate actions, and confirm inputs. Speicher et al. [31] introduced *Controller Pointing* and *Controller Tapping* tech-



(a) System appearance (b) HMD and HMC. (c) Touchpad. (d) AR objects and miniatures of AR objects.

Figure 2.1: System concept of touchpad-based interaction in AR environments. (a) Overall system appearance combining an HMD and touchpad. (b) Video-see-through AR created using a monocular head-mounted camera (HMC) attached to the HMD, with AR objects superimposed on real-time camera images. (c) Robust touchpad tracking achieved by attaching multiple visual markers, supporting accurate manipulation even under occlusion. (d) Miniature AR objects displayed on the touchpad enable quick and precise manipulation of distant AR objects. A pointer and scope assist in object selection and navigation [33].

niques for text entry, using buttons for character selection on virtual keyboards (Fig. 2.2). Bonfert et al. [7] further introduced a variable grip interaction model, where users modulate grip strength using the trigger and grip buttons, allowing more realistic and flexible object manipulation (Fig. 2.3).

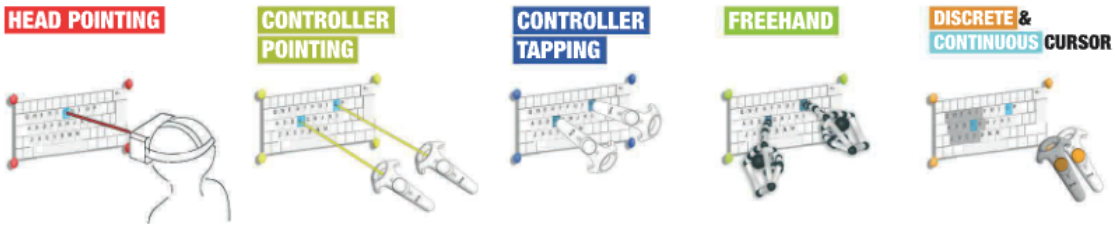


Figure 2.2: Selection-based text entry methods in VR, including Head Pointing (HP), Controller Pointing (CP), Controller Tapping (CT), Freehand (FH), Discrete Cursor (DC), and Continuous Cursor (CC) [31].

• **Triggers**: Triggers act as analog inputs, enabling users to control variable actions such as grabbing, manipulating, and applying force to virtual objects. Grandi et al. [32] introduced techniques using the trigger button for single-handed translations, rotations, and scaling of objects beyond arm's reach (Fig. 2.4). Bonfert et al. [7] further demonstrated how varying trigger pressure simulates loose or firm grips, enhancing realism in object handling.

• **Thumbsticks**: Thumbsticks are essential for continuous transformations such as navigation and object manipulation. Grandi et al. [32] demonstrated that thumbsticks allow users to intuitively adjust object scaling by moving them up or down while holding the trigger, supporting efficient and precise control. These techniques are further illustrated in various collaboration scenarios (Fig. 2.5), where both symmetric and asymmetric setups utilize thumbsticks to facilitate intuitive object manipulation and cooperative tasks in VR and AR environments.



Figure 2.3: Illustration of variable grip adjustment: (A) a firm grip holds the glass upright; (B) loosening the trigger allows the glass to swing downward under gravity while the controller position remains unchanged [7].

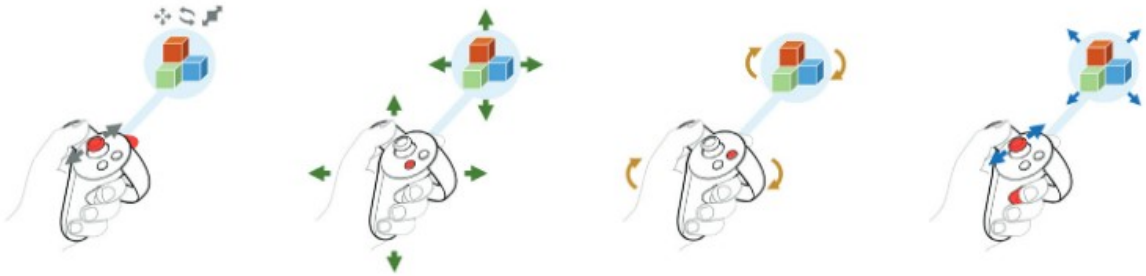
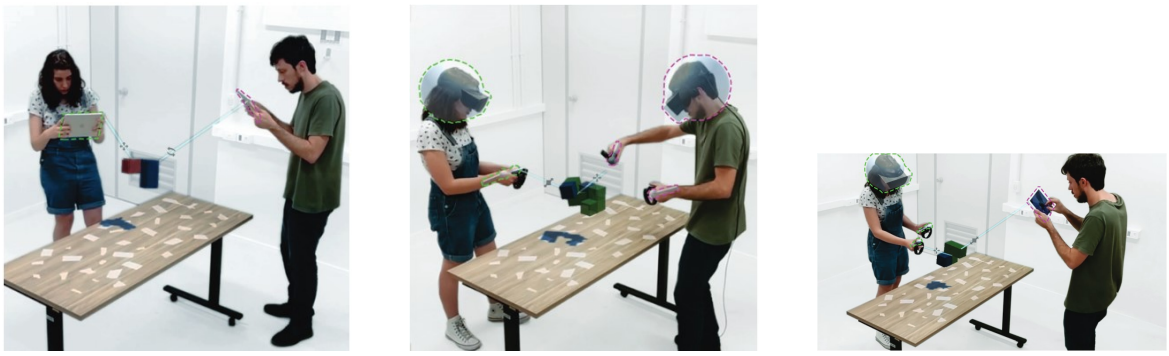


Figure 2.4: Interaction gestures for distant manipulation using the trigger and joystick. Users can translate, rotate, and uniformly scale objects while holding the trigger; transformations can be combined or locked individually for precise control [32].

2.1.2 *External Hardware Augmentations*

In addition to leveraging built-in controller components, researchers have explored external hardware augmentations to expand the physical capabilities and sensory feedback of VR controllers. These enhancements aim to provide richer haptic feedback, simulate realistic physical interactions, and improve user immersion during complex virtual tasks.

• **Haptic Links**, introduced by Strasnick et al. [17], employed electro-mechanically actuated links to dynamically adjust stiffness between two VR controllers. Three prototypes—*Chain*, *Layer-Hinge*, and *Ratchet-Hinge*—were developed to provide different stiffness control mechanisms suitable for various interaction scenarios (Fig. 2.6).
• **GamesBond**, proposed by Ryu et al. [34], creates the illusion of physically connected objects without requiring a physical tether. Instead, it utilizes grip deformation to simulate holding continuous virtual objects, such as ropes or sticks. GamesBond can simulate rigid, soft, and dynamic material properties, enhancing user immersion without constraining hand movement (Fig. 2.7).



(a) Symmetric AR (b) Symmetric VR (c) Asymmetric VR and AR

Figure 2.5: Two users simultaneously interact with a 3D object while sharing the same physical space regardless of the interface. This study investigates the effectiveness of cooperative object manipulation in asymmetric Mixed Reality (MR) environments, where one user interacts in immersive Virtual Reality (VR) and the other in mobile Augmented Reality (AR), as shown in (c). Comparisons are also made with symmetric setups, where both participants use the same interface, either AR (a) or VR (b). The techniques support manipulations beyond arm's reach, where rays drawn from the hands to the object represent selection, and icons indicate the performed actions. Dashed outlines visualize user avatars in the virtual environment, and all virtual elements are synchronized among participants [32].



Figure 2.6: Three Haptic Links prototypes attached to HTC VIVE controllers: (Left to Right) Layer-Hinge, Chain, and Ratchet-Hinge. These prototypes demonstrate variable stiffness mechanisms for bimanual haptic feedback [17].

• **SPIDAR-G&G**, developed by Murayama et al. [35], is a string-based 6-DOF haptic device designed for precise bimanual manipulation. It applies force feedback through tensioned strings, enabling life-like interactions with virtual objects (Fig. 2.8).

2.1.3 *On-Body Interaction*

Beyond leveraging built-in controller components and augmenting them with external devices, researchers have also explored using the body itself as an interactive surface to enhance two-handed VR controller experiences. This approach takes advantage of the body's natural accessibility and tactile feedback, enabling more intuitive, eyes-free, and spatially grounded interactions. In particular, the forearm and other body parts have been investigated as effective surfaces for supporting virtual interactions and reducing cognitive load.

The forearm consistently provides an available surface for contact and is always accessible for interaction. However, the key challenge in utilizing the forearm for interaction in virtual environments lies in accurately detecting its presence and precisely identifying con-

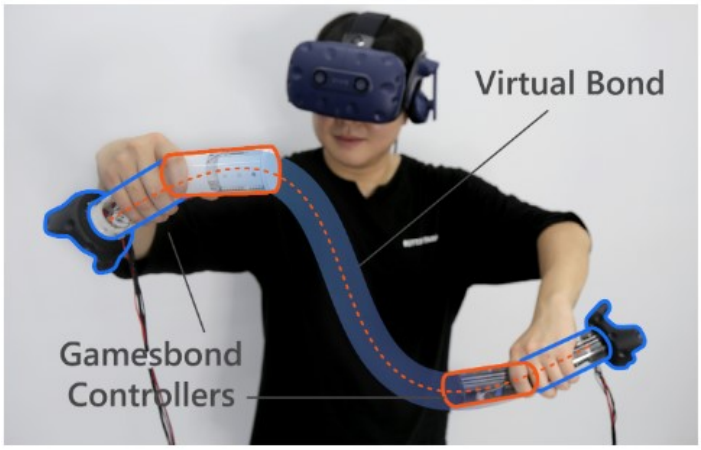


Figure 2.7: GamesBond creates the illusion of physical connection between controllers using grip deformation. It simulates rigid, soft, and dynamic bonds for immersive VR experiences [34].

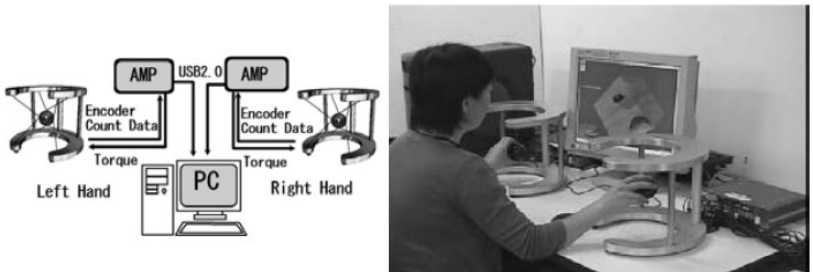


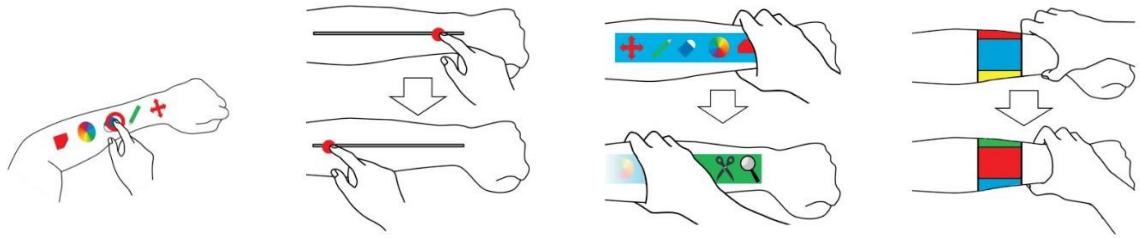
Figure 2.8: System overview and hardware configuration of the SPIDAR-G&G system. The system consists of two 6-DOF string-based haptic devices connected to a PC via USB controllers for real-time feedback (left). The developed prototype (right) positions the devices approximately 40 cm apart, separated by a 17-inch LCD display where virtual objects are projected. Users interact by grasping dedicated grips attached to each haptic device, enabling precise bimanual manipulation within the defined workspace. The setup is designed to ensure a comfortable working posture and effective coordination for bimanual tasks [35].

tact locations. This section reviews prior studies that have explored the forearm and other body parts as interactive surfaces in virtual reality (VR).

The current study utilizes computer vision systems to detect the position of the palm, fingers, and forearm. Azai et al. [36] proposed the use of Leap Motion, mounted in front of the VR headset, to recognize a variety of on-forearm interactions, including touch, drag, rotate, and slide gestures. They also developed a forearm-based menu system, allowing direct manipulation of virtual menus through physical touch on the user's own forearm. This approach offers intuitive feedback and reduces cognitive load by providing tactile sensation (Fig. 2.9).

In subsequent research, Azai et al. [37] introduced the Tap-Tap Menu, which extends on-body interaction to multiple body parts, including the abdomen and upper legs, in addition to the forearm. This design expands the available interaction area and allows users to intuitively control menus through kinesthetic movements and direct touch (Fig. 2.10).

Yu et al. [38] explored a blended interaction technique using the Oculus Quest 2's inside-out tracking cameras. Their system combines on-body interactions with mid-air gestures



(a) Touch Interaction (b) Drag Interaction (c) Slide Interaction (d) Rotation Interaction

Figure 2.9: Forearm-based virtual menu interaction using Leap Motion. This system supports four intuitive gestures: (a) *Touch* for menu selection, (b) *Drag* for scrolling content, (c) *Slide* for switching menu layers by sliding the hand along the forearm, and (d) *Rotation* for changing menu modes or adjusting parameters [36].

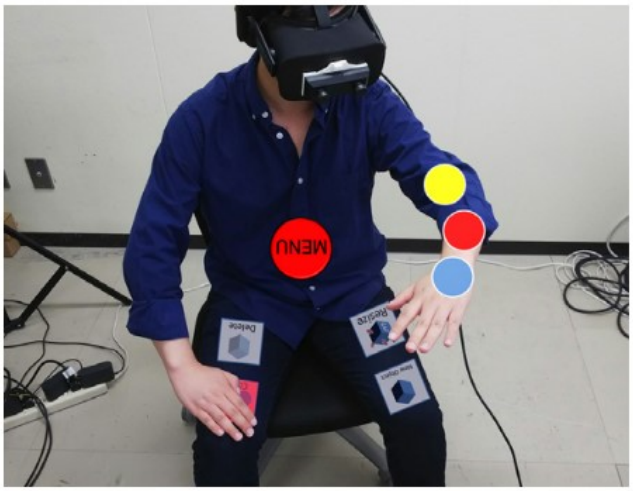


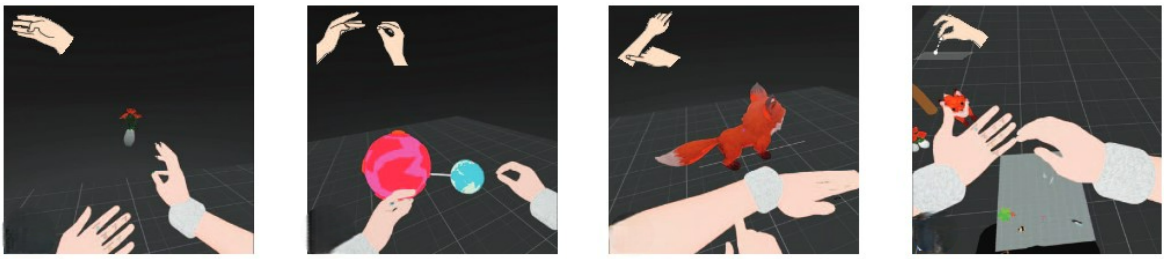
Figure 2.10: Tap-Tap Menu concept illustrating interaction on various body parts. This approach provides tactile feedback and reduces display area limitations [37].

to improve the realism and engagement of VR experiences. This approach allows seamless transitions between precise body-based controls and expressive mid-air gestures, providing a flexible interaction paradigm (Fig. 2.11).

Researchers have also explored the use of wearable projection systems. OmniTouch, developed by Harrison et al. [39], is a shoulder-mounted depth-sensing and projection system that transforms the forearm into a touch-sensitive interactive surface. The system provides multi-touch capabilities without requiring calibration or additional body instrumentation, enabling interactions similar to those on touchscreens (Fig. 2.12).

Further extending this concept, Harrison et al. [24] introduced Armura, a system that combines projection and sensing capabilities to support both graphical output and input directly on the user's body. Unlike earlier systems, Armura also investigated advanced interaction techniques such as cursor control and clutching mechanisms in on-body interaction contexts (Fig. 2.13).

These studies demonstrate that the human body, particularly the forearm, offers a promising platform for interaction in VR. Leveraging tactile feedback, kinesthetic awareness, and innovative sensing technologies can significantly enhance the usability and engagement of VR systems.



(a) Scaling (b) Rotating (c) Translateting (d) Teleporting

Figure 2.11: Blended interaction system using inside-out tracking of Oculus Quest 2. Combines on-body touch interactions with mid-air gestures for enhanced immersion. (A) A user is scaling a vase towards a specific direction by performing thumb-on-finger gestures and mid-air movements. (B) A user is rotating a blue planet around and/or moving it towards a red planet by combining bimanual thumb-on-finger gestures with mid-air input. (C) Finger-on-arm gestures and mid-air input enable users to translate a fox with one degree of freedom. (D) Users can teleport to different locations by manipulating an on-body minimap display [38].

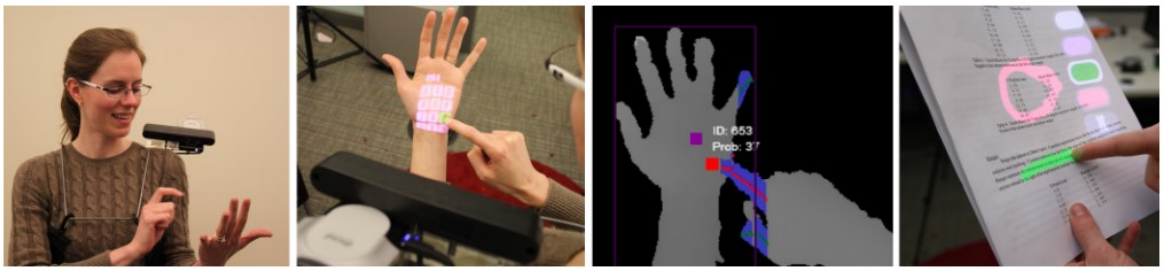


Figure 2.12: OmniTouch wearable system projects interactive interfaces onto everyday surfaces, including the forearm. It enables multi-touch input and finger tracking using depth-sensing technology [39].

2.2 ON-ARM INTERACTION

As a specific case of on-body interaction, on-arm interaction has attracted significant attention due to the forearm's natural accessibility and convenience for device control. Building upon this concept, researchers have developed a range of techniques to transform the arm into an interactive surface, creating intuitive, eyes-free, and always-available input spaces. Various sensing technologies have been explored to enable reliable on-arm interactions, including traditional sensors, electrical signal-based methods, photo-reflective sensing, and wearable skin sensors.

2.2.1 *Traditional Sensor-Based On-Arm Interaction*

A foundational approach for enabling on-arm interaction involves the use of traditional sensors to detect user input directly on the skin surface. One notable example is Skinput, developed by Harrison et al. [25], which leveraged the human body's natural bio-acoustic properties to transmit sound and treat the skin as an interactive input surface. Skinput employed a wearable armband equipped with a bio-acoustic sensor array to detect mechanical vibrations generated by finger taps on the arm and hand. These vibrations, transmit-

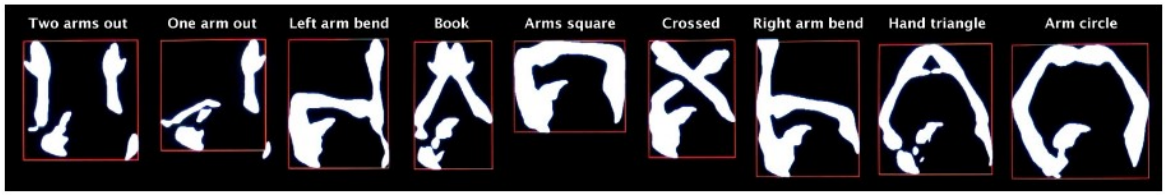


Figure 2.13: Armura system combining real-time sensing and projection to enable complex on-body interactions, including cursor control and multitouch gestures [24].

ted through the body's soft tissues and bones, were analyzed to determine tap locations. In addition to its innovative sensing method, Skinput demonstrated how virtual user interfaces can be projected directly onto the skin, allowing users to interact with projected buttons and menus without requiring handheld devices (Fig. 2.14). This approach supports input without requiring visual attention to external devices, enabling eyes-free and always-available interaction.



Figure 2.14: Skinput system demonstration, where an armband augmented with a pico-projector allows interactive elements to be rendered directly on the skin. This system transforms the arm and hand into interactive input surfaces using projected virtual interfaces and bio-acoustic sensing [25].

Lin et al. [27] introduced PUB (Point Upon Body), a system that enabled eyes-free interaction on the forearm through ultrasonic sensors positioned around the wrist. PUB allowed users to divide the forearm into multiple discrete zones and naturally associate each with different functions. It also provided haptic feedback through the skin, enhancing precision and reducing cognitive load. PUB was demonstrated in practical applications such as controlling a mobile music player and navigating a hierarchical graphical user interface (GUI) on remote displays. For example, a user could bind music controls to their forearm through their mobile device, enabling eyes-free control of playback via the ultrasonic sensor attached to the wrist (Fig. 2.15).

Mujibiya et al. [40] proposed a novel on-body sensing technique using transdermal ultrasound propagation, known as The Sound of Touch. This system actively transmitted low-frequency ultrasound signals through body tissues to sense touch locations, applied pressure, and complex gestures such as arm-grasping. Unlike prior approaches, it provided continuous localization of touch positions along the body surface and could differentiate varying levels of pressure, enabling richer interactions such as slider adjustments and click-like events. The system was designed to be minimally intrusive, requiring only two



Figure 2.15: A user utilizing PUB to bind music controls to the forearm through a mobile device, enabling eyes-free control of the music player via an ultrasonic sensor attached to the wrist [27].

small ultrasound transducers for operation, making it highly practical for integration into wearable devices. As illustrated in (Fig. 2.16), a transducer placed perpendicular to the skin generates surface wave propagation, which is key to accurate touch localization. Experimental studies with 20 participants demonstrated the system's robustness in detecting precise touch locations and accurately classifying touch gestures under different levels of applied pressure.

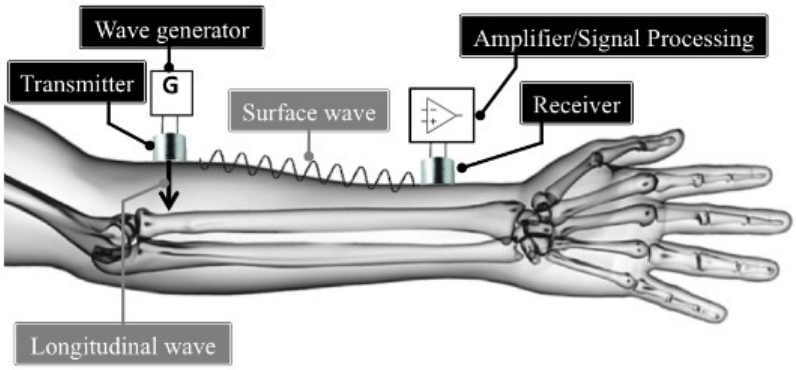


Figure 2.16: The properties of ultrasound signal propagation along the skin. A transducer placed perpendicular to the skin results in surface wave propagation [40].

These sensor-based approaches highlight the potential of integrating bio-acoustic sensing, ultrasonic technology, and transdermal signal propagation to expand the interactive capabilities of the human body. By leveraging the body's natural properties as a communication medium, these methods provide effective solutions for always-available, eyes-free, and context-aware interactions, particularly valuable for enhancing virtual and augmented reality experiences.

2.2.2 *Electrical Signal-Based On-Arm Interaction*

The use of electrical signals has also demonstrated significant promise in accurately identifying the specific locations where a finger touches the arm. One notable example is the

SkinTrack system, introduced by Zhang et al. [41], which utilizes the human body as an electrical waveguide for continuous touch tracking on the skin. This system consists of two key components: (1) a ring worn on the user's finger that emits a continuous, harmless, and imperceptible 80 MHz AC signal, and (2) a wristband worn on the opposite arm, equipped with a structured array of electrodes. When the finger touches the skin, the electrical signal propagates through body tissues, and phase differences between the received signals at multiple electrode pairs are measured to estimate the precise 2D touch location.

As illustrated in (Fig. 2.17), the position of the finger on the skin results in varying phase differences, which the system uses to accurately determine touch location. It is important to note that the waveform ripples shown are illustrative and not to scale. The system achieved a high touch segmentation accuracy of 99% and a mean positional error of just 7.6 mm. Importantly, SkinTrack works reliably even through thin clothing and does not require a direct line of sight, offering a highly practical solution for expanding interaction spaces beyond small device displays.

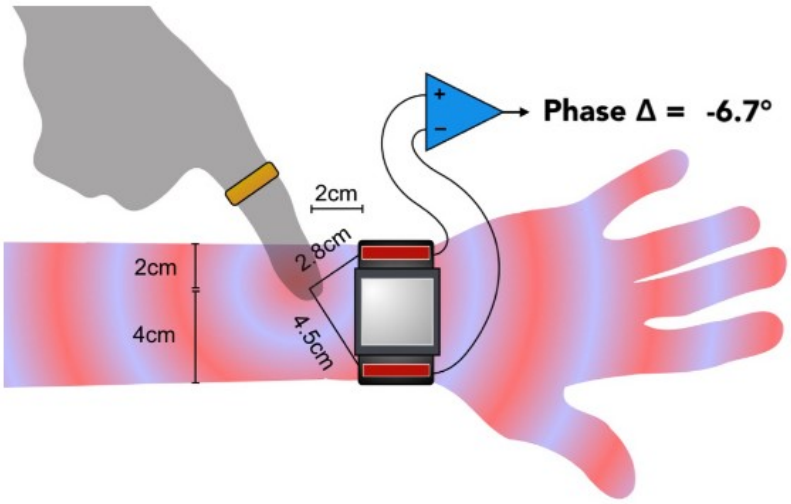


Figure 2.17: SkinTrack system schematic. A finger's location on the skin produces varying phase differences observed by electrode pairs, enabling accurate touch localization. Note that the waveform "ripple" is illustrative and not drawn to scale [41].

Building upon these concepts, Zhang et al. further developed ActiTouch, a system designed specifically for on-skin AR/VR interfaces [42]. ActiTouch advances prior work by integrating electrical touch sensing with computer vision-based spatial tracking. The system requires only a wristband emitter and sensors integrated into a VR/AR headset, eliminating the need for additional finger instrumentation. It utilizes the body's natural conductivity as an RF waveguide to achieve precise and low-latency touch segmentation.

As shown in (Fig. 2.18), dominant RF signal paths change before and after touch events, enabling the system to detect touch interactions by measuring the capacitive coupling effect through electrodes integrated into the headset. This approach also allows distinguishing between hovering and actual touch interactions on the skin—a common limitation in purely vision-based systems.

When combined with Leap Motion for accurate 3D finger tracking, ActiTouch delivers a robust system capable of supporting touchscreen-like interactions directly on the skin.

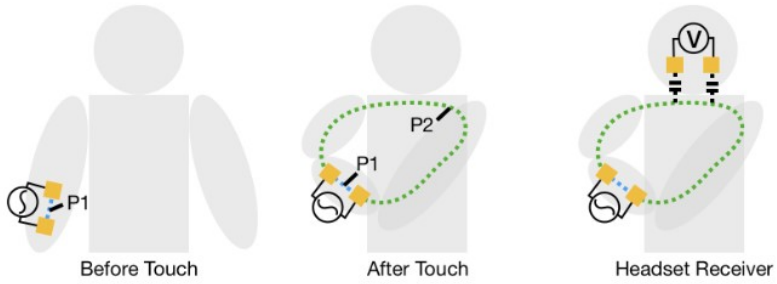


Figure 2.18: Dominant RF signal paths before (left) and after touch (middle). The two electrodes on the headset receiver capacitively couple to P2 for touch detection (right) [42].

Experimental results demonstrated that ActiTouch achieves a touch segmentation accuracy of 93.8% with a mean spatial tracking error of 5.3 mm, making it highly suitable for VR/AR applications where physical controllers may not be desirable.

2.2.3 *Photo-Reflective Sensing for On-Body Interaction*

The use of photo-reflective sensing has demonstrated significant potential in measuring both biological information and user interactions directly on the human body. SenSkin, introduced by Ogata et al. [43], proposed a thin, band-type wearable device equipped with multiple infrared (IR) reflective sensors to detect skin deformation caused by finger touches on the forearm. By analyzing variations in IR light reflection, the system is capable of sensing both normal and tangential forces applied to the skin. This enables not only the detection of touch events but also the differentiation of gestures such as pulling, pushing, and pinching.

As illustrated in (Fig. 2.19), the proposed input interface uses skin deformation to enable rich gesture recognition directly on the forearm. SenSkin also provides natural tactile feedback, allowing users to intuitively perceive the area of contact through direct physical interaction. To classify gestures accurately, the system employs a Support Vector Machine (SVM), achieving high recognition accuracy for various input commands. Its socially acceptable design makes it suitable for daily wear, introducing a soft interface that enhances human-computer interaction without relying on traditional touchscreens.

Building upon this concept, SkinWatch [44] extends photo-reflective sensing to wearable devices, specifically smartwatches. SkinWatch integrates IR photo-reflective sensors directly on the backside of the smartwatch, allowing the detection of finger gestures performed on the adjacent skin area. As shown in (Fig. 2.20), the hardware incorporates twelve sensors installed on the back of the device to enable precise gesture recognition around the wrist.

This configuration significantly expands the interaction space without requiring additional hardware worn on the fingers. The system detects a range of gestures, including directional swipes, pinching, stretching, and rotational movements, using both one-finger and two-finger inputs. By employing machine learning techniques such as SVM, SkinWatch effectively recognizes complex gesture patterns while minimizing false inputs through

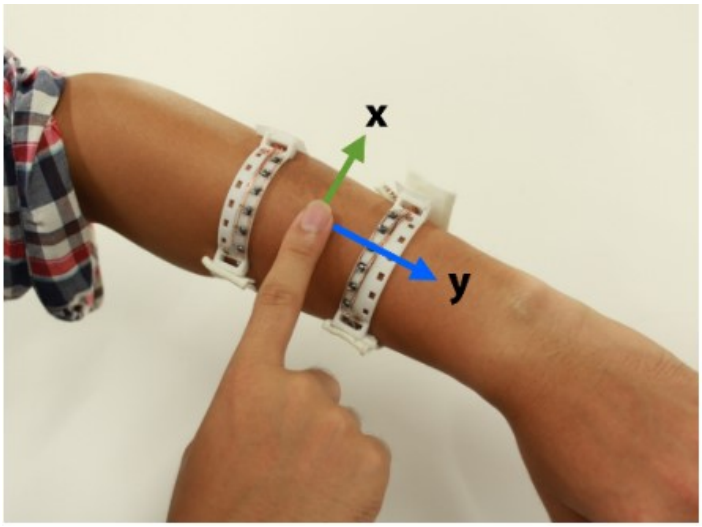


Figure 2.19: SenSkin system: the proposed input interface using skin deformation, enabling recognition of both normal and tangential forces applied to the forearm surface [43].

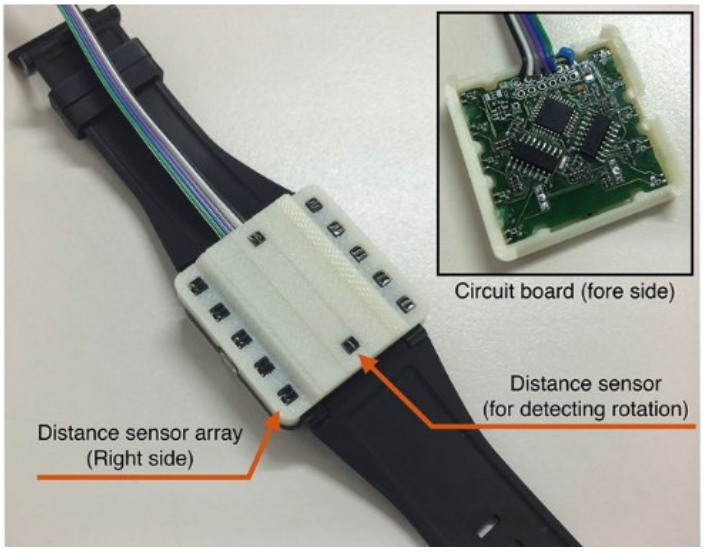


Figure 2.20: SkinWatch hardware prototype: backside of the device. Twelve sensors are installed on the backside of the watch, enabling accurate gesture detection around the wrist area [44].

anti-error mechanisms. The compact sensor design ensures low energy consumption and minimal impact on the watch's size and weight, making it a practical solution for enhancing interaction with wearable devices without obstructing visual feedback on the display.

2.2.4 *Wearable skin sensors*

Wearable skin sensors provide another promising approach for detecting interactions on the arm. One example is the Armstrong study, which investigated the use of non-dominant arm-anchored user interfaces (UIs) to improve interaction performance in virtual environments [23]. In this study, the researchers utilized 3D-printed pads with retroreflective markers attached to the user's forearm, hand, and fingers. These markers were tracked

using a Vicon motion capture system to accurately record arm and hand movements. The study systematically evaluated 72 regions around the arm in terms of reachability, comfort, and selection performance, providing valuable design guidelines for optimized 3D UI layouts. Their results demonstrated that targets located closer to the wrist and medial side of the forearm could be selected more quickly and comfortably than those near the elbow or lateral side. This finding highlights the importance of considering anatomical and ergonomic factors when designing on-arm interaction systems.

Another notable example is the GestureSleeve project by Schneegass et al. [45], which introduced a wearable sleeve made from touch-sensitive textile to recognize a variety of gestures performed on the forearm. The textile consists of two layers of striped electrodes arranged perpendicularly with a force-sensitive fabric sandwiched in between, allowing the detection of both simple touch actions and complex stroke-based gestures. As shown in (Fig. 2.21), this fabric-based input interface significantly expands the input space for smartwatches and wearable devices by enabling natural touch interaction directly on clothing.

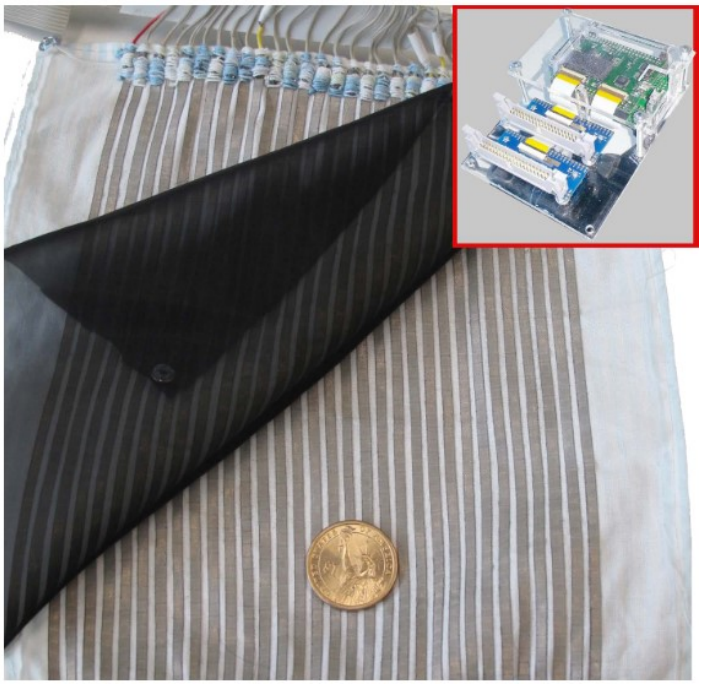


Figure 2.21: The touch-enabled textile used for creating the GestureSleeve. Two layers of striped electrodes are placed perpendicular to each other with a force-sensitive fabric sandwiched in between. The processing electronics is shown in the top right [45].

Users can control applications through intuitive gestures such as swipes, taps, and circular motions. A user study conducted with a fitness tracking application demonstrated that using the GestureSleeve resulted in faster interaction times compared to direct touch input on smartwatch screens, especially during activities like running. This highlights its effectiveness for on-the-go and eyes-free interaction scenarios.

2.3 BIMANUAL ACTION THEORY

Bimanual activities are typically classified into two categories: symmetrical and asymmetrical actions [46]. This section reviews relevant theoretical frameworks and connects them to the proposed arm-tapping interaction technique using VR controllers.

2.3.1 *Asymmetrical Bimanual Actions*

Asymmetrical bimanual actions involve both hands collaborating on a task while performing distinct and complementary roles. Typically, one hand leads and handles precise tasks, while the other supports by providing a spatial reference or stabilizing function—for example, when playing a violin or handwriting. Guiard's Kinematic Chain Model (KCM) [46] offers a foundational understanding of this phenomenon. In the KCM framework, the non-dominant hand establishes the reference frame and handles coarse spatial positioning, while the dominant hand operates within that frame to perform fine, detailed movements.

Hinckley et al. [47] explored virtual object manipulation using both hands, particularly in neurosurgical visualization systems. In their system, the non-dominant hand held a head-shaped prop to control the orientation and scaling of 3D brain models, providing a spatial reference. Simultaneously, the dominant hand manipulated a cutting-plane prop to explore cross-sectional views of brain structures. This bimanual approach allowed surgeons to intuitively rotate and examine 3D models while performing precise cutting operations, improving spatial understanding and reducing task completion time. The research demonstrated that two-handed interactions better leverage users' natural motor skills, leading to more efficient and engaging 3D manipulation experiences.

Kurtenbach et al. [48] introduced the T3 prototype GUI, which integrates tablets, two-handed input, and transparent UI components. In this system, the non-dominant hand controls a transparent tool palette (ToolGlass), positioning it close to the area of interaction, while the dominant hand performs precise selections or manipulations through it. This design minimizes divided visual attention by keeping controls close to the point of focus and maximizes screen real estate for application content. By aligning with the KCM model, the non-dominant hand handles gross positioning of interface elements, and the dominant hand performs detailed interactions, resulting in smoother workflow and reduced cognitive load for tasks such as drawing, object manipulation, and interface navigation.

Balakrishnan and Kurtenbach [49] demonstrated the effectiveness of applying KCM principles to 3D camera control. In their experiments, the non-dominant hand controlled the virtual camera's position and orientation, while the dominant hand performed object selection and manipulation within the scene. This separation of tasks allowed users to maintain a stable spatial frame with one hand while exploring and interacting with objects using the other. Their studies showed a 20% improvement in target selection times and higher user satisfaction when using the bimanual technique compared to traditional one-handed con-

trol schemes. Additionally, the approach enhanced depth perception through continuous camera adjustments, leveraging motion cues critical for 3D scene comprehension.

Building on these insights, recent research has increasingly applied the Kinematic Chain Model (KCM) to arm-based interaction techniques. In such approaches, the non-dominant hand serves as a spatial reference by presenting the forearm as an interactive surface, while the dominant hand performs precise tapping or gestural inputs to issue commands. This design effectively utilizes the always-available physical surface of the body, reduces reliance on visual targeting [24, 25], and enhances proprioceptive feedback, enabling more accurate and natural gesture execution.

As illustrated in (Fig. 2.22), the concept of using both hands to perform asymmetric roles is effectively demonstrated through a book interaction metaphor [24]. In this example, the non-dominant hand holds the spatial reference by displaying virtual content on the palm, while the dominant hand performs the flipping gesture to turn pages and advance through the interface. This visual metaphor not only supports intuitive, eyes-free interaction but also exemplifies the KCM model, emphasizing the natural division of labor between the hands for spatial control and precise action execution.



Figure 2.22: Two hands can create a book interaction metaphor. "Pages" can be turned by flipping the appropriate hand, which transitions the interface. Time advances from left to right. This interaction also visually demonstrates the application of KCM principles, where the non-dominant hand provides a spatial reference and the dominant hand performs the dynamic flipping action [24].

2.3.2 *Symmetric Bimanual Activities*

Symmetric bimanual activities involve both hands performing identical or closely coordinated actions simultaneously, such as clapping or stretching an object. This type of interaction has been extensively studied to understand how cognitive factors—such as attention, task complexity, and visual integration—affect performance.

Balakrishnan and Hinckley [50] explored how visual integration and divided attention influence symmetric bimanual tasks through target-tracking experiments. Their findings revealed that visual integration between task components is critical for successful parallel coordination of both hands. As visual separation between tasks increased, users exhibited more sequential behavior and less parallelism, leading to decreased performance. Task difficulty further compounded these effects, suggesting that without proper visual and conceptual integration, symmetric bimanual actions become cognitively demanding and less efficient.

Building on this, Owen et al. [51] investigated curve manipulation tasks and demonstrated that symmetric bimanual input significantly improves performance when visual, conceptual, and input spaces are well integrated. Their experiments (Fig. 2.23) confirmed that users completed complex curve matching tasks more quickly and accurately using symmetric two-handed techniques compared to unimanual methods. They also highlighted the importance of epistemic actions—physical motor actions used to simplify cognitive processing—which were more effectively supported by symmetric bimanual input. This research underscores the value of bimanual interfaces in reducing cognitive load and enabling more efficient, fluid interaction in complex manipulation tasks.

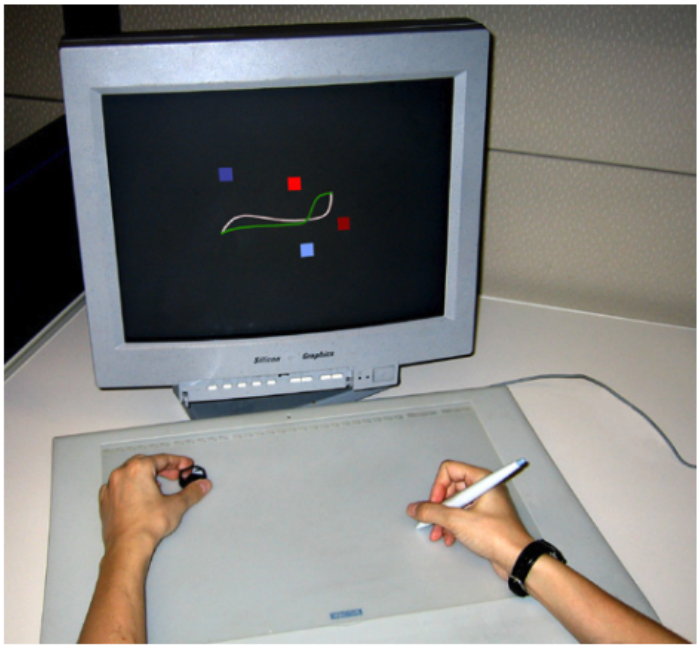


Figure 2.23: Experimental setup for the symmetric bimanual curve manipulation study. The experimental setup allowed participants to control curve parameters using a stylus in their dominant hand, while simultaneously adjusting supporting variables with a small puck held in the non-dominant hand. This configuration leveraged the Wacom tablet's capability to sense both input devices simultaneously, facilitating seamless bimanual interaction. [51].

Latulipe et al. [52] introduced the symSpline technique, a symmetric dual-mouse interface (Fig. 2.24) for spline manipulation. This technique allowed users to simultaneously control tangent angles and curvature adjustments, reducing the need for mode switching and improving the exploration of design solutions. Experimental results showed that symSpline outperformed both asymmetric dual-mouse methods and traditional single-mouse techniques in terms of speed, accuracy, and user preference. The approach was particularly beneficial for novice users, who often struggled with the cognitive load introduced by sequential editing modes. By allowing simultaneous control of multiple parameters, symSpline facilitates a more natural and efficient editing workflow for curve manipulation tasks.

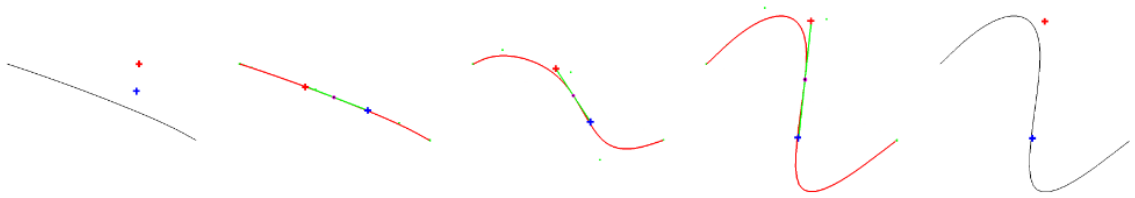


Figure 2.24: symSpline interface concept. symSpline utilizes two cursors to control the ends of the tangent at an edit point. By translating the entire tangent, users can adjust both the position of the edit point and the curve's shape in a continuous and intuitive manner [52].

2.4 CONCLUSION

This review surveyed interaction techniques that ground our work in embodied, efficient input. From Guiard's Kinematic Chain Model (KCM), we take the principle of asymmetric bimanual coordination—one hand as a stable reference, the other as the actor—which has repeatedly improved speed, accuracy, and cognitive load across GUI, 3D manipulation, and camera control. We also examined on-arm interaction literature that treats the body as an always-available surface using diverse sensing (bio-acoustic, ultrasonic, electrical, optical, and wearable skin sensors), highlighting benefits such as eyes-free use and strong proprioceptive cues.

From theory to evidence. Guided by the literature, the remainder of this thesis: (1) builds a data-collection protocol on two device families, (2) trains a tapping *detector* from acceleration/velocity features of both hands (capturing the characteristic peak–trough around impact), (3) trains an *acting-hand (hand-use) detector* to identify which arm is moving vs. stabilising, using per-hand motion magnitudes, (4) trains a *location* classifier from relative hand geometry (positions and inter-controller distance), (5) quantifies *robustness* against seven common non-tapping actions, and (6) integrates the models into a VR shooter prototype to assess the *practical usability* of tap-driven commands.

Implication. The literature thus provides the rationale for an intuitive, low-effort, eyes-light interaction that can be realised with off-the-shelf hardware. Thesis B extends this body of work by delivering an end-to-end, evidence-based implementation and evaluation—moving from conceptual promise to measured performance in real VR tasks.

3

EXPERIMENT ONE: DATA COLLECTION AND MODEL TRAINING

This chapter proposes a method for identifying arm tapping with VR controllers. The method first detects the moment when an arm and a controller contact, and then identifies hand-use cases, and lastly determines tapping positions and forces. In this chapter, we present the data collection procedure and describe the three steps of the method one by one. We conclude the experiment with data analysis and result discussion.

3.1 DATA COLLECTION

This section presents the end-to-end data collection setup used in our study. First, we describe the apparatus—the VR device configuration and software environment used to run the experiment. We then specify the participant cohort and recruitment criteria. Next, we explain the data logging process, detailing how signals were captured from the VR controllers, followed by the feature schema that defines which controller-derived variables were retained for model training. Finally, we outline the labeling procedure used to convert raw signals into ground-truth annotations for tap detection, tap position, hand-use detection, and tap force.

3.1.1 *Apparatus*

The system runs on *off-the-shelf* VR headsets using their bundled controllers. This choice keeps the method portable: any headset that already tracks its controllers can run the tapping system as-is, with **no external cameras, markers, or wearables** required.

3.1.1.1 *Head-Mounted Displays and Controllers*

We utilized two representative platforms to encompass various controller designs and sensing methodologies:

- **Meta Quest 2** (LCD, 1832×1920 per eye) with inside-out tracking and ring-on-the-side controllers.
- **HTC Vive Pro** (dual OLED, 1440×1600 per eye) with Lighthouse tracking and ring-on-top controllers.

3.1.1.2 *Host Machine and Software*

All sessions ran on Windows 10 (Intel i9–9880H, 8 GB RAM, NVIDIA RTX 2080). The experimental app was built in Unity 2022.3.46f1 (C#). For each frame, we logged per-controller poses and kinematics (positions; linear/angular velocities and accelerations), and derived inter-controller geometry (e.g., pairwise distance). This low-level access ensures real-time feasibility and device portability.

This setup prioritizes deployability and robustness. Because it runs on commodity VR hardware, the technique remains practical outside the lab. Using two device families allows us to test cross-device robustness under different sensing stacks (i.e., distinct IMU data paths and tracking methods). Implementing data logging in Unity gives synchronized access to all required signals for labeling and feature extraction, with no need for custom drivers.

3.1.2 *Participants*

We recruited 16 right-handed participants (8 male, 8 female), aged 20–35 ($M = 23.75$, $SD = 0.83$). Only 1 participant reported prior experience with VR devices. We restricted recruitment to right-handed participants to control for asymmetries in bimanual coordination between the tapping hand and the target arm. We received ethics approval in Zhejiang University and carried out the experiment there.

3.1.3 *Experiment Procedure*

Participants completed a *tapping* block in which each hand tapped three target zones on the opposite forearm (near wrist, mid-forearm, near elbow) at two force levels (small/large).

Each trial followed a standard prompt–act–return routine to bound the motor sequence and align data:

1. **Start posture:** both controllers held at chest height.
2. **Prompt:** press the start key (`X` on Quest / disc on Vive); on-screen text specifies target zone and force cue.
3. **Action:** the tapping hand performs a single tap on the instructed zone.
4. **Return:** hands return to the start posture.
5. **End:** press the end key (`Y` on Quest / trigger on Vive).

Figure 3.1 shows the five-step timeline: (a) adopt baseline posture; (b) press start; (c) execute the instructed arm tap; (d) return to baseline; (e) press end. Start/end key presses provide anchors for aligning kinematic peaks with the action phase.

This block yields labeled positives across both hands (left/right), three forearm zones (wrist/mid/elbow), and two force levels, providing the variation needed for training and evaluating the detector/locator.

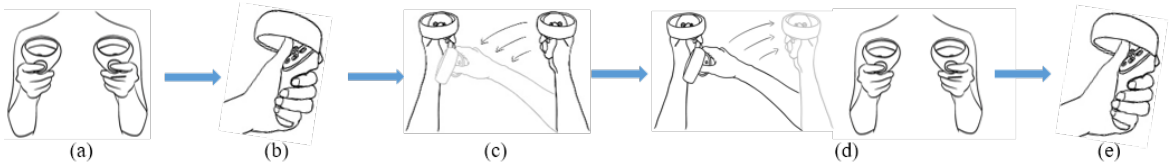


Figure 3.1: Experiment 1 trial timeline and hand poses. Panels (a–e): (a) baseline posture (both controllers at chest height); (b) start cue—press X (Quest) or circular disc (HTC); (c) execute the instructed arm tap (zone and force on screen); (d) return to baseline; (e) end cue—press Y (Quest) or trigger (HTC).

3.1.3.1 *Independent Variables*

Following common HCI experiment design practice, factors were defined a priori and fully crossed within participants (cf. clear separation of factors in prior VR gesture studies).[1]

- **Hand use** (2 levels): left vs. right (active hand taps the opposite forearm).
- **Knock position** (3 levels): near wrist, mid-forearm, near elbow.
- **Force** (2 levels): small vs. large (instructional intensity).
- **Device family** (2 levels): Meta Quest 2 vs. HTC Vive Pro. Device order was counterbalanced across participants to probe cross-device robustness without confounding practice effects.

The explicit declaration and counterbalancing of factors aligns with best practice in VR controller studies where interaction techniques and task scenarios are separated into independent variables prior to analysis.

3.1.3.2 *Dependent Variables*

The following outcomes were recorded on every trial to support analysis and model training:

- **Kinematic signals (per frame, 10 ms):** 3D positions, linear and angular velocities, and accelerations for both controllers; inter–controller distance for geometric features.
- **Frame labels (tap / non-tap):** obtained from the acceleration-based labeling procedure (Sec. 3.1.5); later aggregated to window-level labels via majority rule (Sec. 3.2.1).
- **Tap metadata:** instructed *knock position* (wrist / mid-forearm / elbow) and *force cue* (small / large) stored with each trial for supervising location and force models.
- **Action timing:** duration between *start* and *end* key presses (used for quality control and descriptive statistics).

1 We mirror the clarity of factor specification recommended in comparable controller-gesture studies that enumerate IVs before reporting outcomes.

3.1.4 *Data Logging and Feature Schema*

This section outlines how we sample controller signals and derive features for model training. We record synchronized measurements from both controllers every 10 ms, a rate that is close enough to preserve continuous motion yet spaced enough to capture meaningful change between frames while reducing redundancy and labeling effort. Each frame includes positions, linear and angular velocities, accelerations, and inter-controller geometry (e.g., relative offsets and distance). Our feature set focuses on signals that are both informative and portable: acceleration to capture brief impact transients, velocity to distinguish fast taps from slow repositioning, and inter-controller geometry to encode the approach–contact–retreat pattern of bimanual tapping. Using measurements from both hands reduces false positives arising from unilateral motions, and all features are available on commodity VR controllers, enabling real-time deployment without custom sensors.

3.1.4.1 *Sampling and Logged Signals*

We record all signals once every **10 ms** (10 Hz). Each frame contains synchronized measurements from both controllers, including positions, linear and angular velocities, accelerations, and inter-controller geometry (e.g., pairwise distance).

This sampling rate balances continuity and contrast. Frames are *close enough* to form a continuous motion sequence for tapping, yet *far enough apart* to reveal meaningful change between consecutive frames. In practice, 10 ms avoids near-duplicate samples that add noise and autocorrelation, reduces storage and labeling overhead, and still preserves the dynamics needed for reliable model training and real-time use.

These frame-wise signals are later aggregated into short sliding windows for classification; with our default settings, a window spans three frames (300 ms) and advances by one frame (10 ms) per decision.

3.1.4.2 *Feature List*

Table 3.1 catalogs features and meanings (e.g., `leftV`, `rightA`, `l-dA`, `d-between-x`). For reproducibility, we recompute finite differences offline to avoid drift and exclude the first valid delta in each block.

We select features that are informative for taps yet portable to compute. Axis-wise acceleration provides a short, distinctive transient at impact, making it the most direct cue. Velocity distinguishes brief, high-tempo taps from slow repositioning or drift. Inter-controller geometry (relative positions and distance) captures the approach–contact–retreat pattern of bimanual tapping. Using signals from both hands reduces false positives from unilateral motions by encoding the coordination required for an actual tap. All signals come from commodity VR controllers, enabling real-time use without custom sensors.

Table 3.1: Features used in the CSV files (Experiment 1).

Field	Description
Username	Participant identifier (anonymised).
Group	Scenario / block ID.
No	Sample index within a group (10 ms per record; starts at 2).
Hand	Active/tapping hand: 0 = left controller taps right arm; 1 = right controller taps left arm.
Strength	Tap strength instruction: 0 = small; 1 = large.
Knock-position	Target along the forearm: 0 = near wrist; 1 = mid forearm; 2 = near elbow.
leftV	Velocity magnitude of the left controller, $\|\mathbf{v}_l\|$.
leftA	Acceleration magnitude of the left controller, $\|\mathbf{a}_l\|$.
l-dA	Finite difference of left acceleration magnitude: $\|\mathbf{a}_l(t)\| - \|\mathbf{a}_l(t-\Delta t)\|$.
rightV	Velocity magnitude of the right controller, $\|\mathbf{v}_r\|$.
rightA	Acceleration magnitude of the right controller, $\|\mathbf{a}_r\|$.
r-dA	Finite difference of right acceleration magnitude: $\|\mathbf{a}_r(t)\| - \|\mathbf{a}_r(t-\Delta t)\|$.
between-x	Inter-controller x-axis offset: `lPos-x` − `rPos-x`.
between-y	Inter-controller y-axis offset: `lPos-y` − `rPos-y`.
between-z	Inter-controller z-axis offset: `lPos-z` − `rPos-z`.
d-between-x	Temporal delta of `between-x`: `between-x`(t) − `between-x`$(t-\Delta t)$.
d-between-y	Temporal delta of `between-y`.
d-between-z	Temporal delta of `between-z`.
lPos-x, lPos-y, lPos-z	Left controller world coordinates (ACL).
rPos-x, rPos-y, rPos-z	Right controller world coordinates (ACR).
lRot-x, lRot-y, lRot-z	Left controller angular velocity about x/y/z (AVL).
rRot-x, rRot-y, rRot-z	Right controller angular velocity about x/y/z (AVR).
DIS	Inter-controller distance: $\|\mathbf{p}_l - \mathbf{p}_r\|$.

3.1.5 *Labeling*

Manual labels are required to train the supervised models, so each trial is annotated to establish ground truth from the active hand's acceleration. Annotation follows a consistent, signal-based procedure to ensure reproducibility.

Tap labels are anchored to the active hand's acceleration. For each prompted trial, the dominant *peak–trough* excursion in the acceleration around the action phase is located, and the event is bounded from the point where the signal departs the baseline to the point where it returns to baseline. Concretely, a baseline (mean and variability) is estimated from the pre-action segment; a sharp positive peak followed by a negative trough (or vice versa) near the prompt defines the core of the tap. The provisional start index is the first sample exceeding the baseline band, and the end index is the first sample re-entering the

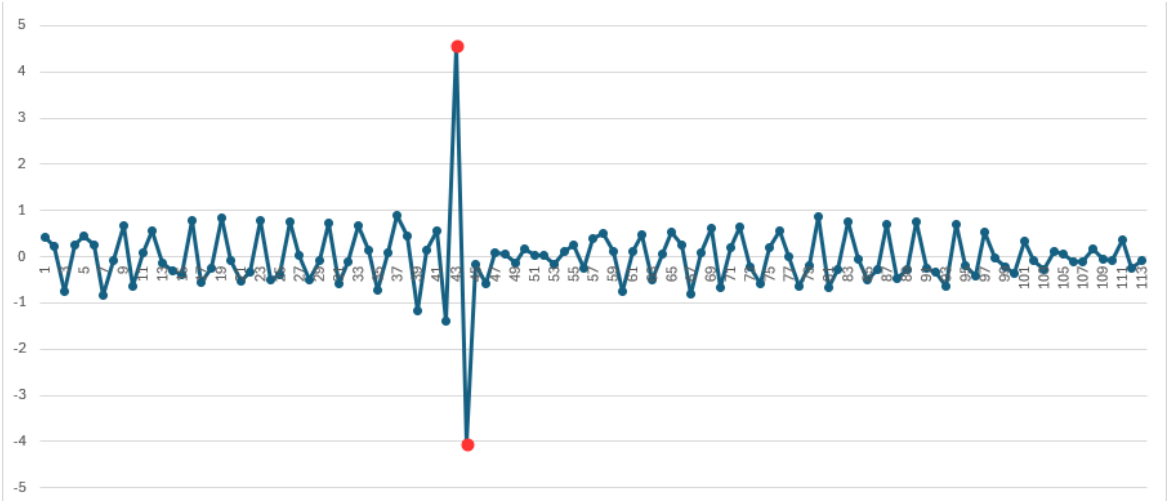


Figure 3.2: Active-hand acceleration around a single tap. The red markers indicate the dominant peak and trough that characterize the impact. The label interval is taken from the point where the signal departs the baseline before the peak to the point where it returns to baseline after the trough; low-amplitude oscillations on both sides reflect normal hand motion outside the tap.

band after the trough. All suggested bounds are visually checked against surrounding kinematics to ensure the label spans the actual contact.

As shown in Fig. 3.2, tap events produce a sharp peak–trough excursion in the active-hand acceleration. Using acceleration (and its finite differences) captures this brief, time-local signature more clearly than position or velocity, making onsets easier to detect and reducing jitter in the start–end boundaries.

We use acceleration because tap events are brief, high-tempo excursions. Acceleration (and its finite differences) yields sharp, time-local signatures that make onsets easier to detect than using position or velocity alone, reducing jitter in start–end boundaries.

3.2 TAPPING IDENTIFICATION

In this section, we describe our approach to tapping identification. We begin with the sliding-window technique used to segment the controller time series into short, overlapping intervals; this framing is essential for preparing data that captures the rise–peak–drop dynamics of a tap and for producing consistent train/evaluation splits. We then detail five supervised models: a tap detector (binary), two tap-position classifiers (one per tapping hand), a hand-use classifier, and a force-level classifier. Together, these models cover when a tap occurs, where it lands on the forearm, which hand performs it, and how strongly it is executed.

3.2.1 *Sliding Windows*

Sliding windows are introduced to turn raw frames into inputs that carry clearer, more reliable information for learning. Instead of training on isolated samples, each example

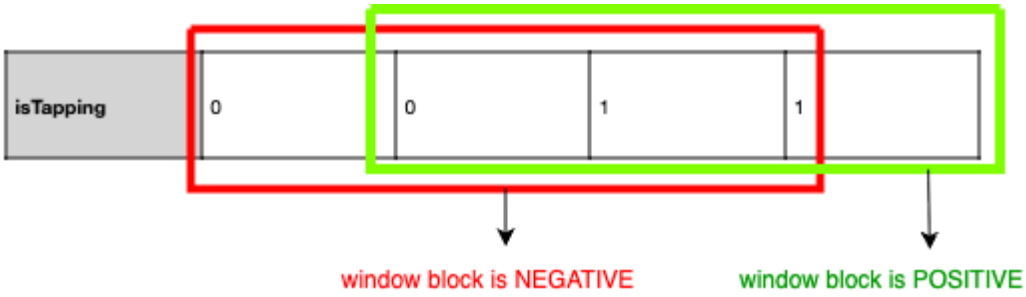


Figure 3.3: Window labeling with size = 3, stride = 1. The red window contains only one tap frame (majority = 0) and is labeled *negative*; the green window contains two tap frames (majority = 1) and is labeled *positive*. Using window blocks provides temporal context for training and mitigates boundary jitter from frame-wise labels.

captures a short temporal trend that characterizes a tap, improving separability from ordinary motion and reducing sensitivity to timing jitter and sensor noise.

In our data, taps manifest as a sharp peak–trough excursion in the active-hand acceleration (Fig. 3.2). Capturing the brief span that covers this excursion provides exactly the context the model needs while avoiding unnecessary delay. We therefore use compact windows of 300 ms (size = 3 frames) advanced every 10 ms (stride = 1), which preserve low-latency decisions and keep input dimensionality fixed. Because windows—not individual frames—serve as training instances, each window must be labeled. We derive the window label from its constituent frame labels using a simple majority rule (Fig. 3.3), which is robust to small boundary misalignments.

3.2.1.1 *Window Configuration*

We use a **window size of 3 frames** (300 ms) with a **stride of 1 frame** (10 ms). This choice follows empirical comparisons of sizes $\{3, 5, 7, 9\}$ and strides $\{1, 2, 3\}$: size = 3 captured the sharpest tap signatures with low latency while avoiding extra noise from longer spans, and stride = 1 maximized temporal resolution for responsive updates. Larger windows slightly smoothed noise but increased decision lag without improving accuracy on our data.

3.2.1.2 *Window Labeling*

Frame-level tap labels from the labeling protocol (Sec. 3.1.5) are mapped to window-level labels using a simple majority rule, as illustrated in Fig. 3.3. A window is labeled *positive* if more than half of its frames are tap; otherwise it is *negative*. With size = 3, this means a window is positive when at least $2/3$ frames are tap. This majority voting reduces sensitivity to minor timing jitter at tap boundaries and better reflects the short, cohesive temporal pattern the model is intended to learn.

3.2.2 *Candidate Classifiers*

To identify the most suitable model for each detection task, we benchmarked a set of classifiers using identical features and subject-separated splits under their default hyperparameters. We selected the top performer by validation accuracy (with precision/recall/F1 as tie-breakers), then fine-tuned its hyperparameters to further improve detection accuracy. Here are all of models that we used to test:

- **Linear / Logistic Regression** — fast, interpretable baselines to sanity-check feature quality and reveal whether simple decision boundaries suffice.
- **Decision Trees** — low-compute, interpretable non-linear splits; well-suited to on-device inference.
- **Random Forests** — bagged trees that reduce variance and provide robust accuracy on tabular window features.
- **ExtraTrees (Extremely Randomized Trees)** — ensembles of trees with randomized split thresholds that further reduce variance and overfitting; strong, fast baselines on tabular data.
- **XGBoost / LightGBM** — gradient-boosted trees capturing complex interactions and handling imbalance effectively.
- **k-Nearest Neighbors** — instance-based classifier testing whether local temporal similarity between windows is sufficient.
- **AdaBoost** — boosting that emphasizes hard examples, complementary to bagging approaches.

3.2.3 *Tapping Detector*

To train an on-device tap detector, we first define *what the model sees* (features), then *how supervision is provided* (labels), and finally *which classifier are benchmarked* (model suite). The design goal is a practical balance of accuracy, latency, and interpretability suitable for real-time VR.

FEATURES We use compact, device-agnostic signals that capture the short-term dynamics of a tap while remaining efficient for on-headset inference. Each window is represented by:

- **Acceleration (axis-wise, both controllers):** the primary cue for impact transients, showing a sharp peak–trough morphology near contact.
- **Velocity (axis-wise, both controllers):** separates brief, high-tempo taps from slow repositioning or drift.

Hand-aware remapping for clarity. To improve left/right tap detection and simplify model inputs, we rename acceleration channels per trial based on the instructed tapping hand:

- If the *right* hand is tapping (left hand steady): `movingArmAcceleration` ← right-hand acceleration; `nonMovingArmAcceleration` ← left-hand acceleration.
- If the *left* hand is tapping (right hand steady): `movingArmAcceleration` ← left-hand acceleration; `nonMovingArmAcceleration` ← right-hand acceleration.

This hand-aware relabeling makes the feature roles consistent across trials ("moving" vs. "non-moving") and helps the classifier focus on the arm that generates the tap while still referencing the other arm to suppress false positives from unilateral motion.

All features are scaled with **min–max normalisation to [0, 1]**. Per-feature minima and maxima are computed on the *training* split and applied unchanged to validation/test/runtime to avoid leakage and keep scaling consistent across devices and sessions.

LABELS For supervised training, the model needs both input features and ground-truth labels. For tap-moment detection, we use a frame-level flag `isTapping` that we manually annotated (see Sec. 3.1.5). Because training is performed on sliding windows, each window receives a single label by majority vote over its frames: if more than half of the frames in the window have `isTapping`=1, the window is labeled *positive*; otherwise it is *negative*. This majority rule reduces boundary jitter and better reflects the short, cohesive temporal pattern of a tap.

CHOSEN CLASSIFIER Under the real-time accuracy/latency/interpretability criterion, we adopt a **Decision Tree** as the primary tap detector. In pilot tests it matched or exceeded linear baselines and approached ensemble performance, while remaining substantially simpler to deploy and debug. The tree consumes the axis-wise acceleration and velocity features from both controllers and affords transparent decision paths for error analysis without a heavy computational footprint.

3.2.4 *Tapping Location Detector*

For tap *location* classification, the goal is to use short windows to infer which forearm zone was contacted. Location is fundamentally geometric, so the model focuses on spatial coordination between both controllers rather than impact dynamics. We therefore represent each window with compact geometric inputs, supervise with zone labels aligned to taps, and benchmark the same classifier suite used for tap detection before selecting a deployable model.

FEATURES Each window is encoded by the **3D coordinates of both controllers** (left/right) as listed in Table 3.1, together with a **derived Euclidean distance between**

the controllers that is computed during preprocessing (i.e., *not* included in Table 3.1). This combination provides a clean geometric description of approach and contact: relative pose plus separation helps disambiguate whether the tapping hand is near the wrist, mid-forearm, or elbow of the opposite arm. All features are scaled with **min–max normalisation to [0, 1]**, using minima and maxima computed on the training split and applied unchanged to validation/test/runtime.

LABELS Window-level targets use the **knock-position** zone label (*wrist / mid-forearm / elbow*). Zone labels are assigned only to windows that are already labeled as *tap* by the detector pipeline, ensuring that supervision reflects geometric differences among true tap events rather than background motion.

CHOSEN CLASSIFIER We adopt **ExtraTrees** for tap location, training *separate models for left-hand taps and right-hand taps*. In our validation, this per-hand split improved zone recall by respecting asymmetric kinematics, while ExtraTrees offered stable accuracy with fast inference and minimal tuning.

3.2.5 *Hand Use Detector*

The hand-use classifier determines *which hand performed the tap*.

FEATURES Each window is encoded only by the velocity magnitudes of the two controllers: `leftV` and `rightV`. Concretely, we take the `leftV`/`rightV` values from every frame in the window (e.g., three frames for a 300 ms window) and feed them directly to the model. Velocity cleanly separates the active (moving) arm from the stabilising arm during a tap. All features are scaled with **min–max normalisation to [0, 1]** using statistics computed on the training split and applied unchanged to validation/test/runtime.

LABELS Windows are annotated with the **Hand** label (*left / right*; see Table 3.1). For training this classifier, we include only windows that have already been marked as *tap* by the detector; non-tap windows are excluded.

CHOSEN CLASSIFIER We use **ExtraTrees (Extremely Randomized Trees)**. In validation, ExtraTrees provided stable accuracy on these compact, tabular features with fast inference and minimal tuning, making it a good fit for real-time VR deployment.

3.2.6 *Force Level Detector*

For *force* detection, the aim is to distinguish instructed tap intensity from short windows that capture impact dynamics. Because commodity controllers do not measure force directly, we rely on motion-derived cues that correlate with intensity.

FEATURES Windows are represented by axis-wise **velocity** and **acceleration** from both controllers (as listed in Table 3.1). These signals reflect impact tempo and magnitude and are the most direct kinematic proxies for perceived force on commodity hardware. Features are scaled with **min–max normalisation to [0, 1]** using statistics computed on the training split and applied unchanged to validation/test/runtime.

LABELS Window-level targets use the **Strength** field from Table 3.1 (*small* vs. *large* tap). Only windows already labeled as *tap* are assigned a force label so that supervision reflects impact dynamics rather than background motion.

CHOSEN CLASSIFIER We use a lightweight **Decision Tree** as the default force estimator for on-device deployment. While more complex ensembles offered marginal gains, the tree provided adequate accuracy, transparent splits for error analysis, and minimal computational overhead given that velocity/acceleration are only indirect proxies for force.

3.3 RESULT ANALYSIS AND DISCUSSION

This section evaluates the trained models and interprets what their metrics mean for real-time deployment. In summary, the *tap detector* attains near–perfect accuracy, the *hand-use detector* is effectively perfect (100% accuracy on held-out windows, cleanly separating left vs. right taps), the *tap-location classifiers* achieve consistently high performance with minor hand/zone asymmetries, and *force detection* remains weak.

3.3.1 *Tapping Detection*

The tap detector achieves near–perfect performance on held-out windows, showing that short, overlapping windows with acceleration/velocity features are sufficient for reliable, low-latency decisions. Table 3.2 reports class-wise and aggregate metrics; the detector attains **100% overall accuracy** across 96 471 windows, with the **tap** class (label 1) reaching **0.98** precision, **0.99** recall, and **0.98** F1. The **non-tap** class (label 0) is effectively perfect (precision/recall/F1 = **1.00**). Because the dataset is imbalanced (non-tap: 94 886 vs. tap: 1665), we include both weighted and macro averages; the macro average of **0.99** provides a conservative summary that is less influenced by class frequency.

These results indicate strong specificity against common controller motions and robust sensitivity to true taps. The perfect non-tap scores align with the two-hand feature design and the windowing scheme, which together suppress spurious activations during ordinary movement. The high tap recall (0.99) suggests that peak–trough transients in acceleration are consistently captured within the 300 ms windows. Given the class imbalance, macro averages are emphasized when comparing models. After this evaluation, a hyperparameter search (10-fold, 108 candidates; total 1080 fits) confirmed that the selected configuration already sits on a favorable accuracy–latency frontier, with no materially better trade-off found within the tested grid.

Table 3.2: Detector performance on the test set (window size = 3, stride = 1).

Class	Precision	Recall	F1-score	Support
0 (non-tap)	1.00	1.00	1.00	94,886
1 (tap)	0.98	0.99	0.98	1,665
Accuracy			1.00	96,471
Macro avg	0.99	0.99	0.99	96,471
Weighted avg	1.00	1.00	1.00	96,471

Table 3.3: Tap location (left-hand taps): test-set metrics.

Class	Precision	Recall	F1-score	Support
0 (wrist)	0.98	0.99	0.98	280
1 (mid-forearm)	0.97	0.92	0.95	275
2 (elbow)	0.95	0.99	0.97	282
Accuracy			0.97	837
Macro avg	0.97	0.97	0.97	837
Weighted avg	0.97	0.97	0.97	837

3.3.2 *Tapping Location Detection*

Location models achieve high accuracy with short windows, indicating that simple geometric cues (controller coordinates plus inter-controller distance) are sufficient to disambiguate wrist, mid-forearm, and elbow. We report per-hand results because splitting by tapping hand improves stability by respecting asymmetric kinematics.

3.3.2.1 *Left Hand*

The left-hand location model reaches **97% overall accuracy** on 837 windows, with balanced performance across all zones (Table 3.3). Class-wise F1 ranges from **0.95** to **0.98**, and macro averages of **0.97** confirm that no single class dominates the result.

These scores suggest that the geometric representation captures zone differences reliably for left-hand taps; minor asymmetries appear at the mid-forearm class (lower recall), which is expected given its intermediate position.

3.3.2.2 *Right Hand*

The right-hand location model attains **94% overall accuracy** on 800 windows (Table 3.4). Class-wise F1 spans **0.92–0.97**, with macro averages of **0.95**. Performance remains strong but is slightly lower than the left-hand model, consistent with handedness and pose variation.

Overall, per-hand training with geometric features yields accurate zone recognition while preserving low latency. The remaining gap to 100% is concentrated in mid-forearm

Table 3.4: Tap location (right-hand taps): test-set metrics.

Class	Precision	Recall	F1-score	Support
0 (wrist)	0.98	0.96	0.97	265
1 (mid-forearm)	0.93	0.91	0.92	275
2 (elbow)	0.93	0.97	0.95	260
Accuracy			0.94	800
Macro avg	0.95	0.95	0.95	800
Weighted avg	0.95	0.94	0.95	800

Table 3.5: Hand-use classifier performance on the test set (window size = 3, stride = 1).

Class	Precision	Recall	F1-score	Support
0 (left)	1.00	1.00	1.00	867
1 (right)	1.00	1.00	1.00	770
Accuracy			**1.00**	1,637
Macro avg	1.00	1.00	1.00	1,637
Weighted avg	1.00	1.00	1.00	1,637

vs. neighboring zones; increasing examples for this class or adding weak orientation cues may further reduce confusion without increasing model complexity.

3.3.3 *Hand Use Detection*

The hand-use classifier cleanly identifies which hand performed the tap. On the held-out test set (window size = 3, stride = 1), it achieves **100% accuracy** with **precision/recall/F1 = 1.00** for both classes (Table 3.5). Class supports are well balanced (left: 867, right: 770), indicating that perfect scores are not an artefact of extreme imbalance. These results confirm that short, windowed velocity magnitudes (`leftV`, `rightV`) are sufficient to separate the moving arm from the stabilising arm during a tap.

3.3.4 *Force Detection*

Force classification was evaluated on *left-hand, wrist taps* using velocity and acceleration features with min–max normalisation. Results indicate that commodity controller kinematics are an inadequate proxy for perceived force in our setup: overall accuracy is **0.81**, with class-wise F1 around **0.80–0.81** (Table 3.6). These scores are notably lower than tap detection and tap location, suggesting the signal does not carry enough stable information for reliable intensity estimation.

These outcomes are consistent with the limitation that the controllers do not sense contact force directly; velocity and acceleration provide only indirect, noisy cues. In addition,

Table 3.6: Force detection (left-hand, wrist taps): test-set metrics.

Class	Precision	Recall	F1-score	Support
0 (small)	0.76	0.84	0.80	122
1 (large)	0.85	0.78	0.81	145
Accuracy			0.81	267
Macro avg	0.81	0.81	0.80	267
Weighted avg	0.81	0.81	0.81	267

training on a single zone (left wrist) constrains variability, and minor timing misalignments around impact likely blur intensity differences across short windows. Given this evidence, the current model is *not* sufficient for robust force detection. Future improvements could incorporate higher-rate IMU streams, contact audio, or haptic microphone data, expand training to multiple zones and hands, and apply calibration per participant to reduce inter-user variance.

3.4 CONCLUSION

This chapter presented a controller-only pipeline for tapping recognition built around simple sampling, reproducible labeling, and short sliding windows. The core idea is to convert raw controller signals into compact, informative inputs suitable for real-time inference with lightweight models.

The apparatus and logging were specified as synchronized measurements from both controllers every 10 ms, followed by min–max normalisation to $[0, 1]$. Ground-truth taps were annotated from active-hand acceleration using a peak–trough procedure, and frame labels were aggregated by majority into 300 ms windows. For model inputs, tap detection used axis-wise acceleration and velocity from both controllers; tap location used the 3D coordinates of both controllers plus a derived inter-controller distance. Supervision targets comprised binary tap/non-tap labels, three-zone location labels (wrist/mid/elbow), two hand labels (left/right), and an exploratory two-level force label (small/large).

A concise suite of classical models was benchmarked and deployable classifiers selected. A **Decision Tree** was chosen for tap detection given its accuracy, low latency, and interpretability; **ExtraTrees** were adopted for tap location with separate models per tapping hand; and an **ExtraTrees** classifier was used for *hand-use detection*. Results show that short, overlapping windows carry sufficient signal for reliable recognition: the tap detector achieved **100%** accuracy (macro F1 **0.99**), the hand-use detector reached **100%** accuracy on held-out windows, and the location models achieved **97%** (left-hand) and **94%** (right-hand) accuracy. By contrast, force detection reached only **0.81** accuracy on left-wrist taps, indicating that commodity kinematics are an inadequate proxy for perceived force in this setup.

Overall, the **end-to-end design**—from windowed feature extraction to lightweight, on-device classifiers—delivers well-performed tap detection, perfect hand-use identification,

and strong location classification using commodity hardware. Limitations include class imbalance (non-tap dominance), modest confusion around mid-forearm versus adjacent zones, and weak force cues. Future work should explore richer or higher-rate sensing (e.g., contact audio, higher-rate IMUs), improved data balancing/augmentation, and light per-user calibration or domain adaptation to stabilise performance across hands, zones, and devices.

4

EXPERIMENT TWO: CLASSIFICATION OF TAPPING AND NON-TAPPING ACTIONS

This chapter starts by explaining how we collected the non-tapping dataset for **Experiment 2**, using the same participants from Experiment 1 to record seven everyday controller actions that can look tap-like. We then outline the **task and procedure**, followed by the study's **independent variables** (action type, device family, hand involvement) and **dependent variables** (detector outputs and evaluation metrics such as FPR, accuracy, precision, recall, and F1). Finally, in **Results Analysis and Discussion**, we examine whether the detector cleanly separates true taps from these non-tapping movements.

4.1 DATA COLLECTION

In this section, we describe the data collection for Experiment 2: the same participants from Experiment 1 performed seven distinct non-tapping activities (listed below). We received ethics approval in Zhejiang University and carried out the experiment there. This dataset characterises the negative class and enables robustness and false-positive analyses for our tapping models.

4.1.1 *Task and Procedure*

Participants completed a *non-tapping* block comprising seven representative controller actions used to probe false positives and estimate robustness during realistic movement.

Seven actions span common VR motions with varied directions and ranges: (1) swing arms forward, (2) place controllers on a table, (3) zoom, (4) collide controllers, (5) tap touchpads, (6) tap in the air with a controller, and (7) draw a circle (Fig. 4.1).

This block introduces realistic near-miss movements (e.g., controller collisions, zooming, air-taps, circle drawing) that can resemble tap dynamics without contact. Together with Experiment 1, it enables assessment of both sensitivity to true taps and specificity against everyday VR motions.

4.1.1.1 *Independent Variables*

- **Action type** (7 levels): swing forward; place on table; zoom; collide controllers; tap touchpads; air-tap; draw circle.

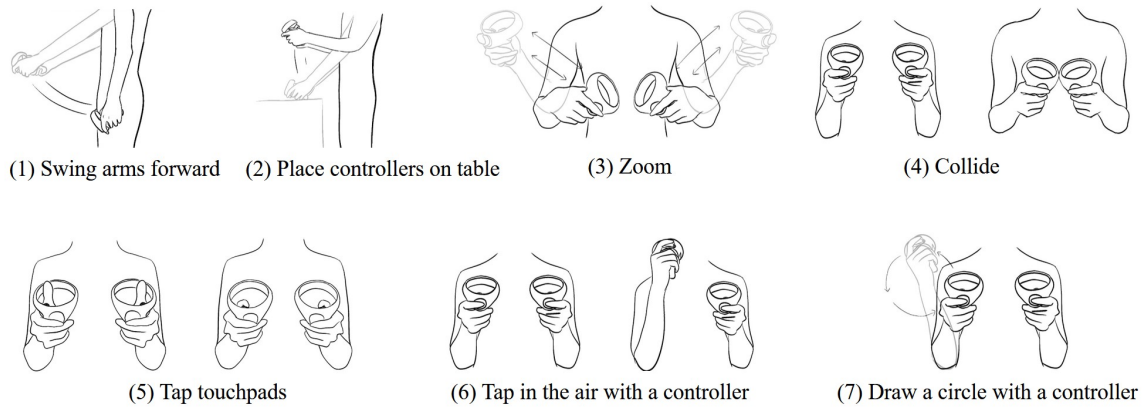


Figure 4.1: Seven representative non-tapping controller actions used to challenge the tap detector. They cover common VR interactions with varied directions, ranges, and bilateral roles.

- **Device family** (2 levels): Meta Quest 2 vs. HTC Vive Pro (order counterbalanced across participants to avoid practice effects).
- **Hand involvement** (2 levels; where applicable): unilateral vs. bilateral execution (e.g., zoom and collide involve both hands; air-tap may be single-hand).

4.1.1.2 *Dependent Variables*

- **Detector outputs**: predicted tap / non-tap for each data to compute *false positive rate* (FPR) by action type.
- **Performance metrics**: accuracy, precision, recall, and F1 for the *non-tap* class per action type; macro/weighted averages reported for comparability.
- **Compliance/timing**: duration per action segment and adherence to instructed motion (used for quality control and descriptive reporting).

4.2 RESULTS ANALYSIS AND DISCUSSION

This experiment evaluates whether the detector can distinguish true taps from seven common non-tapping controller actions. Across the seven cohorts (Types 1–7 in Table 4.1), performance is consistently high: **precision is 1.000** for every cohort, while **recall** and **accuracy** range from 0.9668 to 0.9776, yielding **F1** between 0.9831 and 0.9887.

The uniform precision of 1.000 indicates *zero false positives* in all seven settings; non-tapping actions were never mistaken for taps. This property is critical for real-time interaction because a single false trigger can disrupt the user experience. The small spread in recall (about one percentage point) suggests that detection sensitivity is stable even when movement patterns differ across action types (e.g., larger arm swings vs. subtle repositioning). The lowest recall/accuracy appears in Type 1 (0.9668), which likely reflects partial overlap in dynamics with tap-like motion (e.g., rapid approach without contact), whereas the highest recall occurs in Type 2 (0.9776), where the motion is more distinct from a tap.

Table 4.1: Experiment 2 — Performance by non-tapping action type (tapping vs. non-tapping).

Type	Accuracy	Precision	Recall	F1-score
1	0.9668	1.0000	0.9668	0.9831
2	0.9776	1.0000	0.9776	0.9887
3	0.9695	1.0000	0.9695	0.9845
4	0.9707	1.0000	0.9707	0.9851
5	0.9686	1.0000	0.9686	0.9840
6	0.9711	1.0000	0.9711	0.9853
7	0.9743	1.0000	0.9743	0.9870

Taken together, these results show that short, windowed kinematic features from *both* controllers generalize well beyond the tapping block and maintain strong specificity under realistic distractor movements. Residual misses (recall < 1) are consistent with brief boundary jitter around contact or near-miss behaviors that momentarily resemble taps. Two lightweight mitigations are practical without increasing model complexity: (i) a short refractory period after a positive to suppress duplicate decisions, and (ii) a simple temporal consistency rule that requires two consecutive positive windows before acting. If further improvement is needed for the most confusable action types, adding a weak geometric cue (e.g., inter-controller distance delta) or modest data augmentation around near-miss sequences should reduce the remaining errors while preserving the model's low latency.

4.3 CONCLUSION

This chapter showed that the trained detector can reliably distinguish *tapping* from a diverse set of *non-tapping* controller actions in realistic VR use. Across seven representative action types, precision remained at **1.000** and accuracy/recall stayed in the high 0.96–0.98 range, indicating strong specificity (no false triggers) and stable sensitivity despite variation in arm motion.

These outcomes support real-time deployment: short, windowed kinematics from both controllers provide sufficient evidence for low-latency decisions without heavy models. Residual misses are concentrated around near-miss movements and at boundaries of the tap event, where dynamics momentarily resemble a tap. Two lightweight mitigations are practical for production systems: a brief refractory period after a positive decision to suppress duplicates, and a temporal consistency rule that requires two consecutive positive windows before acting. If additional robustness is required for the most confusable actions, modest data augmentation around near-miss sequences and a weak geometric cue (e.g., change in inter-controller distance) can be added without sacrificing responsiveness.

Overall, the detector demonstrates high, consistent performance across non-tapping behaviors, meeting the accuracy and latency requirements for responsive VR interaction while leaving a clear path for incremental hardening in deployment.

5

EXPERIMENT THREE: APPLICATION-LEVEL INTEGRATION OF TAPPING IN VR

This chapter examines tapping identification within a real-time VR application in a preliminary study. We begin by describing the *apparatus*—the hardware platform and software stack used to run and instrument the study. We then outline the *participant* cohort. Next, we explain how the trained tapping models are integrated into the Unity application. Finally, we present the *results* from user-experience measures and offer a *discussion* of reliability, responsiveness, and design implications for deploying tapping identification in real-time VR.

5.1 APPARATUS

To evaluate tapping in a live VR application, we used a hardware/software setup tuned for stable Unity builds, low-latency inference, and dependable logging. The configuration is:

- **Headset:** *Meta Quest 3*, selected for validated builds, strong tracking, and sufficient CPU/GPU headroom for real-time interaction.
- **Unity build:** *Unity 2022.3.46f1* (LTS) with the standard Quest 3 XR stack. The app records per-frame controller pose and kinematics and runs the trained tap detection/location models on-device in real time.
- **Backend service:** A companion process in *Python 3.11.13* manages session control, light preprocessing, and event exchange with the game.
- **Telemetry & storage:** *Firebase Realtime Database* streams time-stamped events and feature summaries for live monitoring and post-hoc analysis, enabling low-overhead, cross-session aggregation.
- **Rationale & exclusions:** Quest 3 met the latency and tooling requirements for this study. Quest 2 and HTC Vive were excluded because, for this application, their performance/tooling would require additional integration beyond the scope.

5.2 PARTICIPANTS

The study recruits **6 right-handed** adults (age **20–35**). Right-handed recruitment keeps motor patterns consistent with prior experiments and reduces variance from handedness asymmetry during application use. All participants report normal or corrected-to-normal vision and no known motor impairments relevant to the tasks. We conducted a pilot study with the participants to collect their initial feedback.

5.3 APPLICATION IMPLEMENTATION IN UNITY

5.3.1 *System Overview and Data Flow*

Unity samples headset and controller signals *each frame* (pose, velocities, button states) and writes compact raw records to Firebase Realtime Database under a session-scoped path. A lightweight local Python 3.11 service subscribes to this frame stream, performs feature extraction and scaling, runs the trained models (tap detection, tap location, hand use, and force), and publishes detection objects back to the cloud database (e.g., `/sessions/{id}/events` with `type`, `timestamp`, `confidence`, `hand`, and `position`). Unity maintains a real-time listener on the events path and, upon receiving a detection, immediately triggers the bound gameplay actions (mini-map, weapon swap, heal). This division keeps rendering/input responsive inside Unity while isolating ML computation in the external process (Fig. 5.9).

5.3.2 *Telemetry, Data Schema, and Timing*

At runtime, the participant **holds the grip button on the tapping hand**; while held, Unity records headset/controller signals every frame and buffers them locally. When the grip is **released**, Unity closes the segment and uploads a single *batch* to Firebase (Fig. 5.2).

Figure 5.1 shows the device signals exposed in Unity (OpenXR): pose (position/rotation), linear and angular velocity, button states (Grip/Trigger), and tracking. These are the exact fields packaged for upload.

Batch structure. Each batch represents one candidate tap (grip-press → tap → grip-release) and contains:

- `samples`: per-frame entries for `LeftController` and `RightController` with `position` (x, y, z), `velocity` (magnitude), `acceleration` (magnitude), and `timestampMs` (examples in Figs. 5.3–5.4);
- metadata: `sessionId` and `tappingHand` (0 = left, 1 = right).

Batches are written under `/sessions/sessions/batches/{epochMs}`, keeping writes compact and simplifying reconstruction of model-ready sequences during preprocessing. The posture and grip convention are illustrated in Fig. 5.5.



Figure 5.1: Unity device view (OpenXR) showing headset/controller signals used for capture: pose, linear/angular velocity, button states, and tracking.

5.3.3 *Streaming, Inference, and Reliability*

Unity posts raw frame records as compact JSON (e.g., `/sessions/{id}/frames/{t}`) with authenticated, rate-limited writes and exponential backoff to avoid blocking the game loop. A Python 3.11.13 listener consumes these frames, extracts features, applies training-split min–max scaling, and runs the trained models. For each decision, it writes predictions back to Firebase under a session-scoped *predictions* node. Unity subscribes to the `/sessions/{id}/predictions/latest` path and, upon receiving an update, immediately triggers the bound gameplay actions (mini-map, weapon swap, heal). Figure 5.6 shows the prediction layout: the `latest` object exposes the most recent values for `tapping` (0/1), `hand` (0 = left, 1 = right), `position` (zone id or -1 if none), and `ts` (epoch ms). A rolling archive is kept under `predictions/{timestamp}` for post-hoc analysis.

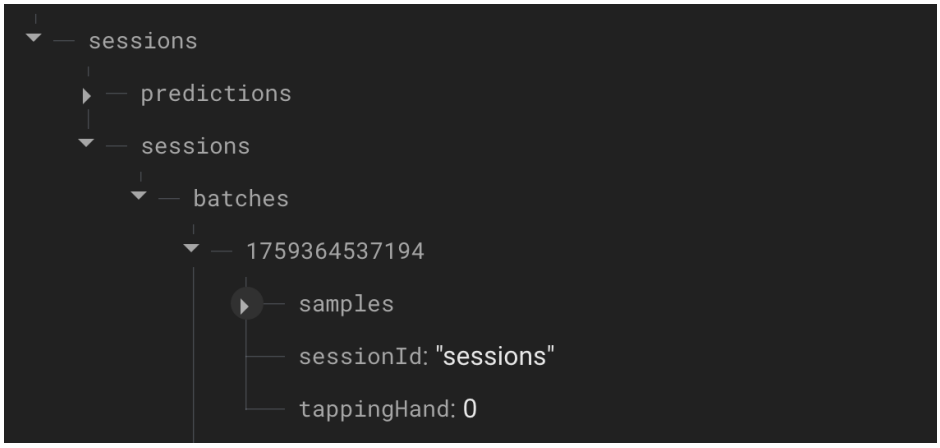


Figure 5.2: Firebase batch container. Each batch spans one grip-press→release episode and includes `samples` plus `sessionId/tappingHand`.



Figure 5.3: Example per-frame entry for `LeftController`: acceleration/velocity magnitudes, 3D position, and `timestampMs`.

5.3.4 *Binding Tapping to Game Features*

The tapping input module raises Unity events when a specific tap gesture is recognised. For example, a *right-hand tap on the left wrist* fires its event; Unity immediately invokes all bound functions (e.g., toggle mini-map or swap weapon). Figure 5.7 shows the `TappingInput` inspector where these tap→action bindings are configured.

- **Right → Left wrist**: toggle the *mini-map*.
- **Left → Right wrist**: *swap weapon* (suppressed when a syringe is held).
- **Left → Right wrist or right elbow while holding a syringe**: *heal* (consume one syringe charge).

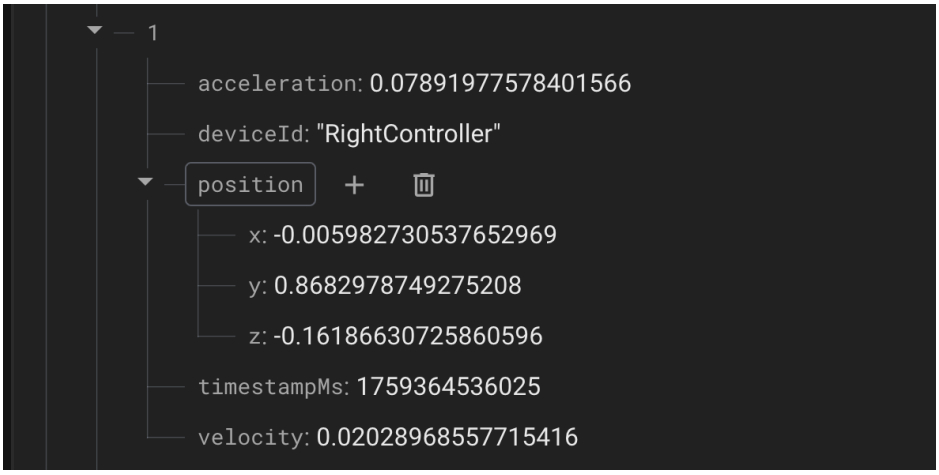


Figure 5.4: Example per-frame entry for `RightController`: acceleration/velocity magnitudes, 3D position, and `timestampMs`.



Figure 5.5: Participant posture used for runtime capture (Sec. 5.3.2). The left hand is positioned to perform a tap on the right hand.

When multiple rules could apply, the more specific rule takes precedence (*heal* > *swap*). Visual (on-wrist status) and a brief haptic tick confirm recognition.

5.3.5 *Game Scenario*

The test scene is a single-level survival task: players eliminate enemies while accessing three utilities via taps (mini-map, weapon swap, heal). Movement uses standard Quest locomotion; difficulty and enemy spawn rate are fixed across sessions. Visual feedback (on-wrist status) and a brief haptic tick confirm detected taps. The scene targets 72 FPS on

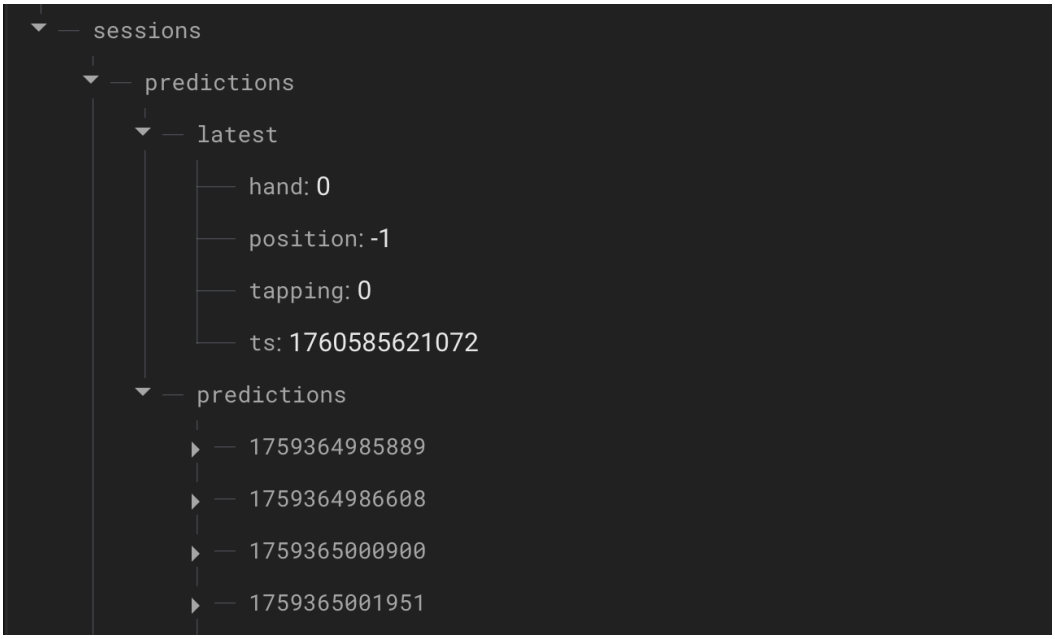


Figure 5.6: Firebase prediction schema. The Python listener writes the most recent decision to `/predictions/latest` (fields: `tapping`, `hand`, `position`, `ts`), which Unity reads to trigger actions. Historical predictions are stored under `/predictions/{timestamp}` for analysis.

Quest 3; tapping events are applied immediately upon receipt from `/predictions/latest` (Sec. 5.3.3). No other game mechanics influence the tapping interaction. Representative frames from the scene are shown in Fig. 5.8.

5.4 EXPERIMENT PROCEDURE

This experiment embeds tapping into a live VR game loop and evaluates whether players can use it reliably to access utilities while fighting zombies. The procedure maps specific bimanual taps to in-game actions, gives participants a short onboarding to learn the gestures, and runs a Unity ↔ Firebase ↔ Python process that detects taps in real time and feeds events back to the application (see Fig. 5.9).

5.4.1 *Interaction Mapping (Taps → Game Actions)*

Each gesture is defined by the tapping hand and the target zone on the opposite forearm. Actions are triggered only when a tap is detected and validated; if multiple rules could apply, the more specific rule (healing while holding a syringe) takes precedence.

- **Right hand → left wrist**: display a *mini-map* on the player's hand (wrist-mounted panel) to aid navigation.
- **Left hand → right wrist**: *swap guns* (cycle to the next weapon) *unless* the player is holding a syringe.

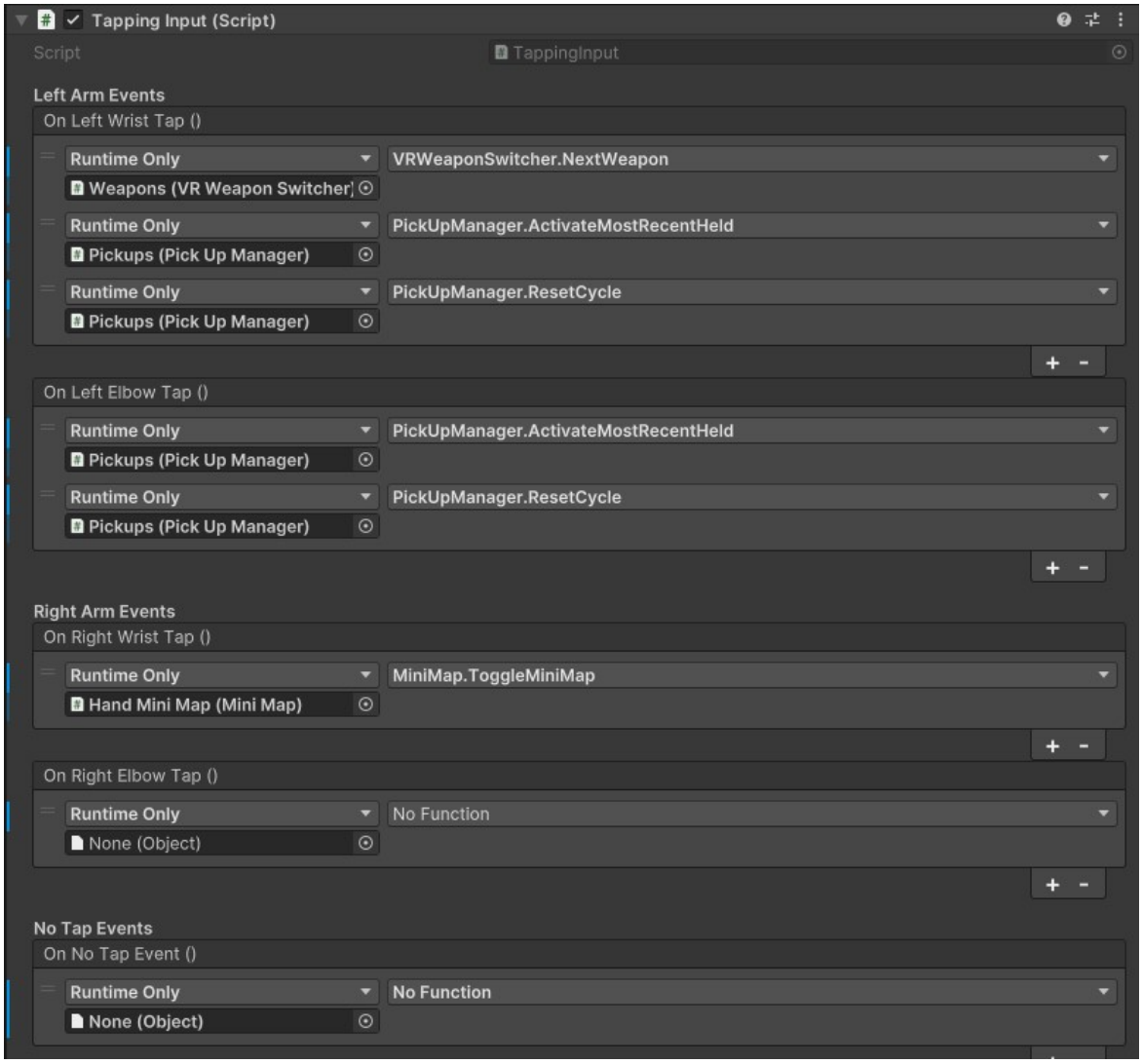
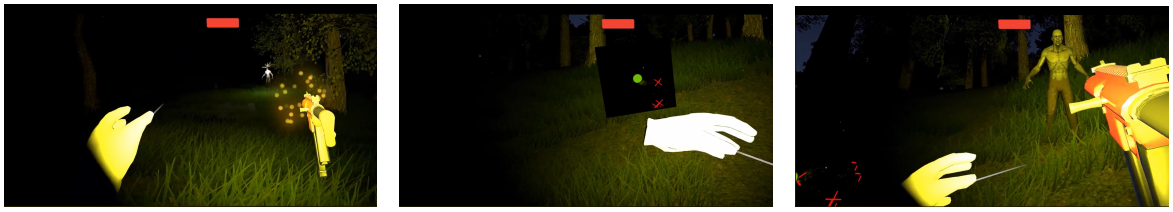


Figure 5.7: Unity inspector for `TappingInput`. Detected tap events (e.g., left/right wrist, left elbow) are bound to game actions such as `MiniMap.ToggleMiniMap`, `VRWeaponSwitcher.NextWeapon`, and `PickUpManager.ActivateMostRecentHeld/ResetCycle`.

- **Left hand → right wrist or right elbow while holding a syringe**: *heal* (apply a health boost and consume one syringe charge).

5.4.2 *Participant Briefing and Practice*

Before gameplay, participants receive a short introduction to the tapping technique: posture, where to tap (wrist vs. elbow), and how firm/brief the contact should be. Each participant practices the three mapped gestures until they can produce consistent detections (confirmed on the wrist-panel indicator). Only after this familiarisation do participants begin the game session.



(a) Combat scene while preparing a heal.

(b) Mini-map shown on wrist after a right→left wrist tap.

(c) Swap to another gun.

Figure 5.8: In-game evidence of tap-driven utilities: healing, wrist mini-map, and combat interactions.

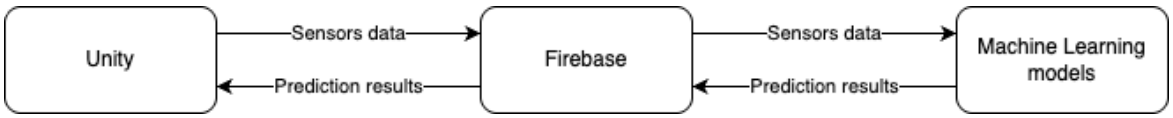


Figure 5.9: Runtime data flow. Unity streams sensor data to Firebase; a local ML process subscribes, performs windowing and normalisation, runs the detector and (if positive) the locator, then writes prediction results back. Unity listens for these events to drive gameplay logic.

5.4.3 *Runtime Detection Pipeline*

During play, Unity streams controller kinematics to Firebase; a Python 3.11.13 listener consumes the stream, applies the trained models, and writes back detection events, which Unity subscribes to in real time:

1. **Unity → Firebase:** per-frame signals are buffered into short windows and pushed as telemetry.
2. **Python detection:** windows are normalised, classified (tap/non-tap, then zone), and an action event is emitted on positive results.
3. **Firebase → Unity:** action events (mini-map, swap, heal) are applied immediately in the game.

This loop keeps latency low while decoupling inference from rendering and physics (Fig. 5.9).

5.4.4 *Gameplay Task and Rules*

Participants play a single-level scenario with the objective to **eliminate all zombies on the island**. Tapping is encouraged as a utility: the mini-map supports spatial planning, weapon swapping adapts to enemy mix, and syringe taps restore health under pressure. Standard movement and shooting are unchanged; only the above three utilities are bound to taps. Performance and qualitative feedback focus on whether tapping improves task flow without causing accidental triggers or cognitive overload.

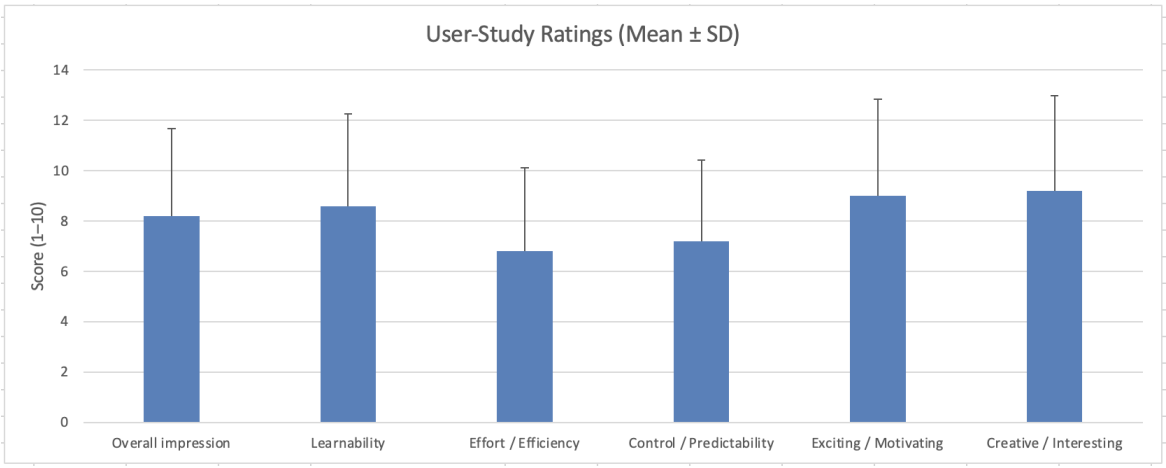


Figure 5.10: User–study ratings (Mean ± SD; 1–10 scale, n=6) for all metrics: Overall impression, Learnability, Effort/Efficiency, Control/Predictability, Exciting/Motivating, Creative/Interesting.

5.5 RESULTS ANALYSIS AND DISCUSSION

We evaluated player perception of the tapping-enabled VR application with a short post-task questionnaire (n=6). The responses provide an updated picture of *usability*, *control*, and *engagement*.

5.5.1 *Overall Impression and Learnability*

Figure 5.10 (single summary chart) visualises all six questionnaire metrics. For the two headline items, **overall impression** averaged **7.5** $\pm$ 1.97 (median 8), indicating a generally positive reaction with one low rating widening the spread, while **learnability** was higher at **8.17** $\pm$ 1.47 (median 8.5), suggesting the technique is easy to pick up after a short briefing and a few practice taps.

5.5.2 *Ease of Use and Control*

As also shown in Fig. 5.10, **effort/efficiency** yielded **6.33** $\pm$ 2.16 (median 6) and **control/predictability** **6.67** $\pm$ 1.86 (median 7). These mid–high scores point to workable interaction with occasional friction—typically during fast movement or near tap boundaries—suggesting that clearer feedback and small timing guards would further stabilise performance.

5.5.3 *Engagement and Creativity*

Engagement-focused items in Fig. 5.10 skew high: **exciting/motivating** averaged **8.67** $\pm$ 1.37 (median 9) and **creative/interesting** **8.33** $\pm$ 2.16 (median 9). Participants found the

contact-based tapping compelling and "game-like"; one lower creativity score likely reflects individual preference rather than a systemic issue.

5.5.4 *Implications and Next Steps*

Taken together, the data suggest that tapping is *learnable and engaging*, with solid perceived control but room to smooth out effort under fast gameplay. We prioritise three refinements:

- **Stronger feedback loops:** persistent on-wrist status (ready/detected), a brief haptic tick on detection, and a short cooldown to suppress duplicates.
- **Robustness during motion:** slightly higher decision thresholds and an optional two-detections rule when locomoting or firing.
- **Mapping support:** a lightweight in-game cheat-sheet (mini-map overlay) reminding players of the three tap mappings to reduce cognitive load.

These adjustments target the mid-range medians on effort and predictability while preserving the high engagement and learnability reported by participants.

6

GENERAL DISCUSSION

6.1 IDENTIFICATION OF TAPPING WITH VR CONTROLLERS

Tapping can be identified reliably from standard controller signals when recognition is done *step by step*: first detect the tapping event, then classify the knock position (wrist / mid-forearm / elbow). This separation keeps decisions precise and fast, and it works well across our three experiments.

Experiment 1 (offline evaluation). The *tap detector* reached **100% accuracy** (tap: precision **0.98**, recall **0.99**; non-tap: precision **1.00**, recall **1.00**). The *hand-use detector* cleanly separated acting vs. target arms with **100% accuracy**. After a positive, the *location* classifier identified forearm zones with high accuracy: **0.97** for *left-hand* taps and **0.94** for *right-hand* taps. Errors concentrated mildly at *mid-forearm*, which is expected given its proximity to adjacent zones.

Experiment 2 (robustness to non-tapping actions). Across seven common controller actions, *precision was* ***1.000*** for every type and accuracy/recall ranged $\approx$**0.967–0.978**. In practice, ordinary motions almost never pass the contact check; the rare near-miss that does is filtered by the knock position classifier. This confirms strong *specificity* with stable *sensitivity* across realistic movements.

Experiment 3 (application). The detector ran in a Unity–Firebase–Python loop and successfully drove gameplay utilities. Post-task ratings with n=6 show the technique is *learnable and appealing*: **overall impression** median **8**; **learnability** median **8.5**. Practical frictions remain under pressure: **effort/efficiency** median **6** and **control/predictability** median **7**. Engagement was consistently high (**exciting/motivating** median **9**; **creative/interesting** median **9**). In runtime logs, the *hand-role (acting-hand) detector* aligned with Unity's controller metadata on all triggered events (no mismatches observed), and offline it achieved **100%** accuracy on held-out windows, confirming reliable left/right assignment alongside tap detection.

In offline tests, a straightforward two-step approach—first detect the tap, then classify its forearm zone (knock position)—performed almost flawlessly and remained highly specific against non-tapping actions. Performance is less uniform in the live VR scene, primarily because the training data were collected in scripted postures while gameplay introduces a wider range of speeds, poses, item holds, and locomotion. The most promising next steps are to expand the dataset with richer in-situ recordings (varied poses, movement speeds, weapon states, and stress), add light augmentation and/or quick per-user calibration, and tighten feedback/timing guards. Given their strong offline results and low compute cost,

the current lightweight models should generalise substantially better with broader data coverage—without requiring heavier architectures or sensors.

6.2 TAPPING PERFORMANCE WITH VR CONTROLLERS

We analyse performance along three practical axes: *hand roles* (which hand acts vs. which serves as the target), *tap position* on the forearm (wrist, mid-forearm, elbow), and *device family*. Together, these factors characterise how well the method works under typical VR use and where designers should expect variance, informing robust gesture-to-action mappings and hardware choices.

6.2.1 *Hand Roles and Performance*

Since one hand acts and the other serves as a steady target, tapping is inherently asymmetrical. In our evaluations, the *tap-moment detector* reached **100%** overall accuracy (macro F1 = **0.99**), and the *acting-hand detector* cleanly separated the moving arm from the stabilising arm with **100%** accuracy (balanced supports). For tap *location*, accuracy was **0.97** for *left-hand* taps and **0.94** for *right-hand* taps, indicating robust identification on both sides with a small asymmetry. Given that many users perform better with their dominant hand acting on the non-dominant forearm, this disparity aligns with kinematic-chain expectations. In practice, the most dependable execution is achieved when frequent or time-sensitive commands are mapped to *dominant* $\rightarrow$ *non-dominant* taps; the reverse mapping remains functional but is marginally less stable.

6.2.2 *Tap Position*

Per-class results indicate that while *mid-forearm* carries the most confusion, *wrist* and *elbow* are the easiest to separate (precision/recall typically ≥ 0.95). As might be expected, the mid-region has more variation in where people aim because it lies between two boundaries. Two straightforward mitigations are effective for deployment: (i) use tolerant thresholds or confirmation feedback close to boundaries; and (ii) prefer larger, well-separated targets (e.g., wrist vs. elbow) for critical actions.

6.2.3 *Robsutness to non-tapping actions*

When challenged with seven representative non-tapping actions, the detector maintained **perfect precision (1.000)** and high recall/accuracy (≈ 0.97). Ordinary motions rarely pass the contact check; the rare near-miss that does is filtered by the tap-position classifier. This pattern indicates strong *specificity* under realistic movement.

6.2.4 *Real-time Consideration*

In the VR application, reliability decreased compared with offline tests. The primary cause is a *data mismatch*: training data came from fixed, scripted poses, whereas gameplay introduced broader variability in posture, speed, and context. Performance is therefore limited more by data coverage than by model capacity. Broadening the dataset (more poses, movement styles, locomotion, object holds, and stress conditions), together with light augmentation and short cooldowns/feedback cues, should recover most of the gap without requiring heavier models or sensors.

6.2.5 *Summary*

In general, tapping works well for all controller families, hands, and positions. The detector is very specific against non-tapping actions, the wrist and elbow are the most separable positions, and dominant-hand taps onto the non-dominant forearm provide the most reliable mapping. The remaining variability is caused by mid-forearm boundary ambiguity and by gameplay conditions that were not simulated during training; these problems can be resolved with focused data expansion and simple interaction protections.

6.3 INTERACTIVE EXPERIENCE WITH TAPPING

This section breaks down the key findings for application use by summarizing how players experienced tapping as an interaction technique, including how quickly it was picked up, how predictable it felt during play, and whether it increased engagement.

6.3.1 *Learnability and Onboarding*

Most participants were able to produce consistent taps after a quick introduction and a few practice trials. This was reflected in post-task ratings, where *learnability* and *overall impression* clustered around a median of **7**/10. Both (i) body-relative instructions ("right hand to left wrist/elbow") and (ii) a clear visual cue on the wrist panel confirming detection were beneficial to learnability in practice. Early hesitation was lessened by these cues, which also directed attention away from gesture memory and toward timing and aim.

6.3.2 *Predictability and Control*

Wider spread and mid-range perceived *control/predictability* (median **5**/10) suggested sporadic friction under pressure. Two causes are identified by qualitative notes and play logs: posture/speed variation during combat (data mismatch with the scripted training set) and boundary cases close to the mid-forearm (position ambiguity). Players were able to adjust their course with the aid of in-the-moment feedback, which included brief haptics

and a persistent "ready/detected" indicator. However, missed detections under high tempo continued to occur.

6.3.3 *Ease of Use*

ease of completing tasks without unnecessary effort received ratings centered around **5**/10. Taps were effective when detection fired without any menus or mode switches. Managing enemies required players to remember the mapping or repeat a gesture, which primarily increased effort. A quick cooldown to avoid double-fires and a wrist-side cheat sheet with the three mappings that are available were two lightweight mitigations that worked well.

6.3.4 *Engagement and Creativity*

Engagement was consistently high in the post-task ratings (n=6). *Exciting/motivating* scored $\{7, 7, 9, 9, 10, 10\}$ with a median of **9**, and *creative/interesting* scored $\{4, 9, 9, 9, 9, 10\}$ with a median of **9** (one low outlier at 4). Players noted that the brief physical contact feels satisfying and "game-like," providing tactile confirmation without extra hardware—an appealing reason to keep taps for quick utilities such as mini-map and heal rather than burying them in menus.

6.3.5 *Summary*

Overall, tapping was *learnable and appealing* (overall impression median **8**; learnability median **8.5**) with strong engagement (medians **9**/**9**). The main UX gap is *predictability under pressure* (effort median **6**, control median **7**). In practice, this is better addressed by richer in-situ training data (more varied poses, speeds, and contexts) and clearer feedback/timing guards (e.g., brief haptic tick, on-wrist status, short cooldown or two-detections rule during locomotion) rather than heavier recognition models. Notably, the acting-hand detector aligned perfectly with runtime events in our logs, and the offline tap-moment detector remained near-perfect—indicating that reliability improvements should focus on boundary cases and gameplay variability, not core recognition capability.

6.4 USER EXPERIENCE AND WORKLOAD

This section summarises subjective experience and perceived workload when using tapping in a live VR game. Participants (n=6) completed brief Likert items covering *overall impression*, *learnability*, *effort/efficiency*, *control/predictability*, and *engagement*.

6.4.1 *Subjective Experience*

Overall sentiment was positive. **Overall impression** had a median of **8** ($\{4,7,8,8,8,10\}$), and **learnability** a median of **8.5** ($\{6,7,8,9,9,10\}$), indicating that the technique is easy to pick up after a short introduction. **Effort/efficiency** showed a median of **6** ($\{4,5,5,7,7,10\}$), and **control/predictability** a median of **7** ($\{4,5,7,7,8,9\}$), reflecting occasional friction under pressure (e.g., rapid movement, posture drift, or taps near the mid-forearm boundary). Engagement was consistently high: **exciting/motivating** scored $\{7,7,9,9,10,10\}$ (median **9**) and **creative/interesting** $\{4,9,9,9,9,10\}$ (median **9**), suggesting the contact-based interaction feels "game-like" and adds hedonic value without extra hardware.

6.4.2 *Implications*

The pattern is clear: tapping is *engaging and learnable*, while perceived ease is limited by *predictability* during high-tempo play. The most direct improvements are (i) broader in-situ training data (more varied poses, locomotion, weapon holds, and speed/stress conditions), and (ii) stronger feedback/timing guards—persistent on-wrist "ready/detected" status, a brief haptic tick on detection, and a short cooldown or two-detections rule during locomotion—to reduce duplicate/missed triggers at peak demand. These changes target workload precisely where it spikes, without additional hardware or heavier recognition models.

6.5 DESIGN IMPLICATIONS

The findings from robustness testing, in-game use, and offline evaluation point to a set of useful recommendations for implementing tapping with generic VR controllers.

6.5.1 *Recognition and Mapping*

- **Stage the decision.** First detect the *tap event*; on positives, infer the *acting hand* (hand-use) and then classify the *tap position* (knock position). This sequencing keeps compute and latency low, prevents spurious classifications on non-tap frames, and cleanly routes actions by hand.
- **Prioritise specificity.** False triggers are more harmful than rare misses. Require brief temporal consistency (e.g., two consecutive positive windows) and add a short cooldown after a detection.
- **Map critical actions to robust positions.** Prefer *wrist* or *elbow* for time-sensitive commands; use *mid-forearm* for secondary actions or with tolerant thresholds.
- **Favour dominant→non-dominant taps** for frequent utilities; the reverse mapping is usable but slightly less stable.

- **Keep directions body-relative.** Instructions like "right hand to left wrist" reduce cognitive load and improve learnability.

6.5.2 *Modeling and Data*

- **Start lightweight.** Decision-tree style models on controller kinematics are accurate, interpretable, and fast enough for on-device use.
- **Match training to gameplay.** Expand data beyond scripted poses: include varied postures, locomotion, weapon holds, speeds, and stress. Light augmentation (jitter, speed scaling) improves generalisation.
- **Stay device-agnostic.** Build features from standard poses/IMUs and relative geometry so the model transfers across controller families.

6.5.3 *Feedback and UX*

- **Always-on state.** Show a small wrist indicator (ready/detected) and play a brief haptic tick on success.
- **Error recovery.** If a tap is missed, surface a subtle hint (e.g., "aim closer to wrist") based on the nearest detected position.
- **Memory aids.** Provide a compact on-wrist cheat sheet for the current tap mappings; keep the vocabulary small (e.g., three utilities).

In combination, these choices yield reliable, low-latency tapping with minimal complexity, while targeting the specific failure modes observed at tap-position boundaries and during high-tempo play.

6.6 LIMITATIONS AND FUTURE WORK

6.6.1 *Data and Generalisation*

Scripted vs. gameplay data. While real gameplay introduces a wider range of variation (posture, speed, locomotion, object holds, stress), training data were gathered in fixed, clearly defined poses. The majority of errors in the live application can be explained by this distribution shift. *Future work:* add targeted augmentation (speed scaling, pose jitter, partial occlusions), expand the dataset with in-the-wild sessions, and mine hard negatives/near-miss segments using active learning.

Sample size and balance. The class balance is skewed (non-tap $\gg$ tap; mid-forearm underrepresented), and the participant pool is modest. *Future work:* create boundary cases to lessen mid-forearm confusion; expand and balance cohorts based on handedness, body size, and play styles.

6.6.2 *Sensing and Features*

Force estimation. Commodity controllers do not sense contact force directly; velocity/acceleration are imperfect proxies, yielding only moderate force accuracy. *Future work:* explore higher-rate IMU access, contact audio, or lightweight haptics/microphones as additional cues; consider per-user force calibration.

Device coverage. Models were validated on common controller families; other devices and hand-tracking (controller-less) remain untested. *Future work:* extend to more headsets/controllers and adapt the contact proxy for bare-hand tracking, while retaining the same recognition logic.

6.6.3 *Model and Runtime*

Fixed hyperparameters. Window size/stride and thresholds were selected empirically and kept global. *Future work:* evaluate adaptive windows (confidence- or motion-aware), and lightweight temporal consistency (e.g., hysteresis, consecutive-positive checks) tuned per context (standing, locomoting, combat).

On-device variability. The Unity–Firebase–Python loop kept latency low in tests, but network and platform variability can still affect responsiveness. *Future work:* move inference fully on-headset, profile worst-case latency, and add fallback behaviors (local queueing, degraded modes) for intermittent connectivity.

6.6.4 *UX and Evaluation*

Feedback and predictability; Mid-range control ratings indicate that users need clearer real-time feedback and guardrails under pressure. *Future work:* add persistent state indicators, brief haptics, short cooldowns, and context-aware confidence thresholds; validate effects with A/B studies.

Longitudinal and accessibility. Results come from short sessions with right-handed users. *Future work:* conduct longitudinal studies to assess learning curves, fatigue, and habit formation; include left-handed and mixed-dominance users; investigate accessibility adaptations (larger targets, alternative mappings).

6.6.5 *Summary*

The current system delivers high offline accuracy and robust non-tap rejection, but real-time reliability is constrained by data coverage and sensing limits for force. The most impactful next steps are (i) *broader, gameplay-matched data*, (ii) *lightweight, on-device temporal/feedback safeguards*, and (iii) *richer cues for intensity*. These directions aim to improve predictability in the wild without sacrificing the simplicity and portability of a controller-only solution.

7

CONCLUSION

This thesis developed and evaluated a controller-only method for recognising arm tapping in VR, taking the idea from data collection and supervised labelling through model selection, robustness testing, and an application-level deployment. The core design is intentionally simple and portable: use short spans of commodity controller kinematics to *detect* a tap event, infer the *acting hand*, and then classify the tap *position* (knock position) on the forearm—so interactions work without external sensors or heavy models.

Data were sampled every 10 ms from both controllers and labelled via an acceleration peak–trough procedure. Frame-wise labels were aggregated into short windows and transformed into compact features: acceleration/velocity for tap detection, and controller coordinates plus inter-controller distance for tap position. Features were min–max normalised to [0,1], and models were trained with subject-separated splits to avoid identity leakage.

Across experiments, the approach proved effective. In offline evaluation, a lightweight *Decision Tree* achieved near-perfect tap detection (overall accuracy **100%**, macro F1 **0.99**); the *acting-hand (hand-use) detector* reached **100%** accuracy, cleanly separating moving vs. stabilising arms; and *tap-position* classification was high for wrist/mid/elbow (**0.97** for left-hand taps, **0.94** for right-hand taps, with most confusion at mid-forearm). In robustness tests against seven common non-tapping actions, precision remained **1.000** and accuracy/recall hovered around **0.97**, demonstrating strong resistance to false triggers. In **Experiment 3** (a Unity–Firebase–Python deployment), tapping reliably drove practical utilities (mini-map, weapon swap, heal) and received positive subjective ratings (overall median **8**, learnability **8.5**, engagement medians **9**/**9**), though perceived effort (**6**) and control (**7**) indicated occasional friction during fast gameplay. Runtime logs showed perfect alignment between detected acting hand and Unity controller metadata on triggered events.

The gap in the live setting is largely a data mismatch: training examples were recorded in fixed, scripted poses, while gameplay produced broader variation in posture, speed, locomotion, object holds, and stress. This limits generalisation more than model capacity. Broadening the dataset to reflect real play—supplemented with light augmentation and brief per-user calibration—should improve transfer without changing the sensing stack. Force detection remains a separate limitation: acceleration/velocity are weak proxies for perceived intensity on commodity controllers, and results were only moderate.

Practically, a *staged recogniser* (detect tap → infer acting hand → classify position) with a strong bias toward specificity works well. Map time-critical actions to robust positions (wrist, elbow), prefer dominant→non-dominant taps for frequent utilities, keep instruc-

tions body-relative, and provide clear feedback (ready/detected indicator, brief haptic tick, short cooldown or two-detections rule during locomotion). These small safeguards address the remaining failure modes—primarily boundary ambiguity and high-tempo movement—without added complexity.

In closing, this thesis shows that accurate tap detection, perfect acting-hand identification, and strong position classification are achievable with off-the-shelf VR controllers and lightweight, interpretable models. Bridging the lab-to-field gap is chiefly a matter of better-matched data and clearer feedback, not heavier architectures or extra hardware—keeping the solution portable and ready to scale across devices and applications.

BIBLIOGRAPHY

[1] HTC Corporation, *Vive australia official website*, https://www.vive.com/au/, Accessed: 27-Apr-2025, 2025.

[2] Meta Platforms, *Meta quest 3s official website*, https://www.meta.com/quest/quest-3s/, Accessed: 2025-05-02, 2024.

[3] Sony Interactive Entertainment, *Playstation vr official website*, https://www.playstation.com/en-au/ps-vr/, Accessed: 2025-05-02, 2025.

[4] D. Corrêa, D. Santos, L. Contini and A. Balbinot, 'Mems accelerometers sensors: An application in virtual reality', *Sensors & Transducers*, vol. 120, no. 9, p. 13, 2010.

[5] B. Delporte, L. Perroton, T. Grandpierre and J. Trichet, 'Accelerometer and magnetometer based gyroscope emulation on smart sensor for a virtual reality application', *Sensors & Transducers.*, vol. 14, no. Special Issue ISSN 1726-5479, p32–p47, 2012.

[6] D. Kao, A. J. Magana and C. Mousas, 'Evaluating tutorial-based instructions for controllers in virtual reality games', *Proceedings of the ACM on human-computer interaction*, vol. 5, no. CHI PLAY, pp. 1–28, 2021.

[7] M. Bonfert, R. Porzel and R. Malaka, 'Get a grip! introducing variable grip for controller-based vr systems', in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, IEEE, 2019, pp. 604–612.

[8] D. Yu, K. Fan, H. Zhang, D. Monteiro, W. Xu and H.-N. Liang, 'Pizzatext: Text entry for virtual reality systems using dual thumbsticks', *IEEE transactions on visualization and computer graphics*, vol. 24, no. 11, pp. 2927–2935, 2018.

[9] Wikimedia Commons contributors, *Meta Quest 3 display unit*, https://en.wikipedia.org/wiki/File:Meta_Quest_3_display_unit.jpg, Accessed: 2025-05-07, 2023.

[10] M. Pesce, *Htc vive*, https://en.wikipedia.org/wiki/File:HTC_Vive.jpg, Accessed: 2025-05-07, 2015.

[11] Wikimedia Commons contributors, *Playstation vr*, https://en.wikipedia.org/wiki/File:PlayStation_VR.jpg, Accessed: 2025-05-07, 2016.

[12] A. Masurovsky, P. Chojecki, D. Runde, M. Lafci, D. Przewozny and M. Gaebler, 'Controller-free hand tracking for grab-and-place tasks in immersive virtual reality: Design elements and their empirical study', *Multimodal Technologies and Interaction*, vol. 4, no. 4, p. 91, 2020.

[13] T. Kot and P. Novák, 'Application of virtual reality in teleoperation of the military mobile robotic system taros', *International journal of advanced robotic systems*, vol. 15, no. 1, p. 1 729 881 417 751 545, 2018.

[14] M. Allgaier, V. Chheang, P. Saalfeld, V. Apilla, T. Huber, F. Huettl, B. Neyazi, I. E. Sandalcioglu, C. Hansen, B. Preim et al., 'A comparison of input devices for precise interaction tasks in vr-based surgical planning and training', *Computers in Biology and Medicine*, vol. 145, p. 105 429, 2022.

[15] S. Çankaya, 'Use of vr headsets in education: A systematic review study', *Journal of Educational Technology and Online Learning*, vol. 2, no. 1, pp. 74–88, 2019.

[16] I. Choi, E. Ofek, H. Benko, M. Sinclair and C. Holz, 'Claw: A multifunctional handheld haptic controller for grasping, touching, and triggering in virtual reality', in *Proceedings of the 2018 CHI conference on human factors in computing systems*, 2018, pp. 1–13.

[17] E. Strasnick, C. Holz, E. Ofek, M. Sinclair and H. Benko, 'Haptic links: Bimanual haptics for virtual reality using variable stiffness actuation', in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[18] H. Yi, J. Hong, H. Kim and W. Lee, 'Dexcontroller: Designing a vr controller with grasp-recognition for enriching natural game experience', in *Proceedings of the 25th ACM Symposium on Virtual Reality Software and Technology*, 2019, pp. 1–11.

[19] M. Baloup, T. Pietrzak and G. Casiez, 'Raycursor: A 3d pointing facilitation technique based on raycasting', in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12.

[20] J. Wagner, W. Stuerzlinger and L. Nedel, 'Comparing and combining virtual hand and virtual ray pointer interactions for data manipulation in immersive analytics', *IEEE transactions on visualization and computer graphics*, vol. 27, no. 5, pp. 2513–2523, 2021.

[21] J. Chatain, D. M. Sisserman, L. Reichardt, V. Fayolle, M. Kapur, R. W. Sumner, F. Zünd and A. H. Bermano, 'Digiglo: Exploring the palm as an input and display mechanism through digital gloves', in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, 2020, pp. 374–385.

[22] B. Fruchard, E. Lecolinet and O. Chapuis, 'Impact of semantic aids on command memorization for on-body interaction and directional gestures', in *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*, 2018, pp. 1–9.

[23] Z. Li, J. Chan, J. Walton, H. Benko, D. Wigdor and M. Glueck, 'Armstrong: An empirical examination of pointing at non-dominant arm-anchored uis in virtual reality', in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.

[24] C. Harrison, S. Ramamurthy and S. E. Hudson, 'On-body interaction: Armed and dangerous', in *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, 2012, pp. 69–76.

[25] C. Harrison, D. Tan and D. Morris, 'Skinput: Appropriating the body as an input surface', in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2010, pp. 453–462.

[26] S. G. Gustafson, B. Rabe and P. M. Baudisch, 'Understanding palm-based imaginary interfaces: The role of visual and tactile cues when browsing', in *Proceedings of the SIGCHI Conference on human factors in computing systems*, 2013, pp. 889–898.

[27] S.-Y. Lin, C.-H. Su, K.-Y. Cheng, R.-H. Liang, T.-H. Kuo and B.-Y. Chen, 'Pub-point upon body: Exploring eyes-free interaction and methods on an arm', in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 481–488.

[28] M. Weigel, A. S. Nittala, A. Olwal and J. Steimle, 'Skinmarks: Enabling interactions on body landmarks using conformal skin electronics', in *proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 3095–3105.

[29] D. Yu, Q. Zhou, T. Dingler, E. Velloso and J. Goncalves, 'Blending on-body and mid-air interaction in virtual reality', in *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2022, pp. 637–646.

[30] J. Son, S. Ahn, S. Kim and G. Lee, 'Improving two-thumb touchpad typing in virtual reality', in *Extended abstracts of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–6.

[31] M. Speicher, A. M. Feit, P. Ziegler and A. Krüger, 'Selection-based text entry in virtual reality', in *Proceedings of the 2018 CHI conference on human factors in computing systems*, 2018, pp. 1–13.

[32] J. G. Grandi, H. G. Debarba and A. Maciel, 'Characterizing asymmetric collaborative interactions in virtual and augmented realities', in *2019 IEEE conference on virtual reality and 3D user interfaces (VR)*, IEEE, 2019, pp. 127–135.

[33] R. Nagashirna, N. Sakata and S. Nishida, 'Study of the touchpad interface to manipulate ar objects', in *2013 23rd International Conference on Artificial Reality and Telexistence (ICAT)*, IEEE, 2013, pp. 78–84.

[34] N. Ryu, H.-Y. Jo, M. Pahud, M. Sinclair and A. Bianchi, 'Gamesbond: Bimanual haptic illusion of physically connected objects for immersive vr using grip deformation', in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–10.

[35] J. Murayama, L. Bougrila, Y. Luo, K. Akahane, S. Hasegawa, B. Hirsbrunner and M. Sato, 'Spidar g&g: A two-handed haptic interface for bimanual vr interaction', in *Proceedings of EuroHaptics*, vol. 2004, 2004, pp. 138–146.

[36] T. Azai, S. Ogawa, M. Otsuki, F. Shibata and A. Kimura, 'Selection and manipulation methods for a menu widget on the human forearm', in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2017, pp. 357–360.

[37] T. Azai, S. Ushiro, J. Li, M. Otsuki, F. Shibata and A. Kimura, 'Tap-tap menu: Body touching for virtual interactive menus', in *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, 2018, pp. 1–2.

[38] D. Yu, Q. Zhou, T. Dingler, E. Velloso and J. Goncalves, 'Blending on-body and mid-air interaction in virtual reality', in *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2022, pp. 637–646.

[39] C. Harrison, H. Benko and A. D. Wilson, 'Omnitouch: Wearable multitouch interaction everywhere', in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 441–450.

[40] A. Mujibiya, X. Cao, D. S. Tan, D. Morris, S. N. Patel and J. Rekimoto, 'The sound of touch: On-body touch and gesture sensing based on transdermal ultrasound propagation', in *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*, 2013, pp. 189–198.

[41] Y. Zhang, J. Zhou, G. Laput and C. Harrison, 'Skintrack: Using the body as an electrical waveguide for continuous finger tracking on the skin', in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 1491–1503.

[42] Y. Zhang, W. Kienzle, Y. Ma, S. S. Ng, H. Benko and C. Harrison, 'Actitouch: Robust touch detection for on-skin ar/vr interfaces', in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, 2019, pp. 1151–1159.

[43] M. Ogata, Y. Sugiura, Y. Makino, M. Inami and M. Imai, 'Senskin: Adapting skin as a soft interface', in *Proceedings of the 26th annual ACM symposium on User interface software and technology*, 2013, pp. 539–544.

[44] M. Ogata and M. Imai, 'Skinwatch: Skin gesture interaction for smart watch', in *Proceedings of the 6th Augmented Human International Conference*, 2015, pp. 21–24.

[45] S. Schneegass and A. Voit, 'Gesturesleeve: Using touch sensitive fabrics for gestural input on the forearm for controlling smartwatches', in *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, 2016, pp. 108–115.

[46] Y. Guiard, 'Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model', *Journal of motor behavior*, vol. 19, no. 4, pp. 486–517, 1987.

[47] K. Hinckley, R. Pausch, D. Proffitt and N. F. Kassell, 'Two-handed virtual manipulation', *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 5, no. 3, pp. 260–302, 1998.

[48] G. Kurtenbach, G. Fitzmaurice, T. Baudel and B. Buxton, 'The design of a gui paradigm based on tablets, two-hands, and transparency', in *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, 1997, pp. 35–42.

[49] R. Balakrishnan and G. Kurtenbach, 'Exploring bimanual camera control and object manipulation in 3d graphics interfaces', in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1999, pp. 56–62.

[50] R. Balakrishnan and K. Hinckley, 'Symmetric bimanual interaction', in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2000, pp. 33–40.

[51] R. Owen, G. Kurtenbach, G. Fitzmaurice, T. Baudel and B. Buxton, 'When it gets more difficult, use both hands: Exploring bimanual curve manipulation', in *Proceedings of Graphics Interface 2005*, 2005, pp. 17–24.

[52] C. Latulipe, S. Mann, C. S. Kaplan and C. L. Clarke, 'Symspline: Symmetric two-handed spline manipulation', in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 2006, pp. 349–358.